FPGAs and Neural Networks

Clemens Lechner

January 24, 2024

Motivation and Goals

- Basics of *neural networks (NNs)* and important *factors for hardware design*
- Why *traditional computer architectures* struggle with NNs
- An overview of state of the art *NN accelerators*
- An *example NN accelerator* architecture on an FPGA
- A short outlook on *future technologies*

Introduction to Neural Networks

Aritificial Neuron

- Fundamental *building block* of neural networks [1]
- Learnable *weights*
- Usually nonlinear *activation function* σ
- *Resources* used:
    - *Memory:* weights (and inputs)
    - *Computational:* Multiply-Accumulate *(MAC)* and activation

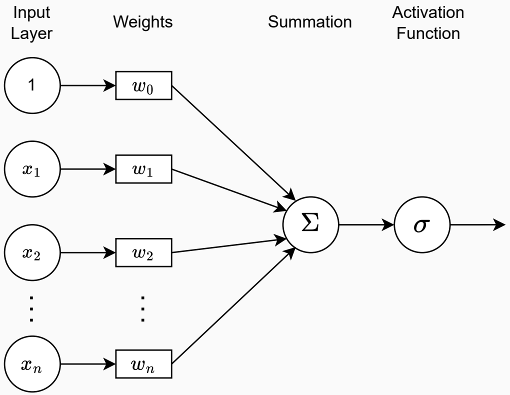


Figure 1: Artificial Neuron

Neural Network

- Network of *Artificial Neurons* [1]
- Architecture *varies* depending on *application* and required *model capacity*

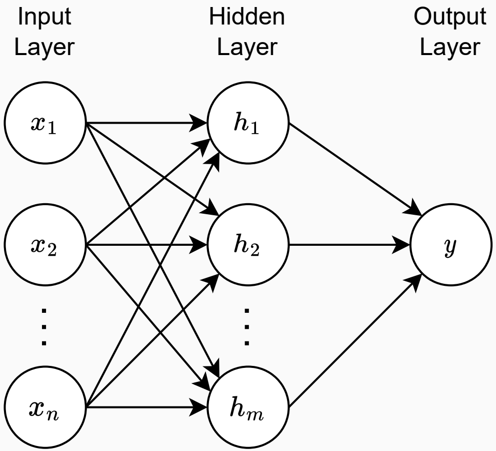


Figure 2: Neural Network

Training: Backpropagation and Gradient Descent

- An appropriate *loss function* is added after the *output layer* [1]
- Loss is *evaluated* for *training samples* consisting of inputs $\langle x \rangle$ and targets $\langle t \rangle$
- *Gradient* of loss is computed and *propagated back* through the network *(Backpropagation)*
- *Weights* are *"nudged"* proportionally to their negative gradients, *minimizing loss (Gradient Descend)*

Training vs. Inference

- Can be viewed as two *separate problems*
- Training:
  - Usually only performed *during development*
  - Resource intensive
- Inference:
  - *Deploy* a pretrained model
  - Less resources needed
- Two *different hardware platforms* could be considered (e.g.: train on server deploy on edge/embedded)

Fields of Application

Selection of *fields of application*: [2]

- Speech recognition
- Computer vision
- Pattern recognition
- Natural language processing
- Finance

Neural Networks on Hardware Platforms

Von Neumann Architecture

- Basis for traditional computer architectures
- *Memory Access Bottleneck* [3]
- Traditionally *no parallelization* [3]
- Modern CPUs still have *low degree of parallelization*

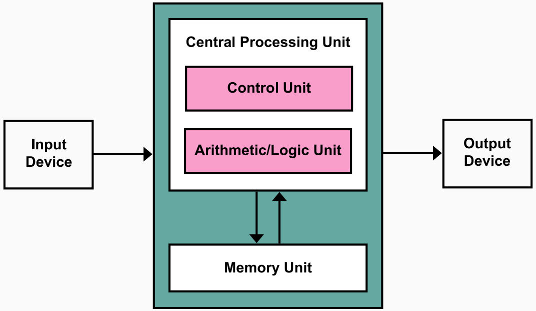


Figure 3: Von Neumann Architecture [4]

