

# HAPPi-Net

Hardware-Aware Performant Perception  
of Neural Networks



Alexander Frickenstein





## **HAPPi-Net: Hardware-Aware Performant Perception of Neural Networks**

**Alexander Frickenstein**

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

**Doktor-Ingenieurs (Dr.-Ing.)**

genehmigten Dissertation.

**Vorsitzender:**

Prof. Dr.-Ing. Georg Sigl

**Prüfende der Dissertation:**

1. Prof. Dr.-Ing. Walter Stechele
2. Prof. Maurizio Martina, PhD.

Die Dissertation wurde am 14.12.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 19.03.2021 angenommen.



Berichte aus der Informatik

**Alexander Frickenstein**

# **HAPPi-Net**

Hardware-Aware Performant Perception of Neural Networks

Shaker Verlag  
Düren 2021

**Bibliographic information published by the Deutsche Nationalbibliothek**

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at <http://dnb.d-nb.de>.

Zugl.: München, Techn. Univ., Diss., 2021

Copyright Shaker Verlag 2021

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the publishers.

Printed in Germany.

ISBN 978-3-8440-8069-8

ISSN 0945-0807

Shaker Verlag GmbH • Am Langen Graben 15a • 52353 Düren

Phone: 0049/2421/99011-0 • Telefax: 0049/2421/99011-9

Internet: [www.shaker.de](http://www.shaker.de) • e-mail: [info@shaker.de](mailto:info@shaker.de)

# Abstract

Artificial neural networks are dominating a vast majority of application scenarios to date, and will surely extend their lead in the near future. Especially, the superior performance of convolutional neural networks (CNNs) for image processing tasks presents a promising use case in innovative and cutting-edge domains, such as robotics and autonomous driving. However, their dominance emerges from an ever-increasing memory intensity and computational complexity. In contrast to the increasing resource demand, real-world applications on embedded devices pose major challenges with regard to limited computing power, memory resources and available energy and/or latency budget for the deployment of CNNs in embedded settings. To counteract these challenges, this thesis presents a tripartite hardware-software co-design paradigm for the efficient application of CNNs on embedded accelerators. This allows the traversal through the design space by either a top-down, meet-in-the-middle or a bottom-up approach. Moreover, six novel optimization methods, on the three levels of abstraction, are presented in this dissertation, which further serve the illustration of the simplified design process. In detail, we present three different architectural optimization methods. One of which is the recently introduced autoencoder-based low-rank filter-sharing (ALF) technique. The compressed CNNs are applied to different central processing units (CPUs) and field programmable gate arrays (FPGAs) using algorithmic optimization techniques. By means of successive exploration and refinement steps, it is shown how more powerful CNN-based applications can be created and make use of orthogonal optimization methods like pruning, quantization and Winograd convolution. Furthermore, the increase in data-level parallelism is achieved by quantized neural networks. Here, binaryDAD-Net, a fully binarized neural network, is presented for semantic drivable area detection. In the same context, requirements of binary neural networks for the design of a runtime reconfigurable processing element, namely OrthrusPE, are made accessible. In summary, we show that the optimization of CNNs for embedded applications, such as in the field of autonomous driving, can only be achieved through the interaction of the three abstraction levels (using expert knowledge) and synergies of different compression techniques to arrive at a fruitful HW-CNN co-design.



