Efficient Hardware-aware Neural Architecture Search for Edge Computing

Hadjer Benmeziane

Thèse de doctorat

Pour obtenir le grade de Docteur de

l'UNIVERSITE POLYTECHNIQUE HAUTS-DE-FRANCE

et de l'INSA HAUTS-DE-FRANCE

Discipline, spécialité selon la liste des spécialités pour lesquelles l'Ecole Doctorale est accréditée :

Intelligence Artificielle et Systèmes Embarqués

Présentée et soutenue par Hadjer Benmeziane

Le 30/08/2023, à Valenciennes

Ecole doctorale :

Ecole Doctorale Polytechnique Hauts-de-France (ED PHF n°635)

Unité de recherche :

Laboratoire d'Automatique, de Mécanique et d'Informatique Industrielles et Humaines (UMR CNRS 8201)

Optimisation Automatique des Applications d'Apprentissage Profond sur Plateformes Matérielles Edges

Président de jury

- Cucu-Grosjean, Liliana. Directrice de recherche. INRIA, Rocquencourt, France.

JURY

Rapporteurs

- Sassatelli, Gilles. Directeur de recherche. CNRS, LIRMM, Univ Montpellier.
- Shafique, Muhamed. Professeur. New-York University, Abu-Dhabi.

Examinateurs

- Cucu-Grosjean, Liliana. Directrice de recherche. INRIA, Rocquencourt, France.

Invités

- Meyer, Brett. Assistant Professeur. Department of Electrical and Computer Engineering McGill University.

Thesis director

- Niar, Smail, Professeur, UPHF, CNRS, UMR 8201 - LAMIH, F-59313 Valenciennes, France.

Thesis co-director :

- El Maghraoui, Kaoutar. Principal Research Scientist. IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA.

Co-supervisor :

- Ouarnoughi, Hamza. Professeur. UPHF, CNRS, UMR 8201 - LAMIH, F-59313 Valenciennes, France.

PhD Thesis

Submitted for the degree of Doctor of Philosophy from

UNIVERSITE POLYTECHNIQUE HAUTS-DE-FRANCE

and INSA HAUTS-DE-FRANCE

Subject :

Artificial Intelligence and Embedded Systems

Presented and defended by Hadjer Benmeziane.

On 30/08/2023, LAMIH

Doctoral school :

Doctoral School Polytechnique Hauts-de-France (ED PHF n°635)

Research unit :

Laboratory of Industrial and Human Automation control Mechanical engineering and Computer science (LAMIH – UMR CNRS 8201)

Efficient Hardware-aware Neural Architecture Search for Edge Computing

President of jury

- Cucu-Grosjean, Liliana. Research Director. INRIA, Rocquencourt, France.

JURY

Reviewers

- Sassatelli, Gilles. Research Director. CNRS, LIRMM, Univ Montpellier.
- Shafique, Muhamed. Professor. New-York University, Abu-Dhabi.

Examiners

- Cucu-Grosjean, Liliana. Research Director. INRIA, Rocquencourt, France.

Invitee

- Assistant Professor. Department of Electrical and Computer Engineering McGill University.

Thesis director

- Niar, Smail, Professor, UPHF, CNRS, UMR 8201 - LAMIH, F-59313 Valenciennes, France.

Thesis co-director :

- El Maghraoui, Kaoutar. Principal Research Scientist. IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA.

Co-supervisor :

- Ouarnoughi, Hamza. Professor. UPHF, CNRS, UMR 8201 - LAMIH, F-59313 Valenciennes, France.

Abstract

It is widely anticipated that inference models based on Deep Neural Networks (DNN) will be actively employed in many edge platforms due to several compelling reasons. Firstly, DNNs have demonstrated exceptional performance in various fields such as computer vision, natural language processing, and speech synthesis. Their ability to extract meaningful features from large datasets enables them to achieve high levels of accuracy and predictive power, making them indispensable for a wide range of tasks. Secondly, deploying DNN-based inference models directly on edge platforms offers several advantages. For instance, executing the inference process locally on edge devices, reduces the reliance on cloud-based computing, thereby minimizing network latency and ensuring real-time responsiveness. This is particularly crucial for time-sensitive applications, such as autonomous vehicles, smart surveillance systems, and Internet of Things (IoT) devices, where rapid decision-making is paramount. Furthermore, employing DNN inference at the edge enhances privacy and security. By keeping sensitive data within the edge device's local environment instead of transmitting it to external servers, the risk of data breaches and privacy violations is significantly reduced. This is of utmost importance in scenarios involving personal data, healthcare information, or confidential business data, where preserving privacy and data sovereignty is imperative.

However, edge platforms often operate under resource-constrained environments, characterized by limited computational power, energy constraints, and intermittent connectivity. DNN models are increasingly larger, making them unfit for such platforms. This has promoted research in automatically designing neural architectures through search for such devices. This method is called Hardware-aware Neural Architecture Search (HW-NAS). Such optimization can lead to reduced energy consumption, lower inference latency, and overall improved performance on edge platforms. HW-NAS is the cornerstone of this thesis. HW-NAS can provide both efficient and accurate, customized models for the target platform. This thesis aims at accelerating and generalizing HW-NAS applicability to many platforms and multiple tasks.

This work introduces innovative solutions to rapidly estimate the efficiency of DNNs for a target HW platform. Our proposed HW-NAS approach encompasses multi-objective optimization techniques, significantly accelerating the search process within sxccupernetwork-based and cell-based search spaces. Within the multi-objective context of HW-NAS, conflicting objectives, such as task-specific performance (e.g., accuracy) and hardware efficiency (e.g., latency and energy consumption), need to be optimized simultaneously. To address this challenge, we define a novel Pareto rank target, leveraging diverse surrogate models employed in HW-NAS. By incorporating multiple objectives and Pareto optimization principles, our approach enables the exploration of trade-offs between task-specific performance and hardware efficiency, ultimately facilitating the identification of superior neural architectures.

We also investigate the human bias induced by current search spaces and propose a non-restrictive search space to find novel operators tailored to a target hardware platform. These methods were validated on image classification benchmarks. We then show how to apply HW-NAS for novel hardware architectures, namely analog in-memory computing hardware.

Finally, we construct a medical imaging NAS benchmark that includes architectures for 11 tasks, including their performance, latency, and energy consumption on several devices, and propose a new HW-NAS approach that, not only includes accuracy and latency as objectives, but also looks for a generalizable architecture that can be fine-tuned for unseen medical tasks.

Keywords Neural Architecture Search, Hardware constraints, Optimization, Edge AI

Résumé

Les modèles d'inférence basés sur les réseaux neuronaux profonds (eng, Deep Neural Networks (DNN)) sont largement utilisés dans de nombreuses platesformes de périphérie pour plusieurs raisons. Premièrement, les DNN ont démontré des performances exceptionnelles dans divers domaines tels que la vision par ordinateur, le traitement du langage naturel et la synthèse vocale. Leur capacité à extraire des caractéristiques significatives à partir de grands ensembles de données leur permet d'atteindre des niveaux jamais atteints de précision et de puissance prédictive, ce qui les rend indispensables pour une large gamme d'applications. Deuxièmement, le déploiement de ces modèles directement sur les plateformes de périphérie offre plusieurs avantages. L'exécution du processus d'inférence localement sur les dispositifs de périphérie réduit la dépendance à l'égard du calcul basé sur le cloud, réduisant ainsi la latence du réseau et garantissant une réactivité en temps réel.

Cependant, les plateformes de périphérie fonctionnent souvent dans des environnements contraints en ressources, caractérisés par une puissance de calcul limitée, des contraintes énergétiques et une connectivité intermittente. Les modèles DNN ne sont par défaut pas adaptés à de telles plateformes. Cela a encouragé la recherche sur la conception automatique d'architectures neuronales adaptées à ces dispositifs. Cette méthode est appelée recherche d'architecture neuronale à contraintes matérielles (eng, Hardware-aware Neural Architecture Search, HW-NAS). HW-NAS est la pierre angulaire de cette thèse. HW-NAS peut fournir des modèles à la fois efficaces et précis. Cette thèse vise à accélérer et à généraliser l'applicabilité de HW-NAS à de nombreuses plateformes et à plusieurs tâches. Ce travail de thèse propose des solutions novatrices pour estimer rapidement l'efficacité d'un DNN déployé sur une plateforme matérielle cible. Notre approche HW-NAS englobe des techniques d'optimisation multi-objectifs, ce qui accélère considérablement le processus de recherche à la fois dans les espaces de recherche basés sur les supernetworks et sur les cellules. Dans le contexte multi-objectif de HW-NAS, des objectifs conflictuels, tels que les performances spécifiques à la tâche (par exemple, la précision) et l'efficacité matérielle (par exemple, la latence et la consommation d'énergie), doivent être optimisés simultanément. Pour relever ce défi, nous définissons un nouvel objectif de rang de Pareto. En incorporant des objectifs multiples et des principes d'optimisation de Pareto, notre approche permet l'exploration des compromis entre les performances spécifiques à la tâche et l'efficacité matérielle. Nous examinons également le biais humain induit par les espaces de recherche actuels et proposons un espace de recherche non restrictif pour trouver de nouveaux opérateurs adaptés à une plate-forme matérielle cible. Ces méthodes ont été validées sur des référentiels de classification d'images.

Dans la seconde partie de la thèse, nous montrons l'utilité de nos méthodes dans des scénarios réels. Premièrement, comment appliquer HW-NAS à de nouvelles plateformes matérielles, notamment les matériels de calcul analogiques en mémoire (eng, in-memory analog devices). Nous avons proposé un HW-NAS dédié à ces plateformes, et nous déduisons les charactérstiques qui différent un réseaux de neuronnes déployé sur ces plateformes, d'un autre déployé sur des plateformes classiques.

Enfin, nous construisons une référence de recherche d'architectures neuronales pour l'imagerie médicale qui inclut des architectures pour 11 tâches, notamment la detection de tumeurs, la segmentation du foie et l'estimation du volume de l'hippocampe. En utilisant cette référence, nous proposons un nouveau HW-NAS qui inclut non seulement l'exactitude et la latence en tant qu'objectifs, mais cherche également une architecture généralisable qui peut être affinée pour de nouvelles tâches médicales.

Mots clés apprentissage profond, optimisation, contraintes matérielles

Acknowledgments

First and foremost, I would like to thank my thesis director Prof. Smail Niar and my supervisor Prof. Hamza Ouarnoughi at Université Polytechnique des Hauts-de-France. The unique research opportunity they provided has been transformative for my academic journey. With their combined expertise, I navigated the complexities of this thesis, and I deeply value the trust they placed in my abilities and the guidance they offered.

I would also like to extend my profound gratitude to my thesis co-director, Dr. Kaoutar El Maghraoui from IBM T.J Watson. Kaoutar has been more than just a supervisor; she has been a mentor in the truest sense of the word. Her guidance, patience, and dedication have been pivotal in shaping my research perspective. I am especially thankful for the invaluable opportunity she provided by allowing me to intern at IBM. The hours she invested in mentoring me, coupled with the hands-on experience at IBM, have enriched my academic journey and provided me with insights that I will carry with me throughout my career.

Heartfelt appreciation is extended to the esteemed jury members for their invaluable time and dedication in accepting, meticulously reading, and offering constructive feedback on this work.

I am grateful to the many researchers I met during my thesis work. I've collaborated with academics from different backgrounds. At present, I'd like to highlight the following co-authors and/or scientists that reviewed my papers and helped me improve them: Irem Boybat, Abu Sebastian, Manuel Le Gallo, Malte J. Rasch, Corey Lammie, Hsinyu Tsai, Ramachandran Muralidhar, Halima Bouzidi, Lotfi Abdelkrim Mecharbat, Ozcan Ozturk, Amine Ziad Ounnoughene, Imane Hamzaoui, Younes Bouhadjar, Abderaouf Gacem, Afaf Alloulal, Mufida Miratul, Rihab Balti and Meyssa Zouambi.

Last, but certainly not least, I would like to thank my family. Their unwavering support, endless patience, and boundless love have been the pillars upon which I leaned throughout this journey. To my parents, whose sacrifices and teachings have shaped who I am today, I owe a debt of gratitude that words can hardly express. They instilled in me the values and skills to live independently, always grounded by unwavering principles and beliefs that they imparted. To my siblings, for their constant encouragement and belief in my abilities, even more than me. While this achievement is a significant milestone, I recognize it as just the beginning of my research journey. Thank you all for being the foundation upon which I build my future endeavors.

List of Publications Included in this Thesis

This thesis contains a number of original research articles which have been published during my PhD candidature. These papers have been slightly modified to improve readability and cohesion in the form of a thesis document. In this Section, a list of publications included in this thesis is presented.

- A Comprehensive Survey on Hardware-aware Neural Architecture Search

[1] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Sma¨ıl Niar, Martin Wistuba, and Naigang Wang. A comprehensive survey on hardware-aware neural architecture search. CoRR, abs/2101.09336, 2021.
[2] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. Hardware-aware neural architecture search: Survey and taxonomy. In Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21, pages 4322–4329, 8 2021.
Location in thesis: Chapter 2

- Accelerating Neural Architecture Search with Rank-Preserving Surrogate Models

[3] Hadjer Benmeziane, Hamza Ouarnoughi, Kaoutar El Maghraoui, and Smail Niar. Accelerating neural architecture search with rank-preserving surrogate models. In Manar Abu Talib, Laila Benhlima, and Kaoutar El Maghraoui, editors, ArabWIC 2021: The 7th Annual International Conference on Arab Women in Computing in Conjunction with the 2nd Forum of Women in Research, Sharjah, ACM 2021.
Location in thesis: Chapter 3
Best Paper Award at ArabWIC 2021

- Multi-Objective Hardware-Aware Neural Architecture Search with Pareto Rank- Preserving Surrogate Models

[4] Hadjer Benmeziane, Hamza Ouarnoughi, Kaoutar El Maghraoui, and Smail Niar. Multi-objective hardware-aware neural architecture search with Pareto rank-preserving surrogate models. 20(2), 2023. ACM Transactions on Architecture and Code Optimization.
Location in thesis: Chapter 3

- Pareto rank surrogate model for hardware-aware neural architecture search

[5] Hadjer Benmeziane, Smail Niar, Hamza Ouarnoughi, and Kaoutar El Maghraoui. Pareto rank surrogate model for hardware-aware neural architecture search. In IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2022.
Location in thesis: Chapter 3

- Pareto Rank-Preserving Supernetwork for HW-NAS

[6] Hadjer Benmeziane, Smail Niar, Hamza Ouarnoughi, and Kaoutar El Maghraoui. Pareto rank-preserving supernetwork for hardware-aware neural architecture search. European Conference on Artificial Intelligence ECAI 2023.
Location in thesis: Chapter 3
Best Poster Award at IBM/AICS 2022

- Grassroots Operator Search for Model Edge Adaptation

[7] Hadjer Benmeziane, Smail Niar, Hamza Ouarnoughi, and Kaoutar El Maghraoui. Grassroots operator search for model edge adaptation. Submitted to Elsevier Future Generation Computer Systems.
Location in thesis: Chapter 4

- CaW-NAS: Compression-aware Neural Architecture Search

[8] Hadjer Benmeziane, Hamza Ouranoughi, Sma¨ıl Niar, and Kaoutar El Maghraoui. Caw-nas: Compression aware neural architecture search. In 25th Euromicro Conference on Digital System Design, DSD, pages 391–397. IEEE, 2022. **Location in thesis:** Chapter 4

- AnalogNAS: A Neural Network Design Framework for Accurate Inference with Analog In-Memory Computing

[9] Hadjer Benmeziane, Corey Lammie, Irem Boybat, Malte J. Rasch, Manuel Le Gallo, Hsinyu Tsai, Ramachandran Muralidhar, Sma¨ıl Niar, Hamza Ouarnoughi, Vijay Narayanan, Abu Sebastian, and Kaoutar El Maghraoui. Analognas: A neural network design framework for accurate inference with analog in-memory computing. 2023. IEEE International Conference on Edge Computing & Communications.
Location in thesis: Chapter 5
Best Paper Award at IEEE Edge 2023

- MED-NAS-Bench: A Generalized Neural Architecture Search for Medical Imaging Analysis

[10] Hadjer Benmeziane, Lotfi Abdelkrim Mecharbat, Smail Niar, Hamza Ouarnoughi, and Kaoutar El Maghraoui. Med-nas-bench: A generalized neural architecture search benchmark for medical imaging analysis. To be Submitted to Nature Methods, 2023.
Location in thesis: Chapter 6

List of publications not included in this thesis

- Real-time style transfer with efficient vision transformers

[11] Hadjer Benmeziane, Hamza Ouarnoughi, Kaoutar El Maghraoui, and Sma¨ıl Niar. Real-time style transfer with efficient vision transformers. In Aaron Yi Ding and Volker Hilt, editors, EdgeSys@EuroSys 2022: Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking, Rennes, France, April 5 - 8, 2022, pages 31–36. ACM, 2022

- HyT-NAS: Hybrid Transformers Neural Architecture Search for Edge Devices

[12] Lotfi Abdelkrim Mecharbat, Hadjer Benmeziane, Hamza Ouranoughi, and Sma¨ıl Niar. Hyt-nas: Hybrid transformers neural architecture search for edge devices. CoRR, abs/2303.04440, 2023

- Treasure What You Have: Exploiting Similarity in Deep Neural Networks for Efficient Video Processing

[13] Hadjer Benmeziane, Halima Bouzidi, Hamza Ouarnoughi, Ozcan Ozturk, and Smail Niar. Treasure what you have: Exploiting similarity in deep neural networks for efficient video processing. CoRR, abs/2305.06492, 2023.

- Skip Connections in Spiking Neural Networks: An Analysis of Their Effect on Network Training

[14] Hadjer Benmeziane, Amine Ziad Ounnoughene, Imane Hamzaoui, and Younes Bouhadjar. Skip connections in spiking neural networks: An analysis of their effect on network training. CoRR, abs/2303.13563, 2023.

Filed Patents

- CO-Design of a Model and Chip for Deep Learning Background (Filed)

Irem Boybat Kara, Hadjer Benmeziane, Manuel Le Gallo-Bourdeau, Kaoutar El Maghraoui, Malte Johannes Rasch, and HsinYu Tsai

Contents

List of Figures

List of Tables

Acronyms

AIHWKit IBM Analog Hardware Acceleration Kit.

AAE Average Absolute Error.

ADC Analog-to-Digital Converter.

AF Architecture Features.

AI Artificial Intelligence.

ASIC Application-specific Integrated Circuit.

AVM Accuracy Variation over one Month.

BO Bayesian Optimization.

BPM Beats Per Minute.

CE Cross-Entropy.

CNN Convolutional Neural Networks.

CPU Central Processing Unit.

DAC Digital-to-Analog Converter.

DL Deep Learning.

DNN Deep Neural Networks.

EA Evolutionary Algorithm.

ECG Electrocardiography.

EHR Electronic Health Records.

FLOPS Floating Point Ops per Second.

FPGA Field-programmable Gate Array.

FSP Flow of Solution Procedure.

GAN Generative Adversarial Networks.

GCN Graph Convolutional Neural Networks.

GPGPU General-purpose Graphics Processing Units.

GPU Graphics Processing Unit.

GRU Gated Recurrent Unit.

HSS Hardware Search Space.

HW-NAS Hardware-aware NAS.

HWA Hardware-aware.

IMC In-memory Computing.

IoT Internet of Things.

KL div Kullback–Leibler divergence.

KWS Keyword Spotting.

LHS Latin Hypercube Sampling.

LSTM Long Short-Term Memory Networks.

MAC Multiply-Accumulate.

MCU Microcontroller Unit.

ML Machine Learning.

MLP Multi-layer Perceptron.

MRAM Magnetic Random Access Memory.

MSE Mean Squared Error.

MVM Matrix-Vector Mutliplication.

NAS Neural Architecture Search.

NSGA Non-dominated Sorting Genetic Algorithm.

NVM Non Von Neuman.

PCM Phase Change Memory.

PPG Photoplethysmography.

PTB Physikalisch Technische Bundesanstalt.

RGB Red Green Blue.

RL Reinforcement Learning.

RMSE Root Mean Square Error.

RNN Recurrent Neural Networks.

RRAM Resistive Random Access Memory.

RS Random Search.

SGD Stochastic Gradient Descent.

SHGO Simplicial Homology Global Optimization.

SNN Spiking Neural Networks.

SoC System-on-Chips.

SOTA State-Of-The-Art.

STD Standard Deviation.

TinyML Tiny Machine Learning.

ViT Vision Transformers.

VPU Vision Processing Unit.

VWW Visual Wake Words.

WL Word Line.

Chapter 1

Introduction

Contents

1.1 Context & Motivation

Deep Neural Networks (DNNs) have emerged as the cornerstone of numerous contemporary Artificial Intelligence (AI) applications. Their groundbreaking application in speech synthesis, computer vision, and language modeling has sparked exponential growth in the adoption of DNNs across various domains. Notably, DNNs have surpassed human accuracy in several areas, highlighting their remarkable efficacy. This superior performance stems from their aptitude for extracting intricate features from raw sensory data through statistical learning and brain-inspired operations. DNN architectures are pivotal in shaping the performance and capabilities of Deep Learning (DL) models. Designers must carefully select the architecture, layer types, and connectivity patterns that best capture the relationships and patterns within the data. Furthermore, optimizing the DNN's hyperparameters, such as learning rates, regularization techniques, and activation functions, demands extensive experimentation and fine-tuning. The sheer scale and depth of modern DNN architectures necessitate substantial computational resources, as well as the expertise to train and validate these models efficiently.

However, the remarkable accuracy achieved by DNNs comes with a trade-off in the form of high computational complexity. Historically, general-purpose compute engines, particularly graphics processing units (GPUs) [22], have been the primary workhorses for DNN processing. In addition, deploying DNNs entails significant memory requirements, as state-of-the-art networks are getting larger each year. A notable example of this evolution is observed in the domain of image classification and object detection. While Convolutional Neural Networks (CNNs) have long been regarded as state-of-the-art models in this field, their supremacy has recently been challenged by the emergence of vision transformers, which use 30x more parameters [23]. Nevertheless, a growing necessity has emerged to execute these DNNs directly on users' edge devices, primarily driven by privacy concerns. This new challenge has given rise to several research directions. Firstly, in the context of the waning days of Moore's Law, there is a realization that advancing compute performance and energy efficiency necessitates specialized hardware tailored specifically for DNN workloads. Secondly, optimization strategies that aim to reduce the computational requirements of DNNs, such as quantization, pruning, and Hardware-aware Neural Architecture Search (HW-NAS), have garnered considerable relevance.

Specifically, HW-NAS [1] has seen significant interest and progress in recent years, as evidenced by the growing number of research papers and publications dedicated to this topic, as shown in Figure 1.1. Neural Architecture Search (NAS) refers to a set of methods that leverage computational algorithms to automatically design the architecture of neural networks. This automated approach eliminates the need for manual trial-and-error iterations, enabling the exploration of a vast search space of potential architectures. HW-NAS takes the concept of NAS a step further by considering hardware-related factors such as memory capacity, computational efficiency, and power consumption during the architecture search process. By doing so, HW-NAS algorithms can intelligently discover architectures that are not only accurate and high-performing but also optimized for deployment on specialized hardware accelerators or resource-constrained devices.

Despite the progress made in HW-NAS, there are several current issues that need to be addressed to further advance this field and ensure its practical applicability. Firstly, current strategies are time-consuming requiring the performance evaluation of each sampled architecture. This evaluation is usually performed using surrogate models and estimation methods. Still, these estimation methods are time-consuming to construct and they do not account for the multi-objectivity in HW-NAS [24]. Secondly, the search space design is crucial as it defines the ranges of performance, the search can explore. However, designing a search space based on previously handcrafted architectures restrict HW-NAS to similar architectures without innovation

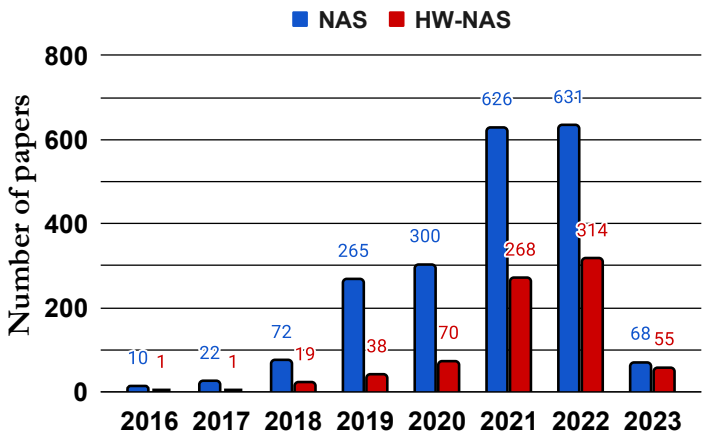

Figure 1.1: Number of papers published on NAS and HW-NAS as of May 2023.

potential. Lastly, there is still a big gap in applying HW-NAS to other DL tasks than image classification. In image classification, the standardization of common benchmarks makes it easier to compare multiple methods. However, the lack of standardized benchmarks and evaluation metrics on multiple DL tasks makes it challenging to compare and validate different methodologies effectively.

In this context, the primary motivation driving this thesis is the need for an efficient and practical HW-NAS approach tailored specifically for edge devices. As edge devices become increasingly prevalent in various domains, building an efficient HW-NAS is a key imperative to enable the deployment of powerful DL models on such resource-constrained devices. The methods proposed in this thesis adapt to multiple DL tasks and are validated on a varied set of hardware platforms.

Detailed motivations can be summarized as follows: (i) Accelerate the architecture performance evaluation process used in HW-NAS. (ii) Reduce the overall search time of HW-NAS, making the method practical in real-world applications. (iii) Improve the design of search spaces, developing a non-restrictive space to discover novel architectures. (iv) Investigate the use of HW-NAS for computer vision tasks including image classification, object detection, and medical segmentation.

1.2 Research Questions

The research questions addressed in this thesis manuscript aim to improve the efficiency and effectiveness of HW-NAS, including its real-world and practical applications. Specifically, the study aims to answer the following questions:

1. What are the key components of HW-NAS, and how can they be optimized to improve performance?

2. How can multi-objective and Pareto-aware surrogate models be developed to enhance the evaluation components of HW-NAS?

3. How can search spaces be enhanced with quantization awareness and free from humanly designed operators to improve the search process?

4. Furthermore, how can HW-NAS be applied to novel hardware platforms, such as analog in-memory computing?

5. How can it be used to optimize benchmarks in medical imaging analysis?

6. What are the key considerations and methodologies for developing a comprehensive benchmark specifically tailored for evaluating NAS methods in medical imaging, and how can such a benchmark be designed to effectively capture the complexities and challenges presented by medical imaging datasets?

7. How can a multi-task NAS methodology be developed to effectively account for the diverse range of medical imaging types and tasks?

By addressing these research questions, this thesis aims to contribute to the field of HW-NAS and advance the state-of-the-art in this important research area.

1.3 Summery of Contributions

This thesis comprises seven significant original research contributions:

1. In [1], a survey of state-of-the-art HW-NAS techniques is presented, providing an overview of the current landscape in HW-NAS, the challenges that concern each component of HW-NAS, and a comparison between the different estimation methods used to evaluate the architecture's performance. In addition, insights were provided for future estimation and search space design strategies and approaches.

2. In [4, 5], we propose a novel multi-objective surrogate model, *HW-PR-NAS*. To train this model, we define a Pareto score for each architecture, given its accuracy, latency, and energy consumption. We then train the surrogate model to learn the ranking of the architectures. Once, used in HW-NAS, HW-PR-NAS achieved up to 2.5x speedup compared to state-of-the-art methods while achieving 98% near true Pareto front, on seven different edge hardware platforms from various classes, including ASIC, FPGA, GPU, and multi-cores.

3. Using the same Pareto score definition, in [6], we adapt the training of a *supernetwork* search space. Supernetworks are a novel way to design HW-NAS search spaces. We propose a supernetwork training methodology that preserves the Pareto ranking between its different subnetworks resulting in neural networks more efficient and accurate for a variety of hardware platforms. The results show a 97% near Pareto front approximation in less than 2 GPU days of search, which provides 2x speed up compared to state-of-the-art methods. We validate our methodology on multiple NAS benchmarks.

4. In [8], we define a novel methodology to extend the search space during the search, allowing the exploration of large search spaces containing in an efficient manner. More specifically, CaW-NAS combines the search for the architecture and its quantization policy. While former works search over a fully quantized search space, we define our search space with quantized and non-quantized architectures. Our search strategy finds the best trade-off between accuracy and latency according to the target hardware. Experimental results on a mobile platform show that our method allows us to obtain more efficient networks in terms of accuracy, execution time, and energy consumption when compared to the state-of-the-art.

5. In [7], we propose a Grassroots Operator Search (GOS). GOS adapts a given model for edge devices by searching for an efficient operator replacement. We express each operator as a set of mathematical instructions that capture its behavior. The mathematical instructions are then used as the basis for searching and selecting efficient replacement operators that maintain the accuracy of the original model while reducing computational complexity. Our approach is grassroots since it relied on the mathematical foundations to construct new and

efficient operators for DL architectures. We demonstrate on various DL models that our method consistently outperforms the original models on two edge devices, namely Redmi Note 7S and Raspberry Pi3, with a minimum of 2.2x speedup while maintaining high accuracy. Additionally, we showcase a use case of our GOS approach in pulse rate estimation on wristband devices, where we achieve state-of-the-art performance, with reduced computational complexity.

6. Analog In-Memory Computing (IMC) is a new approach for building efficient inference accelerators. Current DNNs, however, are not designed for such hardware either in terms of operators or hyperparameters. AnalogNAS [9] proposes a framework for automated DNN design targeting deployment on IMC inference accelerators. We conduct extensive hardware simulations to demonstrate the performance of AnalogNAS on state-of-the-art models in terms of accuracy and deployment efficiency on various tiny machine-learning tasks. We also present experimental results that show AnalogNAS models achieving higher accuracy than state-of-the-art models when implemented on a 64-core in-memory computing chip based on Phase change memory.

7. Medical imaging tasks are an ideal domain for HW-NAS on edge computing due to the increasing demand for efficient and accurate medical imaging systems, the availability of edge devices in healthcare settings, and the need to optimize DL models for such resource-constrained devices. In [10], we designed a NAS benchmark for medical imaging analysis. The benchmark targets eleven tasks, including brain, lung, liver, and pancreas tumor segmentation, hippocampus, spleen, and prostate segmentation, and pneumonia detection. Included in the benchmark are the performance metrics and the hardware efficiency of millions of architectures trained using a supernetwork design. On top of this benchmark, we developed a multi-task HW-NAS methodology that not only finds efficient architectures but is also generalizable to multiple medical tasks.

1.4 Open Source Projects

One of the key contributions of this thesis is the development of two open-source neural architecture search libraries: AnalogNAS and MED-NAS-Bench.

- *AnalogNAS* is a modular and flexible analog-aware NAS python library that is designed to optimize neural network architectures for analog hardware accelerators. This library includes various search algorithms and neural network building blocks, enabling users to easily design and search for analog-efficient neural network architectures. By leveraging AnalogNAS, researchers and engineers can design efficient neural networks for edge devices, IoT devices, and other applications that rely on analog accelerators. The Python package can be installed via PyPi. The code and implementation can be found: https://github.com/IBM/analog-nas. *This library was awarded the IEEE Open Source for Science Award in 2023.*

- The MED-NAS-Bench API represents a significant contribution to the field of medical imaging research. By offering a comprehensive benchmark for NAS in the context of medical imaging, the API serves as a valuable resource for both researchers and practitioners. The Python package can be installed via PyPi. The code and implementation can be found:
https://github.com/IHIaadj/med_nas_bench

1.5 Thesis Organization

As illustrated in Figure 1.2, this manuscript is organized into 7 chapters to convey all of the original research contributions in a coherent way.

The current Chapter, i.e., the introduction, highlighted in pink, delves into the research background and motivation. In addition, research questions are formulated, and the key original contributions of this thesis, and open-source projects are summarized.

The latter of the manuscript is divided into three parts:

- **Literature Review:** provides **a comprehensive overview** of related works and defines the essential components of hardware-aware neural architecture search. This section lays the foundation for the subsequent parts of the manuscript and sets the context for the research presented in the paper. This section is present in Chapter 2.

- **Efficient HW-NAS methods:** this part is dedicated to the contributions made to **accelerate HW-NAS**. It is divided into two chapters. Chapter 3 focuses on the development of **multi-objective and Pareto-aware surrogate models** to enhance the evaluation components of HW-NAS. We describe HW-PR-NAS and PRP-NAS, two estimation strategies targeting two types of search spaces; cell-based and supernetwork.

  Chapter 4 concentrates on the **the design of an efficient search space**. It describes CaW-NAS, an optimized methodology to extend the search space with compressed architectures. And a novel design and search strategy, in which the search space is composed of fine-grained operators, allowing the discovery of novel architectures.

- **Applications of HW-NAS:** In this part, we adapt and use our HW-NAS knowledge on two real-world use cases. Chapter 5 explains how to use HW-NAS in the context of analog in-memory computing, and showcases how architectures differ from one platform to another. Chapter 6 highlights the **development of a NAS benchmark for medical imaging analysis** and the design of a multi-task hardware-aware NAS to enhance medical DL models.

Overall, the manuscript provides a comprehensive and in-depth analysis of hardware-aware neural architecture search, including its essential components, improvements, and applications in conventional and novel hardware platforms. Finally, new benchmarking and scenarios of applications for HW-NAS are also proposed.

Finally, the thesis is concluded in Chapter 7, conclusion and future work. In Figure 1.2, this Chapter is highlighted in the same color as the introduction to indicate a strong link/connection. In the conclusion, the findings in other chapters are summarized concerning the research questions formulated in the introduction, and future research directions are discussed.

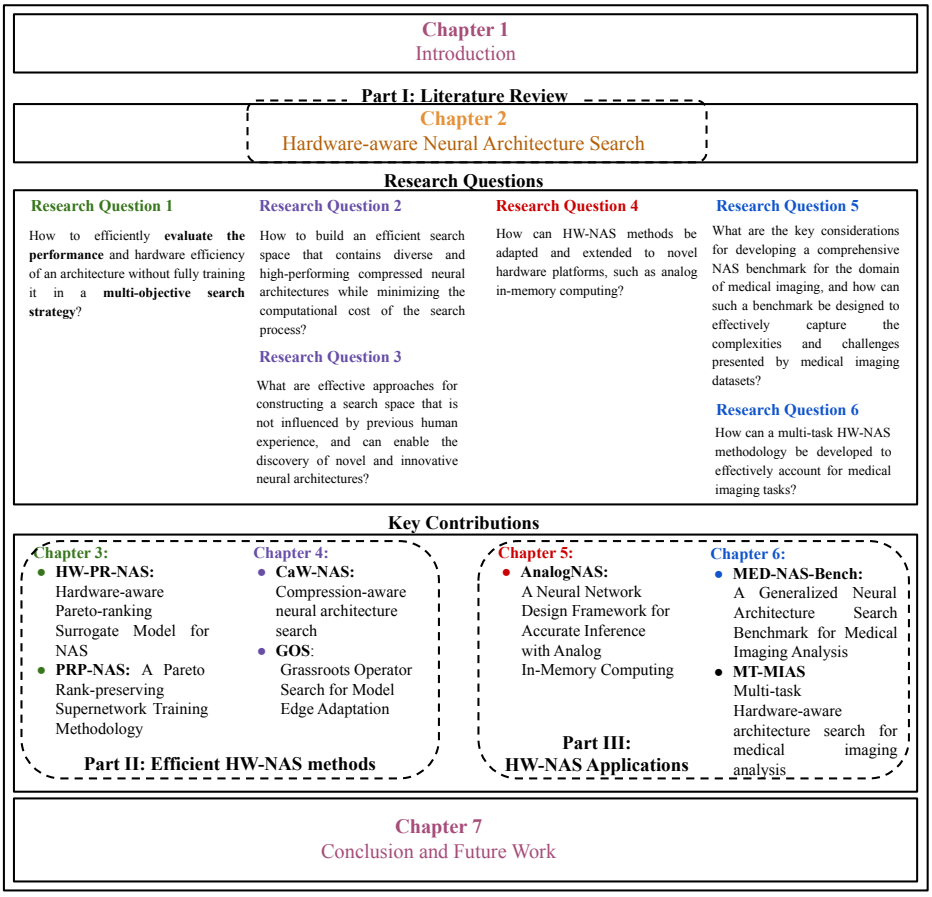

Figure 1.2: Structure of the manuscript.

Part I

Related Works

Chapter 2

Hardware-aware Neural Architecture Search

Contents

The significant advances and breakthroughs of Deep Learning (DL), that have propelled from academic and industrial research labs' circles, are now electrifying the computing industry and transforming the world. DL is now being largely used to solve real-world problems. As DL is computationally demanding, most of the deployments happen in the cloud or on-premises data centers. However, with the arrival of powerful and low-energy consumption Internet of Things (IoT) devices and the growing need to take action in real or near-real-time, DL computations are increasingly moving to the edge. This pushes the development of small yet powerful DL architectures.

In this chapter, we embark on a comprehensive exploration of handcrafted models, Neural Architecture Search (NAS) models, and Hardware-aware NAS (HW-NAS) models, with a focus on their performance and parameter count. We highlight the significance of efficient NAS and HW-NAS methodologies in addressing the increasing demand for resource-efficient deep learning models. Subsequently, we provide a general overview of the NAS framework, outlining its key components and highlighting the relevance of HW-NAS within this context. Furthermore, we present an original taxonomy of HW-NAS approaches, categorizing them based on their specific goals and objectives. Each component of HW-NAS is thoroughly examined, drawing insights from the existing literature. Additionally, we delve into other critical considerations in HW-NAS, such as automatic quantization, reliability, and novel hardware platforms. To provide a comprehensive analysis, we also dedicate a section to discussing the challenges and limitations associated with existing HW-NAS works, shedding light on areas that require further exploration and improvement.

2.1 Handcrafted models Vs. HW-NAS

DL is revolutionizing technology around us across many domains such as computer vision [25, 26, 27, 28], speech processing [29, 30, 31] and natural language processing [32, 33, 34]. These breakthroughs would not have been possible without the availability of big data, the recent algorithmic advancements, the tremendous growth in computational power, and advances in hardware acceleration techniques. However, designing accurate neural networks is challenging due to:

- The variety of data types and tasks that require different neural architectural designs and optimizations.
- The vast amount of hardware platforms makes it difficult to design one globally efficient architecture.

For instance, certain problems require task-specific models, e.g. EfficientNet [35] for image classification and ResNest [36] for semantic segmentation, instance segmentation, and object detection. These networks differ in the proper configuration of their architectures and their hyperparameters. The hyperparameters refer to the pre-defined properties related to the architecture or the training algorithm.

In general, the neural network architecture can be formalized as a Directed Acyclic Graph (DAG) where each node corresponds to an operator applied to the set of its parent nodes [37]. Convolution, pooling, activation, and self-attention are example operators. Linking these operators together gives rise to different architectures. A key aspect of designing a well-performing deep neural network is deciding the type and number of nodes and how to compose and link them. Additionally, the architectural hyperparameters, such as stride and channel number in a convolution, and the training hyperparameters, such as learning rate, number of epochs, and momentum, are also important contributors to the overall performance. Figure 2.1 shows an illustration of some architectural choices for the type of convolutional neural network.

According to this representation, DL architectures can contain hundreds of layers and millions or even billions of parameters. These architectures are either handcrafted

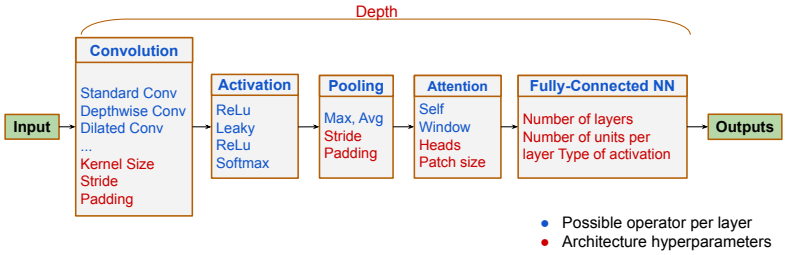

Figure 2.1: Generic DL architecture. For each layer, an operator is chosen among a pre-defined list or operations (convolution, dilated convolution, depthwise convolution, max-pooling, batch_normalization, etc.). The sequence Convolution, Activation, Attention, etc. is repeated several times in DL architectures.

by repetitive experimentation or modified from a handful of existing models. These models have also been growing in size and complexity. This makes handcrafting deep neural networks a complex task that is time-consuming, error-prone and requires deep mathematical expertise. Thus, in recent years, it is not surprising that techniques to automatically design efficient architectures, or Neural Architecture Search (NAS), for a given dataset or task, have surged in popularity.

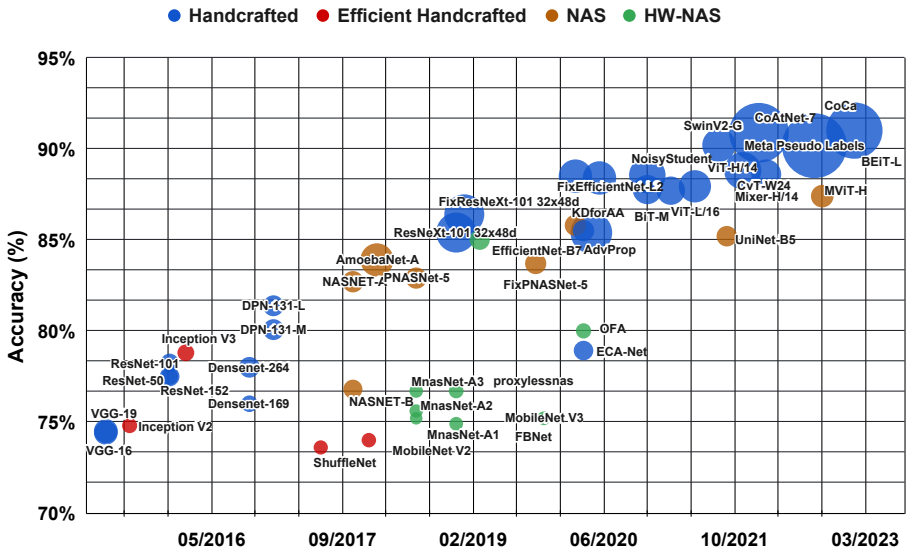

Figure 2.2: Accuracy of various CNN models on ImageNet for Image Classification task with the number of parameters. Inspired by [15]

In figure 2.2, we compare several DL models for the image classification task depending on their Top-1 accuracy and their sizes. Each dot in the plot corresponds to a given DL architecture that has been used for image classification. The dot size correlates with the size of the corresponding model in terms of the number of parameters. The highest value is from CoAtNet-7 [38] which has 2440M parameters. A quick look at the graph reveals the trend to design larger models to better Top-1 accuracy. However, a large size is not necessarily correlated with better accuracy. There have been several efforts to conceive more efficient and smaller networks to achieve comparable Top-1 accuracy performance. We compare four classes of architectural designs: Handcrafted, Efficient handcrafted, NAS, and HW-NAS. Generally,

throughout the years the handcrafted models rank high up in terms of accuracy but are much more complex in terms of architecture's depth and number of parameters. For instance, CoCa [39], which is the state-of-the-art model as of April 2023, has over 2100 million parameters. In the top right quadrant of the figure 2.2 (around the same region as most of the recently handcrafted models), we find some of the models that are automatically created by different NAS techniques. These techniques focus only on improving the model's accuracy without paying attention to the efficiency of the model in terms of its size and latency. Therefore, these NAS models are still large, with the number of parameters ranging between 100M and 700M.

Since 2015, we have noticed the rise of efficient handcrafted models. These models rely on compression methods (see section 2.3) to decrease the model's size while trying to maintain the same accuracy. MobileNet-V2 [40] and Inception [41] are good examples where the number of parameters is between 20M and 80M. Hardware-aware NAS (HW-NAS) emerges as a technique to automatically design efficient DL architectures. This class encompasses work that aims to tweak NAS algorithms and adapt them to find efficient DL models optimized for a target hardware device. HW-NAS began to appear in 2017 and since then achieved state-of-the-art (SOTA) results in resource-constrained environments with Once-for-all (OFA) [42] for example.

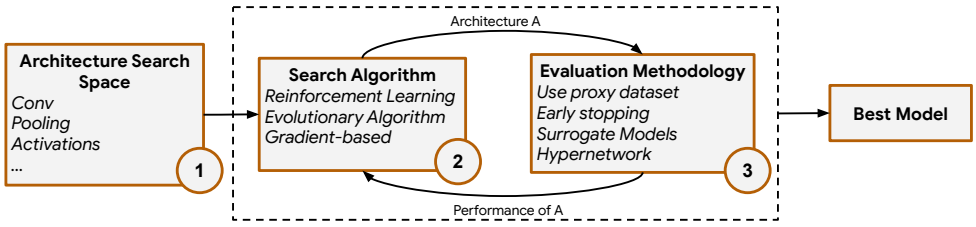

Figure 2.3: Overview of conventional NAS components.

2.2 Conventional Neural Architecture Search

A conventional NAS process requires the definition of three main components: the search space, the search strategy, and the evaluation methodology, as illustrated in figure 2.3.

1. Search Space

The search space draws from a set of neural network architectures to define the neural network operators and how they are connected to form a valid network. It determines the way by which architectures are formed and those which are allowed. For example, NASNet [43] introduced a fixed macro architecture where the search consists of finding the appropriate operators to be used within each block from a set of 12 specified operators (see Section 2.5).

2. Search Algorithm

The search algorithm (also known as search strategy) explores the search space by sampling a population of network architectures' candidates. It evaluates the accuracy of the model using a specific *evaluation methodology*. The measured accuracy will then guide the search strategy to converge towards promising architectures in the search space (see Section 2.6).

3. Evaluation Methodology

The evaluation component trains the architecture on the desired dataset, which often takes considerable time. Many NAS algorithms have incorporated several techniques to speed up the training process such as early stopping or surrogate models (see Section 2.7).

NAS has proven its efficiency by proposing different models in object detection [44] and image classification [45]. However, these models are often composed of millions of parameters and require billions of floating-point operations (FLOPs). This causes a large memory footprint and computation and consequently prevents their usage in resource-constrained systems. Additionally, these models might require specific hardware (GPUs, TPUs, etc.) to allow their deployment in a reasonable time or in real-time applications.

Integrating hardware awareness in the search loop (i.e. HW-NAS) has attracted several researchers and has opened up interesting new research directions over the past few years. Some HW-NAS efforts have achieved SOTA results and have balanced the trade-off between accuracy and hardware efficiency. For example, FBNet [46] has achieved interesting results on ImageNet by using an objective function that minimizes both the cross-entropy error, which leads to better accuracy, and the latency, which penalizes inefficient networks.

This chapter provides a detailed overview of existing HW-NAS research efforts and categorizes them according to their goals and problem formulation. With this survey, we provide a concise review of the NAS variants that focus on precision and hardware awareness.

2.3 Methodologies for Efficient Deep Learning

Current DL models are getting bigger and bigger; especially with the arrival of deep foundation models. However, the compute capacity at the edge is significantly low, which does not match this increasing complexity. This has motivated the research community to find innovative ways to reduce the DL models' size, their required number of floating operations, and their inference latency. This section presents an overview of the efficient DL techniques and where HW-NAS is situated among them.

Figure 2.4 illustrates the taxonomy of different techniques used to optimize DL models. We put in red, blocks, and elements we focus on in this thesis.

Model Compression: Model compression aims to apply to standard DL models, such as ResNet or AlexNet, optimizations that will decrease the model size and the number of FLOPs. This model compression is applied while trying to maintain an acceptable level of accuracy. Relevant surveys [15, 47] on model compression classify the optimizations into these four classes:

- Compact Model: This technique modifies the standard operations used in DNNs. In a CNN, the standard convolution is replaced by more flexible convolution arithmetics that expand the number of feature maps and decrease the number of parameters such as dilated convolution [48] or separable depthwise convolution [49]. In an Recurrent Neural Networks (RNN), cells like S-LSTM [50], or JANET [51] simplify the gates and decrease the number of parameters compared to a regular Long Short-Term Memory Networks (LSTM).

- Tensor Decomposition: a tensor is the fundamental data structure used in machine learning. It can represent vectors, matrices, and even n-dimensional arrays. Therefore, shrinking the tensors allows for accelerating DNNs and reducing their size. Tensor decomposition is an extension of the matrix decomposition

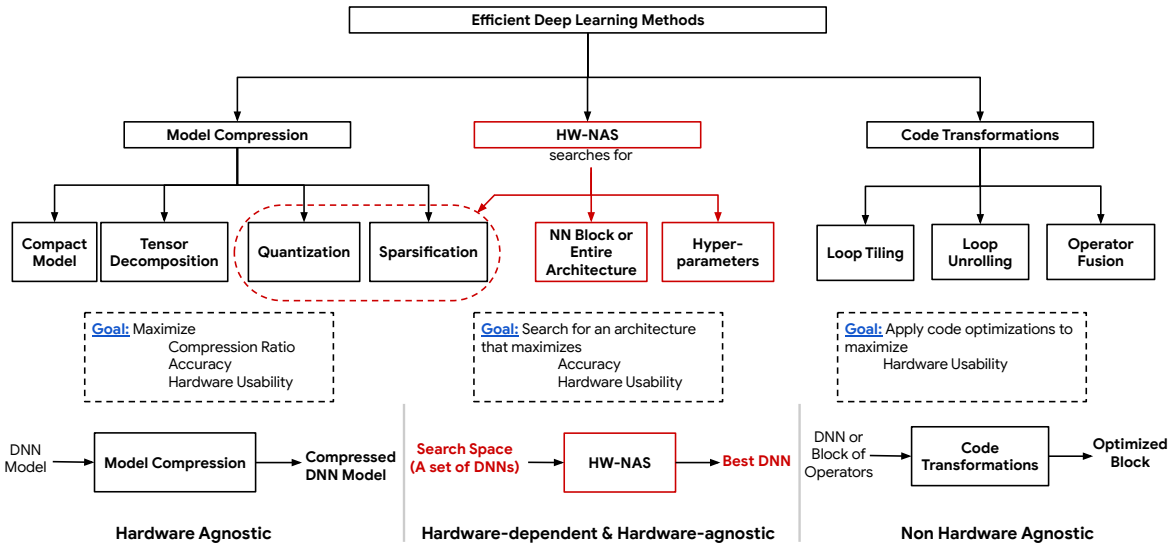

Figure 2.4: Overview of efficient deep learning strategies.

techniques used in mathematical settings. Equation 2.1 formulates the matrix decomposition system where the number of parameters of A and B combined is smaller than the number of parameters of M.

$$M = AB \text{ with } M \in \mathbb{R}^{m \times n}, A \in \mathbb{R}^{m \times r}, B \in \mathbb{R}^{r \times n} \tag{2.1}$$

Note that tensor networks including hierarchical tensor representation (HT) [52] and tensor train decomposition (TT) [53] achieve higher compression rates in a fully-connected network since they usually contain more redundancy.

- Quantization: In DL, quantization [54] refers to converting data objects from a 32-floating point to lower precision or a fixed point integer or even binary. These data objects can be the weights of the layer, the activations (the input data's internal representation), the error value, the gradient values, or the weight update. Each method differs with the chosen bandwidth and the data objects that are quantized.

- Network Sparsification or pruning [55, 56] attempts to compress the model by pruning some weights (edges) or operations (nodes). Usually, the decision of pruning is taken based on its importance, which is directly the weight values or learned via an attention layer.

HW-NAS: Another efficient DL technique is HW-NAS. In HW-NAS, we search for the architecture that maximizes the accuracy and hardware usability among a set of architectures. Note that some HW-NAS can be considered under the model compression techniques as they search for the best bit-width or the best way to prune. We further detail the search for hyperparameters, NN Block, or full architectures in Section 4.

Code Transformations: An alternative approach that is gaining more attraction these recent years is to apply some code transformations that optimize the DNNs on the operator level [57]. These transformations are hardware-specific and require a compiler to apply the right transformation for the right hardware platform automatically. A variety of DL compilers have been developed to apply these transformations, among them: Tiramisu [58], TVM [59], and XLA [60].

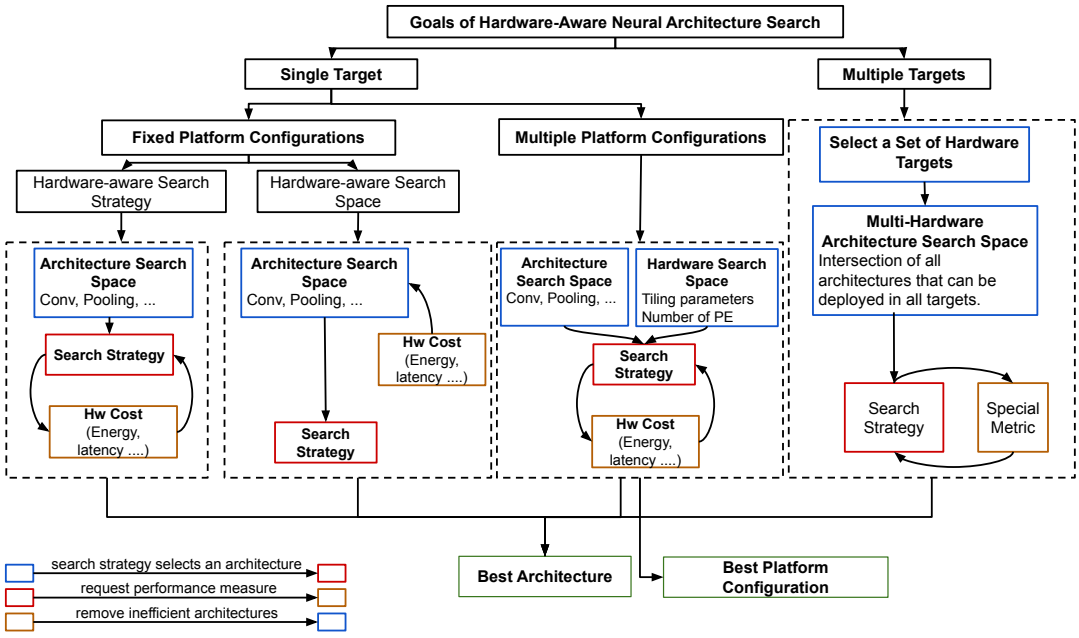

Figure 2.5: Overview of different hardware-aware NAS designs.

2.4 Taxonomy of HW-NAS

Unlike conventional NAS, where the goal is to find the best architecture that maximizes model accuracy, HW-NAS has multiple goals and multiple views of the problem. We can classify these goals into three categories (See figure 2.5 from left to right) :

- **Single Target, Fixed Configuration**: Most of existing HW-NAS fall under this category. The goal is to find the best architecture in terms of accuracy and hardware efficiency for one single target hardware. Consequently, if a new hardware platform has to be used for the NAS, the whole process must be re-executed with the new hardware details to calculate the new hardware's cost. These methods generally define the problem as a constrained or multi-objective optimization problem [46, 61, 62]. Within this category, two approaches are adopted:
  - *Hardware-aware search strategy* where the search is defined as a multi-objective optimization problem. While searching for the best architecture, the search algorithm calls the traditional evaluator component to get the accuracy of the generated architecture but also a special evaluator that measures the hardware cost metric (e.g., latency, memory usage, energy consumption). Both model accuracy and hardware cost guide the search and enable the NAS to find the most efficient architecture.
  - *Hardware-aware Search Space* where a restricted pool of architectures is used. Before the search, either the operators' performance on the target platform is measured or a set of rules that will refine the search space is defined. Refining the search space allows for to elimination of all the architectures' operators that do not perform well on the target hardware. For example, HURRICANE [63] uses different operator choices for three types of mobile processors: Hexagon DSP, ARM CPU, and Myriad Vision Processing Unit (VPU). Accumulated domain knowledge from prior experimentation on a given hardware platform helps narrow down the search space. For instance, they do not use depthwise convolutions for CPU,

squeeze, and excitation mechanisms for VPU and they do not lower the kernel sizes for a DSP. Such gathered empirical information helps to define three different search spaces according to the targeted hardware platform. Note that, after defining the search space with these constraints, the search strategy is similar to the one used by conventional NAS, which means that the search is solely based on the accuracy of the architecture, and no other hardware metric is incorporated.

- **Single Target, Multiple Configurations**: the goal of this category is not only to get the most optimal architecture that gets the best accuracy but also to get an optimal architecture with latency guaranteed to meet the target hardware specification. For example, the authors of FNAS [64] define a new hardware search space containing the different FPGA specifications (e.g., tiling configurations). They also use a performance abstraction model to measure the latency of the searched neural architectures without doing any training. This allows them to quickly prune architectures that do not meet the target hardware specifications. In [65], the authors use the same approach for ASICs and define a hardware search space that contains various ASIC templates.

- **Multiple Targets**: In this third category, the goal is to find the best architecture when given a set of hardware platforms to optimize for. In other words, we try to find a single model that performs relatively well across different hardware platforms. This approach is the most favorable choice, especially in mobile development as it provides more portability. This problem was tackled by [66, 67] by defining a multi-hardware search space. The search space contains the intersection of all the architectures that can be deployed in the different targets. Note that, targeting multiple hardware specifications at once is harder as the best model for a GPU, can be very different from the best model for a CPU. For example, in general, wider models are more appropriate for GPU while deeper models are preferable on CPUs.

2.5 Search Spaces

Two different search spaces have been adopted in the literature: the *Architecture Search Space* and the *Hardware Search Space*.

2.5.1 Architecture Search Space

Definition

The Architecture Search Space is a set of feasible architectures from which we want to find an architecture with high performance. Generally, it defines a set of basic network operators and the manner by which these operators can be connected to construct the computation graph of the model.

We distinguish two approaches to designing an architecture search space:

1. *Hyperparameter optimization for a fixed architecture:* The objective is limited to optimizing the architecture hyperparameters, such as the number of channels, the stride, or the kernel size.

2. *Operator search space:* The search space allows the optimizer to choose connections between operations and the type of operation within each layer.

Both approaches have their advantages and disadvantages but it is worth mentioning that although the former approach reduces the search space size, it requires considerable human expertise to select the macro-architecture and introduces a strong

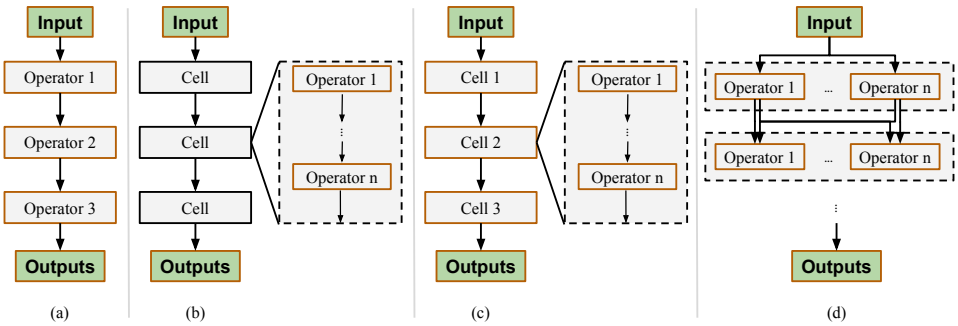

Figure 2.6: Architecture search spaces types. (a) Global search space, (b) Cell-based search space, (c) Hierarchical search space, and (d) supernetwork search space. In orange the operators considered during the search.

bias. Whereas the latter approach decreases the human bias but considerably increases the search space size and hence the search time.

