TECHNISCHE UNIVERSITÄT WIEN
Vienna | Austria

Institut für Computertechnik
Institute of Computer Technology

Embedded Machine Learning

Axel Jantsch

TU Wien, Vienna, Austria

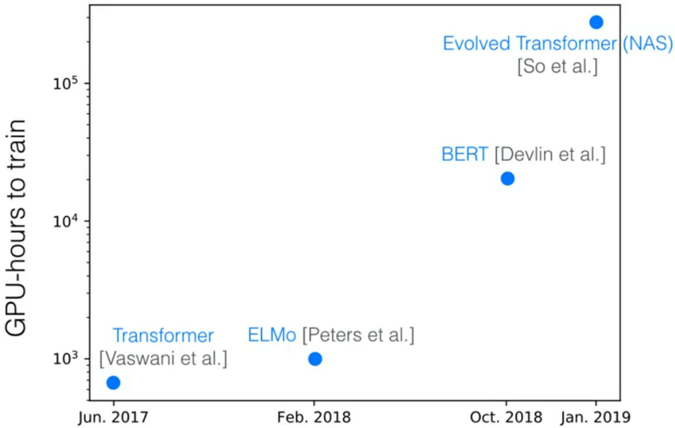
December 9, 2024

Outline

1. Challenge and Motivation
2. Convolutional Neural Networks
   - HW Friendly Optimizations
   - CNN Accelerator Architectures
3. Quantization
4. Profiling
5. Estimation
   - Enhanced Roofline Model
   - Step-wise Linear Model
6. Traffic Light Controller Case Study
7. Summary

Challenge and Motivation

ML is Resource demanding



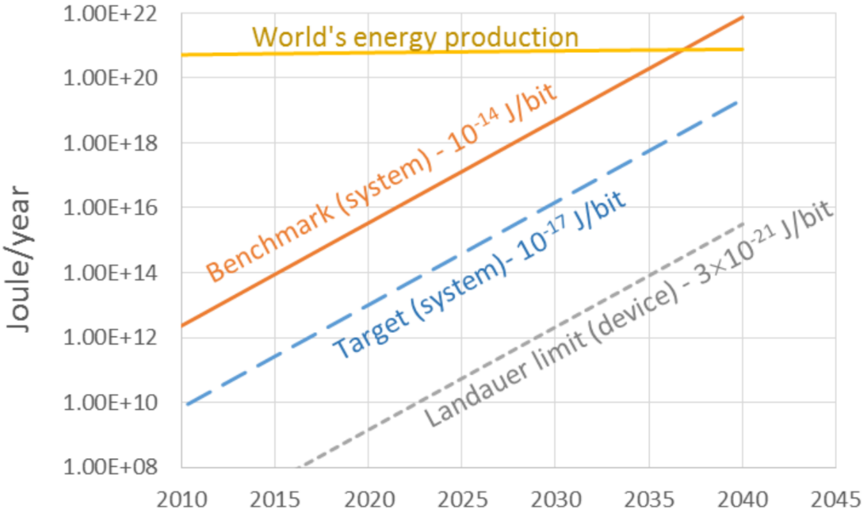
Full architecture search for a big transformer model requires

- 979 M training steps and
- 32 623 h of TPU or 274 120 h on 8 P100 GPUs,
- 656 MWh energy for training
- carbon footprint equivalent to the **lifetime of 5 US cars**.

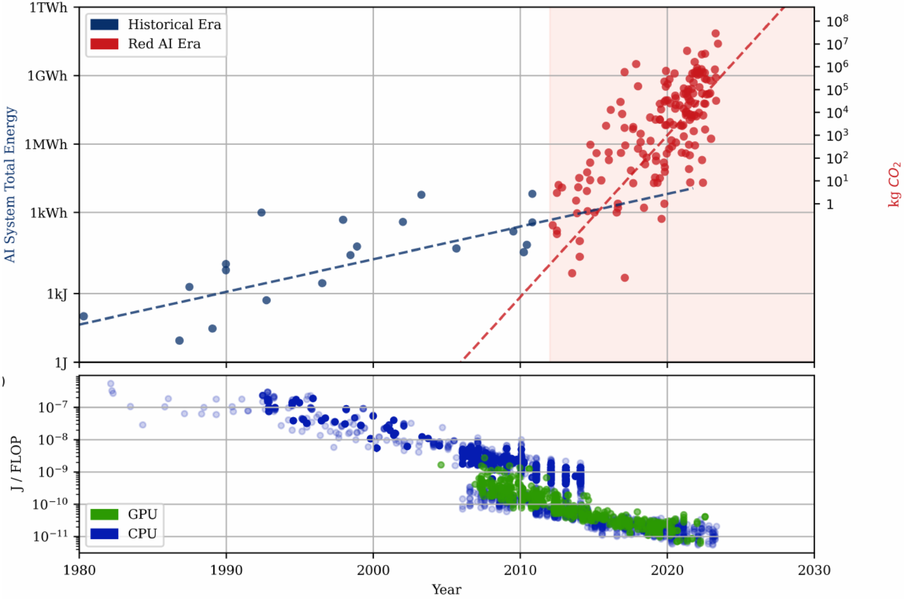
NAS based training is beyond the reach of most organizations

Emma Strubell, Ananya Ganesh **and** Andrew McCallum. "Energy and Policy Considerations for Deep Learning in NLP". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, **july** 2019, **pages** 3645–3650

ML is Resource Usage is Unsustainable

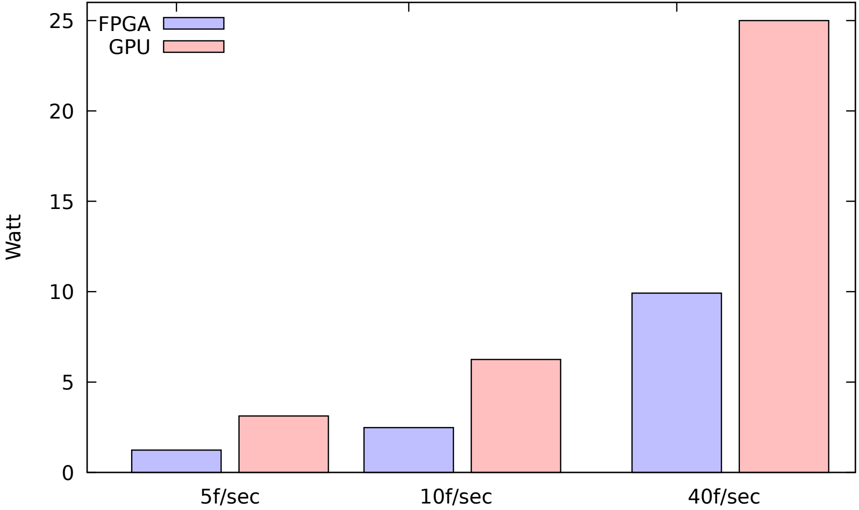


SIA - SRC. *Rebooting the IT Revolution: A Call to Action*. techreport. Semiconductor Industry Association **and** Semiconductor Research Corporation, **september** 2015



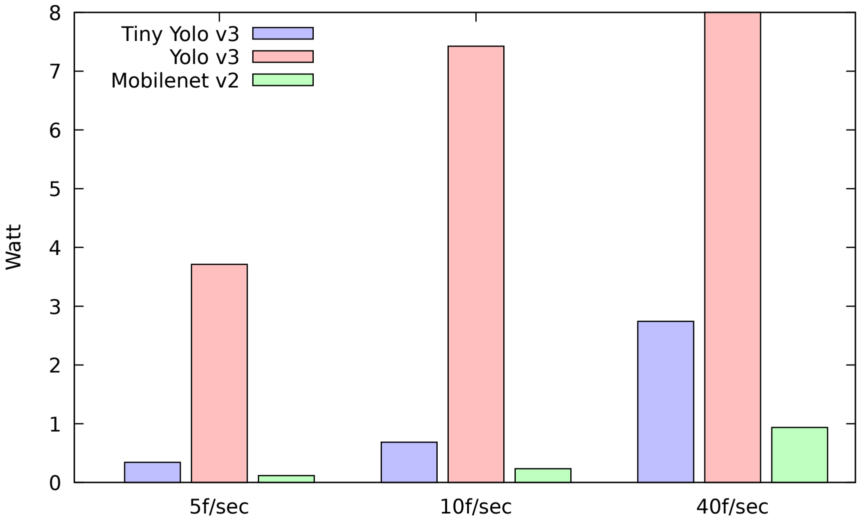
Charles Edison Tripp **andothers**. *Measuring the Energy Consumption and Efficiency of Deep Neural Networks: An Empirical Analysis and Design Recommendations*. 2024. arXiv: 2403.08151 [cs.LG]

Power Consumption in Inference



VGG16 applied to the ImageNet data set based on published papers.

Power Consumption in Inference



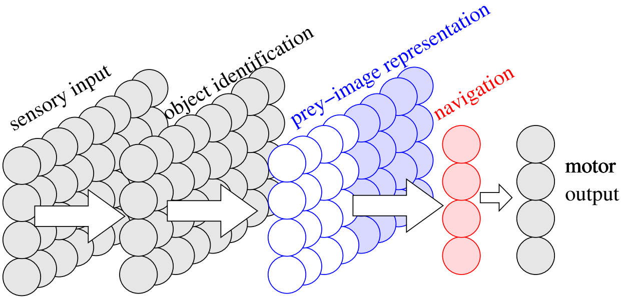
Object detection on the NCS2 platform; own measurements.

KUBOTA
U-10

Dragonfly

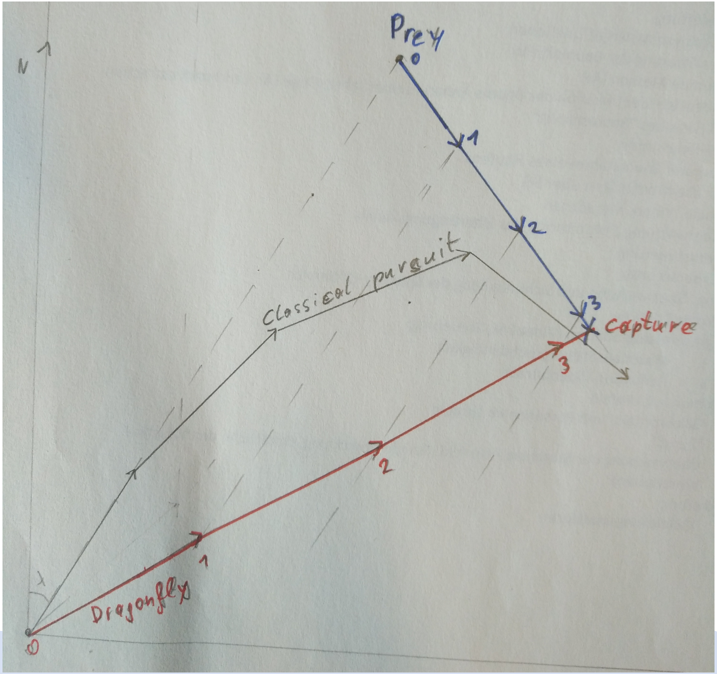
- Brain volume: 1 mm^3
- Weight: 1 mg
- Number of neurons: 1 Million
- Power consumption: 2–8 mW
- 200 frames/second
- 95 % hunting success rate
- Reaction time: 50 ms

Dragonfly



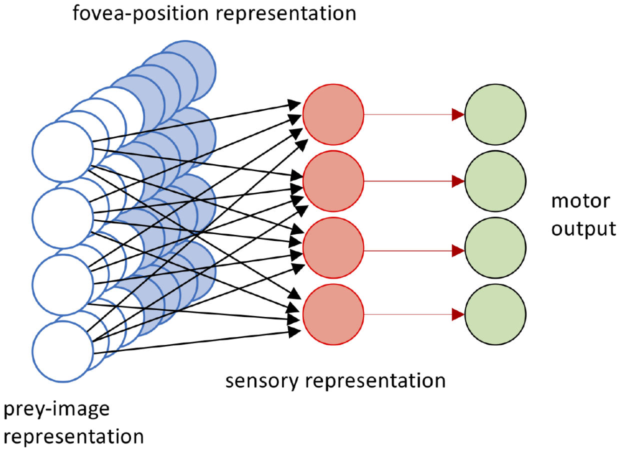
- 50 ms reaction time
- One neuron needs 10 ms to integrate inputs
- 10 ms for the photo detectors and the prey identification
- 5 ms for the muscles to produce force
- leaves 35 ms for route planning
- $\Rightarrow$ maximum 5 layer NN

Proportional Navigation



Dragonfly

- Proportional navigation has been implemented in a 3-layer NN;



Frances S. Chance. "Interception from a Dragonfly Neural Network Model". In: *International Conference on Neuromorphic Systems 2020 (ICONS)*. Oak Ridge, TN, USA. ACM, New York, NY, USA, **july** 2020

In terms of energy efficiency we are about 3-9 orders of magnitude from what is possible and feasible.

What is Special About “Embedded”?

Resource limitations

	Embedded	Server farm
Computation [flop]	$30–1800 \cdot 10^{12}$	$86 \cdot 10^{18}$
Memory [bit]	10^{10}	10^{15}
Power [W]	5–100	$10^3–10^6$
Energy [Wh]	48–1000	$200 \cdot 10^6$

Computation Embedded refers to an Nvidia Jetson Nano running 1 min and 1 hour, respectively.
Computation server refers to the computation needed for the 40 day experiment with AlphaGo Zero
Energy embedded refers to a mobile phone and to a car battery, respectively.
Energy server refers to the 40 day experiment for AlphaGo Zero.
Figures for the **human brain** are taken from table 3.1 in “Of Brains and Computers”, Rabaey 2022.

What is Special About "Embedded"?

Resource limitations

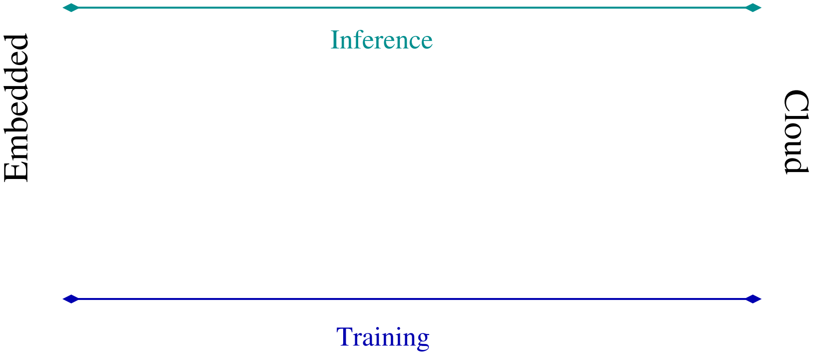
	Embedded	Server farm	Human brain
Computation [flop]	$30\text{–}1800 \cdot 10^{12}$	$86 \cdot 10^{18}$	$10^{16}\text{–}10^{21}$
Memory [bit]	10^{10}	10^{15}	$10^{13}\text{–}10^{16}$
Power [W]	5–100	$10^{3}\text{–}10^{6}$	20
Energy [Wh]	48–1000	$200 \cdot 10^{6}$	2400–12 000

Computation Embedded refers to an Nvidia Jetson Nano running 1 min and 1 hour, respectively.
Computation server refers to the computation needed for the 40 day experiment with AlphaGo Zero
Energy embedded refers to a mobile phone and to a car battery, respectively.
Energy server refers to the 40 day experiment for AlphaGo Zero.
Figures for the **human brain** are taken from table 3.1 in "Of Brains and Computers", Rabaey 2022.