# Zusammenfassung

Künstliche neuronale Netze dominieren derzeit die Mehrheit der Anwendungsszenarien und werden ihren Vorsprung in Zukunft sicherlich noch ausbauen. Insbesondere die überlegene Leistungsfähigkeit von faltenden neuronalen Netzen (engl. Convolutional Neural Networks - CNNs) bei Bildverarbeitungsaufgaben stellt einen vielversprechenden Anwendungsfall in innovativen und zukunftsweisenden Bereichen wie der Robotik und dem autonomen Fahren dar. Die Vorherrschaft von CNNs rührt jedoch von einem fortwährend steigenden Speicherbedarf und wachsender Rechenkomplexität. Eine zentrale Herausforderung stellt dabei der steigende Ressourcenbedarf im Hinblick auf den Einsatz von CNNs in praktischen Anwendungen auf eingebetteten Systemen dar, im Sinne der begrenzten Rechenleistung und Speicherressourcen, des verfügbaren Energiebedarf oder der erlaubten Latenzzeiten. Um diesen Herausforderungen zu genügen, stellt diese Arbeit ein dreigliedriges Hardware-Software-Co-Design-Paradigma für die effiziente Anwendung von CNNs auf eingebetteten Beschleunigern vor. Dies erlaubt das Durchqueren des Designraumes mittels eines Top-Down-, Meet-in-the-Middle- oder Bottom-Up-Ansatzes. Darüber hinaus werden in dieser Dissertation sechs neuartige Optimierungsverfahren vorgestellt, welche auf den drei Abstraktionsebenen fußen, um den vereinfachten Entwurfsprozess weiter veranschaulichen. Im Detail werden drei unterschiedliche architektonische Optimierungsmethoden vorgestellt. Eine dieser Varianten basiert auf der erst kürzlich vorgestellten Autoencoder-basierten Low-Rank-Filter-Sharing (ALF) Technik. Die komprimierten CNNs werden hierbei unter zur Hilfenahme algorithmischer Optimierungstechniken auf verschiedenen Hardwarebeschleunigern (CPUs oder FPGAs) angewendet. Anhand von aufeinanderfolgenden Explorations- und Verfeinerungsschritten wird gezeigt, wie leistungsfähigere Anwendungen auf der Basis von CNNs erstellt werden können, welche orthogonale Optimierungsverfahren wie das Beschneiden, die Quantisierung und die Winograd-Faltung nutzen. Darüber hinaus wird die Erhöhung der Parallelität auf Datenebene durch quantisierte neuronale Netze erreicht. Hier wird binaryDAD-Net, ein vollständig binarisiertes neuronales Netz, zur semantisch Detektion eines befahrbaren Bereiches vorgestellt. Im gleichen Zusammenhang werden Anforderungen von binären neuronalen Netzen an den Entwurf eines zur Laufzeit rekonfigurierbaren Verarbeitungselements, durch OrthrusPE, zugänglich gemacht. Zusammenfassend wird gezeigt, dass die Optimierung von CNNs für eingebettete Anwendungen, beispielsweise im Bereich des autonomen Fahrens, nur durch das Zusammenwirken der drei Abstraktionsebenen (unter Verwendung des Expertenwissens) und den Synergien verschiedener Kompressionstechniken erreicht werden kann, dass in einem vorteilhaften Hardware-CNN Co-Design gipfelt.





# Acknowledgments

First and foremost, I would like to thank my Ph.D. supervisor Prof. Walter Stechele. Walter is an excellent and experienced supervisor. His calm nature has always helped me over the past years, even in difficult moments, to have the necessary vision for further steps. In particular, his faith and certainty in the research project has helped to facilitate a successful cooperation between the university and BMW AG. Thanks to his enthusiasm, the initial idea has developed into a well working research team.

I would also like to thank my advisor Christian Unger for his excellent support. This is especially true for his engagement in challenging the ideas, questioning results and setting great goals.

Moreover, it was a great pleasure for me to work together with the Ph.D. colleagues Manoj-Rohit Vemparala and Nael Fafous, to push and question our ideas and to inspire each other's work. I will miss the endless coffee, sandwiches and long working evenings on campus with them. Manoj and Nael have made a significant contribution to the success of this work, for which I am grateful to them.

For the endless opportunities to expand the research and the support with resources, I would like to thank my department head Simon Fürst from BMW AG. I would also like to thank all colleagues from the feature team Asimov and the BMW autonomous driving campus who have accompanied and supported the project in the past years.

Finally, I would like to thank my family, Mom, Dad, Lukas and Isabel, my girlfriend Julia Leopoldseder and my friends Matthias Kreuz, Florian Schoppe and Dieter Werner for their warm-hearted support over the years.