Generally, in the latter approach, we distinguish three types (See figure 2.6):

- **Layer-wise Seach Space**, where the whole model is generated from a pool of operators. FBNet Search Space [46], for example, consists of a layer-wise search space with a fixed macro architecture that determines the number of layers and dimensions of each layer where the first and last three layers have fixed operators. The remaining layers need to be optimized.
- **Cell-based Search Space**, where the model is constructed from repeating fixed architecture patterns called blocks or cells. A cell is often a small acyclic graph that represents some feature transformation. The cell-based approach relies on the observation that many effective handcrafted architectures are designed by repeating a set of cells. These structures are typically stacked and repeated a number of times to form larger and deeper architectures. This search space focuses on discovering the architecture of specific cells that can be combined to assemble the entire neural network. Although cell-based search spaces are intuitively efficient to look for the best model in terms of accuracy, they lack flexibility when it comes to hardware specialization [46, 62].
- **Hierarchical Search Space**, works in 3 steps: First, the cells are defined, and then bigger blocks containing a defined number of cells are constructed. Finally, the whole model is designed using the generated cells. MNASNet [62] is a good example of this category of search spaces. The authors define a factorized hierarchical search space that allows more flexibility compared to a cell-based search space. This allows them to reduce the size of the total search space compared to the global search space.
- **Supernetwork Search Space**, the idea behind supernetworks is to create a large space of possible architectures that can be efficiently explored to find the best-performing network for a given task. In a supernetwork, the weights of the subnetworks are not fixed but are instead treated as hyperparameters that can be learned during training. This allows for a more flexible search space that can better adapt to the specific requirements of the task at hand.

In existing NAS research works the authors define a macro-architecture that generally determines the type of networks considered in the search space. When considering CNNs, the macro architecture is usually identical to the one shown in figure 2.1. Therefore, many works [46, 61, 62, 66, 68] differ in the number of layers, the set of operations and the values of the possible hyperparameters. Recently, the scope of network type is changing. For instance, NASCaps [69] changes their macro-architecture

to allow the definition of capsules. Capsules network [70] are basically cell-based CNNs where each cell (or capsule) can contain a different CNN architecture.

Other works like [71, 72] focus on transformers and define their macro-architecture as a transformer model. The search consists of finding the number of attention heads and their internal operations. When dealing with hyperparameters only, the macro architecture can define a variety of network types. Authors in [73, 74] mix different definitions, transformers + CNN and transformers + RNN respectively. They define a set of hyperparameters that encompasses the pre-defined parameters for different network types at the same time.

Lately, more work [61, 75] have been considering the use of over-parameterized networks (i.e. supernetworks) to speed up the NAS algorithms. These networks consist of adding architectural learnable weights that select the appropriate operator at the right place. Note that these techniques have been applied to transformers as well [76].

In some research efforts, the pool of operators/architectures is refined with only the models that are efficient in the targeted hardware [66, 67]. The search space size is considerably reduced by omitting all the architectures that cannot be deployed.

Fine-grained Search Space for NAS

In recent research endeavors, various approaches have been proposed to explore novel architectures without human design bias. The term "*fine-grained search spaces*" refers to search spaces that consist of a set of mathematical and code-level functions. Few works consider these search spaces for NAS. The reason is due to the large number of possible operators created from this search space, it is highly impractical to explore it.

AutoML Zero [77] is the only AutoML tool that defines a search space from basic operators. Their goal is to search for the end-to-end learning pipeline, i.e., from architecture building to optimizing the loss function. This work is a seminal step towards the holy grail of AutoML: automatically designing a network and training pipeline for any given dataset. However, their methodology took a tremendous amount of time to come up with already human-designed logistic regression. Recently, BANAT [78] proposes an algebraic representation of the architecture to enable a more general search space definition. This is a promising strategy for efficiently and effectively searching over our huge search spaces.

Other works [78, 79, 80] consider modifying a single operator, namely batch normalization. EvoNorms [79] evolves the normalization operator from basic mathematical functions. They discover novel implementations and functions for the normalization and activation fusion which improved the overall average precision of multiple standard models.

Due to their recent application and high time complexity, low-level search spaces are only considered in NAS with a task-specific objective. In other terms, our work is the first to search for adapting the model for resource-constrained devices using a low-level search space.

To address the efficiency aspect of exploring such complex search spaces, GOS adopts an operator replacement strategy. By iteratively replacing the least efficient operator in the architecture, GOS streamlines the search process, enabling more efficient exploration and discovery of high-performing architectures.

2.5.2 Hardware Search Space (HSS)

Some HW-NAS methods include an HSS component that generates different hardware specifications and optimizations by applying different algorithmic transformations to fit the hardware design. This operation is done before evaluating the model. This co-exploration is effective but increases the search space-time complexity significantly. If we take Field Programmable Gate Arrays (FPGA) as an example, their design space

may include IP instance categories, IP reuse strategies, quantization schemes, parallel factors, data transfer behaviors, tiling parameters, and buffer sizes. It is arguably impossible to consider all these options as part of the search space due to the added search computation cost. Therefore, many existing strategies limit themselves to only a few options.

Definition

The Hardware Search Space is a set of all hardware configurations and properties that can impact the mapping and execution of a neural network. Finding the appropriate set of properties helps design efficient hardware platforms and ensure high efficiency.

Hardware Search Space (HSS) can be further categorized as follows:

- **Parameter-based:** The search space is formalized by a set of different parameter configurations. Given a specific data set, **FNAS** [64] finds the best-performing model, along with the optimization parameters needed for it to be deployed in a typical FPGA chip for DL. Their HSS consists of four tiling parameters for the convolutions. **FNASs** [81] extends **FNAS** by adding more optimization parameters such as loop unrolling. The authors in [82, 83] used a multi-FPGA hardware search space. The search consists of dividing the architecture into pipeline stages that can be assigned to an FPGA according to its memory and DSP slices, in addition to applying an optimizer that adjusts the tiling parameters. Another example is presented in [84], where the adopted approach takes the global structure of an FPGA and adds all possible parameters to its hardware search space including the input buffer depth, memory interface width, filter size, and the ratio of the convolution engine. [85] searches the internal configuration of an FPGA by generating simultaneously the architecture hyperparameters, the number of processing elements, and the size of the buffer. **FPGA/DNN** [86] proposes two components: *Auto-DNN* which performs hardware-aware DNN model search and *Auto-HLS* which generates a synthesizable C code of the FPGA accelerator for the explored DNNs. Additional code optimizations such as buffer reallocation and loop fusion on the resulting C-code are added to automate the hardware selection.

- **Template-based:** In this scenario, the search space is defined as a set of pre-configured templates. For example, **NASAIC** [65] integrates NAS with Application-Specific Integrated Circuits (ASIC). Their hardware search space includes templates of several existing successful designs. The goal is to find the best model with the different possible parallelizations among all templates. In addition to the tiling parameters and bandwidth allocation, the authors in [87] define a set of FPGA platforms and the search finds a coupling of the architecture and FPGA platform that fits a set of pre-defined constraints (e.g., max latency 5ms)

In general, we can classify the targeted hardware platforms into 3 classes focusing on their memory and computation capabilities:

- Server Processors: This type of hardware can be found in cloud data centers, on-premise data centers, edge servers, or supercomputers. They provide abundant computational resources and can vary from CPUs, GPUs, FPGAs, and ASICs. When available, machine learning researchers focus on accuracy. This class is beyond the scope of our survey. In this paper, we focus on HW-NAS on edge devices where the necessity of efficient deep architectures is at its peak.

- Mobile Devices: With the rise of mobile devices, the focus has shifted to enabling fast and efficient DL on smartphones. As these devices are heavily con-

strained with respect to their memory and computational capabilities, the objective of ML researchers shifts to assessing the trade-off between accuracy and efficiency. Many HW-NAS algorithms target smartphones including FBNet [46] and ProxylessNAS [61] (table 2.1). Additionally, because smartphones usually contain system-on-chips with different types of processors, some research efforts [88] have started to explore ways to take advantage of these heterogeneous systems.

- Tiny Devices: The strong growth in the use of microcontrollers and IoT applications gave rise to TinyML [89]. TinyML refers to all machine learning algorithms dedicated to tiny devices, i.e, capable of on-device inference at extremely low power. One relevant HW-NAS method that targets tiny devices is MCUNet [90], which includes an efficient neural architecture search called TinyNAS. TinyNAS optimizes the search space and handles a variety of different constraints (e.g., device, latency, energy, memory) under low search costs. Thanks to the efficient search, MCUNet is the first to achieve >70% ImageNet top-1 accuracy on an off-the-shelf commercial microcontroller. MCUNet also includes an inference engine to identify the right set of code optimization for the searched architecture.

Table 2.1: Classification of HW-NAS based on their targeted Hardware.

Targeted HW	References
Central Processing Unit (CPU)	[63, 84, 88, 91, 92, 93, 94, 95, 96, 97]
Graphics Processing Unit (GPU)	[72, 73, 85, 92, 94, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128]
Micro-Controller Unit (MCU)	[46, 90, 129, 130, 131, 132]
Mobile Devices	[42, 46, 61, 62, 71, 72, 95, 107, 133, 134, 135, 136, 137, 138, 139]
Application-Specific Integrated Circuit (ASIC)	[63, 69, 82, 83, 140, 141, 142, 143, 144]
Edge Tensor Processing Unit (TPU)	[72, 136, 145]
In-Memory-Computing (IMC)	[146, 147]

2.5.3 Current Hardware-NAS Trends

Figure 2.7 shows the different types of platforms that have been targeted by HW-NAS in the literature. In total, we have studied 395 original hardware-aware NAS papers. By target, we mean the platform that the architecture is optimized for. Usually, the search algorithm is executed in a powerful machine, but that is not the purpose of our study. We focus here on the platform where the inference is executed. "*No Specific Target*" means that the HW-NAS incorporates hardware agnostic constraints into the objective function such as the number of parameters or the number of FLOPs. In the figure, the tag "*Multiple*" means multiple types of processing elements have been used in the HW platform. Table 2.1 gives the list of references per targeted hardware.

In figure 2.7 (left), we note that the number of research papers targeting GPUs and CPUs has more or less remained constant. However, we can clearly see that

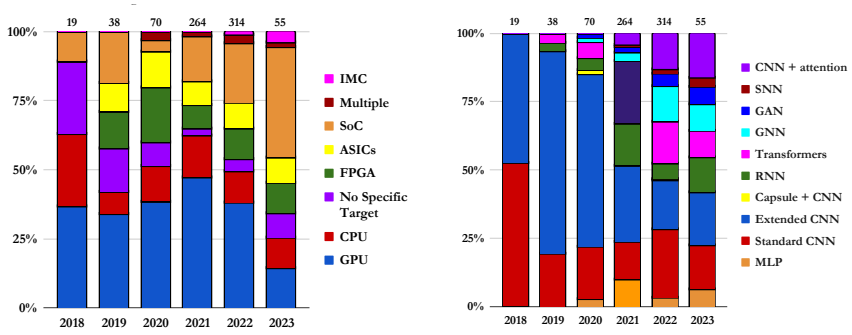

Figure 2.7: Statistics on targeted platforms and type of networks described by the HW-NAS search spaces

FPGAs and ASICs are gaining popularity over the last 3 years. This is consistent with the increasing number of DL edge applications. Another interesting target for HW-NAS is In-memory Computing (IMC). This novel hardware platform optimizes the Matrix-Vector Mutliplication (MVM) and becomes one of the most promising hardware platforms for AI. We discuss searching for a suitable architecture for IMC in section 2.9. Another recent work [66, 67] is to consider multiple platforms at once. This was particularly suitable for mobile settings with different existing System-on-Chips (SoC).

Figure 2.7 (right) illustrates the different DNN operations that compose the architecture search space. We notice that most NAS target CNN architectures. We divide this category into two groups, *standard CNN* which only utilizes a standard convolution, and *extended CNN* which involves special convolution operations such as the depthwise separable convolution or grouped convolutions. However, recent works have started to explore more operators by incorporating RNN, capsule networks [69], transformers [148],Generative Adversarial Networks (GAN) [149] and more recently Graph Convolutional Neural Networks (GCN) [150].

2.6 Optimization strategies

In this section, we describe how the HW-NAS problem is formulated and solved using an optimization strategy.

2.6.1 Hardware-aware NAS Problem Formulation

NAS is the task of finding a well-performing architecture for a given dataset. It is cast as an optimization problem over a set of decisions that define different components of deep neural networks (i.e., layers, hyperparameters). This optimization problem can simply be seen as formulated in equation 2.2.

$$\max_{\alpha \in A} f(\alpha, \delta) \tag{2.2}$$

We denote the space of all feasible architectures as A (also called search space). The optimization method is looking for the architecture α that maximizes the performance metric denoted by f for a given dataset δ. In this context, f can simply be the accuracy of the model.

Although it is important to find networks that provide high accuracy, these NAS algorithms tend to give complex models that cannot be deployed on many hardware devices. To overcome this problem, practitioners consider other objectives, such as the number of model parameters, the number of floating-point operations, and device-specific statistics like the latency or the energy consumption of the model. Different

formulations were used to incorporate the hardware-aware objectives within the optimization problem of neural architecture search. We classify these approaches into two classes, single and multi-objective optimization. The single objective optimization can be further classified as two-stage or constrained optimization. Similarly, the multi-objective optimization approach can be further classified as single or multi-objective optimizations. Please refer to figure 2.8 for a summary of these approaches. These 2 classes are further detailed with examples from the literature in the following sections.

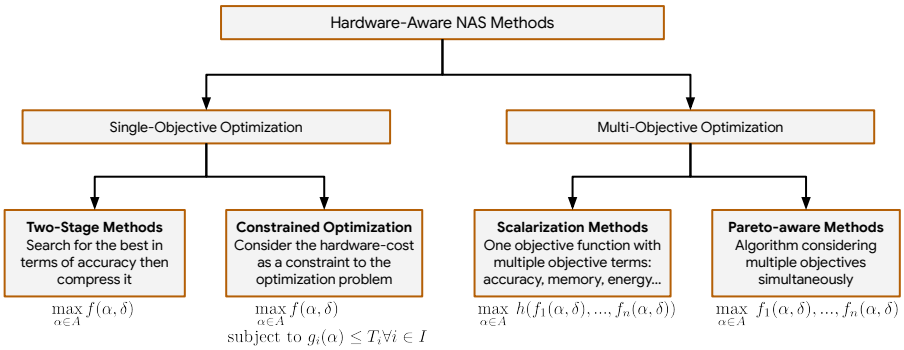

Figure 2.8: HW-NAS problem formulations.

Single-Objective Optimization

Definition

In this class, the search is realized considering only one objective to maximize, i.e., accuracy. Most of the existing work in the literature [61, 62, 84, 87, 151], that tackle the hardware-aware neural architecture search, try to formulate the multi-objective optimization problem into a single objective to better apply strategies like reinforcement learning or gradient-based methods.

We can divide this class into two different approaches: Two-stage optimization and constrained optimization.

Two-Stage optimization: In this first category, we retain the original formulation of the NAS problem and then specify the model for deployment. This approach is sub-optimal, as the final architecture proposed by the NAS is not always the one that gives the best performance on the hardware device. Two-stage optimization consists of applying NAS methods to obtain the best-performing architecture and then in a second stage, specializing this architecture for deployment on a target hardware platform. This specialization performs a series of optimization to fit the hardware requirements. In [116] the authors apply a reinforcement learning agent to find the best quantization bit-width and pruning level after selecting the most accurate model. In reference [42] the authors search over a pre-trained and selected architecture to find the most efficient one in terms of latency and energy consumption.

Constrained optimization: In this approach, the hardware-aware characterizations are considered constraints in the original NAS formulation. The constraints take the form of thresholds to be respected. In this approach, inference time, energy consumption and memory occupation are examples of constraints. The conditions are added as constraints to the optimization problem to enforce requirements like fewer parameters or faster inference time. The threshold and the trade-off between

different constraints can be adapted to practical requirements. For such cases, the single-objective optimization problem defined in Equation 2.2 turns into a constrained optimization problem defined by

$$\begin{aligned} &\max_{\alpha \in A} f(\alpha, \delta) \\ &\text{subject to } g_i(\alpha) \leq T_i \; \forall i \in I \end{aligned} \tag{2.3}$$

Here, g_i corresponds to the different constraints taken into account (e.g., latency, memory, energy consumption) and T_i denotes the respective threshold. As most of the optimization methods used by NAS (i.e. reinforcement learning and evolutionary algorithms) were designed for unconstrained optimization problems, this formulation is hard to be adopted directly. Therefore, many researchers turned to penalty methods to transform the equation into a single objective function that contains the hardware constraints as well as the accuracy measurement [46, 61, 62]. For example, MNASNet [62], uses equation (2.4), where f is the accuracy measurement function, LAT is the latency of the model and T is the threshold. They use a learnable parameter w to control the effect of the hardware constraints on the global objective function.

$$\max_{\alpha \in A} \; f((\alpha) \cdot [LAT(\alpha)/T]^w \tag{2.4}$$

ProxylessNAS [61] uses a loss function that comprises of the cross-entropy (CE) loss and hardware-aware constraints.

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda_1 ||w||^2 + \lambda_2 E[\text{latency}] \tag{2.5}$$

Equation 2.5 illustrates the loss calculated by the reinforcement learning agent used by ProxylessNAS. λ_1 and λ_2 are learnable parameters that adjust the effect of the efficiency of the overall loss. Specifically, a policy is learned that decides whether to add, remove or keep a layer as well as whether to alter its number of filters.

FLASH [146], a dedicated NAS for compute-in-memory devices, optimized an objective function that consists of minimizing the quotient of the accuracy and HW efficiency. They used three different metrics for HW efficiency: Latency, area, and memory consumption.

Multi-Objective Optimization

Definition

To handle multiple fronts in the formalism of a multi-objective optimization problem defined as:

$$\max_{\alpha \in A} \; f_1(\alpha, \delta), f_2(\alpha, \delta), ..., f_n(\alpha, \delta) \tag{2.6}$$

In this scenario, there is often no single optimal solution that simultaneously maximizes every objective function. Additionally, some objective functions can be conflicting. For instance, trying to minimize the number of parameters while aiming at maximizing the accuracy. In these situations, the task boils down to finding Pareto-optimal solutions.

These techniques can:

1. Transform the problem to a single-objective optimization using the scalarization method, also called the weighted sum method or,
2. Solve the multi-objective optimization problem using dedicated heuristics or meta-heuristics such as genetic algorithms or tabu search [114]. In general, this second approach provides not one optimal solution but a set of solutions that form the optimal Pareto front of the multi-objective optimization problem.

Scalarization Methods: One way to solve the multi-objective optimization problem is to use a scalarization approach. Equation 2.7 formulates this method. We use a parameterized aggregation function h to transform the multi-objective optimization problem into a single-objective optimization problem.

$$\max_{\alpha \in A} \; h(f_1(\alpha, \delta), f_2(\alpha, \delta), ..., f_n(\alpha, \delta)) \tag{2.7}$$

The function h can be a weighted sum, a weighted exponential sum, a weighted min-max, or a weighted product. However, in most situations, not all Pareto optimal solutions can be found in solving this problem with a fixed setting of the weights. Therefore, the problem is solved for multiple values of the vector w which requires multiple optimization runs. To mitigate the cost of having multiple runs, researchers commonly use a set of fixed weights according to the desired trade-off between the objectives and the practitioners' preferences. Thanks to the scalarization, the problem becomes a single-objective optimization problem which can be solved by any optimizing methods discussed in Section 2.6. For example, [99] proposes to use the weighted sum as the objective function. The proposed formulation of this function is described by equation 2.8. ACC refers to the accuracy metric, E refers to the energy consumed by the architecture α, and w is a learned parameter to adjust the effect of the energy on the reward function.

$$\max_{\alpha \in A} \; w \cdot ACC(\alpha, \delta) - (1 - w) \cdot E(\alpha) \tag{2.8}$$

Pareto-aware Methods: Recently, there has been an emerging approach to tackle the design of neural architectures as a pure multi-objective problem. This approach considers the conflicting objectives of task-specific performance and hardware efficiency, aiming to strike a balance between the two. The outcome is a set of architectures that lie on the Pareto front, representing the trade-off between performance and efficiency. This Pareto front provides a valuable resource for selecting architectures that best align with the requirements of a target device or deployment scenario. The final set of architectures offers flexibility and adaptability, allowing practitioners to choose the most suitable architecture based on their specific needs, whether it be optimizing for accuracy, latency, power consumption, or other desired criteria. In addition, these architectures can be leveraged to adapt to the hardware battery life, which is a crucial consideration for mobile and battery-powered devices. When the battery is low, selecting an architecture from the Pareto front that prioritizes energy efficiency, can extend the battery life of their devices without compromising significantly on task-specific performance.

For example, the elitist evolutionary algorithm Non-dominated Sorting Genetic Algorithm (NSGA) [152] is used. In this algorithm, the architectures are divided into fronts based on their dominance. The architecture in the i-th front is only dominated by all the architectures in the $1, \ldots, i-1$ fronts. Within each front, the architectures are prioritized by the crowding distance, which is computed by the sum of all the neighborhood distances across all the objectives. HW-NAS works [104, 113, 153] have been using the NSGA-II algorithm to ensure the exploration of diverse architectures in the search space. Moreover, NSGANet [104] uses Bayesian Optimization to profit from search history. MoreMNAS [153] uses a hybrid search strategy combining NSGA-II with reinforcement learning to regulate arbitrary mutations.

2.6.2 Search Algorithms

In this section, we will explore several search algorithms that are commonly used in HW-NAS. Figure 2.9 shows the popularity of the most used search algorithms. These algorithms are borrowed from the operational research field and applied to the HW-NAS problem.

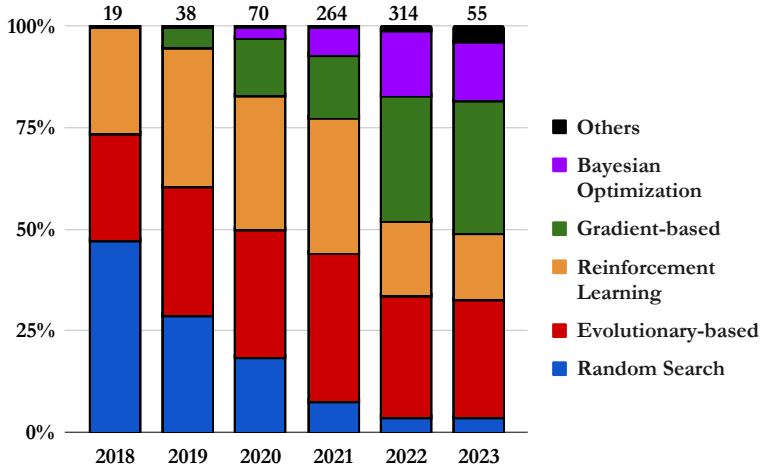

Figure 2.9: Commonly used search algorithms

Reinforcement Learning (RL)

Most HW-NAS methods use reinforcement learning to search for the best architecture [62, 65, 83, 84, 87, 151, 154, 155] because the NAS problem is easily modeled as a Markov Decision Process. The RL controller samples an architecture from the search space and is rewarded according to its accuracy and hardware cost. The controller will then adjust its weights to generate better models. Different works differ on how they represent the agent's policy (set of actions) and how they optimize it.

Using reinforcement learning, **MNASNet** [62] tries to find the Pareto optimal solution of the objective function described in equation 2.4. It uses a *sample-eval-update* loop to train its RNN controller. To generate a block, the controller will first choose two hidden states (i.e., outputs of previous blocks) as inputs. Then, it will select an operation to apply to each one of them. Finally, it selects a combination method (e.g., addition or concatenation) to obtain the final output of the block. This implementation has been defined by NASNet [156] before. Once a model is sampled, it is trained on the target task to get its accuracy and deployed on real phones to get its latency. The system computes the reward value and adjusts the controller parameters accordingly.

A similar approach was used by **FPNet** [151] but the RNN controller predicts only the architectural hyperparameters (i.e., number of filters, filter height and width, stride height and width. etc.) while keeping the macro architecture fixed.

Codesign-NAS [84] proposes to use reinforcement learning to explore both architecture and hardware search spaces. The authors investigated three RL-based search strategies and use the REINFORCE algorithm to improve the accuracy and efficiency of image classification on FPGA. The first strategy consists of combining the two search spaces and updating the CNN and accelerator options at the same time. The second method uses two different controllers, one to learn the CNN architecture and another one to select the best FPGA options. The last one is a conventional NAS where they separately search for the best model in terms of accuracy then as a completely separate step look for the most efficient model. An expected result is that this latter strategy gives bad results when it comes to the constrained environment. An interesting result is that the second method (i.e., phase search) seems to be the most promising achieving higher rewards in most of their experiments.

Evolutionary Algorithm (EA)

Another popular strategy in conventional NAS is the use of evolutionary algorithms [157, 158]. Generally, evolutionary search algorithm-based NAS evolves a population of models, samples some models to generate offsprings by applying some mutations, and finally evaluates the fitness of the offsprings and updates the new generation by adding the best ones to the population. When it comes to integrating the hardware constraints to the NAS algorithms, some research efforts have used evolutionary algorithms [63, 69, 76, 90].

TinyNAS [90] uses a hardware-aware search space approach. It first optimizes the search space to fit the tiny and diverse resource constraints and then performs an evolution search algorithm within the optimized search space to find the most accurate model. The evolution search is performed on a trained super network that contains all the possible sub-networks. First, they sample 100 satisfying networks that fit the resource constraints. Then, just like conventional NAS, they measure the validation accuracy of each model, mutate the offspring and update the new generation. This process is repeated for 30 iterations.

Once-for-all [42] proposes a special technique to train the supernetwork using gradient-based methods called **progressive shrinking**. A supernetwork is another name for an over-parameterized network. In their version, the supernetwork includes the highest values for the width, depth and channel number in each convolution. The process begins by training for larger width, depth, and channel numbers and then fine-tune the network for the smaller sizes. Once the supernetwork is trained, they specialize the network using an evolutionary algorithm for different hardware settings with a combined cost of accuracy and latency.

HAT [76] is interesting because to our knowledge it is the only work that is trying to search for efficient transformers, targeting NLP tasks. The authors perform an evolutionary search with hardware latency constraints on a SuperTransformer. This means the engine only adds SubTransformers with latency smaller than the hardware constraint to the population.

NASCaps [69] proposes a NAS framework that generates Capsule Networks along with CNN. The proposed framework uses a multi-objective Genetic Algorithm (based on NSGA-II, see section 4.2) to pick the Pareto optimal solutions. The two key operations are the *crossover and mutation*. In the crossover, they define the splitting point by ensuring that the generated DNN is made up of at least on initial convolution layer and a minimum of two capsules. No standard convolution is placed between two capsule layers. The mutation is performed by randomly choosing one of the layer descriptors from the candidate network and modifying one of the main parameters of the selected layer.

Gradient-Based Methods

Arguably the most promising search strategy promising in terms of results, Gradient-based methods are increasingly used by hardware-aware NAS [46, 61, 67, 75, 159] and NAS generally. Running the search separated from the evaluation requires a lot of time and computation. Therefore, a common idea is to have a supernetwork that can emulate any child model in the search space. This means that different parts of the graph share weights between their common edges. This idea of *weight sharing* has the advantage of considerably reducing the search time. Gradient-based methods train the supernetwork to simultaneously get the architecture parameters and weights. This technique has been initiated by DARTS [160].

ProxylessNAS [61] is one of the papers pioneering this method. It defines a supernetwork with binary architecture parameters (i.e., 1 implies that the operator is selected in the layer and 0 otherwise). The loss function combines the cross-entropy and the latency to better update the weights and architecture parameters. An approach similar to BinaryConnect [161] is used to update the binary architecture

parameters using an approximation of the gradient w.r.t architecture parameters.

FBNet [46] proposes to use differentiable neural architecture search to discover hardware-aware efficient CNNs. They also use a combination of the cross-entropy and latency to train their supernetwork using stochastic gradient descent. Rather than using binary parameters, they relax the problem of finding the best architecture to finding a distribution that yields to the best model.

SqueezeNAS [75] focuses on semantic segmentation and uses a method very similar to FBNet [46]. **HTAS** [67] uses a gradient-based method to find the best width and depth for their transformable CNNs.

XNAS [159] proposes to use the prediction with expert advice theory [162] for the selection. It leverages the Exponentiated-Gradient algorithm (EG) [163] rather than the classical gradient descent which prevents the decay of architecture weights to promote the selection of arbitrary architectures.

Many recent works [164, 165] have provided evidence that weight sharing does not respect the ranking of architecture. Which yield sub-optimal search results. They both suggested using smaller search spaces to alleviate this issue. However, it is meaningless to search in a narrow search space, as a small search space will lead to a very narrow accuracy range.

One major challenge when applying gradient-based optimization is the loss function must be differentiable w.r.t the architectural parameters. Also, we need to check that the incorporated hardware cost is differentiable. Several methods have been used to make the gradient computation over discrete variables possible and extend the original DARTS proposal [166].

- **Gumbel Softmax** [167] One way to relax the discrete variables is to use the Gumbel Softmax function. It helps insert some random noise following the Gumbel distribution so that the gradient computation is possible. This technique was used by FBNet [46, 168].
- **Estimated Continuous Function** ProxylessNAS [61] mimics the concept of BinaryConnect [161]. They approximately estimate the gradient w.r.t the architecture parameters using the gradient w.r.t the binary gates. To reduce the computational cost, they compare the gates two-by-two by factorizing the task of using one out of N paths into multiple binary selection tasks.
- **REINFORCE algorithm** An alternative approach to BinaryConnect is also proposed by ProxylessNAS [61]. They utilized REINFORCE to train the binarized weights. Furthermore, they combine the gradient-based update rule to the REINFORCE updates to form a new general update rule for the architecture parameters.

OFA [169] introduces a technique for searching optimal architectures adaptable to various hardware and latency constraints. It involves training a supernetwork with multiple sub-networks of different architectures and computational costs. Sub-networks are randomly sampled and trained on different data batches, and their accuracy and computational cost are evaluated on a validation set. The best-performing sub-networks are selected for deployment.

While OFA focuses on finding a single architecture for diverse platforms, our methodology, PRP-NAS [6], aims to simultaneously optimize multiple objectives, resulting in a set of Pareto-optimal solutions. We do not apply for any post-search specialization.

Bayesian Optimization (BO)

Bayesian Optimization (BO) is a powerful technique that has been increasingly applied in HW-NAS to efficiently identify the best neural network architecture for a given hardware platform. Generally, BO is used for hyperparameter optimization.

This approach leverages Bayesian statistical methods to optimize a surrogate function that models the performance of different neural architectures on the hardware platform. The use of Bayesian optimization balances the exploration-exploitation trade-off, using an acquisition function to guide the search toward promising regions of the architecture space. This ultimately leads to more efficient and effective searches, as the number of architectures that need to be evaluated on the hardware platform is reduced. For instance, [170] propose a hardware-aware Bayesian optimization framework for NAS, which they evaluate on several datasets and hardware platforms. The results demonstrate that their approach effectively discovers architectures that outperform state-of-the-art networks while satisfying hardware constraints.

Random Search (RS)

The most convenient and easiest search strategy to implement is the random search strategy. Most of the existing works that have adopted random search optimize the architectural hyperparameters within a fixed macro architecture. They argue that it is more important to design the architecture search space for the targeted hardware platform than to complicate the search strategy and incorporate the hardware constraints in the objective function. NASNet [43], for example, tried both methods (i.e., reinforcement learning and random search). They found that with reinforcement learning the results are slightly better (Top-1 accuracy on CIFAR-10, 0.912 - 0.925).

Li et al. [171] investigate the use of random search on two standard NAS benchmarks (i.e., PTB and CIFAR-10). They use an approach that is similar to ProxylessNAS [61] which allows them to train a single network at a time and thus reduce the memory footprint with weight sharing. As a result, they show that random search with early-stopping is a competitive NAS baseline as it outperforms ENAS [172]. Furthermore, random search with weight-sharing outperforms random search with early stopping, achieving State-Of-The-Art (SOTA) results on Physikalisch Technische Bundesanstalt (PTB). A more recent investigation [107] showed that random search cost time is not negligible and is comparable to NAS methods.
Besides, other HW-NAS uses novel optimization strategies that come from operational research. For example, FLASH [146] uses Simplicial Homology Global Optimization (SHGO) [173] to search for the optimal architecture for an IMC accelerator. SHGO proves to converge with fewer samples than RL.

2.7 HW-NAS Estimation Strategies

In HW-NAS, one needs to estimate task-specific performance and the model's hardware efficiency.

An important component in HW-NAS is the hardware cost measurements. First of all, many metrics have been used in order to evaluate the hardware efficiency of an architecture including the number of FLOPs, the number of parameters, the execution time of the inference named also latency, the energy consumption, the memory footprint, the area of the hardware platform, etc. In this section, the existing approaches for hardware cost measurements are detailed:

- Floating Point Ops per Second (FLOPS) & Model Size: The first HW-NAS approaches that were published in 2016 and 2017 [174, 175] use the number of parameters and number of FLOPs as an objective function to minimize. These techniques assume that the number of operations is positively correlated to the execution time. However, recent works have proved that two models can have the same number of FLOPs but different latencies [63, 76, 176]. For example, NASNet-A and MobileNetV1 have roughly similar numbers of FLOPs, yet, NASNet-A can have slower latency due to the hardware-unfriendly structure. Therefore, using FLOPs as the hardware cost metric is not efficient and may return suboptimal models. On the other hand, using the model size represented

Table 2.2: Summary of Hardware Cost Estimation Methods.

Method	Implementation	HW Cost Metric	References
Real-world measurements	The sampled model is executed on the hardware target	Latency	MNASNet [62] NetAdapt [177] Z. Guo et al. [133] MCUNet [90]
		Energy	NetAdapt [177] MONAS [99] C. Gong et al. [178]
Lookup Table Models	A lookup table is created beforehand and filled with each operator latency on the targeted hardware. Once the search starts, the system will calculate the overall cost from the lookup table	Latency	FBNet [46] HotNAS [87]
Analytical Estimation	Compute a rough estimate using the processing time, the stall time, and the starting time	Latency	FNAS [154] NASCaps [69] A. Anderson et al. [180] Q. Lu et al. [83]
		Energy	NASCaps [69]
		Memory footprint	NASCaps [69]
		Area	NASAIC [65]
Prediction Model	Build an ML model to predict the cost using architecture and dataset features	Latency	proxylessNAS [61] NASAIC [65] NeuNets [105] LEMONADE [99]

by the number of parameters allows for a reduction of the memory footprint and tends to be considered an automatic method to search for compressed models [140].

- Latency: Searching for low-latency architectures at inference time is crucial for real-world applications such as autonomous driving and traffic control. Moreover, resource-limited devices have latency constraints. Thus, a lot of works consider the latency in their objective function and search for the trade-off between inference time and accuracy.

- Energy Consumption: Energy is usually profiled by the provided hardware platform profilers such as nvprof by NVIDIA. The energy can be formalized either as the peak power consumption or the average power, both metrics are used by different HW-NAS works including [99, 177, 178].

- Area: Another metric that interests chip manufacturers is its area. The goal is to get the smallest chip possible that could run the best model. Authors in [65] use MAESTRO [179] to explore the area and power consumption and search for the best model and best ASIC templates within a set of pre-defined ones. The area of the circuit is also a good indicator of static power consumption. These two values are correlated.

- Memory footprint: We can get the model size by calculating the number of parameters it needs to learn but a more efficient way is to profile how much memory it uses while running; this is the memory footprint. Having a low memory footprint is important for edge devices. These devices are not able to run models with high memory footprints. To reduce the memory footprint in edge devices, techniques like those presented in Section 2.8 are applied.

Table 2.2 summarizes all the methods used to estimate the latency, and energy consumption in different HW-NAS methods.

Real-world measurements provide high accuracy in measuring the hardware efficiency of an architecture. MnasNet [62] uses this method in the exploration. It

achieves 75.2% top-1 accuracy with 78ms latency on a Pixel phone platform, which is 1.8x faster than MobileNetV2 with 0.5% higher accuracy. However, this method considerably slows down the search algorithm by averaging hundreds of runs to get precise measurements. Additionally, this strategy is not scalable and requires that all the hardware platforms are available. This solution could be costly and needs a lot of mobile devices and software engineering work. For this reason many works tend to use prediction models [61, 63, 82, 83, 86, 154] or pre-collected lookup tables [46, 84, 87] or analytical estimations [69, 154].

In ProxylessNAS [61] the authors have developed three latency prediction models for three different platforms: mobile phone (Google Pixel 1), GPU (NVIDIA V100), and CPU (Intel Xeon). To build their mobile latency predictors, they use the type of operators, input and output feature map sizes, and other architectural hyperparameters as features. The real values for Pixel 1 have been measured with Tensorflow-Lite as software. On ImageNet, their model achieves 3.1% better top-1 accuracy than MobileNetV2, while being 1.2x faster with measured GPU latency.

In NASCaps [69], the functional behavior of a given specialized CNN and CapsNet hardware accelerator is modeled at a high level, to quickly estimate the memory usage, energy consumption, and latency. The HW platform, the ASIC CapsAcc in the paper, is described at the RTL level. Using a VLSI CAD tool and the RTL specifications, energy, memory, and latency costs of elementary operations are measured. Elementary operations' cost corresponds for example to the number of cycles and energy required to execute a layer. These values are then multiplied by the number of occurrences of each operation in the architecture and accumulated to obtain the total cost for the latency and energy.

Although these techniques are efficient, they require hardware experts to build the models. For the lookup table method, for example, the researcher needs to dedicate a lot of time to optimizing the code of each operator/architecture in the targeted hardware, which requires compilation knowledge. Similarly, to build the best model predictor, the researcher needs expert knowledge to select the best features and verify the results. Therefore, these methods impose a barrier to non-hardware experts.

In order to fairly compare the accuracy of each method, we conducted experiments to compute the latency of each architecture in NAS-Bench-101 [181]. The aim is to compare 4 methods: real-world measurements, lookup table, prediction model, and analytical estimation. For the lookup table, we calculated the latency of each operator used in the cell of the benchmark including Identity, Conv3x3BnRelu, Conv1x1BnRelu, MaxPool3x3, BottleneckConv3x3, BottleneckConv5x5, and MaxPool3x3Conv1x1.

When a cell is generated, we sum the latency of the constructing operators and the latency of the whole cell is obtained. For the prediction model, we used two different models: a simple MLP and XGBoost, both trained on the real-world measurements of a portion of the benchmark (training set). We choose these two methods because they are both used by popular HW-NAS in [61] and [46] respectively. Lastly, for the analytical estimation, we computed the number of MAC for the cell and multiply this value by the latency of one multiply-add tensor instruction.

This experiment was run on a Tesla K80 GPU, the prediction model MLP had to run for 50 epochs with early stopping. NAS-Bench-101 defines more than 400k cells, we have done our test on 165,580 cells. The search algorithm used to calculate the search time is an evolutionary algorithm based on the validation accuracy given by the benchmark and the latency measured by the different methods. Figure 2.10 presents the results of the accuracy values of different methods. In this figure, the performance of the estimation is represented by the Root Mean Square Error (RMSE) which corresponds to the deviation relative to real-world measurements. As expected, the analytical estimation does not produce good results compared to the prediction models or the lookup table method. The prediction models even with a simple XGBoost give the best results and accelerate the search more than 5 times

compared to the real-world measurements.

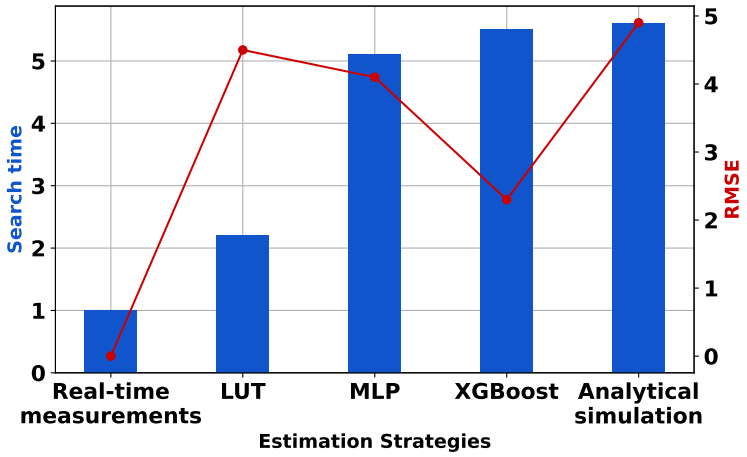

Figure 2.10: Comparison of hardware cost measurement methods. LUT stands for Look Up Table. The speedups are calculated w.r.t Real-world measurements

2.8 Other Considerations for Hardware-aware NAS

Prior to and in parallel with the hardware-aware NAS efforts, researchers have been working on reducing the memory footprint of DL models and execution time to facilitate the efficient deployment and design of hardware-friendly models. Two main methods have been used, namely handcrafting new operators that are more efficient such as separable convolutions [49], grouped convolutions [28] and applying DL optimizations such as quantization [182] and pruning [183]. This latter method is automated by several NAS works to compress the model and make it possible to execute on different hardware accelerators. Moreover, reducing the number of learnable parameters makes the training faster. For instance, SAL [184] reduces the number of parameters of ResNet-56 from 1.22M to 0.36M without significant degradation of the model's accuracy; 0.6% decrease.

In this section, we will define the two most used DL optimizations and review the NAS works that focus on searching for the right optimization parameters.

2.8.1 Automatic Mixed-Precision Quantization

In the DL compression field, *quantization* is one of the most important methods. Starting with BinaryConnect [161] in 2015 to binarize CNNs weights, it is now implemented in many DL software such as PyTorch or TensorFlow. The idea is that the weights of the DL model do not have to be represented in full 32-bit precision and can be represented in 8-bits precision, or even binary values in some cases, without significantly decreasing the model's accuracy. This idea was extended to the activations and weights [185]. Recently, mixed-precision quantization that applies different bit-width values for different layers in the same network has been proposed.

HAQ [186] implemented a dedicated reinforcement learning agent that learns to assign the right bit-width to each layer. Their goal is to specialize the architecture to a specific hardware platform by incorporating hardware constraints and accuracy into the reward function. For each layer, the agent takes two decisions: one for the weights and another one for the activations.

Single-Path [133] proposes an evolutionary algorithm that searches for the mixed-precision quantization policy. However, the search costs a huge amount of

time and processing as the space for mixed-precision is enormous.

BP-NAS [187] cast the mixed-precision quantization problem as a constrained optimization. In addition, it proposes a new constraint that encourages the model to focus on valid architectures while imposing a large punishment for the quantization outside the valid domain.

Although these works present interesting solutions to the mixed-precision problem, the huge search space and computational cost of the learning process still represent a real challenge. Furthermore, mixed-precision representation requires specific MAC architectures with scalable precision. This places a cap on the power efficiency, according to this study [188]. With this in mind, [189] proposes a uniform quantization search algorithm called neural channel expansion (NCE). NCE expands the number of channels of a layer when it is more sensitive to the quantization error while maintaining the same precision level.

2.8.2 Automatic Pruning

The second prominent compression method is *pruning*. Pruning methods eliminate some neurons or connections according to a defined criterion to reduce the number of parameters in the architecture. Generally, we evaluate the importance of the neurons, we prune the least important ones, and finally, we fine-tune the network.

AMC [116] proposes an automatic model compression that looks for the optimal sparsity for each layer during pruning. The authors trained a reinforcement learning agent to predict the best sparsity for given hardware. The reward function includes accuracy and FLOPs after pruning the architecture.

X. Dong and Y. Yang in [190] propose to prune the over-parameterized network without performance damage. They directly search within their NAS process for a network with flexible channel and layer sizes. **ABCPruner** [191] uses an artificial bee colony algorithm to efficiently find the optimal pruned structure. Another worth mentioning paper is **Partial Order Pruning** [192] which proposes a hardware-aware NAS that prunes the search space with a partial order assumption to look for the best speed and accuracy trade-off.

2.8.3 Security and Reliability Considerations in NAS

Other NAS methods try to address safety-critical issues by discovering architectures that are robust against adversarial attacks [193, 194]. RAS [193] formulates robustness as the sum of the accuracies on a bunch of adversarial samples. This robust evaluation makes it easier for the evolutionary algorithm to select better architecture in the population and apply different mutations (e.g., add a block, remove a block, add a connection...). Along with [195], they reveal these observations: first, the more dense the architecture is the more robust it is, second, under computational budget, adding convolution operations to direct connection edge is effective, and finally, flow of solution procedure (FSP) matrix is a good indicator of network robustness. Please note here, that none of the NAS methods that consider security and reliability in the search place is hardware-aware.

2.9 In-memory Computing & HW-NAS

One promising technology for edge hardware accelerators is analog-based in-memory computing, which is herein referred to as analog in-memory computing (IMC). IMC [196] can provide radical improvements in performance and power efficiency, by leveraging the physical properties of memory devices to perform computation and storage at the same physical location. Many types of memory devices, including Flash memory, Phase Change Memory (PCM), and resistive random-access memory (RRAM), can be used for IMC [197]. Most notably, analog IMC can be used

to perform Matrix-vector multiplication operations in $O(1)$ time complexity [198], which is the most dominant operation used for DNN acceleration. Besides, traditional von Neumann architecture, where the processor and memory are separate, leads to significant data movement, which is a bottleneck for AI applications. In-memory computing devices, on the other hand, allow for computation and storage to be integrated, reducing the need for data movement and enabling faster and more energy-efficient computations. Additionally, in-memory computing devices can support analog computation, which is better suited for neural network computations than the digital computation used in traditional processors. As AI applications continue to grow in complexity and scale, the importance of in-memory computing devices for accelerating AI computations will only continue to increase.

It is crucial to design DL architectures that are specifically optimized for such hardware. While in-memory computing devices can significantly improve the speed and efficiency of neural network computations, they have unique characteristics that need to be taken into consideration when designing DL architectures. In-memory Computing (IMC) devices have, for instance, limited precision and may not support all operations that are typically used in DL models. Another important constraint that needs to be considered when designing DL architectures for In-memory Computing (IMC) devices is the presence of noise in the devices. In-memory Computing (IMC) devices are susceptible to various types of noise, such as read noise and write noise, which can affect the accuracy of computations. DL architectures need to be resilient on top of being accurate and hardware efficient.

Many works [199, 200, 201, 202] target In-memory Computing (IMC) accelerators using HW-NAS. FLASH [199] uses a small search space inspired by DenseNet [203] and searches for the number of skip connections that efficiently satisfy the trade-off between accuracy, latency, energy consumption, and chip area. Its surrogate model uses linear regression and the number of skip connections to predict model accuracy. NAS4RRAM [201] uses HW-NAS to find an efficient DNN for a specific Resistive Random Access Memory (RRAM)-based accelerator. It uses an evolutionary algorithm, trains each sampled architecture without hardware-aware training, and evaluates each network on a specific hardware instance.

NACIM [200] uses co-exploration strategies to find the most efficient architecture and the associated hardware platform. For each sampled architecture, networks are trained considering noise variations. This approach is limited by using a small search space due to the high time complexity of training. UAE [202] uses a Monte-Carlo simulation-based experimental flow to measure the device uncertainty induced to a handful of DNN. Similar to NACIM [200], evaluation is performed using hardware-aware training with noise injection.

AnalogNet [204] extends the work of Micronet by converting their final models to analog-friendly models, replacing depthwise convolutions with standard convolutions, and tuning hyperparameters.

Our work, AnalogNAS [9] offers a better adaptation to analog IMC hardware compared to the aforementioned SOTA HW-NAS strategies. This is primarily due to two reasons. Firstly, our search space, which features resnet-like connections, is much larger and more representative. This expanded search space allows us to address the crucial question of identifying architectural characteristics suitable for analog IMC, which cannot be adequately explored with smaller search spaces. Secondly, in addition to the noise injection during HWA training utilized by existing approaches, we directly incorporate the inherent characteristics of analog IMC hardware into the objectives and constraints of our search strategy. Details of the methodology are given in Chapter 5.

2.10 Challenges and Limitations

In this section, we lay down the main challenges and barriers that prevent us to unfold hardware-aware NAS's full potential. Most of the challenges are also applicable to general NAS methods.

2.10.1 Benchmarking and Reproducibility

An important challenge while working on NAS and HW-NAS is the reproducibility of the methods, which allows comparisons and concrete improvements. Unfortunately, due to the use of different search spaces, various training methods, and the required significant computational resources, reproducibility is a difficult step. This difficulty is even higher when it comes to HW-NAS considering the numerous possibilities of targeted hardware devices. To address this issue, many works [148, 181, 205, 206] have proposed different benchmarks and data sets that allow NAS researchers to:

- Eliminate the cost of generating a search space by querying directly a tabular data set.
- Evaluate different search strategies on the same search space which allows a fair comparison between them.
- Provide data sets to be used by accuracy predictor models and hardware cost models.
- Open HW-NAS research to non-hardware experts by proposing datasets that contain the hardware-related metrics. These metrics are usually obtained after optimization of the operators and software that require specific hardware knowledge.

In this section, we will review each NAS benchmark and highlight its strengths and weaknesses. Table 2.3 summarizes the properties of the different benchmarks.

Table 2.3: Comparison of NAS Benchmarks

		NAS-Bench-101 [181]	NAS-Bench-201 [205]	NATS-Bench [207]	NAS-Bench-1shot1 [206]	NAS-Bench-NLP [148]	NAS-Bench-301[208] (DARTS)	HW-NAS-Bench [209]
Arch Search Space		Cell-based CNN	Cell-based CNN	Cell-based CNN	Super Network CNN	Cell-based LSTM	Super Network CNN	Cell-based CNN (NAS-Bench-201 + FBNet)
Size (number of architectures)		423k	15,625	15,625	363,648	14k	10^{18}	$46875 + 10^{21}$
Datasets	CIFAR-10	✓	✓	✓	✓		✓	✓
	CIFAR-100	✓	✓	✓	✓			✓
	ImageNet			✓	✓			✓
	PTB					✓		
	WikiText2					✓		
Metrics	Validation Accuracy	✓	✓	✓	✓	✓	✓	✓
	Training Time	✓	✓	✓	✓	✓	✓	✓
	Trained Parameters	✓	✓	✓		✓		✓
	FLOPs		✓	✓				✓
	Test Accuracy	✓	✓	✓	✓	✓	✓	✓
	Latency		✓	✓	✓		✓	✓
	Energy							✓
Predictor Models							✓	✓
Hw Platforms	GPU	GTX 1080Ti	GTX 1080Ti	Not Mentioned	Not mentioned	Tesla V100-SXM2	Not Mentioned	Edge GPU Jetson TX2
	Edge TPU							Edge TPU Dev Board
	Smartphone							Pixel 3
	Raspberry Pi							Raspi 4
	FPGA							Xilinx ZC706
	ASICs							ASIC-Eyeriss

NAS-Bench-101 [181][1] is a tabular dataset that maps 432k unique architectures to their relative training accuracy, validation accuracy, testing accuracy as well as training time and the number of trained parameters. Each architecture is trained for various numbers of epochs 4, 12, 36, 108. The architectures follow a fixed macro architecture, with searchable cells stacked (See the section 2.5.1). Each cell is constructed with up to 7 layers that can include 3 types of operations: 3x3 conv, 1x1 conv, and 3x3 max-pooling. NAS-Bench-101 allows flexibility by allowing different layers to be used in the stacked cells. The search space is constrained by limiting the number of edges to 9 and the number of nodes to 7 including the input and output nodes for each cell. However, not including operations such as separable convolution or dilated convolutions; which significantly improved the model's size and speedup [40], makes the resulting models parameter-heavy, which is not hardware-friendly in terms of memory footprint. Another downside of this benchmark is the inability to use a one-shot optimizer NAS, this is due to the tabular format. To enable the evaluation of weight-sharing NAS methods, two benchmarks have been released NAS-Bench-201 [205][2] and NAS-Bench-1Shot1 [206][3].

NAS-Bench-201 represents 15,625 architectures using a fixed cell-based macro architecture. Similar to NAS-Bench-101, it uses a predefined set of operations including conv 3x3, conv 1x1, Avg pooling, skip connection and no operation label. Each architecture is trained on three different datasets with different complexities namely, CIFAR-10, CIFAR-100, and imagenet-16. The authors extended this benchmark and presented it a year later NATS-Bench. **NATS-Bench** [207] increased the search

[1]Open sourced at https://github.com/google-research/nasbench
[2]Open sourced at https://github.com/D-X-Y/AutoDL-Projects
[3]Open sourced at https://github.com/automl/nasbench-1shot1

space size by varying the sizes of their architectures.

Similarly, **NAS-Bench-1Shot1** presented a new reformulation to reuse the even much more extensive computation of the NAS-Bench-101 dataset (120 TPU years) to create three new one-shot search spaces with growing complexity containing 6240, 29160, and 363648 architectures. In order to use the expensive experimentation done by NAS-Bench-101, the authors created a one-shot architecture that contains all the discrete cell architectures defined by NAS-Bench-101. This allows the search algorithm to only train the supernetwork and get the performance of each path from the NAS-Bench-101. Therefore, their benchmark construction does not need any additional cost.

Alternatively, **NAS-Bench-NLP** defines a tabular benchmark for NLP tasks. Their cell-based search space is constructed based on an LSTM Macro-architecture borrowed from AWD-LSTM [210]. Each cell can contain up to 24 nodes with 3 hidden states and 3 linear input vectors. With these constraints and a set of operators composed of the most used activations in recurrent cells, they are able to build LSTM, Gated Recurrent Unit (GRU) cells, and many more. The architectures are trained on Penn Tree Bank (PTB) dataset [211] and a sub-sample of the best-performing ones are also trained on WikiText-2 [212], which is a more realistic dataset for real-world NLP problems.

A major disadvantage of these benchmarks is the size and diversity of their search spaces. Indeed, as presented by the experimentation results we obtained in figure 2.11, on the small dataset NAS-Bench-201, a local search, which is the simplest optimization strategy, achieves state-of-the-art results without a significant search time compared to other search strategies, except NAS without training [213]. In this experiment, we compare the different results obtained by different NAS approaches on the NAS-Bench-201. Most of them use the metrics provided by the benchmark along with fine-tuning the architecture to obtain a more accurate validation metric, except NAS without training [213]. The NAS search without any further training achieves decent results within 17sec of the search. By dividing the benchmarks into N mini-batches, they increase their training efficacy. The higher this number is (N), the higher the over-fitting probability on the benchmark. Therefore, using small datasets with complex search algorithms does not yield any good results in terms of the accuracy of the final architecture or efficiency of the search.

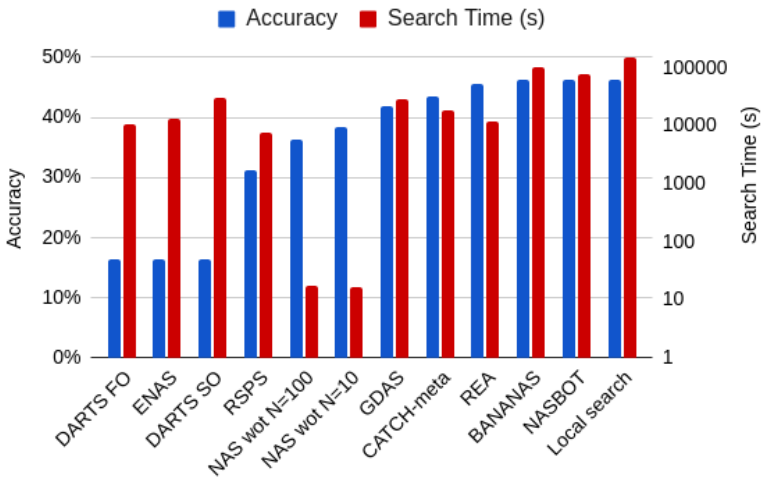

Figure 2.11: Results of different search algorithms on NAS-Bench-201.

NAS-Bench-301 is a much bigger benchmark that was designed to overcome the over-fitting problem on the architectures. It is based on DARTS [160] search space. Since the DARTS search space is far too large to be exhaustively evaluated by real-

world measurements, the authors built a surrogate model capable of predicting the various performance metrics. This model is trained on a subsample of the benchmark, 60k architectures whose latencies have been measured on CPUs.

A more recent paper introduced the first hardware-aware NAS benchmark, **HW-NAS-Bench** [209]. This work extends the number of hardware metrics and records the latency and the energy consumed on 7 hardware devices including commercial: NVIDIA Edge GPU Jetson TX2, NVIDIA Edge GPU Jetson Nano, Raspberry Pi 4, Edge TPU Dev Board, Pixel 3, ASIC-Eyeriss, and Xilinx ZC706 board. Its search space is a combination of FBNet [46] search space and NAS-Bench-201.

Although these datasets provide a good start to test different search strategies, they lack a lot of important operators that can significantly change the resulting architecture for hardware-aware NAS. In addition, we note a growing number of different applications of NAS in various tasks such as image restoration [214, 215, 216, 217], semantic segmentation [68, 75], and medical segmentation [131, 218]. Therefore, there is a growing need for proper benchmarks for each of the diverse tasks.

2.10.2 Transferability of the AI Models

Transferability in NAS is the ability to realize the search for the best architecture on a proxy dataset; that is usually smaller and simpler. The obtained architecture is then used on the targeted dataset. For example, we can use CIFAR-10 [156] during the search and then train the final model on ImageNet [219], which is made of 14 million images and 20,000 classes.

To enhance transferability, previous NAS works used cell-based search spaces. To transfer a model obtained from a cell-based search space, we just need to adjust the input sizes of the cells and deepen the network by adding more cells. However, stacking the same cell seems to be not efficient when incorporating hardware constraints. As MNASNet [62] argued, restricting cell diversity is critical for achieving high accuracy and low latency on mobile settings. Therefore, MNASNet uses a hierarchical search space that diversifies the cells in the architecture as well as the operators within that cell.

Many NAS works [159, 220] have included dedicated evaluations of the transferability of their final model. PNAS [220] proved that CIFAR-10 classification is highly correlated to ImageNet classification, hence transferring a cell or an architecture from CIFAR-10 to ImageNet would produce good results. XNAS [159] transferred their final cell structure on 6 popular classification benchmarks surpassing other conventional NAS methods while taking account of the hardware constraints.