(GP)GPU: Solving Parallelization

- GPUs originally designed for graphics workloads *(high parallelization and memory bandwidth)* [5]
- *GPGPU:* general purpose computing on GPUs [5]
- APIs: CUDA (NVIDIA), OpenCL [6]
- Now: dedicated *"AI GPUs"* like NVIDIA H100 Tensor Core GPU [7]

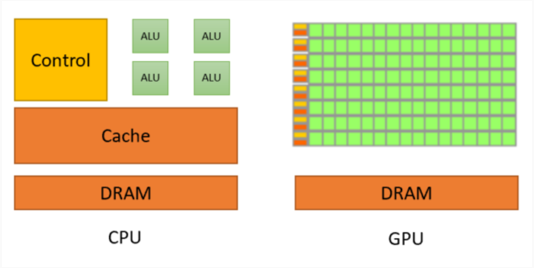


Figure 4: CPU vs GPU architecture [5]

Dedicated Neural Network Accelerators

Try to solve two problems specifically:

- *Parallelization* of matrix multiplication
- Von Neumann *memory bottleneck*

Usually designed as *coprocessors* either as

- *standalone* ASIC (i.e.: Google TPU [8]) or as
- IP in *SoC* or *FPGA*

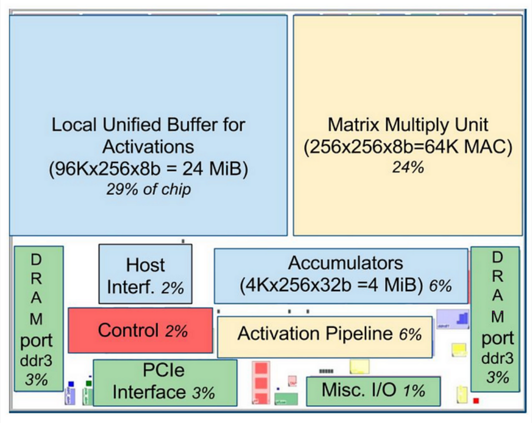

Local Unified Buffer for Activations (96Kx256x8b = 24 MiB) 29% of chip
Matrix Multiply Unit (256x256x8b=64K MAC) 24%
DRAM port ddr3 3%
Host Interf. 2%
Accumulators (4Kx256x32b =4 MiB) 6%
DRAM port ddr3 3%
Control 2%
Activation Pipeline 6%
PCIe Interface 3%
Misc. I/O 1%

Figure 5: Floor plan of Google Tensor Processing Unit (TPU) [8]

NN Accelerators on FPGAs

Currently neural network architectures are subject to *fast-paced improvements*. Thus, the *flexibility of FPGAs* could be leveraged for NN Accelerators:

- Can be tailored to a *specific NN architecture*
- Could complement software via *runtime partial reconfiguration*
- Can leverage *High Level Sythesis* for rapid development
- Rollout of Over-The-Air *(OTA) updates*

Example Architecture: ZynqNet

ZynqNet Overview

- *Image classification* using a *convolutional neural network (CNN)* architecture based on SqueezeNet [6]
- Accelerator for *Inference* on Zynq-7000 Series SoC
- Design for *high throughput* and *real-time* classification

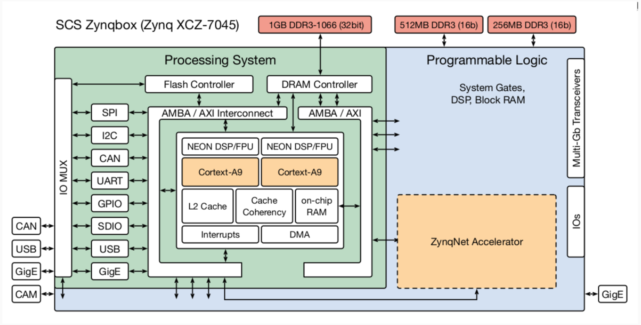


Figure 6: Zynqbox Embedded Platform with ZynqNet Accelerator [6]