# Contents

<b>List of Figures</b>	<b>v</b>
<b>List of Tables</b>	<b>vii</b>
<b>List of Algorithms</b>	<b>ix</b>
<b>List of Abbreviations</b>	<b>xi</b>
<b>List of Symbols</b>	<b>xvii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Objectives . . . . .	3
1.3 Contribution . . . . .	4
1.4 Thesis Outline . . . . .	5
<b>2 Background</b>	<b>9</b>
2.1 Neural Networks for Computer Vision . . . . .	9
2.1.1 Convolutional Neural Network . . . . .	9
2.1.2 Training of Neural Networks . . . . .	14
2.2 Datasets for Image Classification and Semantic Segmentation . . . . .	15
2.2.1 Image Classification . . . . .	15
2.2.2 Semantic Segmentation . . . . .	17
2.3 Optimization Methods for Convolutional Neural Networks . . . . .	18
2.3.1 Pruning . . . . .	19
2.3.2 Quantization . . . . .	21
2.3.3 Binarization . . . . .	22
2.3.4 Efficient Convolution Algorithms . . . . .	23
<b>3 Hardware Aware Performant Perception of Neural Networks</b>	<b>25</b>
3.1 Tripartite Representation of HW-CNN Co-Design . . . . .	26
3.2 Related Work . . . . .	30
3.2.1 DarwinAI's Structural Level Deployment . . . . .	30
3.2.2 The Hardware Vendors Design Process . . . . .	31
3.2.3 Full-Custom Optimization and Hardware . . . . .	34
3.3 Design Methodologies for Lightweight Neural Networks . . . . .	35
3.3.1 HW-Independent Dense Compression of Neural Networks . . . . .	36
3.3.2 SIMD-based Application of Irregular Pruned and Quantized CNNs . . . . .	38

3.3.3	OpenCL-based High Throughput CNN Implementation . . . . .	40
3.3.4	Meet-in-the-Middle Custom Accelerator Design for BNNs . . . . .	41
<b>4</b>	<b>Resource-aware Optimization</b>	<b>43</b>
4.1	Related Work . . . . .	43
4.2	Resource-aware Element-wise Pruning and Quantization . . . . .	46
4.2.1	Binary Search Accelerated Weight Pruning . . . . .	47
4.2.2	Pruning Order . . . . .	47
4.2.3	Magnitude-based Pruning and Fine-tuning . . . . .	49
4.2.4	Clustering-based Weight Sharing . . . . .	51
4.2.5	Fixed-Point Quantization of Cluster Centroids and Activations . . . . .	52
4.2.6	Compressed Sparse Weights . . . . .	53
4.3	Experimental Results . . . . .	53
4.3.1	Memory-based and Performance-based Compression . . . . .	53
4.3.2	Low Latency Applications . . . . .	55
4.3.3	Comparison with State-of-the-Art . . . . .	56
4.4	Discussion . . . . .	57
<b>5</b>	<b>Parallelized SIMD-based Algorithm for Pruned Convolutional Layers</b>	<b>59</b>
5.1	Related Work . . . . .	59
5.2	Dense-Sparse Convolution . . . . .	61
5.2.1	Dense Convolution . . . . .	62
5.2.2	Sparse Convolution . . . . .	65
5.2.3	Kernel-wise Joint Dense-Sparse Convolution . . . . .	66
5.3	Experimental Results . . . . .	67
5.3.1	Ablation Study . . . . .	68
5.3.2	Exploration of Thread-Level Parallelism . . . . .	70
5.3.3	Comparison with State-of-the-Art . . . . .	72
5.4	Discussion . . . . .	73
<b>6</b>	<b>Winograd Convolution and Quantization</b>	<b>75</b>
6.1	Related Work . . . . .	75
6.2	Winograd Convolution using OpenCL . . . . .	77
6.2.1	High Level Synthesis Architecture Design . . . . .	77
6.2.2	WinoCNN Building Blocks and Units . . . . .	78
6.3	Experimental Results . . . . .	81
6.3.1	Ablation Study . . . . .	81
6.3.2	Comparison with State-of-the-Art . . . . .	83
6.4	Discussion . . . . .	85
<b>7</b>	<b>ALF: Autoencoder-based Low-Rank Filter-Sharing</b>	<b>87</b>
7.1	Related Work . . . . .	87
7.2	Sparse Autoencoder-based Filter-wise Pruning . . . . .	89
7.2.1	Structural Elements of ALF-Block . . . . .	90