Case for Embedded ML

- Embedded inference:
  - More energy efficient
  - Bandwidth constraints
  - Latency constraints
  - Not always on-line and connected to a cloud server
  - Security
  - Privacy
- Embedded continuous learning:
  - Customization and specialization
  - Security
  - Privacy

DNNs: Embedded and the Cloud



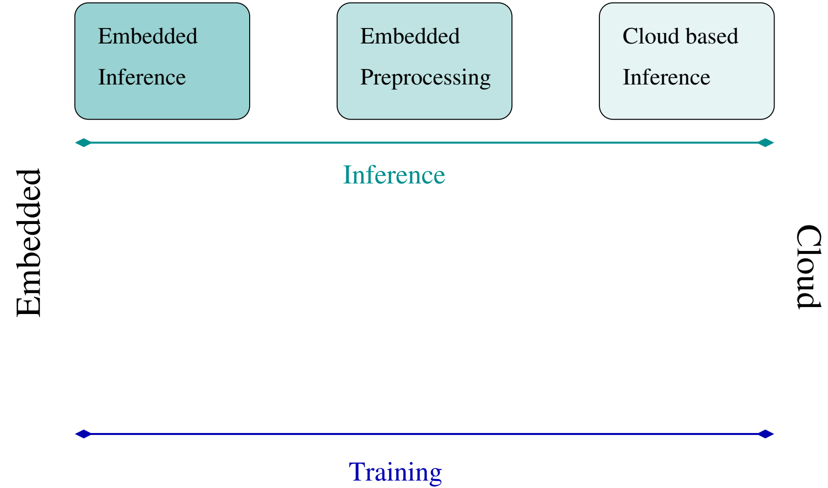
Embedded

Inference

Training

Cloud

DNNs: Embedded and the Cloud



Embedded Inference

Embedded Preprocessing

Cloud based Inference

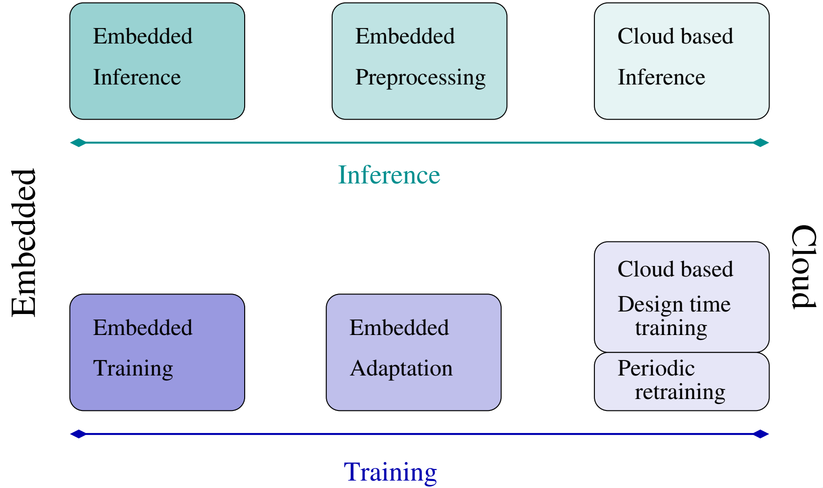
Inference

Embedded

Cloud

Training

DNNs: Embedded and the Cloud



Embedded Inference

Embedded Preprocessing

Cloud based Inference

Inference

Embedded

Cloud

Embedded Training

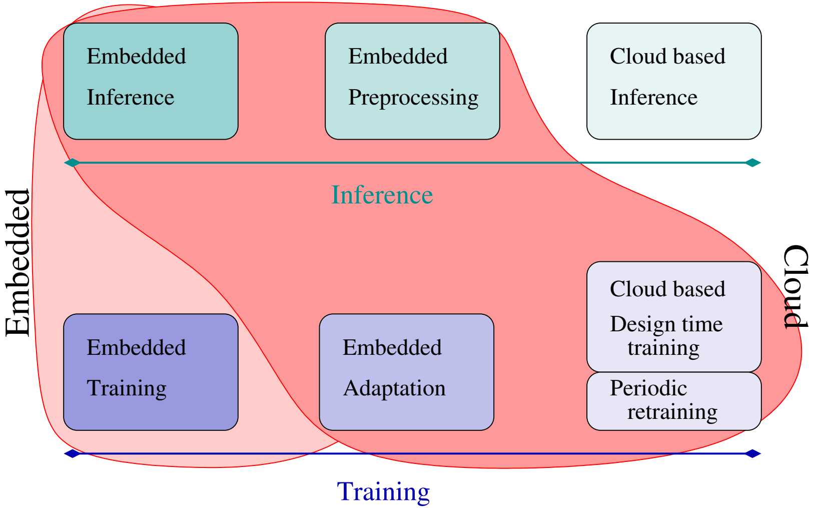
Embedded Adaptation

Cloud based Design time training

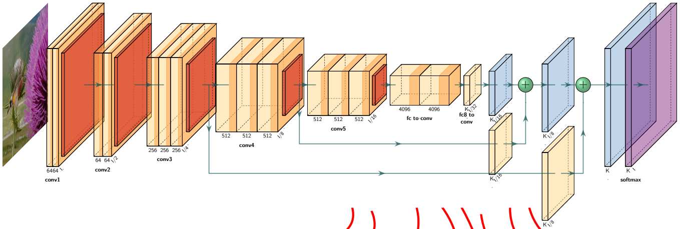
Periodic retraining

Training

DNNs: Embedded and the Cloud



Embedded Machine Learning



DNN Choices

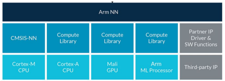
Convolutional layers
Filter kernels
Number of filters
Pooling layers
Filter shape
Stride
Fully connected layer
Number of layers
Regularization
etc.

Mapping Choices

Neuron pruning
Data type selection
Approximation
Retraining
Connection pruning
Weight sparsifying
Regularization
etc.

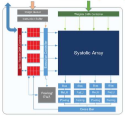
PCIe Gen3 x 8
Flash
PMIC
FPGA
DLA IP
(FP16/11/9)
DDR
DDR

Intel® Vision Products with Intel® Arria® 10 FPGA



ARM NN

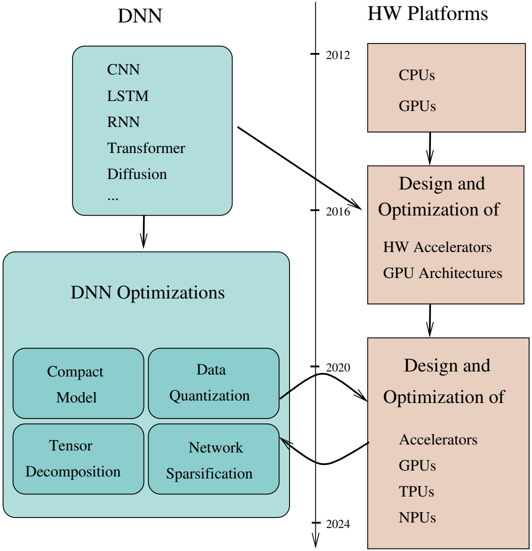
Xilinx DNN Processor (xDNN)



Nvidia Turing

Platform Choices

Platform Selection
Reconfiguration
Batch processing
Deep pipelining
Resource reuse
Hierarchical control
Processing unit selection
Memory allocation
Memory reuse
etc.



DNN

CNN
LSTM
RNN
Transformer
Diffusion
...

DNN Optimizations

Compact Model

Data Quantization

Tensor Decomposition

Network Sparsification

HW Platforms

CPUs
GPUs

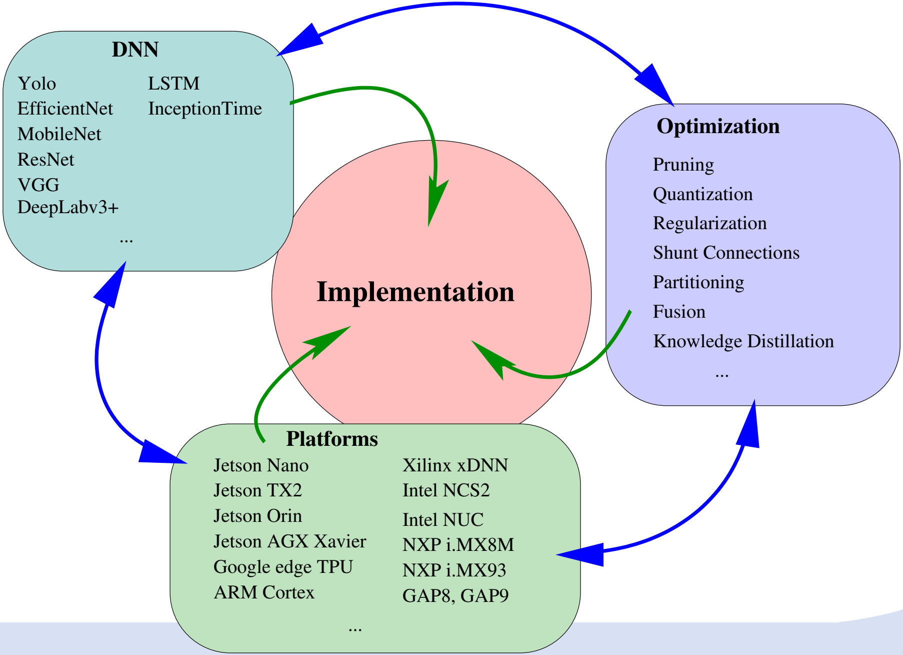
Design and Optimization of

HW Accelerators
GPU Architectures

Design and Optimization of

Accelerators
GPUs
TPUs
NPUs

HW–Algorithm Co–evolution



DNN

Yolo
EfficientNet
MobileNet
ResNet
VGG
DeepLabv3+
...

LSTM
InceptionTime

Implementation

Optimization

Pruning
Quantization
Regularization
Shunt Connections
Partitioning
Fusion
Knowledge Distillation
...

Platforms

Jetson Nano
Jetson TX2
Jetson Orin
Jetson AGX Xavier
Google edge TPU
ARM Cortex

Xilinx xDNN
Intel NCS2
Intel NUC
NXP i.MX8M
NXP i.MX93
GAP8, GAP9
...

Outline

1. Challenge and Motivation
2. Convolutional Neural Networks
   - HW Friendly Optimizations
   - CNN Accelerator Architectures
3. Quantization
4. Profiling
5. Estimation
   - Enhanced Roofline Model
   - Step-wise Linear Model
6. Traffic Light Controller Case Study
7. Summary

Convolutional Neural Networks

Outline

1. Challenge and Motivation
2. Convolutional Neural Networks
   - HW Friendly Optimizations
   - CNN Accelerator Architectures
3. Quantization
4. Profiling
5. Estimation
   - Enhanced Roofline Model
   - Step-wise Linear Model
6. Traffic Light Controller Case Study
7. Summary

Optimization Categories

1. Minimize number of operations to be performed;

Optimization Categories

1. Minimize number of operations to be performed;
   - Loop reordering, unrolling, pipelining

Optimization Categories

1. Minimize number of operations to be performed;
    - Loop reordering, unrolling, pipelining
    - Tiling

Optimization Categories

1. Minimize number of operations to be performed;
    - Loop reordering, unrolling, pipelining
    - Tiling
    - Batching

Optimization Categories

1. Minimize number of operations to be performed;
    - Loop reordering, unrolling, pipelining
    - Tiling
    - Batching
2. Simplify each operation;

Optimization Categories

1. Minimize number of operations to be performed;
    - Loop reordering, unrolling, pipelining
    - Tiling
    - Batching
2. Simplify each operation;
    - Data type optimization

Optimization Categories

1. Minimize number of operations to be performed;
    - Loop reordering, unrolling, pipelining
    - Tiling
    - Batching
2. Simplify each operation;
    - Data type optimization
    - Binarized CNNs

Optimization Categories

1. Minimize number of operations to be performed;
   - Loop reordering, unrolling, pipelining
   - Tiling
   - Batching
2. Simplify each operation;
   - Data type optimization
   - Binarized CNNs
3. Execute the operations as efficient as possible.

Optimization Categories

1. Minimize number of operations to be performed;
    - Loop reordering, unrolling, pipelining
    - Tiling
    - Batching
2. Simplify each operation;
    - Data type optimization
    - Binarized CNNs
3. Execute the operations as efficient as possible.
    - Pruning

Optimization Categories

1. Minimize number of operations to be performed;
   - Loop reordering, unrolling, pipelining
   - Tiling
   - Batching
2. Simplify each operation;
   - Data type optimization
   - Binarized CNNs
3. Execute the operations as efficient as possible.
   - Pruning
   - Layer elimination

Optimization Categories

1. Minimize number of operations to be performed;
   - Loop reordering, unrolling, pipelining
   - Tiling
   - Batching
2. Simplify each operation;
   - Data type optimization
   - Binarized CNNs
3. Execute the operations as efficient as possible.
   - Pruning
   - Layer elimination
   - Layer simplification

Loop Optimizations

Convolution layer algorithm:

```
for (to=0; t<M; to++) {              // output feature map
  for (ti=0; t<N; ti++) {            // input feature map
    for (row=0; row<R; row++) {      // row
      for (col=0; col<C; col++) {    // column
         for (i=0; i<K; i++) {       // filter
           for (j=0; j<K; j++) {
             Ofmap[to][row][col]
               += W[to][ti][i][j] * Ifmap[ti][S*row+i][S*col+j];
}}}}}}
```

M ... number of output feature maps
N ... number of input feature maps
R ... number of rows
C ... number of columns

K ... filter kernel size
S ... stride
W ... weight matrix

Loop Optimizations

Loop reordering to improve cache efficiency;

Loop unrolling to improve parallelism;

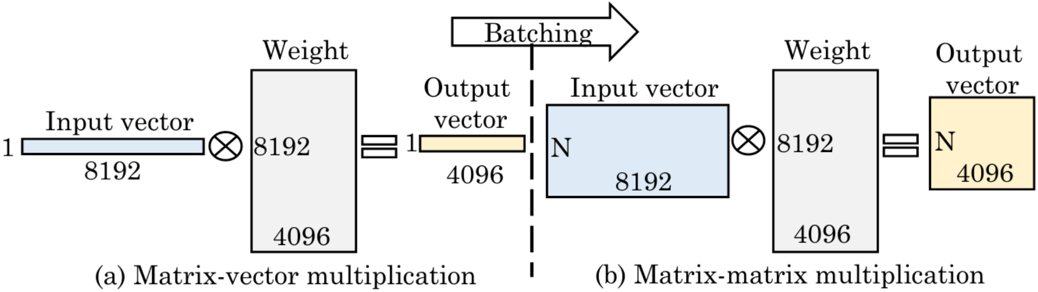
Loop pipelining to improve parallelism.

Loop Tiling

```
for (to=0; t<M; to++)   {         // output feature map
  for (ti=0; t<N; ti++) {         // input feature map
    for (row=0; row<R; row+=Tr) { // tiled row loop
      for (col=0; col<C; col++) { // column
        for (trr=row; trr<min(R,row+Tr); trr++) {
          for (i=0; i<K; i++) {   // filter
            for (i=0; i<K; i++) {
              Ofmap[to][trr][col]
                += W[to][ti][i][j] * Ifmap[ti][S*trr+i][S*col+j];
            }}}}}}}
```

For efficient use of caches.

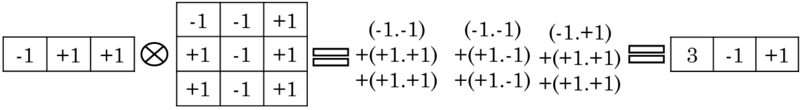
Batching



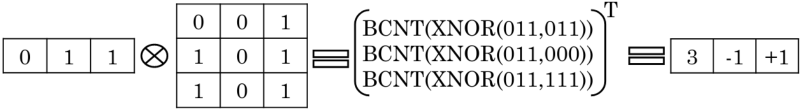
- Reuse of weights;
- Improves throughput;
- Increases latency.

Binarized CNNs (BNN)

- Weights and internal computation are represented as 1b binary numbers;
- Instead of MAC operations, BNNs use XOR and bit-count;
- Attractive for HW and FPGAs



(a) An example of binarized MM



(b) Binarized MM using XNOR and BCNT. -1 is represented using 0.