Convolutional Neural Networks (CNNs)

- NN architecture particularly suited for *operations on image data* [6]
- Several *learnable* convolutional *filter kernels*
- Exploit *locality of information* in images
- *Weight sharing* through convolution operation
- Can be computed using nested loops *(high parallelism through loop unrolling)*

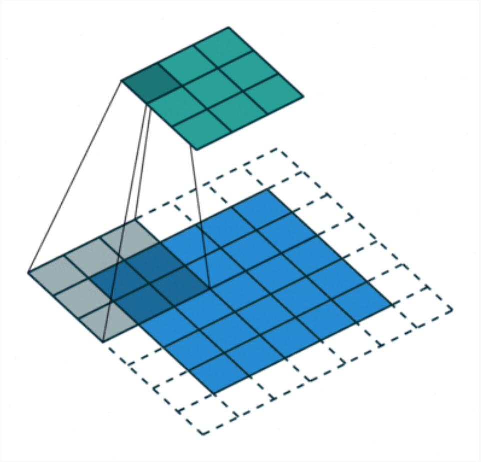


Figure 7: Convolution Operation [9]

Convolutional Neural Networks (CNNs)

- NN architecture particularly suited for *operations on image data* [6]
- Several *learnable* convolutional *filter kernels*
- Exploit *locality of information* in images
- *Weight sharing* through convolution operation
- Can be computed using nested loops *(high parallelism through loop unrolling)*

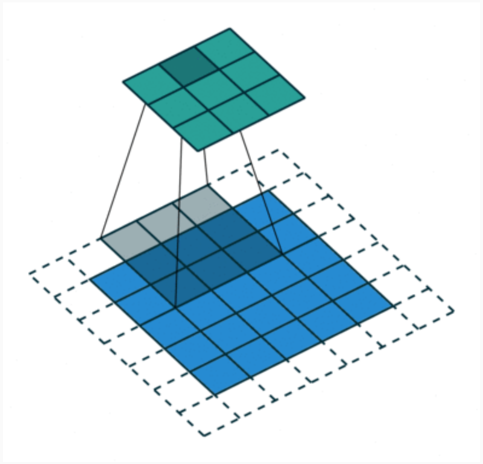


Figure 7: Convolution Operation [9]

Convolutional Neural Networks (CNNs)

- NN architecture particularly suited for *operations on image data* [6]
- Several *learnable* convolutional *filter kernels*
- Exploit *locality of information* in images
- *Weight sharing* through convolution operation
- Can be computed using nested loops *(high parallelism through loop unrolling)*

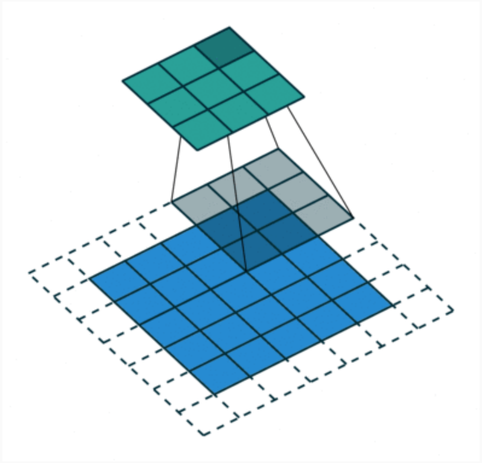


Figure 7: Convolution Operation [9]

Convolutional Neural Networks (CNNs)

- NN architecture particularly suited for *operations on image data* [6]
- Several *learnable* convolutional *filter kernels*
- Exploit *locality of information* in images
- *Weight sharing* through convolution operation
- Can be computed using nested loops *(high parallelism through loop unrolling)*