7.2.2	Training Flow of an ALF-Block based CNN . . . . .	91
7.2.3	Algorithmic Implementation . . . . .	94
7.2.4	Deployment of an ALF-Block . . . . .	95
7.3	Experimental Results . . . . .	95
7.3.1	Design Space Exploration . . . . .	96
7.3.2	Layer-wise Analysis for an Embedded Application . . . . .	99
7.3.3	Comparison with State-of-the-Art . . . . .	100
7.4	Discussion . . . . .	102
<b>8</b>	<b>Binarized Drivable Area Detection Neural Network</b>	<b>105</b>
8.1	Related Work . . . . .	105
8.2	Binary Drivable Area Detection . . . . .	108
8.2.1	Binary Encoder for Feature Extraction . . . . .	108
8.2.2	Binary Bottleneck with Enlarged Receptive Field . . . . .	109
8.2.3	Binary Decoder for Semantic Predictions . . . . .	111
8.2.4	Training and Algorithmic Implementation . . . . .	111
8.3	Experimental Results . . . . .	113
8.3.1	Design Space Exploration . . . . .	113
8.3.2	Training Binary Drivable Area Detection with Automatic Annotations . . . . .	116
8.3.3	Comparison with the State-of-the-Art . . . . .	117
8.4	Discussion . . . . .	119
<b>9</b>	<b>Runtime Configurable Processing Element for BNNs</b>	<b>121</b>
9.1	Related Work . . . . .	121
9.2	Processing Element Providing Two Modes of Operation . . . . .	122
9.2.1	Binary Operating Mode . . . . .	124
9.2.2	Fixed-Point Operating Mode . . . . .	126
9.2.3	Mode Switching and Partial Sum Accumulation . . . . .	126
9.3	Experimental Results . . . . .	128
9.3.1	Resource Utilization Analysis . . . . .	128
9.3.2	Dynamic Power Analysis . . . . .	130
9.4	Discussion . . . . .	131
<b>10</b>	<b>Conclusion</b>	<b>133</b>
	<b>Bibliography</b>	<b>135</b>



