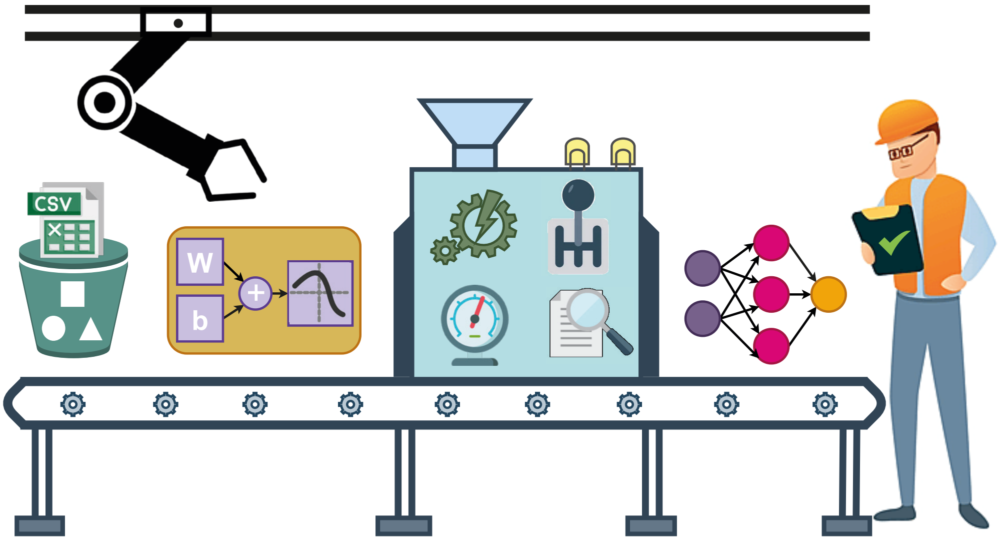
Mälardalen University Doctoral Dissertation 363

Efficient Design of Scalable Deep Neural Networks for Resource-Constrained Edge Devices

Mohammad Loni



Mälardalen University

Mälardalen University Press Dissertations
No. 363

EFFICIENT DESIGN OF SCALABLE DEEP NEURAL NETWORKS FOR RESOURCE-CONSTRAINED EDGE DEVICES

Mohammad Loni

2022

School of Innovation, Design and Engineering

ISBN 978-91-7485-563-0
ISSN 1651-4238
Printed by E-Print AB, Stockholm, Sweden

Mälardalen University Press Dissertations
No. 363

EFFICIENT DESIGN OF SCALABLE DEEP NEURAL NETWORKS FOR RESOURCE-CONSTRAINED EDGE DEVICES

Mohammad Loni

Akademisk avhandling

som för avläggande av teknologie doktorsexamen i datavetenskap vid Akademin för innovation, design och teknik kommer att offentligen försvaras torsdagen den 13 oktober 2022, 13.30 i Delta och online, Mälardalens universitet, Västerås.

Fakultetsopponent: Professor Franz Pernkopf, TU Graz

Akademin för innovation, design och teknik

Abstract

Deep Neural Networks (DNNs) are increasingly being processed on resource-constrained edge nodes (computer nodes used in, e.g., cyber-physical systems or at the edge of computational clouds) due to efficiency, connectivity, and privacy concerns. This thesis investigates and presents new techniques to design and deploy DNNs for resource-constrained edge nodes. We have identified two major bottlenecks that hinder the proliferation of DNNs on edge nodes: (i) the significant computational demand for designing DNNs that consumes a low amount of resources in terms of energy, latency, and memory footprint; and (ii) further conserving resources by quantizing the numerical calculations of a DNN provides remarkable accuracy degradation.

To address (i), we present novel methods for cost-efficient Neural Architecture Search (NAS) to automate the design of DNNs that should meet multifaceted goals such as accuracy and hardware performance. To address (ii), we extend our NAS approach to handle the quantization of numerical calculations by using only the numbers -1, 0, and 1 (so-called ternary DNNs), which achieves higher accuracy. Our experimental evaluation shows that the proposed NAS approach can provide a 5.25x reduction in design time and up to 44.4x reduction in network size compared to state-of-the-art methods. In addition, the proposed quantization approach delivers 2.64% higher accuracy and 2.8x memory saving compared to full-precision counterparts with the same bit-width resolution. These benefits are attained over a wide range of commercial-off-the-shelf edge nodes showing this thesis successfully provides seamless deployment of DNNs on resource-constrained edge nodes.

ISBN 978-91-7485-563-0
ISSN 1651-4238

Sammanfattning

Deep Neural Networks (DNN) bearbetas alltmer på resursbegränsade kantnoder (datornoder som används i t.ex. cyberfysiska system eller i utkanten av beräkningsmoln) på grund av effektivitet, anslutningsmöjligheter och integritetsproblem. Denna avhandling undersöker och presenterar nya tekniker för att designa och distribuera DNN:er för resursbegränsade kantnoder. Vi har identifierat två stora flaskhalsar som hindrar spridningen av DNN på kantnoder: (i) det betydande beräkningsbehovet för att designa DNN som förbrukar en låg mängd resurser i termer av energi, latens och minnesfotavtryck; och (ii) ytterligare bevarande av resurser genom att kvantisera de numeriska beräkningarna av en DNN ger en anmärkningsvärd noggrannhetsförsämring.

För att ta itu med (i), presenterar vi nya metoder för kostnadseffektiv Neural Architecture Search (NAS) för att automatisera designen av DNN:er som ska uppfylla mångfacetterade mål som noggrannhet och hårdvaruprestanda. För att ta itu med (ii) utökar vi vår NAS-metod för att hantera kvantiseringen av numeriska beräkningar genom att endast använda siffrorna -1, 0 och 1 (så kallade ternära DNN), vilket uppnår högre noggrannhet. Vår experimentella utvärdering visar att den föreslagna NAS-metoden kan ge en 5,25$\times$ minskning av designtid och upp till 44,4$\times$ minskning av nätverksstorlek jämfört med state-of-the-art metoder. Dessutom ger den föreslagna kvantiseringsmetoden 2,64% högre noggrannhet och 2,8$\times$ minnesbesparing jämfört med motsvarigheter med full precision med samma bitbreddsupplösning. Dessa fördelar uppnås över ett brett utbud av kommersiella kantnoder, vilket visar att denna avhandling framgångsrikt tillhandahåller sömlös distribution av DNN på resursbegränsade kantnoder.

Abstract

Deep Neural Networks (DNNs) are increasingly being processed on resource-constrained edge nodes (computer nodes used in, e.g., cyber-physical systems or at the edge of computational clouds) due to efficiency, connectivity, and privacy concerns. This thesis investigates and presents new techniques to design and deploy DNNs for resource-constrained edge nodes. We have identified two major bottlenecks that hinder the proliferation of DNNs on edge nodes: (i) the significant computational demand for designing DNNs that consumes a low amount of resources in terms of energy, latency, and memory footprint; and (ii) further conserving resources by quantizing the numerical calculations of a DNN provides remarkable accuracy degradation.

To address (i), we present novel methods for cost-efficient Neural Architecture Search (NAS) to automate the design of DNNs that should meet multifaceted goals such as accuracy and hardware performance. To address (ii), we extend our NAS approach to handle the quantization of numerical calculations by using only the numbers -1, 0, and 1 (so-called ternary DNNs), which achieves higher accuracy. Our experimental evaluation shows that the proposed NAS approach can provide a 5.25× reduction in design time and up to 44.4× reduction in network size compared to state-of-the-art methods. In addition, the proposed quantization approach delivers 2.64% higher accuracy and 2.8× memory saving compared to full-precision counterparts with the same bit-width resolution. These benefits are attained over a wide range of commercial-off-the-shelf edge nodes showing this thesis successfully provides seamless deployment of DNNs on resource-constrained edge nodes.

If you’re the smartest person in a room,
you’re in the wrong room!

Acknowledgments

As a pleasant part of my life, this ongoing journey has been full of unique and memorable moments, and many people had supporting roles to play in different stages of the journey. First, my sincere thanks go to the team of my supervisors. I would like to thank Prof. Mikael Sjödin, for his big encouragement and always supporting me. I am deeply grateful to Prof. Masoud Daneshtalab for always being supportive and kind to me.

I am grateful to my colleagues, Prof. Arash GharehBaghi, and Prof. Hadi Esmaeilzadeh, for providing feedback as co-authors of my published papers. I would like to express my deep gratitude to Prof. Marius Lindauer for giving me the opportunity to visit his research group. Taking part in the everyday morning meetings and weekly paper reading meetings was a completely different and exciting experience. I would like to thank Byung Hoon Ahn for the discussion and for teaching me how to write an advanced scientific article.

Many thanks to my dear friend, Mohammad Riazati (Ramon), for his compassionate advice on my personal life. I would like to thank my parents, my brothers, and close friends for all their acts of kindness.

Above all, I would like to give special thanks to my sweetheart, Fatemeh Poursalim, for her unwavering support and patience with me. Without her, this thesis could not have been conducted.

Mohammad Loni, Västerås, October 2022

List of publications

Papers included in the thesis[1]

Paper A *DeepMaker: A multi-objective optimization framework for deep neural networks in embedded systems*, Mohammad Loni, Sima Sinaei, Ali Zoljodi, Masoud Daneshtalab, Mikael Sjödin. In the Microprocessors and Microsystems Journal, 2020, Elsevier (IF=3.503).

Paper B *TOT-Net: An Endeavour Toward Optimizing Ternary Neural Networks*, Najmeh Nazari, Mohammad Loni, Mostafa E. Salehi, Masoud Daneshtalab, Mikael Sjödin. In the Proceedings of IEEE International Conference on Digital System Design (DSD 2019). Chalkidiki, Greece, August 2019.

Paper C *TAS: Ternarized Neural Architecture Search for Resource-Constrained Edge Devices*, Mohammad Loni, Hamid Mousavi, Mohammad Riazati, Masoud Daneshtalab, and Mikael Sjödin. In the proceeding of IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE). Antwerp, Belgium, March 2022.

Paper D *FastStereoNet: A Fast Neural Architecture Search for Improving the Inference of Disparity Estimation on Resource-Limited Platforms*, Mohammad Loni, Ali Zoljodi, Amin Majd, Byung Hoon Ahn, Masoud Daneshtalab, Mikael Sjödin, and Hadi Esmaeilzadeh. In the Transactions on Systems, Man, and Cybernetics: Systems, 2021, IEEE (IF=11.471).

[1]The included articles have been reformatted to comply with the thesis layout.

Paper E *NAS-PxAF: Neural Architecture Search for Accurate Detecting Paroxysmal Atrial Fibrillation*, Mehdi Asadi, Mohammad Loni, Masoud Daneshtalab, Mikael Sjödin, and Arash Gharehbaghi. In the Transactions on Systems, Man, and Cybernetics: Systems, 2022, IEEE (*Under Review*).

Additional papers, not included in the thesis

1. *A Review on Deep Learning Methods for ECG Arrhythmia Classification*, Zahra Ebrahimi, Mohammad Loni, Masoud Daneshtalab, and Arash Gharehbaghi. In the Expert Systems with Applications, 2020, Elsevier (IF=8.665).

2. *PR-DARTS: Pruning-Based Differentiable Architecture Search*, Mousavi, Hamid, Mohammad Loni, Mina Alibeigi, and Masoud Daneshtalab. Transactions on Neural Networks and Learning Systems, 2022, IEEE (*Under Review*)

Contents

I Thesis **1**

1 Introduction **3**

1.1 Introduction . . . 3

1.2 Thesis Outline . . . 6

2 Background **9**

2.1 Deep Learning . . . 9

2.1.1 Theory behind Neural Networks . . . 10

Transfer Function . . . 10

Neural Network Training . . . 12

Performance Generalization . . . 12

2.1.2 Convolutional Neural Network . . . 13

2.2 Neural Architecture Search (NAS) . . . 15

2.2.1 Search Space . . . 16

2.2.2 Search Method . . . 18

2.2.3 Evaluation Strategy . . . 20

3 Research Statement **23**

3.1 Problem Statement . . . 23

3.2 Research Goals . . . 25

3.3 Research Challenges . . . 26

3.4 Research Methodology . . . 27

3.4.1 Problem definition . . . 28

3.4.2 Consolidate an idea . . . 28

3.4.3 Implementation . . . 29

3.4.4 Evaluation 29

4 Research Contribution **31**

4.1 Contributions Addressing the Research Goals 31

4.1.1 Contribution of subgoal 1 31

4.1.2 Contribution of subgoal 2 31

4.1.3 Contribution of subgoal 3 32

4.1.4 Contribution of subgoal 4 32

4.1.5 Contribution of subgoal 5 33

4.2 Overview of the Included Papers 33

4.2.1 Paper A 33

4.2.2 Paper B 34

4.2.3 Paper C 35

4.2.4 Paper D 36

4.2.5 Paper E 37

4.2.6 Mapping Contributions to Subgoals 38

5 Related Work **39**

5.1 Neural Architecture Search 39

5.1.1 Macro NAS 39

5.1.2 Micro NAS 40

5.1.3 Improving NAS Efficiency 41

5.2 Network Ternarization 41

6 Discussion, Conclusion and Future Work **43**

6.1 Discussion and Conclusion 43

6.1.1 Thesis Storyline. 44

6.1.2 Disparity Estimating Performance 45

6.1.3 Ternary Neural Networks (TNNs) Performance 46

6.1.4 Paroxysmal Atrial Fibrillation Classification Performance 47

6.2 Future Work 50

Bibliography **51**

II Included Papers 65

7 Paper A:
DeepMaker: A multi-objective optimization framework for deep neural networks in embedded systems 67

7.1 Introduction 69
7.2 Related Work 71
7.2.1 Automatic Design of Deep Neural Network Architecture 71
Hyperparameter Optimization 71
Reinforcement Learning 72
Evolutionary-based approaches 72
7.2.2 Neural Network Pruning 73
7.2.3 Automatic Code Approximation Frameworks 73
7.3 Preliminaries 74
7.3.1 Convolutional Neural Networks (CNNs) 74
7.3.2 Multi-Objective Optimization (MOO) 75
7.4 The proposed framework 76
7.4.1 Design Space Exploration 77
7.4.2 Neural Network Pruning 82
7.5 Experimental results 82
7.5.1 Training Datasets 83
7.5.2 Design Space Exploration 83
7.5.3 Neural Network Pruning 87
7.5.4 Hardware Implementation 88
7.6 Conclusions 94
Bibliography 95