Another interesting concept is presented by NAT [221]. They leverage NAS process to directly find transferable weights (i.e., get rid of the fine-tuning stage). The key idea behind NAT is that they start from a supernetwork and adaptively modify it to obtain a task-specific supernetwork. This latter approach can then be used directly to search for architectures within one task without the additional training cost. Under mobile scenarios, they demonstrated the efficacy of NAT on 11 benchmarks including ImageNet.

Overall, taking the transferability of the model into account remains a difficult challenge. Most solutions modify the search space to fit a certain dataset by leveraging a specific macro-architecture. Hardware awareness adds another level to this challenge as the architecture needs to be flexible to adapt to multiple platforms. Note that, transfer learning generalizes the ability to use a model from one dataset to another. In this section, we only talked about NAS that transfers their models from one dataset to another within the same task. Unfortunately, as NAS tasks evolve around image classification mainly, the whole search needs to be executed all over again if we want an architecture for another task.

2.10.3 Transferability of the HW-NAS Across Multiple Platforms

HW-NAS suffers from conditional optimality due to the variety of existing devices. Ideally, we should design different architectures for different platforms. However in real situations, given the prohibitive cost of the search and the cost of training on multiple architectures, we often resort to designing one architecture and deploying it anywhere. Transferring a model from one platform to another or being able to produce hardware transferable models via the NAS process is an interesting challenge for HW-NAS. The main difficulty lies in the variety and complexity of the existing platforms. For example, different platforms might perform well on different types of convolutions. In the following section, we discuss two approaches. The first transfers a single-target HW-NAS to another target by modifying the measurement values. The second takes the final architecture of the NAS process and transforms it to fit another platform. Each approach has its pros and cons as discussed below.

Transfer the entire NAS process: In this approach the whole NAS process needs to be re-executed to suit the new targeted hardware. In a single target HW-NAS, transferring the entire process to another platform is a complex task. Without mentioning the huge computational cost of retraining the entire process, the collection of the hardware constraints can be costly as well. When using real-world measurements, [116] ran the NAS search for three hardware platforms: CPU (Xeon E5-2640 v4), GPU (Tesla V100) and mobile phone (Google Pixel-1). However, using real measurements on the hardware platform considerably slows down the search algorithm and requires the availability of the targeted hardware during the search time. On the other hand, using an analytical estimation requires expert knowledge for the different targeted platforms. When using other collection methods such as the lookup table or the prediction model, we'll need to collect data from the new platforms by running again the entire set of operators. To this end, HW-NAS tries to create a general measurement method. For example, Once-for-all [42] created a lookup table with the reported inference latency on each tested hardware platform (i.e., Samsung S7 Edge, Note8, Note10, Google Pixel1, Pixel2, LG G8, NVIDIA 1080Ti, V100 GPUs, Jetson TX2, Intel Xeon CPU, Xilinx ZU9EG, and ZU3EGFPGAs). According to the used measurement method, transferring the NAS process to target another platform is increasingly difficult and not scalable.

Transfer the final model: An alternative approach is to find the best model for one hardware platform and then try to specialize it for another one. This solution is proposed in [42, 61, 159]. Usually, the specialization is done by compressing the model using quantization which enables the model to fit in tiny devices. However, the specialization is challenging because of the following reasons:

- **An operator may be efficient in one platform and less efficient in another:** In [66], the authors argued that separable convolutions give great results when ran on GPUs but perform badly on CPUs. Additionally, it is common that deeper networks perform well on CPUs while wider ones perform well on GPUs because of the possible parallelization. Table 2.4 demonstrates the comparison between the execution times of different operators on an Intel i7 CPU and NVIDIA TX2 GPU. The results reinforce our assumption that different operators' efficiency varies from one platform to another. Therefore, the best model is highly correlated to the platform we choose which makes designing a HW-NAS that targets multiple platforms very challenging.
- **Limits of the compression methods:** We consider here the quantization and pruning. For these two methods, we know that theoretically, the compression ratio has a threshold that cannot be surpassed. For example, quantizing a

model implies encoding its activations and weights into the minimum possible bit length. Theoretically, this length is 2, with 1 bit used to encode the values. Even without considering the trade-off between accuracy and model size, these methods have limits. This is why we need to start from a model that is already optimized to be able to deploy the model on tiny devices. For this end, [116] is composed of three components. The first component is a multi-objective NAS that searches for the best model in terms of accuracy and latency. Given the resultant model of the first component, the second component searches for pruning possibilities that would preserve the accuracy and decrease the model size. Finally, the last component takes care of quantizing the model with mixed precision. In [222] the authors propose to start from standard architectures such as VGG, ResNet, and GoogleNet and cast the quantization as a neural architecture search problem. This work achieves a minimal loss of accuracy with appreciable memory savings. In addition to the limit for the model size, we also have a limit for the accuracy. The compressed model typically does not have better accuracy than the pre-trained model we started from.

Table 2.4: Comparison between different operators on Intel i7 CPU and NVIDIA TX2 GPU. The convolution operators were used to create a CNN model that was trained for Image Classification on ImageNet. The RNN cells were trained on Text Classification on IMDB dataset [19]. *Results were obtained using PyTorch with a number of samples of 1000.*

Operator	Avg Latency on CPU (ms)	Avg Latency on GPU (ms)	Accuracy
Conv2d	1.33	0.85	0.67
Separable Depthwise Convolution	2.05	0.54	0.64
Dilated convolution	1.36	0.835	0.56
Grouped Convolution	2.27	1.94	0.62
LSTM cell	14.93	2.45	0.57
GRU cell	9.32	2.53	0.65

2.11 Conclusion

In this chapter, we have provided a comprehensive survey and systematic analysis of state-of-the-art Hardware-aware Neural Architectural Search. We reviewed several multi-objective strategies that aim to find the optimal architecture with the highest accuracy while reducing energy, memory, and computational costs. We proposed a HW-NAS taxonomy and categorize existing approaches along four key dimensions: the search space, the search strategy, the acceleration technique, and the hardware cost estimation strategy. We also discussed other considerations in Hardware-aware NAS that include optimizations such as quantization and pruning. Finally, we presented a discussion on future directions that would benefit existing and future HW-NAS researchers. The next chapters will highlight our contributions to over the challenges found in HW-NAS.

Part II

Efficient HW-NAS methods

Chapter 3

Multi-objective Surrogate Model for HW-NAS

Contents

3.1 Context

Hardware-aware NAS (HW-NAS) methods have recently exhibited remarkable success in enhancing the efficiency and performance of neural networks across various tasks. Nevertheless, the search process involved in HW-NAS can be excessively resource-intensive, often spanning multiple days and consuming a substantial amount of energy. This significant energy consumption results in a considerable carbon footprint, surpassing even the annual emissions of a car [223]. Consequently, while HW-NAS presents promising outcomes, its resource-intensive nature raises significant environmental concerns.

One particular aspect that contributes to the inefficiency of HW-NAS is the evaluation methods employed. These methods, used to assess the performance and efficiency of candidate network architectures, often require extensive computational resources and time-consuming training. Recognizing this bottleneck, there is a growing interest in the development of surrogate models within the research community. Surrogate models offer an alternative approach by constructing approximations of the evaluation methods, enabling faster and more efficient exploration of the architecture space. However, using multiple surrogate models for each objective (e.g., accuracy and latency) is suboptimal.

To address the challenge and research question 1, we propose a novel methodology based on surrogate models specifically designed for multi-objective problems. By leveraging our methods, ML engineers can effectively explore a wide range of architectures within a limited timeframe, thereby minimizing the environmental impact, achieving heightened architectural efficiency, and ultimately reducing the time-to-market.

In this chapter, we introduce two valuable contributions: Hardware-aware Pareto Rank Neural Architecture Search (*HW-PR-NAS*) in section 3.2, and Pareto Rank Preserving Supernetwork Training (*PRP-NAS*) in section 3.3. These contributions are specifically designed to cater to different search space definitions, namely cell-based and supernetwork architectures, respectively. They allow an average 3.1x speedup on several benchmarks and a significant energy consumption drop of 30%.

Research Question 1

How to efficiently evaluate the performance and hardware efficiency of an architecture without fully training it in a multi-objective search strategy?

3.2 HW-PR-NAS: Multi-Objective HW-NAS with Pareto Rank-preserving Surrogate Models

HW-NAS techniques are comprised of three components: **the search space, the search algorithm, and the evaluation method** (figure 3.1.A) HW-NAS is formulated as a multi-objective optimization problem. Several HW-NAS approaches aim to find the best architectures with two or more conflicting objectives: e.g., maximizing the accuracy of an architecture while minimizing its inference latency. In multi-objective optimization, the results obtained from the search algorithm are often not a single solution but a set of solutions. These solutions are called *dominant solutions* because they dominate all other solutions in terms of the trade-offs between the targeted objectives. In the case of HW-NAS, the optimization result is a set of architectures that have the best objectives' trade-off (figure 3.1.B). Formally, the set of best solutions is represented by a *Pareto front*.

NAS algorithms involve training multiple DL architectures to adjust the exploration of a huge search space. This requires many hours/days of data center-scale computational resources. This time complexity is exacerbated in the case of HW-NAS

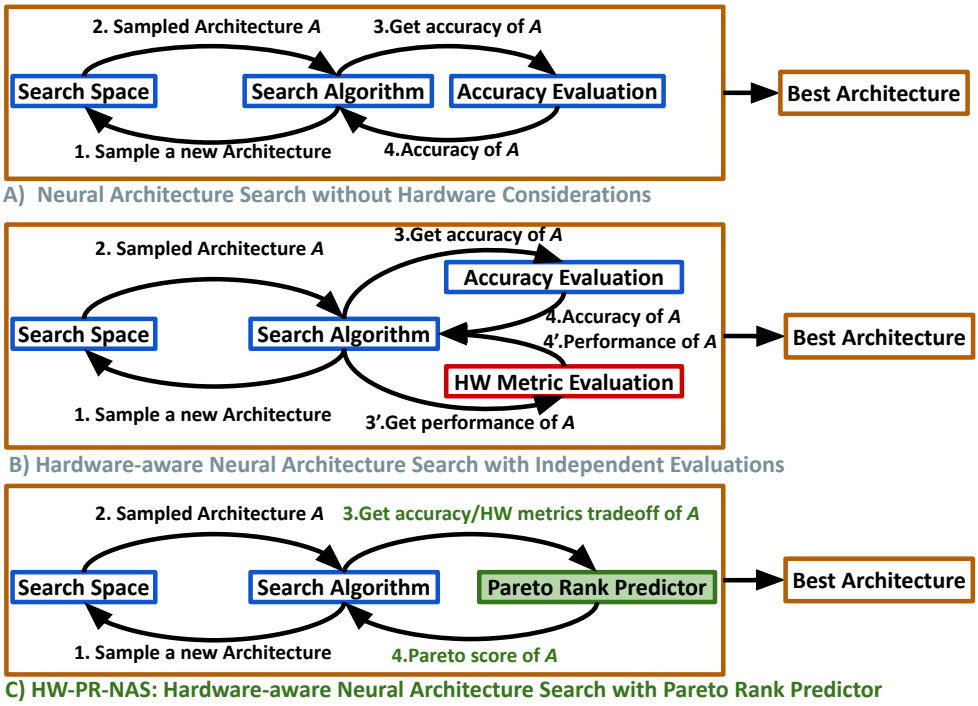

Figure 3.1: Simplified illustration of the use of HW-PR-NAS in a NAS process. *HW Perf* means the Hardware performance of the architecture such as latency, power, etc.

multi-objective assessments, as additional evaluations are needed for each objective or hardware constraint on the target platform. To address this problem, researchers have proposed surrogate-assisted evaluation methods [16, 17]. Surrogate models use analytical or ML-based algorithms to quickly estimate the performance of a sampled architecture without training it. Existing HW-NAS approaches [224] rely on the use of different surrogate-assisted evaluations, whereby each objective is assigned an independently trained surrogate model (figure 3.1.B). However, this introduces false dominant solutions as each surrogate model brings its share of approximation error and could lead to search inefficiencies and falling into local optimum (figures 3.2.A and 3.2.B).

Learning-to-rank theory [16, 225] has been used to improve the surrogate model evaluation performance. This was motivated by the observation that throughout the NAS process, it is more important to correctly rank a sampled architecture relative to other architectures than to compute its exact accuracy. Rank-preserving surrogate models significantly reduce the time complexity of NAS while enhancing the exploration path. However, in the multi-objective context, training each model independently cannot preserve the *Pareto rank* of the architectures (see figure 3.2).

We propose **HW-PR-NAS** [5, 226], **Hardware-aware Pareto-ranking NAS**, a new unified surrogate model trained to address multiple objectives in HW-NAS.

HW-PR-NAS contributions are summarized as follows:

1. We introduce a **flexible and general architecture representation** which allows adapting the surrogate model to include new architecture objective types without incurring additional training costs.

2. We introduce **a novel training methodology for multi-objective HW-NAS surrogate models**. Our surrogate model is trained using a novel ranking loss technique. The goal is to rank the architectures from dominant to non-

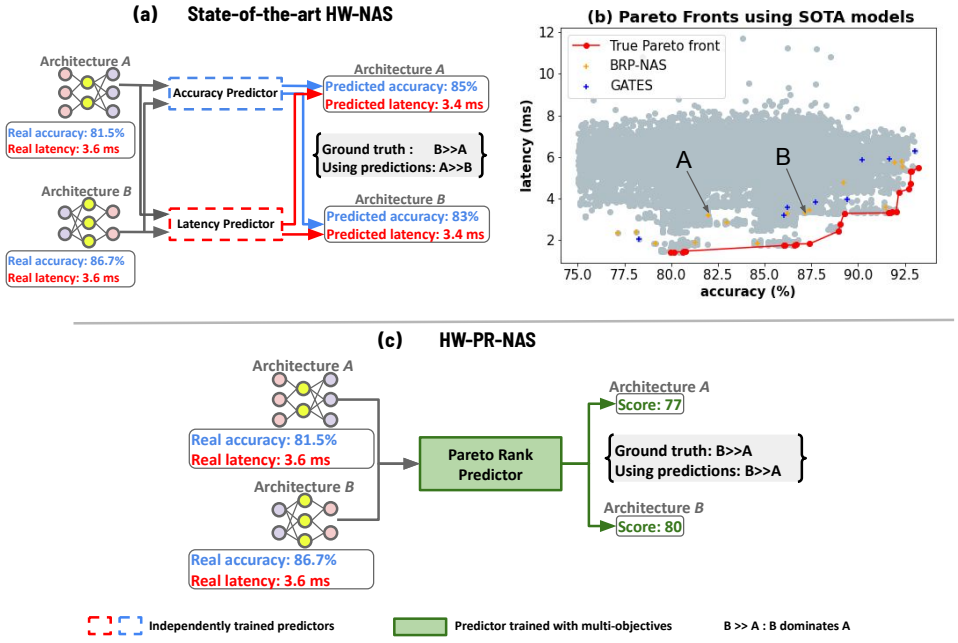

Figure 3.2: This figure illustrates the limitation of state-of-the-art surrogate models alleviated by HW-PR-NAS. a) and b) illustrate how two independently trained predictors exacerbate the dominance error and the results obtained using GATES [16] and BRP-NAS [17]. c) illustrates how we solve this issue by building a single surrogate model.

dominant ones by assigning high scores to the dominant ones. We call these scores *Pareto score*.

Our approach has been evaluated on seven edge hardware platforms, including ASICs, FPGAs, GPUs, and multi-cores. Experimental results demonstrate up to 2.5x speed up while guaranteeing that the search ends near the true Pareto front. To compare our approach with state-of-the-art, we use the normalized hypervolume metric. This metric evaluates the coverage and diversity of a Pareto front against the true Pareto front. Preliminary results show that using HW-PR-NAS is more efficient than using several independent surrogate models to reduce the search time and improve the quality of the Pareto approximation.

3.2.1 Proposed Approach

Figure 3.3 shows an overview of HW-PR-NAS, which is composed of two main components: ❶ **Encoding Scheme** and ❷ **Pareto Rank Predictor**.

Each architecture is encoded into a unique vector and then passed to the Pareto Rank Predictor in the **Encoding Scheme**. The **Pareto Rank Predictor** uses the encoded architecture to predict its Pareto Score (see equation 3.6) and adjusts the prediction based on the Pareto Ranking Loss. The Pareto Score, a value between $0 and 1$, is the output of our predictor. We use a listwise Pareto ranking loss to force the Pareto Score to be correlated with the Pareto ranks.

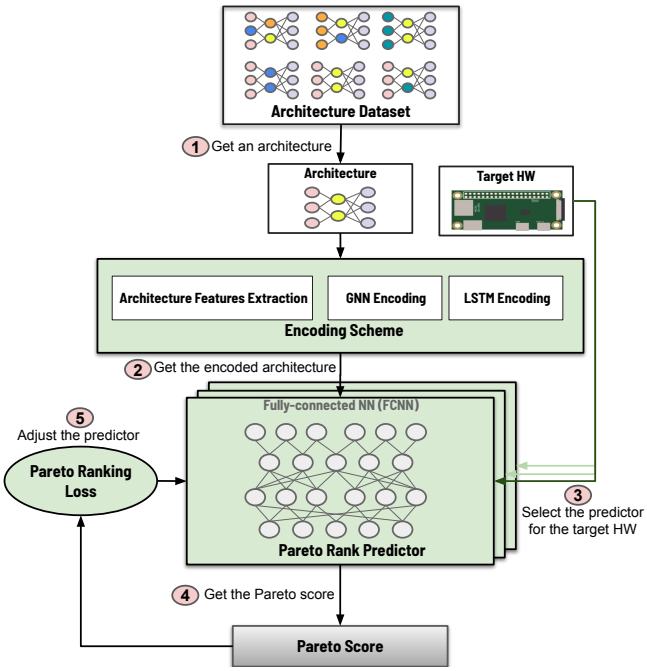

Figure 3.3: General Overview of HW-PR-NAS

Definitions

The following terms are used with their corresponding definitions:

- *Representation:* is the format in which the architecture is stored.
- *Encoding:* is the process of turning the architecture representation into a numerical vector. The surrogate model can then use this vector to predict its rank.
- *Encoding scheme:* is the methodology used to encode an architecture.
- *Encoder:* is a function that takes as input architecture and returns a vector of numbers, i.e., applies the encoding process.
- *Pareto Rank Predictor:* is the last part of the model architecture specialized in predicting the final score of the sampled architecture (see figure 3.3).

Encoding Scheme

To achieve a robust encoding capable of representing most of the key architectural features, HW-PR-NAS combines several encoding schemes (see figure 3.3). Each architecture is described using two different representations: a *Graph Representation*, which uses Directed Acyclic Graphs (DAG), and a *String Representation*, which uses discrete tokens that express the NN layers. For example, using "conv_3x3" to express a 3x3 convolution operation. We use two encoders to represent each architecture accurately. Both representations allow the use of different encoding schemes. Each encoder can be represented as a function E formulated as follows:

$$E : A \rightarrow \xi \tag{3.1}$$

A denotes the search space, and ξ denotes the set of encoding vectors. The encoder E takes an architecture's representation as input and maps it into a continuous space

Table 3.1: Hyperparameters associated with GCN and LSTM encodings and the decoder used to train them.

	Hyperparameter	**Value**
GCN Encoding	Number of layers	2
	hidden depth	128
	hidden dimension	1
	FC dimension	32
LSTM Encoding	Number of layers	2
	Hidden units	[32, 64]
	FC dimension	32
	recurrent dropout	0.2
Decoder	Number of layers	3
	Hidden units	[32,32]

ξ. The encoding result is the input of the predictor.

In our approach, three encoding schemes have been selected depending on their representation capabilities and the literature review [16, 17] (see table 3.3):

1. **Architecture Features Extraction.** From each architecture, we extract several Architecture Features (AF): number of FLOPs, number of parameters, number of convolutions, input size, architecture's depth, first and last channel size, and number of down-sampling.

2. **GCN Encoding.** To efficiently encode the connections between the architecture's operations, we apply a GCN encoding. Each architecture is encoded into its adjacency matrix and operation vector. It is then passed to a GCN [227] to generate the encoding. The output is passed to a dense layer to reduce its dimensionality.

3. **LSTM Encoding.** To represent the sequential behavior of the architecture, we use an LSTM encoding scheme. We pass the architecture's string representation through an embedding layer and an LSTM model. We then reduce the dimensionality of the last vector by passing it to a dense layer.

The resulting encoding is a vector that concatenates the AFs to ensure that each architecture in the search space has a unique and general representation that can handle different tasks [228] and objectives. The hyperparameters describing the implementation used for the GCN and LSTM encodings are listed in table 3.1.

Using a decoder module, the encoder is trained independently from the Pareto rank predictor. The decoder takes the concatenated version of the three encoding schemes and recreates the representation of the architecture. We set the decoder's architecture to be a 4-layer LSTM. In addition, we leverage the attention mechanism to make decoding easier. The encoder-decoder model is trained with the cross entropy loss. Equation 3.2 formulates the cross-entropy loss, denoted as L_{ED}, where $output\_size$ changes according to the string representation of the architecture, y and $\hat{y}$ correspond to the predicted operation and the true operation respectively. This training methodology allows the architecture encoding to be hardware-agnostic.

$$L_{ED} = -\sum_{i=1}^{output\_size} y_i * log(\hat{y}_i) \tag{3.2}$$

The preliminary analysis results in figure 3.4 validate the premise that different encodings are suitable for different predictions in the case of NAS objectives. Figure 3.4 shows the results obtained after training the accuracy and latency predictors with different encoding schemes. Each predictor is trained independently. Using Kendal Tau [229], we measure the similarity of the architectures' rankings between

the ground truth and the tested predictors. Accuracy predictors are sensible to the types of operators and connections in a DL architecture. When using only the AF, we observe a small correlation (0.61) between the selected features and the accuracy, resulting in poor performance predictions. The best predictor is obtained using a combination of GCN encodings, which encodes the connections, node operation, and AF. For latency prediction, results show that the LSTM encoding is better suited. An intuitive reason is that the sequential nature of the operations to compute the latency is better represented in a sequence string format. The last two columns of the figure show the results of the concatenation, which outperforms other representations as it holds all the features required to predict the different objectives.

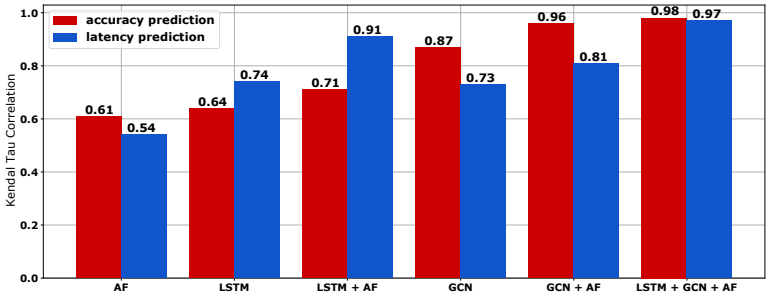

Figure 3.4: Results of different encoding schemes for accuracy and latency predictions on NAS-Bench-201 and FBNet. AF refers to Architecture Features. LSTM refers to Long Short-Term Memory neural network. GCN refers to Graph Convolutional Networks.

These results were obtained with a fixed Pareto Rank predictor architecture. We used a fully-connected neural network (FCNN). Table 3.2 shows the results of modifying the final predictor on the latency and accuracy predictions. While we achieve a slightly better correlation using XGBoost on the accuracy, we prefer to use a 3-layer FCNN for both objectives to ease the generalization and flexibility to multiple hardware platforms.

Table 3.2: Results of different regressors on NAS-Bench-201. KT Corr stands for Kendal Tau Correlation.

	Accuracy		Latency	
	RMSE	KT Corr	RMSE	KT Corr
3-layer FCNN	4.88	0.924	3.238	**0.8817**
XGBoost [230]	3.12	**0.931**	3.216	0.8742
LGBoost [231]	3.58	0.864	3.058	0.8247

Pareto Ranking Predictor

HW-PR-NAS is trained to predict the Pareto front ranks of architecture for multiple objectives simultaneously on different hardware platforms. The predictor uses three fully-connected layers. Due to the hardware diversity illustrated in table 3.4, the predictor is trained on each HW platform. Prior works [224] demonstrated that the best architecture in one platform is not necessarily the best in another. Therefore, the Pareto fronts differ from one HW platform to another.

HW-PR-NAS predictor architecture is the same across the different HW platforms. The only difference is the weights used in the fully-connected layers. The HW platform identifier (*Target HW* in figure 3.3) is used as an index to point to the corresponding predictor's weights.

To train this Pareto ranking predictor, we define a novel *listwise* loss function to predict the Pareto ranks.

Pareto ranks definition In a multi-objective NAS problem, the solution is a set of N architectures $S = s_1, s_2, ..., s_N$. These architectures may be sorted by their Pareto front rank K. The true Pareto front is denoted as F_1 where the rank of each architecture within this front is 1. An architecture is in the true Pareto front if and only if it dominates all other architectures in the search space. According to this definition, we can define the Pareto front ranked 2, F_2, as the set of all architectures that dominate all other architectures in the space except the ones in F_1. Formally, the rank K is the number of Pareto fronts we can have by successively solving the problem for $S - \bigcup_{s_i \in F_k \wedge k<K}$, i.e., the top dominant architectures are removed from the search space each time.

Theoretically, the sorting is done by following these conditions:

$$\forall s_i, s_j \in F_k,\ s_i \not\succ s_j \wedge\ s_j \not\succ s_i \tag{3.3}$$

$$\forall s_i \in F_{k+1}\ \forall s_j \in F_k,\ s_i \not\succ s_j \tag{3.4}$$

$$\forall s_i \in F_{k+1}\ \exists s_j \in F_k,\ s_j \succ s_i \tag{3.5}$$

Equation 3.3 formulates that for all the architectures with the same Pareto rank, no one dominates another. Equation 3.4 formulates that any architecture with a Pareto rank $k+1$ cannot dominate any architecture with a Pareto rank k. Equation 3.5 formulates that for each architecture with a Pareto rank $k+1$, at least one architecture with a Pareto rank k dominates it.

Pareto ranking loss definition Our predictor takes an architecture as input and outputs a score. This score is adjusted according to the Pareto rank. The loss function aims to keep the predictor's outputs; scores $f(a)$, where a is the input architecture, correlated to the actual Pareto rank of the given architecture.

The scores are then passed to a softmax function to get the probability of ranking architecture a. The final output is formulated as follows:

$$out(a) = \frac{\exp f(a)}{\sum_{a \in B} \exp f(a)} \tag{3.6}$$

In this equation, B denotes the set of architectures within the batch, while $|B|$ denotes its size. We then design a listwise ranking loss by computing the sum of the negative likelihood values of each batch's output:

$$L(B) = \sum_{i=1}^{|B|} \{-out(a^{(i),B}) + log \sum_{j=i}^{|B|} exp(out(a^{(j),B})\} \tag{3.7}$$

$a^{(i),B}$ denotes i-th Pareto-ranked architecture in subset B. This loss function computes the probability of a given permutation to be the best, i.e., if the batch contains three architectures a_1, a_2, a_3 ranked (1, 2, 3) respectively. The loss function encourages the surrogate model to give higher values to architecture a_1 then a_2 and finally a_3. We compute the negative likelihood of each architecture in the batch being correctly ranked.

Training procedure To train the HW-PR-NAS predictor with two objectives, the accuracy and latency of a model, we apply the following steps:

1. We build a ground truth dataset of architectures and their Pareto ranks. We randomly extract architectures from NAS-Bench-201 and FBNet using Latin Hypercube Sampling [232]. The batches are shuffled after each epoch. The two benchmarks already give the accuracy and latency results. Thus the dataset creation is not computationally expensive. However, if one uses a new search space, the dataset creation will require at least the training time of 500 architectures.

2. We iteratively compute the ground truth of the different Pareto ranks between the architectures within each batch using the actual accuracy and latency values. Two architectures with a close Pareto score means that both have the same rank.

3. We calculate the loss between the predicted scores and the ground truth computed ranks.

4. Using this loss function, the scores of the architectures within the same Pareto front will be close to each other, which helps us extract the final Pareto approximation.

Algorithm 1 Training methodology of the rank predictor component.

Input: benchmark: Benchmark (α: architecture, acc: accuracy, l: latency)
Output: Trained Surrogate Model
$D \leftarrow Sample(benchmark, dataset\_size)$
$model \leftarrow FCNN$
initialize model
for epoch ¡ max_epochs **do**
  batches = generate_random_batches(D)
  for B in batches **do**
    $R \leftarrow Compute\_Pareto\_Ranks(B)$
    $P \leftarrow model(B)$
    $loss \leftarrow NLL(R, P)$
    backpropagate $loss$ and adjust the weights of the model
  end for
end for

The most important hyperparameter of this training methodology that needs to be tuned is the batch_size. Figure 3.5 shows the empirical experiment done to select the *batch_size*. We set the batch size to 18 as it is, empirically, the best trade-off between training time and accuracy of the surrogate model. The accuracy of the surrogate model is represented by the Kendal tau correlation between the predicted scores and the correct Pareto ranks. This value can vary from one dataset to another. The hyperparameter tuning of the batch_size takes ∼1h for a full sweep of 6 values in this range: [8, 12, 16, 18, 20, 24].

3.2.2 Evaluation Methodology

We compare HW-PR-NAS to existing surrogate model approaches used within the HW-NAS process. The searched final architectures are compared with state-of-the-art baselines in the literature. We target two objectives: accuracy and latency. Our goal is to evaluate: ❶ the quality of the NAS results by using the **normalized hypervolume**, and ❷ the speed-up of HW-PR-NAS methodology by measuring the **search time** of the end-to-end NAS process.

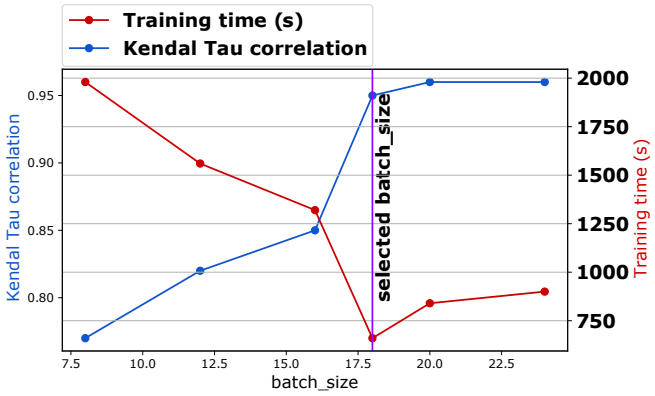

Figure 3.5: Performance of the Pareto rank predictor using different batch_size values during training.

- **Hypervolume** This metric calculates the area from the Pareto front approximation to a reference point. We use the furthest point from the Pareto front as a reference point.
- **Search time** This metric corresponds to the time spent by the end-to-end NAS process, including the time spent training the surrogate models.
- **Search Spaces** Our experiments are initially done on NAS-Bench-201 [233] and FBNet [234] for CIFAR-10 and CIFAR-100. Imagenet-16-120 is only considered in NAS-Bench-201. To validate our results on ImageNet, we run our experiments on ProxylessNAS Search Space [235]. The search space contains 6^{19} architectures, each with up to 19 layers. This is to be on par with various state-of-the-art methods.
- **Search Algorithms** The search algorithms call the surrogate models to get an estimation of the objectives. In our comparison, we use Random Search (RS) and Multi-Objective Evolutionary Algorithm (MOEA). In RS, the architectures are selected randomly, while in MOEA, a tournament parent selection is used. For MOEA, the population size, maximum generations and mutation rate have been set to 150, 250, and 0.9, respectively. The stopping criteria are defined as a maximum generation of 250 and a time budget of 24 hours.

Baselines We compare HW-PR-NAS to the state-of-the-art surrogate models presented in Table 3.3.

Table 3.3: State-of-the-art surrogate models used for HW-NAS. AF stands for architecture features such as the number of convolutions and depth.

Surrogate Model	**Objective**	**Encoding**	**Loss**	**Dataset Size**	**Ranking**
GATES [16]	Accuracy	GCN	Hinge Pairwise	7318	yes
BRP-NAS [17]	Accuracy Latency	GCN GCN	MSE KL Div	900	no
ProxylessNAS [235]	Latency	AF	RMSE	5000	no
LRLC [236]	Accuracy	LSTM	Logistic Loss	1000	yes

Training Implementation HW-PR-NAS training dataset consists of 500 architectures and their respective accuracy and hardware metrics on CIFAR-10, CIFAR-100 and ImageNet-16-120 [237]. These architectures are sampled from both NAS-Bench-201 [233] and FBNet [234] using HW-NAS-Bench [238] to get the hardware metrics on various devices. We used 100 models for validation. Training the surrogate model took 1.5 GPU hours with 10-fold cross-validation. While this training methodology may seem expensive compared to state-of-the-art surrogate models presented in table 3.3, the encoding networks are much smaller, with only two layers for the GNN and LSTM. The comprehensive training of HW-PR-NAS requires 43min on NVIDIA RTX 6000 GPU, which is done only once before the search. Note that if we want to consider a new hardware platform, only the predictor (i.e., three fully connected layers) is trained, which takes up to less than 10min.

Experimental Setup Our surrogate models and HW-PR-NAS process have been trained on NVIDIA RTX 6000 GPU with 24GB memory. To evaluate HW-PR-NAS on edge platforms, we have used the platforms presented in table 3.4. Our implementation is coded using PyMoo [1] for the multi-objective search algorithms and PyTorch for DL architectures.

Table 3.4: Illustrative comparison of edge hardware platforms targeted in this work.

NVIDIA Edge GPU			
Platform	**Computation**	**Memory**	**Power**
Jetson TX2	256-core NVIDIA Pascal	4GB	30W
ASICs			
Platform	**Computation**	**Memory**	**Power**
TPU Board	2 TPU Cores	1GB	12W
Eyeriss	-	eDRAM	<1W
FPGA			
Platform	**Computation**	**Memory**	**Power**
Xilinx Zynq-7000	ZC706	1GB	8-20W
Zynq UltraScale+	ZCU102	4GB	15-40W
Mobile Phones			
Platform	**Computation**	**Memory**	**Power**
Google Pixel 3	8 cores	4GB	8W
Single-board computer			
Platform	**Computation**	**Memory**	**Power**
Raspberry Pi	4-core Cortex-A72 ARM	3GB	12W

3.2.3 End-to-End Results

We compare the different Pareto front approximations to the existing methods to gauge the efficiency and quality of HW-PR-NAS.

Figure 3.6 presents the different Pareto front approximations using HW-PR-NAS, BRP-NAS [17], GATES [16], proxylessnas [235] and LCLR [236].

We averaged the results over five runs to ensure reproducibility and fair comparison. The title of each subgraph is the normalized hypervolume. We notice that our approach consistently obtains better Pareto front approximation on different platforms and different datasets.

Figure 3.7 summarizes the obtained hypervolume of the final Pareto front approximation for each method. The hypervolume indicator encodes the favourite Pareto front approximation by measuring objective function values' coverage. The larger the hypervolume, the better the Pareto front approximation and, thus, the better the corresponding architectures. The results vary significantly across runs when using

[1] Multi-objective optimization in Python. https://pymoo.org/

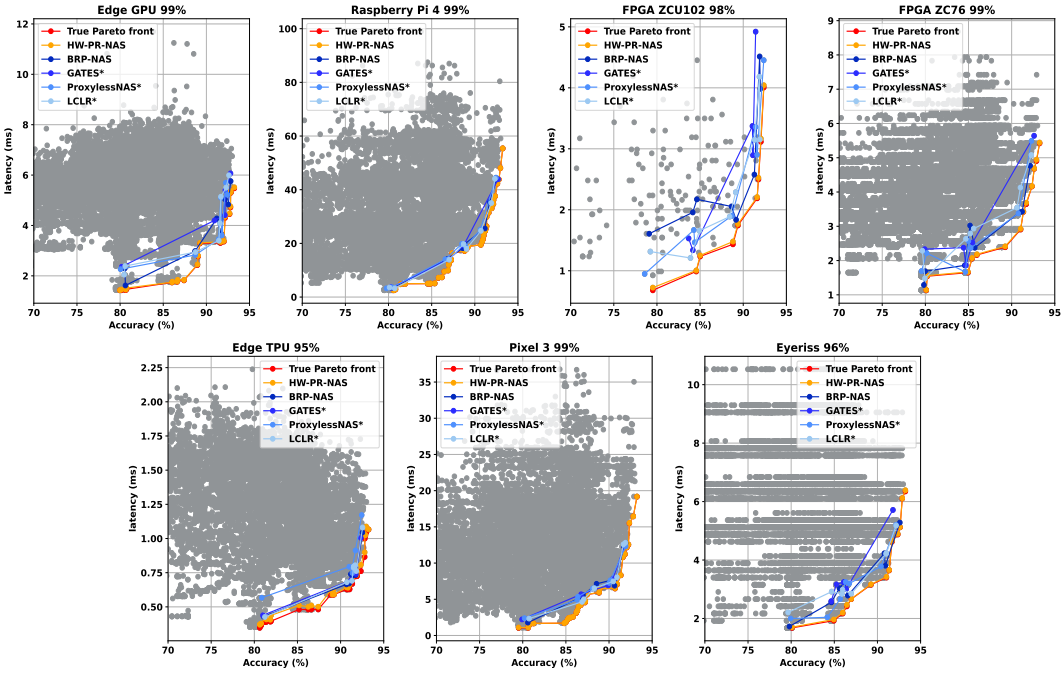

Figure 3.6: Pareto front approximations on CIFAR-10 on edge hardware platforms. We show the true accuracies and latencies of the different architectures and the normalized hypervolume on each target platform.

two different surrogate models. However, using HW-PR-NAS, we can have a decent standard error across runs.

In figure 3.8, we also compare the speed of the search algorithms. HW-PR-NAS achieves a 2.5x speed-up in the search algorithm.

This is due to:

- Using one common surrogate model instead of invoking multiple ones.
- Decreasing the number of comparisons to find the dominant points.
- Requiring a smaller number of operations than GATES and BRP-NAS.

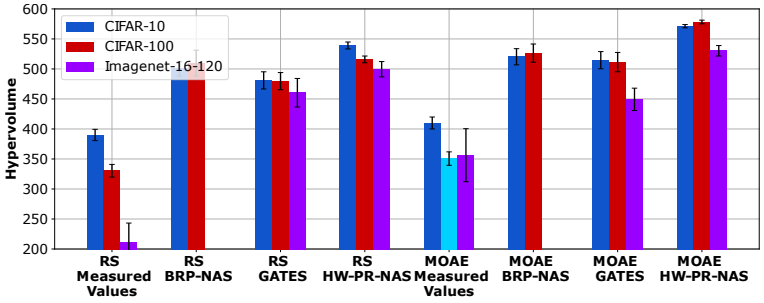

Figure 3.7: Final Hypervolume obtained by each method on the three datasets. We show the means ± standard errors based on 5 independent runs.

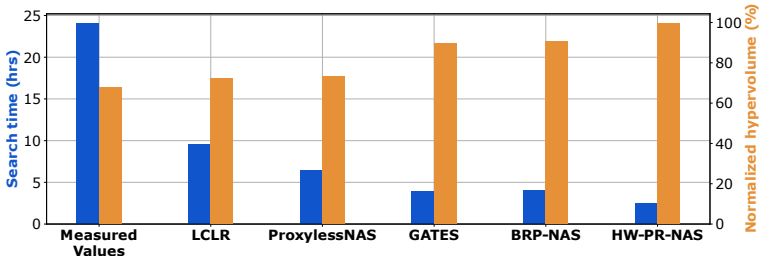

Figure 3.8: Search time of MOAE using different surrogate models on 250 generations with a max time budget of 24 hours.

Table 3.5: Comparison of Optimal Architectures obtained in the Pareto Front for CIFAR-10.

Architecture	Jetson TX2		TPU Board		Google Pixel 3		FPGA ZCU102		Eyeriss		HW Aware
	Acc (%)	Lat (ms)	Acc (%)	Lat (ms)	Acc (%)	Lat (ms)	Acc (%)	Lat (ms)	Acc (%)	Lat (ms)	
GATES [16]	92.83	6.3	91.82	1.37	90.21	23.6	90.73	3.8	93.49	9.27	Yes
BRP-NAS [17]	91.3	5.86	92.01	1.5	90.34	23.1	82.31	3.72	89.6	8.92	Yes
ProxylessNAS [235]	91.86	4.86	93.68	2.1	93.51	21.36	92.68	5.6	93.87	7.56	Yes
HAGCNN [239]	92.57	8.53	92.57	5.43	92.57	24.65	92.57	6.87	92.57	9.54	No
Shapley-NAS [240]	94.37	9.76	**94.37**	4.8	94.37	25.21	94.37	8.62	94.37	9.76	No
AG-Net [241]	94.37	9.76	94.37	4.8	94.37	25.21	94.37	8.62	94.37	9.76	No
β-DARTS [242]	91.55	6.42	91.55	3.53	91.55	25.3	91.55	7.51	91.55	8.46	No
HW-PR-NAS	**95.75**	**4.37**	94.15	**1.12**	**94.43**	**20.5**	**95.2**	**4.0**	**94.72**	**7.91**	Yes

3.2.4 Final Pareto Front Analysis

We analyze the proportion of each benchmark on the final Pareto front for different edge hardware platforms. In Pixel3 (mobile phone), 80% of the architectures come from FBNet. Indeed, this benchmark uses depthwise convolutions, accelerating DL architectures on mobile settings. The depthwise convolution decreases the model's size and achieves faster and more accurate predictions. However, depthwise convolutions do not benefit from the GPU, TPU, and FPGA acceleration compared to standard convolutions used in NAS-Bench-201, which have a higher proportion in the Pareto front of these platforms 54%, 61%, and 58% respectively.

Table 3.5 shows the difference between the final architectures obtained. The best values (in bold) show that HW-PR-NAS outperforms HW-NAS approaches on almost all edge platforms. The depthwise convolution (DW) available in FBNet is suitable for architectures that run on mobile devices such as the Pixel 3. This operation allows fast execution without an accuracy degradation. However, on edge gpu, as the platform has more memory resources, 4GB for the Jetson TX2, bigger models from NAS-Bench-201 with higher accuracy are obtained in the Pareto front.

When our methodology does not reach the best accuracy, see results on TPU Board, our final architecture is 4.28x faster with only 0.22% accuracy drop. Ta-

Table 3.6: Comparison of Optimal Architectures obtained in the Pareto Front for ImageNet

Architecture	Jetson TX2		TPU Board		Google Pixel 3		FPGA ZCU102		HW Aware
	Acc (%)	Lat (ms)	Acc (%)	Lat (ms)	Acc (%)	Lat (ms)	Acc (%)	Lat (ms)	
GATES [16]	74.8	6.35	74.26	6.84	73.4	7.5	72.3	7.51	Yes
BRP-NAS [17]	75.2	5.8	74.39	6.9	74.8	7.4	75.6	6.82	Yes
ProxylessNAS [235]	75.1	5.1	75.2	4.5	74.6	6.8	74.5	5.48	Yes
GPUNet [243]	78.9	8.6	78.9	7.9	78.9	13.5	78.9	8.1	No
FBNetV3 [244]	77.6	5.4	78.6	5.9	76.8	7.6	78.5	5.3	Yes
HW-PR-NAS	**77.68**	**4.67**	**76.75**	**3.25**	**77.24**	**6.2**	**77.84**	**4.6**	Yes

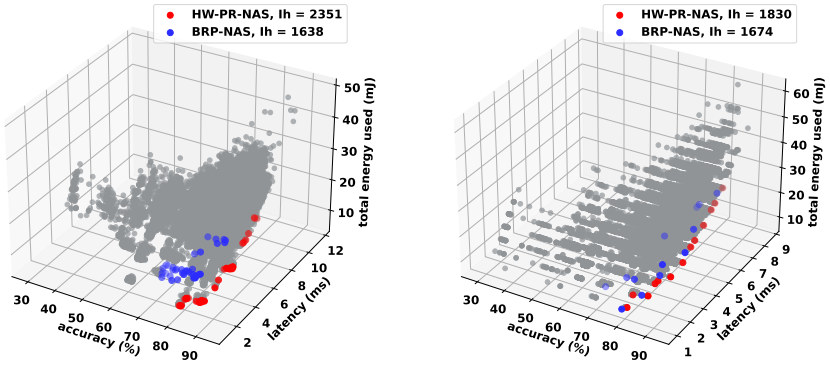

Figure 3.9: Pareto front Approximations using three objectives: accuracy, latency and energy consumption on CIFAR-10 on Edge GPU (left), FPGA (right). Ih corresponds to the hypervolume.

ble 3.6 summarizes the comparison of our optimal model to the baselines on ImageNet. GATES [16] and BRP-NAS [17] are re-run on the same proxylessNAS search space, i.e., we trained the same number of architectures required by each surrogate model, 7318 and 900, respectively. FBNetV3 [234] and ProxylessNAS [235] were re-run for the targeted devices on their respective search spaces. GPUNet [243] targets V100, A100 GPUs. For the comparison, we take their smallest network deployable in the embedded devices listed. Results show that HW-PR-NAS outperforms all other approaches regarding the trade-off between accuracy and latency. However, we do not outperform GPUNet in accuracy but offer a 2x faster counterpart.

3.2.5 Generalization to More Objectives

In this paper, generalization refers to the ability to add any number or type of expensive objectives to HW-PR-NAS. This can simply be done by by fine-tuning the Multi-Layer Perceptron (MLP) predictor. This is possible thanks to the following characteristics: (1) The concatenated encodings have better coverage and represent every critical architecture feature. The proposed encoding scheme can represent any arbitrary architecture. This enables the model to be used with a variety of search spaces. (2) The predictor is designed as one MLP that directly predicts the architecture's Pareto score without predicting the individual objectives.

Figure 3.9 illustrates the model's results with three objectives: accuracy, latency and energy consumption on CIFAR-10. We compare our results against BPR-NAS for accuracy and latency and a lookup table for energy consumption. That means that the exact values are used for energy consumption in the case of BRP-NAS.

The Pareto ranking predictor has been fine-tuned for only five epochs, less than 5min training times. The encoding component was frozen (not fine-tuned). The Pareto front is of utmost significance in edge devices where the battery lifetime is crucial. It allows the application to select the right architecture according to the system's hardware requirements.

Architecture	Jetson TX2	
	Acc (%)	Lat (ms)
DS-CNN [245]	96.02	28
CENet-GCN-40 [246]	96.8	32.5
LeTR [247]	97.56	22.36
KWT [248]	97.28	18.5
HW-PR-NAS-KWS	**97.89**	**13.68**

Table 3.7: Accuracy and Latency Comparison for Keyword Spotting.

3.2.6 Generalisation to other use cases: Keywords Spotting

While the Pareto ranking predictor can easily be generalized to various objectives, the encoding scheme is trained on ConvNets architectures. In this use case, we evaluate the fine-tuning of our encoding scheme over different types of architectures, namely recurrent neural networks (RNNs) on Keyword spotting. The goal is to assess how generalizable is our approach. The task of keyword spotting (KWS) [249] provides a critical user interface for many mobile and edge applications, including phones, wearables, and cars. It detects a triggering word such as "Ok, Google" or "Siri". These applications are typically "always-on", trying to catch the triggering word, making this task an appropriate target for HW-NAS. We use NAS-Bench-NLP for this use case.

NAS-Bench-NLP [250] is a benchmark containing 14k RNNs with various cells such as LSTMs and GRUs. While it is possible to achieve good accuracy using ConvNets, we deliberately use RNNs for KWS to validate the generalization of our encoding scheme.

Similarly to NAS-Bench-201, we extract a subset of 500 RNN architectures from NAS-Bench-NLP. We measure the latency and energy consumption of the dataset architectures on Edge GPU (Jetson Nano). We train our surrogate model. The training is done in two steps described in section 3.2.2. We first fine-tune the encoder-decoder to get a better representation of the architectures. Then using the surrogate model, we search over the entire benchmark to approximate the Pareto front.

Figure 3.10 shows the training loss function. The full training of the encoding scheme on NAS-Bench-201 and FBNet required 80 epochs to achieve a cross-entropy loss of 1.3. Fine-tuning this encoder on RNN architectures requires only eight epochs to obtain the same loss value. This test validates the generalization ability of our encoder to different types of architectures and search spaces.

Figure 3.11 shows the Pareto front approximation result compared to the true Pareto front. After a few minutes of fine-tuning, we can adapt our surrogate model to a new search space and achieve a near Pareto front approximation with 97.3% normalized hypervolume. We select the best network from the Pareto front and compare it to state-of-the-art models from the literature. Table 3.7 shows the results. Our model is 1.35x faster than KWT [248] with 0.33% accuracy increase than LeTR [247].

3.3 PRP-NAS: Pareto Rank-preserving Supernetwork Training

Weight sharing is an estimation strategy used to avoid the training and hence speeds up the search step. It mainly formulates the search space into a *supernetwork*. A supernetwork is an over-parameterized architecture where each path can be sampled. We call a sampled path *a sub-network*. These methods assume that the rank between different sub-networks is preserved when using the supernetwork's weights. Two architectures with the same rank imply that they have the same accuracy. State-

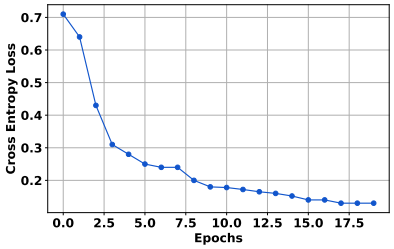

Figure 3.10: Encoder fine-tuning: Cross-entropy loss over epochs.

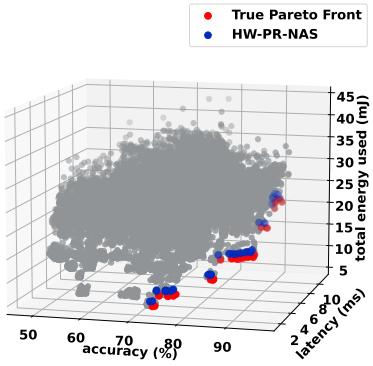

Figure 3.11: Search result using HW-PR-NAS against True Pareto front.

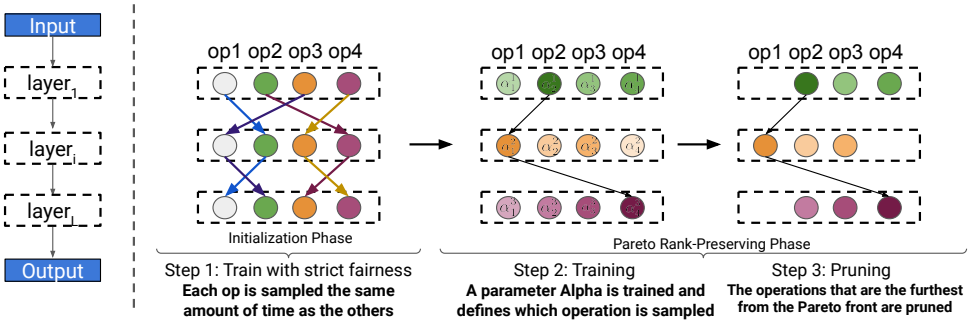

Figure 3.12: Our Pareto Rank-Preserving Training methodology for Supernetwork. The strongest shades illustrate the most important operations for each layer at each iteration. α_o^l corresponds to the parameter alpha associated with layer l and operation o.

of-the-art techniques [251, 252, 253] have highlighted the performance estimation inefficiency used in both categories by computing the ranking correlation between the architectures' actual rankings and the estimated rankings. This issue is called *optimization gap* [254]. Some solutions have been proposed to train the supernetwork with strict constraints on fairness to preserve the ranking for accuracy, such as FairNAS [164]. Others train a graph convolutional network in parallel to fit the performance of sampled sub-networks [255].

However, current solutions have two main drawbacks:

1. In the multi-objective context of HW-NAS, different objectives such as accuracy and latency have to be estimated. The result is a Pareto front; a set of architectures that better respects the trade-off between the conflicting objectives. The ranking based on a single objective is no longer a good estimator. In this setting, we need to take into account the dominance criterion in the ranking.

2. Many of the existing approaches search for efficient architectures through some metrics that rank the architectures based on a single objective such as Kendall tau's correlation between the accuracies of the samples networks [235, 256]. Such metrics do not measure the performance of the Pareto front. Defining a general metric to determine HW-NAS evaluation strategies is long overdue. This metric should account for multiple objectives.

To overcome the aforementioned issues, we propose a new training methodology

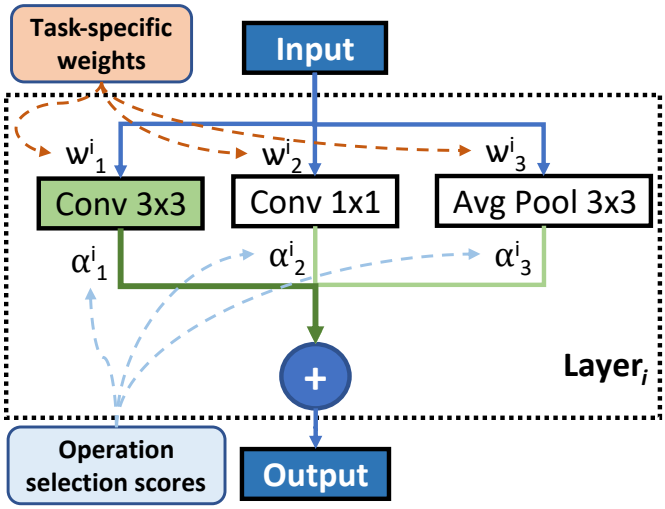

Figure 3.13: Supernetwork definition when coupling task-specific weights W and operation's score parameters α. *Conv 3x3* is the operation with the highest selection score.

for supernetwork to preserve the Pareto ranking of sub-networks in HW-NAS and avoid additional ranking correction steps.

In this section, we present the following contributions:

- We define the **Pareto ranking (PR) as a novel metric** to compare multiple HW-NAS evaluation techniques in the multi-objective context. Our study shows that optimizing this metric while training the supernetwork allows the Vanilla Weight-sharing NAS to achieve a 92% near optimal Pareto front.
- We introduce a novel **Differentiable weight-sharing supernetwork training methodology**. The training optimizes the task-specific loss function (e.g. cross-entropy loss) and a Pareto ranking listwise loss function to accurately select the adequate operation per layer.
- During training, **we prune the operations that are the least likely to be in the architecture of the optimal Pareto front**. The pruning is done by overlapping the worst Pareto-ranked sub-networks and removing the operations that are only used in these sub-networks.

Our methodology has been evaluated on three state-of-the-art NAS benchmarks: NAS-Bench-201 [233], DARTS ([166]) and ProxylessNAS [235]. The obtained results show that our approach allows us to achieve a higher Pareto front approximation compared to current state-of-the-art methods. For example, we obtained 97% Pareto front approximation when One-Shot-NAS-GCN [255] depicts only 87% on NAS-Bench-201.

3.3.1 Proposed Approach

The core motivation for a novel training methodology is to achieve an efficient sub-networks evaluation for HW-NAS. The proposed training methodology must preserve the Pareto ranking between different sub-networks while reducing the overall training time.

Pareto Ranking Similar to HW-PR-NAS, we define the Pareto ranking metric used to train and evaluate the supernetwork.

Solving the multi-objective optimization problem on a set of sub-networks results in a Pareto front, denoted as F_1, i.e., all the architectures have a rank of 1. We achieve

the lower ranks by successfully solving the problem on the set of sub-networks pruned from the previous solutions. The lowest rank is assigned to the sub-networks that do not dominate any sub-network. We formally define the Pareto ranking in equation 3.8, where S is the entire supernetwork, $F_{k'}$ is a set of sub-networks ranked k', and $\succ$ is the dominant operation.

Using this ranking scheme, multiple architectures may have the same rank. This happens when none of them can dominate the others.

$$a \text{ is ranked } k \iff \forall \hat{a} \in S - \bigcup_{s_i \in F_{k\prime} \wedge k\prime > k} , \; a \succ \hat{a} \tag{3.8}$$

Pareto Ranking Correlation. We evaluate the quality of an estimator using ranking correlations such as Kendall's tau-b Correlation or Spearman Correlation. Kendall's tau-b determines whether there is a monotonic relationship between two variables and is suitable when variables contain many tied ranks [224], which is our case. We compute Kendall's Tau-b correlation between the ground truth ranks (i.e. the Pareto ranks obtained from independently training the sub-networks), and the Pareto ranks obtained by evaluating each architecture with the supernetwork shared weights.

Pareto Rank-Preserving Training

Our training methodology aims at preserving the Pareto ranking obtained by the weight-sharing evaluation.

Figure 3.13 shows a representation of a layer in the supernetwork definition and the different parameters we aim to learn. A sub-network is a path from the input to the output. All extracted sub-networks are of the same depth. We train the supernetwork with two goals: 1) enhance the task-specific loss function by adjusting W, the task-specific weights of the original model associated with the neural network operations such as the kernels in convolution, and 2) improve the Pareto ranking loss between its different paths by adjusting α, the weights associated with the operation selection. α measures which operation is critical and which one is selected.

Algorithm 2 and figure 3.12 summarize the training procedure.

- **Step 1: Train with Strict Fairness** We train our supernetwork using FairNAS [164] strict fairness constraint. This step adjusts the weights of all the sub-networks W and gives a good starting point for the Pareto ranking training. Additionally, the accuracy estimation on the task-specific loss at this point is well estimated. We use these estimations to compute the true Pareto ranks in case no accuracy was provided by the benchmark.

- **Step 2: Pareto ranking training** For each iteration, we apply:

  - Training to solve the task: A mini-batch is sampled from the training set, and a sub-network is chosen according to each operation's highest α. The operation's weights are updated using the task-specific loss, e.g., cross-entropy loss for image classification.

  - Pareto rank training: In this phase, we purposefully bias the training towards better Pareto-ranked architectures using the α parameters. α parameters are trained using the loss function provided in equation 3.9. During the forward pass, we Pareto rank the sampled sub-networks. We compute the number of times an operation op_i appears in layer l_j on N top-ranked sub-networks, denoted as $g(op_i, l_j)$. N is a hyperparameter defined before training. We denote by $\hat{g}(op_i, l_j)$, the ground truth. Equation 3.9 computes the hinge loss over all layers in the sampled sub-networks and compares the number of times the operation with the highest α appears in the predicted Pareto front and the ground truth one. Here, m is a fixed margin that controls the amount of penalization for violating the ranks, which is set to 0.1.

$$L = \sum_{j=1}^{L} \sum_{i, g(op_i, l_j) > \hat{g}(op_i, l_j), i \neq (\alpha)} max[0, m - g((\alpha), l_j) - \hat{g}(op_i, l_j)] \quad (3.9)$$

We adjust each operation's α parameters and compute each sampled sub-network's latency using a lookup table. We define the predicted Pareto score according to $P_s = \sum_{op \in a} \alpha_{op}$, i.e., the sum of selected operations' alpha values. Next, we compute the listwise ranking loss defined by the cross entropy between the ranking scores and the Pareto ranks (ground truth).