BCNT= OneCount-ZeroCount

IN	Computation	OUT
000	-1-1-1= -3	101
001	-1-1+1 = -1	111
010	-1+1-1= -1	111
011	-1+1+1= +1	001
100	+1-1-1= -1	111
101	+1-1+1= +1	001
110	+1+1-1= +1	001
111	+1+1+1= +3	011

(c) BCNT using a lookup table (OUT is in 2's complement forr

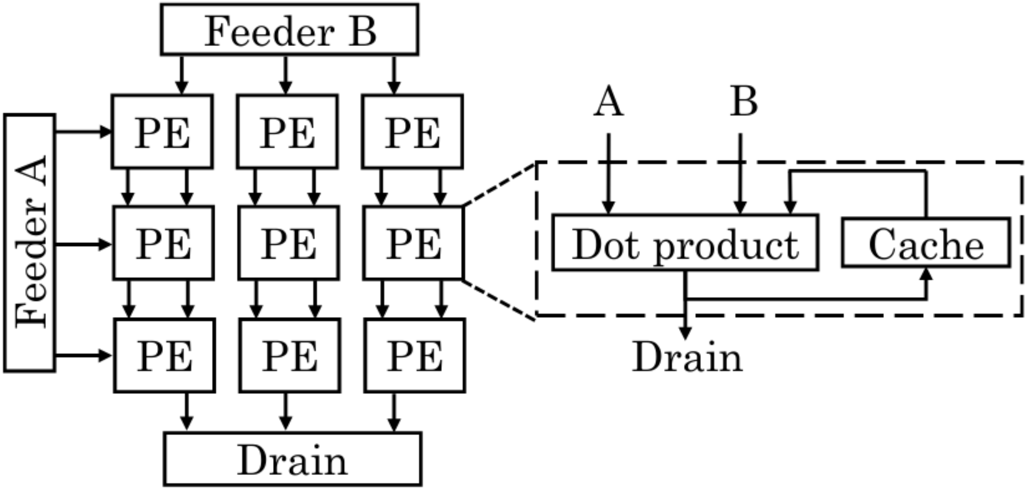
Outline

1. Challenge and Motivation
2. Convolutional Neural Networks
   - HW Friendly Optimizations
   - CNN Accelerator Architectures
3. Quantization
4. Profiling
5. Estimation
   - Enhanced Roofline Model
   - Step-wise Linear Model
6. Traffic Light Controller Case Study
7. Summary

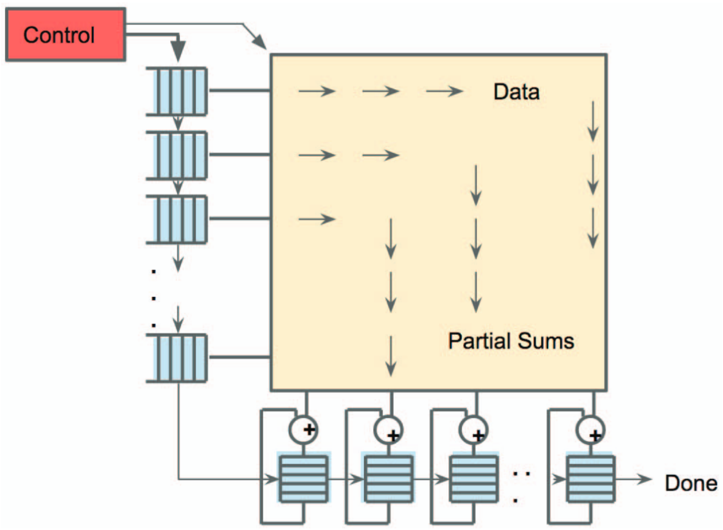
CNN Accelerator Architectures

- Systolic array architecture
- In-memory computing

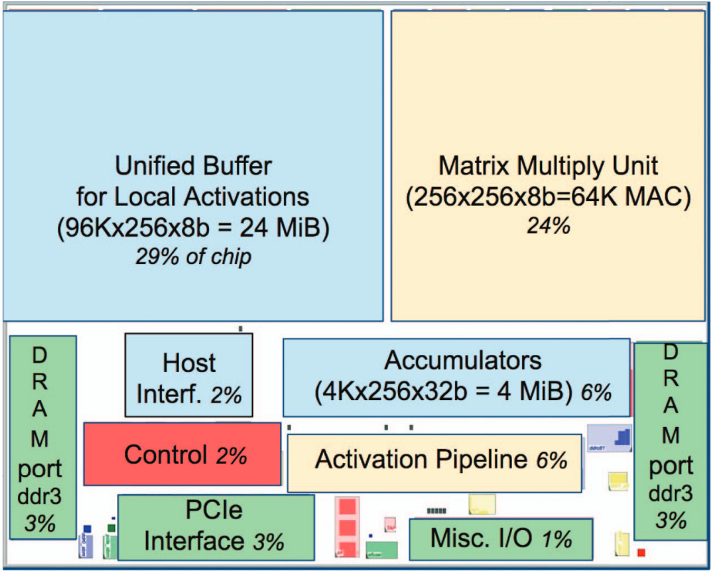
Systolic Arrays



Tensor Flow Unit (TPU)



Tensor Flow Unit (TPU)



Unified Buffer
for Local Activations
(96Kx256x8b = 24 MiB)
29% of chip

Matrix Multiply Unit
(256x256x8b=64K MAC)
24%

D
R
A
M
port
ddr3
3%

Host
Interf. *2%*

Accumulators
(4Kx256x32b = 4 MiB) *6%*

D
R
A
M
port
ddr3
3%

Control *2%*

Activation Pipeline *6%*

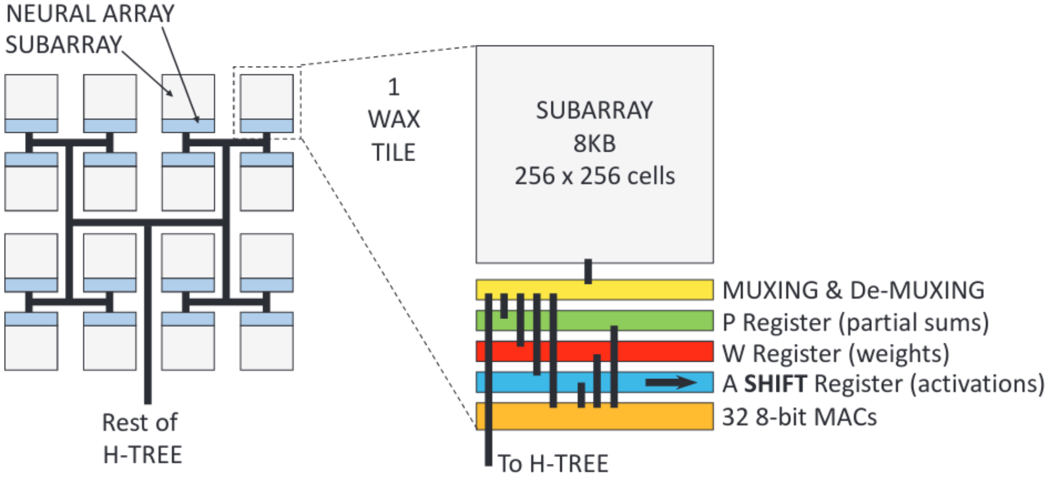
PCIe
Interface *3%*

Misc. I/O *1%*

In-memory Computing

- Storage capacity and memory access bandwidth and latency dominate DNNs.
- Avoid moving data.
- Distribute the MAC units in the memory architecture.

Wire Aware Accelerator (WAX)



Sumanth Gudaparthi, Surya Narayanan, Rajeev Balasubramonian, Edouard Giacomin, Hari Kambalasubramanyam **and** Pierre-Emmanuel Gaillardon. “Wire-Aware Architecture and Dataflow for CNN Accelerators”. In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO ’52. Columbus, OH, USA: Association for Computing Machinery, 2019

Outline

1. Challenge and Motivation
2. Convolutional Neural Networks
   - HW Friendly Optimizations
   - CNN Accelerator Architectures
3. Quantization
4. Profiling
5. Estimation
   - Enhanced Roofline Model
   - Step-wise Linear Model
6. Traffic Light Controller Case Study
7. Summary

QUANTIZATION

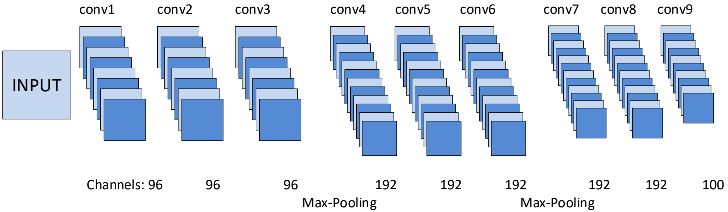
Quantization - Regularization

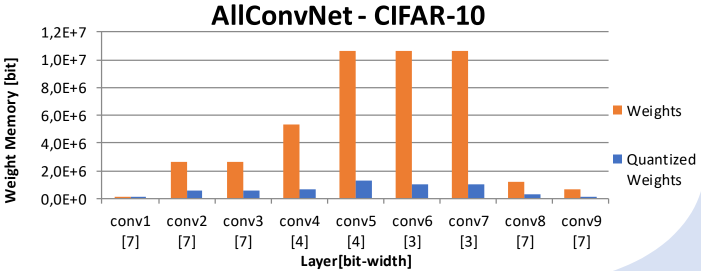
- Using small bit width for weights saves memory, bandwidth and computation;
- Bit width can be different for different layers of the DNN;
- Quantization scheme: Dynamic fixed point, power of 2;
- Retraining after quantization recovers accuracy losses: Regularization;

Matthias Wess, Sai Manoj Pudukotai Dinakarrao **and** Axel Jantsch. “Weighted Quantization-Regularization in DNNs for Weight Memory Minimization towards HW Implementation”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.10 (**october** 2018)

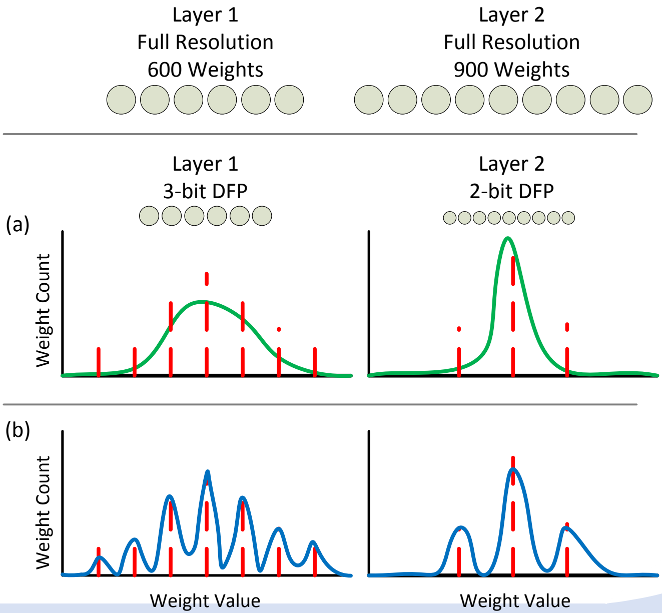
Quantization - Motivation

- DNN quantization
  - Reduces data movement
  - Reduces logic energy
- Layerwise bit-width optimization





Quantization - Motivation



Layer 1
Full Resolution
600 Weights

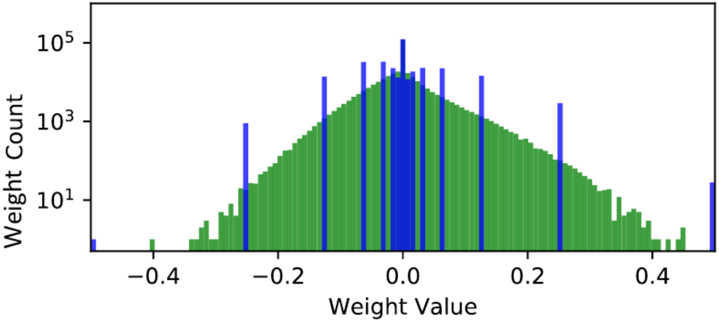
Layer 2
Full Resolution
900 Weights

Layer 1
3-bit DFP

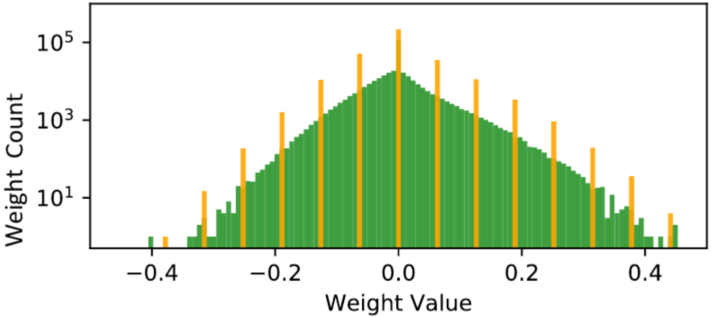
Layer 2
2-bit DFP

Quantization Schemes

Power of Two



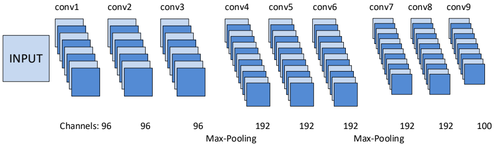
Dynamic Fixed Point



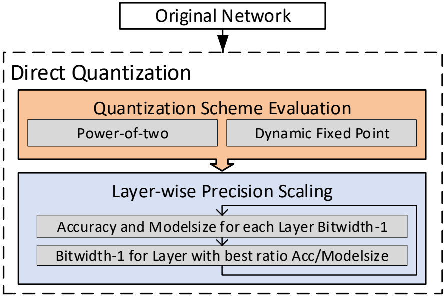
Quantization Flow

Original Network

- Original Network



Quantization Flow



Original Network

Direct Quantization

Quantization Scheme Evaluation

Power-of-two

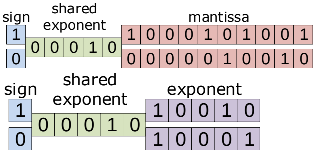
Dynamic Fixed Point

Layer-wise Precision Scaling

Accuracy and Modelsize for each Layer Bitwidth-1

Bitwidth-1 for Layer with best ratio Acc/Modelsize

- Original Network
- Direct Quantization
  - Quantization Schemes



sign | shared exponent | mantissa

1 | 0 0 0 1 0 | 1 0 0 0 1 0 1 0 0 1

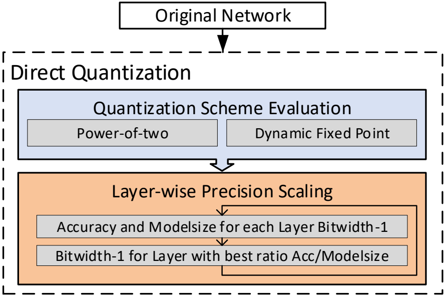
0 | 0 0 0 1 0 | 0 0 0 0 0 1 0 0 1 0

sign | shared exponent | exponent

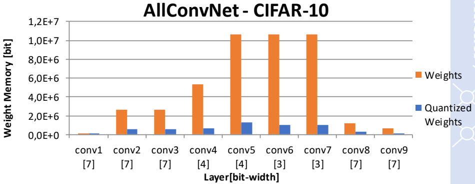
1 | 0 0 0 1 0 | 1 0 0 1 0

0 | 0 0 0 1 0 | 1 0 0 0 1

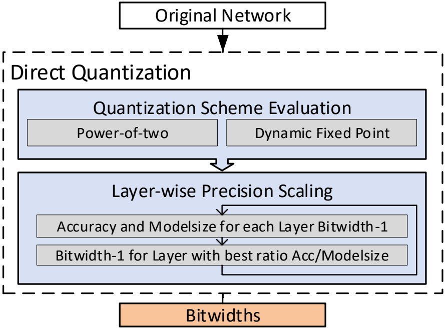
Quantization Flow


Original Network
Direct Quantization
Quantization Scheme Evaluation
Power-of-two
Dynamic Fixed Point
Layer-wise Precision Scaling
Accuracy and Modelsize for each Layer Bitwidth-1
Bitwidth-1 for Layer with best ratio Acc/Modelsize

- Original Network
- Direct Quantization
  - Quantization Schemes
  - Layer-wise Precision Scaling