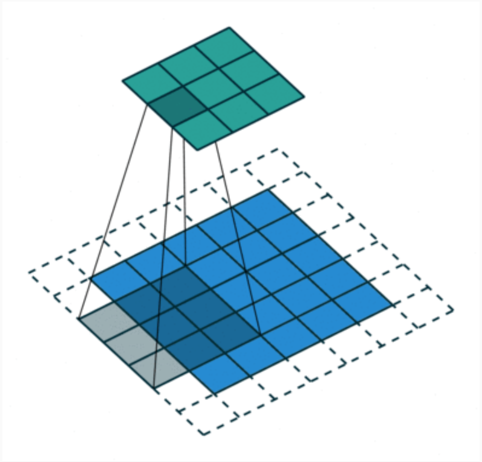


Figure 7: Convolution Operation [9]

Convolutional Neural Networks (CNNs)

- NN architecture particularly suited for *operations on image data* [6]
- Several *learnable* convolutional *filter kernels*
- Exploit *locality of information* in images
- *Weight sharing* through convolution operation
- Can be computed using nested loops *(high parallelism through loop unrolling)*

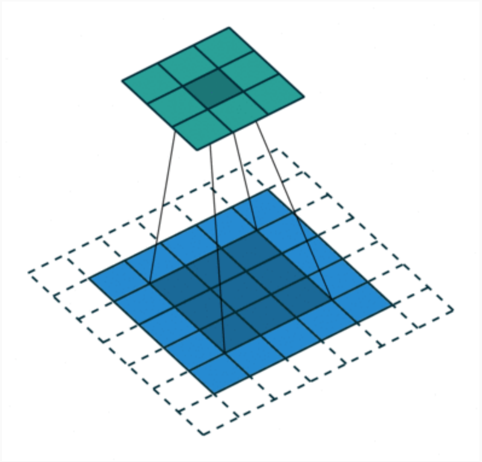


Figure 7: Convolution Operation [9]

Convolutional Neural Networks (CNNs)

- NN architecture particularly suited for *operations on image data* [6]
- Several *learnable* convolutional *filter kernels*
- Exploit *locality of information* in images
- *Weight sharing* through convolution operation
- Can be computed using nested loops *(high parallelism through loop unrolling)*

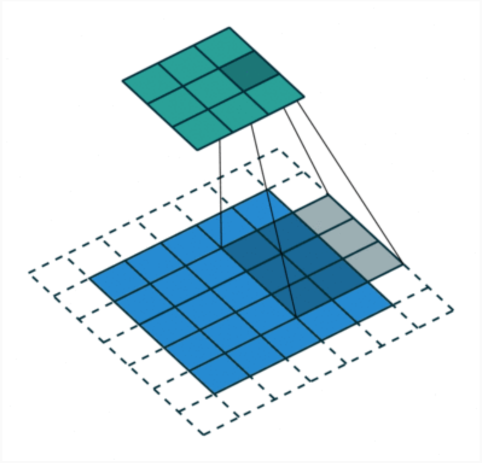


Figure 7: Convolution Operation [9]

ZynqNet Topology

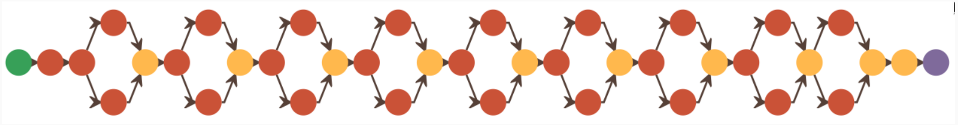


Figure 8: High-Level Visualization of the ZynqNet Topology. Red dots symbolize *Convolutional Layers* with ReLu Activations, yellow dots *Concatenations* or the final *Average Pooling*. [6]

Figure 9: Sample Images from the *ImageNet Dataset* (white shark, banana, volcano, fire engine, pomeranian, space shuttle, toilet paper) [6]

ZynqNet Hardware Architecture

Implemented as *AXI-Peripheral* [6]

- $N_{PE} = 16$ parallel MAC units for filter dot products
- Uses single-precision *floating-point arithmetic*
- *Caches* for
  - Image (ICache)
  - Outputs (OCache)
  - Weights (WCache)
  - Global Pooling (GPoolCache)
- Implemented with *ideal caching* strategy