To compute the ground truth, we iteratively calculate the loss and latencies of the sub-networks to determine the optimal Pareto front using equation 3.8. This takes only a few milliseconds, thanks to the small number of selected sub-networks.

- **Step 3: Pruning by Pareto Ranking Sub-networks** We drop sub-networks furthest from the optimal Pareto front to accelerate the training. First, we select the sub-networks belonging to the two first Pareto ranks. Then, based on the hypervolume improvement (HVI) [257], we select n sub-networks. The operations never used by any sub-network in this selection are removed for each layer. Equation 3.10 illustrates how the hypervolume improvement is computed in this context. o_{ij} denotes operation i in layer j, and P refers to the current set of sub-networks constituting the Pareto front approximation. HV denotes the hypervolume function and $\{S_{o_{ij}}\}$ denotes the set of sampled sub-networks using operation i in layer j, and P refers to the current set of sub-networks constituting the Pareto front approximation. HV denotes the hypervolume function and $\{S_{o_{ij}}\}$ denotes the set of sampled sub-networks using operation i in layer j. When computing the hypervolume, a reference point is required. We carefully selected the reference point by examining a range of pre-sampled sub-networks with different accuracy and latency values. Similar to multi-objective HW-NAS, we chose the sub-network that resulted in the highest HVI for the initial round of sub-network ranking.

$$HVI(o_{ij}, P) = HV(P \bigcup \{S_{o_{ij}}\}) - HV(P - \{S_{o_{ij}}\}) \quad (3.10)$$

Finally, going over all the layers to select the operations with the highest α would suffice to find the most efficient DNN within the search space.

Figure 3.14 shows the training results. We compare our methodology to FairNAS [164] strict fairness training. During training, the Pareto ranking correlation increases with the quality of the estimations. When using our training methodology without considering the alpha parameters, the ranking correlation saturates at 0.7. FairNAS achieves the same behavior with reduced variance among the different training runs. However, if we consider the alpha parameters, the selection is more efficient and the architectures' rankings are well represented with 0.94.

The FairNAS step is crucial in optimizing the classification loss and ensuring that each operation is trained fairly. We conducted experiments to compare the performance of our methodology with and without FairNAS. Using training from scratch (without FairNAS), where the initial model's weights were random, we found that the model's weights were poorly selected, resulting in a badly chosen set of operations that could negatively impact accuracy and convergence. The top-1 accurate sub-networks only achieved 86.3%. We found that the Kendall-tau correlation of the training without FairNAS was only 0.62 after 500 epochs, significantly lower than the correlation obtained with FairNAS, see figure 3.2. These results will be added to Figure 3 to highlight the necessity of the FairNAS step.

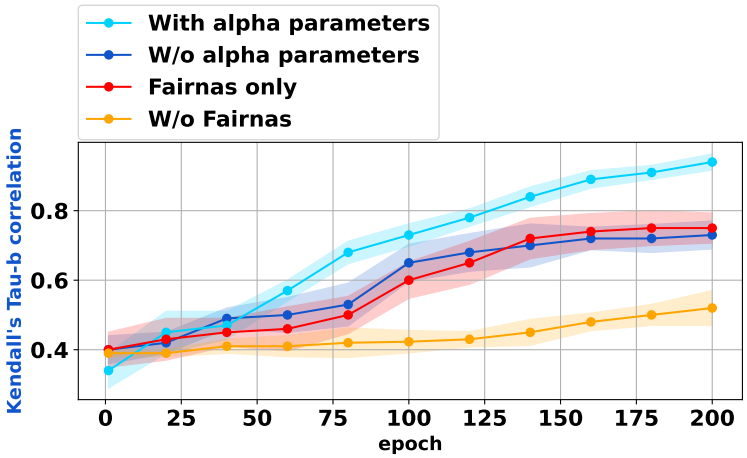

Figure 3.14: Training performance computed with the Kendall's Tau Correlation between the independently trained Pareto ranks and the estimated Pareto ranks obtained by training the supernetwork.

Algorithm 2 Supernetwork Training Algorithm

Input: Search space S, number of epochs for fairness training N_f, number of epochs for Pareto training N_p, Supernetwork parameters (W, α), training dataloader D, task-specific loss $Loss$, Pareto raking loss $Loss_{PR}$, number of sampled sub-network n
Procedure: Train
Initialize W and α for each operation in Supernetwork
Strict fairness training for N_f epochs
for i=1 to N_p **do**
  for data, labels in D **do**
    Build *model* with $argmax(\alpha)$ following step 2
    Reset gradients to zero for all W parameters
    Calculate gradients based on $Loss$, data, labels and update W by gradients
  end for
  end for
  Sample n sub-networks, *models*
  Compute: Pareto rank of *models*, $Loss_{PR}$ between scores and Pareto rank.
  Update α by gradients
  Apply pruning following step 3
  end for
end for

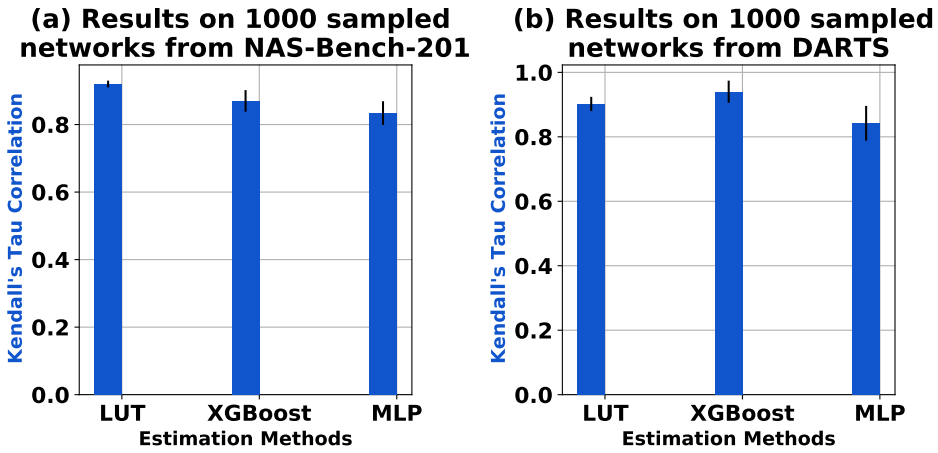

Figure 3.15: Comparison of latency estimators on Jetson Nano.

Latency Estimation
To estimate the latency of each sub-network during training, we used a lookup table approach that contains a mapping between each operation and its latency on each hardware platform. To populate the look-up table, we first generated a large set of candidate sub-networks using a random search. We then measured the latency of each operation and averaged it across multiple sub-networks.

During training, we estimated the latency of each sub-network by summing up the corresponding latencies in the look-up table.

In this section, we compare different latency estimators to validate the use of LUT during the search. Figure 3.15 shows the results. We randomly extract 1000 architectures from NAS-Bench-201 and 1000 from DARTS. We measure the exact latency on Jetson Nano for each architecture. We train two predictor-based models, namely XGBoost and MLP with 3 layers. The training dataset contains 700 architectures and 300 were used for testing. On NAS-Bench-201, the architectures have a sequential execution which made LUT the most accurate in terms of latency ranking the architectures. On DARTS, XGBoost prediction was the most suitable method. But, LUT was not far with 0.915 against 0.942. Computing the LUT in our algorithm is simpler. Using a hook during the forward function on a PyTorch model is sufficient and much more direct than calling a surrogate model. We thus use this strategy to estimate the latency in our method.

3.3.2 Evaluation Methodology

Search Spaces: Several search spaces have been used to evaluate our method's performance. NAS-Bench-201 ([233]) is a tabular benchmark that contains 15k convolutional neural networks. Each architecture is trained on CIFAR-10, CIFAR-100 and ImageNet-16-120 [237]. We use the latency and energy consumption values obtained from HW-NAS-Bench [256]. DARTS [166] is a supernetwork benchmark that contains 10^{18} architectures. Each architecture is trained on CIFAR-10 and is transferable to ImageNet. We also validate our methodology on ImageNet using ProxylessNAS search space [235] whose size goes to 6^{19}. When the true latency is not available in the benchmark, we use a lookup table to estimate it. Our preliminary analysis showed that lookup table achieve 0.92 latency rank correlation on a 1000 sampled architecture from DARTS and NAS-Bench-201; a +5% than XGBoost predictor.
Training Hyperparameters The training hyperparameters are listed in Table 3.8. It takes 2, 3.8, 3.8 GPU-days for NAS-Bench-201, DARTS and ProxylessNAS search space to train each supernetwork to fullness. Our training is 5x faster than previous

works due to the pruning strategy. To be consistent with previous works, we do not employ data augmentation tricks such as cutout or mixup. We also do not employ any special operations such as squeeze-and-excitation. All these methods can further improve the scores on the test set.

Table 3.8: Training Hyperparameters

Benchmark	Hyperparameter	Value
NAS-Bench-201	Nf	20
	Np	50
	n	50
	batch_size	128
	lr	0.01
	optim	SGD
	momentum	0.9
DARTS	Nf	30
	Np	150
	n	100
	batch_size	256
	lr	0.025
	optim	SGD
	momentum	0.9
ProxylessNAS Search Space	Nf	30
	Np	150
	n	100
	batch_size	256
	lr	0.025
	optim	SGD
	momentum	0.9

3.3.3 Search Results

In these experiments, we consider two objectives: accuracy and latency (inference time). The latency is either given by HW-NAS-Bench [256] or computed using a lookup table as explained in the Proposed Approach section.

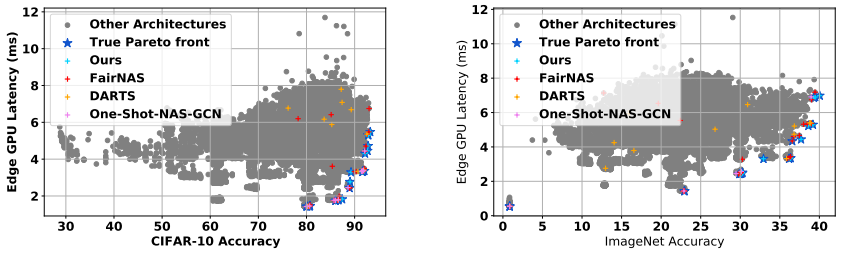

Figure 3.16: Pareto front approximation comparison on CIFAR-10 and ImageNet.

Figure 3.16 shows the Pareto front approximations obtained using different methods on NAS-Bench-201 for CIFAR-10 and ProxylessNAS Search space for ImageNet. We obtain a 10% hypervolume increase on NAS-Bench-201 and a 43% hypervolume increase on ImageNet compared to the best baselines: One-Shot-NAS-GCN and FairNAS, respectively.

Table 3.9: Comparison on NAS-Bench-201 CIFAR-10 on Edge GPU (Jetson Nano) and Mobile phone (Pixel 3).

Architecture	Edge GPU			Mobile Phone : Pixel 3			HW Aware	GPU Days
	Top-1 Test Acc.	Params (M)	Latency (ms)	Top-1 Test Acc.	Params (M)	Latency (ms)		
DARTS [166]	68.3 ± 0.08	3.4	5.36	68.3 ± 0.08	3.4	11.4	No	4
ENAS [258]	53.89 ± 0.16	4.6	6.32	53.89 ± 0.16	4.6	19.8	No	0.16
GDAS [259]	90.89 ± 0.08	3.4	5.21	90.89 ± 0.08	3.4	10.36	No	0.21
FairNAS [164]	93.23±0.18	3.2	4.68	92.4 ± 0.15	3.6	8.65	Yes	10
PRP-NAS-BL (Ours)	92.34 ± 0.05	3.0	**2.3**	89.54 ± 0.07	2.8	**3.6**	Yes	2
PRP-NAS-BA (Ours)	**94.37 ± 0.02**	4.5	4.35	**94.2 ± 0.03**	4.3	5.6	Yes	2
PRP-NAS-O (Ours)	93.65 ± 0.01	4.3	3.64	93.74 ± 0.00	3.4	4.61	Yes	2

Search on NAS-Bench-201

Table 3.9 shows the results of our methodology on NAS-Bench-201 compared to state-of-the-art methods. PRP-NAS-BL, PRP-NAS-BA and PRP-NAS-O are three sampled architectures from our final Pareto front. BL stands for "Best Latency". BA stands for "Best Accuracy", and O stands for "Optimal". Notably, our architecture obtains highly competitive results. The optimal architecture, *PRP-NAS-O*, outperforms current state-of-the-art methods in accuracy and latency. Including hardware awareness during the search allows us to obtain flexible results according to the targeted hardware platform. Besides, multiple training runs show the stability of our method compared to other baselines. The acceleration in the search cost is mainly due to applying the pruning while training. This cost can vary according to the used GPU. We used GPU V100 to train the supernetwork. Results on other targeted platforms are presented in table 3.10. Our methodology consistently finds better Pareto extracted solutions regardless of the targeted hardware platform.

Table 3.10: Comparison to baselines on CIFAR-10 on FPGA ZCU-102 and Raspberry Pi3

Architecture	FPGA ZCU102			Raspberry Pi 3			HW Aware	GPU Days
	Top-1 Test Acc.	Params (M)	Latency (ms)	Top-1 Test Acc.	Params (M)	Latency (ms)		
DARTS	68.3 ± 0.08	3.4	7.32	68.3 ± 0.08	3.4	45.36	No	4
ENAS	53.89 ± 0.16	4.6	8.91	53.89 ± 0.16	4.6	35.8	No	0.16
GDAS	90.89 ± 0.08	3.4	4.98	90.89 ± 0.08	3.4	41.8	No	0.21
FairNAS	92.9±0.23	3.4	5.12	92.51 ± 0.9	3.3	34.15	Yes	10
PRP-NAS-BL (Ours)	91.35 ± 0.04	3.2	**3.6**	88.7 ± 0.03	2.4	**7.6**	Yes	2
PRP-NAS-BA (Ours)	**94.37** ± 0.005	4.9	6.8	**93.68** ± 0.05	4.68	40.7	Yes	2
PRP-NAS-O (Ours)	93.55 ± 0.04	4.2	4.23	92.54 ± 0.02	3.6	18.5	Yes	2

Search on ImageNet

Similar conclusions can be extracted when searching on ImageNet. Figure 3.17 summarizes the results. Our supernetwork is now based on proxylessNAS [235]. Our optimal model surpasses FairNAS-A (+1.9%) and One-Shot-NAS-GCN (+1.7%) while running faster. Training on Imagenet is time-consuming due to the difference in image resolution, which explains the increase in the search cost. We still surpass most of the methods in terms of search time. We compare two ProxylessNAS architectures; ProxylessNAS-R is specific to Mobile inference.

When using data augmentation and architecture tricks, namely Squeeze-and-excitation and AutoAugment, in the optimal architecture, we achieve 78.6% accuracy on Imagenet.

In addition, we compare our methodology to OFA [169]. Using the proxylessNAS supernetwork, we are able to find a 1.39x faster architecture with a drop of 0.56%. Using our training methodology on OFA's supernetwork directly allows us to find a +2.2% accurate and 1.5x faster architecture on Pixel 3 and a +1.4% accurate and 2.3x faster architecture on Jetson Nano. These results can be attributed to

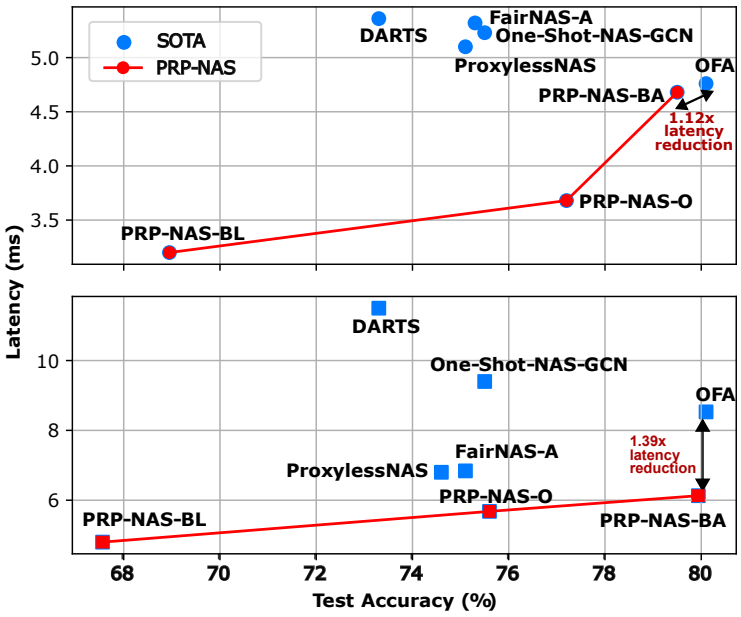

Figure 3.17: Comparison with state-of-the-art ImageNet results.

the fact that our methodology searches for architectures that are optimized for each specific hardware platform, whereas OFA optimizes for a variety of hardware platforms simultaneously. By focusing on specific hardware platforms, our methodology is able to tailor the architecture to the specific computational resources, and latency requirements of each platform, leading to superior results on those platforms.

Ranking Quality

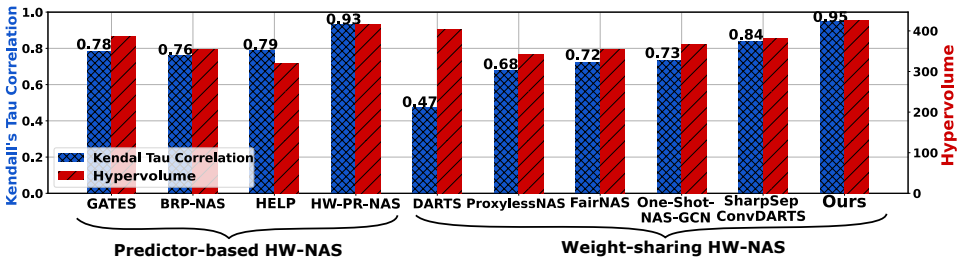

Figure 3.18: Kendall's Tau-b correlation and hypervolume comparison using different estimators on DARTS.

We compare different estimators used in HW-NAS using Kendall's Tau Correlation between the predicted Pareto ranks and the Pareto ranks obtained from independently training the architectures. These latter are extracted from NAS-Bench-201. Figure 3.18 shows the correlation results. In general, it is more complex to train a supernetwork to respect the Pareto ranks because of the impact of the sub-networks on each other, i.e., the outputs of each layer are summed together. The increase in Kendall's tau correlation of the previous weight-sharing methodology is due to the improvement in the accuracy estimation provided by the supernetwork.

Predictor-based evaluators use the learning-to-rank theory and train their predictors only to predict the ranking. Methods such as GATES [260] or BRP-NAS [261] train many independent predictors, one for each objective. HW-PR-NAS [226] trains

a single predictor to fit the Pareto ranks. However, their methodology is not flexible for supernetwork training.

Ablation Study

We validate the results of our pruning algorithm by comparing the results of our algorithm with and without it in table 3.11. Without pruning, the search time exponentially increases from 3.8 GPU days to 15.1. However, the hypervolume improves slightly. The final most accurate architecture is in both Pareto front obtained with and without pruning. The optimal architecture using pruning is better in terms of accuracy and latency. The latency is computed on Jetson Nano Edge GPU.

Model	Test Acc (%)	Latency (ms)	Search Hyper-volume	GPU days
PRP-NAS-O	**93.65**	3.64	423.45	**3.8**
PRP-NAS-O-no_pruning	92.1	3.26	433.09	15.1

Table 3.11: Ablation results of Pruning of Pareto ranking for CIFAR-10.

Number of sampled sub-networks

Figure 3.19 shows the effect of increasing the number of sampled sub-networks on the search results. Generally, increasing the number of samples, increases the hypervolume. The hypervolume is used to evaluate Pareto front approximations. It computes the area contained by the Pareto front points found by the search and a reference point. Our reference point is set as a pre-sampled architecture from the supernetwork, with a low accuracy and high latency. When the number of sampled sub-networks is too high, each layer's output is the sum of multiple operations that can or cannot be within the final Pareto front which induces a bias when adjusting the *alpha* parameters.

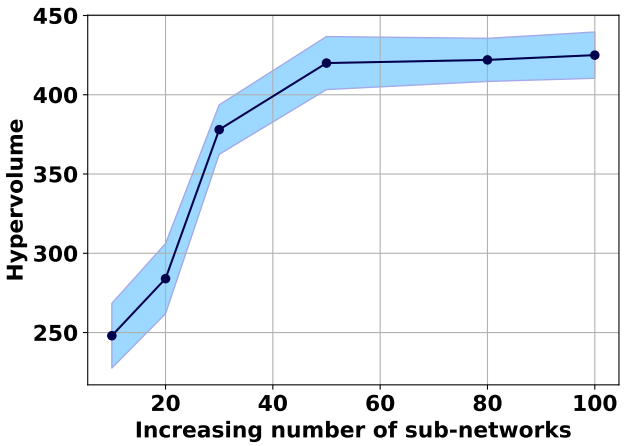

Figure 3.19: Hypervolume analysis with an increasing number of sampled sub-networks for the final Pareto front throughout the search (higher is better) on NAS-Bench-201.

Analysis of α parameter

Figure 3.20 illustrates the evolution of alpha parameters for each operation in layers 1 and 2 during the training. It clearly shows how alpha favors one operation over the others during training. At the end of the training, we take the operations with the highest alpha that represents the operations constructing architectures in the final Pareto front. If one layer has a clear candidate such as layer 1, with conv3x3 that exceeds 60%, this operation is then chosen. If a layer contains multiple operations with similar alpha values, we construct all the path of that layer.

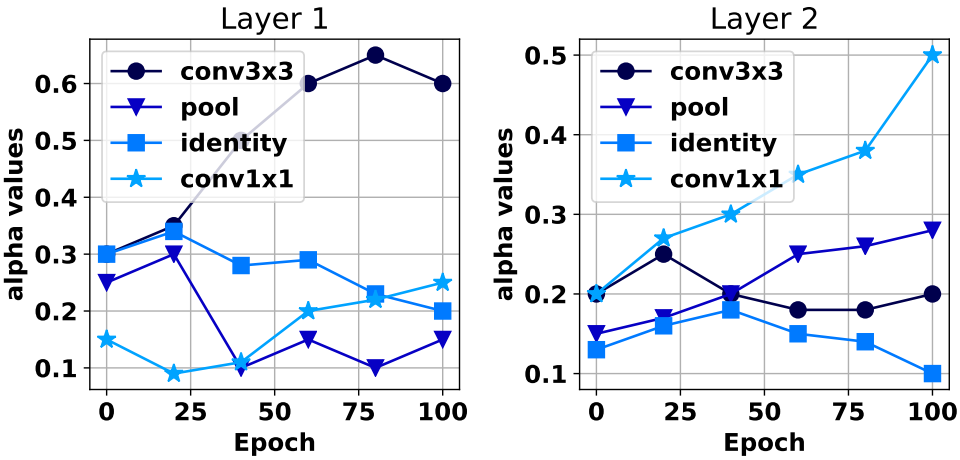

Figure 3.20: Analysis of trained alpha values for layers 1 and 2

3.3.4 Battery Usage Preservation

The amount of energy consumed by each model can be different. It is mainly attributed to the number of multi-adds computed. We take supernetwork usage to another level by adequately scheduling the run of different sub-networks according to the system's battery life. In this experiment, the training is done with two objectives: accuracy and energy consumption. Once the training is done, only the Pareto front solutions are kept in the supernetwork, thanks to the pruning. We further select, from the final Pareto front, s architectures. In this experiment $s = 5$. The total size of the supernetwork is then reduced to 20.5MB, comparable to MobileNet-V3 Large with 21.11MB. We deploy the model on a smartphone application that is always on. The application repeats the inference classification of one image. The application initially uses the sub-network with the highest accuracy. We switch to a lower-accurate model every five hours for better energy preservation. Figure 3.21 shows the results of the system's battery life while running the application for 24 hours. We use three scenarios:

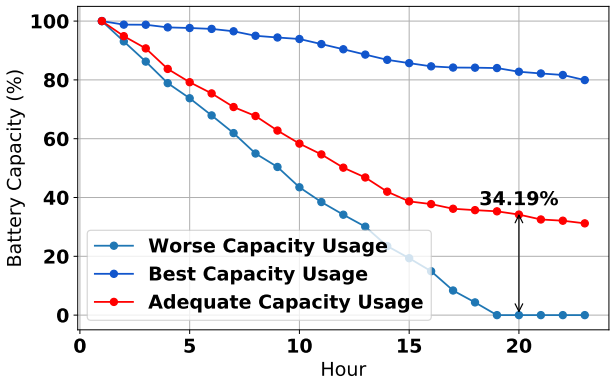

Figure 3.21: Battery life management.

1. **Worst Battery Usage:** From the Pareto front, we select the most accurate architecture: *PRP-NAS-BA*. This is the only architecture the application runs and is the only one loaded in memory.

2. **Best Battery Usage:** Similar to the worst battery usage, we select the most energy-efficient.

3. **Adequate Battery Usage:** We load the complete supernetwork and switch the sub-network every 5 hours.

Using this strategy helps save up to 34% of the battery life while using highly accurate models most of the time. The average accuracy of the five selected sub-networks is 75.2%.

3.4 Conclusion

This chapter introduces HW-PR-NAS and PRP-NAS. Both contributions enhance the evaluation component of HW-NAS.

HW-PR-NAS is a surrogate model-based HW-NAS methodology, to accelerate HW-NAS while preserving the quality of the search results on cell-based and global search spaces. HW-PR-NAS proposes a novel encoding methodology that offers several advantages: (1) It generalizes well with small datasets, which decreases the time required to run the complete NAS on new search spaces and tasks, (2) It is also flexible to any hardware platforms and any number of objectives. This approach was evaluated on seven hardware platforms such as Jetson Nano, Pixel 3, and FPGA ZCU102. Experimental results show that HW-PR-NAS delivers a better Pareto front approximation (98% normalized hypervolume of the true Pareto front) and a 2.5x speedup in search time. We show that HW-PR-NAS outperforms state-of-the-art HW-NAS approaches on seven edge platforms.

PRP-NAS analyzes Hardware-aware weight-sharing NAS where the multi-objective context requires the estimator to preserve the Pareto rankings between sub-networks accurately. Contrary to existing approaches that estimate each objective independently, we propose a supernetwork training methodology able to preserve the Pareto rankings during the search. We achieve 97% near Pareto front approximation on NAS-Bench-201, DARTS, and ProxylessNAS Search Spaces. We find a 77.2% accuracy model on ImageNet while only training the supernetwork for 3.8 days. Using the supernetwork capabilities, we saved up to 34% of the battery capacity with an average accuracy of 75.2% on ImageNet.

Chapter 4

Enhancing HW-NAS Search Space

Contents

4.1 Context

Undoubtedly, the most critical component of HW-NAS is its search space. The search space defines the types of architectures that are explored and the range of performance that can be achieved. A large and well-designed search space allows for a broader exploration of the design space and can lead to the discovery of more efficient and effective architectures. However, an overly large search space can also lead to increased computational costs and longer search times, making it impractical for many real-world applications. Therefore, the design of an effective search space is a critical challenge in HW-NAS research. In this chapter, we present two contributions to the search space definition.

First, when exploring compression methods for neural networks, a critical challenge is how to efficiently define a search space. Compression techniques, such as pruning and quantization, can significantly reduce the model size and computational cost of neural networks, but the search space for finding an optimal compressed architecture can be large and complex. Therefore, we present *CaW-NAS*, compression-aware neural architecture search, to answer research question 2.

Research Question 2

What is an efficient and effective method for defining a search space that contains diverse and high-performing compressed neural architectures while minimizing the computational cost of the search process?

Second, current search spaces are typically designed using either a macro-architecture that closely resembles a standard handcrafted architecture or a list of pre-defined operations obtained from handcrafted architectures. While this design approach allows for the identification of the top-performing combinations of pre-existing operations, it falls short when it comes to discovering novel and innovative architectures. As a result, the search process may be constrained by the pre-existing assumptions and biases inherent in these design choices, limiting the potential for true innovation in neural architecture design. Hence, propose *GOS*, Grassroots operation search to answer research question 3.

Research Question 3

What are effective approaches for constructing a search space that is not influenced by previous human experience, and can enable the discovery of novel and innovative neural architectures?

4.2 CaW-NAS: Compression-aware Neural Architecture Search

Quantization, as a model compression technique, aims at decreasing the bit width used to represent the model's weights and activations in memory. This dramatically reduces both the memory requirement and computational cost. However, the quantization induces a drop in the model's accuracy which varies according to the architecture, the chosen bit width, and the dataset.

Including the quantization bit width in the search space will considerably increase the computational complexity of HW-NAS and makes the search impractical. In traditional methods, a multi-stage search is used. A first search for the architecture using HW-NAS is applied and then a second search for the quantization bit width is realized. This second stage is called *specialization*. However, this specialization delivers a sub-optimal solution as the drop in accuracy induced by quantization depends on the architecture obtained by the HW-NAS in the first stage.

To solve the large search space problem, APQ [140] proposes a supernetwork in which each operator (e.g. conv, fully connected, etc.) is defined with its associated bit width. APQ uses a surrogate model to predict the accuracy of the quantized model. However, a supernetwork restricts the search space depending on the model's macro-architecture.

We present CaW-NAS, a solution that dynamically extends the search space with the quantized versions during the search. Our approach obtains more efficient architectures by merging the two stages.

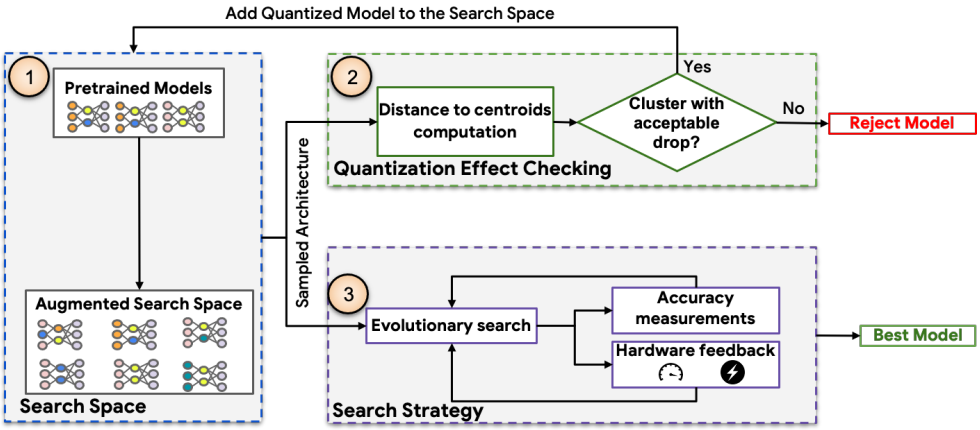

Figure 4.1: Overview of CaW-NAS: Compression Aware Neural Architecture

The contributions made by CaW-NAS are threefold:

- We propose a search strategy in the HW-NAS that considers the quantization accuracy drop. We dynamically extend the search space with the quantized version of the sampled architecture during the search if its accuracy drop is acceptable.
- We apply a clustering strategy to identify architectures with small/null drops in the quantization. We first use these insights to answer the following question: *What is the relation between quantization drop and architecture characteristics such as depth, width, and operators?* and *How does the quantization drop change when different bit widths are selected?* These results guide the exploration of the extended search space without impacting the delays.
- We validate our methodology on two benchmarks: NAS-Bench-201 [262] and a customized search space consisting of state-of-the-art standard models. We tested our method on a mobile phone platform, the Xiaomi Redmi Note 7. When compared to state-of-the-art architectures, namely APQ-B [140], our method obtained a neural architecture with a 36.5% reduction in inference time and a 1.24% increase in accuracy.

4.2.1 Proposed Approach

Figure 4.1 shows an overview of our proposed approach CaW-NAS. Our system is composed of three main parts described as follows.

1. **Search Space:** Our search space is initially defined by two subsets: 1) state-of-the-art pre-trained standard models and 2) NAS-Bench-201 search space [262]. During the NAS process, the search space is extended by quantized models that give an acceptable accuracy drop compared to their full-precision versions.

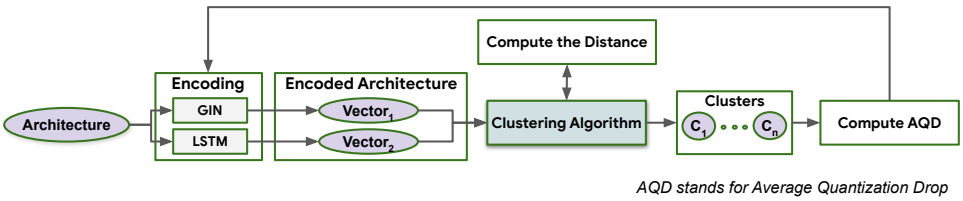

Figure 4.2: Clustering strategy to analyze the quantization sensitivity

2. **Quantization Effect Checking:** To check the quantization effect on the model's accuracy, we need to determine to which cluster belongs the sampled architecture. The cluster creation details are presented in subsection 4.2.2. Each cluster represents the sensitivity to quantization measured with the drop in accuracy. Initially, the search space only defines nonquantized architecture. Then, for each sampled architecture, and if the architecture isn't sensitive, its 8-bit quantized version is added to the search space. During the search, the search space gets bigger, and the population considers quantized architectures and nonquantized ones at the same time. By doing so, we avoid falling into a sub-optimal solution.

3. **Search Strategy:** It represents a standard multi-objectives hardware-aware NAS. We use an evolutionary search algorithm to find the best architectures in terms of accuracy and hardware efficiency trade-off. The search algorithm evaluates the accuracy and the hardware efficiency of each architecture using prediction models, named surrogate models.

The most important parts in CaW-NAS are the quantization-related parts, namely cluster creation (see figure 4.2) and *Quantization Effect Checking* (see figure 4.1). For the sake of brevity, we regroup them under the term *Quantization Analysis*. The next section describes how the quantization analysis is done using clustering.

4.2.2 Quantization Analysis

Figure 4.2 details the cluster creation part. We apply the clustering on a set of pre-trained models, which allows us to generalize the results to a broad set of search spaces. We first encode each architecture into two vectors using a Graph Isomorphism Network (GIN) [263] and a two-layer LSTM. These two vectors enable us to compute the distance between different architectures. We then use a k-means algorithm to cluster similar architectures together. The number of clusters is a user-defined hyperparameter. It depends on the desired accuracy and the search time limit. Having a large number of clusters increases the accuracy, provides better architectures, but increases the search time. In the initial clustering step, we manually assign an architecture to each cluster. For example, if the number of clusters is two we assign Resnet18 and InceptionV3 to the clusters. If the number of clusters is three, VGG16 is assigned to the third cluster. These initial points have been chosen from empirical tests. Then, we iterate over the set of models. For each architecture, we compute the average of the distances between the graph encoding vectors and the LSTM encoding vectors with the centroids of each cluster. In the last step, we check each cluster's Average Quantization Drop (AQD). We maximize the distance between the AQD of different clusters by fine-tuning the encoding GIN and LSTM and repeating the clustering.

Once we obtain the final clusters, we analyze the architectures in each cluster to understand the sensitivity of the architecture to the quantization.

1. **Effect of increasing depth and width on quantization accuracy.** From each architecture, we derive four variants with an increasing number of blocks but

the same output channels in each block for the depth analysis and five variants with increasing output channels but the same depth as the original model. The depth varies from 12 to 56, whereas the width varies from 16 to 256. We compute the percentage of similar clusters with the original architecture.

Figure 4.3 (left) shows the impact of varying the architecture's depth on the quantization effect. The number of clusters and the bitwidth are fixed to 3 and 8 respectively. Increasing the depth of the architecture increases the drop in accuracy, and the architectures are clustered differently. We can conclude that the deeper the architecture, the higher will be the quantization effect.

Figure 4.3 (right) shows the results of the quantization on the increasing width scenario. Increasing the width of the architecture doesn't necessarily increase or decrease the accuracy drop. We find that variants with different widths often stay within the same cluster.

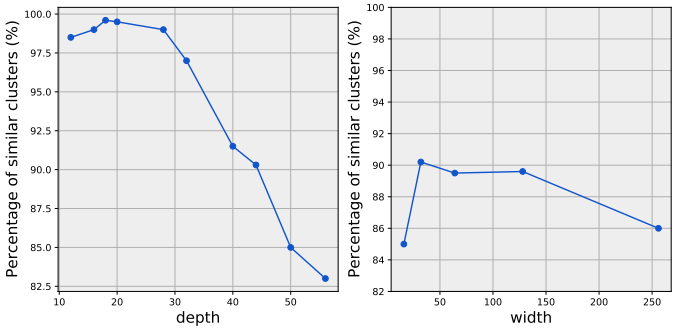

Figure 4.3: Quantization effect on increasing depth and width in the architectures

2. **Effect of quantization on different convolution variants**
Among the standard models, we can find architectures using different convolution variants: The standard convolution, the grouped convolution, and depthwise convolution. We compute the Average Quantization Drop (AQD) for each variant and present the results in figure 4.4. The AQD is computed for each block with the same output channel within architectures with the same depth. We can notice that the depthwise convolution is the most sensitive to quantization. This operator is used in architectures targeting mobile settings because it uses fewer parameters, which makes the models smaller. However, the quantization on the depthwise convolution results in a significant drop, which validates our initial assumption: The architecture search space should include quantized and non-quantized architectures even for edge devices.

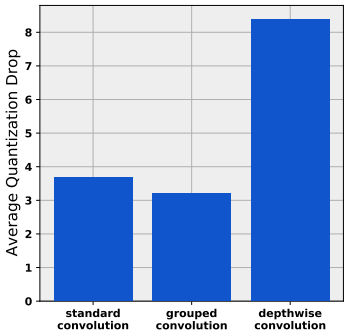

Figure 4.4: Quantization effect on different convolution variants

3. **Effect of increasing bit width on the clusters**
We also analyze the effect of decreasing the bit width from 8-bit to 4-bit and check if we have the same clusters. The percentage of similar clusters obtained is 78% which suggests that the clusters change from one quantization scheme to another.

The search space should then include the original non-quantized architectures, the 8-bit quantized architectures, and the 4-bit quantized architectures.

4.2.3 Search Strategy

In this section, we describe the search algorithm used in CaW-NAS. We use an adapted multi-objective evolutionary algorithm. The different steps are given in Algorithm 3. The algorithm implements blocks 2 and 3 in Figure 4.1.

After the clustering, we calculate the centroids of the cluster and the AQD. The AQD lets us know which cluster is sensitive to quantization. We check to which *cluster* (step 7) each architecture belongs during the search and decide whether to add its quantized version to the search space or ignore it (steps 8 and 9). Then, we compute the accuracy and latency predictions (step 10). If the architecture is quantized, the accuracy is calculated using the predicted accuracy of the non-quantized version minus the AQD of the cluster it belongs to. Using these evaluations, we construct the Pareto front, the set of non-dominated architectures in the space (step 11).

Algorithm 3 Search Algorithm

Input: search space S, centroids C, population size p, mutation probability m_prob, maximum iteration max_iter, time budget $time\_lim$
Output: Pareto front non_dominated
P = RandomPopulation(S, p)
non_dominated = $\emptyset$
i = 0
while $i \leq max\_iter$ AND $t \leq time\_lim$ **do**
  for Each architecture α in P **do**
    if α is not quantized **then**
      cluster = ComputeDistance(α, C)
      if cluster is not sensitive **then**
        Add Quantized(α) to S
      end if
    end if
  end for
  fitness = compute_fitness(P)
  non_dominated = sorting(P, fitness, k)
  mutate architecture of P with m_prob
  select n new architectures from S, add them to P
end while

4.2.4 Evaluation Methodology

All the models have been implemented using PyTorch. The global hyperparameters are listed in table 4.1.

Quantization All the quantization results and accuracy computation are done as follows. First, the fp32 models are pre-trained on ImageNet or CIFAR-10. Next, we add 30 epochs of quantization-aware training. We used two quantization schemes: 8-bit and 4-bit. We use the quantization schemes implemented in PyTorch-quantization

Clustering	
k	3
max_iter	250
tol	0,01
Search Algorithm	
population size	100
m_prob	0,8
max_iter	100
time_budget	1hr

Table 4.1: CaW-NAS hyperparameters

in the Nvidia toolkit[1]. Specifically, we quantize the weights and activations of the model with the same bit width for all our tests.

Clustering We encode each architecture into two vectors. The first is obtained with 2-layer LSTMs with 225 hidden units. The second with a 2-layer GIN with 300 hidden units. We use scikit-learn k-means implementation with the hyperparameters shown in table 4.1 (top).

Evolutionary search algorithm We experiment on two different search spaces. The first one is denoted *pretrained models*. It is constructed from 95 pre-trained models trained on ImageNet with increasing architecture depth and width. The width is increased by multiplying the output channels by a widening factor taken from 1 to 4. The depth is increased by duplicating the blocks within the architecture. Each block preserves the same number of output channels. We duplicate the block by a depth factor that varies from 1 to 4. The total number of architectures in this adapted search space is 3,112,960. The second search space is NAS-Bench-201 [262] where the architectures are trained for CIFAR-10.

We select the top 25 architectures to produce the next generation during the search. We mutate them and get 50 more architectures and we randomly sample 25 more from the search space. This random generation adds more exploration and allows us to include quantized architectures in space. We don't use any crossover.

Setup All obtained models are executed on a Redmi S7 mobile phone to gather latency and energy consumption values.

4.2.5 Search Results

In this section, we describe the final search results obtained using CaW-NAS.

Exploration Analysis

First, we study how many quantized models are considered during the search. This analysis gives information about whether we add quantized models to the search space, whether these models are considered and selected, and how many quantized models are considered over iteration search, which indicates that the search on a mixed quantized and non-quantized search space is interesting.

Figure 4.5 shows the number of quantized models in the search space and the population over search iteration. The number of quantized models keeps increasing in the search space because we find more and more architecture in the right cluster,

[1]https://docs.nvidia.com/deeplearning/tensorrt/pytorch-quantization-toolkit/docs/userguide.html

i.e., not sensitive to the quantization. The increase slows down due to the population's increasing number of quantized models. The number of quantized models keeps increasing and saturates at 61% of the population size. This result validates that even after 250 iterations, we find non-quantized architectures close to quantized architectures in terms of latency and accuracy.

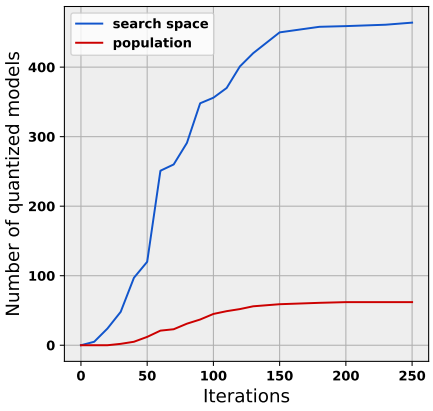

Figure 4.5: Number of quantized architectures in the search space and population over iterations.

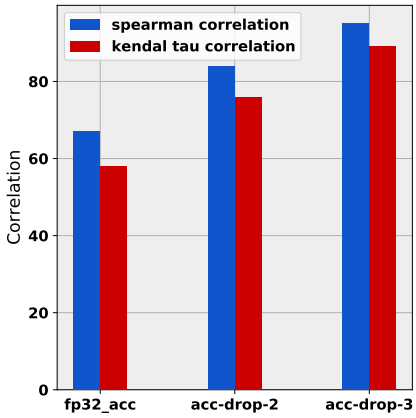

Figure 4.6: Ranking correlation of accuracy proxies. *acc-drop-x* refers to the accuracy proxy with x clusters.

Accuracy estimation analysis
During our search algorithm, we evaluate the accuracy of the quantized models by estimating the fp32 accuracy using a surrogate model [5] and subtracting the AQD of the architecture's cluster; we call this evaluation *accuracy proxy*. We analyze the ranking correlation between the quantized accuracy and different proxies; namely, the fp32 accuracy predicted by the surrogate model, the accuracy proxy obtained using 2 clusters, and the accuracy proxy obtained using 3 clusters. Figure 4.6 shows the results of this test. Hence, the end-to-end results described below are executed using three clusters.

Model	top-1 Accuracy	Parameters (M)	Latency (ms)	Energy (mJ)	Search time (GPU hours)
MobileNetV2	72.00	3.40	51.20	74.24	-
MobileNetV3 - quantized	73.80	5.40	44.00	58.61	-
ProxylessNAS	72.60	5.70	18.52	25.89	200
APQ - B	74.20	4.50	8.75	12.92	2400+0.5N
APQ - C	75.2	4.25	8.16	13.29	2400+0.5N
CaW-Net-A(Ours)	61.90	3.60	1.50	4.68	0.8+2N
CaW-Net-B (Ours)	78.22	4.52	8.33	11.86	0.8+2N
CaW-Net-C (Ours)	**75.22**	**4.18**	**6.30**	**10.43**	**0.8+2N**

Table 4.2: Comparison with state-of-the-art efficient models on ImageNet. N is the number of training for NAS on a new platform.

End-to-end Search The final Pareto front results are represented in figure 4.7. We compare our Pareto front approximation to the result of two search strategies: random

search on the non-quantized search space only (RS-w/o quantized) and random search on the larger mixed search space (RS-with quantized). In the pre-trained model search space, we can not compute the optimal Pareto front using a brute force method because the search space is too big. Nevertheless, figure 4.7 shows that we accurately approximate the Pareto front. We select from the Pareto front the architecture that best represents the trade-off between latency and accuracy.

Table 4.2 shows the comparison of our final architecture against state-of-the-art architectures obtained using different search strategies.

CaW-Nets are the models obtained using CaW-NAS. A and B represent respectively the models with minimum and maximum accuracy within the Pareto front. We use the same notation as APQ [140], where N denotes the number of training needed to perform the NAS strategy for a new platform. Note that N in our strategy decreases over the search iteration due to the increasing number of quantized architectures in the population.

From the Pareto front approximation, we manually extracted three architectures: the architecture with the lowest latency *CaW-Net-A*, the architecture with the highest accuracy *CaW-Net-B*, and *CaW-Net-C* an architecture with comparable accuracy to APQ-C. We can observe that our approach can find architectures with a good trade-off in terms of accuracy, latency, and energy consumption, with a considerably low number of parameters.

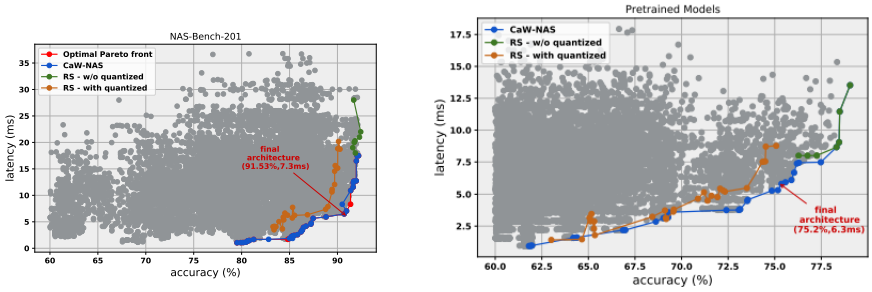

Figure 4.7: Pareto front approximation results. Top figure: NAS-Bench-201 for CIFAR-10, Bottom figure: Pretrained Models for ImageNet

4.3 Grassroots Operator Search for Model Edge Adaptation

The definition of the search space is a critical step in NAS, as it determines the range of possible architectures and can significantly impact their performance. The size of the search space matters. A large search space hinders the exploration but diversifies the results. In contrast, a small search space restricts architectural diversity. Currently, there are three primary approaches to defining the search space in HW-NAS [1]:

1. *Cell-based search space*, which involves searching for a repeated cell, also called *block*, within a pre-defined macro-architecture. The cell is defined by a list of operators, such as convolution and batch normalization, and an adjacency matrix that defines the connections between the operators. NAS-Bench-101 [264] is a common NAS benchmark designed using this definition.

2. *Hierarchical search space* [265] that extends the cell-based approach by selecting the operators composing the cell, defining the cell-level connections, and merging multiple cells.

3. *Supernetwork search space* [166], in which each architecture is represented as a subgraph within a larger, more complex network called the *supernetwork*. the

weights of the supernetwork are typically shared among all subgraphs, allowing the subgraphs to share computation and enabling efficient exploration of the search space. The supernetwork is then called an over-parameterized network. The subgraphs can differ in terms of their connectivity, layer types, layer sizes, and other architectural hyperparameters.

A prevalent limitation of such definitions is the bias introduced by the dependence on human-designed architectures, which restricts the search algorithms from exploring novel and innovative operations and architectures. This bias towards previously handcrafted architectures hinders the discovery of more efficient and effective models for specific tasks. Consequently, there is a need to develop novel methodologies that can help discover more optimized architectures and operations that can perform well on various devices and scenarios without relying on pre-existing models. Such methodologies would be the holy grail of NAS, as they would enable the creation of truly novel architectures that can push the limits of deep learning performance even further.

One solution would be to define a completely random search space where the architecture and operations are generated from scratch and then evaluated based on their performance. However, given the vast search space, such approaches require a massive amount of computational resources and are often infeasible for practical use. AutoML-Zero [77], for example, presents a strategy capable of defining the architecture and the training procedure from standard mathematical operations using reinforcement learning. This approach breaks the innovation barrier for NAS but at a significant time complexity price. Due to this highly complex search, AutoML-Zero only achieves linear regression on the MNIST dataset, which is impractical for complex and real-world datasets.

Indeed, selecting the right set of operators for a specific task is crucial, but the actual implementation of the operator can also greatly impact the hardware efficiency of the DL model. In order to overcome this challenge, recent works have focused on using DL compilers [58, 60] that can automatically select the most efficient implementation and optimization for a given hardware. These compilers use techniques such as code generation and optimization, which allow for the automatic translation of high-level DL operators to hardware-specific low-level code. By doing so, they can greatly improve the efficiency of DL models on different hardware devices, including edge devices. The use of deep learning compilers highlights the importance of not only selecting the right operator but also optimizing its implementation to achieve the best possible hardware performance. MCUNet [266] combines the use of NAS and DL compiler, called *TinyEngine*, to efficiently look for the best architecture as well as its best implementation in an iterative manner. However, their search space includes a set of standard DL operators. Current operators' implementations are designed for resource-expensive hardware platforms and do not conform to edge constraints.

This section presents a search algorithm that adapts the architecture to edge devices without previous human experience. To overcome the time complexity of AutoML-Zero, we apply our search algorithm on a specific layer at each iteration. Specifically, our method, in the first step, analyzes each layer's latency and memory occupancy distributions in a given model. In the second step, the most inefficient layer is optimized. Costly operators in this layer are replaced by efficient operators. We express an operator as a set of mathematical instructions that capture its behavior. For example, standardization is expressed by subtracting the mean of the input over a mini-batch and dividing it by the standard deviation of that input.

The mathematical instructions are then used as a basis for searching and selecting efficient replacement operators that maintain the accuracy of the original model while reducing computational complexity. We consider a model as a set of layers such as convolution. Each layer corresponds to a sequence of operators implemented by a graph of mathematical instructions. Table 4.3 gives the list of mathematical instructions considered in this work.

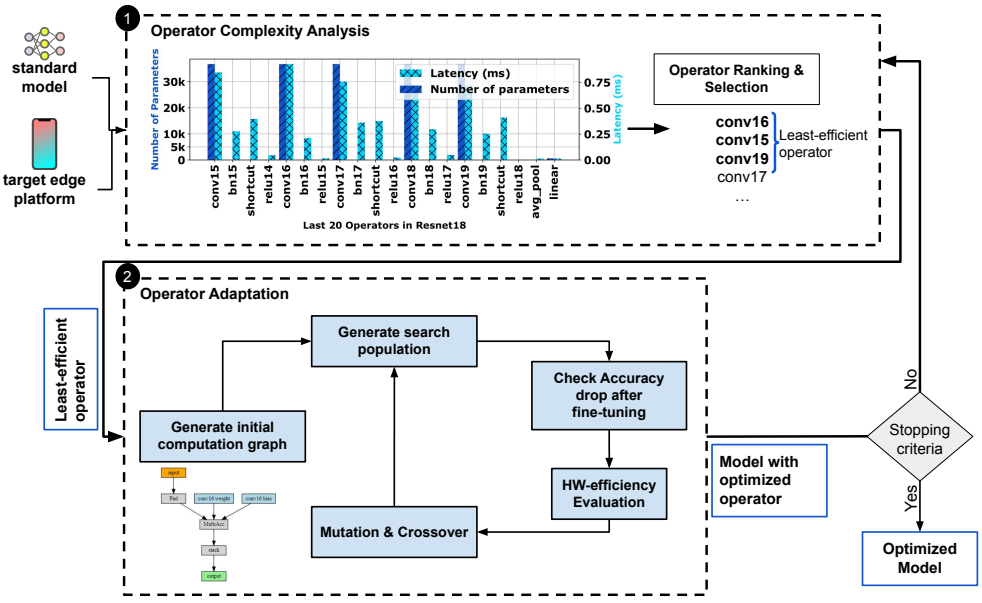

Figure 4.8: Overview of the operator replacement methodology.

We repeat these two steps until we find an architecture suited for the targeted edge device without an accuracy drop. Our technique aims at breaking the time-consuming barrier of non-restrictive search spaces while searching for new and innovative architectural designs.

We summarize the contributions proposed by GOS as follows.

- We present a new adaptation methodology via operator replacement. We replace the most hardware-inefficient layer iteratively by building a new operator from scratch with minimal human bias.
- We develop an optimized multi-objective evolutionary search algorithm that effectively selects the appropriate operator for deploying an efficient architecture on the targeted device. By doing so, we enable the deployment of deep learning models on edge devices with improved efficiency and without sacrificing accuracy.

Our methodology has been validated with different types of architectures: Convolutional neural networks (ConvNets) and Vision transformers (ViT). In particular, we identified a novel convolution implementation suitable for Raspberry Pi, which is a significant contribution to the field of edge computing. Additionally, we applied our methodology for Pulse Rate estimation with PPG sensors and achieved a state-of-the-art result. Overall, our approach consistently improves the model's hardware efficiency with an average of 2x speedup without any loss in the model's accuracy. These results demonstrate the effectiveness and versatility of our methodology for optimizing deep learning models for different hardware platforms and applications.

4.3.1 Proposed Approach

Figure 4.8 shows the overall structure of our methodology. Given a model, denoted as m, our goal is to adapt it to a targeted edge platform. In this methodology, we define an operator as a set of operations applied in a layer. The operator can be a single layer, as defined in common DL frameworks, such as a convolution, or a fused layer such as ReLU-BN [267]. The process goes through two stages:

1. *Operator Complexity Analysis:* First, the process extracts the least efficient operator by running N_i inference on the edge device. The inefficiency is computed with different objectives, such as latency and the number of parameters. The number of parameters reflects the size of the operator. Additional criteria such as energy consumption may be added. Among the list of operators in m, the least efficient operator is selected based on algorithm 4. If the model is not deployable on the target platform, i.e., the size of the network exceeds the memory capacity, we select the operator with the highest number of parameters from the table *num_param* in algorithm 4. Otherwise, we rank the architectures with latency and number of parameters in descending order and select the first operator. Our strategy of ranking is as follows: if the architecture is deployable on the target device, the number of parameters is a less important objective, we rank the operators based on the latency and if two operators are of close latencies then we consider the number of parameters. This behavior is checked at each iteration. If more criteria are considered, then the ranking should be multi-objective [268]. This operator corresponds to the slowest operator that has the highest number of parameters possible. If an operator is selected, it cannot be selected for another optimization iteration. In a CNN, it is common knowledge that the least-efficient operator is the convolution. However, according to its input and output shape, the convolution may be optimized differently. To efficiently select the operator to be replaced, we define N_o as the maximum number of similar operators and select the top operators each time. For example, if the N_o least efficient operators are all convolutions, we will replace them all with the same generated optimized operator.

2. *Operator Adaptation:* Then, we adapt the selected operator by searching for a variation that can keep the same input and output shapes but optimizes the computations. This phase is done with an evolutionary search on a set of mathematical operations. Section 4.3.1 and section 4.3.2 describe the search space and methodology respectively. During the search, only the parameters of the adapted operator are fine-tuned.

The two steps are repeated until satisfactory hardware efficiency is reached or a maximum number of layers have been replaced.

Operator Search Space

Unlike previous HW-NAS search spaces that are based on pre-defined operator sets, our search space is defined with a set of mathematical operations. The operator is represented with a computation graph. The computation graph is a directed acyclic graph (DAG) with N nodes and E edges. The nodes correspond to the operations such as matrix multiplication, square root, and element-wise addition. The edges describe the inputs and outputs of each node. Figure 4.8 (step 2) shows an example of such a graph.

Each node in the context can be classified into one of the following three types:

- Instruction: this node corresponds to any mathematical instruction in table 4.3.
- Input: this node corresponds to the input feature maps or weights that are given as operands to the instruction node.
- Constant: this introduces hyperparameters fixed in the mathematical instruction equation. These constants can be tuned and mutated during the search.

We constrain the generated computation graphs with $1 < N <= 20$ and $1 < E <= 25$. These values have been fixed by analyzing standard models' operators. During the generation, the input node is fixed, and its shape is defined by the output of the previous operation in m. The output node's shape is also known as it is

Algorithm 4 Least-efficient Operator Selection

Input: Model m, Number of inference N_i
$is\_deployable \leftarrow deploy(m)$
if not is_deployable **then**
  for each o in m **do**
    # for each operator o get its number of parameters
    $num\_param[o] \leftarrow number\_of\_params(o)$
  end for
  return argmax(num_param, N_o)
  # return the operator with highest value in num_param and its number of occurrences N_o
end if
for each o in m **do**
  $latency[o] \leftarrow average\_latency(o, N_i)$ # compute the mean latency of each operator o for N_i inferences
  $num\_param[o] \leftarrow number\_of\_params(o)$
end for
return Top N_o similar operators
return the operator with highest value in num_param and its number of occurrences N_o
here we consider first the latency and then the number of parameters in the ranking

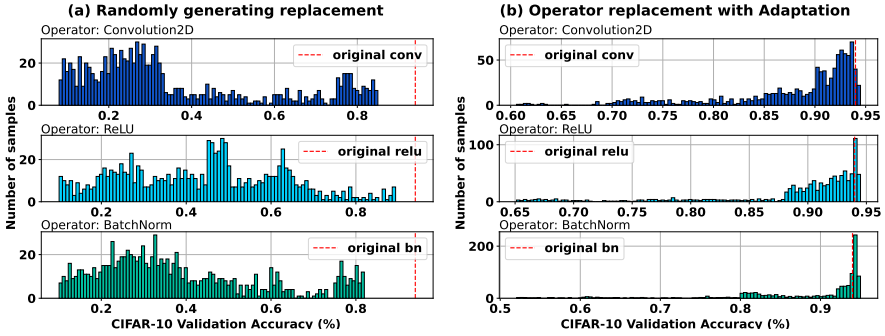

Figure 4.9: CIFAR-10 accuracy histograms of 1k architectures randomly generated (a) and adapted from the original operator (b).

constrained by the input shape of the next operator in m. To ensure a valid network, we optionally add a reshape operation at the end of the computation graph to keep the same output shape as expected by the next operator in the given model. Nodes that can't be reached from the input or that do not have a path to the output are considered unused and therefore pruned from the computation graph.