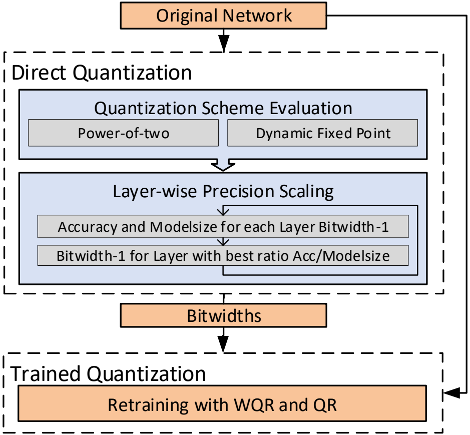
Quantization Flow


Original Network
Direct Quantization
Quantization Scheme Evaluation
Power-of-two
Dynamic Fixed Point
Layer-wise Precision Scaling
Accuracy and Modelsize for each Layer Bitwidth-1
Bitwidth-1 for Layer with best ratio Acc/Modelsize
Bitwidths

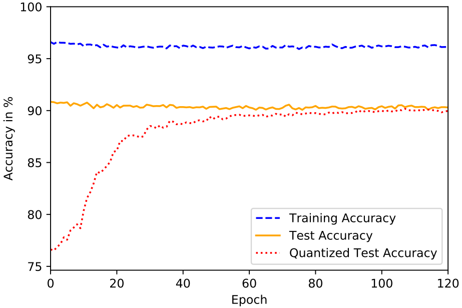
- Original Network
- Direct Quantization
  - Quantization Schemes
  - Layer-wise Precision Scaling
- Bitwidths

$$[7\ 7\ 7\ 4\ 4\ 3\ 3\ 7\ 7]$$

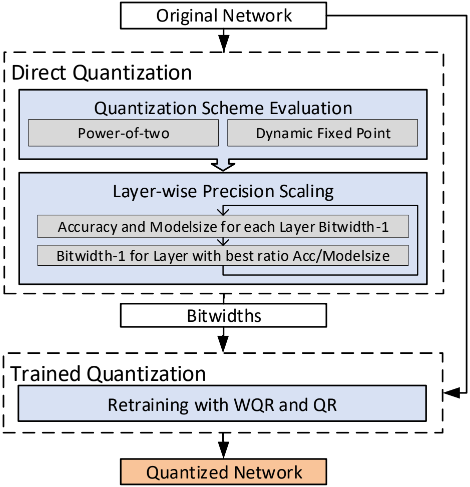
Quantization Flow


Original Network
Direct Quantization
Quantization Scheme Evaluation
Power-of-two
Dynamic Fixed Point
Layer-wise Precision Scaling
Accuracy and Modelsize for each Layer Bitwidth-1
Bitwidth-1 for Layer with best ratio Acc/Modelsize
Bitwidths
Trained Quantization
Retraining with WQR and QR

- Original Network
- Direct Quantization
  - Quantization Schemes
  - Layer-wise Precision Scaling
- Bitwidths
- Trained Quantization



Quantization Flow



Original Network

Direct Quantization

Quantization Scheme Evaluation

Power-of-two

Dynamic Fixed Point

Layer-wise Precision Scaling

Accuracy and Modelsize for each Layer Bitwidth-1

Bitwidth-1 for Layer with best ratio Acc/Modelsize

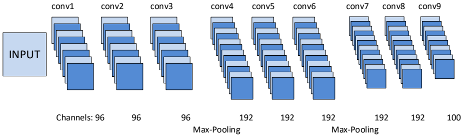
Bitwidths

Trained Quantization

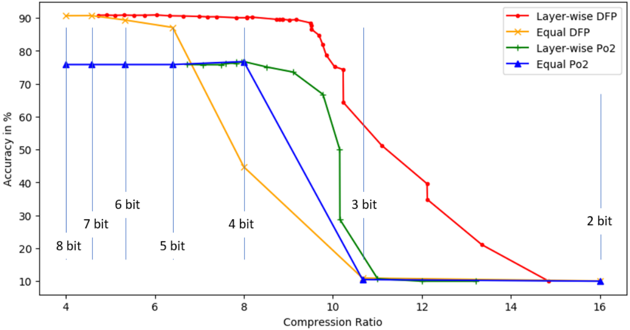
Retraining with WQR and QR

Quantized Network

- Original Network
- Direct Quantization
  - Quantization Schemes
  - Layer-wise Precision Scaling
- Bitwidths
- Trained Quantization
- Quantized Network



Layer-wise Precision Scaling



Layerwise DFP:

- At 8-bit $\Delta Acc \approx 0$
- Accuracy drops at 4-bit

Layerwise Po2:

- max(Accuracy) at $>$3-bit
- Accuracy drops at 3-bit

Quantization-Regularization

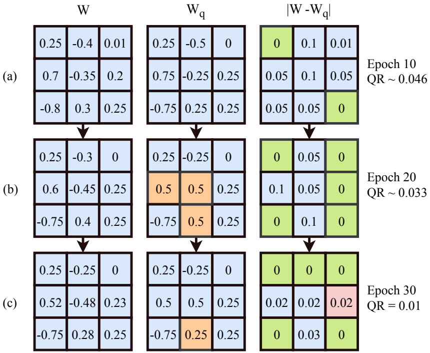
$$QR = \sum_{n}^{N} \sum_{i}^{\mathrm{card}(W_n)} \frac{|W_{n_i} - Wq_{n_i}|}{\max(Q_n) * \mathrm{card}(W_n)}$$

$$\text{Modified Loss} = \text{Loss} + \lambda_1 * QR$$

Quantization-Regularization

$$QR = \sum_n^N \sum_i^{\text{card}(W_n)} \frac{|W_{n_i} - Wq_{n_i}|}{\max(Q_n) * \text{card}(W_n)}$$

$$\text{Modified Loss} = \text{Loss} + \lambda_1 * QR$$

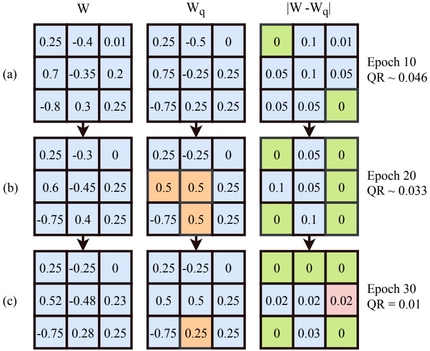


	W			W_q			|W -W_q|			
(a)	0.25	-0.4	0.01	0.25	-0.5	0	0	0.1	0.01	Epoch 10
	0.7	-0.35	0.2	0.75	-0.25	0.25	0.05	0.1	0.05	QR ~ 0.046
	-0.8	0.3	0.25	-0.75	0.25	0.25	0.05	0.05	0	
(b)	0.25	-0.3	0	0.25	-0.25	0	0	0.05	0	Epoch 20
	0.6	-0.45	0.25	0.5	0.5	0.25	0.1	0.05	0	QR ~ 0.033
	-0.75	0.4	0.25	-0.75	0.5	0.25	0	0.1	0	
(c)	0.25	-0.25	0	0.25	-0.25	0	0	0	0	Epoch 30
	0.52	-0.48	0.23	0.5	0.5	0.25	0.02	0.02	0.02	QR = 0.01
	-0.75	0.28	0.25	-0.75	0.25	0.25	0	0.03	0	

Quantization-Regularization

$$QR = \sum_{n}^{N} \sum_{i}^{\text{card}(W_n)} \frac{|W_{n_i} - Wq_{n_i}|}{\max(Q_n) * \text{card}(W_n)}$$

$$\text{Modified Loss} = \text{Loss} + \lambda_1 * QR$$



(a) Epoch 10 QR ~ 0.046

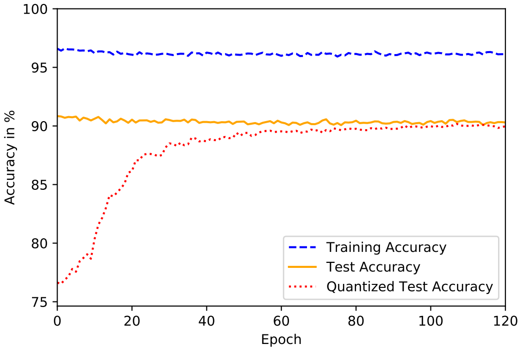
W			W_q			|W -W_q|		
0.25	-0.4	0.01	0.25	-0.5	0	0	0.1	0.01
0.7	-0.35	0.2	0.75	-0.25	0.25	0.05	0.1	0.05
-0.8	0.3	0.25	-0.75	0.25	0.25	0.05	0.05	0

(b) Epoch 20 QR ~ 0.033

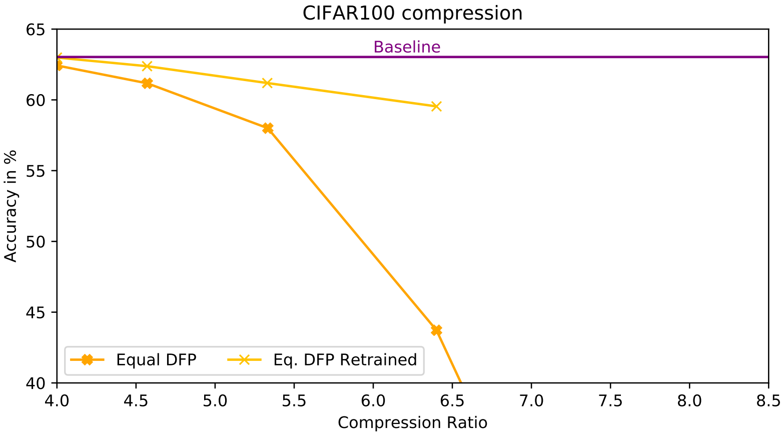
W			W_q			|W -W_q|		
0.25	-0.3	0	0.25	-0.25	0	0	0.05	0
0.6	-0.45	0.25	0.5	0.5	0.25	0.1	0.05	0
-0.75	0.4	0.25	-0.75	0.5	0.25	0	0.1	0

(c) Epoch 30 QR = 0.01

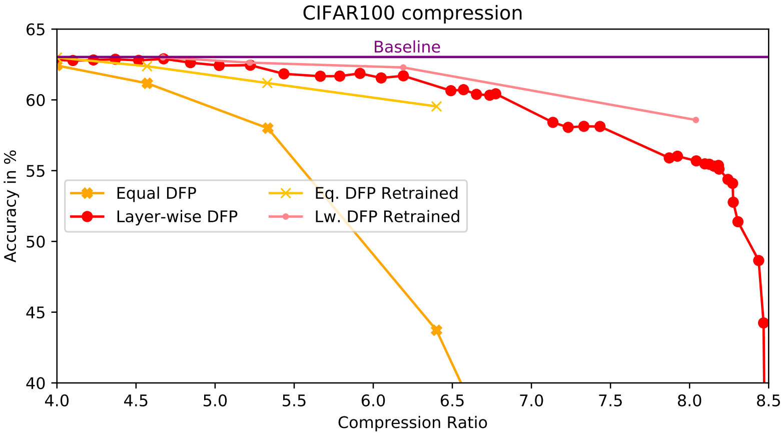
W			W_q			|W -W_q|		
0.25	-0.25	0	0.25	-0.25	0	0	0	0
0.52	-0.48	0.23	0.5	0.5	0.25	0.02	0.02	0.02
-0.75	0.28	0.25	-0.75	0.25	0.25	0	0.03	0



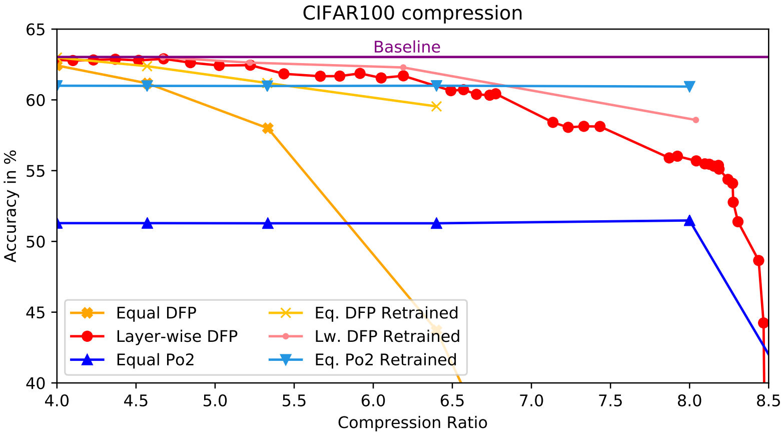
CIFAR-100


CIFAR100 compression
Baseline
Accuracy in %
65
60
55
50
45
40
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
Compression Ratio
Equal DFP
Eq. DFP Retrained

CIFAR-100


CIFAR100 compression
Baseline
Equal DFP
Layer-wise DFP
Eq. DFP Retrained
Lw. DFP Retrained
Accuracy in %
65
60
55
50
45
40
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
Compression Ratio

CIFAR-100



CIFAR100 compression
Baseline
Accuracy in %
65
60
55
50
45
40
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
Compression Ratio
Equal DFP
Layer-wise DFP
Equal Po2
Eq. DFP Retrained
Lw. DFP Retrained
Eq. Po2 Retrained

Quantization - Conclusion

- Different layers require different precision;

Quantization - Conclusion

- Different layers require different precision;
- Dynamic Fixed Point is an effective alternative to integer or floating point representation;
  - DFP is usually preferable to Po2
  - DFP does not always require retraining
  - Po2 can be more efficient in some low bit-width cases
  - Po2 requires only shift operations

Quantization - Conclusion

- Different layers require different precision;
- Dynamic Fixed Point is an effective alternative to integer or floating point representation;
  - DFP is usually preferable to Po2
  - DFP does not always require retraining
  - Po2 can be more efficient in some low bit-width cases
  - Po2 requires only shift operations
- Retraining adjusts the network to the available weight values;
  - QR can alleviate accuracy degradation induced by quantization
  - QR is easy to implement and can be combined with other techniques

Quantization - Conclusion

- Different layers require different precision;
- Dynamic Fixed Point is an effective alternative to integer or floating point representation;
    - DFP is usually preferable to Po2
    - DFP does not always require retraining
    - Po2 can be more efficient in some low bit-width cases
    - Po2 requires only shift operations
- Retraining adjusts the network to the available weight values;
    - QR can alleviate accuracy degradation induced by quantization
    - QR is easy to implement and can be combined with other techniques
- Weight memory reduction of 4-8x is common;

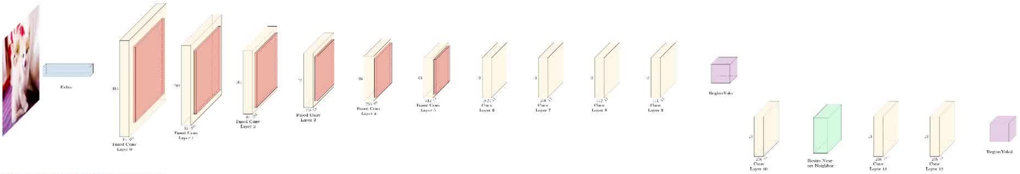
Outline

1. Challenge and Motivation
2. Convolutional Neural Networks
   - HW Friendly Optimizations
   - CNN Accelerator Architectures
3. Quantization
4. **Profiling**
5. Estimation
   - Enhanced Roofline Model
   - Step-wise Linear Model
6. Traffic Light Controller Case Study
7. Summary