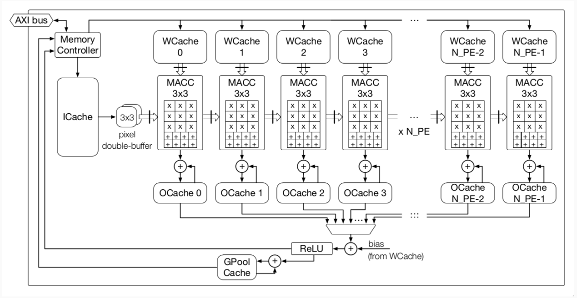
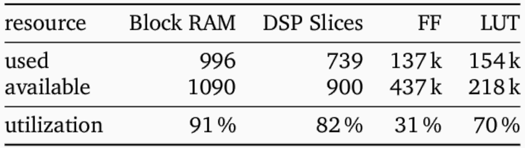


Figure 10: Block Diagram of ZynqNet accelerator [6]

Results (1)

- Maximum operating frequency: $f_{max} = 200\,\mathrm{MHz}$ [6]
- Throughput: $1.95\,\mathrm{s}$ per frame vs $45\,\mathrm{s}$ on ARM CPU at $f = 100\,\mathrm{MHz}$
- Stated improvements:
  - Configuring the FPGA clock at $f_{max} = 200\,\mathrm{MHz}$
  - Using 16-bit *fixed-point* instead of single-precision floating-point
  - Resolving a *pipeline flushing issue* in the High-Level-Synthesis (slow-down factor of 6.2x)

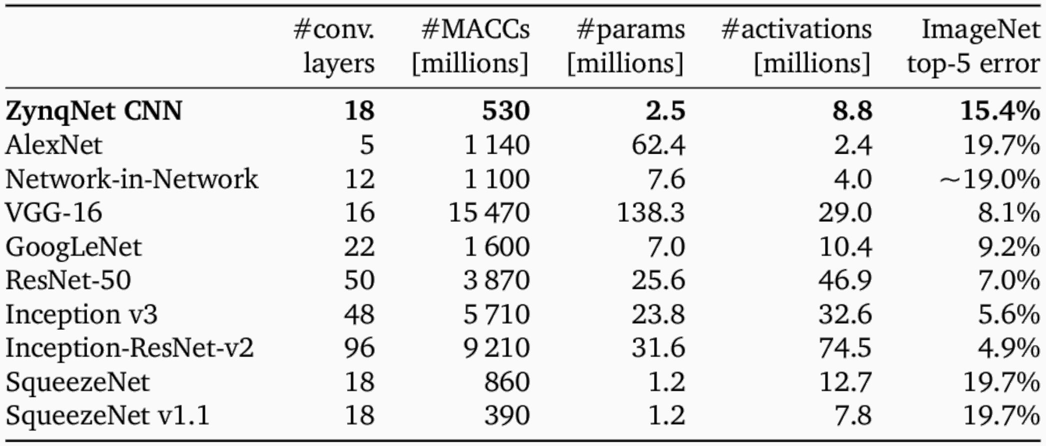
Table 1: Resource Requirements and FPGA Utilization of ZynqNet when synthesized for Zynq XC-7Z045. [6]



resource	Block RAM	DSP Slices	FF	LUT
used	996	739	137 k	154 k
available	1090	900	437 k	218 k
utilization	91 %	82 %	31 %	70 %

Results (2)

Table 2: Comparison of ZynqNet CNN to CNN Architectures from prior work [6]



	#conv. layers	#MACCs [millions]	#params [millions]	#activations [millions]	ImageNet top-5 error
ZynqNet CNN	**18**	**530**	**2.5**	**8.8**	**15.4%**
AlexNet	5	1 140	62.4	2.4	19.7%
Network-in-Network	12	1 100	7.6	4.0	~19.0%
VGG-16	16	15 470	138.3	29.0	8.1%
GoogLeNet	22	1 600	7.0	10.4	9.2%
ResNet-50	50	3 870	25.6	46.9	7.0%
Inception v3	48	5 710	23.8	32.6	5.6%
Inception-ResNet-v2	96	9 210	31.6	74.5	4.9%
SqueezeNet	18	860	1.2	12.7	19.7%
SqueezeNet v1.1	18	390	1.2	7.8	19.7%