# List of Figures

2.1	Topology of the LeNet-5. . . . .	11
2.2	Visualization of a 2D-convolutional layer. . . . .	11
2.3	Visualization of a fully-connected layer. . . . .	12
2.4	Non-linear activation functions commonly used in neural networks. . . . .	13
2.5	Sample images from the MNIST dataset. . . . .	16
2.6	Sample images from the CIFAR-10 dataset. . . . .	16
2.7	Sample images from the ImageNet dataset. . . . .	17
2.8	Sample image from the CityScapes dataset. . . . .	17
2.9	Overview of common optimization methods for neural networks. . . . .	19
2.10	Weight, kernel, channel and filter-wise pruning of convolutional layers. . . . .	20
2.11	Difference between linear, logarithmic and non-linear quantization. . . . .	22
2.12	GEMM-based execution of a 2D-convolutional layer. . . . .	24
3.1	Tripartite representation of the HW-SW co-design paradigm. . . . .	27
3.2	Tripartite representation of the HW-CNN co-design paradigm. . . . .	28
3.3	Classification of SotA optimizations in the context of the HW-CNN co-design. . . . .	30
3.4	Intel Deep Learning deployment toolkit. . . . .	32
3.5	NVIDIA TensorRT is an SDK for high-performance DL inference. . . . .	33
3.6	Xilinx deep neural network development kit. . . . .	34
3.7	Classification of our optimizations in the context of the HW-CNN co-design. . . . .	36
3.8	Design methodology for HW-independent parallel deployment of pruned NN. . . . .	37
3.9	Design methodology for SIMD-based application of pruned/quantized CNNs. . . . .	39
3.10	Design methodology for HLS-based high throughput CNN implementation. . . . .	41
3.11	Design methodology for a meet-in-the-middle accelerator design for BNNs. . . . .	42
4.1	Structural overview of the resource-aware optimization framework. . . . .	46
4.2	Overview of the binary search accelerated element-wise pruning scheme. . . . .	47
4.3	Sensitivity of VGG16 against magnitude-based pruning. . . . .	48
4.4	Heuristics for pruning and fine-tuning of VGG16. . . . .	51
4.5	Performance improvements of RAO-based optimized variants of VGG16. . . . .	55
5.1	Dense-sparse convolution kernel for vectorized SIMD-based accelerator. . . . .	62
5.2	Non-zero weights at each kernel of VGG16 after element-wise pruning. . . . .	65
5.3	Acceleration of weight-wise and kernel-wise pruned variants of VGG16. . . . .	69
5.4	Ablation study of the dense-sparse convolution kernel. . . . .	70
5.5	Comparison of sparse, dense and dense-sparse convolutions for various layers. . . . .	71
5.6	Scalability of the dense-sparse convolution kernel on different CPUs. . . . .	72



*List of Figures*

6.1	Structural overview of the OpenCL-based CNN accelerator WinoCNN. . . . .	78
6.2	Throughput of 16-bit quantized variants of FCN-8s executed on WinoCNN. . .	84
7.1	Overview of the autoencoder-based low-rank filter-sharing block. . . . .	90
7.2	Structure of an ALF compressed variant of a convolutional layer. . . . .	95
7.3	Design space exploration for an additional expansion layer for ALF. . . . .	96
7.4	Design space exploration of ALF compressed variants of Plain-20. . . . .	97
7.5	Analysis of ALF-related training hyperparameters for Plain-20 on CIFAR-10. .	98
7.6	Layer-wise energy consumption and latency of ALF compressed CNN variants. .	100
7.7	Pareto-front analysis of the ALF technique and the SotA. . . . .	102
8.1	Structural overview of the binary drivable area detection neural network. . . . .	108
8.2	Comparison of the BPAC and the binary bottleneck of binaryDAD-Net. . . . .	110
8.3	Quantitative results of SotA models and binaryDAD-Net on CityScapes dataset. .	119
9.1	Running example of OrthrusPE in the binary Hadamard product mode. . . . .	125
9.2	Two operating modes of OrthrusPE on the 7 Series FPGA DSP48E1 Slice. . . . .	125
9.3	SIMD register utilization of OrthrusPE with and without partial operations. . .	126
9.4	Switch count and partial result memory analysis for OrthrusPE. . . . .	128
9.5	Synthesis results of OrthrusPE for LUT utilization across different frequencies. .	129
9.6	Dynamic power estimation of OrthrusPE at different design target frequencies . .	130
10.1	An optimal solution depends on the target application and its user group. . . . .	133