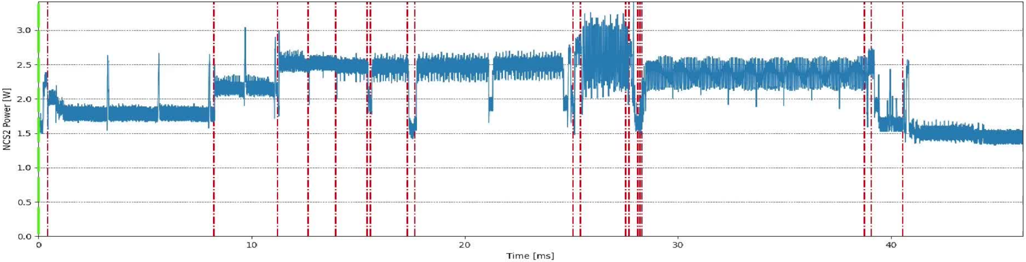
PROFILING

Power and Performance Profiling

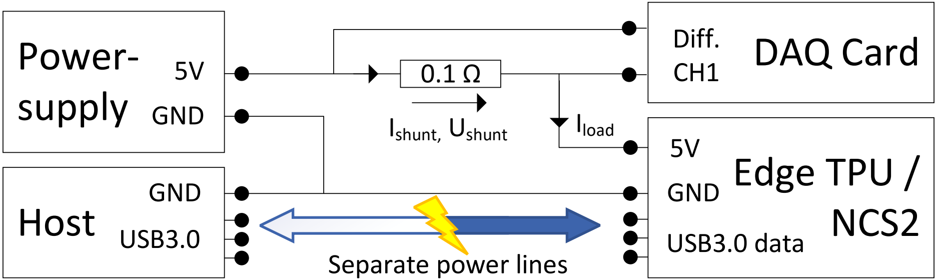
Yolov3-tiny power profile on NCS2

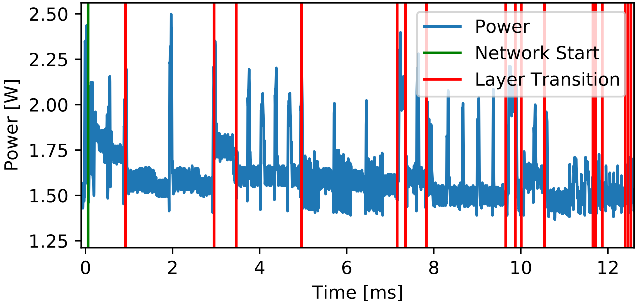


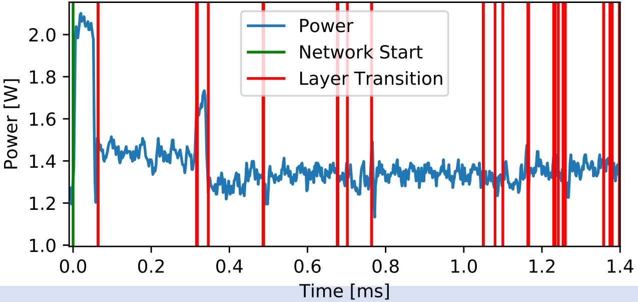
00.000 ms



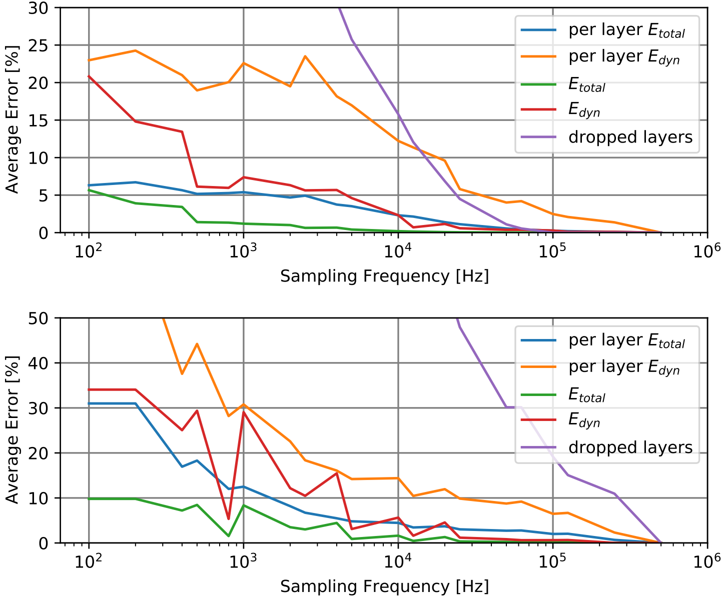
Experimental Setup





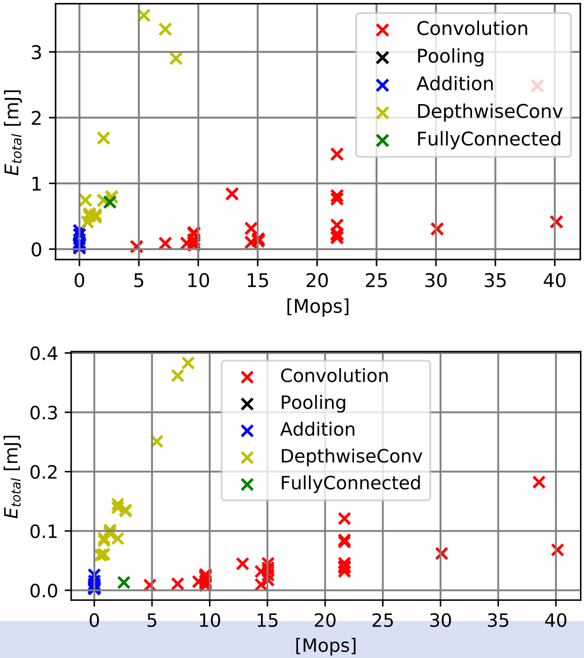


MobileNetV2 on NCS2 and Coral Edge TPU



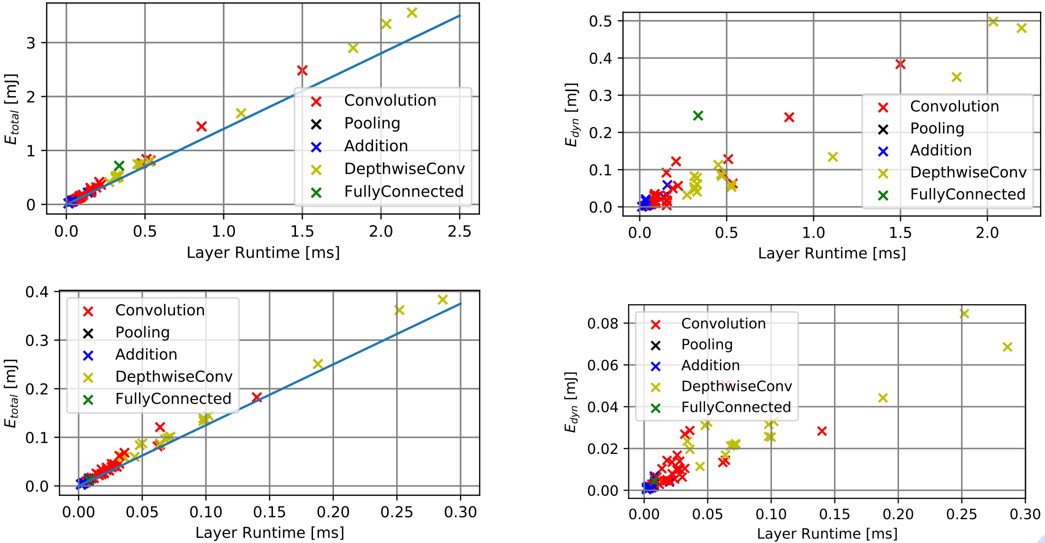
MobileNetV2 on NCS2 and Coral Edge TPU

The error in % with respect to 500 kHz sampling frequency.



MobileNetV2 on NCS2 and Coral Edge TPU

Energy versus number of operations.



MobileNetV2 on NCS2 and Coral Edge TPU; Energy versus latency.

Profiling Results

HW	Network	*nireq*	F_{thr}(fps)	T_{lat}(ms)	P (mW)	E_{total}(mJ)	E_{base}(mJ)	E_{dyn}(mJ)	$E\,/\,Gop$(mJ)	$E\,/\,Mpar$(mJ)
NCS2	Tiny YOLOv3	1	21.2	41	2165	101.93	65.91	36.02	18.32	11.52
		2	35.3	52	2670	75.55	39.61	35.94	13.58	8.54
	5.6 Gop	3	43.1	46	2995	69.42	32.45	36.97	12.47	7.85
	8.8 Mpar	4	43.1	44	2954	68.54	32.48	36.06	12.32	7.75
	YOLOv3	1	2.6	363	2505	960.92	537.04	423.88	14.69	15.61
		2	4.4	400	3413	769.61	315.69	453.92	11.76	12.50
	65.8 Gop	3	4.7	425	3615	764.89	296.22	468.67	11.69	12.42
	61.6 Mpar	4	4.9	390	3604	742.50	288.43	454.07	11.35	12.06
	MobileNetV2	1	49.3	21	1806	36.60	28.37	8.23	60.84	10.55
		2	87.2	23	2118	24.29	16.06	8.23	40.38	7.00
	0.6 Gop	3	90.4	31	2164	23.95	15.49	8.46	39.81	6.90
	3.4 Mpar	4	92.4	53	2162	23.39	15.15	8.24	38.88	6.74

HW	Network	*Freq*	F_{thr}(fps)	T_{lat}(ms)	P (mW)	E_{total}(mJ)	E_{base}(mJ)	E_{dyn}(mJ)	$E\,/\,Gop$(mJ)	$E\,/\,Mpar$(mJ)
Edge TPU	Tiny YOLOv3	std	46.3	22.3	1407	30.40	22.28	8.12	5.46	3.44
		max	51.0	19.6	1528	29.95	20.21	9.73	5.38	3.39
	YOLOv3	std	6.3	158.3	1519	240.50	163.27	77.23	3.68	3.91
		max	7.0	142.0	1657	235.36	147.29	88.06	3.60	3.82
	MobileNetV2	std	331.3	3.0	1422	4.29	3.11	1.18	7.13	1.24
		max	512.3	1.9	1658	3.23	2.02	1.21	5.37	0.93

Power and Performance Profiling

- NCS2, Edge TPU and Nvidia platforms
- Detailed, per layer latency and power profiling
- Number of operations is a poor predictor for latency and energy
- Latency and energy usage correlate fairly well
- Hardware setting have significant influence
- 100 kHz sampling frequency is required for 5 % accuracy

Matthias Wess, Dominik Dallinger, Daniel Schnöll, Matthias Bittner, Maximilian Götzinger **and** Axel Jantsch. "Energy Profiling of DNN Accelerators". In: *Proceedings of the 26th Euromicro Conference on Digital System Design (DSD)*. Durres, Albania, **september** 2023

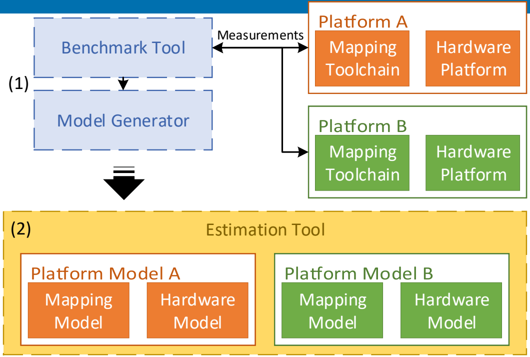
Outline

1. Challenge and Motivation
2. Convolutional Neural Networks
   - HW Friendly Optimizations
   - CNN Accelerator Architectures
3. Quantization
4. Profiling
5. Estimation
   - Enhanced Roofline Model
   - Step-wise Linear Model
6. Traffic Light Controller Case Study
7. Summary

ESTIMATION

Estimation

- Two leading performance estimation tools: ANNETTE and Blackthorn
- For NCS2, Xilinx FPGA, and Jetson
- Combine analytic, statistical model and partial measurements



Matthias Wess, Marco Ivanov, Christian Unger, Anvesh Nookala, Alexander Wendt **and** Axel Jantsch. "ANNETTE: Accurate Neural Network Execution Time Estimation With Stacked Models". In: *IEEE Access* 9 (2021), **pages** 3545–3556

Martin Lechner **and** Axel Jantsch. "Blackthorn: Latency Estimation Framework for CNNs on Embedded Nvidia Platforms". In: *IEEE Access* (2021)

Matthias Wess, Daniel Schnöll, Dominik Dallinger, Matthias Bittner **and** Axel Jantsch. "Conformal Prediction based Confidence for Latency Estimation of DNN Accelerators: A Black-box Approach". In: *IEEE Access* (2024)

Outline

1. Challenge and Motivation
2. Convolutional Neural Networks
   - HW Friendly Optimizations
   - CNN Accelerator Architectures
3. Quantization
4. Profiling
5. Estimation
   - Enhanced Roofline Model
   - Step-wise Linear Model
6. Traffic Light Controller Case Study
7. Summary

Roofline Model

- Performance bound model for multi-core processors

Samuel Williams, Andrew Waterman **and** David Patterson. “Roofline: an insightful visual performance model for multicore architectures”. In: *Commun. ACM* 52.4 (**april** 2009), **pages** 65–76

Roofline Model

- Performance bound model for multi-core processors
- Bound and bottleneck analysis

Samuel Williams, Andrew Waterman **and** David Patterson. “Roofline: an insightful visual performance model for multicore architectures”. In: *Commun. ACM* 52.4 (**april** 2009), **pages** 65–76

Roofline Model

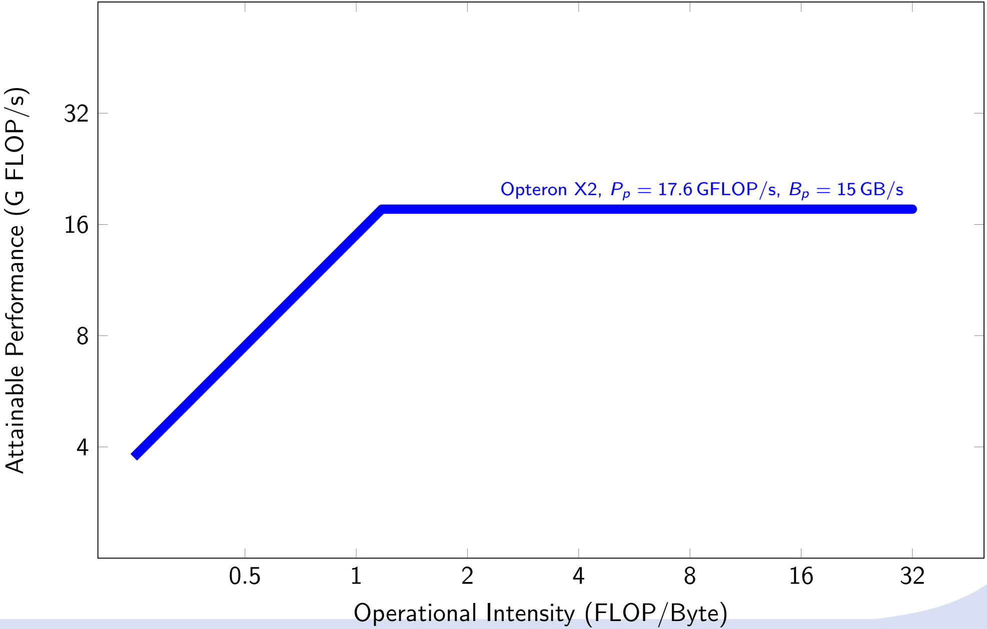
- Performance bound model for multi-core processors
- Bound and bottleneck analysis
- Main bounds due to
  - P_p: Maximum attainable operations per second (in FLOP/second)
  - B_p: Maximum attainable memory traffic between processor (including cache hierarchy) and DRAM (in Byte/second)