Neural Network Quantization

Reducing weight and activation precision for inference. [10]

- Enables *small, low-latency and energy efficient* neural network solutions.
- Two main classes of algorithms:
  - Post-Training-Quantization *(PTQ)*
  - Quantization-Aware-Training *(QAT)*
- Example: Moving from 32 to 8 bits:
  - *Memory* overhead decreases by *factor of 4*.
  - *Computational cost* for matrix multiplications decreases quadratically by *factor of 16*.
- Moving from *Floating-Point to Fixed-Point* eliminates need for floating-point logic

Quantization Grids Comparison

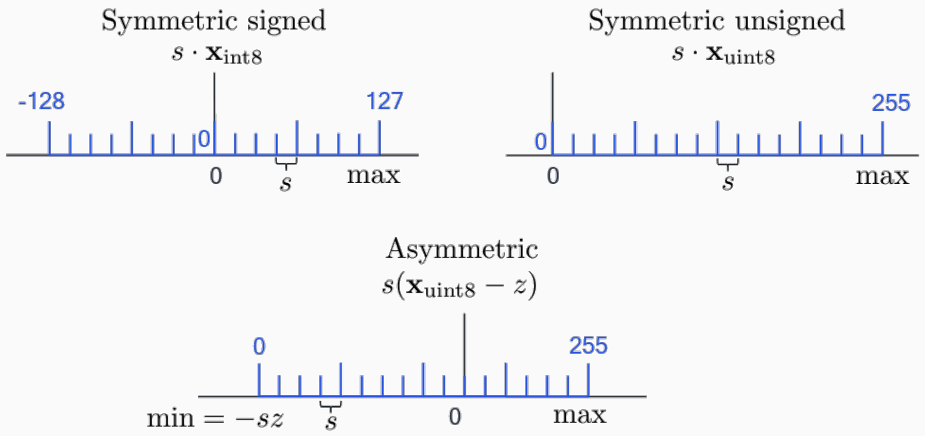

Symmetric signed
$s \cdot \mathbf{x}_{\text{int8}}$
-128
127
0
0
s
max
Symmetric unsigned
$s \cdot \mathbf{x}_{\text{uint8}}$
255
0
0
s
max
Asymmetric
$s(\mathbf{x}_{\text{uint8}} - z)$
0
255
$\min = -sz$
s
0
max

Figure 11: A visual explanation of the different uniform quantization grids for a bit width of 8. [10]

Post-Training-Quantization (PTQ)

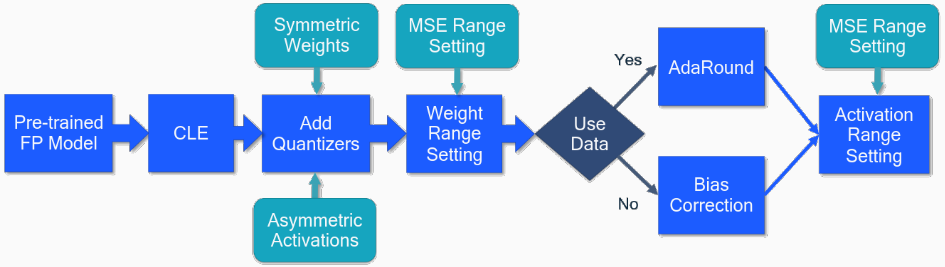
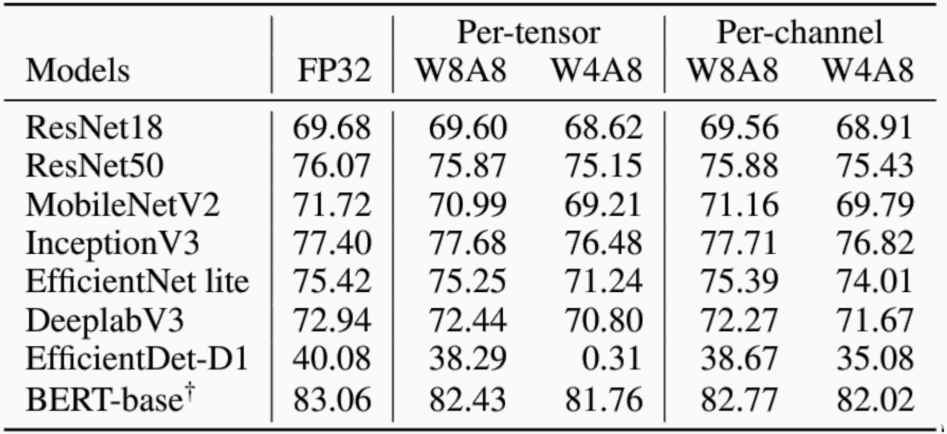