Table 4.3 shows the basic operations in the search space, including arithmetic, linear algebra, probability, and aggregation operations. The aggregation operations enable to merge between the output of multiple nodes. We include code optimizations such as loop tiling and unrolling as special aggregation functions that are called to optimize the generated operator's code. Note that this is a general application of these optimizations that can be hardware-specifically defined by a compiler.

Note that each operator has a list of hyperparameters dedicated to it. These hyperparameters are illustrated in the equations as constants in table 4.3.

Example of Operator Computation Graph In this paragraph, we explain how the convolution 2D is turned into a computation graph. In its simplest form, the convolution 2D can be formulated as in equation 4.1, where N is the batch size, C

Table 4.3: List of mathematical instructions defining the search space

Category	Instruction	Equation
Linear Algebra	Matrix multiplication	$\mathbf{C} = \mathbf{AB}$
	Matrix addition and subtraction	$\mathbf{C} = \mathbf{A} + \mathbf{B}$ or $\mathbf{C} = \mathbf{A} - \mathbf{B}$
	Vector multiplication	$\mathbf{c} = \mathbf{Ab}$
	Matrix inversion	$\mathbf{A}^{-1}$
	Dot product	$\mathbf{a}^\top \mathbf{b}$
	Determinant	$\det(\mathbf{A})$
	Trace	$\mathrm{tr}(\mathbf{A})$
	Eigenvalues and eigenvectors	$\mathbf{Av} = \lambda \mathbf{v}$
	Singular value decomposition (SVD)	$\mathbf{A} = \mathbf{U\Sigma V}^\top$
	QR decomposition	$\mathbf{A} = \mathbf{QR}$
	Cholesky decomposition	$\mathbf{A} = \mathbf{LL}^\top$
	Matrix pseudoinverse	$\mathbf{A}^\dagger$
	Matrix rank	$\mathrm{rank}(\mathbf{A})$
	Hadamard product	$\mathbf{C} = \mathbf{A} \odot \mathbf{B}$
	Kronecker product	$\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$
	Outer product	$\mathbf{C} = \mathbf{ab}^\top$
	Vector norm	$\|\mathbf{x}\|$
	Matrix norm	$\|\mathbf{A}\|$
	Frobenius norm	$\|\mathbf{A}\|_F$
	Identity matrix	$\mathbf{I}$
	Zero matrix	$\mathbf{0}$
Calculus	Gradients	$\nabla_\theta L(\theta)$
	Partial derivatives	$\frac{\partial f}{\partial x}$
	Chain rule	$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$
Activation Functions	Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$
	ReLU	$\mathrm{ReLU}(x) = \max(0, x)$
	Tanh	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
	Softmax	$\mathrm{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$
Convolution	cross-correlation	$(f * g)(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} f(x-i, y-j) g(i, j)$
Pooling	Max pooling	$\mathrm{maxpool}(x_{i:i+s,j:j+s}) = \max_{m=1}^{s} \max_{n=1}^{s} x_{i+m,j+n}$
	Average pooling	$\mathrm{avgpool}(x_{i:i+s,j:j+s}) = \frac{1}{s^2} \sum_{m=1}^{s} \sum_{n=1}^{s} x_{i+m,j+n}$
Probability and Statistics	Probability distributions	$p(x)$
	Bayesian inference	$p(\theta\|x) = \frac{p(x\|\theta)p(\theta)}{p(x)}$
Aggregation Function	Summation	$\sum_{i=1}^{n} x_i$
	Mean	$\frac{1}{n} \sum_{i=1}^{n} x_i$
	Maximum	$\max(x_1, x_2, ..., x_n)$
	Minimum	$\min(x_1, x_2, ..., x_n)$
	Square Root	$\sqrt{x}$
	Concatenation	$[A\ B]$
	Weighted Mean	$\frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$

denotes the number of channels, H is the height of input planes in pixels, and W is the width in pixels. *in* and *out* refer to the input and output respectively. $*$ in the equation denotes the cross-correlation operation.

$$conv2D(N, C_{out}) = bias(C_{out}) + \sum_{k=0}^{k=C_{in}-1} weight(C_{out}, k) * input(N, k) \tag{4.1}$$

The convolution first splits the input into weight-shaped chunks. We compute the multiply-accumulate of each of these chunks with the weights (i.e., kernels), using the cross-correlation instruction. We then sum up all the multiplied values over the input channels C_{in}. Finally, we add the bias to each output channel C_{out}.

To create the computation graph, we divide the equation into instructions found in table 4.3. Figure 4.10 shows the complete convolution 2D graph with a 2-dimensional input and 2 kernels. To have a compact and simple graph, we include the constant nodes inside the instruction node as a list of hyperparameters. In the rest of the paper and for the sake of clarity, we use high-level operator names such as Linear for the matrix multiplication between weight and input matrices.

In this search space, we perform small-scale experiments with random sampling to understand its behaviors. The purpose is to measure the sparsity of the search space and to determine the number of valid and accurate operations generated dur-

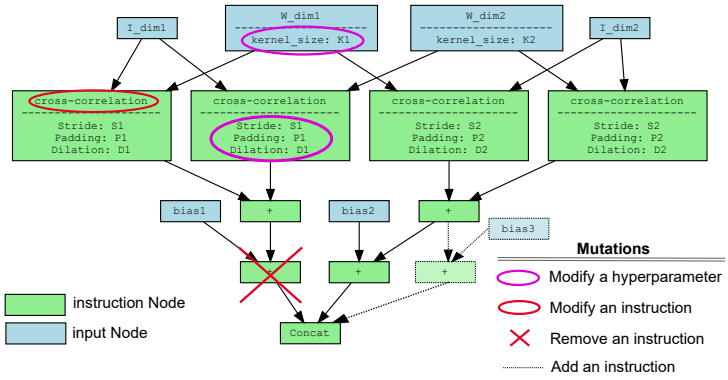

Figure 4.10: Detailed computation graph of the standard convolution 2D including the possible mutations applied to it.

ing the exploration. In this experiment, we replace all similar operators at once. For example, we replace all convolutions in the model with a generated replacement. Figure 4.9 (a) shows the results of 1000 randomly generated operator replacements for three operators: Conv2D, max-pooling, and batch normalization, in resnet-18 [269]. Random generation, inspired by EvoNorm [79], starts from the input node and sequentially selects an operation from the search space. In all the cases, the ImageNet accuracy drops significantly for most of the replacements, which reflects the high sparsity of our search space. In figure 4.9 (b), rather than randomly generating the operator replacement, we start with the original operations but adapt one operation in the computation graph. The adaptation is performed while being aware to keep the same arity and type of arguments for each operation. With adaptation, the results are much closer to the original accuracy of the model but the complexity is modified.

4.3.2 Search Algorithm

Given an operator computation graph, the search algorithm aims at finding a variant that preserves the accuracy of the model with reducing complexity. We rely on an evolutionary algorithm for this purpose. The evolutionary algorithm allows us to handle the sparse search space by exploring a population of valid computation graphs. The computation graph is considered valid if it maintains the shapes of the input and output data and if there exists a path from every intermediate node, including the input node, to the output node. Besides, mutation and crossover provide an efficient way to generate complex adaptations. We use tournament selection which ensures that the best individuals have a higher chance of being selected, while still allowing for some diversity in the population. This helps to prevent premature convergence and promotes the discovery of novel solutions in our large search space.

Mutations The mutation operations involve modifying the computation graph. Figure 4.10 summarizes the possible mutations applied on the conv2D computation graph. Each instruction node in the computation graph is typed with the corresponding type in table 4.3. The most important mutation is modifying any intermediate node with a possible operation. For each operation, we associate a list of possible replacements. The replacement satisfies two constraints: (1) having the same argument's type and arity, (2) the output shape is equal or can be converted to the original output shape by adding a reshape operation. The replacement operation from the list is selected uniformly at random. We also allow for a modification of the aggregation function, and an addition or deletion of a node. When adding or removing a node,

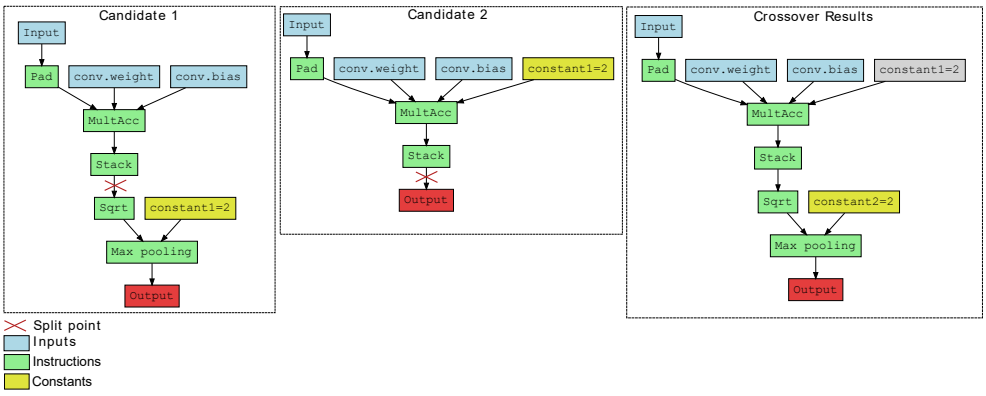

Figure 4.11: Illustration of the cross-over operation.

we make sure that a path from the input to the output is still possible and that no unused node appears in the graph.

The mutations also include modifying the hyperparameter of the operator. The hyperparameters are properties associated with a vertex in the computation graph. For each instruction, a list of possible hyperparameters; i.e., constants, is available. For each hyperparameter, we constrain the ranges with specified values obtained from the literature. For example, the output channel size of a convolution may change. This mutation may reduce the accuracy of the model. If this is the case, the operator is invalidated and is not considered in the novel population.

Crossover In general, the crossover is not applied to NAS algorithms. When we consider high-level operators, it is rarely the case to find a splitting point where the shapes fit. However, in our case, the crossover is beneficial and allows more flexibility. Algorithm 5 and figure 4.11 detail the crossover procedure. We perform a crossover between two computation graphs in our population. Because all the variants start from the same point, we have more chances to find a split point. We perform a pre-order traversal of the two computation graphs and store all the possible split points. We randomly select a split point between each pair of computation graphs and generate offspring.

Multi-objective Fitness Function The evaluation is specific to the given model and task. We do not generalize the resulting operation to multiple standard models because our goal is to adapt the network for a given hardware platform in a practical time. This allows a more flexible and multi-objective fitness function.

The fitness function evaluates the performance of the adapted operator, formulated in equation 4.2. In our methodology, we consider hardware efficiency with multiple objectives. Our definition considers latency and the number of parameters. But one can add other objectives such as energy consumption or memory occupancy. We rely on the crowding distance [270] to minimize multiple objectives under an accuracy constraint. The crowding distance is calculated for each solution in a Pareto front and is based on the distances between neighboring solutions in the objective space. The solutions with larger crowding distances are preferred in the selection process, as they represent areas of the objective space with lower solution density, and hence are more diverse and representative of the Pareto front.

During the search, we want to maximize the hardware efficiency of the adapted operator while keeping the difference between the loss of the original model m and the model with the adapted operator, denoted as $m_{adapted}$, minimal. We add a small value, ϵ, to ensure exploration. We fine-tune the network after adapting the operator for a few epochs. This fine-tuning is done with all the other operator's weights frozen.

Algorithm 5 Crossover procedure

```
Input: Two computation graphs of two operators (o1 and o2)
split_points = []
Stack s = Stack()
Push (o1, o2) to s
# pre-order traversal of both computation graphs
while s not empty do
  Pop a node pair (o1, o2) from the top of the stack
  while o1 not empty and o2 not empty do
    Pop a node pair (o1, o2) from the top of the stack
  end while
  if shape(o1.output) == shape(o2.input) then
    Add (o1, o2) to split_points  # add to possible split point
  end if
  for child of o1 do
    Add (child, o2) to stack
  end for
  for child of o2 do
    Add (o1, child) to stack
  end for
  Uniformly select (o1, o2) from split_points  # randomly select a split points
  between all the possibilities
  Perform a merge illustrated in Figure 4.11
end while
```

The operator's latency is computed with the difference between the original model's latency and the latency of the adapted model. The number of parameters can be reduced or increased by adding weight input to the computation graph.

$$\begin{aligned} &Min_o(LAT(o), PARAM(o)) \\ &\text{subject to } ACC(m_{adapted}) > ACC(m) - \epsilon \end{aligned} \tag{4.2}$$

4.3.3 Evaluation Methodology

We first conducted our experiments on two edge devices: Raspberry Pi 3 Model B and Redmi Note 7S mobile phone. The Raspberry Pi 3 Model B is equipped with a Broadcom BCM2837 SoC with a 1.2 GHz quad-core ARM Cortex-A53 CPU, and 1GB RAM, and runs the Raspbian operating system. The Redmi Note 7S mobile phone is equipped with a Qualcomm Snapdragon 845 SoC with an octa-core CPU and 8GB RAM, running the Android 10 operating system.

To evaluate the performance of our proposed method, we used three popular deep learning models: ResNet18 [269], InceptionV3 [271], and MobileNetV2 [272]. We implemented our approach using Python 3.7 and the PyTorch 1.8.1 deep learning framework. All three architectures were initially trained for Imagenet. The experiment goal is to adapt them for edge devices by changing the most inefficient operators. We measured the accuracy of each model on the validation set and recorded each model's latency and energy consumption during inference. We averaged these numbers for 100 inferences to correctly estimate hardware efficiency. The latency and energy consumption are measured with an inference batch size of 1. For fine-tuning, we use SGD with a mini-batch size of 128. The learning rate is set to 0.003. We use a weight decay of 0.0001 and a momentum of 0.9.

The search is set to do 50 iterations per operator replacement. The stopping criterion is the modification of at least 10 layers in the model. The probability of mutation is set to 0.8 and the cross-over probability to 0.6. We use an epsilon of 1%,

i.e., assume that a 1% drop in accuracy is acceptable. The epsilon should be tailored to the dataset and task at hand. Empirical tuning was done to select these values.

Due to the on-search fine-tuning and hardware efficiency computation on-device, our search takes about 1h04min. This time is highly practical as this adaptation is only done once.

Search Setup The search is achieved on a much more compute-intensive setup. Our search was conducted using an NVIDIA GPU 3070, a high-performance graphics processing unit known for its advanced parallel computing capabilities. The GPU was connected to a powerful workstation equipped with an Intel Core i9 processor and 32 gigabytes of RAM, ensuring sufficient computational resources for the search process.

4.3.4 Optimizing an architecture for Edge Devices

Table 4.4 presents the overall hardware efficiency improvement achieved by applying GOS to the evaluated models on both edge devices. Our operator replacement method consistently outperformed the original models with an average speedup of 3.17. Notably, our search was able to find a variant that improved the accuracy of ResNet models by 6.13% and 5.34% for Raspberry Pi and Redmi Note 7S, respectively.

Interestingly, InceptionV3 was found to be unsuitable for deployment on Raspberry Pi due to its large network size. To tackle this issue, our search began optimizing by selecting operators that use the largest amount of parameters, which led to a reduction in the number of parameters and enabled the discovery of a deployable variant.

Although our search space does not directly optimize energy consumption, we observed that our models presented lower energy consumption due to the reduction in the number of parameters and operations.

Furthermore, GOS was able to find a variant of MobileNetV3 that is 2.2x faster with only a minor accuracy drop of 0.4%, even though the original model was already optimized for mobile devices. Overall, our search consistently outperformed the original models, indicating the effectiveness of GOS in achieving hardware efficiency improvements.

In comparing our strategy to other HW-NAS approaches, namely Once-for-All [169] and FBNetV3 [244], we found that our operator replacement method yielded superior results. Our approach consistently outperformed Once-for-All and FBNetV3, showcasing an average speedup of 1.26. This performance advantage highlights the effectiveness of our method in optimizing neural architectures specifically for hardware constraints and further solidifies the value of GOS in achieving superior hardware efficiency improvements. Note that our method can also be used as a specialization phase after the use of these high-level NAS.

Analysis of Resulting operations In this paragraph, we discuss the novel operators that were generated through our operator search method and the improvements they bring to the models. Table 4.5 presents the novel equations for the most efficient operators that replaced the standard convolution 2D, batch normalization, and activation functions. Table 4.6 summarizes the notations. Our discussion is focused on each device separately.

In general, the last convolution 2D operators of the models are the most inefficient ones. Therefore, in all the models, we automatically optimized these operators using GOS. For the Raspberry Pi device, we modified these operators by adding a dilation rate to the convolutions, similar to dilated convolutions [273]. However, in our operator, the dilation rate is applied within the filter matrix itself, by adding 1, 2, or 3 zeroed columns between different columns of the filter matrix. This modification

Table 4.4: Performance comparison of original models and adapted models on Raspberry Pi 3 and Redmi Note 7S

Edge Device	Model	Variant	# Parameters	Top-1 Accuracy (%)	Latency (ms)	Energy (J)	Speedup
Raspberry Pi	**Resnet18**	Original	11M	69.3	382.54	1320	5.76
		GOS	9.3M	75.43	66.32	220	
	Inceptionv3	Original	25M	78.2	-	-	-
		GOS	7.2M	79.47	101.3	438.3	
	MobileNetV3	Original	2.9M	75.2	94.32	348	2.82
		GOS	2.9M	74.32	33.44	253	
	FBNetV3 [244]	Original	5.3M	79.1	25.4	238	1.52
		GOS	5.1M	83.4	16.7	187	
	OFA [169]	Original	4.9M	74.2	22.3	211	1.16
		GOS	3.2M	79.3	19.2	204	
Redmi Note 7S	**Resnet18**	Original	11M	69.3	93.43	119.4	4.51
		GOS	11.5M	74.64	20.7	78.8	
	Inceptionv3	Original	25M	78.2	83.5	132.6	3.72
		GOS	23.4M	77.9	22.4	104.5	
	MobileNetV3	Original	2.9M	75.2	76.3	76.54	2.23
		GOS	2.6M	74.8	34.2	78.43	
	FBNetV3 [244]	Original	5.3M	79.1	21.6	67.9	1.18
		GOS	4.8M	81.4	18.3	87.3	
	OFA [169]	Original	4.6M	76.5	34.6	56.42	1.21
		GOS	3.7M	83.4	28.5	58.3	

enables the operator to have a larger receptive field without increasing its size, which can be helpful in capturing features at different scales in an image. This operator is particularly efficient in Raspberry Pi, which has limited computational resources, as it reduces the number of operations needed to process the input.

On the other hand, for the Redmi Note 7S, the model's last convolution 2D operators were modified to a depthwise convolution [274]. Similarly to Raspberry Pi, the dilated rate is applied here as well. The use of dilated filters and depthwise convolution allowed for an increase in hardware efficiency. It is worth noting that we did not start with a depthwise separable convolution, except for MobileNetV3. Instead, our operator search method converges to similar operations. In addition, for resnet18, the search applied a pooling layer at the end of the convolutions. This is done to further reduce the feature map size and enhance the latency. Interestingly, this did not impact negatively the accuracy. However a similar operator was tested on InceptionV3 and MobileNetV3 and a 5%, 6.7% drop in accuracy was seen.

The first convolutions are particularly different from the last ones because of the input shape. In the first convolutions, the channel size is smaller, while the height and width of the feature maps are large. The opposite happens at the end of the model. The first convolutions, even in MobileNetV3, were turned into standard convolutions. The search only changed the hyperparameters of these operators, using 5x5 kernels for some and adding different padding.

The generated batch normalization operation uses a polynomial regression to regress the batch normalization values after the standardization. By incorporating the polynomial regression into the batch normalization equation, this method can improve the accuracy of the normalization while maintaining a fast computation time.

The search algorithm almost never changes the activation functions, as they are usually fast and already efficient. However, we forced the model to change the activation and look for a more efficient version. The resulting equation is shown in table 4.5. The equation is a leaky version of ReLU. When removing the activation functions

from the list of instructions, the search failed to find a differentiable equation.

Table 4.5: Efficient Operators Equations for Raspberry Pi and Redmi Note 7S

Device	Convolution2D	Batch Normalization	Activation
Raspberry Pi	$\mathbf{y} = \sum_{i,j} \mathbf{w}i, jxi + d_{i,j}, j + d_{i,j}$	$\frac{x-\mu}{\sigma+\epsilon} \times \alpha(x-\mu)^3 + \beta(x-\mu) + \gamma$	$max(0.01x, x)$
Redmi Note 7S	$\sum_{j=1}^{C_{in}} \sum_{r=1}^{H_k} \sum_{c=1}^{W_k} W_{i,j,r,c} \cdot I_{i+(r-1)d^r j, j+(c-1)d^c j, k}$	$\frac{x-\mu}{\sigma+\epsilon} \times \alpha(x-\mu)^3 + \beta(x-\mu) + \gamma$	$max(0.03x, x)$

Table 4.6: Notation Summary

Symbol	Description
C_{in}	Number of input channels
C_{out}	Number of output channels
H_k	Height of the kernel
W_k	Width of the kernel
K_w	Number of weights in the kernel
d_j	Dilation rate for input channel j
$d^r j$	Dilation rate for input channel j in the row direction
$d^c j$	Dilation rate for input channel j in the column direction
I	Input tensor
W	Weight tensor
γ	Scaling parameter in batch normalization
β	Bias parameter in batch normalization
a_0, a_1, a_2	Polynomial coefficients in fast batch normalization
x	Input to fast batch normalization
σ	Standard deviation in batch normalization
ϵ	Small constant for numerical stability

Effect of number of instructions In the previous experiments, we fixed the maximum number of instructions per operator to 20. Here, we justify this value and analyze the effect of changing the number of instructions on operator generation, using the same search space and fitness evaluation as described in Section 4.3.3.

We varied the number of instructions used to define each operator ranging from 5 to 40 instructions with a step of 5 and compared the resulting architectures' performance. Specifically, we evaluated the accuracy and inference time of the architectures on the Imagenet dataset using the same hardware setup as in the previous experiments. Figure 4.12 shows the results.

The results showed that increasing the number of instructions used to define each operator generally leads to an improvement in the architectures' performance. This improvement stabilizes after 20 in which we obtain the results shown in table 4.4. Below 20, the operators become badly implemented and the accuracy drops. Above 25, the instruction set is too large and the operators apply redundant instructions which increases the latency. In addition, the search time highlighted in green increases with the increase of the maximum number of instructions per operator. This is due to the increased latency and fine-tuning time induced by more complex and redundant operators.

4.3.5 Use Case: Pulse Rate Estimation

The ability to estimate pulse rate continuously is a critical feature in heart attack detection. Estimating pulse rate is essential for measuring workout intensity during exercise and resting heart rate, which is often used to determine cardiovascular fitness. Using mobile wearable devices provides valuable insights into a wearer's health. Due to the limited hardware resources, the model needs to be small and fast to provide

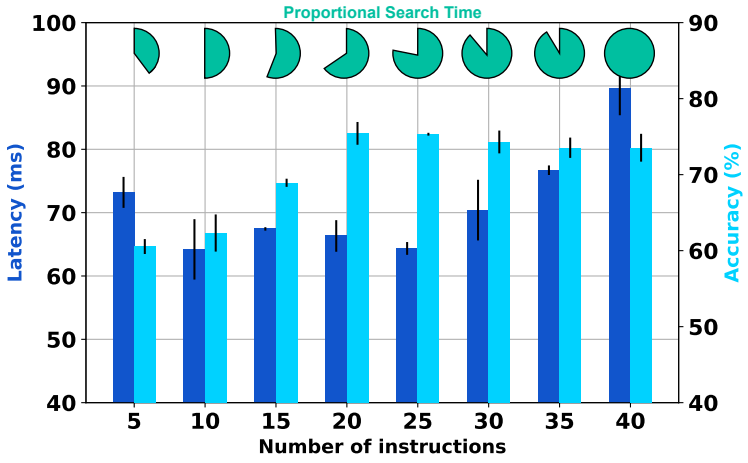

Figure 4.12: Tuning of the maximum number of instructions per operator while searching for resnet18 GOS variant on Raspberry Pi.

real-time results. This task also requires efficient processing of sensor data, which is a critical aspect of hardware-aware NAS.

Background on Pulse Rate Estimation

Pulse rate estimation [275] has been the subject of extensive research in the field of physiological monitoring. Various approaches utilize photoplethysmography (PPG) signals captured from wearable devices, such as wrist-worn sensors or fingertip sensors, to estimate the pulse rate. Among the state-opf-the-art pulse rate estimation models, we find: DeepHeart [276], CNN-LSTM [277], and NAS-PPG [278].

DeepHeart uses an ensemble of denoising convolutional neural networks (DCNNs) to denoise contaminated PPG signals that are then passed through spectrum-analysis-based calibration to estimate the final pulse rate.

CNN-LSTM uses a hybrid convolutional and LSTM neural network. The proposed model comprised two convolutional layers, two LSTM layers, one concatenation layer, and three fully connected layers including a softmax.

NAS-PPG is the first NAS applied to pulse rate estimation. Their search space is defined with a convolutional macro-architecture comprising time-distributed convolutions and two final LSTM layers. Thanks to their automatic search, they provide the best performance on Troika dataset [20].

This task serves as an excellent validation use case for our methodology, specifically tailored to edge devices, where the presence of limited computational resources and power constraints presents unique challenges. By effectively optimizing pulse rate estimation models for edge devices, we showcase the practicality and robustness of our approach in overcoming these constraints and meeting the specific requirements of edge environments.

Experiments & Results

In this use case, we focus on estimating the Beats Per Minute (BPM) based on PPG and accelerometer raw data. The same previously used search hyperparameters are applied.

Dataset For this task, we are using the Troika dataset [20]. Troika is a publicly available dataset and contains measurements from three sensors to estimate the heart rate of the wearer: an Electrocardiography (ECG) sensor, a PPG sensor, and an

accelerometer sensor. The dataset was collected in a study where participants were asked to perform a set of activities while wearing the sensors, including running, cycling, and sitting. The dataset contains 12 recordings from 8 participants, aged 18 to 35, with each recording lasting 5 minutes. The ground truth heart rate for each recording was obtained from the ECG sensor, which is considered the most accurate method for measuring heart rate.

Wrist-band device Our latency results were extracted from a Xiaomi Mi Smart Band 6. The wristband is composed of a built-in PPG biosensor and a 3-axis accelerometer sensor which extracts the input to our algorithm. The operating system installed is Android 6.0. And the battery of the system was at maximum when extracting the latencies.

Preprocessing In models designed for pulse rate estimation, preprocessing of the raw data plays a critical role. Preprocessing is used to clean and enhance raw PPG signals before the actual analysis. The dataset was divided into subjects, and each subject included the PPG and accelerometer as time-series data. The PPG and individual accelerometer signals undergo bandpass filtering with a range of 40BPM to 240BPM (0.66Hz and 40Hz respectively) to eliminate irrelevant signals. In the next step, characteristic frequencies are calculated for each signal within 8-second windows, with a 6-second overlap between adjacent windows. The characteristic frequencies for the PPG data are then compared against the dominant characteristic frequencies for each accelerometer axis per time window. It is possible that a dominant PPG frequency may be similar to accelerometer frequencies. To ensure an accurate estimation of pulse rate, we compare up to 10 PPG frequencies that potentially represent the pulse rate to find one that is not similar to accelerometer frequencies. In cases where no alternate frequency is found, we select the PPG frequency with the largest magnitude as the pulse rate.

Models We optimize two models using our methodology:

- *RF_Model*: We first optimize a simple machine-learning model consisting of two blocks. The first block consists of a bandpass filter and a Fourier transform. The PPG signal contains information about the blood flow in the capillaries. This signal is a combination of various frequencies, including the pulse rate. By applying a bandpass filter to the PPG signal, frequencies outside the range of interest are eliminated. The Fourier transform is then applied to the filtered signal to extract the characteristic frequencies that correspond to the pulse rate. This process helps to remove noise and artifacts from the signal and facilitates accurate estimation of the pulse rate. The second block consists of a random forest regressor [279]. While this first model is fast, it is not optimal in terms of performance.
- *PPG_NAS_Model*: PPG_NAS [278] is a dedicated NAS for PPG signal analysis and pulse rate estimation. The authors generated an optimized model for pulse rate estimation. The model consists of a 1D convolution layer followed by 2 LSTM layers and a final fully-connected layer. The architecture is designed to minimize the number of parameters and maximize the accuracy of pulse rate estimation. This is a state-of-the-art model in terms of the accuracy of the regression and hardware efficiency on wristband devices.

Results Table 4.7 shows the overall Average Absolute Error (AAE) results on multiple subjects and under different actions: running ($T1$), cycling ($T2$), and sitting ($T3$), as well as their latency and number of parameters. We additionally compare our results to other state-of-the-art models, namely NAS-PPG [278], CNN-LSTM [277], and DeepHeart [276].

Table 4.7: Results of Average Absolute Error for Pulse Rate estimation on TROIKA Dataset [20]

Action	Subject	RF Optimized (Ours)	PPG_NAS Optimized (Ours)	RF_Model	PPG_NAS [278]	CNN-LSTM [277]	DeepHeart [276]
T1	1	1.43	0.8	5.9	0.95	0.47	1.47
T1	2	2.08	1.33	1.57	1.22	3.88	2.94
T1	3	3.76	0.06	4.24	0.43	1.52	0.47
T1	4	2.08	0.69	8.68	0.69	2.31	1.02
T1	5	1.05	0.83	2.74	0.72	1.72	2.66
T1	6	4.24	0.72	4.49	0.49	1.47	0.75
T1	7	1.51	0.71	4.8	0.99	2.85	3.45
T1	8	2.57	1.3	10.81	0.87	2.18	2.48
T1	9	3.87	1.44	7.41	1.06	4.9	0.54
T1	10	4.49	0.98	11.18	0.64	0.34	0.72
T1	11	3.7	0.87	20.16	1.01	4.46	1.06
T1	12	2.63	0.77	5.37	0.67	1.79	0.73
T2	13	5.24	1.96	5.56	1.62	3.01	4.8
T2	14	4.2	1.84	23.32	1.95	7.6	2.94
T2	15	9.89	1.24	9.92	0.59	1.58	0.11
T3	16	5.22	0.57	5.49	0.61	0.9	1.63
T3	17	1.32	1.14	1.58	1.32	6.1	1.84
T3	18	1.59	0.48	5.98	0.55	0.31	1.64
T3	19	0.2	0.54	0.61	0.47	0.12	0.18
T2	20	4.76	1.92	12.05	1.99	4.37	4.02
T3	21	2.52	1.18	4.65	0.39	0.38	0.06
T3	22	0.83	0.93	4.23	0.83	1.26	2.25
T2	23	2.88	1.54	8.17	1.38	4.26	0.94
All		3.13	1.03	7.34	0.93	2.51	1.68
Latency (ms)		2.38	2.68	1.64	5.6	11.8	13.54
Number of parameters (M)		0.08	0.564	0.02	1.1	3.3	4.4

Over all subjects and actions, our models outperform their state-of-the-art counterparts with a lesser number of parameters and faster execution.

Figure 4.13 shows the final architectures proposed by our Grassroots operator search. The optimized final models of RF_Model_optimized and PPG_NAS_optimized were obtained by modifying their respective base models. RF_Model_optimized underwent two stages of modifications. First, the fast Fourier transform was altered to extract 30 points instead of the 10 peaks in the original model. Additionally, a linear layer was added to act as a smoother filter that selects the 10 most significant peaks. In the second stage, the random forest regressor was replaced with a multilayer perceptron (MLP) using a hyperbolic tangent (tanh) activation function. While the RandomForestRegressor is highly efficient, it reduces the model's accuracy, so the search favored MLP. As for PPG_NAS_optimized, the LSTM layers in the original model were replaced with gated recurrent units (GRUs), which use fewer parameters and do not affect the performance. The convolution layer was also modified to use a dilated-like convolution (as shown in the table), and the final linear function was adjusted to have no activation at the end. These optimizations resulted in more accurate and efficient models for pulse rate estimation.

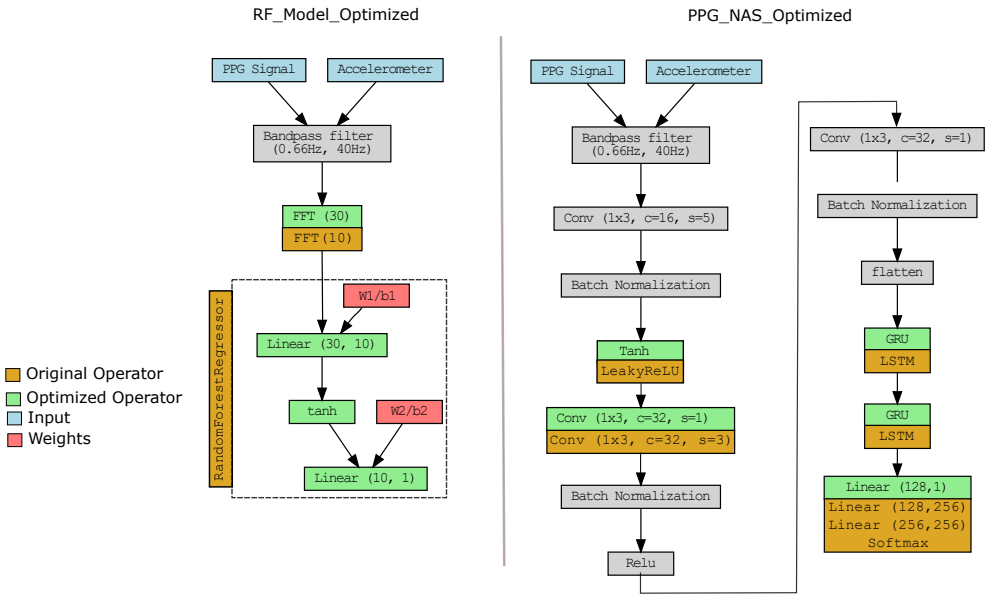

Figure 4.13: Pulse Rate Estimation final Models. We do not display the weights node for PPG_NAS for the sake of clarity.

4.4 Conclusion

In conclusion, this chapter has presented our contribution to the search space of HW-NAS.

We have presented *CaW-NAS*, a dedicated Hardware-aware Neural Architecture Search (HW-NAS) that takes into account the quantization effect during the NAS process. Our method extends the search space with the quantized architectures that minimize the quantization impact on the accuracy. Using our extended search space, we implemented a hardware-aware search strategy and we obtained the Pareto front approximation on a custom search space made of pre-trained models and NAS-Bench-201. On NAS-Bench-201, our approach was able to produce a Pareto front that contains most of the optimal quantized and non-quantized architectures. Our final model improves the state-of-the-art with 1.33 acceleration and an accuracy of 75.22%.

Our Grassroots Operator Search (GOS) is a novel approach for optimizing neural network architectures for resource-constrained devices. GOS leverages the use of mathematical equations to replace common operations such as convolution, batch normalization, and activation functions with more efficient ones, resulting in models that are optimized for low-power devices such as Raspberry Pi and mobile phones. We demonstrated the effectiveness of our approach through experiments on popular architectures, including ResNet18, InceptionV3, and MobileNetV3, achieving significant improvements in inference time and energy consumption compared to the original models. Additionally, we applied GOS to a real-world healthcare problem, namely pulse rate estimation, in which we present a 2x faster network with a 0.12 average error drop. Overall, our results highlight the potential of our approach for creating efficient neural networks for resource-constrained devices.

Part III

Applications of HW-NAS

Chapter 5

AnalogNAS: A Neural Network Design Framework for Accurate Inference with Analog In-Memory Computing

Contents

5.1 Context

Research Question 4

How can hardware-aware NAS methods be adapted and extended to novel hardware platforms, such as analog in-memory computing, and how can these methods be optimized for these platforms?

With the growing demands of real-time DL workloads, today's conventional cloud-based AI deployment approaches do not meet the ever-increasing bandwidth, real-time, and low-latency requirements. Edge computing brings storage and local computations closer to the data sources produced by the sheer amount of IoT objects, without overloading network and cloud resources. As DNN are becoming more memory and compute-intensive, edge AI deployments on resource-constrained devices pose significant challenges. These challenges have driven the need for specialized hardware accelerators for on-device ML and a plethora of tools and solutions targeting the development and deployment of power-efficient edge AI solutions. One such promising technology for edge hardware accelerators is analog-based In-memory Computing (IMC), which is herein referred to as *analog IMC*.

Analog IMC [196] can provide radical improvements in performance and power efficiency, by leveraging the physical properties of memory devices to perform computation and storage at the same physical location. Many types of memory devices, including Flash memory, PCM, and RRAM, can be used for IMC [197]. Most notably, analog IMC can be used to perform MVM operations in $O(1)$ time complexity [198], which is the most dominant operation used for DNN acceleration. In this novel approach, the weights of linear, convolutional, and recurrent DNN layers are mapped to crossbar arrays (tiles) of NVM elements. By exploiting basic Kirchhoff's circuit laws, MVM can be performed by encoding inputs as WL voltages and weights as device conductances. For most computations, this removes the need to pass data back and forth between CPU and memory. This back-and-forth data movement is inherent in traditional digital computing architectures and is often referred to as the *von Neumann bottleneck*. Because there is greatly reduced movement of data, tasks can be performed in a fraction of the time, and with much less energy.

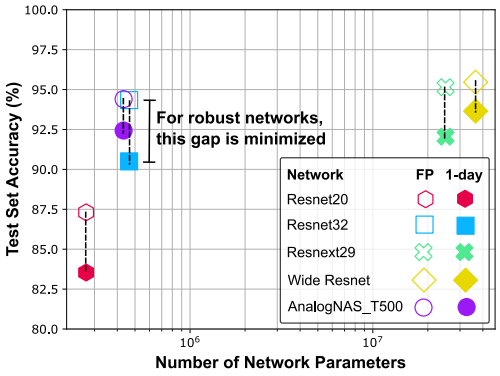

Figure 5.1: The effect of PCM conductance drift after one day on standard CNN architectures and one architecture (`AnalogNAS_T500`) obtained using HW-NAS, evaluated using CIFAR-10.

NVM crossbar arrays and analog circuits, however, have inherent non-idealities, such as noise, temporal conductance drift, and non-linear errors, which can lead to imprecision and noisy computation [280]. These effects need to be properly quantified and mitigated to ensure the high accuracy of DNN models. In addition to the hardware constraints that are prevalent in edge devices, there is the added complexity of

designing DNN architectures which are optimized for the edge on a variety of hardware platforms. This requires hardware-software co-design approaches to tackle this complexity, as manually-designed architectures are often tailored for specific hardware platforms. For instance, MobileNet [281] uses a depth-wise separable convolution that enhances CPU performance but is inefficient for GPU parallelization [282]. These are bespoke solutions that are often hard to implement and generalize to other platforms.

HW-NAS [283] is a promising approach that seeks to automatically identify efficient DNN architectures for a target hardware platform. In contrast to traditional NAS approaches that focus on searching for the most accurate architectures, HW-NAS searches for highly accurate models while optimizing hardware-related metrics. Existing HW-NAS strategies cannot be readily used with analog IMC processors without significant modification for three reasons: (i) their search space contains operations and blocks that are not suitable for analog IMC, (ii) lack of a benchmark of hardware-aware trained architectures, and (iii) their search strategy does not include noise injection and temporal drift on weights. To address these challenges and answer research question 4, we propose *AnalogNAS*, a novel HW-NAS strategy to design dedicated DNN architectures for efficient deployment on edge-based analog IMC inference accelerators. This approach considers the inherent characteristics of analog IMC hardware in the search space and search strategy.

Fig. 5.1 depicts the necessity of our approach. As can be seen, when traditional DNN architectures are deployed on analog IMC hardware, non-idealities, such as conductance drift, drastically reduce network performance. Networks designed by *AnalogNAS* are extremely robust to these non-idealities and have much fewer parameters compared to equivalently-robust traditional networks. Consequently, they have reduced resource utilization.

Our specific contributions can be summarized as follows:

- We design and construct a search space for analog IMC, which contains ResNet-like architectures, including ResNext [284] and Wide-ResNet [285], with blocks of varying widths and depths;
- We train a collection of networks using HWA training for image classification, Visual Wake Words (VWW), and Keyword Spotting (KWS) tasks. Using these networks, we build a surrogate model to rank the architectures during the search and predict robustness to conductance drift;
- We propose a global search strategy that uses evolutionary search to explore the search space and efficiently finds the right architecture under different constraints, including the number of network parameters and analog tiles;
- We conduct comprehensive experiments to empirically demonstrate that AnalogNAS can be efficiently utilized to carry out architecture searches for various edge tiny applications and investigate what attributes of networks make them ideal for implementation using analog AI;
- We validate a subset of networks on hardware using a 64-core IMC chip based on Phase Change Memory (PCM).

5.2 Preliminaries

5.2.1 Analog IMC Accelerator Mechanisms

Analog IMC accelerators are capable of performing MVM operations $\mathbf{Y}^T = \mathbf{X}^T\mathbf{W}$ using the laws of physics, where $\mathbf{W}$ is an $M \times N$ matrix, $\mathbf{X}$ is a $M \times 1$ vector, and $\mathbf{Y}$ is a $N \times 1$ vector. When arranged in a crossbar configuration, $M \times N$, NVM devices can be used to compute MVM operations. This is done by encoding elements of $\mathbf{X}$ as WL

voltages, denoted using $\mathbf{V}$, and elements of $\mathbf{W}$ as conductances of the unit cells, denoted using $\mathbf{G}$. Negative conductance states cannot be directly encoded/represented using NVM devices. Consequently, differential weight mapping schemes are commonly employed, where either positive weights, i.e., $\mathbf{W}^+ = \max(\mathbf{W}, 0)$, and negative weights, i.e., $\mathbf{W}^- = -\min(\mathbf{W}, 0)$, are encoded within unit cells, using alternate columns, or on different tiles [198].

The analog computation, i.e., $\mathbf{I} = \mathbf{VG}$ is performed, where the current flow to the end of the N-th column is $I_N = \sum_{i=0}^{M} G_{i,N} V_i$. Typically, DAC are required to encode WL voltages and ADC are required to read the output currents of each column. The employed analog IMC tile, its weight mapping scheme, and computation mechanism are depicted in Fig. 5.2.

5.2.2 Temporal Drift of Non-Volatile Memory Devices

Many types of NVM devices, most prominently, PCM, exhibit temporal evolution of the conductance values referred to as the conductance drift. This poses challenges for maintaining synaptic weights reliably [197]. Conductance drift is most commonly modelled using Eq. (5.1), as follows:

$$G(t) = G(t_0)(t/t_0)^{-\nu}, \tag{5.1}$$

where $G(t_0)$ is the conductance at time t_0 and ν is the drift exponent. In practice, conductance drift is highly stochastic because ν depends on the programmed conductance state and varies across devices. Consequently, when reporting the network accuracy at a given time instance (after device programming), it is computed across multiple experiment instances (trials) to properly capture the number of accuracy variations.

5.2.3 HWA-training and analog hardware accuracy evaluation simulation

To simulate training and inference on analog IMC accelerators, the AIHWKit [286] is used. The AIHWKit is an open-source Python toolkit for exploring and using the capabilities of in-memory computing devices in the context of artificial intelligence and has been used for HWA training of standard DNN with hardware-calibrated device noise and drift models [287].

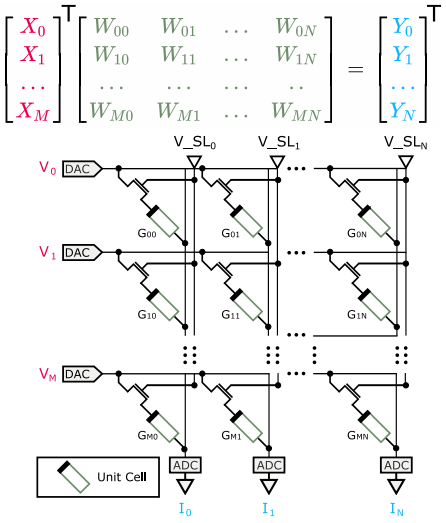

Figure 5.2: Employed analog IMC tile and weight mapping scheme.

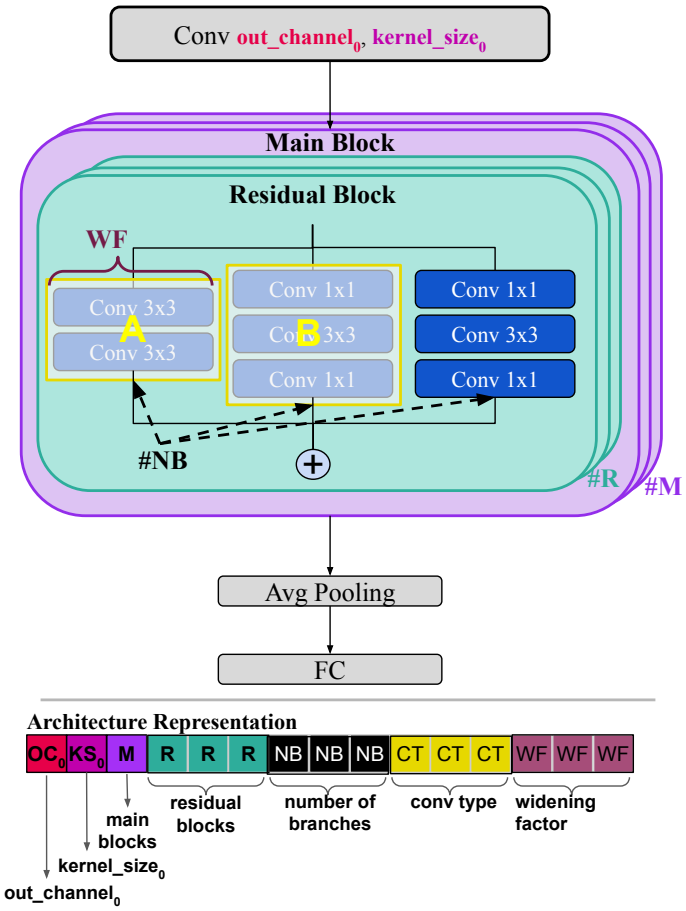

Figure 5.3: Resnet-like macro architecture.

5.3 AnalogNAS: Proposed Approach

The objective of AnalogNAS is to find an efficient network architecture under different analog IMC hardware constraints. AnalogNAS comprises three main components: (i) a resnet-like search space, (ii) an analog-accuracy surrogate model, and (iii) an evolutionary-based search strategy. We detail each component in the following subsections.

5.3.1 Resnet-like Search Space

Resnet-like architectures have inspired many manually designed SOTA DL architectures, including Wide ResNet [285] and EfficientNet [288]. Their block-wise architecture offers a flexible and searchable macro-architecture for NAS [43]. Resnet-like architectures can be implemented efficiently using IMC processors, as they are comprised of a large number of MVM and element-wise operations. Additionally, due to the highly parallel nature of IMC, Resnet architectures can get free processing of additional input/output channels. This makes Resnet-like architectures highly amenable to analog implementation.

Fig. 5.3 depicts the macro-architecture used to construct all architectures in our search space. The architecture consists of a series of M distinct main blocks. Each main block contains R residual blocks. The residual blocks use skip connections with or without downsampling. Downsampling is performed using 1x1 convolution layers when required, i.e., when the input size does not match the output size. The residual

block can have B branches. Each branch uses a convolution block. We used different types of convolution blocks to allow the search space to contain all standard architectures such as Resnets [269], ResNext [284], and Wide Resnets [285]. The standard convolution blocks used in Resnets, commonly referred to as *BottleNeckBlock* and *BasicBlock*, are denoted as A and B respectively. We include variants of A and B in which we inverse the order of the ReLU and Batch normalization operations. The resulting blocks are denoted as C and D. Table 5.1 summarizes the searchable hyperparameters and their respective ranges. The widening factor scales the width of the residual block. We sample architectures with different depths by changing the number of main and residual blocks. The total size of the search space is approximately 73B architectures. The larger architecture would contain 240 convolutions and start from an output channel of 128 multiplying that by 4 for every 16 blocks.

5.3.2 Analog-accuracy Surrogate Model

Evaluation Criteria

To efficiently explore the search space, a search strategy requires evaluating the objectives of each sampled architecture. Training the sampled architectures is very time-consuming; especially when HWA retraining is performed, as noise injection and I/O quantization modeling greatly increases the computational complexity. Consequently, we build a surrogate model capable of estimating the objectives of each sampled architecture in IMC devices. To find architectures that maximize accuracy, stability, and robustness against IMC noise and drift characteristics, we have identified three objectives: the 1-day accuracy, AVM, and the 1-day accuracy's standard deviation.

The 1-day accuracy is the primary objective that most NAS algorithms aim to maximize. It measures the performance of an architecture on a given dataset. When weights are encoded using IMC devices, the accuracy of the architecture can drop over time due to conductance drift. Therefore, we have selected the 1-day accuracy as a metric to measure the architecture's performance.

The Accuracy Variation over one Month (AVM) is the difference between the 1-month and 1-sec accuracy. This objective is essential to measure the robustness over a fixed time duration. A 30-day period allows for a reasonable trade-off between capturing meaningful accuracy changes and avoiding short-term noise and fluctuations that may not reflect long-term trends.

The 1-day accuracy standard deviation measures the variation of the architecture's performance across experiments, as discussed in Section 5.2.2. A lower standard deviation indicates that the architecture produces consistent results on hardware deployments, which is essential for real-world applications.
To build the surrogate model, we follow two steps: Dataset creation and Model training:

Dataset Creation

The surrogate model will predict the rank based on the 1-day accuracy and estimates the AVM and 1-day accuracy standard deviation using the MSE. Since the search space is large, care has to be taken when sampling the dataset of architectures that will be used to train the surrogate model.

The architectures of the search space are sampled using two methods: (i) LHS [232] and (ii) NAS with full training. A more detailed description of the AnalogNAS algorithm is presented in Section 5.3.3. We use LHS to sample architectures distributed evenly over the search space. This ensures good overall coverage of different architectures and their accuracies. NAS with full training is performed using an

Table 5.1: Searchable hyper-parameters and their respective ranges.

Hyper-parameter	**Range**
out_channel_0	Discrete Uniform Distribution [8,128]
kernel_size_0	Discrete Uniform Distribution [3,7]
M	Discrete Uniform Distribution [1, 5]
R*	Discrete Uniform Distribution [1, 16]
B*	Discrete Uniform Distribution [1, 12]
ConvBlock*	Uniform Choice [A; B; C; D]
Widening Factor*	Uniform Distribution [1,4]

*The hyper-parameter is repeated for each main block. ConvBlock refers to different Conv-Relu-BN blocks.

evolutionary algorithm to collect high-performance architectures. This ensures good exploitation when reaching well-performing regions.

In Fig. 5.4, we present a visualization of the search space coverage, which does not show any clustering of similarly performing architectures at the edge of the main cloud of points. Thus, it is not evident that architectures with similar performance are located close to each other in the search space. This suggests that traditional search methods that rely on local optimization may not be effective in finding the best-performing architectures. Instead, population-based search strategies, which explore a diverse set of architectures, could be more effective in finding better-performing architectures. Our search strategy extracted 400 test points, and we found that architectures were distributed throughout the main cloud, indicating that our dataset covers a diverse portion of the search space, despite the limited size of only 1,000.

Each sampled architecture is trained using different levels of weight noise and HWA training hyper-parameters using the AIHWKit [286]. Specifically, we modify the standard deviation of the added weight noise between [0.1, 5.0] in increments of 0.1. The tile size was assumed to be symmetric and varied in [256, 512], representing 256-by-256 and 512-by-512 arrays respectively. Training with different configurations allowed us to generalize the use of the surrogate model across a range of IMC hardware configurations, and to increase the size of the constructed dataset.

Model training

To train the surrogate model, we used a hinge pair-wise ranking loss [16] with margin $m = 0.1$. The hinge loss, defined in Eq. (5.2), allows the model to learn the relative

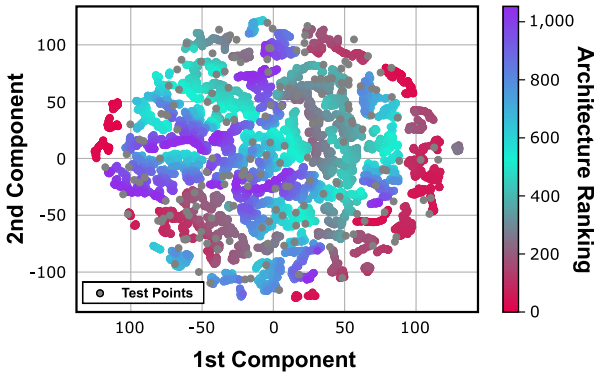

Figure 5.4: t-Distributed Stochastic Neighbor Embedding (t-SNE) visualization of the sampled architectures for CIFAR-10.

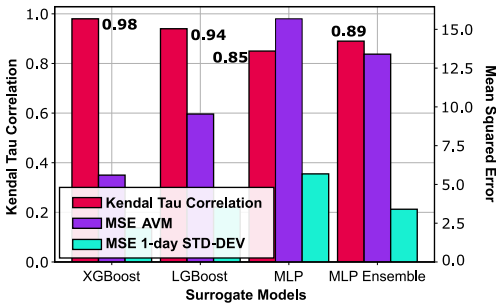

Figure 5.5: Surrogate models comparison.

ranking order of architectures rather than the absolute accuracy values [16, 289].

$$L(\{a_j, y_j\}_{j=1,\ldots,N}) = \sum_{j=1}^{N} \sum_{\{i,j|y_i>y_j\}} \max[0, m - P(a_i) - P(a_j)] \tag{5.2}$$

a_j refers to architectures indexed j, and y_j to its corresponding 1-day accuracy. $P(a)$ is the predicted score of architecture a. $P(a)$ during training, the output score is trained to be correlated with the actual ranks of the architectures. Several algorithms were tested. After an empirical comparison, we adopted Kendall's Tau ranking correlation [290] as the direct criterion for evaluating ranking surrogate model performance. Fig. 5.5 shows the comparison using different ML algorithms to predict the rankings and AVM. Our dataset is tabular. It contains each architecture and its corresponding features. XGBoost outperforms the different surrogate models in predicting the architectures' ranking order, the AVM of each architecture, and the 1-day standard deviation.

5.3.3 Search Strategy

Fig. 5.6 depicts the overall search framework. Given a dataset and a hardware configuration readable by AIHWKit, the framework starts by building the surrogate model presented in Section 5.3.2. Then, we use an optimized evolutionary search to efficiently explore the search space using the surrogate model. Similar to traditional evolutionary algorithms, we use real number encoding. Each architecture is encoded into a vector, and each element of the vector contains the value of the hyper-parameter, as listed in Table 5.1.

5.3.4 Problem Formulation

Given the search space S, our goal is to find an architecture α, that maximizes the ratio of 1-day accuracy to 1-day standard deviation, subject to constraints on the number of parameters and the AVM. The number of parameters is an important metric in IMC, because it directly impacts the amount of on-chip memory required to store the weights of a DNN. Eq. (5.3) formally describes the optimization problem as follows:

$$\begin{aligned} \max_{\alpha \in S} \quad & \frac{\mathrm{ACC}(\alpha)}{\sigma(\alpha)} \\ \text{s.t} \quad & \psi(\alpha) < T_p \\ & \mathrm{AVM}(\alpha) < T_{\mathrm{AVM}}. \end{aligned} \tag{5.3}$$

ACC refers to the 1-day accuracy objective, σ denotes the 1-day accuracy's standard deviation, and ψ is the number of parameters. T_p and T_{AVM} are user-defined

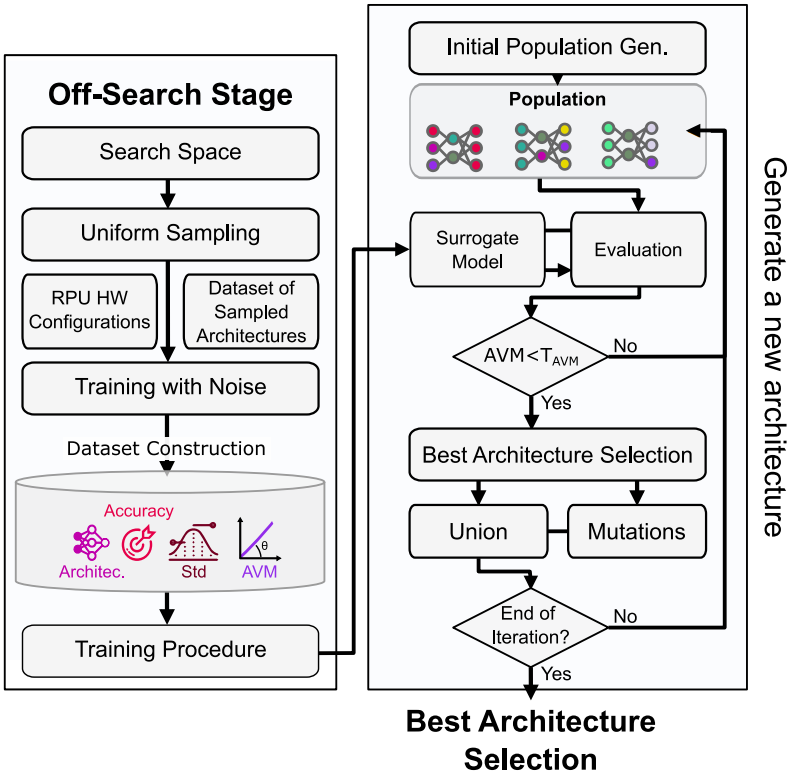

Figure 5.6: Overview of the AnalogNAS framework.

thresholds that correspond to the maximum number of parameters and AVM, respectively.

5.3.5 Search Algorithm

Our evolutionary search algorithm, i.e., AnalogNAS, is formally defined using Algorithm 6. AnalogNAS is an algorithm to find the most accurate and robust neural network architecture for a given analog IMC configuration and task. The algorithm begins by generating a dataset of neural network architectures, which are trained on the task and evaluated using AIHWKit. A surrogate model is then created to predict the efficiency of new architectures. The algorithm then generates a population of architectures using an LHS technique and selects the top-performing architectures to be mutated and generate a new population. The process is repeated until a stopping criterion is met, such as a maximum number of iterations or a time budget. Finally, the most robust architecture is returned. In the following, we detail how the population initialization, fitness evaluation, and mutations are achieved.

Population Initialization

The search starts by generating an initial population. Using the LHS algorithm, we sample the population uniformly from the search space. LHS ensures that the initial population contains architectures with different architectural features. LHS is made faster with parallelization by dividing the sampling into multiple independent subsets, which can be generated in parallel using multiple threads.