8 Paper B:
TOT-Net: An Endeavour Toward Optimizing Ternary Neural Networks 101

8.1 Introduction 103
8.2 Background 105
8.2.1 Convolutional neural networks 105
8.2.2 Piece-wise activation functions 106
8.2.3 Ternary weight network 107
8.3 Related Work 108

8.3.1 Network Quantization 108
8.3.2 Neural Network Optimization 109
8.4 Architecture 110
8.4.1 Ternary Neural Networks 110
8.4.2 Ternary Neural Networks Optimization 115
8.5 Experimental Results 117
8.5.1 The Results of Classification Accuracy 117
8.5.2 Activation Function 119
8.5.3 Learning Rate 120
8.6 Conclusion and Future Work 122
Bibliography 123

9 Paper C:
TAS: <u>T</u>ernarized Neural <u>A</u>rchitecture <u>S</u>earch for Resource-Constrained Edge Devices 129
9.1 Introduction 2
9.2 Related Work 3
9.3 Ternarized Neural Architecture Search 4
9.3.1 Specialized Ternary NAS: Motivation 4
9.3.2 Search Space 5
9.3.3 Cell Template for TAS 6
9.3.4 New Ternary Quantization Method 7
Ternary Weights 7
9.3.5 Search Objective 9
9.4 Experiments 10
9.4.1 Experimental Setup 10
9.4.2 Results on CIFAR-10 11
9.4.3 Results on FPGA Implementation 12
9.5 Conclusion 12
Bibliography 13

10 Paper D:
FastStereoNet: A Fast Neural Architecture Search for Improving the Inference of Disparity Estimation on Resource-Limited Platforms 17
10.1 Introduction 2

10.2 Related Work 4
10.2.1 Disparity Estimation on Resource-limited devices . . . 4
10.2.2 Neural Architecture Search (NAS) 5
Macro NAS 5
Micro NAS 6
Improving NAS Efficiency 7
10.3 Architecture Design Space 7
10.3.1 Siamese Network Architecture 7
10.3.2 Representation of CNN Architectures 9
10.3.3 Efficient NAS by Transferred Weights Mechanism . . 11
10.4 Search Strategy 13
10.4.1 Proposed Method 13
10.4.2 FastStereoNet Practical Proof of Convergence 16
10.4.3 FastStereoNet Complexity Analysis 16
10.5 Experimental Setup 19
10.5.1 Evaluation Dataset 19
10.5.2 Hardware Specifications 19
10.5.3 Experimental Configuration 19
10.6 Results 21
10.6.1 Disparity Estimation Performance 22
10.6.2 Estimation Error Across Different Metrics 24
10.6.3 Smoothing Comparison 24
10.6.4 Search Convergence Analysis 24
10.6.5 Analyzing Search Methods 26
10.6.6 Reproducibility of the Results 27
10.6.7 FPGA Implementation Results 27
10.7 Conclusion 33
10.8 Disparity Estimation Pipeline A1
10.9 Simulated Annealing vs Genetic Algorithm A2
10.10 Traditional Complexity Analysis of FastStereoNet A2
10.11 Empirical Complexity Analysis of FastStereoNet A3
10.12 Smoothing FastStereoNet Outputs A3
10.13 Hardware Implementation Details A5
10.14 Latency Prediction A7
10.15 Reproducibility of Results A13
10.16 Analyzing Frequency Selection of the Dominant Operations . A15

10.17 Illustration of the FastStereoNet optimized architecture A17
Bibliography A17
Bibliography A23

11 Paper E:
NAS-PxAF: Neural Architecture Search for Accurate Detecting Paroxysmal Atrial Fibrillation 31
11.1 Introduction 2
11.2 Background: Paroxysmal Atrial Fibrillation 3
11.3 Related Works 5
11.3.1 NAS for ECG 5
11.3.2 PxAF Diagnosis Using DL Methods 6
11.4 Methodology 6
11.4.1 Method Overview 6
11.4.2 ECG Signal Processing 8
11.4.3 CNN Architecture Search 8
11.5 Experimental Setup 10
11.5.1 Database Preparation 10
11.5.2 Configuration Setup 11
11.5.3 Baseline for Comparison 11
11.6 Performance Measurement 13
11.7 Experimental Results 13
11.7.1 PxAF Classification Performance 13
11.7.2 ROC Analysis 13
11.8 Discussion 16
11.9 Conclusion 18
Bibliography 19

I

Thesis

Chapter 1

Introduction

1.1 Introduction

Learning is a task that humans can perform very well in most circumstances but is difficult for computers to accomplish. Machine learning is the field devoted to studying how computers can learn and improve their performance by gaining knowledge, making intelligent decisions, or recognizing complex patterns from a set of data.

Deep Neural Network (DNN) is a subset of machine learning algorithms containing more than one hidden layer and can be used to classify multi-dimensional input data. Recently, DNNs have made unprecedented progress in many learning tasks such as pattern recognition [1], image processing [2], image classification [3], speech processing [4], Natural language processing [5], and signal processing [6]. Advantages of DNNs against traditional machine learning techniques include requiring less domain knowledge for the problem they are trying to solve [3]. In addition, DNNs easily scale because accuracy improvement usually is achievable either by augmenting the training dataset or increasing the complexity of the network architecture. On the other hand, shallow learning models such as decision trees and Support Vector Machines (SVMs) are inefficient for many modern applications. Due to the vast usage of DNNs, they have been deployed in many platforms, from high-performance workstations to resource-constrained edge devices. Convolutional Neural Networks (CNNs) are one of the most important Deep Learning (DL) algorithms that provide highest accuracy for many computer vision (e.g., depth estimation [7], object detection [8], coronavirus detection from X-ray- or CT- images [9], etc) and biomedical signal processing (e.g., heart arrhythmia classification [10], lung sound classification [11]) tasks.

The life-cycle of CNNs comprises two major stages as described in the following: **Stage 1: Design & Train.** Designing and training a CNN with the primary goal of achieving maximum accuracy and inference efficiency. Despite the success of manually designing DL models, they do not always scale up with increasing model complexity. They also require massive trial-and-error resulting in enormous costs for both human and computational resources. Neural Architecture Search (NAS) emerged as a viable solution to automatically design efficient CNNs for a specific task requiring little human effort [12]. **Stage 2: Inference.** Inference is the deployment of a trained DL model to a target device using low-level compilation tools [13]. Despite significant advances of CNNs in both stages, CNNs still suffer from implementation difficulties:

- CNNs' impressive results have been obtained by increasing the CNN computational size leading to explosive training cost doubling every few months [14]. Handling such a large amount of computational requirements poses severe challenges for scalable design and efficient model deployment. Furthermore, it can produce excessive carbon dioxide, as NAS is estimated to produce up to 284019 Kg of CO_2 emissions [15]. This can result in harmful impacts on climate and severe consequences for ecosystems and biodiversity [16], as CO_2 is the leading cause of global warming.

- CNN applications suffer from the skyrocketing increase of the data volume caused by the billions of edge devices (up to 30.9 billion by 2025) connected across the globe [17], which are expected to create over 175 ZB of data by 2025 [18]. Alongside decreasing the innovation landscape of DL models, it reduces the efficiency of commodity computing devices.

- For more accurate results, CNNs are becoming more complex models containing hundreds of deep layers and millions of floating-point operations. Thus, having too many trainable parameters negatively impacts computational performance. Due to limited processing and power budget, the problem is more pronounced in deploying CNNs on resource-constrained edge devices.

Many prior works attempted to reduce the computational complexity and frequent memory accesses of CNNs (see Section 5). In general, the efficiency of the CNN implementation can be enhanced via the following techniques:

1. Many CNN hardware accelerators are proposed to overcome the computational cost and huge memory footprint of CNNs by parallel computing and efficient data reuse [19, 20, 21].

2. Model pruning is a practical method to minimize the size of the network and refine the network accuracy by removing redundant network connections and fine-tuning weights [22, 23, 24].

3. Low-rank matrix factorization is a technique that can reduce the computation and memory costs of DNNs by decomposing a large matrix into multiple smaller ones [25, 26, 27].

4. Model quantization is an impressive method trying to reduce the memory footprint and computation complexity of CNNs by representing the floating-point weights and/or activation functions as fixed-point and with fewer bits [28, 29, 30, 31].

5. Hardware-aware NAS methods jointly optimize multiple objectives enabled by considering either a proxy (e.g., model operations) or latency estimators to involve the awareness of hardware performance as a NAS objective [32, 33, 34].

This thesis aims to devise novel methods for designing CNN architectures that maximize efficiency and minimize design costs. Considering the limited time of the Ph.D. studies, this dissertation focuses on the fourth (Paper B and Paper C) and the fifth (Paper A, Paper D, and Paper E) approaches to amortize the design cost of CNNs and improve the model efficiency at inference time.

In Papers A, D, and E, the main focus is on proposing a cost-efficient NAS technique in order to improve the accuracy of image classification (Paper A), disparity estimation (Paper D), and arrhythmia classification (Paper E) tasks. Additionally, Papers A and D reduce the computational burden of CNNs by considering the complexity of the network as the second search objective. Paper C extends the idea of Paper B by integrating the ternarization mechanism into the NAS process to improve the accuracy of ternarized networks. The overview of the thesis contributions is shown in Figure 1.1.

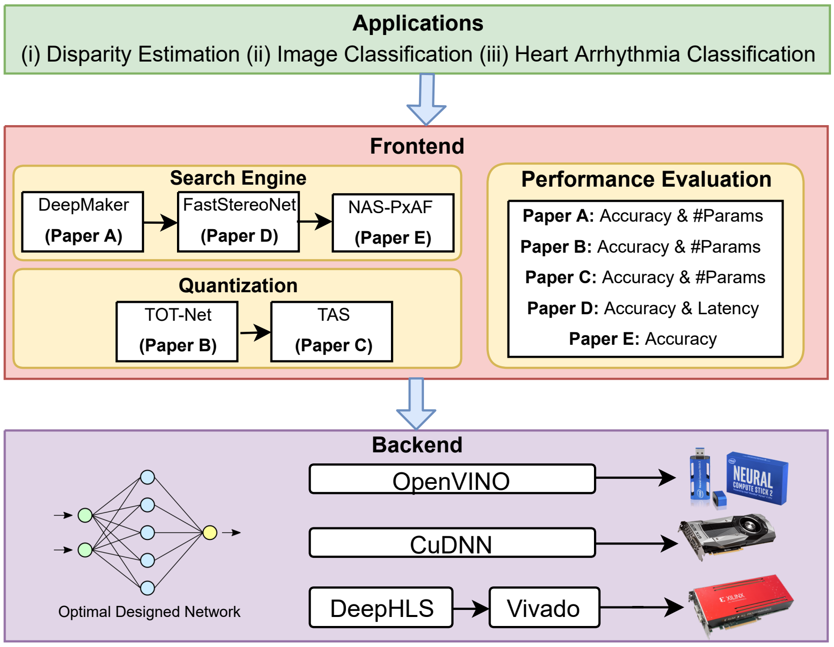


Figure 1.1: The overview of the contributions of the thesis.

1.2 Thesis Outline

This thesis is organized into two parts. The first part includes six chapters, which are presented in the following order: Chapter 1 gives an overview of the thesis. Chapter 2 presents background and preliminaries of this thesis. Chapter 3 presents research questions, research challenges, research goals, and the research methodology. In Chapter 4, we describe the contributions of the thesis to the realization of the research goals. Chapter 5 reviews significant research works in the domain of NAS and network ternarization. Finally, in Chapter 6, we conclude the first part of the thesis with a discussion on our key results as well as possible directions for future work. The second part of the thesis is given as a collection of the included publications that present the thesis's technical contributions in detail.

Chapter 2

Background

In this chapter, we first present Deep Learning (DL) and Convolutional Neural Networks (CNNs). Next, we present the preliminaries of neural architecture search (NAS).

2.1 Deep Learning

Learning is a task that humans can perform very well in most circumstances but is difficult for computers to accomplish. Machine learning is the field devoted to studying how computers can learn and/or improve their performance by gaining knowledge, making predictions, making intelligent decisions, or recognizing complex patterns from a set of data.