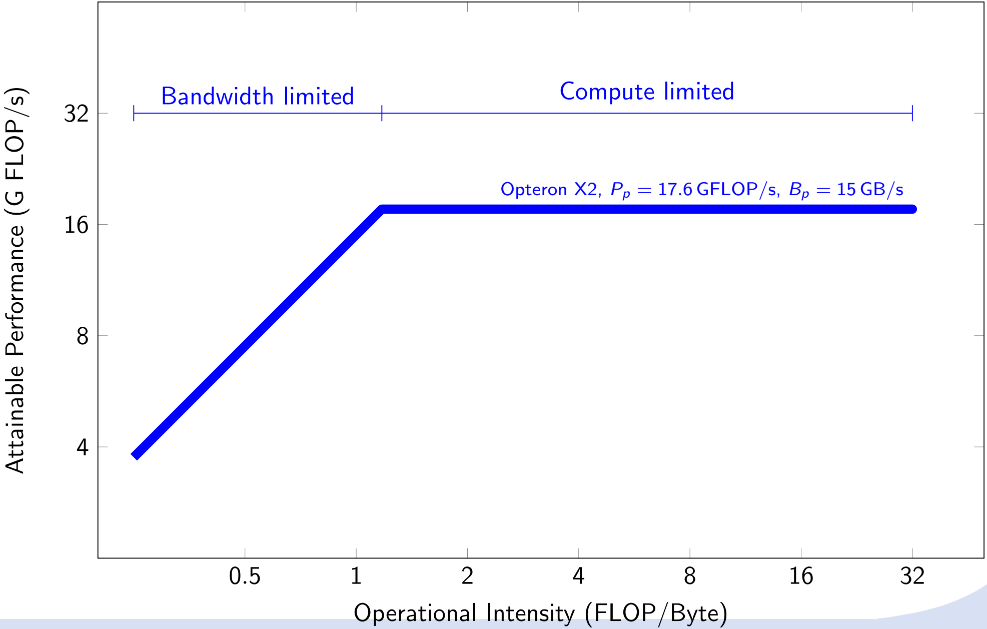
Samuel Williams, Andrew Waterman **and** David Patterson. “Roofline: an insightful visual performance model for multicore architectures”. In: *Commun. ACM* 52.4 (**april** 2009), **pages** 65–76

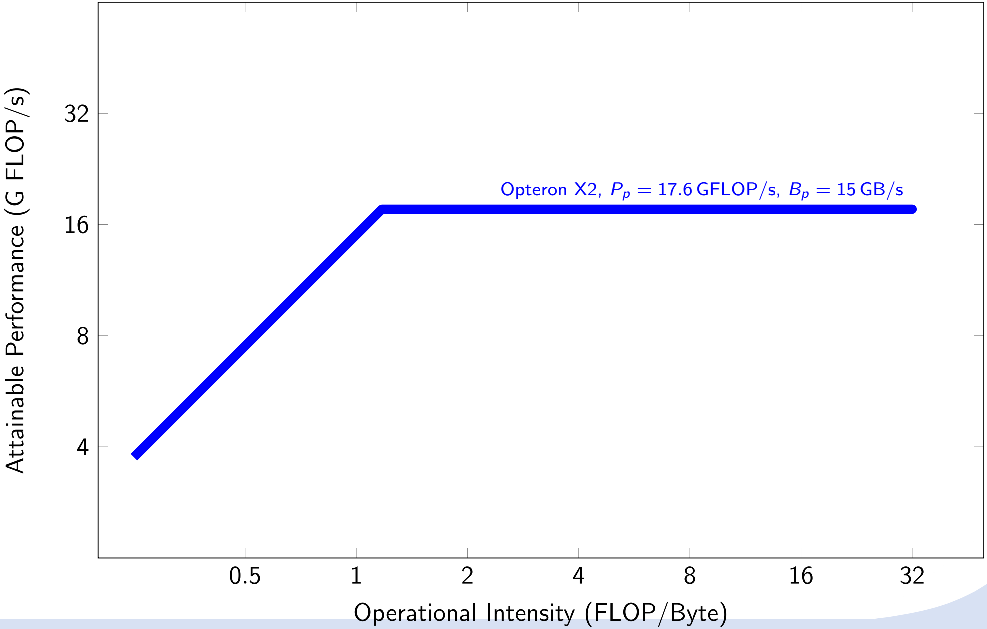
Roofline Model

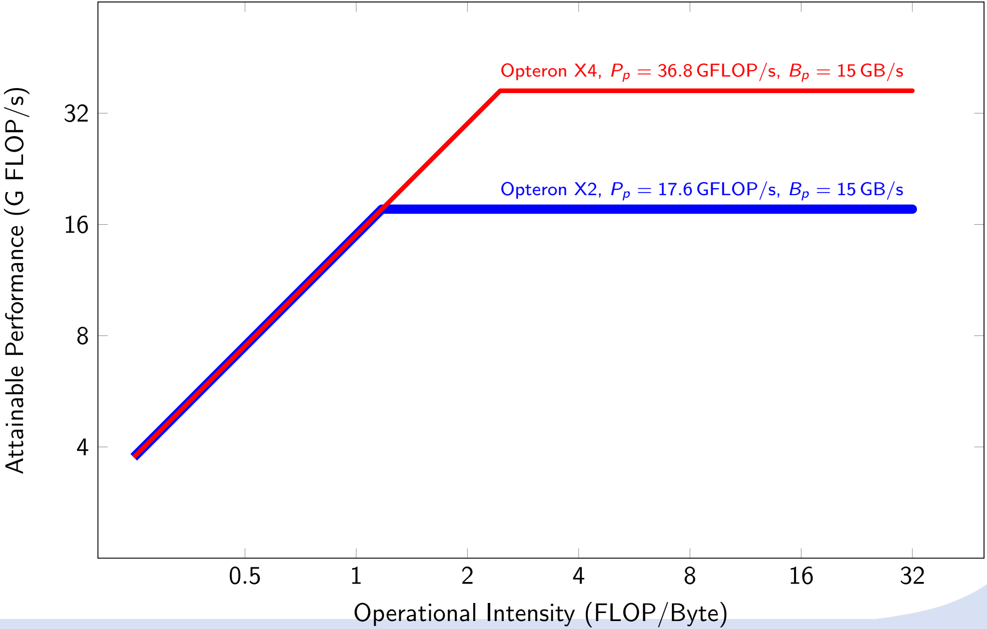
- Performance bound model for multi-core processors
- Bound and bottleneck analysis
- Main bounds due to
  - P_p: Maximum attainable operations per second (in FLOP/second)
  - B_p: Maximum attainable memory traffic between processor (including cache hierarchy) and DRAM (in Byte/second)
- $o = \frac{P}{B}$: Operational intensity is performance per memory traffic (in FLOP/Byte)

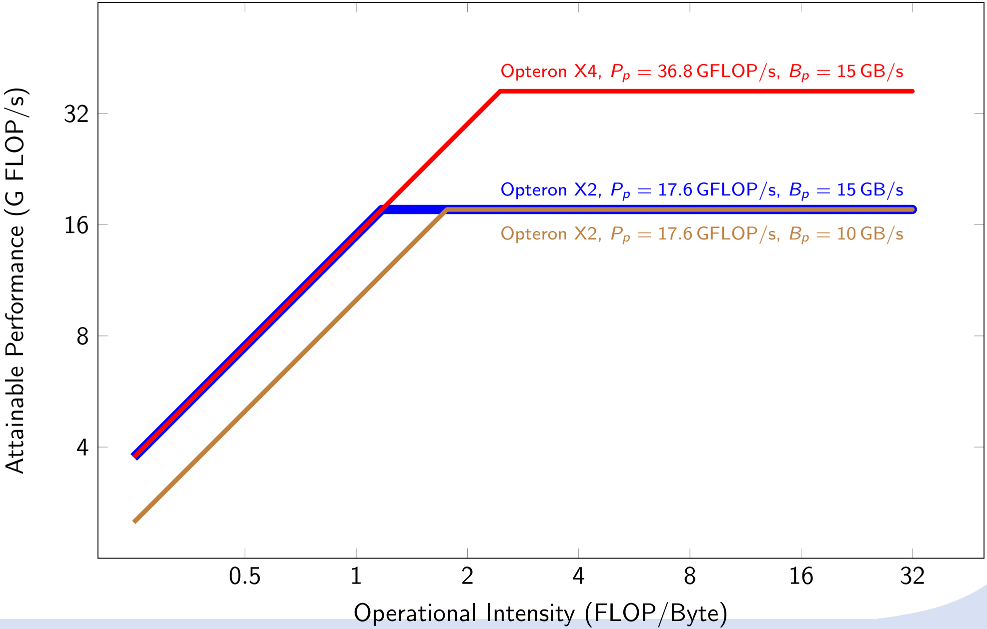
Samuel Williams, Andrew Waterman **and** David Patterson. “Roofline: an insightful visual performance model for multicore architectures”. In: *Commun. ACM* 52.4 (**april** 2009), **pages** 65–76


Opteron X2, $P_p = 17.6\,\mathrm{GFLOP/s}$, $B_p = 15\,\mathrm{GB/s}$
Attainable Performance (G FLOP/s)
32
16
8
4
0.5
1
2
4
8
16
32
Operational Intensity (FLOP/Byte)


Bandwidth limited
Compute limited
Opteron X2, $P_p = 17.6\,\text{GFLOP/s}$, $B_p = 15\,\text{GB/s}$
Attainable Performance (G FLOP/s)
32
16
8
4
0.5
1
2
4
8
16
32
Operational Intensity (FLOP/Byte)


Opteron X2, $P_p = 17.6\,\text{GFLOP/s}$, $B_p = 15\,\text{GB/s}$
Attainable Performance (G FLOP/s)
4
8
16
32
Operational Intensity (FLOP/Byte)
0.5
1
2
4
8
16
32


Opteron X4, $P_p = 36.8\,\text{GFLOP/s}$, $B_p = 15\,\text{GB/s}$
Opteron X2, $P_p = 17.6\,\text{GFLOP/s}$, $B_p = 15\,\text{GB/s}$
Attainable Performance (G FLOP/s)
32
16
8
4
0.5
1
2
4
8
16
32
Operational Intensity (FLOP/Byte)


Opteron X4, $P_p = 36.8\,\text{GFLOP/s}$, $B_p = 15\,\text{GB/s}$
Opteron X2, $P_p = 17.6\,\text{GFLOP/s}$, $B_p = 15\,\text{GB/s}$
Opteron X2, $P_p = 17.6\,\text{GFLOP/s}$, $B_p = 10\,\text{GB/s}$
Attainable Performance (G FLOP/s)
32
16
8
4
0.5
1
2
4
8
16
32
Operational Intensity (FLOP/Byte)

Refined Roofline Model

$$\hat{T}_{\text{roof}_n}(f_n, r_n) = \max(\frac{f_n}{P_{\text{peak}}}, \frac{r_n}{B_{\text{peak}}})$$

with

n	... Layer
$\hat{T}_{\text{roof}_n}$	... Latency for layer n
f_n	... No of operations
r_n	... No of bytes to be transferred
P_{peak}	... Peak performance (FLOP/s))
B_{peak}	... Peak bandwidth (Byte/s)

Matthias Wess, Marco Ivanov, Christian Unger, Anvesh Nookala, Alexander Wendt **and** Axel Jantsch. "ANNETTE: Accurate Neural Network Execution Time Estimation With Stacked Models". In: *IEEE Access* 9 (2021), **pages** 3545–3556

Refined Roofline Model

$$\hat{T}_{\text{ref}_n}(f_n, r_n) = max(\frac{f_n}{P_{\text{peak}} u_{\text{eff}_n}}, \frac{r_n}{B_{\text{peak}}})$$

$$u_{\text{eff}}(\vec{x}) = \prod_{i=1}^{A} \frac{x_i / s_i}{\lceil x_i / s_i \rceil}$$

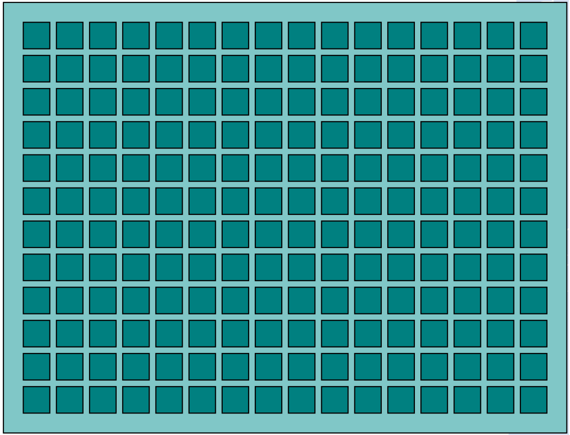
with

u_{eff_n} . . . utilization efficiency
$\vec{s}$ . . . Number of resources
$\vec{x}$ . . . Number of operations

Refined Roofline Model

12x6x128 Input Feature Map
1x1 Convolution
256 Output Feature Map

16x12 PE Array



$$\vec{s} = (16 \times 12)$$

$$\vec{x} = (12 \times 6 \times 128 \times 256 \times 1 \times 1)$$

$$u_{\mathrm{eff}}(\vec{x}) = \prod_{i=1}^{A} \frac{x_i/s_i}{\lceil x_i/s_i \rceil} = \frac{12/12}{1} \cdot \frac{6/16}{1} = 0.375$$

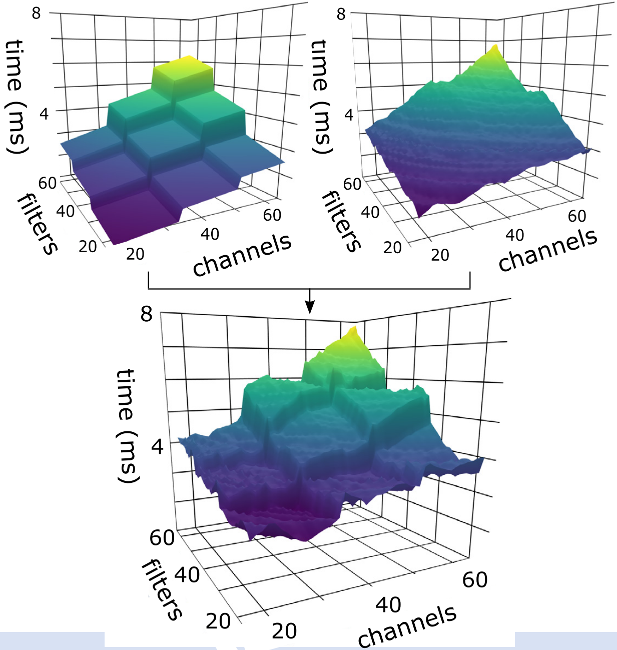
Statistical Model

- Regresssion model to estimate utilization efficiency u_{stat}
- Feature vector for 2D convolution: $(h, w, c, f, k_h, k_w, \text{stride}, \#\text{ops}, \#\text{in}, \#\text{out}, \#\text{weights})$.
- Random forest prediction method

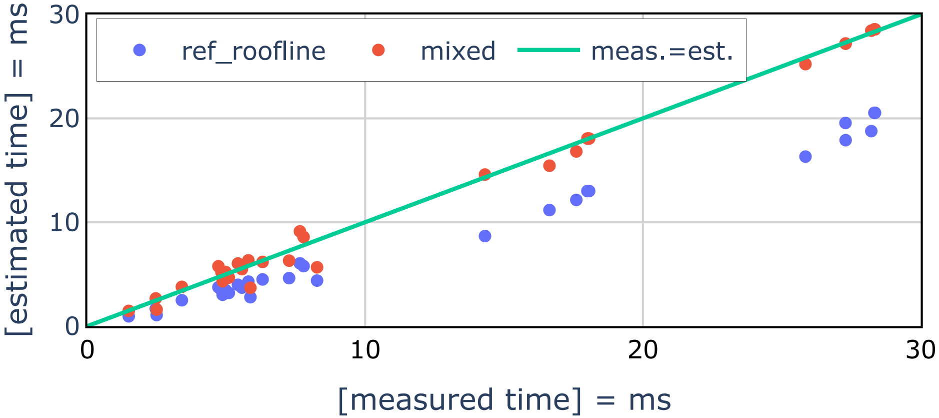
$$\hat{T}_{\text{stat}_n}(f_n, r_n) = max\left(\frac{f_n}{P_{\text{peak}}\,u_{\text{stat}_n}}, \frac{r_n}{B_{\text{peak}}}\right)$$

Mixed Model

$$\hat{T}_{\text{mixed}_n}(f_n, r_n) = max\left(\frac{f_n}{P_{\text{peak}}\, u_{\text{eff}_n}\, u_{\text{stat}_n}}, \frac{r_n}{B_{\text{peak}}}\right)$$


time (ms)
8
4
filters
60
40
20
channels
20
40
60
time (ms)
8
4
filters
60
40
20
channels
20
40
60
time (ms)
8
4
filters
60
40
20
channels
20
40
60