Fitness Evaluation

We evaluate the population using the aforementioned analog-accuracy surrogate model. In addition to the rankings, the surrogate model predicts the AVM of each

Algorithm 6 AnalogNAS algorithm.

Input: Search space: S, RPU Configuration: rpu_config, target task: $task$, population size: $population\_size$, AVM threshold: T_{AVM}, parameter threshold: T_p, number of iterations: N, $time\_budget$
Output: Most efficient architecture for rpu_config in S
Begin
D = sample(S, dataset_size)
HW_Train(D, $task$)
AVM = compute_AVM(D)
surrogate_model = XGBoost
train(surrogate_model, D, AVM)
repeat
  population = LHS(population_size, T_p)
  AVM, ranks = surrogate_model(population)
until AVM $> T_{\text{AVM}}$
while $i < N$ or $time < time\_budget$ **do**
  top_50 = select(population, ranks)
  mutated = mutation(top_50, T_p)
  population = top_50 $\bigcup$ mutated
  AVM, ranks = surrogate_model(population)
end while
return top_1(population, ranks)

architecture. As previously described, the AVM is used to gauge the robustness of a given network. If the AVM is below a defined threshold, T_{AVM}, the architecture is replaced by a randomly sampled architecture. The new architecture is constrained to be sampled from the same hypercube dimension as the previous one. This ensures efficient exploration.

Selection and Mutation

We select the top 50% architectures from the population using the predicted rankings. These architectures are mutated. The mutation functions are classified as follows:

Depth-related mutations modify the depth of the architectures. Mutations include adding a main block, by increasing or decreasing M or a residual block R, or modifying the type of convolution block, i.e., $\{A, B, C, D\}$, for each main block.

Width-related mutations modify the width of the architectures. Mutations include modifying the widening factor W of a main block or adding or removing a branch B, or modifying the initial output channel size of the first convolution, OC.

Other mutations modify the kernel size of the first convolution, KS, and/or add skip connections, denoted using ST.
Depth- and width-related mutations are applied with the same probability of 80%. The other mutations are applied with a 50% probability. In each class, the same probability is given to each mutation. The top 50% architectures in addition to the mutated architectures constitute the new population. For the remaining iterations, we verify the ranking correlation of the surrogate model. If the surrogate model's ranking correlation is degraded, we fine-tune the surrogate model with the population's architectures. The degradation is computed every 100 iterations. The surrogate model is tested on the population architectures after training them. It is fine-tuned if Kendall's tau correlation drops below 0.9.

5.4 Evaluation Methodology

We detail the hyper-parameters used to train the surrogate model and different architectures on CIFAR-10, VWW, and KWS tasks.

Surrogate model training We trained a surrogate model and dataset of HWA trained DNN architctures for each task. The sizes of the datasets were 1,200, 600, and 1,500, respectively. An additional 500 architectures were collected during the search trials for validation. All architectures were first trained without noise injection (i.e., using vanilla training routines), and then converted to AIHWKit models for HWA retraining. The surrogate model architecture used was XGBoost. For VWW and KWS, the surrogate model was fine-tuned from the image classification XGBoost model.

Image classification training We first trained the network architectures using the CIFAR-10 dataset [291], which contains 50,000 training and 10,000 test samples, evenly distributed across 10 classes. We augmented the training images with random crops and cutouts only. For training, we used SGD with a learning rate of 0.05 and a momentum of 0.9 with a weight decay of 5e-4. The learning rate was adjusted using a cosine annealing learning rate scheduler with a starting value of 0.05 and a maximum number of 400 iterations.

VWW training We first trained the network architectures using the VWW dataset [292], which contains 82,783 train and 40,504 test images. Images are labeled 1 when a person is detected, and 0 when no person is present. The image preprocessing pipeline included horizontal and vertical flipping, scale augmentation [293], and random RGB color shift. To train the architectures, we used the RMSProp optimizer [294] with a momentum of 0.9, a learning rate of 0.01, a batch normalization momentum of 0.99, and a l_2 weight decay of 1e-5.

KWS training We first trained the network architectures using the KWS dataset [295], which contains 1-second long incoming audio clips. These are classified into one of twelve keyword classes, including "silence" and "unknown" keywords. The dataset contains 85,511 training, 10,102 validation, and 4,890 test samples. The input was transformed to $49 \times 10 \times 1$ features from the Mel-frequency cepstral coefficients [296]. The data pre-processing pipeline included applying background noise and random timing jitter. To train the architectures, we used the Adam optimizer [297] with a decay of 0.9, a learning rate of 3e-05, and a linear learning rate scheduler with a warm-up ratio of 0.1.

Search Algorithm
The search algorithm was run five times to compute the variance. The evolutionary search was executed with a population size of 200. If not explicitly mentioned, the AVM threshold was set to 10%. The width and depth mutation probability was set to 0.8. The other mutations' probability was set to 0.5. The total number of iterations was 200. After the search, the obtained architecture for each task was compared to SOTA baselines for comparison.

5.5 Experiment Results

The final architecture compositions for the three tasks are listed in Table 5.2. In addition, figure 5.10 highlights the architectural differences between AnalogNAS_T500 and resnet32. We modified T_p to find smaller architectures. To determine the optimal architecture for different parameter thresholds, we use TX, where X represents the

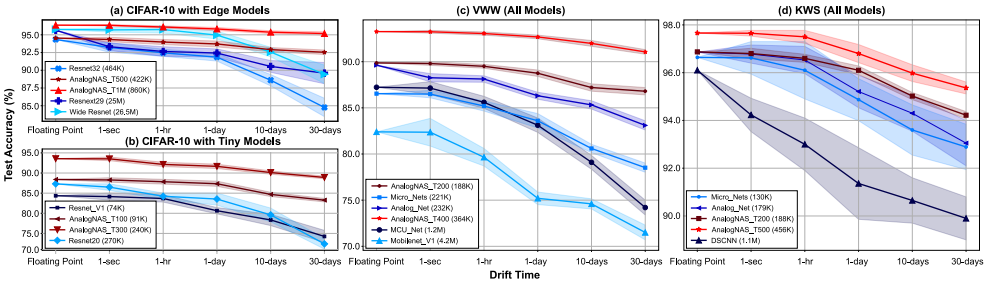

Figure 5.7: Simulated hardware comparison results on three benchmarks: (a,b) CIFAR-10, (c) VWW, and (d) KWS. The size of the marker represents the size (i.e., the number of parameters) of each model. The shaded area corresponds to the standard deviation at that time.

Table 5.2: Final Architectures for CIFAR-10, VWW, and KWS. Other networks for VWW and KWS are not listed, as they cannot easily be represented using our macro-architecture.

Network	**Macro-Architecture Parameter**						
	OC_0	**KS_0**	**M**	**R***	**B***	**CT***	**WF***
CIFAR-10							
Resnet32	64	7	3	(5, 5, 5)	(1, 1, 1)	(B, B, B)	(1, 1, 1)
AnalogNAS_T100	32	3	1	(2,)	(1,)	(C,)	(2,)
AnalogNAS_T300	32	3	1	(3, 3)	(1, 1)	(A, B)	(2, 1)
AnalogNAS_T500	64	5	1	(3,)	(3,)	(A,)	(2,)
AnalogNAS_T1M	32	5	2	(3, 3)	(2, 2)	(A, A)	(3, 3)
VWW							
AnalogNAS_T200	24	3	3	(2, 2, 2)	(1, 2, 1)	(B, A, A)	(2, 2, 2)
AnalogNAS_T400	68	3	2	(3, 5)	(2, 1)	(C, C)	(3, 2)
KWS							
AnalogNAS_T200	80	1	1	(1,)	(2,)	(C,)	(4,)
AnalogNAS_T400	68	1	2	(2, 1)	(1, 2)	(B, B)	(3, 3)

*As depicted in Fig. 5.3, thse macro-architecture parameters comprise multiple instances.

threshold T_p in K units (e.g., T100 refers to the architecture with a threshold of 100K parameters). When searching for T200 and T100, the probability of increasing the widening factor or depth to their highest values, was lessened to 0.2.

In Fig. 5.7, the simulated hardware comparison of the three tasks is depicted. Our models outperform SOTA architectures with respect to both accuracy and resilience to drift. On CIFAR-10, after training the surrogate model, the search took 17 minutes to run. The final architecture, T500, is smaller than Resnet32, and achieved +1.86% better accuracy and a drop of 1.8% after a month of inference, compared to 5.04%. This model is $\sim 86\times$ smaller than Wide Resnet [285], which has 36.5M parameters. Our smallest model, T100, was $1.23\times$ bigger than Resnet-V1, the SOTA model bench-marked by MLPerf [298]. Despite not containing any depth-wise convolutions, Resnet_V1 is extremely small, with only 70k parameters. Our model offers a +7.98% accuracy increase with a 5.14% drop after a month of drift compared to 10.1% drop for Resnet_V1. Besides, our largest model, *AnalogNAS_1M*, outperforms Wide Resnet with +0.86% in the 1-day accuracy with a drop of only 1.16% compared to 6.33%. In addition, the found models exhibit greater consistency across experiment trials, with an average standard deviation of 0.43 over multiple drift times as

opposed to 0.97 for SOTA models.

Similar conclusions can be made about VWW and KWS. In VWW, current baselines use a depth-wise separable convolution that incurs a high accuracy drop on analog devices. Compared to AnalogNet-VWW and Micronets-VWW, the current SOTA networks for VWW in analog and edge devices, our T200 model has a similar number of parameters (x1.23 smaller) with a +2.44% and +5.1% 1-day accuracy increase respectively. AnalogNAS was able to find more robust and consistent networks with an average AVM of 2.63% and a standard deviation of 0.24. MCUNet [266] and MobileNet-V1 present the highest AVM. This is due to the sole use of depth-wise separable convolutions.

On KWS, the baseline architectures, including DSCNN [245], use hybrid networks containing recurrent cells and convolutions. The recurrent part of the model ensures high robustness to noise. While current models are already robust with an average accuracy drop of 4.72%, our model outperforms tiny SOTA models with 96.8% and an accuracy drop of 2.3% after a month of drift. Critically, our AnalogNAS models exhibit greater consistency across experiment trails, with an average standard deviation of 0.17.

5.5.1 Comparison with Random Search

In accordance with commonly accepted NAS methodologies, we conducted a comparative analysis of our search approach with Random Search. Results, presented in Fig. 5.8, were obtained across five experiment instances. Our findings indicate that Random Search was unable to match the 1-day accuracy levels of our final models, even after conducting experiments for a duration of four hours and using the same surrogate model. We further conducted an ablation study to evaluate the effectiveness of our approach by analyzing the impact of the LHS algorithm and surrogate model. The use of a random sampling strategy and exclusion of the surrogate model resulted in a significant increase in search time. The LHS algorithm helped in starting from a diverse initial population and improving exploration efficiency, while the surrogate model played a crucial role in ensuring practical search times.

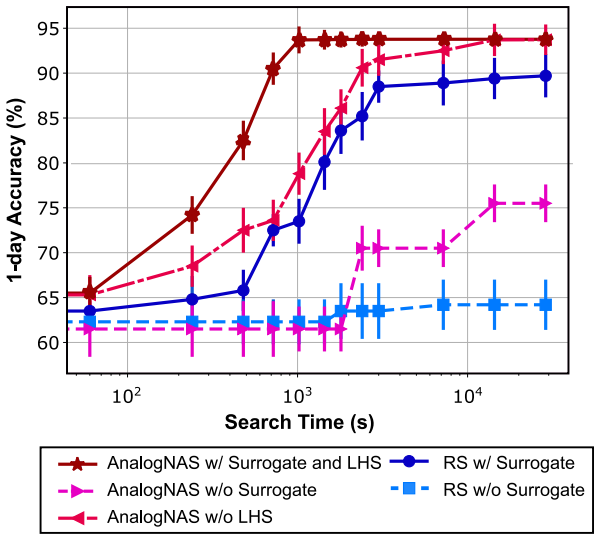

Figure 5.8: Ablation study comparison against Random Search (RS). Mean and standard deviation values are reported across five experiment instances (trials).

5.5.2 Search Time and AVM Threshold Trade-Off

During the search, we culled architectures using their predicted AVM, i.e., any architecture with a higher AVM than the AVM threshold was disregarded. As listed in Table 5.3, we varied this threshold to investigate the trade-off between T_{AVM} and the search time. As can be seen, as T_{AVM} is decreased, the delta between AVM and T_{AVM} significantly decreases. The correlation between the search time and T_{AVM} is observed to be non-linear.

Table 5.3: AVM threshold variation results on CIFAR-10.

T_{AVM} (%)	1.0	3.0	5.0*
1-day Accuracy	88.7%	93.71%	93.71%
AVM	0.85%	1.8%	1.8%
Search Time (min)	34.65	28.12	17.65

*Overall results computed with T_{AVM} (%) = 5.0.

5.6 Experimental Hardware Validation & Architecture Performance Simulations

5.6.1 Experimental Hardware Validation

An experimental hardware accuracy validation study was performed using a 64-core IMC chip based on PCM [299]. Each core comprises a crossbar array of 256x256 PCM-based unit-cells along with a local digital processing unit [300]. This validation study was performed to verify whether the simulated network accuracy values and rankings are representatives of those when the networks are deployed on real physical hardware. We deployed two networks for the CIFAR-10 image classification task on hardware: AnalogNAS_T500 and the baseline ResNet32 [269] networks from Fig. 5.7(a).

To implement the aforementioned models on hardware, after HWA training was performed, a number of steps were carried out. First, from the AIHWKit, unit weights of linear (dense) and unrolled convolutional layers, were exported to a state dictionary file. This was used to map network parameters to corresponding network layers. Additionally, the computational inference graph of each network was exported. These files were used to generate proprietary data-flows to be executed in-memory. As only hardware accuracy validation was being performed, all other operations aside from MVM were performed on a host machine connected to the chip through an FPGA. The measured hardware accuracy was 92.05% for T500 and 89.87% for Resnet32, as reported in Table 5.4. Hence, the T500 network performs significantly better than Resnet32 also when implemented on real hardware. This further validates that our proposed AnalogNAS approach is able to find networks with a similar number of parameters that are more accurate and robust on analog IMC hardware.

5.6.2 Simulated Hardware Energy and Latency

We conducted power performance simulations for AnalogNAS_T500 and ResNet32 models using a 2D-mesh-based heterogeneous analog IMC system with the simulation tool presented in [21]. The simulated IMC system consists of one analog fabric with 48 analog tiles of 512x512 size, on-chip digital processing units, and digital memory for activation orchestration between CNN layers. Unlike the accuracy validation experiments on the 64-core IMC chip, the simulated power performance assumes all

Table 5.4: Experimental hardware accuracy validation and simulated power performance on the IMC system in [21].

Architecture	**ResNet32**	**AnalogNAS_T500**
Hardware Experiments		
Accuracy*	89.87%	92.05%
Simulated Hardware Power Performance		
Total weights	464,432	416,960
Total tiles	43	27
Network Depth	32	17
Execution Time (msec)	0.434	0.108
Inferences/s/W	43,956.7	54,502

*The mean accuracy is reported across three experiment repetitions.

intermediate operations to be mapped and executed on-chip. Our results, provided in Table 5.4, show that AnalogNAS_T500 outperformed ResNet32 in terms of both execution time and energy efficiency.

We believe that this power performance benefit is realized because, in analog IMC hardware, wider layers can be computed in parallel, leveraging the $O(1)$ latency from analog tiles, and are therefore preferred over deeper layers. It is noted that both networks exhibit poor tile utilization and that the tile utilization and efficiency of these networks could be further improved by incorporating these metrics as explicit constraints. This is left to future work and is beyond the scope of AnalogNAS.

5.7 Architectural Recommendation for Analog AI

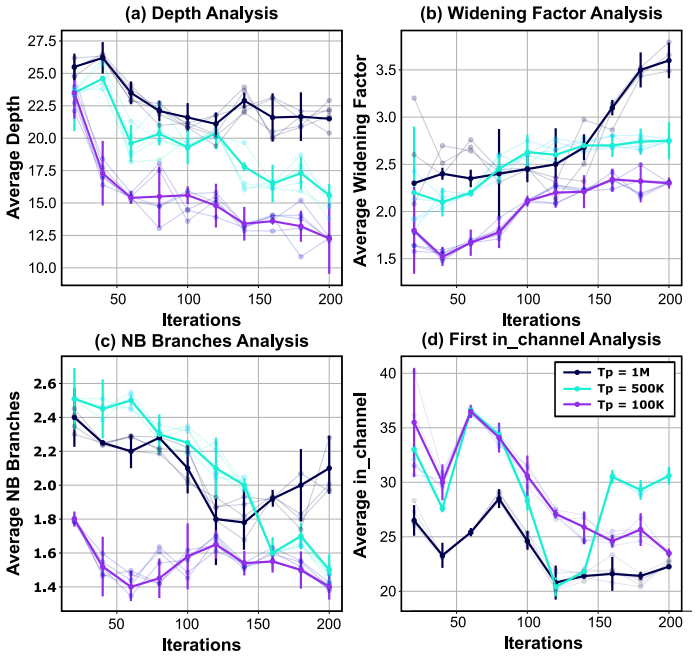

Figure 5.9: Evolution of architecture characteristics in the population during the search for CIFAR-10. Random individual networks are shown.

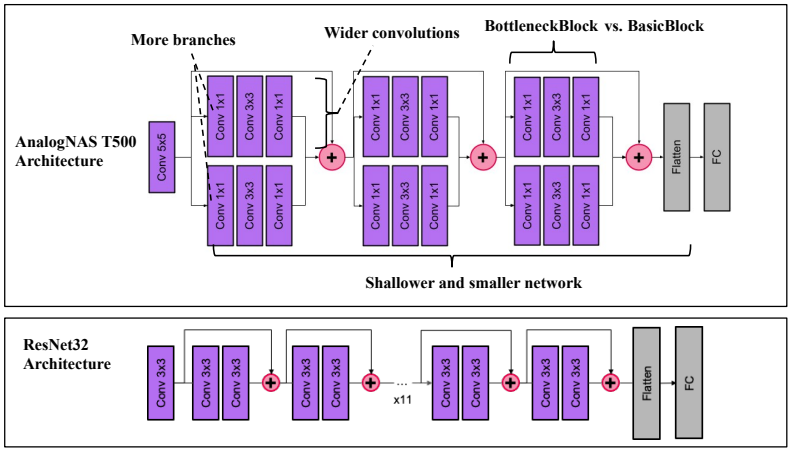

Figure 5.10: Architectural differences between AnalogNAS_T500 and Resnet32.

During the search, we analyzed the architecture characteristics and studied which types of architectures perform the best on IMC inference processors. The favored architectures combine robustness to noise and accuracy performance. Fig. 5.9 shows the evolution of the average depth, the average widening factor, the average number of branches, and the average first convolution's output channel size of the search population for every 20 iterations. The depth represents the number of convolutions. A sampled architecture has a widening factor per block. To compute the average widening factor, we first computed the average widening factor per architecture by dividing the sum of the widening factors by the number of blocks contained in the architecture. Then, we calculated the average widening factor across all architectures. Similar computations were performed for the average number of branches.

For each plot, the search was run 5 times and the mean is represented in each point. The plotted error corresponds to one standard deviation from that mean. Starting from a random population obtained using LHS, the population evolves through different width and depth-related mutations. During this analysis, we want to answer the following questions: (i) Does the search favor wide or deep networks? And subsequently, are wider architectures more noise resilient? (ii) what architectures are exploited by the search for different tasks when constraining the number of parameters?

5.7.1 Are Wider or Deeper Networks More Robust to PCM Device Drift?

From Fig. 5.9, it can be observed that the depth of all networks decreases during the search. This trend is especially seen when we constrain the model's size to 100K and 500K parameters. During the search, the widening factor also increases, allowing the blocks to have wider convolutions. The number of branches is highly dependent on T_p. This number is, on average, between 1 and 2. The branches are the number of parallel convolutions in a block disregarding the skip connection. In the literature, architectures such as ResNext, that support a higher number of branches, have a number of parameters around 30M. It is still interesting to get blocks with two branches, which also reflects an increase in the width of the network by increasing the number of features extracted within the same block. The average output channel size of the first convolution decreases during the search. Its final value is around the same number of output channels as standard architectures, i.e., between 16 and 32. This follows the general trend of having wider convolutions in deeper network positions.

5.7.2 Types Of Architectures

The final architectures for each task and the number of parameters constraint are different. They all, however, show an increasing expansion ratio in the convolution block. This increase allows the blocks' convolutions to utilize more of the tile and thus reduce the noise effect from the non-utilized cells in the crossbar. For CIFAR-10, architectures behave like Wide Resnet [285] while respecting the number of parameters constraint. For the VWW task, the architectures are deeper. The input resolution is 224×224, which requires more feature extraction blocks. However, they are still smaller than SOTA architectures, with a maximum depth of 22. As depth is essential to obtain high accuracy for the VWW task, no additional branches are added. For the KWS task, the architectures are the widest possible, maximizing the tile utilization for each convolutional layer.

5.8 Conclusion

In this chapter, we propose an efficient NAS methodology dedicated to analog in-memory computing for TinyML tasks entitled AnalogNAS. The obtained models are accurate, noise and drift-resilient, and small enough to run on resource-constrained devices. Experimental results demonstrate that our method outperforms SOTA models on analog hardware for three tasks of the MLPerf benchmark: image classification on CIFAR-10, VWW, and KWS. Our AnalogNAS_T500 model implemented on physical hardware demonstrates $> 2\%$ higher accuracy experimentally on the CIFAR-10 benchmark than ResNet32. Calculated speed and energy efficiency estimates reveal a $> 4\times$ reduction in execution time, in addition to $> 1.2\times$ higher energy efficiency for AnalogNAS_T500 compared with ResNet32 when evaluated using a system-level simulator.

Chapter 6

HW-NAS for Medical Imaging Analysis

Contents

Edge devices, such as smartphones and wearable devices, are increasingly being utilized for real-time medical image analysis, enabling rapid diagnosis and treatment decisions at the point of care. By incorporating HW-NAS techniques, we can effectively optimize DL architectures for edge medical tasks. However, the lack of standardized NAS benchmarks tailored for medical imaging on edge devices poses a significant challenge in developing efficient and practical approaches. In this chapter, we propose a dedicated NAS benchmark for medical imaging analysis, *MED-NAS-Bench*, which encompasses a large number of architectures per task. Additionally, we design a multi-task architecture search, called *MT-MIAS*, that leverages this benchmark to optimize multiple medical imaging tasks concurrently. In what follows, we present challenges faced in medical imaging tasks and the methodologies we have adopted to address some of these challenges.

6.1 Context

With the transition to electronic health records (EHR) over the last decade, the healthcare industry has witnessed an exponential increase in the amount of available EHR data. This surge in data presents a remarkable opportunity to harness the power of artificial intelligence (AI) and unlock its potential to revolutionize the healthcare system. In recent years, deep learning (DL) practitioners have played a significant role in leveraging AI to address critical challenges in medical imaging. They have developed a range of sophisticated models [301, 302, 303] specifically designed to tackle the segmentation of complex structures such as brain tumors, liver tumors, and lung tumors. By employing DL techniques, these models have shown great promise in accurately delineating these structures from medical imaging data. By providing accurate segmentation, these models supply clinicians with detailed insights into the organ's structure and aid in determining the extent of diseases.

Given the state-of-the-art results of NAS, researchers have delved into exploring its potential applications in medical imaging and DL-based segmentation tasks. By employing NAS techniques, DL practitioners can optimize the architecture and parameters of segmentation models, leading to improved accuracy. While current medical NAS methods [304, 305, 306] achieve state-of-the-art results in many tasks, they also face several challenges that researchers are actively working to overcome. Among these challenges: (1) the lack of a common benchmark to compare different NAS methods against. Currently, there is a lack of standardized benchmarks specifically tailored for evaluating the performance of NAS models in medical imaging tasks. (2) Given the lack of benchmarks, researchers face difficulties building performance surrogate and estimation models. This exacerbates the computational complexity and resource requirements of these methods.

In addition, the sensitive nature of healthcare data necessitates the need for edge inference, where computations are performed locally on devices rather than relying on cloud-based processing. This enables faster and more secure analysis of healthcare data while preserving patient privacy. To achieve efficient and effective edge inference in healthcare, the search for optimized deep learning architectures is crucial. While HW-NAS has shown success in visual tasks such as image classification and object detection, its application to medical imaging and healthcare-specific tasks is an area of active research. The unique requirements of medical image analysis, such as high resolution, multi-modal inputs, and complex structures, necessitate the development of HW-NAS techniques specifically adapted to the healthcare domain.

In this chapter, we first attempt to address the following research question:

Research Question 5

What are the key considerations and methodologies for developing a comprehensive benchmark specifically tailored for evaluating NAS methods in the domain of medical imaging, and how can such a benchmark be designed to effectively capture the complexities and challenges presented by medical imaging datasets?

We present *MED-NAS-Bench*, a generalized neural architecture search benchmark for medical imaging analysis. This benchmark not only includes task-specific metrics such as dice score, Jaccard score, and f1-score, but also the hardware efficiency measured with the latency and energy consumption on Raspberry Pi and Laptop AMD Ryzen 7 6800H [1]. These devices are cost-effective and readily available for hospitals and clinicians, ensuring practicality and affordability in real-world healthcare settings. By employing *MED-NAS-Bench*, researchers and practitioners can evaluate the performance and efficiency of NAS models in medical imaging analysis, enabling informed decision-making and facilitating the adoption of optimized architectures for various medical imaging tasks.

Medical imaging data is inherently complex and heterogeneous, with variations in imaging modalities, anatomical structures, and disease presentations. By incorporating multi-task learning into HW-NAS, the architecture search process can leverage the relationships and dependencies between different segmentation tasks. This allows for the discovery of architectures that can effectively capture the intricacies of multiple anatomical structures or diseases simultaneously. This allows the deployment of a single architecture that can perform well on a range of different tasks. We then formulate our research question 6, as follows:

Research Question 6

How can a multi-task HW-NAS methodology be developed to effectively account for the diverse range of medical imaging types and tasks?

To address this challenge, we present MT-MIAS, a multi-task architecture search for medical imaging analysis that significantly increases the efficiency and productivity of medical imaging models on edge devices.

6.2 MED-NAS-Bench

In this section, we present the methodology employed to construct MED-NAS-Bench. The construction of MED-NAS-Bench involved several key steps to ensure its effectiveness and suitability for evaluating NAS methods in the context of medical imaging tasks.

6.2.1 Datasets

MED-NAS-Bench targets 11 medical tasks, ten of which are obtained from the medical segmentation decathlon (MSD) [18], plus a pulmonary diseases detection dataset.

Medical Segmentation Decathlon (MSD) [18] is a widely recognized benchmark dataset and challenge specifically focused on medical image segmentation tasks. The MSD was created to address the need for standardized evaluation and comparison of segmentation methods across a range of anatomical structures and diverse imaging modalities, including computed tomography (CT), magnetic resonance imaging

[1] Our benchmark is under active development, we are in the process of collecting measurement on Jetson Nano, Jetson NX, and other hardware platforms.

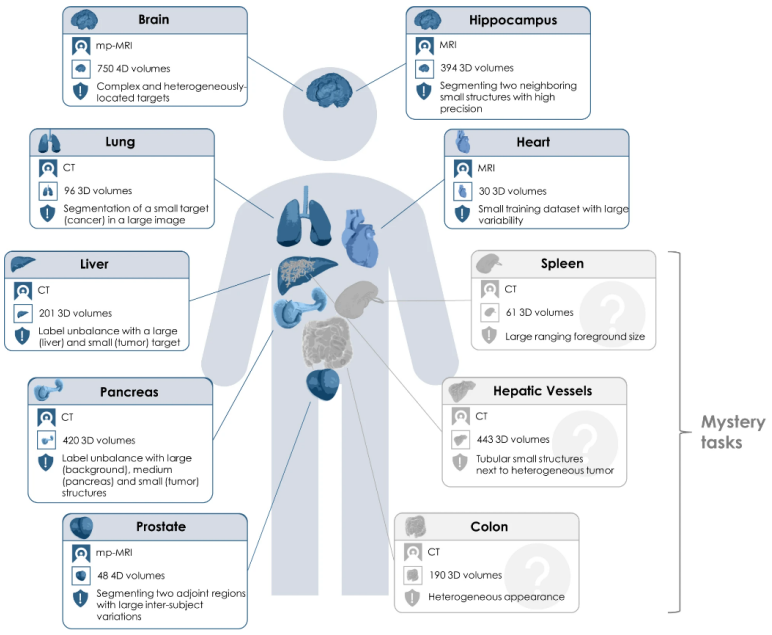

Figure 6.1: Overview of the ten different tasks of the Medical Segmentation Decathlon (MSD) [18]

(MRI), and positron emission tomography (PET). The dataset consists of 10 different medical imaging segmentation tasks, including brain, lung, liver, colon, pancreas tumors, left atrium, prostate peripheral and transition zones, hepatic vessels, hippocampus, and spleen segmentation. Each task comprises a large number of volumetric images along with the corresponding ground truth segmentations, providing a comprehensive and diverse collection of medical imaging data.

Image Segmentation

Image segmentation is the process of dividing an image into distinct and meaningful regions or segments, with the goal of extracting and isolating objects or areas of interest within the image.

Referring to Figure 6.1, MSD was thoughtfully designed to promote generalizability across various medical tasks. The decathlon framework was structured in a two-step process to facilitate comprehensive evaluation and encourage the development of robust segmentation models. In the initial phase, participants were provided with 7 tasks highlighted in blue, focusing on well-defined segmentation challenges. This allowed participants to explore and develop their best-performing models for these specific tasks. Subsequently, a f ine-tuning step was introduced, where participants were presented with mystery tasks (i.e;, gray tasks). By fine-tuning their models on these mystery tasks, participants had the opportunity to test the generalization capability of their proposed solutions across different medical tasks. A similar process will be followed in our multi-task HW-NAS methodology in section 6.3.

The NIH Chest X-Ray Dataset [307] is a widely used dataset in the field of medical imaging, specifically focused on chest X-ray analysis. This dataset was compiled by the National Institutes of Health (NIH) and consists of 112,120 X-ray images with disease labels from 30,805 unique patients along with associated radiologist-labeled annotations. It encompasses a diverse range of chest conditions, including abnormalities such as pneumonia, lung nodules, and pleural effusion, as well as normal chest

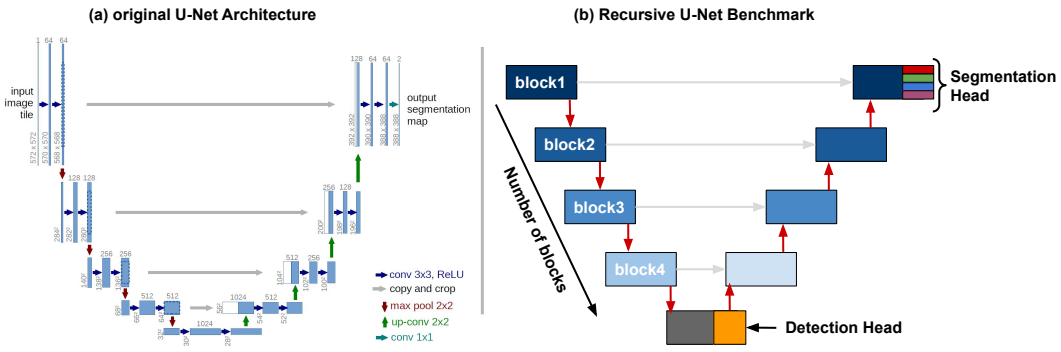

Figure 6.2: Search Space of MED-NAS-Benchmark

X-rays for comparison.

6.2.2 Benchmark Design

To build a search space, one needs to define the set of architectures and ranges of hyperparameters that will be explored during the NAS process. In this study, our search space construction draws inspiration from DARTS, as proposed in the referenced work [166]. DARTS introduces the concept of a supernetwork, which is an over-parameterized network where each layer can compute different operations simultaneously. These operations are typically organized into blocks, allowing for flexible and efficient exploration of architectural configurations. To build a supernetwork, we need to define the general macro-architecture that is followed, including the depth of the network, and specify the candidate operations that can be performed at each layer.

High-level overview

Our search space resembles a U-Net-like architecture. U-Net [301] is a DL architecture that has shown remarkable performance in medical image segmentation tasks. It consists of an encoder-decoder structure with skip connections, enabling effective feature extraction and precise localization of structures within the image. The encoder portion captures high-level features while the decoder recovers spatial information, and the skip connections help in preserving fine-grained details. By incorporating U-Net-like characteristics into our search space, we aim to leverage the strengths of this architecture in addressing the complexities and challenges presented by medical imaging datasets.

Figure 6.2 presents a visual representation of the search space definition, highlighting the key differences between the proposed search space and the conventional U-Net architecture. The search space is designed to accommodate different medical imaging tasks, allowing for both detection and segmentation.

In scenarios where the task involves detection, such as pulmonary disease classification, the model is constructed using a sequence of blocks. The model terminates at the detection head, which is a fully connected layer with a number of output classes specific to the task. This design enables the model to focus solely on the detection aspect, efficiently capturing the necessary features for classification.

For segmentation tasks, the U-Net architecture is employed recursively, following the characteristic U-shape structure. Each down block is paired with an associated up-sampling block on the opposite side of the U, enabling precise localization and segmentation of structures within the images. This recursive implementation ensures the model can effectively leverage both local and global contextual information,

Table 6.1: Details of the blocks and operations searched in the benchmark.

Block	**Operations**	**Respective Upsampling block**
Zero	-	-
Identity	IdentityLayer	IdentityLayer
A	LinearLayer(in, out)	LinearLayer(out, in)
B	2DConv(in, out, k=3, use_bn=false, act=relu)	T2DConv(out, in, k=2, s=2)
C	2DConv(in, out, k=3, use_bn=true, act=relu)	T2DConv(out, in, k=2, s=2)
D	2DConv(in, out, k=3, use_bn=false, act=leakyrelu)	T2DConv(out, in, k=2, s=2)
E	2DConv(in, out, k=3, use_bn=true, act=leakyrelu)	T2DConv(out, in, k=2, s=2)
F	[B, C]	T2DConv(out, in, k=2, s=2)
G	[D, E]	T2DConv(out, in, k=2, s=2)
H	3DConv(in, out, k=3, use_bn=false, act=relu)	T3DConv(out, in, k=2, s=2)
3DConv(in, out, k=3, use_bn=true, act=relu)	T3DConv(out, in, k=2, s=2)	I
J	3DConv(in, out, k=3, use_bn=false, act=leakyrelu)	T3DConv(out, in, k=2, s=2)
K	3DConv(in, out, k=3, use_bn=true, act=leakyrelu)	T3DConv(out, in, k=2, s=2)
L	[H, I]	T3DConv(out, in, k=2, s=2)
M	[J, K]	T3DConv(out, in, k=2, s=2)

enhancing the accuracy and completeness of the segmentation results.

The number of down blocks is tuned and searched for each task. We define the maximum depth of the network to be 10 blocks. We define the same supernetwork for all tasks. This allows us to assess the generalization ability of the architectures.

Operator Selection

For the operations, we include commonly used operations in medical image segmentation tasks. These operations comprise 2D and 3D convolutions to capture spatial features at different scales, max pooling and average pooling to downsample feature maps and skip connections to enable effective information flow between the encoder and decoder sections.

To organize these operations, we group them into blocks, which serve as architectural choices at each layer of the supernetwork. Each block consists of multiple candidate operations, allowing the supernetwork to select and combine them during the architectural search process. By incorporating these blocks, the supernetwork gains the flexibility to adapt its architecture based on the specific requirements of the medical imaging task. It is important to note that we have not included kernel size tuning in our study. This decision is based on existing literature, which has demonstrated that the same receptive fields achieved with kernel sizes of 5 or 7 can be effectively obtained by combining multiple convolutions with a kernel size of 3. Table 6.1 details the list of blocks used the high-level architecture.

By using variants of the convolution, the search space offers flexibility based on the type of input data it can handle, allowing for the search and utilization of either a 3D U-Net or a 2D U-Net. When dealing with medical imaging, the input data can vary in dimensionality. For instance, 3D U-Net is designed to process volumetric data, such as CT scans or MRI volumes, where each pixel or voxel contains three spatial dimensions: width, height, and depth. On the other hand, 2D U-Net is suitable for analyzing single slices or images that possess only two dimensions: width and height. Figure 6.3 provides an overview of the architecture size and types incorporated into the search space. To ensure comprehensive coverage of different tasks, we define task-specific supernetworks within the search space. Each supernetwork is trained using

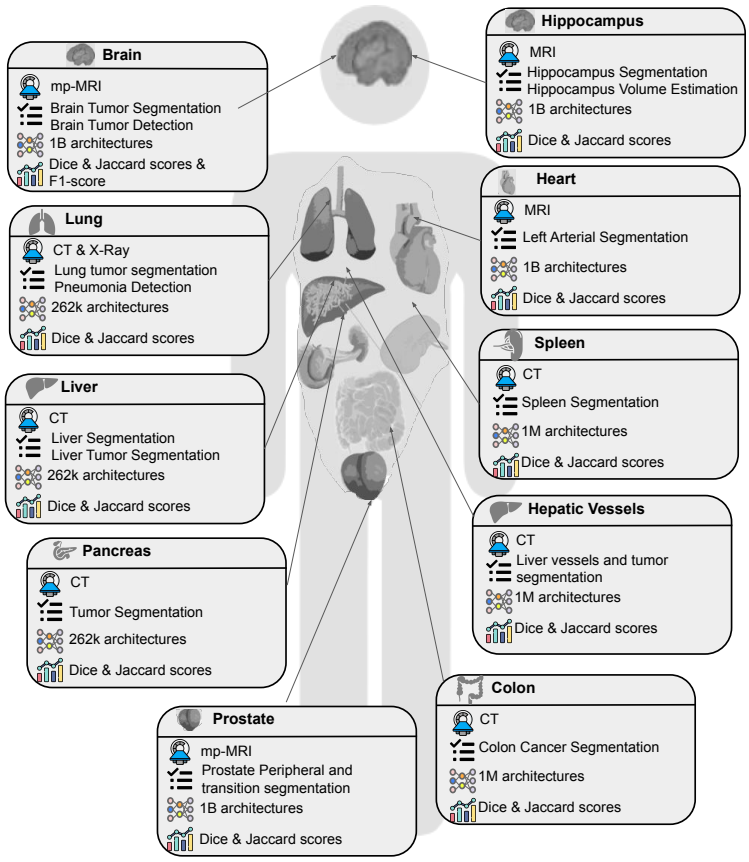

Figure 6.3: Overview of MED-NAS-Bench. Inspired from MSD [18]

a predefined loss function that aligns with the objectives of the respective task.

The search space for MRI data consists of 1,073,741,824 architectures, considering 15 different blocks. For CT scans involving lung, liver, and pancreas tasks, there are 262,144 architectures with 9 blocks. For CT scans of the mystery tasks with smaller datasets, the search space expands to 4,194,304 architectures, allowing for the use of the 3D convolution blocks. These diverse search spaces enable researchers to find optimal architectural configurations tailored to specific medical imaging tasks.

6.2.3 Evaluation methodology

For each dataset, a training hyperparameter tuning is achieved on 20 randomly sampled architectures from the supernetwork. The resulting must-used values are then used to train the general supernetwork for that task. Table 6.2 summarizes the hyperparameters for each task. Notice that jointly enumerating architectures and hyper-parameters will result in billions of architecture hyper-parameter pairs and is infeasible in practice. Therefore, we first optimize the hyper-parameters to a proper value which can accommodate different sub-networks in the same supernetwork.

Table 6.2: Datasets details and training hyperparameters.

Hyper-parameter	**Brain**	**Hippo-campus**	**Lung**	**Liver**	**Prost-ate**	**Heart**	**Panc-reas**	**Colon**	**Hepa-tic Ves-sels**	**Spleen**	**Chest**
Learning Rate	0.001	0.0005	0.001	0.001	0.0001	0.001	0.001	0.001	0.0005	0.001	0.001
Learning Rate Decay	0.9	0.95	0.95	0.9	0.95	0.9	0.95	0.9	0.95	0.9	0.95
Weight Decay	0.0001	0.0005	0.0005	0.0001	0.0005	0.0001	0.0005	0.0001	0.0005	0.0001	0.0005
Batch Size	16	8	32	16	16	32	16	32	8	16	32
Optimizer	Adam	RMS-prop	Adam	Adam	Adam	RMS-prop	Adam	Adam	RMS-prop	Adam	Adam
Loss Function	Dice Loss	BCE Loss	Dice Loss	Dice Loss	Dice Loss	Dice Loss	Dice Loss	Dice Loss	Dice Loss	Dice Loss	Cross-Entropy Loss
Evaluation Metric	Dice Coeff	Dice Coeff	Dice Coeff	Dice Coeff	Dice Coeff	Dice Coeff	Dice Coeff	Dice Coeff	Dice Coeff	Dice Coeff	f1-score
Dropout Rate	0.2	0.3	0.1	0.2	0.3	0.2	0.1	0.2	0.3	0.1	0.2
Number sampled subnet-works	100	250	150	150	150	100	150	150	150	150	250

Metrics During training each architecture, we record the following metric covering both model effectiveness and efficiency: train/validation/test loss value and evaluation metric at each epoch, the model latency, energy consumption, and the number of parameters. The targeted hardware with their respective specifications for the experiments is provided in table 6.3. For each task, the evaluation metrics are listed in figure 6.3. In all segmentation tasks (i.e., all tasks except chest x-ray disease detection), we use the dice and Jaccard scores.

Table 6.3: MED-NAS-Bench Hardware specifications.

Hardware	Processor	RAM	Storage	Operating System
Raspberry Pi 3	quad-core ARM Cortex-A53 CPU	1GB	32G	Raspbian
Laptop	AMD Ryzen 7 6800H	16GB	1T	Microsoft Windows 11

Dice Score

The Dice coefficient, also known as the Dice similarity coefficient or Dice index, measures the agreement between the predicted segmentation and the ground truth segmentation. It is computed as twice the intersection of the predicted and ground truth regions divided by the sum of their individual volumes. The Dice coefficient ranges from 0 to 1, where a value of 1 indicates a perfect overlap between the segmentations.

Jaccard Score

The Jaccard score, also referred to as the Jaccard index or Intersection over Union (IoU), is another measure of overlap between two sets. It is calculated as the intersection of the predicted and ground truth regions divided by the union of the two regions. Like the Dice coefficient, the Jaccard score ranges from 0 to 1, with a value of 1 indicating a perfect match between the segmentations.

Training hardware This project was granted access to the computing resources of the Institut du développement et des ressources en informatique scientifique (IDRIS) of the Centre national de la recherche scientifique (CNRS) [2] in France, which meant that model training was carried out on the Jean Zay computer cluster of IDRIS. Additionally, more training was done on DataCrunch [3]. The training of all supernetworks was achieved on NVIDIA A100 GPUs.

6.2.4 Performance Distribution

In this section, we carry out empirical analyses to gain insights for our proposed benchmark. All the following analyses are based on the average performances of three random seeds.

Overall Performance

Figure 6.4 presents a visualization of the performance distribution of 100k sampled architectures in each dataset. The performance metric depends on the task and is reflected in table 6.2. In general, a considerable number of architectures achieve satisfactory results across different datasets, but those with exceptionally strong performance are scarce. This strengthens the need for an efficient architecture search methodology for this benchmark. This result also indicates the possibility to generalize across medical tasks as the macro-architecture remains the same. Notably, the brain, pancreas, chest, and hepatic vessel tasks are considered more complex due to the intricate nature of the anatomical structures involved. Consequently, achieving optimal performance for these tasks poses a greater challenge. The boxplots for these

[2] http://www.idris.fr/
[3] https://cloud.datacrunch.io/

datasets may show wider interquartile ranges and outliers, indicating the presence of architectures with varying degrees of effectiveness.

In contrast, the hippocampus, heart, and spleen datasets are comparatively smaller in size, which can result in architectures that generalize well to the test data. Consequently, we observe narrower interquartile ranges and fewer outliers in their respective boxplots, indicating a higher consistency and performance of the architectures for these tasks.

Conversely, there is significant variation in the latency and energy consumption among the architectures, as depicted in Figure 6.5. To further investigate this aspect, we generated additional boxplots representing the latency and energy consumption specifically on Raspberry Pi3 and laptop hardware platforms.

Interestingly, while the performance of the architectures exhibits a wide range, the hardware efficiency, represented by latency and energy consumption, appears to be more constrained. This observation suggests that achieving high hardware efficiency can be a more challenging task when compared to achieving varied performance levels.

Upon analyzing the boxplots for different datasets, we observe that the heart, hippocampus, and brain tasks tend to have lower hardware efficiency with higher latency and energy consumption on both Raspberry Pi3 and laptop hardware. This can be attributed to several factors, including the larger image sizes and the use of MRI modality in these tasks, which often require more computational resources and result in increased hardware demands.

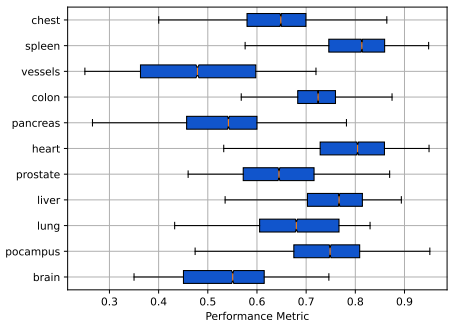

Figure 6.4: MED-NAS-Bench performance across datasets

Besides, Table 6.4 provides insights into the number of architectures that are deemed non-deployable across various hardware platforms. To determine this count on the laptop, we simulate a realistic scenario by considering the presence of pre-loaded tools such as 3D Slicer [4] and OHIF [5] in the memory. The large input sizes the brain, hippocampus, and heart datasets, induce a large number of non-deplyable architectures in both hardware platfroms. Besides, due to the small memory size of the Raspberry Pi3, a large number of the architectures is not deployable.

Ranking Evaluation

In this section, we conduct a comparison between our weight-sharing strategy, used to compute the performance of all sub-networks in the benchmark and an alternative approach of building a surrogate model. The objective is to assess the effectiveness of our weight-sharing strategy in preserving the ranking of architectures, as compared to independently trained architectures.

To evaluate the ranking correlation, we employ Kendall's tau correlation coefficient. This metric enables us to measure the extent to which the ranking of archi-

[4] 3D Slicer image computing platform: https://www.slicer.org/
[5] OHIF tool: https://ohif.org/

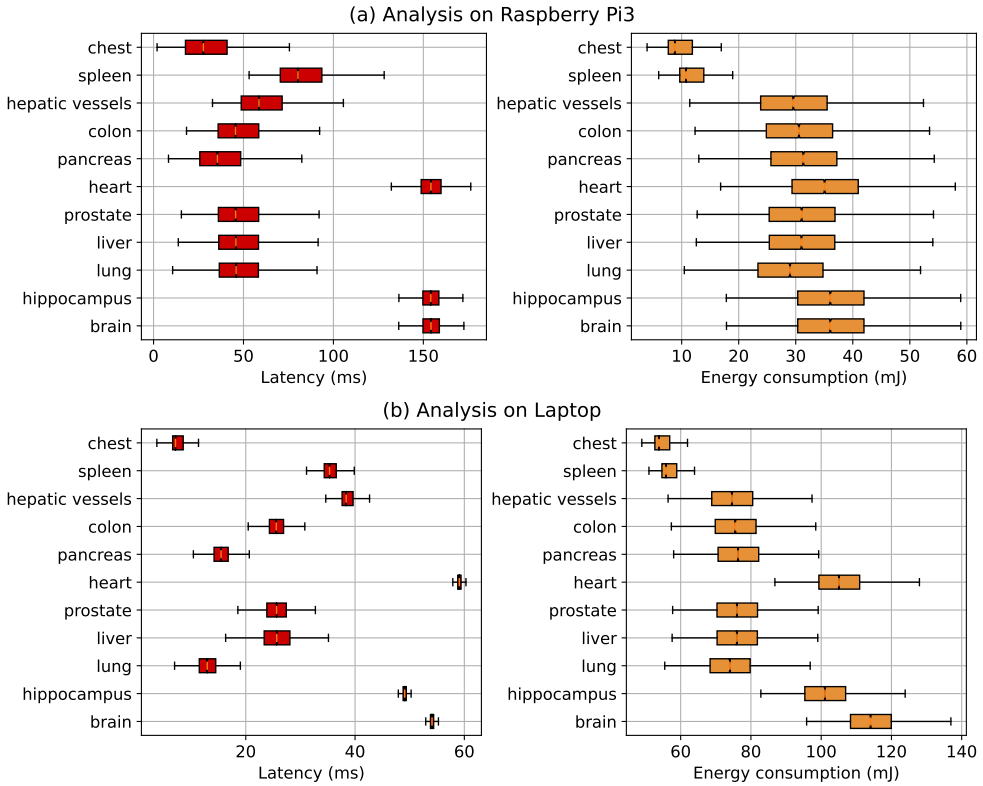

Figure 6.5: MED-NAS-Bench hardware efficiency across datasets on Raspberry Pi3 and Laptop.

Table 6.4: Number of Non Deployable architectures in edge platforms

datasets	Number of Non deployable architectures	
	Raspberry Pi 3	Laptop
brain	11204	2154
hippocampus	11204	2154
lung	5438	0
liver	5438	0
prostate	5438	0
heart	10342	1853
pancreas	5438	0
colon	5438	0
Hepatic Vessels	5438	0
Spleen	0	0
Chest	0	0

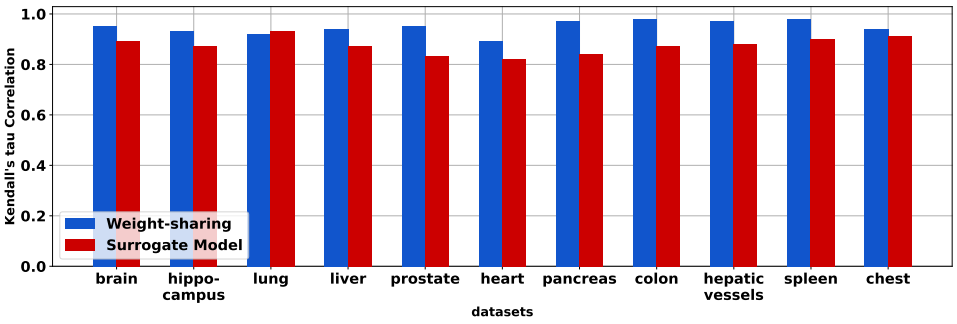

Figure 6.6: Ranking correlation experiments across datasets.

tectures obtained from our weight-sharing strategy aligns with the rankings of independently trained architectures. To obtain a robust evaluation, we independently trained 10,000 architectures for each dataset in the benchmark.

For the surrogate model approach, we employ the Gates architecture [16] as our surrogate model. The Gates architecture, based on Graph Convolutional Neural Networks (GCN), is a state-of-the-art surrogate model for single objective estimation. We trained the surrogate model using 3,000 architectures from the benchmark.

Upon analyzing the results, see Figure 6.6, we observed that our weight-sharing strategy consistently outperformed the surrogate model across all datasets. The rankings obtained from our weight-sharing strategy showed stronger preservation compared to the surrogate model. This outcome underscores the effectiveness and superiority of our weight-sharing strategy in capturing the architectural performance variations and preserving the ranking of architectures in the NAS benchmark.

6.2.5 Architecture Distribution

As described in Section 6.2.2, our benchmark design includes a macro architecture and a list of potential blocks for each layer. In order to gain insights into how different choices of operations contribute to the effectiveness of the models, we select the top 1000 architectures from each dataset and examine the frequency of block choices. These architectures are selected from a pool of 10,000 independently trained architectures to ensure meaningful insights.

The results are depicted in Figure 6.7. From the analysis, we observe the following trends.

First, in datasets where 3D convolution was listed as one of the available options, there is a low occurrence of these blocks in the top-performing architectures. For instance, in brain tumor segmentation, blocks (J), (K), (L), and (M) do not appear at all. This suggests that utilizing 2D convolutions on individual slices of the input volume is a more efficient approach for the given segmentation task. However, this may induce additional execution time and energy consumption for loading the slices and processing them sequentially, which is done in edge devices with low memory.

Second, similar to NAS for image classification tasks, the Identity block plays a significant role in enhancing the model's performance. This indicates that the presence of direct connections, such as skip connections, can contribute to improved feature representation and information flow within the network, leading to better segmentation results.

6.2.6 Cross-datasets Correlations

Figure 6.8 presents a heatmap illustrating the ranking correlation between different datasets within the search space. The ranking correlation measures how consistently

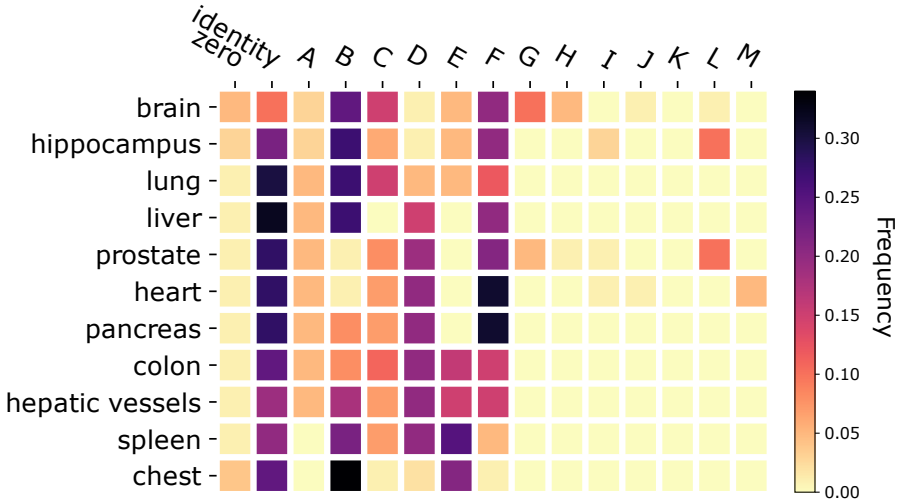

Figure 6.7: Blocks operation frequency in top 1000 architectures for each dataset.

the performance rankings of architectures across different datasets align with each other.

The observed pattern in the heatmap indicates that architectures tend to have similar rankings across datasets with the same modality. This means that architectures that perform well on MRI datasets are also likely to perform well on other MRI datasets, and likewise for CT datasets.

This finding highlights the importance of considering modality-specific characteristics when designing and optimizing architectures for medical image segmentation. It suggests that certain architectural configurations may be better suited for handling the unique imaging properties and challenges associated with a specific modality, such as MRI or CT.

6.2.7 State-of-the-art Search Methodologies

We present the outcomes of employing state-of-the-art search methodologies on our benchmark, which provide valuable baselines for the performance and effectiveness of the top-found architectures. We perform two experiments: (1) single-objective optimization, in which we consider only the task-specific performance, and (2) multi-objective optimization, where we simultaneously optimize task-specific performance, latency, and energy consumption.

Task-specific performance optimization

Recently, several works propose NAS methodologies for medical image segmentation. Specifically, MixSearch [304], C2FNAS [305], and BiX-NAS [306] stood out as state-of-the-art search methodologies.

MixSearch [304] is a method for searching for domain-generalized medical image segmentation architectures. It involves searching for generalizable U-shape architectures on a composited dataset that mixes medical images from different domains.

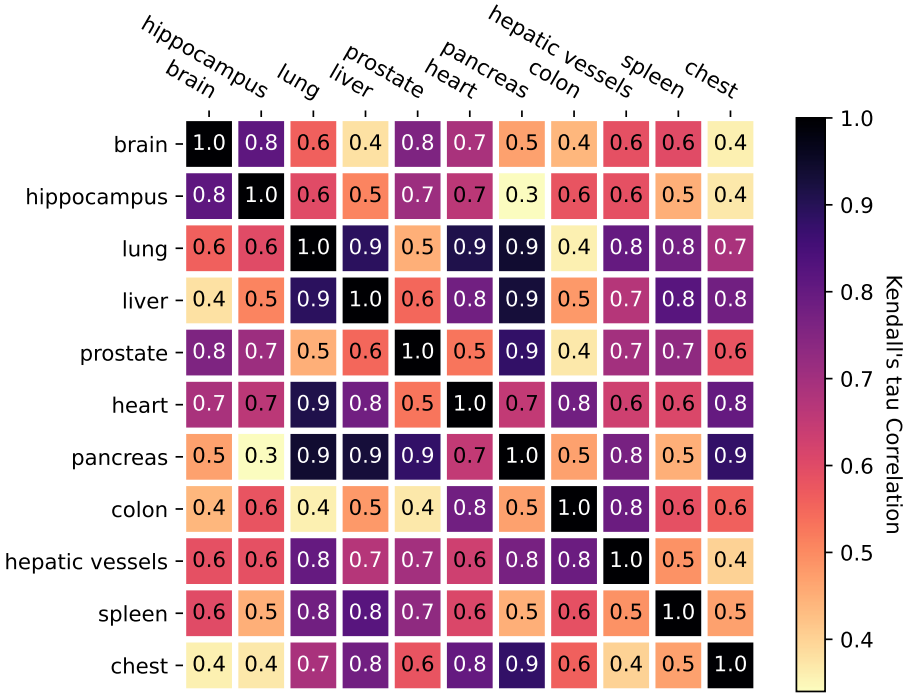

Figure 6.8: Cross-datasets ranking correlation

Their search space resembles the benchmark definition but contains fewer operations.

C2FNAS [305] is a coarse-to-fine NAS method for 3D medical image segmentation. The method involves searching for a coarse architecture first, followed by a fine architecture that refines the coarse architecture.

Similarly, BiX-NAS [306] involves a two-stage NAS methodology to reduce the network computational cost by sifting out ineffective multi-scale features at different levels and iterations.

Table 6.5 shows the results of state-of-the-art networks and search methodologies, described above on our benchmark. The metrics are defined in section 6.2.3. From the results obtained, several key observations can be made regarding single-objective optimization using state-of-the-art methods:

- The design of our benchmark, based on the U-Net macro-architecture commonly used in state-of-the-art search spaces, enables easy adaptation of various search methodologies to our benchmark, allowing for exploration of a larger number of architectures.

- Classical search algorithms, such as random search (RS) and evolutionary algorithms (EA), were found to be inefficient on our benchmark due to the extensive number of architectures. In our experiments, RS was executed with 1000 iterations per dataset, while EA utilized 1000 iterations, a population size of 100, and a mutation probability of 0.8. No crossover was applied, and the mutation operation involved switching one layer's block to another.

- The search process on our benchmark yielded improved architectures compared to the original architectures found and presented in previous papers. For instance, the C2FNAS architecture achieved superior performance on all tasks within our benchmark when compared to the original results reported in their respective paper. Other search methods, such as MixSearch and BiX-NAS do not provide architectures for these specific tasks considered in our benchmark.

Table 6.5: Results of state-of-the-art search methodologies on MED-NAS-Bench. Dice and Jc stand for the dice and Jaccard scores respectively. EA and RS stand for evolutionary algorithm and random search, both are classical search algorithms. *C2FNAS_O* is the original architecture proposed by C2FNAS.