Deep Neural Networks (DNNs) are a subset of machine learning algorithms that are proposed to classify multi-dimensional input data. Recently, DNNs provide maximum performance in many learning tasks such as pattern recognition [1], image processing [2], image classification [3], speech processing [4], Natural language processing [5], and signal processing [6]. Advantages of DNNs against traditional machine learning techniques include providing the best results for large-scale datasets and requiring less domain knowledge for the problem they are trying to solve [3].

We briefly present the theory behind neural networks in the rest of this section. Afterward, we introduce convolutional neural networks as the target optimization tasks in this thesis.

2.1.1 Theory behind Neural Networks

Neural Network (NN), so-called Multi-layer Perceptron (MLP), is constructed by artificial neurons grouped in one or more layers. Figure 2.1 pictures the functionality of an artificial neuron. An artificial neuron consists of input values, weights, a bias, and an activation function. Each layer is either an input layer, hidden layer, or output layer, where the hidden layers extract features of input data to produce the final output (see Figure 2.2.a). In general, the different layers of an MLP have a different number of neurons. Regarding the functionality of artificial neurons, the input is multiplied by the weight, then added with the bias, which produces the activation function input. Together the summation and activation function represents the transfer function defining the neuron output. Hence the characteristics of the NN are determined by the transfer function [35].

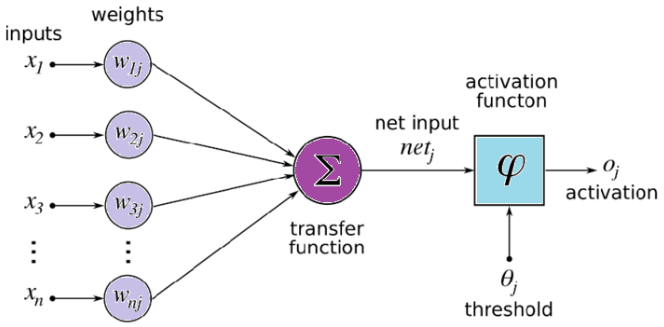


Figure 2.1: Structure of an artificial neuron, where x represents the neuron input, w weight, net_j the activation function input, θ_j the activation function threshold, and o_j is the output of neuron.

Transfer Function

As we mentioned before, the summation and the activation function compose the transfer function. According to [36], the activation functions are categorized in three classes: activation by inner product, distance, or a combination of both. Activation by inner product, also known as weighted activation, is a commonly used technique that establishes the base of sigmoidal transfer functions. The base of Gaussian transfer functions is activation by distance defined as the euclidean distance between the input vectors and a reference. The neural network layers or artificial neurons are not required to utilize the same activation functions [37]. With that said, the proper selection of network activation functions significantly influences the performance of DNNs. Some frequently used activation functions are discussed in the following [38]:

Sigmoidal activation function. It is a non-linear function that is repeatedly used in subsequent layers, affecting the output according to Equation 11.1. The function transforms the input into a value between zero and one, with the hard indications tendency. In other words, the output is more likely to be a high value or a low value rather than a middle value.

$$F(n) = \frac{1}{1+e^{-n}} \tag{2.1}$$

However, the main disadvantage of this approach is not responding to input values close to the function endpoints causing the vanishing gradients problem. The problem determines whether the neuron activates or not, leading to slow down the learning process [35, 39].

Tanh activation function. Tanh is a scaled version of the sigmoidal activation functions. Therefore it inherits sigmoidal properties such as non-linearity. Tanh transforms the input value between minus one to one by using Equation 9.1.

$$F(n) = \frac{2}{1+e^{-2n}} - 1 \tag{2.2}$$

Tanh suffers from the vanishing gradients problem for the same reason as sigmoid. The main distinction between the two is their sensitivity to input data. Tanh is more sensitive than sigmoid since it has sharper derivative [39].

Relu. [40] It is a popular non-linear activation function. The output of Relu produces i as output if i is positive and zero as output if i is negative (Equation 11.2). Opposed to Tanh, the output of Relu is not shielded by the function allowing the output to be in the range of zero to infinity.

$$F(n) = max(0, n) \tag{2.3}$$

Not activating neurons when the input value is negative has both benefits and drawbacks. Although less activated neurons are superior for increasing efficiency in deep networks, it gives birth to a negative phenomenon called dying Relu. This phenomenon results from a zero gradient, which happens when the input of the neuron is repeatedly a negative value, causing the neuron to stop learning. Leaky Relu [41] is a variation of Relu that attempts to avoid a zero gradient by multiplying the 0.01 value to the Relu negative inputs for minimizing sensitivity to the dying ReLU problem [39].

Neural Network Training

A dataset represents the prospective environment and required objects of interest to train the NN. The dataset is initially divided into two sets: train and test. The training data are divided into sets of training and validation. In general, it is optional how to divide the dataset into these three sets. Hagan et al. [35] propose to consider 70% for training, 15% for validation, and the remaining 15% for test. In some scenarios, when we deal with huge datasets, we can consider 90% for training, 5% for validation, and the remaining 5% for the test. NNs update initial weights, which are usually selected at random based on the error made from the training dataset. The backpropagation algorithm alters the network weights in every epoch to recognize the optimal weights. Epoch is one iteration of the entire training dataset. For most of the time, the dataset is too large to be processed by the hardware platforms at once. Thus one epoch is divided into batches, or mini-batches [42]. The training will be performed repeatedly until the model is deemed sufficient, or the learning is stopped [43, 44].

Performance Generalization

The performance of neural networks is defined as model's ability to generalize, which is evaluated by measuring how the model performs on unseen data. To provide a robust model, increasing the generalization performance is critical. However, the generalization performance is affected by the presence of errors, such as interpolation and extrapolation errors [35]. Interpolation errors, known as overfitting, occur when the prediction accuracy is high for the training dataset and arbitrarily exposed to a new dataset [45, 46, 35]. Extrapolation error, known as underfitting, happens due to a lack of variations in the training dataset. Underfitting causes low prediction accuracy [46, 35]. The concept of methods that can be directly applied to prevent the overfitting and underfitting problems is described below.

Dropout regularization. Dropout is a technique used in the training process for preventing the overfitting problem. Dropout removes neurons randomly during training to take samples from different narrowed-down architectures. Figure 2.2 shows the difference between a network leveraging dropout at training time and a network that does not. The amount of neurons to drop is determined by the retain probability p. It is recommended to select a high p value for input layers and convolutional layers, while others get a standard probability of 0.5 [47, 48].

Batch normalization. Normalization is applied for each mini-batch in order to address the internal covariance shift problem. This technique normalizes the input to decrease the required training time which results in producing a non-deterministic output.

Transfer learning. The main aim of transfer learning is to reduce the time required to find the optimal neural network weights. Transfer learning reuses knowledge from a pre-trained model on a new task. The method may decrease the number of required training epochs [49]. In addition, transfer learning can improve generalization performance since the effect of viewing the first dataset persists even after extensive training [50].

Pre-training. We train a network on a dataset before re-training and fine-tuning the same network on another dataset to decrease the training error. This technique improves the generalization performance for smaller datasets, while large datasets reap more generalization benefits [51].

2.1.2 Convolutional Neural Network

In recent years, several deep learning models have been proposed to improve the accuracy of different learning tasks. Convolutional Neural Network (CNN) is one of the most popular deep learning architectures that attained state-of-the-art results in many application domains, especially in computer vision tasks [3]. CNNs have been successfully deployed on many platforms, from high-performance workstations to mobile embedded devices.

In general, a CNN consists of multiple back-to-back layers connected in a feed-forward manner. The main layers include the convolution, normalization, pooling, and fully-connected layers. The convolutional layer extracts high-

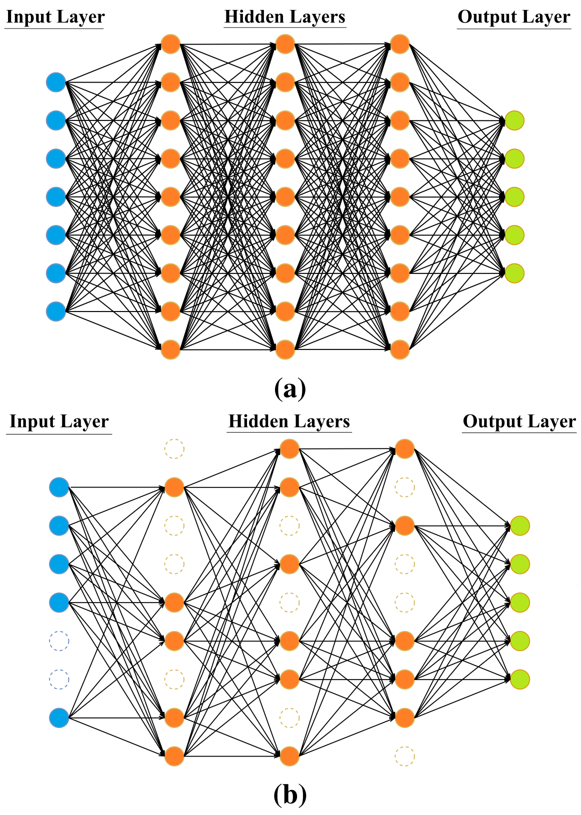


Figure 2.2: Illustrating a) an example of a NN architecture, and b) the same NN architecture during one mini-batch of the dropout regularization.

level abstraction of its inputs called a feature map by using various filters. Equation 8.1 shows the operation of a 3D convolutional layer that convolves the inputs via a filter $W \in R^{C \times X \times Y}$ for each feature map where C, X, and Y are the number of input channels and spatial dimensions of the filter, respectively. Many multiply and accumulate (MAC) operations are required to obtain one point of the output feature map.

$$conv3D = f_{act}(\sum_{k=0}^{C-1}\sum_{i=0}^{X-1}\sum_{j=0}^{Y-1} I\,[k]\,[X-i]\,[Y-j] \times W\,[k]\,[i]\,[j]) \tag{2.4}$$

Where $conv3D$, I, and W are the output feature maps, input feature maps, and $k \times k$ weight filters, respectively. Pooling layers perform down-sampling on data to decrease the amount of computation. Usually, in CNNs, pooling layers such as max pooling and average pooling are used after some convolutional layers. Max-pooling selects the maximum feature map, and Average-pooling computes the average feature maps in the pooling window. In general, after distinguishing high-level abstraction features, fully-connected layers are employed for classification. A significant portion of computations, over 90%, are performed in the convolutional layers where fully-connected layers are mainly memory-bound [17]. Fig. 2.3 shows an overview of convolutional neural networks.

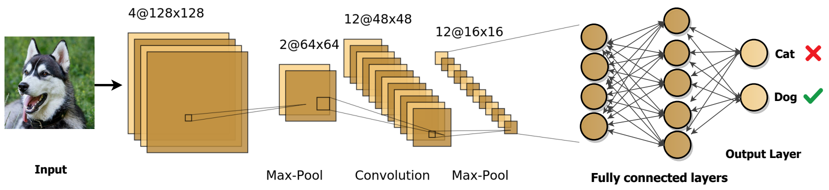


Figure 2.3: The general architecture of CNN.

2.2 Neural Architecture Search (NAS)

Neural Architecture Search (NAS) refers to the automatic optimization process of network architectures for a new task. Figure 2.4 shows the overview of the NAS algorithm. NAS starts with a set of predefined operations to form the search space. NAS uses a search method to explore among a large number of candidate architectures. All selected candidate architectures are trained and ranked. We perform the performance evaluation on the test set to evaluate the network architecture. Then, the search method is updated according to the ranking information of the previous candidates to obtain a set of new candidate architectures. After terminating the search process, the most promising network architecture is delivered to the user as the final optimal architecture. In the following, the functionality of each NAS stage is described.

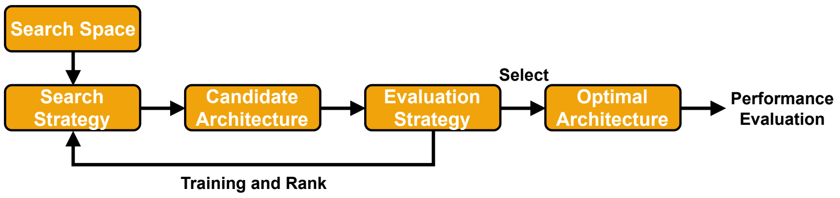


Figure 2.4: The overview of the NAS framework.

2.2.1 Search Space

We classify the search spaces into two essential categories: discrete and continuous. Discrete NAS search strategies are mainly categorized as *macro NAS* and *micro NAS* [52].

- *Macro NAS* strategies directly search the entire neural network architecture. In other words, NAS finds an optimal network architecture within a huge search space with the granularity of operations. Although the *macro NAS* strategies yield a flexible search space, larger search spaces enforce higher search costs. Figure 2.5 illustrates examples of two common *macro NAS* search spaces with a chain-based connection structure. Figure 2.5.a shows a simple example of a chain-based architecture. Figure 2.5.b shows a chain-based architecture with supporting skip connections to provide more diversity.

- *Micro NAS* strategies, so-called cell-based NAS, use pre-learned neural cells, where each cell is usually well-optimized for comparatively proxy tasks. Figure 2.6 shows an example of NASNet [53] as one of the first studies using this *micro NAS* idea. *Micro NAS* strategies try to find the optimal interconnection among neural cells by stacking many copies of the cells. Although *micro NAS* strategies drastically decrease the search time, they might not be optimal for unseen tasks [54, 55].