# List of Tables

4.1	Layer-wise compression results for LeNet-5 of RAO. . . . .	54
4.2	Layer-wise compression results for VGG16 of RAO. . . . .	54
4.3	Comparison of the RAO method with the state-of-the-art for LeNet-5. . . . .	56
4.4	Comparison of the RAO method with the state-of-the-art for VGG16. . . . .	56
5.1	Comparison of the performance of the DSC kernel with the state-of-the-art. . . . .	73
6.1	Accuracy comparison of mixed precision VGG16. . . . .	82
6.2	Resource consumption of mixed-precision VGG16 implementation. . . . .	82
6.3	Comparison of the mIOU for FCN-8s with different quantizations. . . . .	83
6.4	Comparison of WinoCNN with other HW accelerators for VGG16 and FCN-8s. . . . .	84
7.1	Comparison of the performance of ALF with the SotA on CIFAR-10. . . . .	101
7.2	Comparison of the performance of ALF with the SotA on ImageNet. . . . .	101
8.1	Selection of a SotA encoder/decoder configuration. . . . .	114
8.2	Local binary approximation of the encoder and decoder. . . . .	115
8.3	Choosing the bottleneck for the binary drivable area detection network. . . . .	116
8.4	BinaryDAD-Net’s performance on automatic annotations. . . . .	117
8.5	Prediction quality of multi and two class DeepLabV3+ and binaryDAD-Net. . . . .	117
8.6	Comparison of binaryDAD-Net with SotA semantic segmentation models. . . . .	118
9.1	Requirements of common BNN and their respective hardware operations. . . . .	123
9.2	Frequency dependent resource utilization of OrthrusPE-based implementations. . . . .	129



# List of Algorithms

5.1	SIMD-based Vectorized Winograd Convolution. . . . .	64
5.2	SIMD-based Vectorized Sparse Convolution. . . . .	66
7.1	Training an L-layer ALF-Net. . . . .	94
8.1	Algorithmic implementation for the training of binaryDAD. . . . .	112



# List of Abbreviations

- AD** autonomous driving.
- ADMM** alternating direction method of multipliers.
- AI** artificial intelligence.
- ALF** autoencoder-based low-rank filter-sharing.
- ALU** arithmetic logic unit.
- AMC** AutoML for Model Compression.
- ANN** artificial neural network.
- ARCS** Architecture of Computing Systems.
- ASIC** application-specific integrated circuit.
- ASPP** Atrous Spatial Pyramid Pooling.
- AVX** Advanced Vector Extensions.
- BC** Bayesian Compression.
- BEV** battery electric vehicle.
- binaryDAD** binary drivable area detection.
- BLAS** basic linear algebra subprograms.
- BNN** binarized neural network.
- BPAC** Binary Parallel Atrous Convolution.
- BRAM** block random-access memory.
- CNN** convolutional neural network.
- CPU** central processing unit.
- CRS** compressed row storage.
- CRV** Conference on Computer and Robot Vision.

*List of Abbreviations*

- CV** computer vision.
- CVPR** Conference on Computer Vision and Pattern Recognition.
- DAC** Design Automation Conference.
- DAD** drivable area detection.
- DATE** Design, Automation & Test in Europe Conference and Exhibition.
- DC** Deep Compression.
- DDPG** deep deterministic policy gradient.
- DL** deep learning.
- DLA** Deep Learning Accelerator.
- DNN** deep neural network.
- DNNDK** Deep Neural Network Deployment Kit.
- DNS** Dynamic Network Surgery.
- DPU** Deep Learning Processing Unit.
- DRAM** dynamic random-access memory.
- DSC** dense-sparse convolution.
- DSD** dense-sparse-dense.
- DSP** digital signal processor.
- ECU** electronic control unit.
- EIE** Efficient Inference Engine.
- FCN** Fully Convolutional Network.
- FFT** fast Fourier transform.
- FPGA** field programmable gate array.
- FPGM** Filter Pruning via Geometric Median.
- GAN** Generative Adversarial Network.
- GCC** GNU Compiler Collection.
- GEMM** general matrix-matrix multiplication.

- GOPS** giga operations per second.
- GPU** graphics processing unit.
- HDL** hardware description language.
- HIL** hardware-in-the-loop.
- HLS** high-level synthesis.
- HW** hardware.
- HW-CNN** hardware-convolutional neural network.
- IC** integrated circuit.
- ICRA** International Conference on Robotics and Automation.
- L-BFGS** limited-memory Broyden-Fletcher-Goldfarb-Shanno.
- L-OBS** Layer-wise Optimal Brain Surgeon.
- L2P** Learning to Prune.
- LCNN** Lookup-CNN.
- LRN** Local Response Normalization.
- LUT** look-up table.
- LWC** Learning both Weights and Connections.
- MAC** multiply-accumulate.
- MIT** Massachusetts Institute of Technology.
- MKL** Math Kernel Library.
- ML** machine learning.
- MLP** multi-layer perceptron.
- MMX** Multi Media Extension.
- MSE** mean squared error.
- NAS** neural architecture search.
- NCC** normalized compute complexity.
- NN** neural network.