Figure 12: Standard PTQ pipeline according to [10]. CLE stands for Cross-layer-equalization.

PTQ Performance

Table 3: Performance (average over 5 runs) of PTQ pipeline for various models and tasks. *Higher is better* in all cases. [10]



		Per-tensor		Per-channel	
Models	FP32	W8A8	W4A8	W8A8	W4A8
ResNet18	69.68	69.60	68.62	69.56	68.91
ResNet50	76.07	75.87	75.15	75.88	75.43
MobileNetV2	71.72	70.99	69.21	71.16	69.79
InceptionV3	77.40	77.68	76.48	77.71	76.82
EfficientNet lite	75.42	75.25	71.24	75.39	74.01
DeeplabV3	72.94	72.44	70.80	72.27	71.67
EfficientDet-D1	40.08	38.29	0.31	38.67	35.08
BERT-base†	83.06	82.43	81.76	82.77	82.02

Quantization-Aware-Training (QAT)

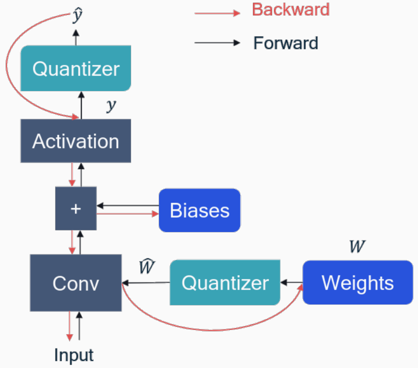


Figure 13: Forward and backward computation graph for Quantization-Aware-Training [10]

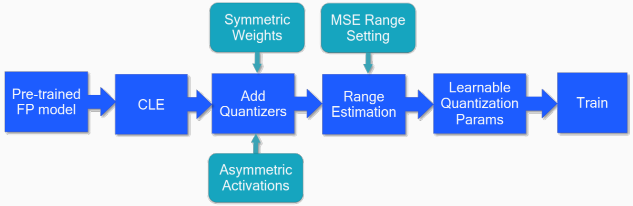
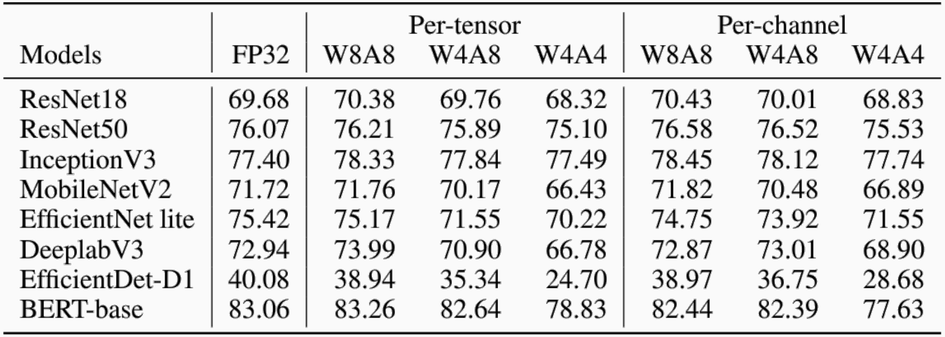


Figure 14: Standard QAT pipeline according to [10]. CLE stands for Croll-layer-equalization.