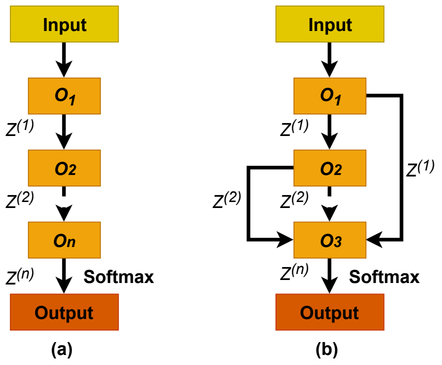


Figure 2.5: (a) A simple example of a chain-based architecture. O_i is an operation and the i^{th} operation in the architecture and $z^{(i)}$ is the o_i output feature map. (b) Extend the example by adding skip connections to provide more diversity. The input passes a series of operations to obtain the final output.

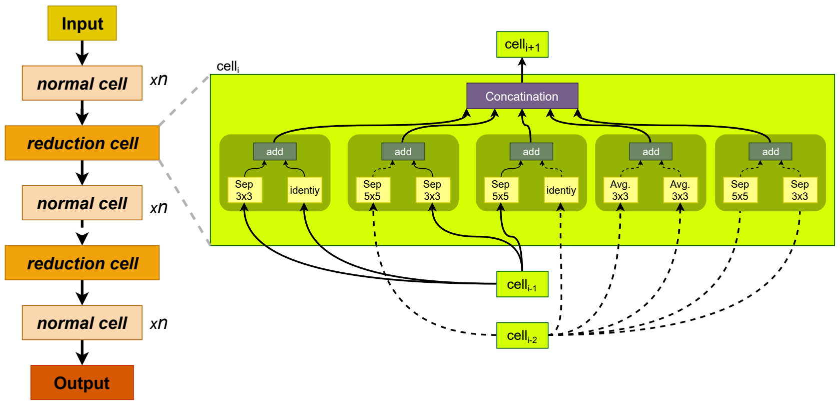


Figure 2.6: The structure of the search space leveraged in NASNet [53]. The search space is based on two cells, including normal and reduction cells. Normal cell extracts advanced features without changing the spatial resolution. The reduction cell reduces the spatial resolution. Multiple normal cells are connected to a reduction cell to design the final architecture, and this structure is repeated multiple times.

On the other hand, we have continuous search spaces which are mainly optimized by the stochastic gradient descent (SGD) algorithm [56, 57]. DARTS [58] as one of the earliest implementations of the continuous search space, tries to relax a discrete search space continuously. DARTS uses gradients to optimize the search space efficiently. DARTS utilize the NASNet cell-based search space [53]. DARTS learns a neural cell as the key building block of the final architecture. The learned cells are stacked to form either a convolutional network.

The search space is represented by a directed acyclic graph (DAG) constructed by N sequentially connected nodes. DARTS assumes each cell has two input nodes and one output node. To construct a convolutional cell, the input nodes are the output of the cells in the previous two layers. To construct a recurrent cell, one input is from the current time step, and the other input is feed-backed from the previous time step. The output of the cell is calculated by applying a concatenation operation to all intermediate nodes. For a discrete search space, each intermediate node can be expressed as $x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)})$ where $x_{(j)}$ is a potential feature representation in the cell, and $x_{(i)}$ is previous intermediate node $x_{(i)}$ through a directed edge operation $o_{(i,j)}$. Therefore, to learn the cell architecture, operations on the DAG edges should be learned. DARTS makes the discrete search space continuous by relaxing the selection of candidate operations to a softmax of all possible operations. Figure 2.7 presents continuous relaxation and discretization of search space in DARTS [58].

2.2.2 Search Method

Recently, different search methods have been proposed to explore the space of neural architectures. Random search, Bayesian optimization, evolutionary optimization, reinforcement learning (RL), and gradient-based algorithms are the most popular search methods in the community. In the following, these search methods are briefly described.

- ***Random Search.*** Random search randomly selects a specific number of candidate architectures (a sample size) from the architectural space. Random search evaluates the selected candidate architectures (e.g., by calculating accuracy). Then, it identifies the best architecture in the sample, stores it in memory, and repeats this process. If the new architecture

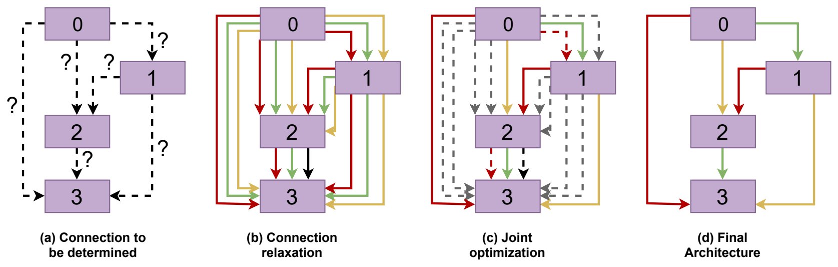


Figure 2.7: (a) The structure of a cell that aims to be learned. The operations on edges are unknown. (b) Illustrating the continuous relaxation of the cell-based search space. Each edge is a mixture of all candidate operations. (c) Joint optimizing the probability of mixed operations and network weights with gradient descent method. (d) Final network architecture.

is better than the previous one, the previous architecture will be replaced by the new architecture. The search will be stopped after a pre-defined number of iterations. Random search is proven to be a strong baseline for hyper-parameter optimization [59].

- ***Bayesian Optimization (BO).*** Bayesian Optimization is one of the most popular methods for hyper-parameter optimization [60, 61]. BO is based on the Bayesian paradigm. BO sets a prior over the optimization function and collects the information from the previous sample to update the posterior of the optimization function. A utility function selects the next sample point to maximize the optimization function [62].

- ***Reinforcement Learning (RL).*** RL methods are useful for modeling sequential Markov decision processes where an agent interacts with an environment to maximize its future benefit. To use RL for NAS problems, the design of a CNN architecture can be considered as the agent's action, with the action space identical to the search space. The agent's reward is the estimate of the performance of the trained architecture on test data.

- ***Evolutionary Methods.*** Evolutionary methods are an alternative to RL approaches by using evolutionary algorithms for optimizing the neural architecture. Evolutionary algorithms consist of the following key operators, including initialization, random parent selection, cross-over, mutation, and survivor selection. In general, evolutionary methods are highly sensitive to the choices for cross-over, mutation, and the fitness function that controls the behavior of the search process. Cross-over and mutation operators guide the diversity trade-off in the population. Similarly, the choice of fitness functions reflects the optimization objective.

- ***Gradient-Based Methods.*** While the methods above employ a discrete search space, Liu et al. [58] propose DARTS, a continuous relaxation to enable direct gradient-based optimization. DARTS optimizes both the network architecture and the network weights by alternating gradient descent steps on training data for weights and validation data for architectural parameters.

We need to note that it is essential to consider multiple search objectives in practice, even with conflict. For example, the number of floating-point operations and device-specific statistics such as latency and energy consumption are popular objectives considered in some studies [17, 63, 54, 64]. The neural search problem is formulated as a multi-objective optimization problem to consider the additional objectives. In general, multi-objective NAS separates decision-making into two steps: first, a set of candidates is obtained without considering any trade-offs between the different objectives, then the decision for a superior solution is made in the second step.

2.2.3 Evaluation Strategy

In general, there exist four techniques to reduce the evaluation cost of candidates during the search process:

1. **Lower Fidelity Estimation:** Reducing the training time is performed by ❶ training with fewer epochs, ❷ training on a subset of the dataset, ❸ down-scale models, and ❹ down-scale data. Although low-fidelity approximations remarkably reduce the computational cost, they also introduce bias in the estimation by performance underestimation. This may not be a problem if the search strategy only relies on ranking different architectures and the relative ranking remains stable, and the difference between the approximations and the full evaluation is not too big [65].

2. **Learning Curve Extrapolation:** Reducing the training time by performance extrapolation after just a few training epochs. Figure 2.8 shows an example of an early training termination to predict the final accuracy from the premature learning curve (solid line). This significantly reduces the number of required training iterations.

3. **Weight Inheritance/Network Morphisms:** Initializing the weights of new candidate architectures based on weights of other architectures that have been trained before, e.g., a parent model, is another approach to speed up performance estimation. This avoids training from scratch.

4. **One-Shot Models/Weight Sharing:** Treating all architectures as different sub-graphs of a super-graph (the one-shot model) and sharing the weights between architectures that have joint edges in the super-graph. This significantly improves the performance estimation of architectures since no training is required.

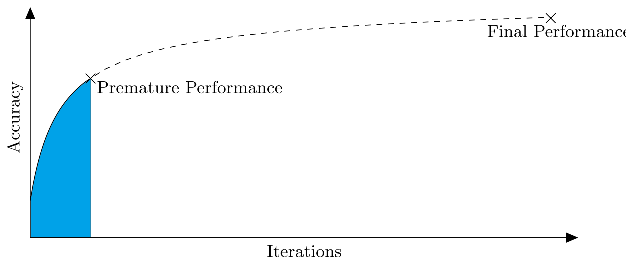


Figure 2.8: Example of early termination of the training strategy to accelerate the performance of evaluations.

Chapter 3

Research Statement

3.1 Problem Statement

Starting with AlexNet's win in the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC), CNNs have changed the landscape by providing superb capabilities in extracting high-dimensional structures from enormous data volumes. Meanwhile, mobile embedded devices such as smartwatches and medical tools have become ubiquitous. Therefore, there is a massive request for on-device Deep Learning (DL) services such as health monitoring, object recognition, and language translation [66, 67, 68].

Although CNNs significantly increase the accuracy for image classification, visual recognition, and many other tasks [10, 69, 70], state-of-the-art results are accompanied by increasing the size of CNNs. As shown in Figure 3.1, the amount of computing power needed to train state-of-the-art CNNs is growing at a rate of 15$\times$ every two years. Consequently, tweaking CNN architectures using NAS methods becomes costly due to requiring high-performance clusters to train NAS candidates (e.g., [71] requires 3800 GPU days).

On the other hand, the nature of embedded devices imposes intrinsic memory and computing bottlenecks that make the deployment of DL applications impossible in practice [7, 55, 73]. For example, An off-the-shelf ARM Cortex$^{\text{TM}}$-A7 embedded device only has 256KB on-chip memory. It is impossible to run CNNs such as ResNet-18 [74] on this device due to exceeding the peak memory by 339$\times$. Even the 2-bit quantized version of ResNet-18 still

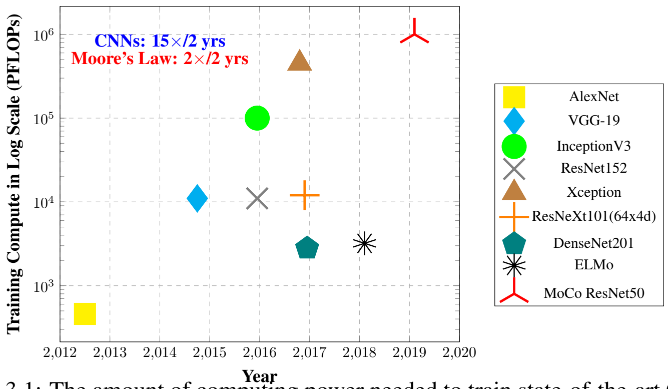


Figure 3.1: The amount of computing power needed to train state-of-the-art CNNs, measured in petaflops [72].

exceeds the memory limit by ≈22× [75], showing a significant gap between the desired and available resources of tiny edge devices. As a consequence, we have to deploy state-of-the-practice CNNs on the off-chip memory of AI accelerators causing more energy consumption [63, 19]. Therefore, there is a serious need to reduce the size and computational complexity of CNNs to make their training and deployment feasible for resource-constrained devices.

Network quantization is a widely used technique that significantly reduces the computational burden of CNNs. A ternary Neural Network (TNN) [76, 77, 78, 79, 29, 80], where both weights and/or activation functions are quantized to ternary tensors ({-1, 0, +1}) is a variation of network quantization techniques that comes with the benefits of model compression and operation acceleration. TNNs are reported to have up to 16× memory compression ratio [76] and 20× speed-up on FPGA [81], in comparison with full-precision networks. However, TNNs still suffer from a substantial accuracy drop issue, hampering their widespread use in practice (up to ≈17% accuracy drop compared to full-precision networks on ImageNet [82]).

In addition, Hardware-aware NAS methods are inefficient in practice due to leveraging lower fidelity estimation methods to predict hardware performance metrics such as FLOPs. Thus, developing cost-efficient search methods that use accurate hardware performance estimators is essential. To accommodate these requirements, we defined the following research questions;

1. What is the best CNN architecture with the highest accuracy that is implementable on a target resource-constrained (battery and memory) device?