*List of Abbreviations*

**NoC** network-on-chip.

**Op** operation.

**Param** parameter.

**PCI** Peripheral Component Interconnect.

**PE** processing element.

**QNN** quantized neural network.

**RAO** resource-aware optimization.

**ReLU** rectified linear unit.

**RL** reinforcement learning.

**RNN** recurrent neural network.

**RTL** register-transfer level.

**SDK** software development kit.

**SGD** stochastic gradient descent.

**SIMD** single instruction multiple data.

**SoC** system-on-chip.

**SotA** state-of-the-art.

**SQNR** signal-to-quantization-noise ratio.

**SRAM** static random-access memory.

**SSE3** Streaming SIMD Extensions 3.

**STE** straight-through estimator.

**SVM** support vector machine.

**SW** software.

**TDG** training data generator.

**TPU** Tensor Processing Unit.

**VHDL** Very High Speed Integrated Circuit Hardware Description Language.

*List of Abbreviations*

**VLSI** very-large-scale integration.

**VPU** Vision Processing Unit.

**XPE** Xilinx Power Estimator.



# List of Symbols

- $A$  Inverse Winograd transformation matrix.
- $A^{l-1}$  Input feature map.
- $A^{l-1}$  Binarized version of the input feature map.
- $A_B^{l-1}$  Input feature map of binaryDAD's bottleneck.
- $A_D^{l-1}$  Input feature map of binaryDAD's decoder.
- $A_E^{l-1}$  Input feature map of binaryDAD's encoder.
- $A^l$  Output feature map.
- $A_{\text{inter}}^l$  Intermediate features of an ALF-block.
- $B$  Winograd transformation matrix for activations.
- $B^l$  Set of binarized weights.
- $C_i$  Number of channels of an input feature map.
- $C_o$  Number of channels of an output feature map.
- $C_{\text{code}}$  Number of channels of the low-rank approximation.
- $D$  Parallelization factor of WinoCNN.
- $G$  Winograd transformation matrix for weights.
- $H_i$  Height of an input feature map.
- $H_o$  Height of an output feature map.
- $I$  Input image.
- $K$  Kernel dimensions.
- $L$  Layer of a neural network.
- $M$  Parallelization factor of WinoCNN.
- $M_{\text{ABC}}$  Number of binary weight bases of ABC-Net.
- $N$  Classes for classification.

### List of Symbols

$N_{\text{ABC}}$  Number of binary activation bases of ABC-Net.

$N_{\text{hadamard}}$  Number of bits per Hadamard product.

$P$  Padding (measured in pixels).

$P_{\text{Orthrus}}$  Set of pixels of OrthrusPE.

$Q$  Bit-width of quantized weights and activations.

$S$  Stride (measured in pixels).

$W$  Set of weights/trainable parameters.

$W^l$  Set of weights of a given layer.

$W_B^l$  Set of weights of binaryDAD-Net's bottleneck.

$W_D^l$  Set of weights of binaryDAD-Net's decoder.

$W_E^l$  Set of weights of binaryDAD-Net's encoder.

$W_i$  Width of the input feature map.

$W_o$  Width of the output feature map.

$W_{\text{code}}$  Low-rank approximation of the weights.

$W_{\text{dec}}$  Decoder filters of ALF.

$W_{\text{enc}}$  Encoder filters of ALF.

$W_{\text{rec}}$  Reconstructed parameters of ALF.

$X$  Set of input neurons.

$Y$  Ground-truth label.

$\Theta_l$  Batch norm parameters.

$\alpha$  Scaling factor for binarized weights.

$\beta$  Scaling factor for binarized input activations.

$\ell_1$  Absolute value criterion.