QAT Performance

Table 4: Performance (average over 3 runs) of QAT pipeline for various models and tasks. *Higher is better* in all cases. [10]



		Per-tensor			Per-channel		
Models	FP32	W8A8	W4A8	W4A4	W8A8	W4A8	W4A4
ResNet18	69.68	70.38	69.76	68.32	70.43	70.01	68.83
ResNet50	76.07	76.21	75.89	75.10	76.58	76.52	75.53
InceptionV3	77.40	78.33	77.84	77.49	78.45	78.12	77.74
MobileNetV2	71.72	71.76	70.17	66.43	71.82	70.48	66.89
EfficientNet lite	75.42	75.17	71.55	70.22	74.75	73.92	71.55
DeeplabV3	72.94	73.99	70.90	66.78	72.87	73.01	68.90
EfficientDet-D1	40.08	38.94	35.34	24.70	38.97	36.75	28.68
BERT-base	83.06	83.26	82.64	78.83	82.44	82.39	77.63

Outlook: Breaking the Memory Bottleneck

In-Memory Computing (IMC)

Idea: *Compute in situ* instead of in separate memory and compute units [11]
Problem: Low density of SRAM; difficulties combining DRAM with logic

- Multiple new *nonvolatile memory (NVM)* technologies are developed to overcome IMC's integration issues.
- *Analog or mixed approaches* to solve common operations are explored: e.g. matrix-vector multiplication via Ohm's and Kirchhoff's laws in a memory array.

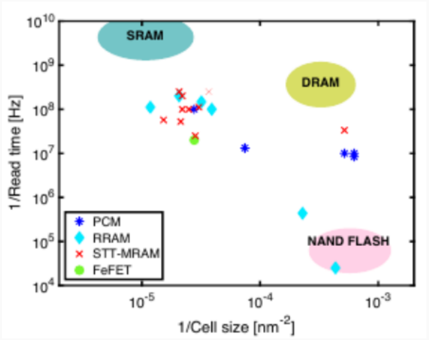


Figure 15: Performance characteristics of various emerging memory demonstrators. [11]

References i

References

[1] F. Pernkopf and C. Knoll, Computational intelligence (lecture notes), 2021.

[2] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, State-of-the-art in artificial neural network applications: A survey,, 2018. [Online]. Available: https://www.cell.com/heliyon/pdf/S2405-8440(18)33206-7.pdf.

[3] I. Arikpo, F. Ogban, and I. Eteng, Von neumann architecture and modern computers, Global Journal of Mathematical Sciences, vol. 6, no. 2, pp. 97–103, 2007.

[4] W. Commons, Von neumann architecture, (2013), [Online]. Available: https://commons.wikimedia.org/wiki/File:Von_Neumann_Architecture.svg.

References ii

[5] P. Gupta, Cuda refresher: Reviewing the origins of gpu computing,, 2020. [Online]. Available: https://developer.nvidia.com/blog/cuda-refresher-reviewing-the-origins-of-gpu-computing/.

[6] D. Gschwend, Zynqnet: An fpga-accelerated embedded convolutional neural network, arXiv preprint arXiv:2005.06892, 2020.

[7] Nvidia h100 tensor core gpu datasheet, NVIDIA Corporation, 2023. [Online]. Available: https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet.

[8] N. P. Jouppi, C. Young, N. Patil, *et al.*, In-datacenter performance analysis of a tensor processing unit, in Proceedings of the 44th annual international symposium on computer architecture, 2017, pp. 1–12.

[9] S. Saha, A comprehensive guide to convolutional neural networks,, 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

References iii

[10] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, A white paper on neural network quantization, arXiv preprint arXiv:2106.08295, 2021.

[11] S. Yu, H. Jiang, S. Huang, X. Peng, and A. Lu, Compute-in-memory chips for deep learning: Recent trends and prospects, IEEE Circuits and Systems Magazine, vol. 21, no. 3, pp. 31–56, 2021. DOI: 10.1109/MCAS.2021.3092533.