2. How could we deal with the significant search cost of common NAS methods?

3. How to develop accurate hardware performance estimation models?

4. How could we reduce the accuracy degradation of common quantization methods?

3.2 Research Goals

To answer research questions, we research efficient NAS methods for designing low latency and resource-efficient CNN architectures for a wide range of devices such as Field Programmable Gate Arrays (FPGAs), embedded Graphics Processing Unit (GPU), and Intel® Neural Compute Stick 2 (NCS2). Plus, we accomplished a study on quantizing the weight and network activation functions to achieve a higher level of memory saving. The output of our studies is published in five papers (see Section 4). All in all, the overall goal of this thesis is formulated as follows:

Overall goal: Developing an automated framework that deploys CNNs into edge devices by (i) enabling scalable design of complex CNNs for a diverse set of edge devices with minimal cost; and (ii) quantizing CNNs with minimal accuracy degradation compared to floating-point counterparts. For more clarification, the overall goal is divided into the five following subgoals:

- **Subgoal 1:** Analyzing the characteristics of CNNs by focusing on computing potential in order to identify performance bottlenecks.

- **Subgoal 2:** Design of accurate latency estimation models for various edge devices to improve NAS search efficiency.

- **Subgoal 3:** Devising novel optimization techniques to search large-scale design spaces in a short time. We can leverage the latency estimator designed in the subgoal 2 to efficiently run more intelligent optimization techniques.

- **Subgoal 4:** Decreasing the computational cost and memory footprint of CNNs by proposing a novel NAS method that designs ternarized CNNs while providing higher level of accuracy.
- **Subgoal 5:** Evaluating how the proposed solutions improve the classification accuracy while decreasing the high computing cost and memory footprint of CNNs.

3.3 Research Challenges

According to NAS algorithms (Section 2.2), designing the CNN architecture involves three essential challenges. In the rest of this section, we address the main barriers to designing CNN architecture and our proposed solutions to tackle these challenges. The main NAS challenges are:

1. **Properly defining search space and involved hyper-parameters.** The search space is defined by the predefined architectural hyper-parameters and the corresponding operation set. For example, architectural template, kernel size, the number of channels of the convolutional layer, and the connectivity method of operations are among the most important search space parameters. The impact of the search space parameters on the final NAS performance is significant since these parameters determine which architectures can be searched by the NAS [56]. The selection of the search space is therefore critical since it has a substantial influence on the search cost and the quality of results.

2. **Properly selecting the search method.** The search method determines how to explore the search space, which is often large or even unbounded. It is desirable to quickly find well-performing neural architectures while avoiding being converged to a region of sub-optimal solutions. In other words, selecting the most suitable search method balancing the exploration-exploitation trade-off is critical.

3. **Properly selecting the evaluation strategy.** NAS methods try to find a neural architecture that maximizes some performance measurements, such as accuracy. Thus, there is a need to evaluate the performance of

candidate architectures. The simplest way is to train the candidate architecture on training data and evaluate its performance on validation data. However, training each architecture requires extensive computing capacity, which is the main bottleneck of NAS methods. For example, NASNet [53] used Reinforcement Learning (RL) to spend 2000 GPU days to design the best architecture for CIFAR-10 [83] and ImageNet [84]. Similarly, AmoebaNet [85] needs 3150 GPU days using an evolutionary algorithm. This naturally raises the need for some methods for accelerating performance evaluation. State-of-the-art studies propose performance estimation models to estimate accuracy and/or latency of CNNs. In this thesis, we only utilize latency estimation models (Paper D). It is worth mentioning that maximizing the accuracy of performance estimators for a given pair of hardware-dataset is a time-consuming task.

Here, the imminent question is - which NAS configuration is superior? In general, there is no clear answer to this question since it depends on the task, the size of the dataset, user constraints, search objectives, available computing power, etc. Table 3.1 summarizes NAS configurations used in our research papers.

Table 3.1: Summarizing the thesis contributions regarding the NAS structure.

	Search Space	Search method	Evaluation Strategy	Optimization Objective
Paper A	Discrete / macro NAS	NSGA-II [86]	Lower Fidelity Estimation	Accuracy and # Network Parameters
Paper B	Discrete / macro NAS	Genetic Algorithm	Lower Fidelity Estimation	Accuracy
Paper C	Continuous / micro NAS	SGD	Full-training	Accuracy
Paper D	Discrete / macro NAS	Meta-heuristic	Full-training & Latency Estimator	Accuracy and Latency
Paper E	Continuous / micro NAS	SGD	Full-training	Accuracy

3.4 Research Methodology

For doing scientific research and walking on the right path toward preparing a concrete thesis, leveraging research methodology is critical. The scientific method [87] provides how to facilitate new questions and formulate problems. Holz et al. [88] discuss the four major steps, including problem formulation, proposing a solution, implementation, and evaluation. Even though there was no solid research methodology at the beginning of the Ph.D. program, we followed Holz's research methodology in our research. We started with a literature review on similar methods aiming to tackle the problem; then, we continued with working on our idea to cover the weaknesses of the proposed solution. The implementation and evaluation phases were the last steps in our research journey. Figure 8.2 illustrates the research methodology used in our research.

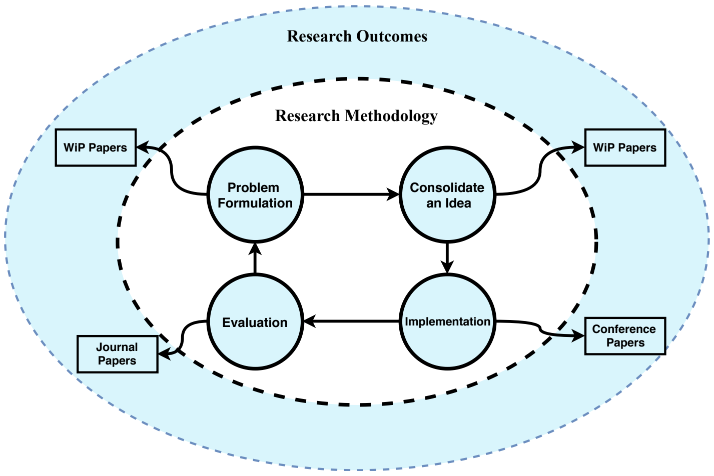


Figure 3.2: Research Methodology.

3.4.1 Problem definition

To initiate our research, we have examined state-of-the-art and state-of-the-practice studies. We first investigated prestigious computer architecture, computer vision, and machine learning venues such as ISCA, DATE, DAC, MICRO, FCCM, ECCV, CVPR, ICCV, ICLR, ICML, etc. After that, we discussed our findings with other researchers and our industrial partners. The research goals were formulated as an outcome of the problem formulation step. Plus, we found some ideas for the subgoals.

3.4.2 Consolidate an idea

After the literature review, we consolidated our ideas by focusing on the papers that showed remarkable results. Then, we summarized subgoals as a subgoal. Paper A and Paper D propose new methods to improve the current state-of-the-art by considering the second optimization objective. In Paper C, we extended the essential idea of Paper B. In addition, Paper D extends the idea of Paper A. Finally, Paper E uses a one-shot NAS method to classify an important pathological sign of ECG signal for Paroxysmal Atrial Fibrillation (PxAF).

3.4.3 Implementation

The implementation results on a wide range of resource-constrained devices are presented based on either hardware implementation (Papers A, C, D) or software implementation (Papers B, E). Measurements based on practical experience helped us understand our proposed solutions' actual impact.

3.4.4 Evaluation

In the evaluation step, comparative studies using the introduced metrics are considered. Depending on the results of the evaluation step, the problem formulation and proposed solution could be revised and continued with the later steps. This process is repeated until the results are acceptable. The results/outcomes of each step could be presented as papers, reports, and presentations in work-in-progress sessions, workshops, conferences, and journals.

Chapter 4

Research Contribution

In this section, we present our contributions (Papers A-E) to achieve the research goals (Section 3.2).

4.1 Contributions Addressing the Research Goals

4.1.1 Contribution of subgoal 1

We need to analyze the characteristics of CNNs to find bottlenecks in the way of accuracy and computing efficiency. As a result, we figured out CNNs are complex models with a significant memory footprint (Paper A and Paper B). Plus, the distribution of weights has a remarkable impact on the performance of TNNs (Paper C). Moreover, the backbone architecture of TNNs is not efficient for quantizing weights and/or activation functions (Paper C). Last but not least, the results represented in Paper D indicate the total number of network floating-point operations and neural network parameters have a low correlation with network inference time (latency) for large-scale datasets.

4.1.2 Contribution of subgoal 2

NAS should be hardware-aware for real-world applications to satisfy device-specific constraints (e.g., memory usage or latency). Existing hardware-aware NAS methods leverage network FLOPs as a proxy for hardware performance.

However, FLOPs is a highly inaccurate proxy since the latency of a CNN architecture could differ based on its degree of parallelism and memory access cost. Thus, neglecting hardware details will undoubtedly lead to inefficient search results. To address this challenge, we propose to (i) design a fully learning-based latency estimator for a target device; and (ii) integrate it in the search process to directly design a CNN architecture for a target device. Paper D covers subgoal 2.

4.1.3 Contribution of subgoal 3

Different optimization techniques have been proposed to design the architecture of CNNs, such as RL, random search, Bayesian optimization, and evolutionary methods. However, the inefficient exploration of the design space of architectural parameters is the main problem of prior works. Plus, most of the prior studies suffer from significant search costs. Based on the achievements of subgoal 1, subgoal 2, and literature review, we proposed fast search methods that find near-optimal solutions. In Papers C and E, we leverage a cell-based one-shot NAS method to design accurate neural architectures in a short time. In Paper D, we utilize a multi-stage meta-heuristic optimization method to provide a guided exploration scheme. Papers C, D, and E cover subgoal 3.

4.1.4 Contribution of subgoal 4

Based on the achievements of subgoal 1 and literature review, we proposed a novel NAS method that designs ternarized CNNs, dubbed TAS. TAS is a potential method that reduces the computing cost and memory footprint of CNNs while keeping accuracy at a safe margin. Recently, many proposed studies (see Section 5) addressed these issues. Although they have significantly decreased the computational load of CNNs, they have suffered from accuracy degradation, especially for large datasets. On the other hand, TAS is a gradient-based NAS method that efficiently reduces the accuracy gap between ternary and full-precision counterparts. TAS is the first to (i) simultaneously ternarize and design neural architectures; (ii) propose a new cell template for ternary networks with maximum gradient propagation; and (iii) provide a novel learnable quantizer that adaptively relaxes the ternarization mechanism from the data (kernel weights and activation functions) distribution. As the second contribution, we propose a novel piece-wise activation function and optimized learning rate for different datasets to improve the accuracy of ternarized neural networks. Paper B and Paper C cover subgoal 4.

4.1.5 Contribution of subgoal 5

Once we figured out the fundamental behavior of CNNs and proposed optimization solutions which are based on network architecture optimization and network ternarization, the evaluation between the state-of-the-art and the proposed solutions is conducted. In this subgoal, we consider several metrics to verify the functionalities of the proposed methods, including accuracy, network compression rate, computing performance (latency), search cost, reproducibility of results, etc. To evaluate the hardware performance, we consider a wide range of edge devices, including Xilinx Zynq FPGA, NVIDIA GPU, Intel® NCS 2, and ARM Processor. As a result, we confirm a notable decrease in the computational complexity in Papers A, B, C, and D. In addition, Paper E provides higher PxAF classification accuracy compared to widely-accepted baseline methods including ResNet-18 [74] and Auto-Sklearn [89].

4.2 Overview of the Included Papers

The main contributions of the thesis are organized and presented in the form of a collection of papers. Additional papers listed at the beginning of the thesis also strengthen the contributions of the thesis. A summary of the included papers is as follows:

4.2.1 Paper A

DeepMaker: A multi-objective optimization framework for deep neural networks in embedded systems [90].

Abstract. Deep Neural Networks (DNNs) are compute-intensive learning models with growing applicability in a wide range of domains. Due to their computational complexity, DNNs demand implementations that utilize custom hardware accelerators to meet performance and response time as well as classification accuracy constraints. In this paper, DeepMaker framework is

proposed, which aims to automatically design a highly robust DNN architecture for embedded devices as the closest processing unit to the sensors. DeepMaker explores and prunes the design space to find improved neural architectures. Our proposed framework takes advantage of a multi-objective evolutionary approach, which exploits a pruned design space inspired by a dense architecture. Unlike recent works that mainly have tried to generate highly accurate networks, DeepMaker also considers the network size factor as the second objective to build a highly optimized network fitting with limited computational resource budgets while delivers comparable accuracy level. In comparison with the best result on CIFAR-10 and CIFAR-100 dataset, a generated network by DeepMaker presents up to 26.4 compression rate while loses only 4% accuracy. In addition, DeepMaker maps the generated CNN on the commodity programmable devices including ARM Processor, High-Performance CPU, GPU, and FPGA.

Personal Contribution. I am the initiator, the main driver and the author of all parts in this paper. Mr. Ali Zoljodi helped us in preparing the results of network pruning algorithm. Ms. Sima Sinaei helped us with a nice review and reorganize the presentation structure of this paper. The other co-authors have contributed with valuable reviews.

4.2.2 Paper B

TOT-Net: An Endeavour Toward Optimizing Ternary Neural Networks [77].