Method	Brain		Hippocampus		Lung		Liver		Prostate	
	Dice (%)	Jc (%)	Dice (%)	Jc (%)	Dice (%)	Jc (%)	Dice (%)	Jc (%)	Dice (%)	Jc (%)
U-Net [301]	57.54	51.32	80.4	74.15	54.7	47.77	75.87	69.43	77.5	72.05
U-Net++ [302]	58.98	53.67	82.63	75.65	61.3	55.4	78.38	72.4	78.97	71.17
Att. U-Net [303]	62.4	57.4	83.24	76.14	65.7	59.54	74.6	70.28	77.78	70.74
nnU-Net [308]	61.20	54.68	89.66	85.23	69.2	62.36	84.48	78.89	82.7	78.4
C2FNAS_O [305]	61.98	55.76	88.67	82.24	70.44	63.73	83.94	79.3	81.82	74.9
EA	61.56	53.54	85.6	80.21	70.8	64.38	80.9	74.53	74.56	69.87
RS	53.5	47.4	62.45	54.85	56.7	51.21	67.88	61.1	68.4	61.94
MixSearch [304]	**65.78**	**59.65**	88.67	83.31	**81.3**	**76.08**	**87.43**	**82.23**	**86.79**	**79.79**
C2FNAS [305]	64.88	54.6	**90.54**	**83.19**	79.4	73.12	86.44	80.32	83.56	79.01
BiX-NAS [306]	63.87	56.4	89.68	84.07	75.6	68.47	87.12	82.11	81.5	75.61

Method	Heart		Pancreas		Colon		Hepatic Vessels		Spleen		Chest
	Dice (%)	Jc (%)	Dice (%)	Jc (%)	Dice (%)	Jc (%)	Dice (%)	Jc (%)	Dice (%)	Jc (%)	F1-score
U-Net	85.6	79.92	64.56	59.93	54.32	49.53	38.5	30.98	89.54	84.11	95.32
U-Net++	84.32	77.89	63.87	57.03	59.82	53.64	48.93	44.49	88.95	84.6	95.38
Att. U-Net	85.78	78.95	64.76	58.41	45.7	37.91	56.73	49.69	90.56	83.66	95.78
nnU-Net	92.77	88.39	65.9	58.47	56	51.55	66.08	59.54	96	91.34	96.8
C2FNAS_O	92.49	88.1	67.59	60.68	58.9	53.7	67.65	63.54	96.28	89.49	96.34
EA	85.76	79.84	65.3	59.64	50.8	43.23	55.78	51.07	89.76	84.29	95.78
RS	75.3	69.19	54.21	46.27	46.7	41.22	39.76	35.41	80.56	73.83	89.56
MixSearch	89.53	84.73	68.43	63.2	57.8	52.1	**71.65**	**65.12**	96.75	92.27	97.54
C2FNAS	**94.56**	**89.24**	67.82	60.18	**60.67**	**53.85**	65.42	57.46	**97.34**	**93.16**	**98.68**
BiX-NAS	94.32	86.98	**69.84**	**63.66**	57.59	50.23	66.78	61.47	96.76	90.07	96.83

- It is worth noting that no single search methodology consistently outperformed the others across all tasks within our benchmark. The performance of different search methodologies varied depending on the specific dataset and task at hand.

Multi-objective Optimization

The current medical NAS methodologies primarily focus on single-objective optimization, neglecting the consideration of multiple objectives during the search process. This limitation prompted us to explore and establish baselines for multi-objective optimization using classical optimization algorithms. In this section, we employ three well-known algorithms: NSGA-II [309], EEEA [310], and BANANAS [170]. These algorithms are described in detail in section 2.6.1.

To evaluate the performance of these algorithms, we present the obtained Pareto fronts in Figure 6.9 and Figure 6.10. The Pareto front represents a set of architectures that achieve a trade-off between different objectives. Each point on the Pareto front represents an architecture that is non-dominated by any other architecture, indicating the best achievable performance for the given objectives. The Pareto fronts provide valuable insights into the trade-offs between task-specific performance, latency, and energy consumption in the MED-NAS-Bench.

The obtained Pareto fronts demonstrate the effectiveness of the employed multi-objective optimization algorithms on the MED-NAS-Bench for various datasets. While the exact Pareto front is difficult to compute due to the large size of the benchmark, we observe that the methods consistently achieve near Pareto front optimality. This indicates that the algorithms successfully explore the trade-offs between task-specific performance, latency, and energy consumption, providing a diverse set of architectures with different trade-off profiles for each dataset. These results highlight the efficacy of searching for efficient medical analysis architectures.

Table 6.6 presents the hypervolume values obtained from the multi-objective op-

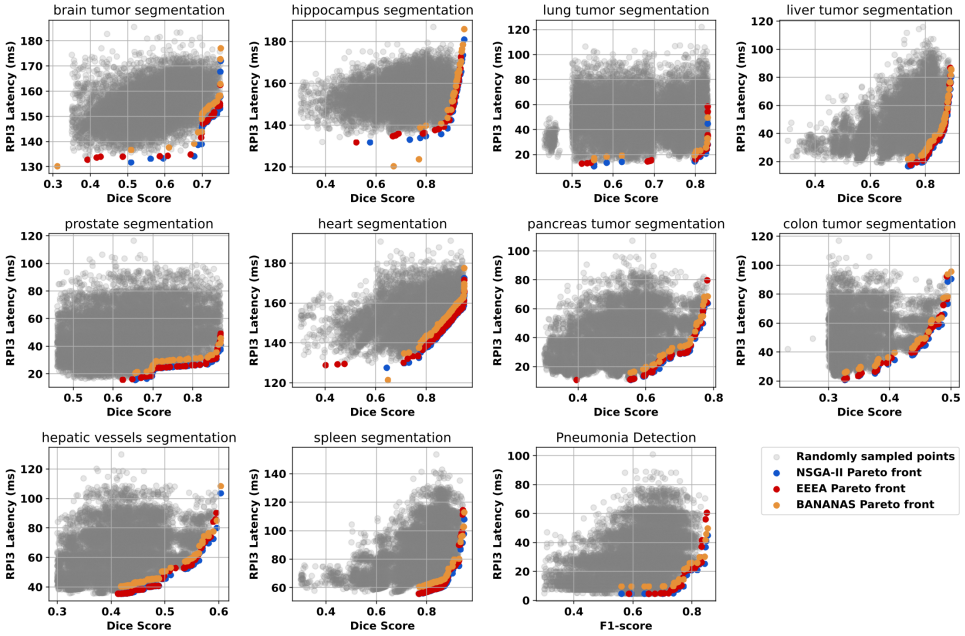

Figure 6.9: Pareto front results of SOTA multi-objective optimizations on Raspberry Pi3.

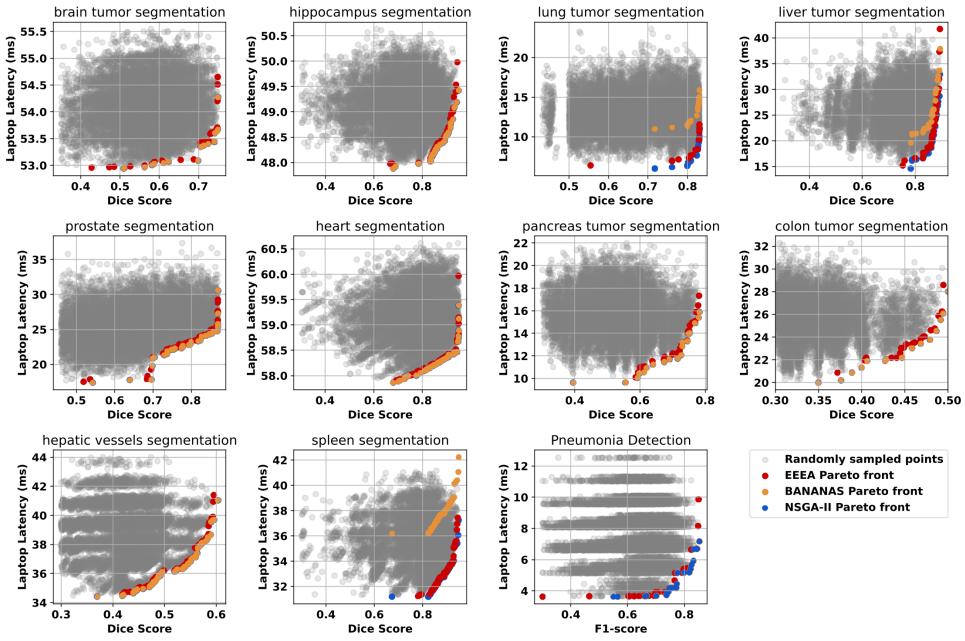

Figure 6.10: Pareto front results of SOTA multi-objective optimizations on Laptop.

timization of task-specific performance, latency, and energy consumption simultaneously. The hypervolume is a measure that quantifies the extent to which the obtained Pareto front dominates the reference point, which is defined as the maximum values of latency and energy consumption, and the minimum values of the dice score or f1-score. The maximum and minimum values are computed using the sampled architectures. We gain insights into the overall quality and coverage of the Pareto front achieved by the optimization algorithms. A higher hypervolume value indicates a larger region of the objective space covered by the Pareto front, indicating better trade-off solutions between the conflicting objectives. Among the tested methods, BANANAS consistently demonstrated competitive results across the evaluated datasets. In particular, BANANAS outperformed the other methods, especially when considering the Raspberry Pi3 hardware platform.

Table 6.6: Hypervolume values of multi-objective optimization considering performance, latency, and energy consumption.

Method	Raspberry Pi3			Laptop		
	NSGA-II	EEEA	BANANAS	NSGA-II	EEEA	BANANAS
Brain	628.719	601.26	**716.22**	2086.93	2031.8	**2133.58**
Hippocampus	643.16	499.2	**762.53**	**2674.93**	2304.21	2512.42
Lung	719.43	764.56	**869.48**	**5011.25**	4875.3	4971.15
Liver	526.8	437.46	**640.46**	**4639.42**	4346.75	3958.43
Prostate	808.72	869.85	**908.37**	4649.85	4692.54	**4752.32**
Heart	706.79	688.89	**766.02**	3068.7	**3575.2**	3463.25
Pancreas	885.23	849.54	**1001.78**	**3710.92**	3156.6	3185.71
Colon	866.44	815.41	**883.45**	2278.75	2563.67	**2716.87**
Hepatic Vessels	761.44	789.46	**817.18**	2786.02	3567.43	**3695.33**
Spleen	633.04	612.39	**802.5**	**2103.35**	2034.67	1942.85
Chest	479.85	441.02	**651.13**	1863.17	2314.66	**2648.43**

6.3 MT-MIAS

In this section, we introduce a novel approach for multi-task medical NAS. Unlike traditional approaches that focus on single-task optimization, our methodology aims to simultaneously optimize multiple medical tasks within a unified framework, considering hardware efficiency at the same time. By leveraging the inherent relationships and the shared information among different tasks, our approach offers improved efficiency and effectiveness in the architecture search for medical applications. We describe the key components and strategies employed in our multi-task NAS methodology and present experimental results showcasing its performance on DARTS [166] and MED-NAS-Bench.

6.3.1 Search Methodology

In the context of weight-sharing NAS, two categories of methods have emerged: gradient-based optimization and evolutionary-based approaches. In gradient-based optimization, the training of the supernetwork and the search for an optimal architecture are performed simultaneously. However, it is not without its limitations. Firstly, this approach often incurs a high computational cost due to the simultaneous optimization of weights and architecture parameters. The iterative nature of the optimization process can be time-consuming, especially for complex datasets and large-scale architectures. In addition, these methods suffer from gradient instability which has been discussed in section 3.3. On the other hand, evolutionary-based approaches utilize evolutionary algorithms to iteratively explore and optimize the supernetwork by leveraging shared weights to estimate performance.

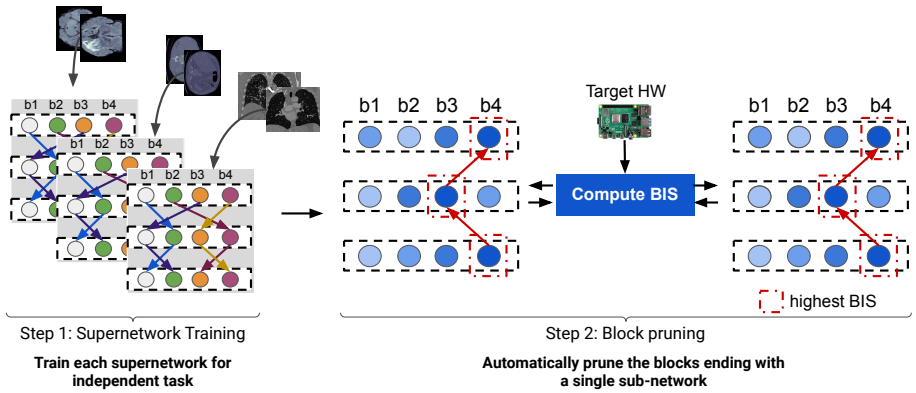

Figure 6.11: Overview of MT-MIAS steps.

Our methodology simplifies this latter approach, specifically focusing on optimizing the second stage, which is the search methodology. By doing so, we streamline the search process and eliminate the need for modifications to the search algorithm methodology when utilizing any supernetwork search space. This makes our methodology adaptable and applicable to various deep-learning tasks. In this section, we validate our methodology using DARTS [166] for image classification and MED-NAS-Bench for medical image segmentation. However, the same methodology can be easily extended to other tasks by incorporating suitable task-specific datasets and evaluation metrics.

Overview

Figure 6.11 provides an illustration of the overall search mechanism employed by MT-MIAS. Rather than relying on traditional optimization strategies, our approach draws inspiration from pruning methodologies found in the literature. Specifically, we leverage the concept of neuron importance introduced in NISP [311]. This approach assigns an importance score to each neuron in a layer and prunes those that do not significantly impact accuracy.

In MT-MIAS, we extend this idea by introducing the concept of block importance score (BIS). For each block within a layer in a supernetwork, a BIS is computed. The BIS serves as a measure to evaluate the importance of a block in achieving the final performance. By iterating through the supernetwork in a backward manner, we can identify and retain only the most crucial block for each layer based on their respective BIS.

To incorporate hardware efficiency into the search process, we consider the latency of the blocks, ensuring that the selected sub-network remains efficient in terms of computational requirements. Additionally, we introduce a generalization factor that allows us to select a common block when iterating over multiple supernetworks, promoting a more generalized architecture suitable for multi-task learning.

By combining the block importance scores, hardware efficiency considerations, and generalization factor, we can identify the most efficient and effective sub-network within the supernetwork.

Objective function

Our goal in the MT-MIAS methodology is to determine which block in layer l of the supernetwork is the most influencing the performance of the network. This determination is made after the supernetwork has been fully trained.

We define the block importance score as a non-negative value w.r.t. a block and use s_l to represent the vector of block importance scores in the l-th layer. We define the final layer indicator of the l-th layer as a binary vector s_l^*, computed based on block importance scores s_l such that $s_{l,b}^* = 1$ if and only if $s_{l,b}$ is the top value in s_l. b represents the block index. To increase efficiency, we select from each layer a block, following the objective function below:

$$s_{l,b}^* = \begin{cases} 1, & arg\max_{s_l} s_{l,b} * Avg(\alpha(a_{b \in a}))/LAT(b). \\ 0, & else. \end{cases} \tag{6.1}$$

Here, $s_{l,b}$ is the importance score of block b in layer l, α is a function that computed the validation performance of an architecture, $a_{b \in a}$ is the set of sub-networks that contains block b in their paths, LAT computes the latency of a block. In this function, we set the index of the block b to 1, if the block maximizes the importance score and minimizes its latency. The solution to Equation 6.1 indicates which block should be selected in an arbitrary layer.

The problem of defining s_l^* can be formulated as a binary integer program, finding the optimal block to select in Equation 6.1. However, it is hard to obtain efficient analytical solutions by directly optimizing Equation 6.1. NISP [311] derived an upper bound on this objective, and show that a sub-optimal solution can be obtained by minimizing the upper bound on a simple fully connected network to achieve pruning. Interestingly, their formulation is still valid in the case of supernetwork block pruning as we intend to achieve. The theoretical validity of NISP's methodology in the case of supernetwork block pruning lies in the consistent underlying principles and the extension of importance scoring to blocks. Considering that a block is a sequence of layers, we can adapt NISP's methodology by focusing on the last convolutional layer of each block. By doing so, we effectively consider the impact of the entire block on the overall network performance.

To enhance generalization across tasks, we define $s_{t,l,b}$ as the importance score of the b in layer l of the supernetwork trained for task t. We then add to Equation 6.1, a regularization term that includes the generalization ability of block b, denoted as $g_{T,l,b}$, where T is the set of tasks we are targeting. Equation 6.2 formulates the generation score.

$$g_{T,l,b} = \sum_{t \in T} s_{t,l,b} \tag{6.2}$$

This term sums the block importance score over multiple subnetworks. We can then identify the most generalizable block by maximizing this term. In our optimization function, we directly include this objective in the maximization of Equation 6.1.

Block Importance Score (BIS)

To quantify the importance of each block within layer l, we introduce the block importance score (BIS), denoted as $s_{l,b}$. The BIS is calculated as Equation 6.3:

$$s_{l,b} = |W^{(l+1)}|^T s_{l+1,b} \tag{6.3}$$

Here, $W^{(l+1)}$ represents the weights of the last convolutional layer in block b of layer $l+1$. The transpose operation, denoted by ^{T}, is applied to the vectorized weights to align the dimensions for computation. The vector s_{l+1} represents the importance scores of blocks in layer $l+1$ obtained during the previous stage of the search mechanism.

The BIS computation involves the magnitude of the weights $W^{(l+1)}$ and the importance scores s_{l+1} from the subsequent layer. This approach takes into account the influence of each block's weight in the subsequent layer and the importance of

the blocks themselves. It captures the contribution of a block in layer l towards the overall performance of the supernetwork.

We compute s_n, i.e., the final layer's blocks importance score, considering the importance of each weight based on their contribution to the overall loss. Mathematically, the computation of the initial block importance score can be expressed as follows:

$$s_{n,b} = |W^{(n,b)}| \cdot \left|\left| \frac{\partial L}{\partial W^{(n,b)}} \right|\right|_F \tag{6.4}$$

Here, L represents the loss function, and w_i denotes each weight in the last blocks of the supernetwork. The partial derivative $\frac{\partial L}{\partial w_i}$ represents the gradient of the loss with respect to the individual weights. $_F$ denotes the L^2 norm.

Backward Selection Algorithm

We describe the Backward Selection Algorithm used in MT-MIAS to select the most influential blocks within the supernetwork. The algorithm iteratively evaluates the block importance scores ($s_{l,b}$) and selects the most important block in each layer, facilitating the construction of a generalized architecture.

Algorithm 7 Backward Selection Algorithm

Input: Supernetworks for multiple tasks with trained weights
for l in reverse order of layers **do**
  Initialize empty set s_l
  for each task supernetwork **do**
    Compute block importance scores $s_{l,b}$ using Equation 6.3
    Add $s_{l,b}$ to s_l
  end for
  Compute latencies of all blocks
  Compute validation performance α
  Optimize objective function using Equation 6.1
  Select block across all tasks as the most important block in layer l
end for
Return Selected blocks for each layer

The Backward Selection Algorithm now considers multiple supernetworks, each corresponding to a different task. It starts by initializing the block importance scores ($s_{l,b}$) for all blocks in each layer l. These scores serve as an initial estimation of the importance of each block.

Next, the algorithm iterates through the layers in reverse order. For each layer, it initializes an empty set S to store the block importance scores computed across all task supernetworks. It then iterates over each task supernetwork and computes the block importance scores using Equation 6.3. The resulting scores are added to set S.

During each iteration, the algorithm identifies and selects the block with the highest block importance score ($s_{l,b}$) across all tasks as the most influential block in that layer. By considering the scores from multiple supernetworks, the algorithm captures the importance and relevance of each block across different tasks.

Finally, the algorithm returns the selected blocks for each layer, providing a pruned architecture that retains the most important and influential blocks within the supernetwork, considering the requirements of multiple tasks.

To optimize the computation of validation accuracy used in the objective function of each layer for multiple sub-networks while avoiding duplicates, we employ an efficient strategy that significantly reduces computational redundancy. Instead of individually evaluating the accuracy of each sub-network, we leverage a caching

mechanism to store and reuse intermediate results. At the beginning of the optimization process, we initialize an empty cache. As we traverse through different sub-networks, we check if a particular configuration has been encountered before by querying the cache. If the configuration exists in the cache, we retrieve the precomputed accuracy without the need for redundant computations. This approach eliminates the repetitive evaluation of identical or similar sub-networks, leading to substantial computational savings.

6.3.2 Experiments Methodology

To evaluate the performance and effectiveness of MT-MIAS, we conducted experiments using multiple scenarios: MIAS, MT-MIAS, and MT-MIAS-C. Each scenario explores different aspects of the methodology and aims to achieve specific objectives.

- **MIAS Scenario:** In this scenario, the objective does not include the generalization term and the optimization is performed on each task's supernetwork independently. The purpose of this scenario is to compare the performance of MT-MIAS with the traditional single-task architecture search approaches.
- **MT-MIAS Scenario:** This scenario includes the generalization term in the objective function and aims to discover a generic architecture that can perform well across multiple tasks. The objective is to find blocks that are influential and contribute to improved performance across various tasks. The generalization term helps to guide the search toward a more versatile and adaptable architecture.
- **MT-MIAS-C Scenario:** In this scenario, we relax the objective definition to allow the selection of multiple blocks per layer. The goal is to construct a supernetwork that is completely deployable, ensuring a balance between performance and model complexity. To achieve this, we introduce a constraint that checks, at each iteration, whether the number of parameters of the selected blocks in each layer exceeds a certain threshold. This threshold is defined based on the number of parameters of the U-Net layers.

For the tasks, we selected the DARTS [166] and MED-NAS-Bench benchmarks. Both benchmark defines a supernetwork model. In the DARTS benchmark, we aimed to find a general architecture that performs well for ImageNet and CIFAR-10 classification tasks. Once the general architecture is discovered, we aim to further generalize it for CIFAR-100 classification. This allows us to evaluate the transferability and scalability of the architectures discovered by MT-MIAS. In the MED-NAS-Bench benchmark, following the approach of MSD [18], we retained the same unseen tasks proposed by MSD, along with the pneumonia detection task. To enable the search for a general architecture, we pruned the non-common paths in the network. This ensures that the search focuses on identifying blocks that are relevant across multiple medical imaging tasks.

Training Hyperparameters The hyperparameters used to train the supernetwork in MED-NAS-Bench are listed in Section 6.2.3. For DARTS, Table 6.7 summarizes the training hyperparameters. For ImageNet, DARTS supernetwork was already trained.

Hardware Setup We conduct the experiments on Raspberry Pi3. The same one was used to construct MED-NAS-Bench. The search, however, is achieved on a much more compute-intensive setup. Our search was conducted using an NVIDIA GPU 3070, a high-performance graphics processing unit known for its advanced parallel computing capabilities. The GPU was connected to a powerful workstation equipped with an Intel Core i9 processor and 32 gigabytes of RAM, ensuring sufficient computational resources for the search process.

Table 6.7: Hyperparameters for Training DARTS on CIFAR-10 and CIFAR-100

Hyperparameter	**CIFAR-10**	**CIFAR-100**
Learning Rate	0.1	0.05
Batch Size	128	64
Number of Training Epochs	200	200
Weight Decay	3e-4	3e-4
Optimizer	SGD	SGD
Momentum	0.9	0.9
Learning Rate Schedule	Step-wise decay	Cosine Annealing
Learning Rate Decay	0.1 at epochs 100	-

6.3.3 Search Results

The search results provide an evaluation of the MT-MIAS methodology on different datasets and tasks. In this section, we present the search results for DARTS, MED-NAS-Bench, and the generalization performance to unseen datasets.

DARTS Search Results

Table 6.8 presents a comprehensive summary of the search results obtained for ImageNet and CIFAR-10 using various methodologies. Our approach, MT-MIAS, demonstrates superior performance compared to current state-of-the-art methodologies in the DARTS search space. Notably, employing the MIAS methodology, which generates independent architectures for each dataset, we were able to identify architectures that strike a favorable balance between accuracy and latency. However, these architectures did not surpass the latency performance achieved by our PRP-NAS methodology. Interestingly, we found that the architecture discovered by MIAS aligns with the Pareto front identified by PRP-NAS, where PRP-NAS utilizes a Pareto score to optimize the search for optimal trade-offs. To delve into further details on PRP-NAS, we encourage readers to explore Section 3.3.

In comparison, MT-MIAS provides a shared architecture for both CIFAR-10 and ImageNet, with a minimal accuracy drop of 2.56% and 0.1% for ImageNet and CIFAR-10, respectively. Nonetheless, MT-MIAS successfully identifies state-of-the-art architectures, outperforming existing approaches. Notably, the most performing architecture is discovered by MT-MIAS-C, where we relax the problem definition to allow the selection of a block per task per layer under the constraint of the number of parameters. This relaxation enables us to construct a deployable supernetwork on Raspberry Pi3. The resulting supernetwork consists of two paths, one for CIFAR-10 and another for ImageNet, which can be intertwined by utilizing a single block for both datasets. This variant exhibits superior accuracy compared to existing methodologies and other variants.

Table 6.8: Comparison results of MT-MIAS against state-of-the-art methodologies in DARTS. Lat corresponds to the Raspberry PI3 Latency.

Methods	ImageNet			CIFAR-10			Hardware aware
	Acc (%)	Lat (ms)	GPU days	Acc (%)	Lat (ms)	GPU days	
DARTS [166]	73.3 ± 0.03	78.34	4	68.3 ± 0.08	45.36	4	No
ProxylessNAS (GPU) [235]	75.1 ± 0.00	74.64	8.3	92.89 ± 0.16	35.8	0.16	Yes
PC-DARTS [312]	75.5 ± 0.1	83.55	10	90.89 ± 0.08	41.8	0.21	No
P-DARTS [313]	72.22 ± 0.16	83.4	4	92.51 ± 0.9	34.15	10	No
PRP-NAS	77.5 ± 0.02	**75.3**	3.8	93.68 ± 0.05	40.7	2	Yes
MIAS	81.43 ± 0.01	75.75	2	93.97 ± 0.06	42.53	0.87	Yes
MT-MIAS	78.87 ± 0.07	82.35	2	93.87 ± 0.04	**35.27**	0.87	Yes
MT-MIAS-C	**82.4 ± 0.08**	84.32	2	**94.43 ± 0.12**	40.83	0.93	Yes

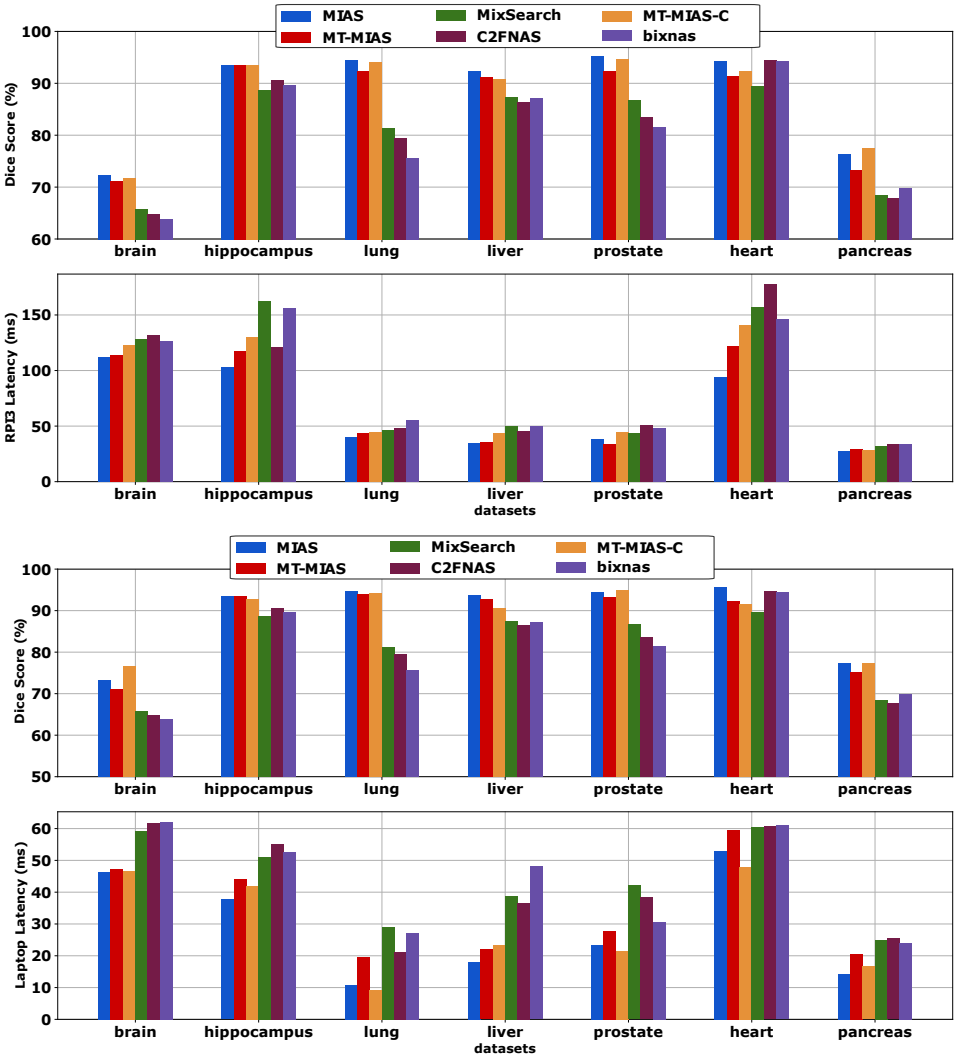

Figure 6.12: Comparative results of MT-MIAS on MED-NAS-Bench, both on Raspberry PI3 (RPI3) and laptop.

MED-NAS-Bench Results

Our experimental results on MED-NAS-Bench showcase the superiority of our methodology compared to state-of-the-art search algorithms, particularly when targeting segmentation tasks on Raspberry Pi3 and laptop hardware platforms. Similarly to DARTS results, we can conclude that our *MIAS* variant shows a nice trade-off between dice score and latency on both hardware platforms. Moreover, our *MT-MIAS* methodology yields a single architecture that surpasses existing works in terms of segmentation performance and holds potential for deployment on edge devices.

Additionally, our *MT-MIAS-C* variant presents a remarkable breakthrough by delivering a state-of-the-art supernetwork designed to tackle a wide range of segmentation tasks. The performance of this variant surpasses all other variants in terms of the dice score, demonstrating its ability to achieve highly accurate segmentation results across diverse medical imaging datasets.

Table 6.9: Results on unseen datasets of MED-NAS-Bench. (T) means that the architecture was fine-tuned for the target task.

Method	**Colon**	**Hepatic Vessels**	**Spleen**	**Chest**	Hardware
	Dice (%)	Dice (%)	Dice (%)	F1-score	aware
U-Net [301]	54.32	38.5	89.54	95.32	-
U-Net++ [302]	59.82	48.93	88.95	95.38	-
Att. U-Net [303]	45.7	56.73	90.56	95.78	-
nnU-Net [308]	56	66.08	96	96.8	-
C2FNAS [305]	58.9	67.65	96.28	96.34	No
EA	50.8	55.78	89.76	95.78	No
RS	46.7	39.76	80.56	89.56	No
MixSearch [304]	57.8	71.65	96.75	97.54	No
C2FNAS [305]	60.67	65.42	97.34	98.68	No
BiX-NAS [306]	57.59	66.78	96.76	96.83	No
MT-MIAS	55.46	60.98	88.56	87.45	Yes
MT-MIAS-C	56.35	60.45	93.61	86.77	Yes
MT-MIAS (T)	63.45	70.54	95.66	93.45	Yes
MT-MIAS-C (T)	64.5	68.7	97.65	98.65	Yes

Table 6.10: Results on CIFAR-100 (Unseen dataset for DARTS). (T) means that the architecture was fine-tuned for the target task.

Methods	CIFAR-100
	Acc (%)
DARTS	68.3
ProxylessNAS (GPU)	69.54
PC-DARTS	67.35
P-DARTS	70.46
PRP-NAS	87.65
MT-MIAS	68.43
MT-MIAS-C	68.95
MT-MIAS (T)	89.34
MT-MIAS-C (T)	86.87

Unseen datasets and Generalization

To thoroughly assess the generalization capabilities of our methodology, we conducted comprehensive experiments on a set of previously unseen datasets, including CIFAR-100, Colon, Hepatic Vessels, Spleen, and Chest X-Ray. The architectures discovered by both the *MT-MIAS* and *MT-MIAS-C* variants were specifically trained and evaluated on these datasets to determine their performance in novel settings.

To present the outcomes of our evaluation, we provide the results for the CIFAR-100 and MED-NAS-Bench unseen tasks in Table 6.10 and Table 6.9, respectively. We examined two variants of the architectures. Firstly, we tested the found architectures without any additional fine-tuning, apart from modifying the last layer to accommodate the specific number of classes in each dataset. Secondly, the architectures denoted with (T) underwent fine-tuning on the respective tasks, leading to improved accuracy.

Remarkably, our fine-tuned models surpassed existing search methodologies across all unseen datasets. Even without fine-tuning, the architectures demonstrated commendable performance, emphasizing their inherent ability to generalize well to novel datasets and tasks.

6.4 Conclusion

In this chapter, we introduced MED-NAS-Bench, a NAS benchmark that serves as a fundamental tool for evaluating and comparing different architecture search algorithms. MED-NAS-Bench provides a comprehensive set of diverse medical imaging tasks, allowing researchers to test the performance and generalization capabilities of their methodologies in the medical domain. This benchmark is a significant contribution to the field, as it fills the gap in evaluating NAS algorithms specifically for medical applications.

Furthermore, we presented our novel methodology, MT-MIAS, which leverages the power of MED-NAS-Bench for multi-task architecture search. By utilizing the rich dataset of medical imaging tasks, MT-MIAS aims to discover architectures that exhibit high performance, efficiency, and generalization across multiple medical imaging tasks. We described the key components and techniques employed in MT-MIAS, including the block importance score computation, backward selection algorithm, and regularization for generalization.

Through extensive experiments and evaluations on MED-NAS-Bench, we demonstrated the effectiveness and superiority of MT-MIAS compared to existing methodologies. Our approach achieved state-of-the-art results in terms of accuracy and latency, showcasing its potential to advance the field of multi-task architecture search in the medical domain.

The availability of MED-NAS-Bench and the success of MT-MIAS contribute to the broader research community by providing a benchmark dataset and a powerful methodology baseline for exploring and optimizing architectures in the medical imaging domain. These contributions open up new possibilities for developing efficient and effective models that can have a significant impact on medical diagnosis, treatment, and patient care.

Chapter 7

Conclusion and Future Work

Contents

7.1 Conclusion

In this thesis, an innovative approach for designing and developing Hardware-aware Neural Architecture Search (HW-NAS) for edge computing is presented. The objective was to optimize HW-NAS specifically tailored for edge devices, considering the hardware constraints and limitations associated with such computing environments. This thesis work gave answers to the following research questions:

1. What are the key components of hardware-aware neural architecture search, and how can they be optimized to improve performance?
2. How can multi-objective and Pareto-aware surrogate models be developed to enhance the evaluation components of HW-NAS?
3. How can search spaces be enhanced with quantization awareness and free from humanly designed operators to improve the search process?
4. Furthermore, how can hardware-aware neural architecture search be applied to novel hardware platforms, such as analog in-memory computing?
5. How can it be used to optimize benchmarks in medical imaging analysis?
6. What are the key considerations and methodologies for developing a comprehensive benchmark specifically tailored for evaluating Neural Architecture Search (NAS) methods in medical imaging, and how can such a benchmark be designed to effectively capture the complexities and challenges presented by medical imaging datasets?
7. How can a multi-task Neural Architecture Search (NAS) methodology be developed to effectively account for the diverse range of medical imaging types and tasks?

In Chapter 2, an extensive survey is conducted to propose a novel taxonomy for HW-NAS techniques, providing a comprehensive analysis of its different components, comparing evaluation techniques, and highlighting its limitations and challenges.

Upon this review, we pinpointed the following areas of research that need improvements: (1) the estimation methods that were used target only a single objective, and

yet in the context of HW-NAS multiple objectives have to be considered, such as accuracy, latency, and energy consumption. (2) the restrictive search space used by existing methods was holding back HW-NAS from finding innovative architectures. (3) Existing HW-NAS focus on theoretical DL tasks such as image classification. The lack of a standardized benchmark is the main reason behind it. To enable HW-NAS on real-world problems and different tasks, one needs to develop such benchmarks.

Chapter 3 addresses the first research question by proposing a novel multi-objective surrogate model, called *HW-PR-NAS*. Through the proposed Pareto score definition, our model was able to predict a ranking in terms of trade-offs between the different targeted objectives. It has been determined that when used with a classical evolutionary algorithm, our model outperforms existing work, finding 98% near Pareto front solutions on multiple cell-based search spaces.

As HW-PR-NAS has not been intially designed for Supernetwork search spaces, it requires additional computational complexity. We then proposed a Pareto rank preserving supernetwork training methodology, named *PRP-NAS*, to fill this gap. PRP-NAS leverage the Pareto score definition directly during the training of the supernetwork. PRP-NAS shows a 97% near Pareto front approximation in less than 2 GPU days of search, which provides 2x speed up compared to state-of-the-art methods. We validate our methodology on NAS-Bench-201, DARTS, and ImageNet.

In Chapter 4, the second and third research questions regarding HW-NAS search space were addressed. In edge device deployment, DL models undergo a series of optimizations to shrink their model size. The main used optimization is quantization. However, we need first to answer the following questions: which layer to quantize? and to which degree it should be tuned?. To this end, combining the search for a quantization scheme and architecture is extremely challenging. By doing so, the search space size becomes impractical. We thus presented Compression-aware Neural Architecture Search (CaW-NAS). CaW-NAS extends the search space during the search with architectures that are worth quantizing, i.e., that does not induce a large accuracy drop. CaW-NAS model improves state-of-the-art with 1.33 acceleration and an accuracy of 75.22% on ImageNet.

NAS search spaces are still humanly biased, which can lead to use over-parameterized components. To eliminate human design bias in the search space, we presented Grassroots Operator Search (GOS). GOS leverages the use of mathematical equations to replace common operations such as convolution, batch normalization, and activation functions with more efficient ones, resulting in models that are optimized for low-power edge devices. We demonstrated the effectiveness of our approach through experiments on popular architectures, including ResNet18, InceptionV3, and MobileNetV3, achieving significant improvements in inference time and energy consumption compared to the original models. Additionally, GOS has been validated on a real-world healthcare problem, namely pulse rate estimation, in which we present a 2x speedup with a 0.12 Mean Average Error (MAE) drop.

Chapter 5 addresses research question 5, where we presented HW-NAS approach targeting Analog In-Memory Computing (IMC). IMC is a novel computing paradigm that has the potential to revolutionize edge computing by enabling highly efficient and resilient computations directly within memory cells, thereby significantly improving the performance and energy efficiency of edge devices. We proposed an efficient HW-NAS methodology dedicated to analog in-memory computing for TinyML tasks entitled AnalogNAS. The obtained models are accurate, noise and drift-resilient, and small enough to run on resource-constrained devices. Experimental results demonstrate that our method outperforms state-of-the-art models on analog hardware for three tasks of the MLPerf benchmark: image classification on CIFAR-10, Visual Wake Words, and Keyword Spotting.

In Chapter 6, research questions 6 and 7 were addressed. The context of medical imaging analysis presents an attractive target for HW-NAS for edge computing. Due to the sensitivity of data and the critical timing of some diagnoses, developing

efficient DL models would highly benefit research and society. To advance HW-NAS research in such a domain, we developed *MED-NAS-Bench*, the first NAS Benchmark for medical tasks. MED-NAS-Bench includes performance metrics for millions of architectures, associated with their latency, and energy consumption on multiple edge hardware platforms. We then presented a multi-task medical imaging architecture search, *MT-MIAS*. Using MT-MIAS, we could find a generalized architecture able to solve multiple medical tasks and provide state-of-the-art performance. This architecture is highly practical as deploying a single architecture benefits edge devices giving faster response and preserving energy.

Overall, this thesis has investigated many limitations of HW-NAS and provided concrete and practical solutions. Contributions have been made to several areas including accelerating HW-NAS process with efficient surrogate models, improving the search space design, and consolidating our knowledge of HW-NAS on practical applications. We applied HW-NAS to discover tailored DL models for analog in-memory computing and enhance their energy efficiency. Finally, we extended the application of HW-NAS to medical tasks, where we built a specialized benchmark and implemented a dedicated multi-task search methodology.

7.2 Future Work

As stated in the Introduction (Chapter 1), the primary motivation of this thesis was to design fast, efficient, and practical HW-NAS strategies. Using multi-objective surrogate models and flexible search spaces, we found efficient DL models for a variety of tasks by proposing a fast and practical framework, under edge constraints.

In this thesis, two types of estimation strategies, namely Predictive Models and Weight-sharing Supernetworks, were thoroughly investigated. We highlighted the advantages and drawbacks of both strategies and pinpoint the applications in which each one of them is best.

Additionally, several other methods, including zero-shot estimation [213] and learning-curve extrapolation [236], have emerged. However, these methods still face certain limitations in terms of multi-objectivity, scalability, and accuracy, leaving room for further exploration and refinement in future research endeavors. Also, while we provided a thorough analysis of HW-NAS estimation strategies, a comprehensive comparison between top architectures using metrics such as Centered Kernel Alignment (CKA) [314] or Similarity analysis to evaluate the generalization ability and robustness of these models across different datasets and tasks is missing from the literature, this could potentially be an additional key focus of MED-NAS-Bench.

Figure 7.1 summarizes the directions of the future work for each component of HW-NAS.

While these are ongoing research areas with potential for near-future achievements, the following specific future directions can be pursued to further enhance HW-NAS:

1. **Pushing the Boundaries of Predefined Search Spaces.** Further exploration can be done to expand predefined search spaces, enabling the discovery of novel and unconventional architectures that may exhibit superior performance on edge devices. Existing works [77, 79, 80], including our work GOS [7], have already demonstrated promising advancements in this direction, defining their search space with mathematical instructions. However, these are still time-consuming and these methods can be further improved. This improvement can involve incorporating unconventional operations or unconventional connectivity patterns within the search space, allowing for more diverse and innovative architectures to be discovered.

2. **Search Algorithms and Large Search Spaces:** It is crucial to develop efficient and scalable search algorithms that can effectively explore large search

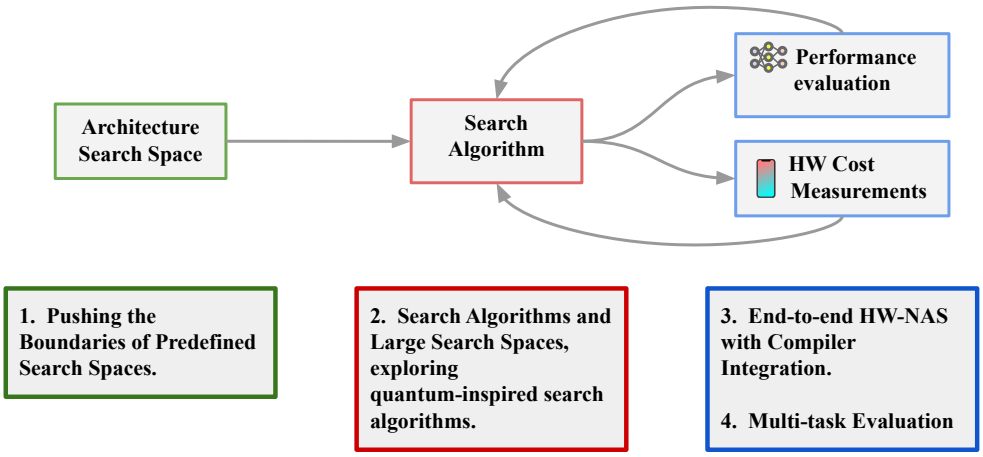

Figure 7.1: Future Works on top of HW-NAS framework.

spaces within practical time constraints. Further advancements in optimization algorithms and meta-learning techniques can facilitate the search process and enable the exploration of larger and more complex search spaces. This would allow for the discovery of highly optimized architectures that align with the hardware constraints of edge devices, while still meeting performance requirements. Exploring quantum-inspired search algorithms [315] holds promise in tackling the challenges of searching large spaces more efficiently. Leveraging quantum computing principles such as superposition and entanglement, these algorithms have the potential to enhance the search process and expedite the discovery of optimal architectures for edge computing.

3. **End-to-end HW-NAS with Compiler Integration.** Future research can focus on developing end-to-end HW-NAS methodologies that not only encompass architecture search and optimization but also incorporate the compiler in the loop. This integration would enable seamless collaboration between the hardware-aware neural architecture search process and the compiler, resulting in optimized neural architectures that are directly compatible and efficiently executable on edge devices. By considering the compiler as an integral component in the HW-NAS pipeline, the overall system can effectively leverage hardware-specific optimizations, code generation techniques, and resource allocation strategies, leading to highly efficient and performance-optimized solutions for edge computing scenarios. An example of such integration can be seen in MCUNet [266], which dedicated a compiler specifically for a particular type of microcontroller. Generalizing this approach to incorporate various deep learning compilers, including Tiramisu [58], and XLA [60], would be a valuable direction to explore.

4. **Multi-task investigation** Multi-task deep learning models are crucial to reducing the memory occupancy and execution time in edge devices. Investigating how sharing knowledge and architectural components across multiple related tasks can lead to more efficient and effective neural architectures for edge devices could be an interesting avenue to explore. This approach could leverage the synergistic relationships between different tasks and exploit shared representations to enhance model performance and reduce resource requirements. For instance, in the context of an autonomous driving car, instead of employing a scheduler to switch between different models for object detection, lane

detection, and semantic segmentation, a multi-task learning approach could be explored. By jointly training a single neural architecture on these tasks, the model can benefit from the shared information and potentially achieve better overall performance, reduced memory footprint, and improved inference efficiency.

We envision future research in Hardware-aware Neural Architecture Search (HW-NAS) for edge computing to overcome current limitations, paving the way for automated and seamless integration of architecture search into existing AutoML systems. This advancement holds the potential to democratize deep learning, allowing practitioners to harness its power for a wide range of use-cases in the context of edge computing. Moreover, we strongly believe that HW-NAS has the capacity to drive advancements in the field of deep learning as a whole by incorporating desirable properties, such as explainability and proper uncertainty calibration, into the search process, thereby enhancing the overall performance and reliability of machine learning systems in edge computing scenarios.

Bibliography

[1] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smaïl Niar, Martin Wistuba, and Naigang Wang. A comprehensive survey on hardware-aware neural architecture search. *CoRR*, abs/2101.09336, 2021. URL https://arxiv.org/abs/2101.09336. V, 2, 4, 81

[2] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. Hardware-aware neural architecture search: Survey and taxonomy. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4322–4329, 8 2021. V

[3] Hadjer Benmeziane, Hamza Ouarnoughi, Kaoutar El Maghraoui, and Smaïl Niar. Accelerating neural architecture search with rank-preserving surrogate models. In Manar Abu Talib, Laila Benhlima, and Kaoutar El Maghraoui, editors, *ArabWIC 2021: The 7th Annual International Conference on Arab Women in Computing in Conjunction with the 2nd Forum of Women in Research, Sharjah, United Arab Emirates, August 25 - 26, 2021*, pages 21:1–21:6. ACM, 2021. V

[4] Hadjer Benmeziane, Hamza Ouarnoughi, Kaoutar El Maghraoui, and Smail Niar. Multi-objective hardware-aware neural architecture search with pareto rank-preserving surrogate models. 20(2), 2023. ISSN 1544-3566. V, 4

[5] Hadjer Benmeziane, Smail Niar, Hamza Ouarnoughi, and Kaoutar El Maghraoui. Pareto rank surrogate model for hardware-aware neural architecture search. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2022. V, 4, 47, 80

[6] Hadjer Benmeziane, Smail Niar, Hamza Ouarnoughi, and Kaoutar El Maghraoui. Pareto rank-preserving supernetwork for hardware-aware neural architecture search. In *Submitted to European Conference on Artificial Intelligence ECAI 2023*, 2023. V, 4, 29

[7] Hadjer Benmeziane, Smail Niar, Hamza Ouarnoughi, and Kaoutar El Maghraoui. Grassroots operator search for model edge adaptation. In *Submitted to EMBEDDED SYSTEMS WEEK Conference*, 2023. VI, 4, 145

[8] Hadjer Benmeziane, Hamza Ouranoughi, Smaïl Niar, and Kaoutar El Maghraoui. Caw-nas: Compression aware neural architecture search. In *25th Euromicro Conference on Digital System Design, DSD*, pages 391–397. IEEE, 2022. VI, 4

[9] Hadjer Benmeziane, Corey Lammie, Irem Boybat, Malte J. Rasch, Manuel Le Gallo, Hsinyu Tsai, Ramachandran Muralidhar, Smaïl Niar, Hamza Ouarnoughi, Vijay Narayanan, Abu Sebastian, and Kaoutar El Maghraoui. Analognas: A neural network design framework for accurate inference with analog in-memory computing. 2023. VI, 5, 35

[10] Hadjer Benmeziane, Smail Niar, Hamza Ouarnoughi, and Kaoutar El Maghraoui. Med-nas-bench: A generalized neural architecture search benchmark for medical imaging analysis. In *Submitted to Nature Methods*, 2023. VI, 5

[11] Hadjer Benmeziane, Hamza Ouarnoughi, Kaoutar El Maghraoui, and Smaïl Niar. Real-time style transfer with efficient vision transformers. In Aaron Yi Ding and Volker Hilt, editors, *EdgeSys EuroSys 2022: Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking, Rennes, France, April 5 - 8, 2022*, pages 31–36. ACM, 2022. VII

[12] Lotfi Abdelkrim Mecharbat, Hadjer Benmeziane, Hamza Ouranoughi, and Smaïl Niar. Hyt-nas: Hybrid transformers neural architecture search for edge devices. *CoRR*, abs/2303.04440, 2023. VII

[13] Hadjer Benmeziane, Halima Bouzidi, Hamza Ouarnoughi, Ozcan Ozturk, and Smaïl Niar. Treasure what you have: Exploiting similarity in deep neural networks for efficient video processing. *CoRR*, abs/2305.06492, 2023. VII

[14] Hadjer Benmeziane, Amine Ziad Ounnoughene, Imane Hamzaoui, and Younes Bouhadjar. Skip connections in spiking neural networks: An analysis of their effect on network training. *CoRR*, abs/2303.13563, 2023. VII

[15] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020. XIII, 13, 15

[16] Xuefei Ning, Yin Zheng, Tianchen Zhao, Yu Wang, and Huazhong Yang. A generic graph-based neural architecture encoding scheme for predictor-based NAS. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XIII*, volume 12358 of *Lecture Notes in Computer Science*, pages 189–204. Springer, 2020. XIII, 47, 48, 50, 54, 55, 57, 58, 106, 107, 128

[17] Lukasz Dudziak, Thomas C. P. Chau, Mohamed S. Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas D. Lane. BRP-NAS: prediction-based NAS using gcns. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems*, 2020. XIII, 47, 48, 50, 54, 55, 57, 58

[18] Michela Antonelli, Annika Reinke, Spyridon Bakas, Keyvan Farahani, Annette Kopp-Schneider, Bennett A Landman, Geert Litjens, Bjoern Menze, Olaf Ronneberger, Ronald M Summers, et al. The medical segmentation decathlon. *Nature communications*, 13(1):4128, 2022. XV, 119, 120, 123, 137

[19] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, 2011. URL http://www.aclweb.org/anthology/P11-1015. XVII, 41

[20] Zhilin Zhang, Zhouyue Pi, and Benyuan Liu. TROIKA: A general framework for heart rate monitoring using wrist-type photoplethysmographic signals during intensive physical exercise. *IEEE Trans. Biomed. Eng.*, 62(2):522–531, 2015. XVII, 93, 95

[21] Shubham Jain, Hsinyu Tsai, Ching-Tzu Chen, Ramachandran Muralidhar, Irem Boybat, Martin M. Frank, Stanisław Woźniak, Milos Stanisavljevic, Praneet Adusumilli, Pritish Narayanan, Kohji Hosokawa, Masatoshi Ishii, Arvind

Kumar, Vijay Narayanan, and Geoffrey W. Burr. A heterogeneous and programmable compute-in-memory accelerator architecture for analog-ai using dense 2-d mesh. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 31(1):114–127, 2023. doi: 10.1109/TVLSI.2022.3221390. XVIII, 113, 114

[22] Sparsh Mittal and Shraiysh Vaishay. A survey of techniques for optimizing deep learning on gpus. *Journal of Systems Architecture*, 99:101635, 2019. ISSN 1383-7621. doi: https://doi.org/10.1016/j.sysarc.2019.101635. 2

[23] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. Do vision transformers see like convolutional neural networks? In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 12116–12128, 2021. 2

[24] Colin White, Arber Zela, Robin Ru, Yang Liu, and Frank Hutter. How powerful are performance predictors in neural architecture search? In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 28454–28469, 2021. 2

[25] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013. 12

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 12

[27] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014. 12

[28] 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. 12, 33

[29] Daniel S. Park, Yu Zhang, Ye Jia, Wei Han, Chung-Cheng Chiu, Bo Li, Yonghui Wu, and Quoc V. Le. Improved noisy student training for automatic speech recognition. *Interspeech 2020*, Oct 2020. 12

[30] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig. The microsoft 2016 conversational speech recognition system. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5255–5259, 2017. 12

[31] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29 (6):82–97, 2012. 12

[32] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the*

Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, pages 4171–4186, 2019. 12

[33] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12:2493–2537, 2011. 12

[34] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016. 12

[35] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6105–6114, 2019. 12

[36] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, Jonas Mueller, R Manmatha, et al. Resnest: Split-attention networks. *arXiv preprint arXiv:2004.08955*, 2020. 12

[37] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 12

[38] Zihang Dai, Hanxiao Liu, Quoc V. Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 3965–3977, 2021. 13

[39] Jiahui Yu, Zirui Wang, Vijay Vasudevan, Legg Yeung, Mojtaba Seyedhosseini, and Yonghui Wu. Coca: Contrastive captioners are image-text foundation models. *CoRR*, abs/2205.01917, 2022. doi: 10.48550/arXiv.2205.01917. URL https://doi.org/10.48550/arXiv.2205.01917. 14

[40] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. URL http://arxiv.org/abs/1801.04381. 14, 37

[41] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2818–2826, 2016. doi: 10.1109/CVPR.2016.308. URL https://doi.org/10.1109/CVPR.2016.308. 14

[42] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *8th International Conference on Learning Representations, ICLR*. OpenReview.net, 2020. URL https://openreview.net/forum?id=HylxE1HKwS. 14, 22, 24, 28, 40

[43] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018. doi: 10.1109/CVPR.2018.00907. 14, 30, 104

[44] Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V. Le. NAS-FPN: learning scalable feature pyramid architecture for object detection. *CoRR*, abs/1904.07392, 2019. URL http://arxiv.org/abs/1904.07392. 15

[45] A. Howard, M. Sandler, B. Chen, W. Wang, L. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le. Searching for mobilenetv3. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, 2019. 15

[46] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Conference on Computer Vision and Pattern Recognition, CVPR*, pages 10734–10742. Computer Vision Foundation / IEEE, 2019. 15, 17, 19, 22, 25, 28, 29, 31, 32, 39

[47] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *CoRR*, abs/1710.09282, 2017. URL http://arxiv.org/abs/1710.09282. 15

[48] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.07122. 15

[49] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 15, 33

[50] Xiaodan Zhu, Parinaz Sobhani, and Hongyu Guo. Long short-term memory over tree structures. *CoRR*, abs/1503.04881, 2015. URL http://arxiv.org/abs/1503.04881. 15

[51] Jos van der Westhuizen and Joan Lasenby. The unreasonable effectiveness of the forget gate. *CoRR*, abs/1804.04849, 2018. URL http://arxiv.org/abs/1804.04849. 15

[52] Le Song, Mariya Ishteva, Ankur P. Parikh, Eric P. Xing, and Haesun Park. Hierarchical tensor decomposition of latent tree graphical models. In *Proceedings of the 30th International Conference on Machine Learning,ICML*, volume 28, pages 334–342, 2013. URL http://proceedings.mlr.press/v28/song13.html. 16

[53] Yassine Zniyed, Rémy Boyer, André L. F. de Almeida, and Gérard Favier. A tt-based hierarchical framework for decomposing high-order tensors. *SIAM J. Sci. Comput.*, 42(2):A822–A848, 2020. 16

[54] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 1737–1746, 2015. 16

[55] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations, ICLR*, 2016. URL http://arxiv.org/abs/1510.00149. 16

[56] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *CoRR*, abs/1710.01878, 2017. URL http://dblp.uni-trier.de/db/journals/corr/corr1710.html#abs-1710-01878. 16

[57] M. Li, Y. Liu, X. Liu, Q. Sun, X. You, H. Yang, Z. Luan, L. Gan, G. Yang, and D. Qian. The deep learning compiler: A comprehensive survey. *IEEE Transactions on Parallel and Distributed Systems*, 32(3):708–727, 2021. doi: 10.1109/TPDS.2020.3030548. 16

[58] Riyadh Baghdadi, Jessica Ray, Malek Ben Romdhane, Emanuele Del Sozzo, Abdurrahman Akkas, Yunming Zhang, Patricia Suriana, Shoaib Kamil, and Saman P. Amarasinghe. Tiramisu: A polyhedral compiler for expressing fast and portable code. In Mahmut Taylan Kandemir, Alexandra Jimborean, and Tipp Moseley, editors, *International Symposium on Code Generation and Optimization, CGO*, pages 193–205, 2019. 16, 82, 146

[59] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Q. Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. TVM: an automated end-to-end optimizing compiler for deep learning. In Andrea C. Arpaci-Dusseau and Geoff Voelker, editors, *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 578–594. USENIX Association, 2018. URL https://www.usenix.org/conference/osdi18/presentation/chen. 16

[60] Amit Sabne. Xla : Compiling machine learning for peak performance, 2020. 16, 82, 146

[61] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *CoRR*, abs/1812.00332, 2018. URL http://arxiv.org/abs/1812.00332. 17, 19, 20, 22, 24, 25, 28, 29, 30, 31, 32, 40

[62] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *CoRR*, abs/1807.11626, 2018. URL http://arxiv.org/abs/1807.11626. 17, 19, 22, 24, 25, 27, 31, 39

[63] L. Zhang, Y. Yang, Y. Jiang, W. Zhu, and Y. Liu. Fast hardware-aware neural architecture search. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2959–2967, 2020. 17, 22, 28, 30, 32