$\ell_2$  Quadratic mean square criterion.

$\epsilon$  Precision of quantized values.

$\eta_S$  Theoretical speedup of Amdahl's law.

$\lambda$  Strength of regularization.

- $\lambda_{\text{prune}}$  Pruning rate dependent scaling factor for ALF.
- $\lambda_{\text{wd}}$  Weight decay scaling factor for the training of an ALF-block.
- A** 2D-input feature map matrix for the GEMM operation.
- B** 2D-weight matrix for the GEMM operation.
- C** 2D-output feature map matrix for the GEMM operation.
- D** Dataset gathering paired samples of images  $I$  and corresponding labels  $Y$ .
- $\mathcal{K}_H$  Estimation of the memory requirement of Huffman coded data.
- $\mathcal{L}_{\text{CE}}$  Cross entropy loss.
- $\mathcal{L}_{\text{ae}}$  Autoencoder-specific loss function.
- $\mathcal{L}_{\text{prune}}$  Regularize the pruning mask.
- $\mathcal{L}_{\text{reg}}$  Regularization term.
- $\mathcal{L}_{\text{task}}$  Task-specific loss function.
- $\mathcal{M}$  Binary mask used for pruning.
- $\mathcal{O}$  Overall search complexity.
- BS** Batch size.
- FB** Number of fractional bits.
- IB** Number of integer bits.
- Pr** Pruning rate which is the ratio of pruned (zero) to the overall number of weights.
- S** Sign bit of quantized values.
- $\mu$  SGD-based momentum.
- $\nu$  Learning rate of CNN optimizer.
- $\nu_{\text{ae}}$  Autoencoder learning rate.
- $\sigma$  Activation function.
- $\tau$  Sparsity threshold.
- $\theta$  Fraction of zeroized to non-zeroized filters in an ALF-block.
- $\tilde{A}^l$  Linear combinations of intermediate features.
- $\tilde{Y}$  Prediction of the CNN.

## List of Symbols

- $\tilde{\mu}_i$  Fixed-point quantized centroid.
- $\zeta$  ALF-specific hyperparameter to distinguish if a filter is active or dropped.
- $a$  Single output pixel.
- $a^{l-1}$  Tensor slice/tile of the input feature map.
- $a^l$  Tensor slice/tile of the output feature map.
- $b$  Individual bias trainable parameter.
- $b^l$  Binarize slice of the weights.
- $c$  Index of a class/centroid.
- $d$  Dilation rate.
- $e$  Learning epochs, *i.e.* one iteration throughout all training samples.
- $f$  Frequency.
- $f_p$  Parallel fraction code.
- $f_s$  Sequential fraction code.
- $f_{\max}$  Upper bound of the quantization limits.
- $f_{\min}$  Lower bound of the quantization limits.
- $g$  Noise scale.
- $g_A$  Gradients of the activations.
- $g_B$  Gradients of the binary weights.
- $g_W$  Gradients of the weights.
- $h^{l-1}$  Binarize tensor slice of the input feature map.
- $i$  Index parameter.
- $l$  Index parameter for a layer.
- $m_i$  Individual elements of the binary pruning mask.
- $m_{\text{slope}}$  Slope of the sensitivity of the ALF-specific pruning factor.
- $n_p$  Number of parallel threads/cores.
- $p$  Number of quantization intervals.
- $p_i$  Pseudo probability.

- $pr_{\mathbf{high}}$  Upper bound of binary search based pruning.
- $pr_{\mathbf{low}}$  Lower bound of binary search based pruning.
- $pr_{\mathbf{max}}$  Maximum desired pruning rate of an ALF-block.
- $pr_{\mathbf{mid}}$  Actual pruning rate of binary search based pruning.
- $t$  Training step.
- $v_t$  SGD-based accumulator at training step  $t$ .
- $w$  Individual weight/trainable parameter.
- $w^l$  Tensor slice of the weights.
- $x$  Individual input neuron.
- $y$  Individual output neuron of a neural network.