Abstract. High computation demands and big memory resources are the major implementation challenges of Convolutional Neural Networks (CNNs) especially for low-power and resource-limited embedded devices. Many binarized neural networks are recently proposed to address these issues. Although they have significantly decreased computation and memory-footprint, they have suffered from accuracy loss especially for large datasets. In this paper, we propose TOT-Net, a ternarized neural network with [-1, 0, 1] values for both weights and activation functions that has simultaneously achieved a higher level of accuracy and less computational load. In fact, first, TOT-Net

introduces a simple bitwise logic for convolution computations to reduce the cost of multiply operations. To improve the accuracy, selecting proper activation function and learning rate are influential, but also difficult. As the second contribution, we propose a novel piece-wise activation function, and optimized learning rate for different datasets. Our findings first reveal that 0.01 is a preferable learning rate for the studied datasets. Third, by using an evolutionary optimization approach, we found novel piece-wise activation functions customized for TOT-Net. According to the experimental results, TOT-Net achieves 2.15%, 8.77%, and 5.7/5.52% better accuracy compared to XNOR-Net on CIFAR-10, CIFAR-100, and ImageNet top-5/top-1 datasets, respectively.

Personal Contribution. Ms. Najme Nazari is the initiator and the main driver in this paper. I have done the optimization part with the evolutionary method, obtaining the experiments, and I was responsible for writing the paper. Other co-authors have contributed with valuable discussion and reviews.

4.2.3 Paper C

TAS: Ternarized Neural Architecture Search for Resource-Constrained Edge Devices [75].

Abstract. Ternary Neural Networks (TNNs) compress network weights and activation functions into 2-bit representation resulting in remarkable network compression and energy efficiency. However, there remains a significant gap in accuracy between TNNs and full-precision counterparts. Recent advances in Neural Architectures Search (NAS) promise opportunities in automated optimization for various deep learning tasks. Unfortunately, this area is unexplored for optimizing TNNs. This paper proposes TAS, a framework that drastically reduces the accuracy gap between TNNs and their full-precision counterparts by integrating quantization into the network design. We experienced that directly applying NAS to the ternary domain provides accuracy degradation as the search settings are customized for full-precision networks. To address this problem, we propose (i) a new cell template for ternary networks with maximum gradient propagation; and (ii) a novel learnable quantizer that adaptively relaxes the ternarization mechanism from the distribution of the weights and activation functions. Experimental results reveal that TAS delivers 2.64% higher accuracy and $\approx 2.8\times$ memory saving over competing methods with the same bit-width resolution on the CIFAR-10 dataset. These results suggest that TAS is an effective method that paves the way for the efficient design of the next generation of quantized neural networks.

Personal Contribution. I am the initiator, the main driver, and the author of all parts of this paper. Mr. Hamid Mousavi did the implementation parts. Mr. Mohammad Riazati was responsible for implementing the best models on FPGA and reviewing the paper. The other co-authors have contributed with valuable reviews.

4.2.4 Paper D

FastStereoNet: A Fast Neural Architecture Search for Improving the Inference of Disparity Estimation on Resource-Limited Platforms [7].

Abstract. Convolutional Neural Networks (CNNs) provide the best accuracy for disparity estimation. However, CNNs are computationally expensive, making them unfavorable for resource-limited devices with real-time constraints. Recent advances in Neural Architectures Search (NAS) promise opportunities in automated optimization for disparity estimation [91, 54]. However, the main challenge of the NAS methods is the significant amount of computing time to explore a vast search space (e.g., 1.6×10^{29} [92]) and costly training candidates. To reduce the NAS computational demand, many proxy-based NAS methods have been proposed. Despite their success, most of them are designed for comparatively small-scale learning tasks. In this paper, we propose a fast NAS method, called FastStereoNet, to enable resource-aware NAS within an intractably large search space. FastStereoNet automatically searches for hardware-friendly CNN architectures based on Late Acceptance Hill Climbing (LAHC), followed by Simulated Annealing (SA). FastStereoNet also employs a fine-tuning with transferred weights mechanism to improve the convergence of the search process. Collection of these ideas provides competitive results in terms of search time and strikes a balance between accuracy and efficiency. Compared to the state-of-the-art [91],

FastStereoNet provides 5.25$\times$ reduction in search time and 44.4$\times$ reduction in model size. This benefits are attained while yielding a comparable accuracy that enables seamless deployment of disparity estimation on resource-limited devices. Finally, FastStereoNet significantly improves the perception quality of disparity estimation deployed on FPGA and Intel® NCS2 accelerator in a significantly less onerous manner.

Personal Contribution. I am the initiator, the main driver, and the author of all parts of this paper. I also perform the hardware implementation parts. Mr. Ali Zoljodi did the implementation parts. Dr. Amin Majd analyzed the search complexity, and the other co-authors have contributed with valuable reviews.

4.2.5 Paper E

NAS-PxAF: Neural Architecture Search for Accurate Detecting Paroxysmal Atrial Fibrillation. (*Under Review*)

Abstract. This paper presents a novel application of the Neural Architecture Search (NAS) method: classification of Paroxysmal Atrial Fibrillation (PxAF) from electrocardiogram (ECG) signal. PxAF is a pathological characteristic of ECG that can lead to fatal conditions such as heart attack and therefore requires special attention. The paper proposes an innovative method for ECG classification by integrating an especial signal processing phase with a convolutional neural network where NAS finds the optimal classification architecture. Experimental results show that the proposed NAS-based method not only enhances state-of-the-art, but also improves the classification performance of the two widely-accepted baseline methods, ResNet-18, and the Auto-Sklearn, by 3.4% and 10.6%, respectively.

Personal Contribution. I am the initiator, and the author of all parts of this paper. Mr. Mehdi Asadi did the implementation parts. Prof. Arash Gharehbaghi helped us with designing the signal processing pipeline and reviewing the paper. The other co-authors have contributed with valuable reviews.

4.2.6 Mapping Contributions to Subgoals

Mapping of the research subgoals to the contributed papers are shown in Table 4.1.

Table 4.1: Mapping of the research goals to the contributions.

	subgoal 1	subgoal 2	subgoal 3	subgoal 4	subgoal 5
Paper A	✓				✓
Paper B	✓			✓	✓
Paper C	✓		✓	✓	✓
Paper D	✓	✓	✓		✓
Paper E			✓		✓

Chapter 5

Related Work

In this section, we review the most significant related studies in the areas of *automatic design of CNN architectures* and *CNN Quantization techniques*, respectively.

5.1 Neural Architecture Search

To enable more accurate learning results, selecting the architectural parameters of CNNs is crucial since the network architecture strongly affects the inference time, memory footprint, accuracy, and network generalization proficiency. However, the manual design of CNNs is overwhelming due to requiring a lot of trial-and-error and deep expertise. Therefore, NAS methods have emerged as a potential alternative to decrease efficiency risk and design cost. Inspired by [52], we first categorized NAS methods as *macro NAS* and *micro NAS*. Then, we review recent papers that utilize proxy tasks to improve NAS efficiency since dealing with the time-consuming evaluation and huge search space are the main NAS difficulties [85, 53].

5.1.1 Macro NAS

Macro NAS methods try to *directly* design the entire neural network architecture from scratch [93, 94, 95, 55]. In other words, NAS finds an optimal architecture within a huge search space with the granularity of operations. It provides a highly flexible search space. However, the larger search space enforces more search costs.

Several search methods have been proposed to find the optimal solution effectively. Using the random search method is challenging due to extremely random sampling in the search space [12]. On the other hand, Bayesian-based methods suffer from immense computational cost, are suitable only for searching architectures with a fixed-length space, and focus on low-dimensional continuous problems [17]. Reinforcement learning (RL) is a popular approach for updating the network generator weights [95, 55, 92, 93]. [95] proposed an LSTM-based controller to configure the CNN descriptions, then trains this LSTM with RL to maximize the classification accuracy. [92] proposes ENAS, an efficient NAS trying to search an optimal sub-graph within a large computational graph by employing a meta-controller. In macro NAS, the size of search space exponentially growth by increasing the depth of network [93, 95], e.g., a network with less than 12 layers results in 1.6×10^{29} distinct architectures [92]. It is not widely feasible to efficiently search such a large space in a reasonable time (requiring thousand GPU hours). Thus, [92, 93, 95] prune the search space by limiting the depth of CNNs causing limited achievable accuracy [52]. Alternatively, a group of research studies relies on evolutionary search methods where the best architecture is designed by iteratively refining a population of candidate architectures [96, 90, 85, 97, 98]. However, all these methods are costly, requiring hundreds of GPU days. In contrast, we propose a fast search method that designs efficient CNNs from scratch.

5.1.2 Micro NAS

Micro NAS methods search the inner architecture of learning cells, while the interconnection among neural cells is defined by stacking several copies of the discovered cells [85, 92, 58, 53, 91, 52]. Although *micro NAS* methods highly decrease the search time, they might not be optimal for any unseen tasks since each cell is usually well-optimized for comparatively proxy tasks. For example, most of them are optimized for the CIFAR-10 dataset [53, 92], which do not guarantee to be optimal for large-scale datasets [91, 55].

5.1.3 Improving NAS Efficiency

In general, NAS is a time-consuming process. This results from: (i) candidates' notorious training time (e.g., [71] needs 3800 GPU days); and (ii) huge search space. To improve NAS speed and to reduce the NAS computational cost, a variety of techniques have been proposed to utilize proxy tasks [12]. HyperNet generates weights for candidate networks and evaluates them without full training from scratch [94]. [63, 99] use partial training for accuracy prediction at the cost of noisy evaluations. Plus, using a proxy for accuracy introduces estimation bias since accuracy will typically be underestimated [65]. We did not use accuracy predictors in this thesis. Sharing weights among potential networks decreases the search time by two orders of magnitude [92]. [93, 100, 101] use network to network transformation for reusing weights of previously discovered networks to amortize the training cost.

5.2 Network Ternarization

Network quantization is an effective solution that enables deploying CNNs onto edge devices. Some recent researches [102, 103, 104] surpass quantized neural networks and have aggressively reduced precision even to 1-bit to construct binarized neural networks (BNNs). However, BNNs suffer from accuracy loss, especially for large datasets. [105] tried to address this issue for BNNs by proposing an efficient training strategy. BinaryConnect [104] eliminates multiplication in the forward pass by substituting full-precision weights with -1 and 1 values. Some works such as [106] and [107] applied reduced precision to reduce memory storage and computation. However, they still require computation-intensive MAC operations to perform the convolutional operations, hence, suffer from heavy operations.

Ternary neural networks (TNNs) provide a fair trade-off between accuracy and complexity [76, 82]. TWN [76], known as one of the earliest ternarization efforts, proposed $\{$-1, 0, +1$\}$ as quantization values to improve the accuracy of binary networks. TWN utilized two symmetric thresholds (Δ) alongside a scaling factor (α) for each layer to quantize into $\{$-Δ, +$\Delta\}$. However, there remains a significant performance gap between TWN and the full-precision counterparts for large-scale datasets (7.5% accuracy loss on ImageNet for ResNet-18) due to using symmetric thresholds. To solve this problem, TTQ uses two full-precision scaling factors (α_n, α_p) for positive and negative values [82]. LQ-Nets [29] applies proper quantization bits by a learnable quantizer, and ReLeQ [30] automates CNN quantization based on a Reinforcement Learning (RL) algorithm, respectively. TRQ [79] claimed that the existing thresholding algorithms are not accurate enough to map the full precision to ternary values. Therefore, TRQ introduced a recursive ternary quantization on full-precision weights for a refined reconstruction rather than directly thresholding. In addition to designing the ternarization mechanisms, several hardware compilers have been proposed [78, 108, 81] to expedite the ternarized network on energy-efficient edge devices. Although prior studies have improved the ternarization mechanism, they all have a naïve assumption on the distribution of the weights and/or activation functions.

Chapter 6

Discussion, Conclusion and Future Work

In this chapter, we first discuss a summary of experimental results and conclude the thesis. Finally, we present a list of potential future research directions.

6.1 Discussion and Conclusion

According to the literature review, we identified room for developing Neural Architecture Search (NAS) and network quantization to tackle the challenges of deploying large-scale CNNs on embedded mobile devices. NAS is a powerful tool to accomplish the intended aims in this area. Nonetheless, there is no well-detailed solution that gives the maximum accuracy for any unseen task. In other words, many NAS specifications such as fitness function, design space operations, and termination condition depend on the task under study and the user's constraints. In addition, NAS is a time-consuming process that requires a high-performance system because of the enormous cost of training candidates. Finally, network FLOPs or network parameters cannot be reliable search objectives due to low correlation with network latency or power consumption. The second technique utilized in this thesis is network quantization, a popular method for reducing the computational cost and memory footprint of CNNs. Despite providing remarkable computing cost alleviation, quantization techniques suffer from significant accuracy loss.

6.1.1 Thesis Storyline.

In this Section, we present the storyline of included publications.

- DeepMaker [90] introduces a multi-objective Neural Architecture Search (NAS) algorithm by leveraging the NSGA-II algorithm [86] to design hardware-friendly architectures (Paper A).
- In Paper D, we introduce a multi-objective Neural Architecture Search (NAS) algorithm that combines Late Acceptance Hill Climbing followed by Simulated Annealing [7]. We devise a latency estimator to accurately estimate the inference time of the candidate neural architectures on a range of target devices. We also integrate a transferred weights mechanism to reuse the weights of ancestors to expedite the overall search speed.
- In Paper E, we introduce a novel combination of time-frequency analysis in conjunction with the NAS method for classifying an important pathological sign of ECG signal: Paroxysmal Atrial Fibrillation (PxAF). The method used the wavelet transform along with the recurrence images of the transformed signal to constitute input images to a CNN. The architecture of CNN was optimized by using a gradient-based NAS method.
- Paper B proposes the TOT-Net framework [77], a solution for ternarizing CNNs with [-1, 0, 1] values for both weights and activation functions. TOT-Net introduces a simple bit-wise logic for convolutional layers to reduce the cost of multiply operations. TOT-Net proposes a novel piecewise activation function and optimized learning rate for different datasets to improve the classification accuracy.
- Paper C proposes TAS, a gradient-based NAS method that efficiently reduces the accuracy gap between ternary and full-precision architectures while providing a significant compression ratio. TAS is robust to quantization error by adding inter-cell skip connections to the DARTS cell template to convey gradient propagation effectively. To our knowledge, TAS [75] is the first method that designs accurate ternary neural networks.