[64] Weiwen Jiang, X. Zhang, E. Sha, L. Yang, Q. Zhuge, Y. Shi, and J. Hu. Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search. *56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019. 18, 21

[65] Lei Yang, Zheyu Yan, Meng Li, Hyoukjun Kwon, Weiwen Jiang, Liangzhen Lai, Yiyu Shi, Tushar Krishna, and Vikas Chandra. *Co-Exploration of Neural Architectures and Heterogeneous ASIC Accelerator Designs Targeting Multiple Tasks*. 2020. ISBN 9781450367257. 18, 21, 27, 31

[66] Grace Chu, Okan Arikan, Gabriel Bender, Weijun Wang, Achille Brighton, Pieter-Jan Kindermans, Hanxiao Liu, Berkin Akin, Suyog Gupta, and Andrew Howard. Discovering multi-hardware mobile models via architecture search. *CoRR*, abs/2008.08178, 2020. URL https://arxiv.org/abs/2008.08178. 18, 19, 20, 23, 40

[67] Y. Jiang, X. Wang, and W. Zhu. Hardware-aware transformable architecture search with efficient search space. In *IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2020. doi: 10.1109/ICME46284.2020.9102721. 18, 20, 23, 28, 29

[68] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L. Yuille, and Fei-Fei Li. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Conference on Computer Vision and Pattern Recognition, CVPR*, pages 82–92, 2019. 19, 39

[69] Alberto Marchisio, Andrea Massa, Vojtech Mrazek, Beatrice Bussolino, Maurizio Martina, and Muhammad Shafique. Nascaps: A framework for neural architecture search to optimize the accuracy and hardware efficiency of convolutional capsule networks. In *ICCAD*, 2020. 19, 22, 23, 28, 31, 32

[70] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3856–3866, 2017. URL http://papers.nips.cc/paper/6975-dynamic-routing-between-capsules. 20

[71] Yong Guo, Yin Zheng, Mingkui Tan, Qi Chen, Jian Chen, Peilin Zhao, and Junzhou Huang. NAT: neural architecture transformer for accurate and compact architectures. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 735–747, 2019. URL http://papers.nips.cc/paper/8362-nat-neural-architecture-transformer-for-accurate-and-compact-architectures. 20, 22

[72] Henry Tsai, Jayden Ooi, Chun-Sung Ferng, Hyung Won Chung, and Jason Riesa. Finding fast transformers: One-shot neural architecture search by component composition. *CoRR*, abs/2008.06808, 2020. URL https://arxiv.org/abs/2008.06808. 20, 22

[73] Md Shahriar Iqbal, Jianhai Su, Lars Kotthoff, and Pooyan Jamshidi. Flexibo: Cost-aware multi-objective optimization of deep neural networks. 2020. 20, 22

[74] Wei Niu, Zhenglun Kong, Geng Yuan, Weiwen Jiang, Jiexiong Guan, Caiwen Ding, Pu Zhao, Sijia Liu, Bin Ren, and Yanzhi Wang. Real-time execution of large-scale language models on mobile, 2020. 20

[75] Albert Shaw, Daniel Hunter, Forrest Iandola, and Sammy Sidhu. Squeezenas: Fast neural architecture search for faster semantic segmentation, 2019. 20, 28, 29, 39

[76] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. HAT: Hardware-aware transformers for efficient natural language processing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7675–7688, 2020. 20, 28, 30

[77] Esteban Real, Chen Liang, David R. So, and Quoc V. Le. Automl-zero: Evolving machine learning algorithms from scratch. In *Proceedings of the 37th International Conference on Machine Learning, ICML*, volume 119, pages 8007–8019, 2020. 20, 82, 145

[78] Simon Schrodi, Danny Stoll, Binxin Ru, Rhea Sukthanker, Thomas Brox, and Frank Hutter. Towards discovering neural architectures from scratch. *CoRR*, abs/2211.01842, 2022. doi: 10.48550/arXiv.2211.01842. 20

[79] Hanxiao Liu, Andy Brock, Karen Simonyan, and Quoc Le. Evolving normalization-activation layers. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems, NeurIPS*, 2020. 20, 87, 145

[80] Angelica Chen, David M. Dohan, and David R. So. Evoprompting: Language models for code-level neural architecture search. *CoRR*, abs/2302.14838, 2023. doi: 10.48550/arXiv.2302.14838. 20, 145

[81] Ye Yu, Yingmin Li, Shuai Che, Niraj K. Jha, and Weifeng Zhang. Software-defined design space exploration for an efficient AI accelerator architecture. *CoRR*, abs/1903.07676, 2019. URL http://arxiv.org/abs/1903.07676. 21

[82] Weiwen Jiang, Lei Yang, Edwin Hsing-Mean Sha, Qingfeng Zhuge, Shouzhen Gu, Yiyu Shi, and Jingtong Hu. Hardware/software co-exploration of neural architectures. *CoRR*, abs/1907.04650, 2019. URL http://arxiv.org/abs/1907.04650. 21, 22, 32

[83] Qing Lu, Weiwen Jiang, Xiaowei Xu, Yiyu Shi, and Jingtong Hu. On neural architecture search for resource-constrained hardware platforms. *CoRR*, abs/1911.00105, 2019. URL http://arxiv.org/abs/1911.00105. 21, 22, 27, 31, 32

[84] Mohamed S. Abdelfattah, undefinedukasz Dudziak, Thomas Chau, Royson Lee, Hyeji Kim, and Nicholas D. Lane. Best of both worlds: Automl codesign of a cnn and its hardware accelerator. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, 2020. ISBN 9781450367257. 21, 22, 24, 27, 32

[85] Weiwei Chen, Ying Wang, Shuang Yang, Chen Liu, and Lei Zhang. You only search once: A fast automation framework for single-stage dnn/accelerator co-design. In *Design, Automation & Test in Europe Conference & Exhibition, DATE*, pages 1283–1286, 2020. doi: 10.23919/DATE48585.2020.9116474. URL https://doi.org/10.23919/DATE48585.2020.9116474. 21, 22

[86] Cong Hao, Xiaofan Zhang, Yuhong Li, Sitao Huang, Jinjun Xiong, Kyle Rupnow, Wen-mei Hwu, and Deming Chen. Fpga/dnn co-design: An efficient design methodology for iot intelligence on the edge. In *Proceedings of the 56th Annual Design Automation Conference*, 2019. ISBN 9781450367257. 21, 32

[87] Weiwen Jiang, Lei Yang, Sakyasingha Dasgupta, Jingtong Hu, and Yiyu Shi. Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start, 2020. 21, 24, 27, 31, 32

[88] B. Lu, J. Yang, L. Y. Chen, and S. Ren. Automating deep neural network model selection for edge inference. In *IEEE First International Conference on Cognitive Machine Intelligence (CogMI)*, pages 184–193, 2019. 22

[89] Why tinyml is a giant opportunity. https://venturebeat.com/2020/01/11/why-tinyml-is-a-giant-opportunity/. 22

[90] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. In *Advances in Neural Information Processing Systems (NeurIPS'20)*, 2020. 22, 28, 31

[91] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, Peter Vajda, Matt Uyttendaele, and Niraj K. Jha. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Conference on Computer Vision and Pattern Recognition, CVPR*, pages 11398–11407, 2019. 22

[92] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Bowen Shi, Qi Tian, and Hongkai Xiong. Latency-aware differentiable neural architecture search. 2020. 22

[93] Song Bian, Weiwen Jiang, Qing Lu, Yiyu Shi, and Takashi Sato. NASS: optimizing secure inference via neural architecture search. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI - 24th European Conference on Artificial Intelligence*, volume 325, pages 1746–1753, 2020. 22

[94] AJ Piergiovanni, Anelia Angelova, and Michael S. Ryoo. Tiny video networks, 2019. 22

[95] Javier Fernandez-Marques, Paul N. Whatmough, Andrew Mundy, and Matthew Mattina. Searching for winograd-aware quantized networks. 2020. 22

[96] C. Fu, H. Chen, Z. Yang, F. Koushanfar, Y. Tian, and J. Zhao. Enhancing model parallelism in neural architecture search for multidevice system. *IEEE Micro*, 40(5):46–55, 2020. doi: 10.1109/MM.2020.3004538. 22

[97] Mohammad Loni, Sima Sinaei, Ali Zoljodi, Masoud Daneshtalab, and Mikael Sjödin. Deepmaker: A multi-objective optimization framework for deep neural networks in embedded systems. *Microprocessors and Microsystems*, 73:102989, 2020. 22

[98] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. Ppp-net: Platform-aware progressive search for pareto-optimal neural architectures, 2018. URL https://openreview.net/forum?id=B1NT3TAIM. 22

[99] Chi-Hung Hsu, Shu-Huan Chang, Da-Cheng Juan, Jia-Yu Pan, Yu-Ting Chen, Wei Wei, and Shih-Chieh Chang. MONAS: multi-objective neural architecture search using reinforcement learning. *CoRR*, abs/1806.10332, 2018. URL http://arxiv.org/abs/1806.10332. 22, 26, 31

[100] An-Chieh Cheng, Chieh Hubert Lin, Da-Cheng Juan, Wei Wei, and Min Sun. Instanas: Instance-aware neural architecture search. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*, pages 3577–3584, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/5764. 22

[101] Lile Cai, Anne-Maelle Barneche, Arthur Herbout, Chuan Sheng Foo, Jie Lin, Vijay Ramaseshan Chandrasekhar, and Mohamed M. Sabry Aly. TEA-DNN: the quest for time-energy-accuracy co-optimized deep neural networks. In *IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 1–6, 2019. doi: 10.1109/ISLPED.2019.8824934. URL https://doi.org/10.1109/ISLPED.2019.8824934. 22

[102] N. Mitschke, M. Heizmann, K. Noffz, and R. Wittmann. Gradient based evolution to optimize the structure of convolutional neural networks. In *25th IEEE International Conference on Image Processing (ICIP)*, pages 3438–3442, 2018. doi: 10.1109/ICIP.2018.8451394. 22

[103] Vladimir Nekrasov, Hao Chen, Chunhua Shen, and Ian D. Reid. Fast neural architecture search of compact semantic segmentation models via auxiliary cells. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 9126–9135, 2019. 22

[104] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. Nsga-net: Neural architecture search using multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019. doi: 10.1145/3321707.3321729. URL https://doi.org/10.1145/3321707.3321729. 22, 26

[105] Atin Sood, Benjamin Elder, Benjamin Herta, Chao Xue, Costas Bekas, A. Cristiano I. Malossi, Debashish Saha, Florian Scheidegger, Ganesh Venkataraman, Gegi Thomas, Giovanni Mariani, Hendrik Strobelt, Horst Samulowitz, Martin Wistuba, Matteo Manica, Mihir R. Choudhury, Rong Yan, Roxana Istrate, Ruchir Puri, and Tejaswini Pedapati. Neunets: An automated synthesis engine for neural network design. *CoRR*, abs/1901.06261, 2019. URL http://arxiv.org/abs/1901.06261. 22, 31

[106] Xin Li, Yiming Zhou, Zheng Pan, and Jiashi Feng. Partial order pruning: For best speed/accuracy trade-off in neural architecture search. In *Conference on Computer Vision and Pattern Recognition, CVPR*, pages 9145–9153. Computer Vision Foundation / IEEE, 2019. 22

[107] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 481–497, 2019. 22, 30

[108] Jiahui Yu and Thomas S. Huang. Network slimming by slimmable networks: Towards one-shot architecture search for channel numbers. *CoRR*, abs/1903.11728, 2019. URL http://arxiv.org/abs/1903.11728. 22

[109] S. Hong, D. Kim, and M. Choi. Memory-efficient models for scene text recognition via neural architecture search. In *IEEE Winter Applications of Computer Vision Workshops (WACVW)*, pages 183–191, 2020. 22

[110] Dilin Wang, Meng Li, Lemeng Wu, Vikas Chandra, and Qiang Liu. Energy-aware neural architecture optimization with fast splitting steepest descent. *CoRR*, abs/1910.03103, 2019. URL http://arxiv.org/abs/1910.03103. 22

[111] Sian-Yao Huang and Wei-Ta Chu. Ponas: Progressive one-shot neural architecture search for very efficient deployment. *arXiv preprint arXiv:2003.05112*, 2020. 22

[112] E. Lee and C. Lee. Neuralscale: Efficient scaling of neurons for resource-constrained deep neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1475–1484, 2020. 22

[113] Ye-Hoon Kim, B. Reddy, Sojung Yun, and Chanwon Seo. Nemo : Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy. 2017. 22, 26

[114] K. Hu, S. Tian, S. Guo, N. Li, L. Luo, and L. Wang. Recurrent neural architecture search based on randomness-enhanced tabu algorithm. In *International Joint Conference on Neural Networks (IJCNN)*, 2020. 22, 25

[115] Maria Baldeon-Calisto and Susana K. Lai-Yuen. Adaresu-net: Multiobjective adaptive convolutional neural network for medical image segmentation. *Neurocomputing*, 392:325 – 340, 2020. ISSN 0925-2312. 22

[116] Song Han, Han Cai, Ligeng Zhu, Ji Lin, Kuan Wang, Zhijian Liu, and Yujun Lin. Design Automation for Efficient Deep Learning Computing. *arXiv e-prints*, art. arXiv:1904.10616, April 2019. 22, 24, 34, 40, 41

[117] Woong Bae, Seungho Lee, Yeha Lee, Beomhee Park, Minki Chung, and Kyu-Hwan Jung. Resource optimized neural architecture search for 3d medical image segmentation. In Dinggang Shen, Tianming Liu, Terry M. Peters, Lawrence H. Staib, Caroline Essert, Sean Zhou, Pew-Thian Yap, and Ali R. Khan, editors,

Medical Image Computing and Computer Assisted Intervention - MICCAI, volume 11765, pages 228–236, 2019. 22

[118] Christoph Schorn, Thomas Elsken, Sebastian Vogel, Armin Runge, Andre Guntoro, and Gerd Ascheid. Automated design of error-resilient and hardware-efficient deep neural networks. *CoRR*, abs/1909.13844, 2019. URL http://arxiv.org/abs/1909.13844. 22

[119] Thomas Cassimon, Simon Vanneste, Stig Bosmans, Siegfried Mercelis, and Peter Hellinckx. Using neural architecture search to optimize neural networks for embedded devices. In Leonard Barolli, Peter Hellinckx, and Juggapong Natwichai, editors, *Advances on P2P, Parallel, Grid, Cloud and Internet Computing*, 2020. 22

[120] Bin Wang, Yanan Sun, Bing Xue, and Mengjie Zhang. Evolving deep neural networks by multi-objective particle swarm optimization for image classification. *CoRR*, abs/1904.09035, 2019. URL http://arxiv.org/abs/1904.09035. 22

[121] Martin Ferianc, Hongxiang Fan, and Miguel Rodrigues. Vinnas: Variational inference-based neural network architecture search, 2020. 22

[122] Zhichao Lu, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 35–51, 2020. 22

[123] Yibo Hu, Xiang Wu, and Ran He. Tf-nas: Rethinking three search freedoms of latency-constrained differentiable neural architecture search, 2020. 22

[124] Guohao Li, Mengmeng Xu, Silvio Giancola, Ali K. Thabet, and Bernard Ghanem. LC-NAS: latency constrained neural architecture search for point cloud networks. *CoRR*, abs/2008.10309, 2020. URL https://arxiv.org/abs/2008.10309. 22

[125] Zhihang Yuan, Xin Liu, Bingzhe Wu, and Guangyu Sun. Enas4d: Efficient multi-stage cnn architecture search for dynamic inference. *ArXiv*, abs/2009.09182, 2020. 22

[126] P. Achararit, M. A. Hanif, R. V. W. Putra, M. Shafique, and Y. Hara-Azumi. Apnas: Accuracy-and-performance-aware neural architecture search for neural hardware accelerators. *IEEE Access*, 8:165319–165334, 2020. doi: 10.1109/ACCESS.2020.3022327. 22

[127] Z. Lu, I. Whalen, Y. Dhebar, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti. Multi-objective evolutionary design of deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*, pages 1–1, 2020. doi: 10.1109/TEVC.2020.3024708. 22

[128] Yao Yang, Andrew Nam, Mohamad M. Nasr-Azadani, and Teresa Tung. Resource-aware pareto-optimal automated machine learning platform. *CoRR*, abs/2011.00073, 2020. URL https://arxiv.org/abs/2011.00073. 22

[129] Igor Fedorov, Ryan P. Adams, Matthew Mattina, and Paul N. Whatmough. Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers. *CoRR*, abs/1905.12107, 2019. URL http://arxiv.org/abs/1905.12107. 22

[130] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas Navarro, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul N. Whatmough. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers, 2020. 22

[131] Yu Weng, Tianbao Zhou, Yujie Li, and Xiaoyu Qiu. Nas-unet: Neural architecture search for medical image segmentation. *IEEE Access*, 7:44247–44257, 2019. 22, 39

[132] Thomas Cassimon, Simon Vanneste, Stig Bosmans, Siegfried Mercelis, and Peter Hellinckx. Designing resource-constrained neural networks using neural architecture search targeting embedded devices. *Internet of Things*, page 100234, 2020. ISSN 2542-6605. 22

[133] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1904.00420*, 2019. 22, 31, 33

[134] Yunyang Xiong, Ronak Mehta, and Vikas Singh. Resource constrained neural network architecture search: Will a submodularity assumption help? In *International Conference on Computer Vision, ICCV*, pages 1901–1910, 2019. 22

[135] Xin Xia and Wenrui Ding. Hnas: Hierarchical neural architecture search on mobile devices. 2020. 22

[136] Yunyang Xiong, Hanxiao Liu, Suyog Gupta, Berkin Akin, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Vikas Singh, and Bo Chen. MobileDets: Searching for Object Detection Architectures for Mobile Accelerators. *arXiv e-prints*, 2020. 22

[137] Z. Yu, Y. Qin, X. Xu, C. Zhao, Z. Wang, Z. Lei, and G. Zhao. Auto-fas: Searching lightweight networks for face anti-spoofing. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 996–1000, 2020. 22

[138] Javier García López, Antonio Agudo, and Francesc Moreno-Noguer. E-dnas: Differentiable neural architecture search for embedded systems. 22

[139] Y. Li, X. Jin, J. Mei, X. Lian, L. Yang, C. Xie, Q. Yu, Y. Zhou, S. Bai, and A. L. Yuille. Neural architecture search for lightweight non-local networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10294–10303, 2020. 22

[140] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han. Apq: Joint search for network architecture, pruning and quantization policy. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2075–2084, 2020. doi: 10.1109/CVPR42600.2020.00215. 22, 31, 75, 81

[141] Jaeseong Lee, Duseok Kang, and Soonhoi Ha. S3NAS: fast npu-aware neural architecture search methodology. *CoRR*, abs/2009.02009, 2020. URL https://arxiv.org/abs/2009.02009. 22

[142] Ching-Chen Wang, Ching-Te Chiu, and Jheng-Yi Chang. Efficientnet-elite: Extremely lightweight and efficient cnn models for edge devices by network candidate search. 2020. 22

[143] Yongan Zhang, Yonggan Fu, Weiwen Jiang, Chaojian Li, Haoran You, Meng Li, Vikas Chandra, and Yingyan Lin. Dna: Differentiable network-accelerator co-search. *arXiv preprint arXiv:2010.14778*, 2020. 22

[144] Kanghyun Choi, Deokki Hong, Hojae Yoon, Joonsang Yu, Youngsok Kim, and Jinho Lee. Dance: Differentiable accelerator/network co-exploration. *arXiv preprint arXiv:2009.06237*, 2020. 22

[145] Suyog Gupta and Berkin Akin. Accelerator-aware neural network design using automl. 2020. 22

[146] Guihong Li, Sumit K. Mandal, Umit Y. Ogras, and Radu Marculescu. Flash:fast neural architecture s earch with hardware optimization. *ACM Trans. Embed. Comput. Syst.*, 20(5s), sep 2021. ISSN 1539-9087. doi: 10.1145/3476994. URL https://doi.org/10.1145/3476994. 22, 25, 30

[147] Weiwen Jiang, Qiuwen Lou, Zheyu Yan, Lei Yang, Jingtong Hu, Xiaobo Sharon Hu, and Yiyu Shi. Device-circuit-architecture co-exploration for computing-in-memory neural accelerators. *IEEE Trans. Computers*, 70(4):595–605, 2021. doi: 10.1109/TC.2020.2991575. URL https://doi.org/10.1109/TC.2020.2991575. 22

[148] Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, and Evgeny Burnaev. Nas-bench-nlp: Neural architecture search benchmark for natural language processing, 2020. 23, 36, 37

[149] Royson Lee, Łukasz Dudziak, Mohamed Abdelfattah, Stylianos I. Venieris, Hyeji Kim, Hongkai Wen, and Nicholas D. Lane. Journey towards tiny perceptual super-resolution. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 85–102, 2020. 23

[150] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. Graph neural architecture search. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 1403–1409. International Joint Conferences on Artificial Intelligence Organization, 7 2020. doi: 10.24963/ijcai.2020/195. URL https://doi.org/10.24963/ijcai.2020/195. Main track. 23

[151] Y. Yang, C. Wang, L. Gong, and X. Zhou. Fpnet: Customized convolutional neural network for fpga platforms. In *International Conference on Field-Programmable Technology (ICFPT)*, pages 399–402, 2019. 24, 27

[152] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002. 26

[153] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Hailong Ma. Multi-objective reinforced evolution in mobile neural architecture search. *CoRR*, abs/1901.01074, 2019. URL http://arxiv.org/abs/1901.01074. 26

[154] X. Zhang, W. Jiang, Y. Shi, and J. Hu. When neural architecture search meets hardware implementation: from hardware awareness to co-design. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 25–30, 2019. 27, 31, 32

[155] G. Bender, H. Liu, B. Chen, G. Chu, S. Cheng, P. J. Kindermans, and Q. V. Le. Can weight sharing outperform random architecture search? an investigation with tunas. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14311–14320, 2020. doi: 10.1109/CVPR42600.2020.01433. 27

[156] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition, 2018. 27, 39

[157] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning, 2016. URL http://arxiv.org/abs/1611.01578. 28

[158] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, page 2902–2911, 2017. 28

[159] Niv Nayman, Asaf Noy, Tal Ridnik, Itamar Friedman, Rong Jin, and Lihi Zelnik-Manor. XNAS: neural architecture search with expert advice. In *Advances in Neural Information Processing System 32 (NeurIPS)*, pages 1975–1985, 2019. URL http://papers.nips.cc/paper/8472-xnas-neural-architecture-search-with-expert-advice. 28, 29, 39, 40

[160] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *7th International Conference on Learning Representations, ICLR*, 2019. 28, 38

[161] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, page 3123–3131, 2015. 28, 29, 33

[162] Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006. ISBN 0521841089. 29

[163] Jyrki Kivinen and Manfred K Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *information and computation*, 132(1):1–63, 1997. 29

[164] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 12219–12228. IEEE, 2021. doi: 10.1109/ICCV48922.2021.01202. 29, 60, 62, 63, 67

[165] Xiang Li, Chen Lin, Chuming Li, Ming Sun, Wei Wu, Junjie Yan, and Wanli Ouyang. Improving one-shot NAS by suppressing the posterior fading. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 13833–13842. Computer Vision Foundation / IEEE, 2020. doi: 10.1109/CVPR42600.2020.01385. 29

[166] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018. URL http://arxiv.org/abs/1806.09055. 29, 61, 65, 67, 81, 121, 133, 134, 137, 138

[167] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *5th International Conference on Learning Representations, ICLR*, 2017. URL https://openreview.net/forum?id=rkE3y85ee. 29

[168] Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. Mixed precision quantization of convnets via differentiable neural architecture search. *CoRR*, abs/1812.00090, 2018. URL http://arxiv.org/abs/1812.00090. 29

[169] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=HylxE1HKwS. 29, 67, 90, 91

[170] Colin White, Willie Neiswanger, and Yash Savani. BANANAS: bayesian optimization with neural architectures for neural architecture search. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI*, pages 10293–10301. AAAI Press, 2021. 30, 131

[171] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, pages 367–377. PMLR, 2020. 30

[172] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, pages 4092–4101, 2018. URL http://proceedings.mlr.press/v80/pham18a.html. 30

[173] Stefan C. Endres, Carl Sandrock, and Walter W. Focke. A simplicial homology algorithm for lipschitz optimisation. *J. Glob. Optim.*, 72(2):181–217, 2018. doi: 10.1007/s10898-018-0645-y. 30

[174] Sean C. Smithson, Guang Yang, Warren J. Gross, and Brett H. Meyer. Neural networks designing neural networks: Multi-objective hyper-parameter optimization. *CoRR*, abs/1611.02120, 2016. URL http://arxiv.org/abs/1611.02120. 30

[175] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. *CoRR*, abs/1711.06798, 2017. URL http://arxiv.org/abs/1711.06798. 30

[176] Halima Bouzidi, Hamza Ouarnoughi, Smail Niar, and Abdessamad Ait El Cadi. Performance prediction for convolutional neural networks in edge devices, 2020. 30

[177] Tien-Ju Yang, Andrew G. Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *ECCV 15th European Conference Computer Vision*, volume 11214, pages 289–304, 2018. doi: 10.1007/978-3-030-01249-6_18. URL https://doi.org/10.1007/978-3-030-01249-6_18. 31

[178] Chengyue Gong, Zixuan Jiang, Dilin Wang, Yibo Lin, Qiang Liu, and David Z Pan. Mixed precision neural architecture search for energy efficient deep learning. In *ICCAD*, pages 1–7, 2019. 31

[179] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, page 754–768, 2019. ISBN 9781450369381. 31

[180] A. Anderson, J. Su, R. Dahyot, and D. Gregg. Performance-oriented neural architecture search. In *International Conference on High Performance Computing Simulation (HPCS)*, pages 177–184, 2019. doi: 10.1109/HPCS48598.2019.9188213. 31

[181] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the*

36th International Conference on Machine Learning, ICML, volume 97 of *Proceedings of Machine Learning Research*, pages 7105–7114. PMLR, 2019. 32, 36, 37

[182] Xishan Zhang, Shaoli Liu, Rui Zhang, Chang Liu, Di Huang, Shiyi Zhou, Jiaming Guo, Yu Kang, Qi Guo, Zidong Du, et al. Adaptive precision training: Quantify back propagation in neural networks with fixed-point numbers. *arXiv preprint arXiv:1911.00361*, 2019. 33

[183] Umar Asif, Jianbin Tang, and Stefan Harrer. Ensemble knowledge distillation for learning improved and efficient networks. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI - 24th European Conference on Artificial Intelligence*, 2020. doi: 10.3233/FAIA200188. URL https://doi.org/10.3233/FAIA200188. 33

[184] Ghouthi Boukli Hacene. *Processing and learning deep neural networks on chip*. Theses, Ecole nationale supérieure Mines-Télécom Atlantique, October 2019. URL https://tel.archives-ouvertes.fr/tel-02438921. 33

[185] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4107–4115, 2016. URL http://papers.nips.cc/paper/6573-binarized-neural-networks. 33

[186] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: hardware-aware automated quantization. *CoRR*, abs/1811.08886, 2018. URL http://arxiv.org/abs/1811.08886. 33

[187] Haibao Yu, Qi Han, Jianbo Li, Jianping Shi, Guangliang Cheng, and Bin Fan. Search what you want: Barrier panelty NAS for mixed precision quantization. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *ECCV - 16th European Conference Computer Vision*, volume 12354, pages 1–16, 2020. doi: 10.1007/978-3-030-58545-7_1. URL https://doi.org/10.1007/978-3-030-58545-7_1. 34

[188] V. Camus, L. Mei, C. Enz, and M. Verhelst. Review and benchmarking of precision-scalable multiply-accumulate unit architectures for embedded neural-network processing. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(4):697–711, 2019. doi: 10.1109/JETCAS.2019.2950386. 34

[189] Seongmin Park, Beomseok Kwon, Kyuyoung Sim, Jieun Lim, Tae-Ho Kim, and Jungwook Choi. Uniform-precision neural network quantization via neural channel expansion, 2021. URL https://openreview.net/forum?id=oGq4d9TbyIA. 34

[190] Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. In *Advances in Neural Information Processing Systems*, pages 760–771, 2019. 34

[191] Mingbao Lin, Rongrong Ji, Yuxin Zhang, Baochang Zhang, Yongjian Wu, and Yonghong Tian. Channel pruning via automatic structure search. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI*, pages 673–679. ijcai.org, 2020. doi: 10.24963/ijcai.2020/94. URL https://doi.org/10.24963/ijcai.2020/94. 34

[192] Xin Li, Yiming Zhou, Zheng Pan, and Jiashi Feng. Partial order pruning: For best speed/accuracy trade-off in neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 34

[193] Danilo Vasconcellos Vargas and Shashank Kotyan. Evolving robust neural architectures to defend from adversarial attacks. *CoRR*, abs/1906.11667, 2019. URL http://arxiv.org/abs/1906.11667. 34

[194] Ekin Dogus Cubuk, Barret Zoph, Samuel S. Schoenholz, and Quoc V. Le. Intriguing properties of adversarial examples. In *6th International Conference on Learning Representations, ICLR*. OpenReview.net, 2018. 34

[195] Minghao Guo, Yuzhe Yang, Rui Xu, Ziwei Liu, and Dahua Lin. When NAS meets robustness: In search of robust architectures against adversarial attacks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 628–637. IEEE, 2020. doi: 10.1109/CVPR42600.2020.00071. URL https://doi.org/10.1109/CVPR42600.2020.00071. 34

[196] Abu Sebastian, Manuel Le Gallo, Riduan Khaddam-Aljameh, and Evangelos Eleftheriou. Memory devices and applications for in-memory computing. *Nature Nanotechnology*, 15(7):529–544, Jul 2020. 34, 101

[197] Vinay Joshi, Manuel Le Gallo, Irem Boybat, Simon Haefeli, Christophe Piveteau, Martino Dazzi, Bipin Rajendran, Abu Sebastian, and Evangelos Eleftheriou. Accurate deep neural network inference using computational phase-change memory. *CoRR*, abs/1906.03138, 2019. 34, 101, 103

[198] Corey Lammie, Wei Xiang, Bernabé Linares-Barranco, and Mostafa Rahimi Azghadi. Memtorch: An open-source simulation framework for memristive deep learning systems. *Neurocomputing*, 485:124–133, 2022. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2022.02.043. 35, 101, 103

[199] Guihong Li, Sumit K. Mandal, Ümit Y. Ogras, and Radu Marculescu. FLASH: fast neural architecture search with hardware optimization. *ACM Trans. Embed. Comput. Syst.*, 20(5s):63:1–63:26, 2021. 35

[200] Weiwen Jiang, Qiuwen Lou, Zheyu Yan, Lei Yang, Jingtong Hu, Xiaobo Sharon Hu, and Yiyu Shi. Device-circuit-architecture co-exploration for computing-in-memory neural accelerators. *IEEE Trans. Computers*, 70(4):595–605, 2021. 35

[201] Zhihang Yuan, Jingze Liu, Xingchen Li, Longhao Yan, Haoxiang Chen, Bingzhe Wu, Yuchao Yang, and Guangyu Sun. NAS4RRAM: neural network architecture search for inference on rram-based accelerators. *Sci. China Inf. Sci.*, 64 (6), 2021. 35

[202] Zheyu Yan, Da-Cheng Juan, Xiaobo Sharon Hu, and Yiyu Shi. Uncertainty modeling of emerging device based computing-in-memory neural accelerators with application to neural architecture search. In *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 859–864, 2021. 35

[203] Guoqing Li, Meng Zhang, Jiaojie Li, Feng Lv, and Guodong Tong. Efficient densely connected convolutional neural networks. *Pattern Recognit.*, 109: 107610, 2021. 35

[204] C. Zhou, F. Redondo, J. Buchel, I. Boybat, X. Comas, S. R. Nandakumar, S. Das, A. Sebastian, M. Le Gallo, and P. N. Whatmough. Ml-hw co-design of noise-robust tinyml models and always-on analog compute-in-memory edge accelerator. *IEEE Micro*, 42(06):76–87, 2022. ISSN 1937-4143. doi: 10.1109/MM.2022.3198321. 35

[205] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJxyZkBKDr. 36, 37

[206] Arber Zela, Julien Siems, and Frank Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. In *8th International Conference on Learning Representations, ICLR*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SJx9ngStPH. 36, 37

[207] Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. Nats-bench: Benchmarking NAS algorithms for architecture topology and size. *CoRR*, abs/2009.00437, 2020. URL https://arxiv.org/abs/2009.00437. 37

[208] Julien Niklas Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. {NAS}-bench-301 and the case for surrogate benchmarks for neural architecture search, 2021. URL https://openreview.net/forum?id=1flmvXGGJaa. 37

[209] Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, Cong Hao, and Yingyan Lin. {HW}-{nas}-bench: Hardware-aware neural architecture search benchmark. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=_0kaDkv3dVf. 37, 39

[210] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. In *6th International Conference on Learning Representations, ICLR*, 2018. URL https://openreview.net/forum?id=SyyGPP0TZ. 38

[211] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Comput. Linguistics*, 19(2):313–330, 1993. 38

[212] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Byj72udxe. 38

[213] Joseph Mellor, Jack Turner, Amos J. Storkey, and Elliot J. Crowley. Neural architecture search without training. *CoRR*, abs/2006.04647, 2020. URL https://arxiv.org/abs/2006.04647. 38, 145

[214] G. J. van Wyk and A. S. Bosman. Evolutionary neural architecture search for image restoration. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019. 39

[215] Haokui Zhang, Ying Li, Hao Chen, and Chunhua Shen. IR-NAS: neural architecture search for image restoration. *CoRR*, abs/1909.08228, 2019. URL http://arxiv.org/abs/1909.08228. 39

[216] Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. Autogan: Neural architecture search for generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3224–3234, 2019. 39

[217] Kary Ho, Andrew Gilbert, Hailin Jin, and John P. Collomosse. Neural architecture search for deep image prior. *CoRR*, abs/2001.04776, 2020. URL https://arxiv.org/abs/2001.04776. 39

[218] Zhuotun Zhu, Chenxi Liu, Dong Yang, Alan Yuille, and Daguang Xu. V-nas: Neural architecture search for volumetric medical image segmentation. In *2019 International Conference on 3D Vision (3DV)*, pages 240–248. IEEE, 2019. 39

[219] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015. 39

[220] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018. 39

[221] Zhichao Lu, Gautam Sreekumar, Erik D. Goodman, Wolfgang Banzhaf, Kalyanmoy Deb, and Vishnu Naresh Boddeti. Neural architecture transfer. *CoRR*, abs/2005.05859, 2020. URL https://arxiv.org/abs/2005.05859. 39

[222] Marcelo Gennari Do Nascimento, Theo W. Costain, and Victor Adrian Prisacariu. Finding non-uniform quantization schemes using multi-task gaussian processes. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *ECCV - 16th European Conference Computer Vision*, volume 12362, pages 383–398, 2020. 41

[223] Han Cai, Ji Lin, Yujun Lin, Zhijian Liu, Haotian Tang, Hanrui Wang, Ligeng Zhu, and Song Han. Enable deep learning on mobile devices: Methods, systems, and applications. *ACM Trans. Des. Autom. Electron. Syst.*, 27(3), 2022. ISSN 1084-4309. doi: 10.1145/3486618. 46

[224] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smaïl Niar, Martin Wistuba, and Naigang Wang. A comprehensive survey on hardware-aware neural architecture search. *CoRR*, abs/2101.09336, 2021. 47, 51, 62

[225] Hadjer Benmeziane, Hamza Ouarnoughi, Kaoutar El Maghraoui, and Smaïl Niar. Accelerating neural architecture search with rank-preserving surrogate models. In *The 7th Annual International Conference on Arab Women*, pages 21:1–21:6. ACM, 2021. 47

[226] Hadjer Benmeziane, Smaïl Niar, Hamza Ouarnoughi, and Kaoutar El Maghraoui. Pareto rank surrogate model for hardware-aware neural architecture search. In *International IEEE Symposium on Performance Analysis of Systems and Software, ISPASS 2022, Singapore, May 22-24, 2022*, pages 267–276. IEEE, 2022. doi: 10.1109/ISPASS55109.2022.00040. URL https://doi.org/10.1109/ISPASS55109.2022.00040. 47, 68

[227] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJU4ayYgl. 50

[228] Vincenzo Di Massa, Gabriele Monfardini, Lorenzo Sarti, Franco Scarselli, Marco Maggini, and Marco Gori. A comparison between recursive neural networks and graph neural networks. In *International Joint Conference on Neural Networks*, pages 778–785. IEEE, 2006. 50

[229] Llukan Puka. Kendall's tau. In Miodrag Lovric, editor, *International Encyclopedia of Statistical Science*, pages 713–715. Springer, 2011. 50

[230] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 785–794. ACM, 2016. doi: 10.1145/2939672.2939785. URL https://doi.org/10.1145/2939672.2939785. 51

[231] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3146–3154, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html. 51

[232] Michael D. McKay. Latin hypercube sampling as a tool in uncertainty analysis of computer models. In Robert C. Crain, editor, *Proceedings of the 24th Winter Simulation Conference, Arlington, VA, USA, December 13-16, 1992*, pages 557–564. ACM Press, 1992. 53, 105

[233] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. 54, 55, 61, 65

[234] Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, and Joseph E. Gonzalez. Fbnetv3: Joint architecture-recipe search using predictor pretraining. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 16276–16285. Computer Vision Foundation / IEEE, 2021. 54, 55, 58

[235] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *7th International Conference on Learning Representations, ICLR*. OpenReview.net, 2019. 54, 55, 57, 58, 60, 61, 65, 67, 138

[236] Martin Wistuba and Tejaswini Pedapati. Learning to rank learning curves. In *Proceedings of the 37th International Conference on Machine Learning*, pages 10303–10312. PMLR, 2020. 54, 55, 145

[237] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the CIFAR datasets. *CoRR*, abs/1707.08819, 2017. URL http://arxiv.org/abs/1707.08819. 55, 65

[238] Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, Cong Hao, and Yingyan Lin. Hw-nas-bench: Hardware-aware neural architecture search benchmark. In *9th International Conference on Learning Representations, ICLR 2021*, 2021. 55

[239] Ting-Ting Wang, Shu-Chuan Chu, Chia-Cheng Hu, Han-Dong Jia, and Jeng-Shyang Pan. Efficient network architecture search using hybrid optimizer. *Entropy*, 24(5):656, 2022. doi: 10.3390/e24050656. URL https://doi.org/10.3390/e24050656. 57

[240] Han Xiao, Ziwei Wang, Jiwen Lu, and Jie Zhou. Shapley-NAS: Discovering operation contribution for neural architecture search, 2022. URL https://openreview.net/forum?id=F7nD--1JIC. 57

[241] Jovita Lukasik, Steffen Jung, and Margret Keuper. Learning where to look - generative NAS is surprisingly efficient. *CoRR*, abs/2203.08734, 2022. doi: 10.48550/arXiv.2203.08734. URL https://doi.org/10.48550/arXiv.2203.08734. 57

[242] Peng Ye, Baopu Li, Yikang Li, Tao Chen, Jiayuan Fan, and Wanli Ouyang. β-darts: Beta-decay regularization for differentiable architecture search. *CoRR*, abs/2203.01665, 2022. doi: 10.48550/arXiv.2203.01665. URL https://doi.org/10.48550/arXiv.2203.01665. 57

[243] Linnan Wang, Chenhan Yu, Satish Salian, Slawomir Kierat, Szymon Migacz, and Alex Fit-Florea. Gpunet: Searching the deployable convolution neural networks for gpus. *CoRR*, abs/2205.00841, 2022. doi: 10.48550/arXiv.2205.00841. URL https://doi.org/10.48550/arXiv.2205.00841. 57, 58

[244] Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, and Joseph E. Gonzalez. Fbnetv3: Joint architecture-recipe search using predictor pretraining. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 16276–16285. Computer Vision Foundation / IEEE, 2021. 57, 90, 91

[245] Peter Mølgaard Sørensen, Bastian Epp, and Tobias May. A depthwise separable convolutional neural network for keyword spotting on an embedded system. *EURASIP J. Audio Speech Music. Process.*, 2020(1):10, 2020. 59, 112

[246] Xi Chen, Shouyi Yin, Dandan Song, Peng Ouyang, Leibo Liu, and Shaojun Wei. Small-footprint keyword spotting with graph convolutional network. In *IEEE Automatic Speech Recognition and Understanding Workshop, ASRU 2019, Singapore, December 14-18, 2019*, pages 539–546. IEEE, 2019. doi: 10.1109/ASRU46091.2019.9004005. URL https://doi.org/10.1109/ASRU46091.2019.9004005. 59

[247] Kevin Ding, Martin Zong, Jiakui Li, and Baoxiang Li. LETR: A lightweight and efficient transformer for keyword spotting. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2022, Virtual and Singapore, 23-27 May 2022*, pages 7987–7991. IEEE, 2022. doi: 10.1109/ICASSP43922.2022.9747295. URL https://doi.org/10.1109/ICASSP43922.2022.9747295. 59

[248] Axel Berg, Mark O'Connor, and Miguel Tairum Cruz. Keyword transformer: A self-attention model for keyword spotting. In Hynek Hermansky, Honza Cernocký, Lukás Burget, Lori Lamel, Odette Scharenborg, and Petr Motlícek, editors, *Interspeech 2021, 22nd Annual Conference of the International Speech Communication Association, Brno, Czechia, 30 August - 3 September 2021*, pages 4249–4253. ISCA, 2021. doi: 10.21437/Interspeech.2021-1286. URL https://doi.org/10.21437/Interspeech.2021-1286. 59

[249] Assaf Hurwitz Michaely, Xuedong Zhang, Gabor Simko, Carolina Parada, and Petar Aleksic. Keyword spotting for google assistant using contextual speech recognition. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 272–278, 2017. doi: 10.1109/ASRU.2017.8268946. 59

[250] Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim V. Fedorov, Alexander Filippov, and Evgeny Burnaev. Nas-bench-nlp: Neural architecture search benchmark for natural language processing. *IEEE Access*, 10:45736–45747, 2022. doi: 10.1109/ACCESS.2022.3169897. URL https://doi.org/10.1109/ACCESS.2022.3169897. 59

[251] Yuge Zhang, Zejun Lin, Junyang Jiang, Quanlu Zhang, Yujing Wang, Hui Xue, Chen Zhang, and Yaming Yang. Deeper insights into weight sharing in neural architecture search. *CoRR*, abs/2001.01431, 2020. 60

[252] Jiefeng Peng, Jiqi Zhang, Changlin Li, Guangrun Wang, Xiaodan Liang, and Liang Lin. Pi-nas: Improving neural architecture search by reducing supernet training consistency shift. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 12334–12344. IEEE, 2021. doi: 10.1109/ICCV48922.2021.01213. URL https://doi.org/10.1109/ICCV48922.2021.01213. 60

[253] Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. Few-shot neural architecture search. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 12707–12718. PMLR, 2021. URL http://proceedings.mlr.press/v139/zhao21d.html. 60

[254] Lingxi Xie, Xin Chen, Kaifeng Bi, Longhui Wei, Yuhui Xu, Lanfei Wang, Zhengsu Chen, An Xiao, Jianlong Chang, Xiaopeng Zhang, and Qi Tian. Weight-sharing neural architecture search: A battle to shrink the optimization gap. *ACM Comput. Surv.*, 54(9):183:1–183:37, 2022. doi: 10.1145/3473330. URL https://doi.org/10.1145/3473330. 60

[255] Xin Chen, Lingxi Xie, Jun Wu, Longhui Wei, Yuhui Xu, and Qi Tian. Fitting the search space of weight-sharing NAS with graph convolutional networks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, 2021*, pages 7064–7072. AAAI Press, 2021. 60, 61

[256] Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, Cong Hao, and Yingyan Lin. Hw-nas-bench: Hardware-aware neural architecture search benchmark. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=_0kaDkv3dVf. 60, 65, 66

[257] Michael T. M. Emmerich, André H. Deutz, and Jan Willem Klinkenberg. Hypervolume-based expected improvement: Monotonicity properties and exact computation. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2011, New Orleans, LA, USA, 5-8 June, 2011*, pages 2147–2154. IEEE, 2011. doi: 10.1109/CEC.2011.5949880. URL https://doi.org/10.1109/CEC.2011.5949880. 63

[258] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4092–4101. PMLR, 2018. URL http://proceedings.mlr.press/v80/pham18a.html. 67

[259] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four GPU hours. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 1761–1770. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00186. URL http://openaccess.thecvf.com/content_CVPR_2019/html/Dong_Searching_for_a_Robust_Neural_Architecture_in_Four_GPU_Hours_CVPR_2019_paper.html. 67

[260] Xuefei Ning, Yin Zheng, Tianchen Zhao, Yu Wang, and Huazhong Yang. A generic graph-based neural architecture encoding scheme for predictor-based NAS. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XIII*, volume 12358 of *Lecture Notes in Computer Science*, pages 189–204. Springer, 2020. doi: 10.1007/978-3-030-58601-0_12. URL https://doi.org/10.1007/978-3-030-58601-0_12. 68

[261] Lukasz Dudziak, Thomas Chau, Mohamed S. Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas D. Lane. BRP-NAS: prediction-based NAS using gcns. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 68

[262] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. 75, 79

[263] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km. 76

[264] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *36th International Conference on Machine Learning*, volume 97, pages 7105–7114. PMLR, 2019. 81

[265] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *6th International Conference on Learning Representations, ICLR*, 2018. 81

[266] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems*, 2020. 82, 112, 146

[267] Wei Niu, Jiexiong Guan, Yanzhi Wang, Gagan Agrawal, and Bin Ren. Dnnfusion: accelerating deep neural networks execution with advanced operator fusion. In Stephen N. Freund and Eran Yahav, editors, *42nd ACMInternational Conference on Programming Language Design and Implementation PLDI*, pages 883–898, 2021. 83

[268] Hadjer Benmeziane, Smaïl Niar, Hamza Ouarnoughi, and Kaoutar El Maghraoui. Pareto rank surrogate model for hardware-aware neural architecture search. In *International IEEE Symposium on Performance Analysis of Systems and Software, ISPASS*, pages 267–276, 2022. 84

[269] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 770–778. IEEE Computer Society, 2016. 87, 89, 105, 113

[270] Xiangxiang Chu and Xinjie Yu. Improved crowding distance for NSGA-II. *CoRR*, abs/1811.12667, 2018. 88

[271] G. Jignesh Chowdary, Narinder Singh Punn, Sanjay Kumar Sonbhadra, and Sonali Agarwal. Face mask detection using transfer learning of inceptionv3. In Ladjel Bellatreche, Vikram Goyal, Hamido Fujita, Anirban Mondal, and P. Krishna Reddy, editors, *Big Data Analytics - 8th International Conference, BDA*, volume 12581, pages 81–90, 2020. 89

[272] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Conference on Computer Vision and Pattern Recognition, CVPR*, pages 4510–4520, 2018. 89

[273] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR*, 2016. 90

[274] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 1800–1807. IEEE Computer Society, 2017. 91

[275] Pankaj, Ashish Kumar, Rama Komaragiri, and Manjeet Kumar. A review on computation methods used in photoplethysmography signal analysis for heart rate estimation. *Archives of Computational Methods in Engineering*, 29(2): 921–940, 2022. 93

[276] Xiangmao Chang, Gangkai Li, Guoliang Xing, Kun Zhu, and Linlin Tu. Deepheart: A deep learning approach for accurate heart rate estimation from PPG signals. *ACM Trans. Sens. Networks*, 17(2):14:1–14:18, 2021. 93, 94, 95

[277] Heewon Chung, Hoon Ko, Hooseok Lee, and Jinseok Lee. Deep learning for heart rate estimation from reflectance photoplethysmography with acceleration power spectrum and acceleration intensity. *IEEE Access*, 8:63390–63402, 2020. 93, 94, 95

[278] Seok Bin Song, Jung Woo Nam, and Jin Heon Kim. Nas-ppg: Ppg-based heart rate estimation using neural architecture search. *IEEE Sensors Journal*, 21 (13):14941–14949, 2021. doi: 10.1109/JSEN.2021.3073047. 93, 94, 95

[279] Mouna Benchekroun, Baptiste Chevallier, Hamza Beaouiss, Dan Istrate, Vincent Zalc, Mohamad Khalil, and Dominique Lenne. Comparison of stress detection through ECG and PPG signals using a random forest-based algorithm. In *44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society, EMBC 2022, Glasgow, Scotland, United Kingdom, July 11-15, 2022*, pages 3150–3153. IEEE, 2022. 94

[280] I. Boybat, B. Kersting, S. Ghazi Sarwat, X. Timoneda, R. L. Bruce, M. BrightSky, M. Le Gallo, and A. Sebastian. Temperature sensitivity of analog in-memory computing using phase-change memory. In *2021 IEEE International Electron Devices Meeting (IEDM)*, pages 28.3.1–28.3.4, 2021. doi: 10.1109/IEDM19574.2021.9720519. 101

[281] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 102

[282] Gangzhao Lu, Weizhe Zhang, and Zheng Wang. Optimizing depthwise separable convolution operations on gpus. *IEEE Transactions on Parallel and Distributed Systems*, 33(1):70–87, 2022. doi: 10.1109/TPDS.2021.3084813. 102

[283] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smaïl Niar, Martin Wistuba, and Naigang Wang. A comprehensive survey on hardware-aware neural architecture search. *CoRR*, abs/2101.09336, 2021. 102

[284] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 5987–5995. IEEE Computer Society, 2017. 102, 105

[285] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Richard C. Wilson, Edwin R. Hancock, and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference, BMVC*. BMVA Press, 2016. 102, 104, 105, 111, 116

[286] Malte J. Rasch, Diego Moreda, Tayfun Gokmen, Manuel Le Gallo, Fabio Carta, Cindy Goldberg, Kaoutar El Maghraoui, Abu Sebastian, and Vijay Narayanan. A flexible and fast pytorch toolkit for simulating training and inference on analog crossbar arrays. In *3rd IEEE International Conference on Artificial Intelligence Circuits and Systems*, pages 1–4. IEEE, 2021. 103, 106

[287] Malte J. Rasch, Charles Mackin, Manuel Le Gallo, An Chen, Andrea Fasoli, Frederic Odermatt, Ning Li, S. R. Nandakumar, Pritish Narayanan, Hsinyu Tsai, Geoffrey W. Burr, Abu Sebastian, and Vijay Narayanan. Hardware-aware training for large-scale and diverse deep learning inference workloads using in-memory computing-based accelerators, 2023. URL https://arxiv.org/abs/2302.08469. 103

[288] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 10096–10106. PMLR, 2021. 104

[289] Hadjer Benmeziane, Smaïl Niar, Hamza Ouarnoughi, and Kaoutar El Maghraoui. Pareto rank surrogate model for hardware-aware neural architecture search. In *International IEEE Symposium on Performance Analysis of Systems and Software, ISPASS*, pages 267–276. IEEE, 2022. 107

[290] Hervé Abdi. The kendall rank correlation coefficient. *Encyclopedia of Measurement and Statistics. Sage, Thousand Oaks, CA*, pages 508–510, 2007. 107

[291] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 110

[292] Aakanksha Chowdhery, Pete Warden, Jonathon Shlens, Andrew Howard, and Rocky Rhodes. Visual wake words dataset. *CoRR*, abs/1906.05721, 2019. 110

[293] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Haibin Lin, Zhi Zhang, Yue Sun, Tong He, Jonas Mueller, R. Manmatha, Mu Li, and Alexander J. Smola. Resnest: Split-attention networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, CVPR*, pages 2735–2745. IEEE, 2022. 110

[294] Mahesh Chandra Mukkamala and Matthias Hein. Variants of rmsprop and adagrad with logarithmic regret bounds. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 2545–2553. PMLR, 2017. 110

[295] P. Warden. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *ArXiv e-prints*, April 2018. URL https://arxiv.org/abs/1804.03209. 110

[296] Jorge Martínez, Héctor Pérez-Meana, Enrique Escamilla Hernández, and Masahisa Mabo Suzuki. Speaker recognition using mel frequency cepstral coefficients (MFCC) and vector quantization (VQ) techniques. In Pedro Bañuelos Sánchez, Roberto Rosas-Romero, and Mauricio Javier Osorio Galindo, editors, *22nd International Conference on Electrical Communications and Computers, CONIELECOMP*, pages 248–251. IEEE, 2012. 110

[297] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations*, 2015. 110

[298] Colby R. Banbury, Vijay Janapa Reddi, Peter Torelli, Nat Jeffries, Csaba Király, Jeremy Holleman, Pietro Montino, David Kanter, Pete Warden, Danilo Pau, Urmish Thakker, Antonio Torrini, Jay Cordaro, Giuseppe Di Guglielmo, Javier M. Duarte, Honson Tran, Nhan Tran, Wenxu Niu, and Xuesong Xu. Mlperf tiny benchmark. In Joaquin Vanschoren and Sai-Kit Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. 111

[299] Manuel Le Gallo, Riduan Khaddam-Aljameh, Milos Stanisavljevic, Athanasios Vasilopoulos, Benedikt Kersting, Martino Dazzi, Geethan Karunaratne, Matthias Braendli, Abhairaj Singh, Silvia M. Mueller, Julian Buechel, Xavier Timoneda, Vinay Joshi, Urs Egger, Angelo Garofalo, Anastasios Petropoulos, Theodore Antonakopoulos, Kevin Brew, Samuel Choi, Injo Ok, Timothy Philip, Victor Chan, Claire Silvestre, Ishtiaq Ahsan, Nicole Saulnier, Vijay Narayanan, Pier Andrea Francese, Evangelos Eleftheriou, and Abu Sebastian. A 64-core mixed-signal in-memory compute chip based on phase-change memory for deep neural network inference, 2022. URL https://arxiv.org/abs/2212.02872. 113

[300] R. Khaddam-Aljameh, M. Stanisavljevic, J. Fornt Mas, G. Karunaratne, M. Braendli, F. Liu, A. Singh, S. M. Müller, U. Egger, A. Petropoulos, T. Antonakopoulos, K. Brew, S. Choi, I. Ok, F. L. Lie, N. Saulnier, V. Chan, I. Ahsan, V. Narayanan, S. R. Nandakumar, M. Le Gallo, P. A. Francese, A. Sebastian, and E. Eleftheriou. Hermes core – a 14nm cmos and pcm-based in-memory compute core using an array of 300ps/lsb linearized cco-based adcs and local digital processing. In *2021 Symposium on VLSI Technology*, pages 1–2, 2021. 113

[301] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells III, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention - MICCAI*, volume 9351 of *Lecture Notes in Computer Science*, pages 234–241. Springer, 2015. 118, 121, 131, 140

[302] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: A nested u-net architecture for medical image segmentation. In Danail Stoyanov, Zeike Taylor, Gustavo Carneiro, Tanveer F. Syeda-Mahmood, Anne L. Martel, Lena Maier-Hein, João Manuel R. S. Tavares, Andrew P. Bradley, João Paulo Papa, Vasileios Belagiannis, Jacinto C. Nascimento, Zhi Lu, Sailesh Conjeti, Mehdi Moradi, Hayit Greenspan, and Anant Madabhushi, editors, *Deep Learning in Medical Image Analysis - and - Multimodal Learning for Clinical Decision Support - 4th International Workshop, DLMIA , and 8th International Workshop, ML-CDS 2018, Held in Conjunction with MICCAI*, volume 11045 of *Lecture Notes in Computer Science*, pages 3–11. Springer, 2018. 118, 131, 140

[303] Ozan Oktay, Jo Schlemper, Loïc Le Folgoc, Matthew C. H. Lee, Mattias P. Heinrich, Kazunari Misawa, Kensaku Mori, Steven G. McDonagh, Nils Y. Hammerla, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. Attention u-net: Learning where to look for the pancreas. *CoRR*, abs/1804.03999, 2018. 118, 131, 140

[304] Luyan Liu, Zhiwei Wen, Songwei Liu, Hong-Yu Zhou, Hongwei Zhu, Weicheng Xie, Linlin Shen, Kai Ma, and Yefeng Zheng. Mixsearch: Searching for domain generalized medical image segmentation architectures. *CoRR*, abs/2102.13280, 2021. 118, 129, 131, 140

[305] Qihang Yu, Dong Yang, Holger Roth, Yutong Bai, Yixiao Zhang, Alan L. Yuille, and Daguang Xu. C2FNAS: coarse-to-fine neural architecture search for 3d medical image segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 4125–4134. Computer Vision Foundation / IEEE, 2020. 118, 129, 130, 131, 140

[306] Xinyi Wang, Tiange Xiang, Chaoyi Zhang, Yang Song, Dongnan Liu, Heng Huang, and Weidong Cai. Bix-nas: Searching efficient bi-directional architecture for medical image segmentation. In Marleen de Bruijne, Philippe C. Cattin, Stéphane Cotin, Nicolas Padoy, Stefanie Speidel, Yefeng Zheng, and Caroline Essert, editors, *Medical Image Computing and Computer Assisted Intervention - MICCAI*, volume 12901 of *Lecture Notes in Computer Science*, pages 229–238. Springer, 2021. 118, 129, 130, 131, 140

[307] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3462–3471, 2017. 120

[308] Fabian Isensee, Paul F Jaeger, Simon AA Kohl, Jens Petersen, and Klaus H Maier-Hein. nnu-net: a self-configuring method for deep learning-based biomedical image segmentation. *Nature methods*, 18(2):203–211, 2021. 131, 140

[309] Zhichao Lu, Kalyanmoy Deb, Erik D. Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I*, volume 12346 of *Lecture Notes in Computer Science*, pages 35–51. Springer, 2020. doi: 10.1007/978-3-030-58452-8_3. URL https://doi.org/10.1007/978-3-030-58452-8_3. 131

[310] Chakkrit Termritthikun, Yeshi Jamtsho, Jirarat Ieamsaard, Paisarn Muneesawang, and Ivan Lee. Eeea-net: An early exit evolutionary neural architecture search. *Eng. Appl. Artif. Intell.*, 104:104397, 2021. 131

[311] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I. Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S. Davis. NISP: pruning networks using neuron importance score propagation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 9194–9203. Computer Vision Foundation / IEEE Computer Society, 2018. 134, 135

[312] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. PC-DARTS: partial channel connections for memory-efficient architecture search. In *8th International Conference on Learning Representations, ICLR*. OpenReview.net, 2020. 138

[313] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *IEEE/CVF International Conference on Computer Vision, ICCV*, pages 1294–1303. IEEE, 2019. 138

[314] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey E. Hinton. Similarity of neural network representations revisited. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3519–3529. PMLR, 2019. URL http://proceedings.mlr.press/v97/kornblith19a.html. 145

[315] Daniela Szwarcman, Daniel Civitarese, and Marley Vellasco. Quantum-inspired neural architecture search. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019. doi: 10.1109/IJCNN.2019.8852453. 146