Mixed Model



Test subset of NASBench data set
NCS2 platform

Mixed Roofline Model - Results

Device	Model Type	Measured (ms)	MAE (ms)	MAPE (%)
NCS2	Roofline	226.3	67.8	30.0
	Ref. Roofline	219.7	64.9	29.6
	Statistical	233.8	18.5	7.9
	Mixed	200.9	**15.0**	**7.4**
ZCU102	Roofline	19.8	6.1	30.9
	Ref. Roofline	15.0	4.1	27.2
	Statistical	41.7	2.5	6.0
	Mixed	25.6	**0.9**	**3.5**

Network execution time for 12 networks

(4 Inception, 2 ResNet, 1 FPN, 1 Open Pose, 2 MobileNet, 2 Yolo)

MAE ... Mean Absolute Error

MAPE ... Mean Absolute Percentage Error

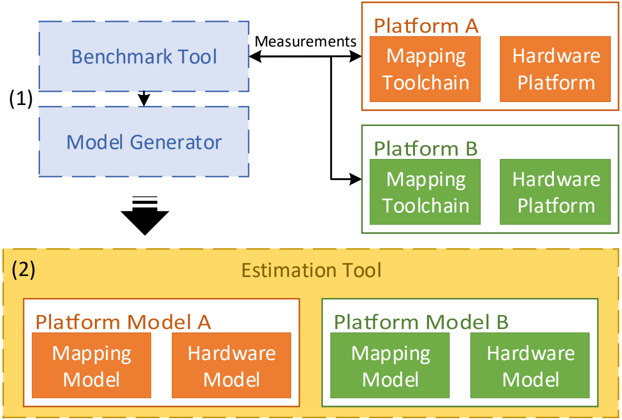
Matthias Wess, Marco Ivanov, Christian Unger, Anvesh Nookala, Alexander Wendt **and** Axel Jantsch. "ANNETTE: Accurate Neural Network Execution Time Estimation With Stacked Models". In: *IEEE Access* 9 (2021), **pages** 3545–3556

Outline

1. Challenge and Motivation
2. Convolutional Neural Networks
   - HW Friendly Optimizations
   - CNN Accelerator Architectures
3. Quantization
4. Profiling
5. **Estimation**
   - Enhanced Roofline Model
   - Step-wise Linear Model
6. Traffic Light Controller Case Study
7. Summary

Estimation Framework Blackthorn

- Assuming step-wise, linear functions for resource usage
- Measuring selected points
- Function fitting based on measurements
- Hierarchical fitting for each dimension: filter, channel,



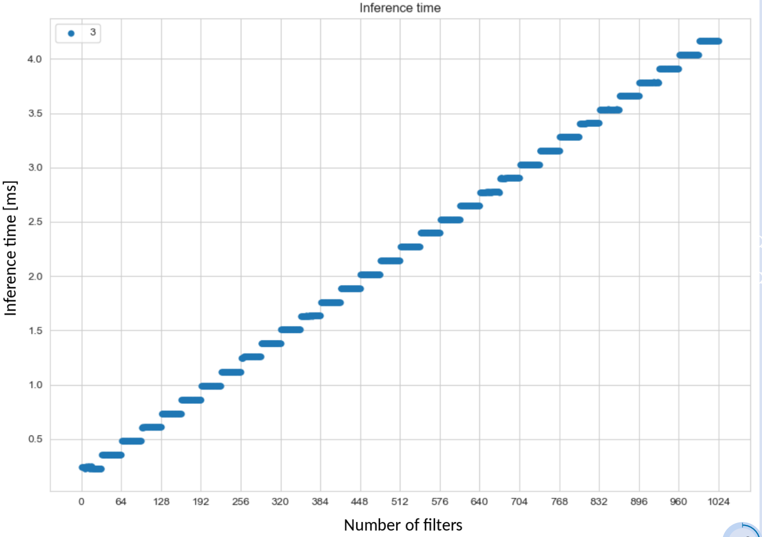
Inference Run Time Estimation

Assumption:

- Inference time as a function of problem size is a combination of step and linear functions due to limited parallel resources.
-

Example:

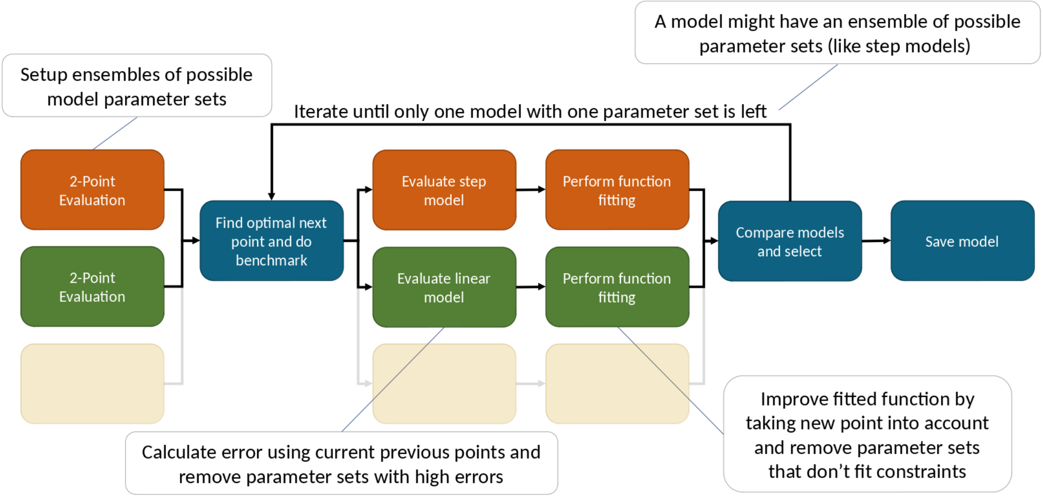
- Single convolutional layer sweep
- 32x32x64 with k filter and kernel size 3



Inference Run Time Estimation

- Assumption:
  The inference time can be approximated by a combination of linear and step functions for each dimension, such as filter, channels, etc.
- Determining the function based on selected measurements
- Goals: automatic computation of estimation functions for latency, power consumption and various platforms.

Automatic Estimation Function Generation



Setup ensembles of possible model parameter sets

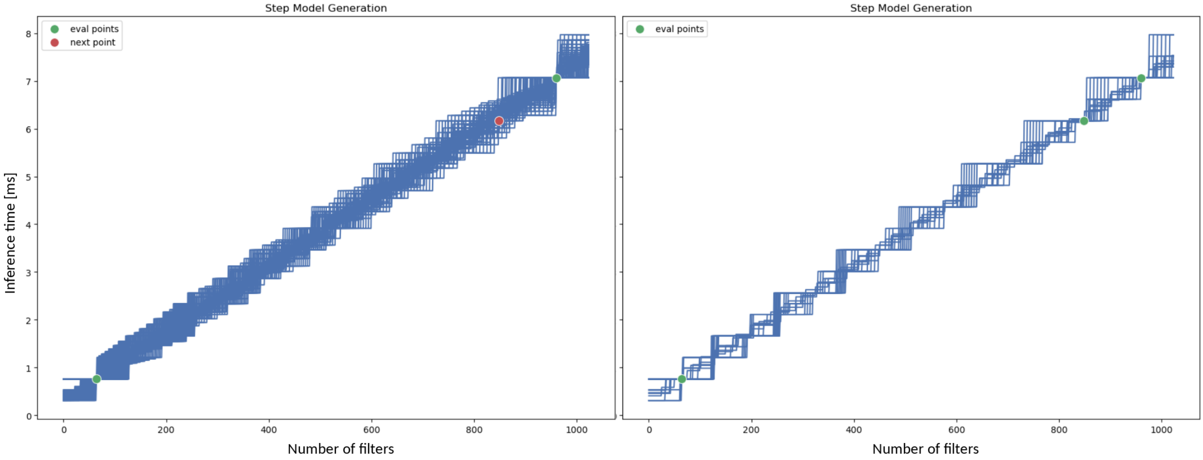
A model might have an ensemble of possible parameter sets (like step models)

Iterate until only one model with one parameter set is left

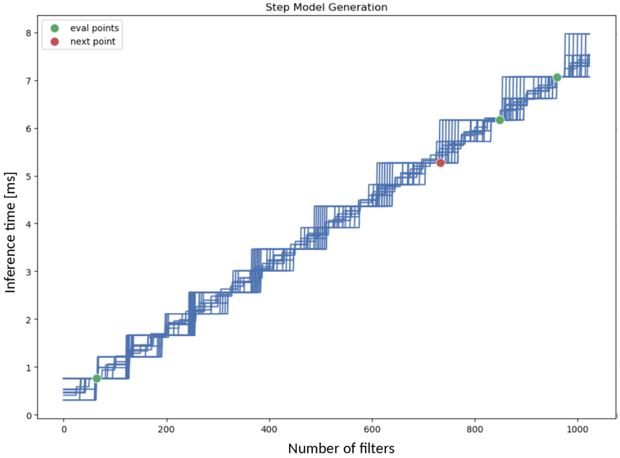
Calculate error using current previous points and remove parameter sets with high errors

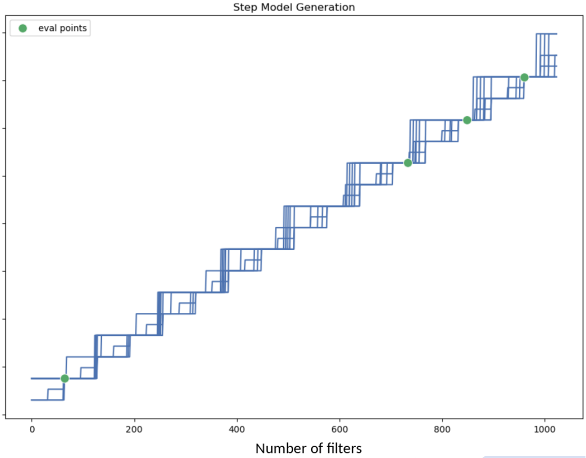
Improve fitted function by taking new point into account and remove parameter sets that don't fit constraints

Iterative Refinement

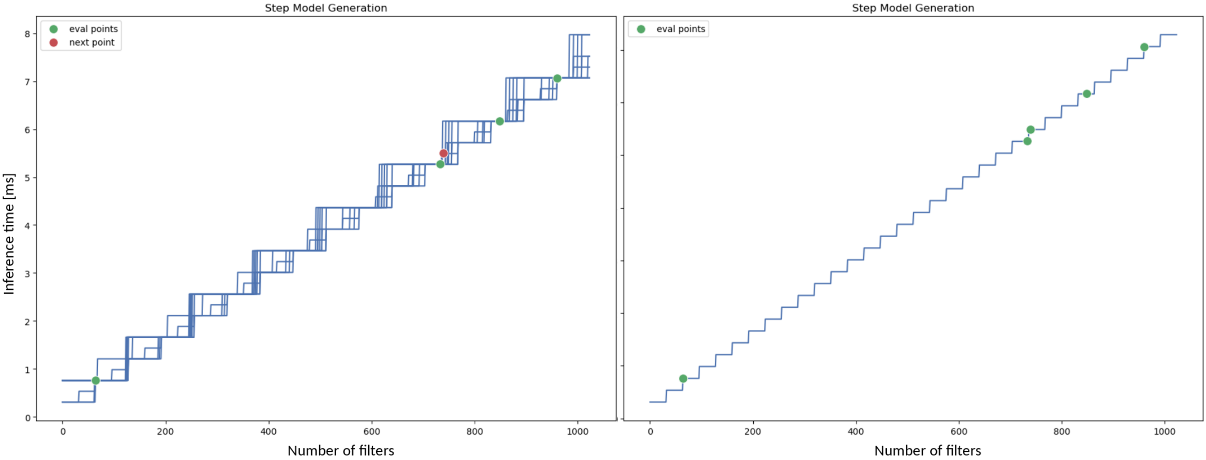


Iterative Refinement



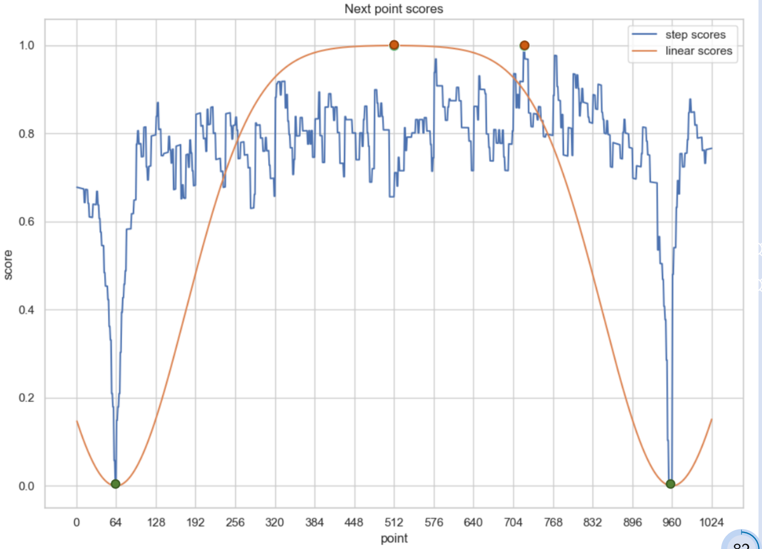


Iterative Refinement



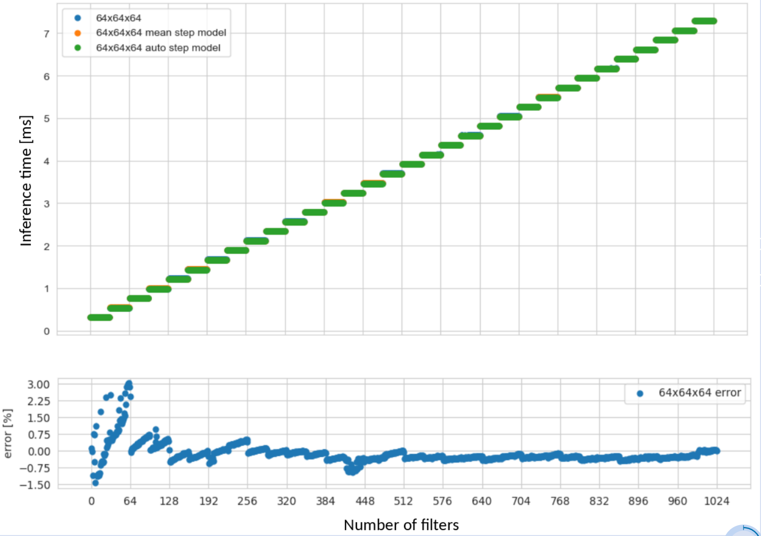
Next Point Selection

- Linear function criteria:
  - Point furthest away from previous points
- Step function criteria
  - Point with most unique discrete levels
  - Point with largest range of values
  - Point farthest away from previous points
- Next point selection: Point with highest score



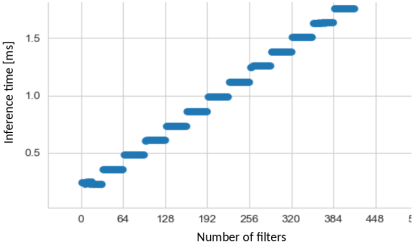
Method Evaluation

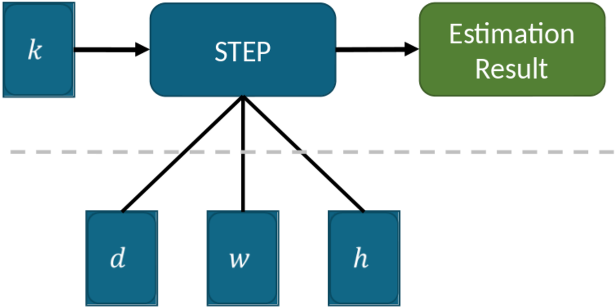
- Results after 3 iterations (5 measurement points)
- Execution times:
    - Full sweep: 3-4 h
    - Proposed approach: 2-5 minutes



2D Example

- Phase 1: Estimate function in single dimension: number of filters
- Result: step function