Regarding the involved issues in deploying CNNs on embedded platforms using NAS and quantization, a summary of the results is presented in the following.

6.1.2 Disparity Estimating Performance

Table 6.1 compares the proposed method in Paper D with the other cutting-edge architectures regarding the D1-all accuracy, Intel® NCS2 inference time, FLOPs, and the search cost evaluation metrics. We consider end-to-end latency (data transfer time + computation time) as the evaluation inference metric. Batch size is equal to 1 in all the experiments. We believe taking inference time, represented in second(s), is not reliable as the only metric for comparing the implementation efficiency of two different networks. The reasons come from the fact that the inference time even on the same device depends on various factors, such as the learning API (Torch [109], TensorFlow [110], etc.), compiler settings, and hardware acceleration libraries (NVIDIA® cuDNN, Intel® OpenVINO™, etc.); and Therefore, we also report *network compression rate*, $\frac{Accuracy}{FLOPs}$, and $NID = \frac{Accuracy}{NetworkParameters}$ in Table 6.1 as three hardware-independent alternative metrics.

Table 6.1: Comparing the FastStereoNet results (Paper D) with state-of-the-art methods on KITTI 2015. Unsuccessful implementation are shown in the red cells (mainly due to the limited on-chip memory or OpenVINO™ limited supporting operations).

Architecture	**FLOPs** ($\times10^6$)	**Search Method**	**Network Compression Rate**($\times$)†	**NID** ($\times10^6$)	$\frac{\textbf{Accuracy}}{\textbf{FLOPs}}$ ($\times10^6$)	**Search Cost (GPU Days)**⋈	**Error (%) Without Quantization**	**Intel® NCS2 Inference Time (Sec.)**∗
DenseDisp [54]	1.56	Meta-heuristic	102	89.3	58.98	2	7.99 (D1-all)	0.626 (0.017≠)
AutoDispNet-BOHB-CSS-ft† [91]	160	RL	1	0.88	-	42	OUT_OF_MEMORY‡	
DenseMapNet [111]	-	Hand-Crafted	-	-	-	-	2.52 (EPE)	0.45
Vid2Depth [112]	-	Hand-Crafted	-	-	-	-	0.163 (Abs. Rel.)	0.276
GC-Net [113]	-	Hand-Crafted	-	27.75	-	-	OUT_OF_MEMORY‡	
Content-CNN [114]	2	Hand-Crafted	80	13.6	47.73	-	4.54 (D1-all)	0.7
GA-Net-deep [115]	-	Hand-Crafted	-	-	-	-	OUT_OF_MEMORY‡	
PSM-Net [116]	30	Hand-Crafted	5.3	6.1	-	-	OUT_OF_MEMORY‡	
DispNet-CSS-ft∓ [117]	195	Hand-Crafted	0.8	0.84	-	-	OUT_OF_MEMORY‡	
FastStereoNet (FLOPs)	2.08	Meta-heuristic	76.9	68.24	44.94	8	6.51 (D1-all)	0.64 (0.028≠)
FastStereoNet (NCS2)	3.6	Meta-heuristic	44.4	40.07	26.6	8	4.22 (D1-all)	0.64 (0.028≠)

† The baseline for comparing the compressing rate over the FLOPs metric.
∓ Reported in [91].
∗ The results are compiled with Intel® OpenVINO™.
‡ Undesired state which happens whenever the Intel® NCS2 on-chip memory cannot be allocated due to the huge network size.
⋈ All the methods used 1× NVIDIA® GTX 1080ti for evaluating the candidates.
≠ Average computation time (kernel time) for 10000 times re-running network inference.

FastStereoNet obtains 95.78% accuracy with 640ms total inference time on the Intel® NCS2 accelerator, 2.39M parameters, and 3.6M FLOPs. In comparison with the *AutoDispNet-BOHB-CSS-ft* NAS method, FastStereoNet presents 44.4× more network compression rate while delivering a comparable accuracy (less than 2.05% accuracy loss). Also, FastStereoNet can be successfully implemented on the Intel® NCS2 accelerator, while *AutoDispNet-BOHB-CSS-ft* fails to be implemented in the Intel® NCS2 accelerator due to huge memory footprint. In terms of search time, although FastStereoNet directly searches a huge space with the minimum size of 2×24^{10} candidates, FastStereoNet is still 5.25× faster than *AutoDispNet* which is a cell-based search that usually takes a shorter time to find a solution. Compared to DenseDisp [54], FastStereoNet provides 3.77% higher accuracy as it uses a more complex design space. DenseDisp yields 4× faster search compared to FastStereoNet. However, this comes from the fact that DenseDisp trains the candidate architectures for few epochs to estimate the accuracy while FastStereoNet fully trains the network for all candidates.

Reporting the inference time on the Intel® NCS2 accelerator is motivated by the following three observations: 1) some real-time neural architectures such as DispNet cannot be deployed on some of the resource-limited devices due to their high memory footprint; 2) the cutting-edge learning models use complex operations which are not usually supported by commodity embedded devices, e.g., Intel® NCS2 does not support `GatherNd` operation used by MADNet [118]; and 3) Most of the resource-limited devices only support quantized operations such as 8-bit floating-point (FP8) or 16-bit floating-point (FP16). However, despite our expectation, the 16-bit quantization decreases the accuracy significantly compared to the full precision implementation (float) as illustrated in Fig. 10.6.c. As such, the models tailored to high-performance GPUs may not be helpful for any resource-constrained devices. To the best of our knowledge, FastStereoNet is the only solution that tackles all the challenges mentioned above by providing a clear perception (Fig. 10.6.h) on the Intel® NCS2 accelerator. FastStereoNet achieves the clear perception with low latency by: (i) considering network inference time as the second search objective yields highly customized architectures for a given target device; and (ii) utilizing an independent disparity refinement which is not quantized during hardware implementation; thus, it refines quantization drawbacks.

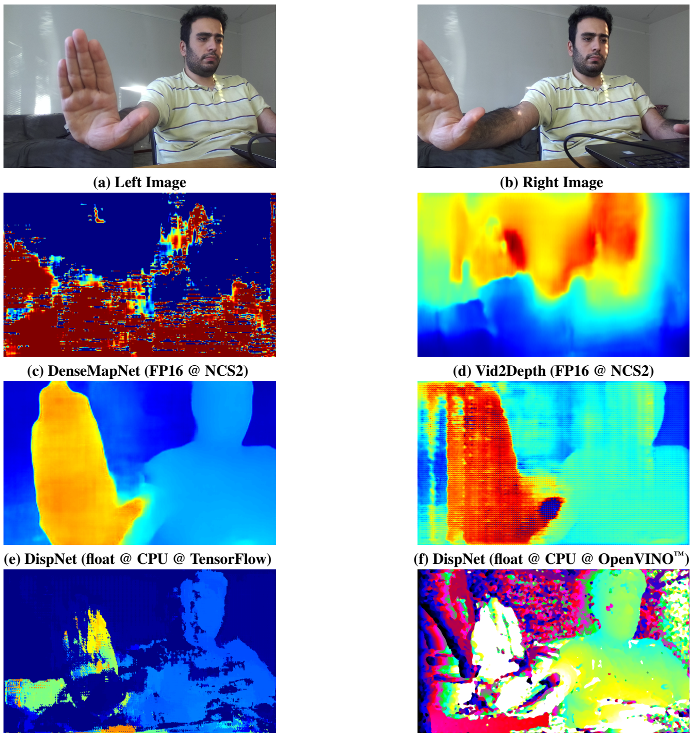
6.1.3 Ternary Neural Networks (TNNs) Performance

Table 6.2 compares the results of TAS proposed by Paper C against state-of-the-art approaches on the CIFAR-10 dataset. Note that the compression ratio is determined by measuring memory utilization. Consider q is quantization resolution (q-bit) of layer l, L is the maximum number of layers in each network, $\#W_l$ and $\#W_l^t$ are the number of weights in layer l for full-precision (32-bit) and ternary networks, respectively. Hence, the compression ratio is expressed as $\frac{\sum_{l=1}^{L} \#W_l \times 32}{\sum_{l=1}^{L} \#W_l^t \times q}$. ResNet-18 [74] is selected as the compression ratio baseline in our experiments.

TAS significantly outperforms all existing methods in terms of accuracy and compression ratio. TAS obtains a 0.94% accuracy improvement with a 45.73× higher compression ratio than full-precision ResNet-18. In comparison with TRQ [79] with the same quantization resolution, TAS achieves 2.64% accuracy improvement with 2.84× higher compression ratio. We can see that trivially extending DARTS to design TNNs (*DARTS+Ternarization*) results in a 6.04% accuracy degradation. Compared to binary NAS [119], TAS provides 1.28% higher accuracy without compromising memory saving.

6.1.4 Paroxysmal Atrial Fibrillation Classification Performance

Paper E presents a novel application of the Neural Architecture Search (NAS) method: classification of Paroxysmal Atrial Fibrillation (PxAF) from electrocardiogram (ECG) signal. We propose an innovative method for ECG classification by integrating an especial signal processing phase with convolutional neural network where NAS finds the optimal classification architecture. Table 11.2 compares the results of the proposed method in Paper E with the state-of-the-art and state-of-practice classification methods. Results show that our proposed method provides the most accurate classification result compared to all counterparts.


(a) Left Image

(b) Right Image

(c) DenseMapNet (FP16 @ NCS2)

(d) Vid2Depth (FP16 @ NCS2)

(e) DispNet (float @ CPU @ TensorFlow)

(f) DispNet (float @ CPU @ OpenVINO™)

(g) GC-Net (float @ CPU)

(h) FastStereoNet (FP16 @ NCS2)

Figure 6.1: Illustrating the output of different studied disparity estimators: (a) left image, (b) right image,(c) DenseMapNet (FP16 @ NCS2), (d) Vid2Depth (FP16 @ NCS2), (e) DispNet (float @ CPU @ TensorFlow), (f) DispNet (float @ CPU @ OpenVINO™), (g) GC-Net (float @ CPU), and (h) FastStereoNet (FP16 @ NCS2). FastStereoNet is the only solution that yields clear disparity perception on Intel® NCS2.

Table 6.2: Comparing the TAS results with state-of-the-art methods on CIFAR-10 dataset.

Method (Backbone Arch.)	**# bits (W/A)**‡	**Compression Ratio** $(\times)^{\dagger}$	**Top-1 Accuracy (%)**
Full-precision (ResNet-18) [74]	32/32	1	91.0
TBN (VGG-7) [80]	2/32	1.33	90.85
TWN (ResNet-18) [76]	2/32	16.06	92.56
TRQ (ResNet-18) [79]	2/2	16.06	89.3
Binary NAS (A) [119]	1/1	45.73	90.66$*$
DARTS+Ternarization [58]	2/32	-	85.9
TAS (Ours)	2/2	**45.73**	**91.94**

† The baseline for comparing the compressing ratio is ResNet-18.
‡ (Weights/Activation Function).
$*$ Experiments obtained by re-running the official implementation.

Table 6.3: Comparing the results of Paper E with state-of-the-art and state-of-the-practice methods.

Method	**Accuracy (%)**	**Search Method**
Random Search [74]	95.1	Random Search†
Pourbabaee et al. [120]‡	91.0	Manual
ResNet-18 [74]	94.3	Manual
Auto_Sklearn [89]	87.15	Bayesian Optimization
Paper E (Ours)	**97.7**	SGD

† Using the same search space as DARTS [58].
‡ Reporting the best results by CNN architecture with a K-nearest neighbor (KNN) classifier.

6.2 Future Work

In the rest of the research journey, we mainly aim to leverage the idea of meta-learning to reduce the cost of designing hardware performance estimation models. In addition, considering other hardware constraints such as energy and silicon footprints (on FPGAs) is one of our interests. We also identify room for some research directions on optimizing the architecture of time-based classifiers such as transformers. Therefore, we can extend our proposed method to support designing the optimal architecture for transformers. Language translation, speech recognition, and market analysis are among the applications that can benefit from optimized transformers.

Bibliography

[1] Nasser M Nasrabadi. Pattern recognition and machine learning. *Journal of electronic imaging*, 16(4):049901, 2007.

[2] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[4] Dong Yu and Li Deng. *AUTOMATIC SPEECH RECOGNITION*. Springer, 2016.

[5] Li Deng and Yang Liu. *Deep Learning in Natural Language Processing*. Springer, 2018.

[6] Li Deng and Dong Yu. Deep learning for signal and information processing. *Microsoft Research Monograph*, 2013.

[7] Mohammad Loni, Ali Zoljodi, Amin Majd, Byung Hoon Ahn, Masoud Daneshtalab, Mikael Sjödin, and Hadi Esmaeilzadeh. Faststereonet: A fast neural architecture search for improving the inference of disparity estimation on resource-limited platforms. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2021.

[8] Najda Vidimlic, Alexandra Levin, Mohammad Loni, and Masoud Daneshtalab. Image synthesisation and data augmentation for safe object detection in aircraft auto-landing system. In *16th International Joint*

Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, VISIGRAPP 2021, 8 February 2021 through 10 February 2021, volume 5, pages 123–135. SciTePress, 2021.

[9] JT Thirukrishna, Sanda Reddy Sai Krishna, Policherla Shashank, S Srikanth, and V Raghu. Survey on diagnosing corona virus from radiography chest x-ray images using convolutional neural networks. *Wireless Personal Communications*, pages 1–10, 2022.

[10] Zahra Ebrahimi, Mohammad Loni, Masoud Daneshtalab, and Arash Gharehbaghi. A review on deep learning methods for ecg arrhythmia classification. *Expert Systems with Applications: X*, page 100033, 2020.

[11] Truc Nguyen and Franz Pernkopf. Lung sound classification using snapshot ensemble of convolutional neural networks. In *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 760–763. IEEE, 2020.

[12] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[13] Mingzhen Li, Yi Liu, Xiaoyan Liu, Qingxiao Sun, Xin You, Hailong Yang, Zhongzhi Luan, Lin Gan, Guangwen Yang, and Depei Qian. The deep learning compiler: A comprehensive survey. *IEEE Transactions on Parallel and Distributed Systems*, 32(3):708–727, 2020.

[14] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green ai. *Communications of the ACM*, 63(12):54–63, 2020.

[15] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[16] Philip Smith and Nicolas Howe. *Climate change as social drama: Global warming in the public sphere*. Cambridge University Press, 2015.

[17] Mohammad Loni, Masoud Daneshtalab, and Mikael Sjödin. Adonn: Adaptive design of optimized deep neural networks for embedded systems. In *2018 21st Euromicro Conference on Digital System Design (DSD)*, pages 397–404. IEEE, 2018.

[18] David Reinsel-John Gantz-John Rydning. The digitization of the world from edge to core. *Framingham: International Data Corporation*, page 16, 2018.

[19] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. From high-level deep neural models to fpgas. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, page 17. IEEE Press, 2016.

[20] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 367–379. IEEE Press, 2016.

[21] Francesco Conti, Pasquale Davide Schiavone, and Luca Benini. Xnor neural engine: A hardware accelerator ip for 21.6-fj/op binary neural network inference. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2940–2951, 2018.

[22] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[23] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[24] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5687–5695, 2017.

[25] Gintare Karolina Dziugaite and Daniel M Roy. Neural network matrix factorization. *arXiv preprint arXiv:1511.06443*, 2015.

[26] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[27] Samuel Cahyawijaya, Genta Indra Winata, Holy Lovenia, Bryan Wilie, Wenliang Dai, Etsuko Ishii, and Pascale Fung. Greenformer: Factorization toolkit for efficient deep neural networks. *arXiv preprint arXiv:2109.06762*, 2021.

[28] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pages 764–775. IEEE Press, 2018.

[29] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 365–382, 2018.

[30] Amir Yazdanbakhsh, Ahmed T Elthakeb, Prannoy Pilligundla, and FatemehSadat Mireshghallah Hadi Esmaeilzadeh. Releq: An automatic reinforcement learning approach for deep quantization of neural networks. *arXiv preprint arXiv:1811.01704*, 2018.

[31] Lei Deng, Peng Jiao, Jing Pei, Zhenzhi Wu, and Guoqi Li. Gxnor-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework. *Neural Networks*, 100:49–58, 2018.

[32] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[33] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[34] 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. *arXiv preprint arXiv:1704.04861*, 2017.

[35] Howard B Demuth, Mark H Beale, Orlando De Jess, and Martin T Hagan. *Neural network design*. Martin Hagan, 2014.

[36] Wlodzislaw Duch and Norbert Jankowski. Survey of neural transfer functions. *Neural Computing Surveys*, 2(1):163–212, 1999.

[37] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[38] Mina Basirat and Peter M Roth. Learning task-specific activation functions using genetic programming. In *VISIGRAPP (5: VISAPP)*, pages 533–540, 2019.

[39] Avinash Sharma V. Understanding activation functions in neural networks, 2017. accessed: 20 May 2020.

[40] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[41] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[42] Sagar Sharma. Epoch vs batch size vs iterations, 2017. accessed: 20 May 2020.

[43] J.Brownlee. A gentle introduction to the challenge of training deep learning neural network models, 2019. accessed: 20 May 2020.

[44] J.Torres. Learning process of a neural network, 2018. accessed: 20 May 2020.

[45] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[46] Haotian Zhang, Lin Zhang, and Yuan Jiang. Overfitting and underfitting analysis for deep learning based end-to-end communication systems. In *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6. IEEE, 2019.

[47] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[48] Sungheon Park and Nojun Kwak. Analysis on the dropout effect in convolutional neural networks. In *Asian conference on computer vision*, pages 189–204. Springer, 2016.

[49] Lorien Y Pratt. Discriminability-based transfer between neural networks. In *Advances in neural information processing systems*, pages 204–211, 1993.

[50] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.

[51] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[52] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1761–1770, 2019.

[53] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

[54] Mohammad Loni, Ali Zoljodi, Daniel Maier, Amin Majd, Masoud Daneshtalab, Mikael Sjödin, Ben Juurlink, and Reza Akbari. Densedisp: Resource-aware disparity map estimation by compressing siamese neural architecture. In *IEEE World Congress On Computational Intelligence (WCCI) 2020*, July 2020.

[55] 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 2019*, 2019.

[56] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A survey on neural architecture search. *arXiv preprint arXiv:1905.01392*, 2019.

[57] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in neural information processing systems*, pages 7816–7827, 2018.

[58] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[59] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019.

[60] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

[61] Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, et al. Hyperparameter optimization: Foundations, algorithms, best practices and open challenges. *arXiv preprint arXiv:2107.05847*, 2021.

[62] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. Hyperparameter optimization for machine learning models based on bayesian optimization. *Journal of Electronic Science and Technology*, 17(1):26–40, 2019.

[63] Mohammad Loni, Ali Zoljodi, Sima Sinaei, Masoud Daneshtalab, and Mikael Sjödin. Neuropower: Designing energy efficient convolutional neural network architecture for embedded systems. In *International Conference on Artificial Neural Networks*, pages 208–222. Springer, 2019.

[64] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. Nsga-net: a multi-

objective genetic algorithm for neural architecture search. *arXiv preprint arXiv:1810.03522*, 2018.

[65] Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *arXiv preprint arXiv:1807.06906*, 2018.

[66] Bokai Cao, Lei Zheng, Chenwei Zhang, Philip S Yu, Andrea Piscitello, John Zulueta, Olu Ajilore, Kelly Ryan, and Alex D Leow. Deepmood: modeling mobile phone typing dynamics for mood detection. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 747–755, 2017.

[67] Lichao Sun, Yuqi Wang, Bokai Cao, S Yu Philip, Witawas Srisa-An, and Alex D Leow. Sequential keystroke behavioral biometrics for mobile user identification via multi-view deep learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 228–240. Springer, 2017.

[68] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[69] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629, 2018.

[70] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1651–1669, 2018.

[71] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[72] Amir Gholami, Zhewei Yao, Sehoon Kim, Michael W Mahoney, and Kurt Keutzer. Ai and memory wall. *RiseLab Medium Post*, 2021.

[73] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. *arXiv preprint arXiv:2007.10319*, 2020.

[74] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[75] Mohammad Loni, Hamid Mousavi, Mohammad Riazati, Masoud Daneshtalab, and Mikael Sjödin. Tas: Ternarized neural architecture search for resource-constrained edge devices. In *Design, Automation & Test in Europe Conference & Exhibition DATE'22, 14 March 2022, Antwerp, Belgium*. IEEE, March 2022.

[76] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[77] Najmeh Nazari, Mohammad Loni, Mostafa E. Salehi, Masoud Daneshtalab, and Mikael Sjödin. Tot-net: An endeavour toward optimizing ternary neural networks. In *2019 22st Euromicro Conference on Digital System Design (DSD)*. IEEE, 2019.

[78] Peng Chen, Bohan Zhuang, and Chunhua Shen. Fatnn: Fast and accurate ternary neural networks. *arXiv preprint arXiv:2008.05101*, 2020.

[79] Yue Li, Wenrui Ding, Chunlei Liu, Baochang Zhang, and Guodong Guo. Trq: Ternary neural networks with residual quantization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 8538–8546, 2021.

[80] Diwen Wan, Fumin Shen, Li Liu, Fan Zhu, Jie Qin, Ling Shao, and Heng Tao Shen. Tbn: Convolutional neural network with ternary inputs and binary weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 315–332, 2018.

[81] Yao Chen, Kai Zhang, Cheng Gong, Cong Hao, Xiaofan Zhang, Tao Li, and Deming Chen. T-dla: An open-source deep learning accelerator for ternarized dnn models on embedded fpga. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 13–18. IEEE, 2019.

[82] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

[83] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 55, 2014.

[84] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[85] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[86] 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.

[87] G Dodig-Crnkovic. Scientific methods in computer science:[ ]/gordana dodig-crnkovic. In *Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde*, 2002.

[88] Hilary J Holz, Anne Applin, Bruria Haberman, Donald Joyce, Helen Purchase, and Catherine Reed. Research methods in computing: what are they, and how should we teach them? *ACM SIGCSE Bulletin*, 38(4):96–114, 2006.

[89] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. *arXiv:2007.04074 [cs.LG]*, 2020.

[90] 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.

[91] Tonmoy Saikia, Yassine Marrakchi, Arber Zela, Frank Hutter, and Thomas Brox. Autodispnet: Improving disparity estimation with automl. *arXiv preprint arXiv:1905.07443*, 2019.

[92] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pages 4095–4104. PMLR, 2018.

[93] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[94] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

[95] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2017.

[96] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019.

[97] Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Jiancheng Lv. Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE Transactions on Cybernetics*, 2020.

[98] Masanori Suganuma, Masayuki Kobayashi, Shinichi Shirakawa, and Tomoharu Nagao. Evolution of deep convolutional neural networks using cartesian genetic programming. *Evolutionary Computation*, 28(1):141–163, 2020.

[99] Mohammad Loni, Amin Majd, Abdolah Loni, Masoud Daneshtalab, Mikael Sjödin, and Elena Troubitsyna. Designing compact convolutional neural network for embedded stereo vision systems. In *2018 IEEE 12th International Symposium on Embedded Multicore/Manycore Systems-on-Chip (MCSoC)*, pages 244–251. IEEE, 2018.

[100] Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. Path-level network transformation for efficient architecture search. In *In-*

ternational Conference on Machine Learning, pages 678–687. PMLR, 2018.

[101] Thomas Elsken, Frank Hutter, and Jan Hendrik Metzen. Efficient multi-objective neural architecture search via Lamarckian evolution. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[102] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.

[103] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[104] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[105] Wei Tang, Gang Hua, and Liang Wang. How to train a compact binary neural network with high accuracy? In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[106] Yufei Ma, Naveen Suda, Yu Cao, Jae-sun Seo, and Sarma Vrudhula. Scalable and modularized rtl compilation of convolutional neural networks onto fpga. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8. IEEE, 2016.

[107] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M Aamodt, and Andreas Moshovos. Stripes: Bit-serial deep neural network computing. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12. IEEE, 2016.

[108] Adrien Prost-Boucle, Alban Bourge, Frédéric Pétrot, Hande Alemdar, Nicholas Caldwell, and Vincent Leroy. Scalable high-performance architecture for convolutional ternary neural networks on fpga. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–7. IEEE, 2017.

[109] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. Torch: a modular machine learning software library. Technical report, Idiap, 2002.

[110] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[111] Rowel Atienza. Fast disparity estimation using dense networks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3207–3212. IEEE, 2018.

[112] Reza Mahjourian, Martin Wicke, and Anelia Angelova. Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5667–5675, 2018.

[113] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. End-to-end learning of geometry and context for deep stereo regression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 66–75, 2017.

[114] Wenjie Luo, Alexander G Schwing, and Raquel Urtasun. Efficient deep learning for stereo matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5695–5703, 2016.

[115] Feihu Zhang, Victor Prisacariu, Ruigang Yang, and Philip HS Torr. Ganet: Guided aggregation net for end-to-end stereo matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 185–194, 2019.

[116] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5410–5418, 2018.

[117] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train

convolutional networks for disparity, optical flow, and scene flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4040–4048, 2016.

[118] Alessio Tonioni, Fabio Tosi, Matteo Poggi, Stefano Mattoccia, and Luigi Di Stefano. Real-time self-adaptive deep stereo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 195–204, 2019.

[119] Dahyun Kim, Kunal Pratap Singh, and Jonghyun Choi. Learning architectures for binary networks. In *European Conference on Computer Vision*, pages 575–591. Springer, 2020.

[120] Bahareh Pourbabaee, Mehrsan Javan Roshtkhari, and Khashayar Khorasani. Deep convolutional neural networks and learning ecg features for screening paroxysmal atrial fibrillation patients. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(12):2095–2104, 2018.