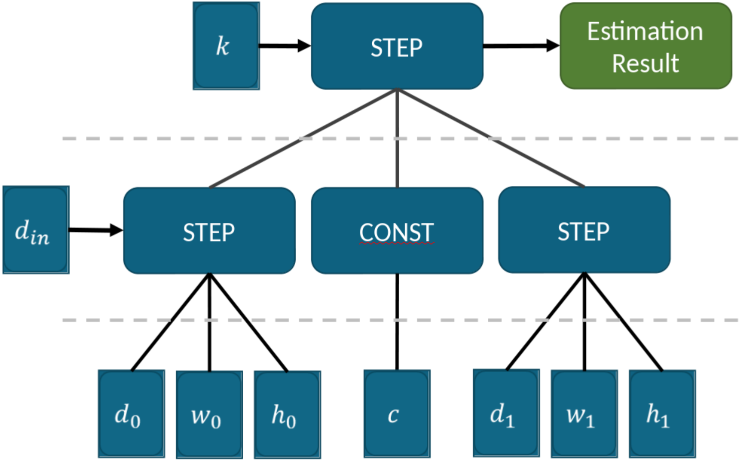




$$0{,}216 + \left\lfloor \frac{k-1}{32} \right\rfloor 0{,}01286$$

2D Example

- Phase 2: Test how d, w and h behave in the next dimension
- Next dimension: input channels d_{in}
- Result:
  - Step function: $d_{0=0.1418}$, $w_0 = 8$, $\mathrm{h}_0 = 0.0106$
  - Constant: $c = 32$
  - Step function: $d_1 = 0.044$, $w_1 = 8$, $h_1 = 0.0121$



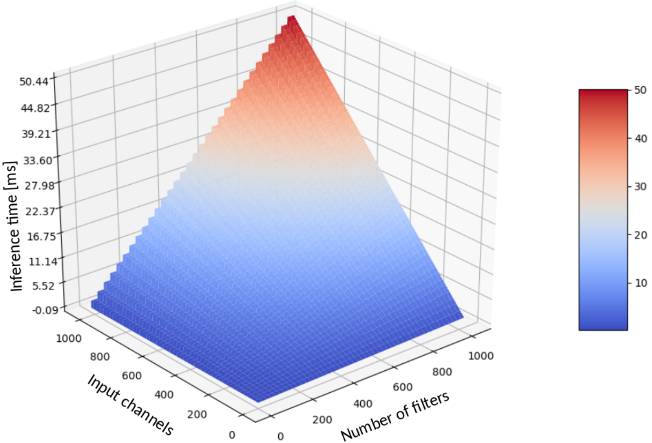
$$0.1418 + \left\lfloor \frac{d_{in} - 1}{8} \right\rfloor 0.0106 + \left\lfloor \frac{k - 1}{32} \right\rfloor \left(0.044 + \left\lfloor \frac{d_{in} - 1}{8} \right\rfloor 0.0121 \right)$$

2D Example

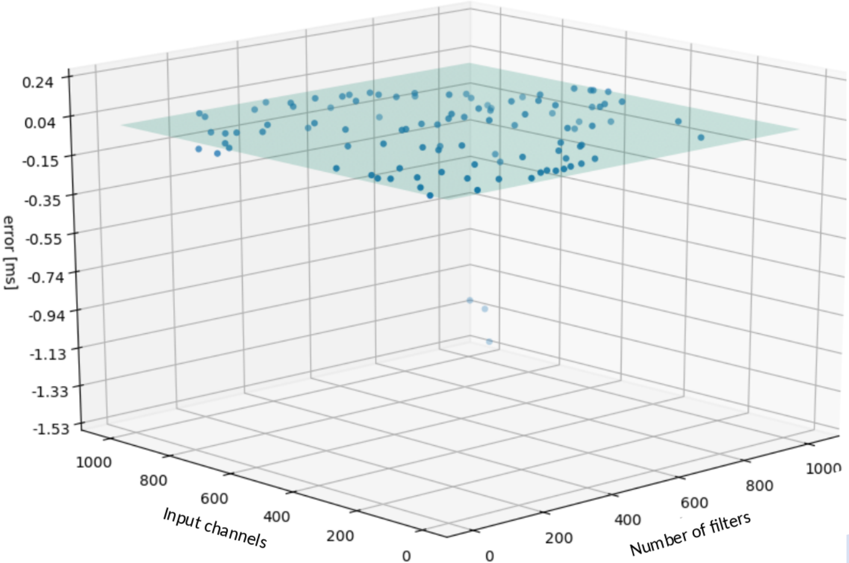
Generated model:

$$
\begin{aligned}
f(d_{\text{in}}, k) \\
&= 0.1418 + \lfloor \frac{d_{\text{in}} - 1}{8} \rfloor 0.0106 \\
&+ \lfloor \frac{k-1}{32} \rfloor \left(0.044 + \lfloor \frac{d_{\text{in}} - 1}{8} \rfloor 0.0121 \right)
\end{aligned}
$$

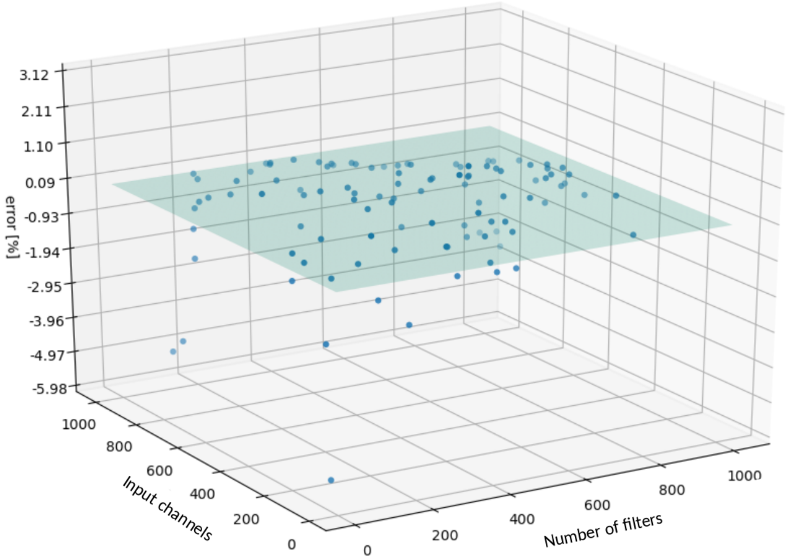
- Meausrement points: 112
- Execution time: 32 minutes



2D Example - Error



2D Example - Error



2D Example - Error

Slice through 2D plane at $k = 1024$

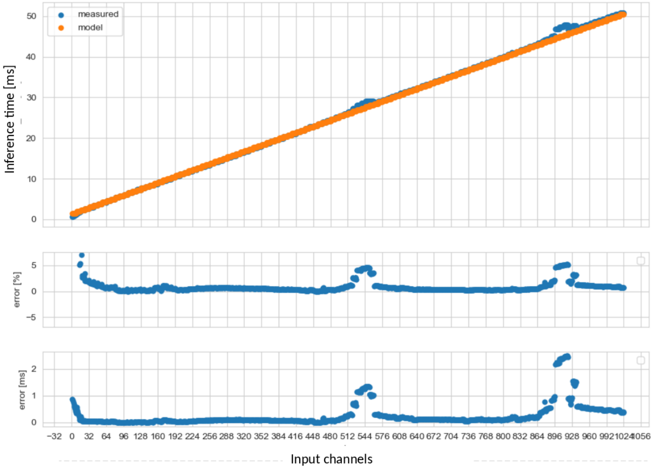
$$f(d_{\text{in}}, k)$$

$$= 0.1418 + \lfloor \frac{d_{\text{in}} - 1}{8} \rfloor 0.0106$$

$$+ \lfloor \frac{k - 1}{32} \left(0.044 + \lfloor \frac{d_{\text{in}} - 1}{8} \rfloor 0.0121\right)$$

$$f(d_{\text{in}}, 1024)$$

$$= 1.5058 + \lfloor \frac{d_{\text{in}} - 1}{8} \rfloor 0.3857$$

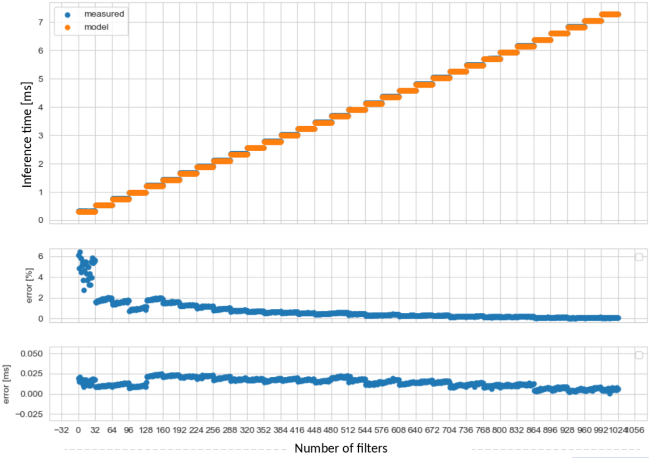


2D Example - Error

Slice through 2D plane at $d_{\text{in}} = 128$

$$
\begin{aligned}
&f(d_{\text{in}}, k) \\
&\quad = 0.1418 + \lfloor \frac{d_{\text{in}} - 1}{8} \rfloor 0.0106 \\
&\quad + \lfloor \frac{k - 1}{32} \rfloor \left(0.044 + \lfloor \frac{d_{\text{in}} - 1}{8} \rfloor 0.0121 \right)
\end{aligned}
$$

$$
\begin{aligned}
&f(128, k) \\
&\quad = 0.3008 + \lfloor \frac{k - 1}{32} \rfloor 0.2255
\end{aligned}
$$



Blackthorn Estimation Results

Device	Network	Measured (ms)	MAE (ms)	MAPE (%)
Jetson Nano	AlexNet	27.8	1.5	5.5
	VGG16	154.9	0.7	0.5
	ResNet 50	49.2	1.1	2.3
	MobileNetV2	13.7	0.5	3.6
Jetson TX2	AlexNet	11.2	0.8	6.7
	VGG16	61.2	0.9	1.4
	ResNet 50	21.4	1.0	4.8
	MobileNetV2	6.7	0.3	4.2

Network execution time
MAE ... Mean Absolute Error
MAPE ... Mean Absolute Percentage Error

Martin Lechner **and** Axel Jantsch. "Blackthorn: Latency Estimation Framework for CNNs on Embedded Nvidia Platforms". In: *IEEE Access* (2021)

Latency Estimation Summary

- Exploiting the discrete nature of HW resources
- Systematic benchmarking of a platform and a set of network layer types
- Fast estimation function for latency for any new network with known layer types
- Results for several platforms are robust

Network	**Estimation Error [%]**			
	NCS2	**ZCU102**	**Jetson Nano**	**Jetson TX2**
YoloV3	4.1	3.2	-	-
MobileNetV2	4.3	4.2	3.6	4.2
ResNet50	8.2	1.2	2.4	4.8
FPN Net	9.3	7.5	-	-
AlexNet	5.2	4.8	5.5	6.6
VGG16	11.3	6.2	0.5	1.4

Outline

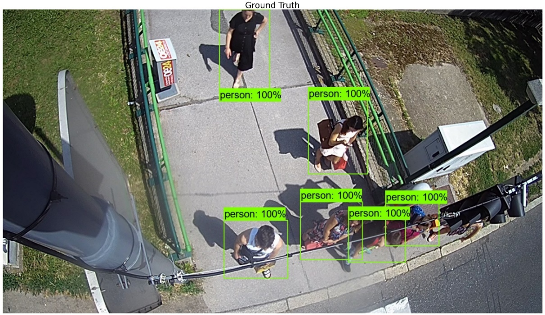
1. Challenge and Motivation
2. Convolutional Neural Networks
   - HW Friendly Optimizations
   - CNN Accelerator Architectures
3. Quantization
4. Profiling
5. Estimation
   - Enhanced Roofline Model
   - Step-wise Linear Model
6. **Traffic Light Controller Case Study**
7. Summary

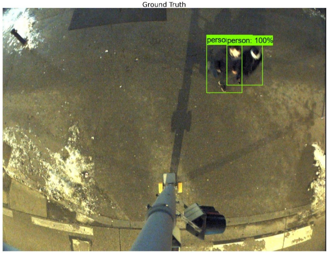
TRAFFIC LIGHT CONTROLLER CASE STUDY

Traffic Light Controller

Data set:

- training: 19087 images
- positive examples 47%
- validation: 13184
- positive examples 26%
- Resolution: 1280x720
- Issue: Validation 4h/network
  → validation set: 1319





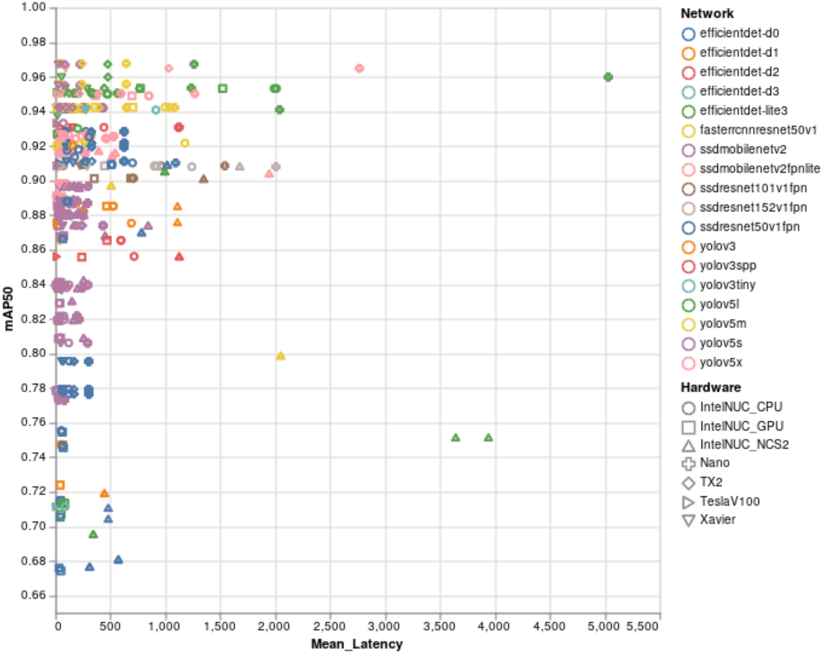
Platforms under Study

Name	Performance [T op/s]	Memory [GB]	Power [W]	Cost [€]
NVIDIA Xavier AGX	32	16	10–30	800
NVIDIA Jetson TX2	1.3	4	7.5–15	260
NVIDIA Jetson Nano	0.5	4	5–10	120
Intel NCS2	1	0.5	5	80
Intel NUC CPU (i7-8650U)	22.4	32	15	600
Intel NUC GPU (Intel UHD 620)	0.8	32	15	600
Tesla V100	130	32	250	>1000

Networks under Study

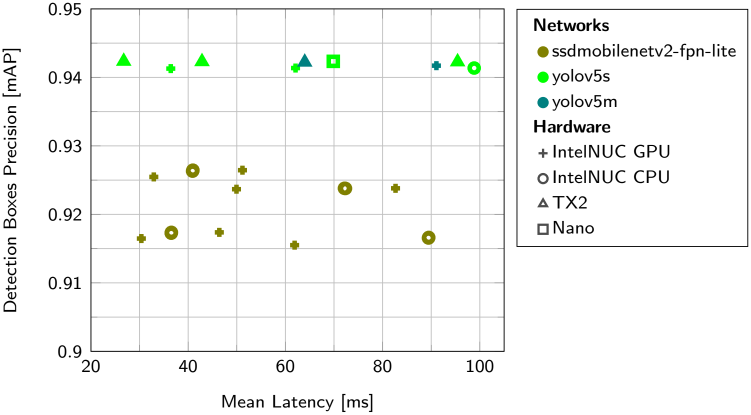
Name	Framework used	No of parameters (10^6)
ssdmobilenetv2fpnlite	Tensorflow	2.8
efficientdet-d0	Tensorflow	3.9
ssdmobilenetv2	Tensorflow	4.5
yolov5s	Pytorch	7.0
yolov3tiny	Pytorch	8.6
yolov5m	Pytorch	21.0
yolov5l	Pytorch	46.6
ssdresnet50v1fpn	Tensorflow	50.7
yolov3	Pytorch	61.4
yolov3spp	Pytorch	62.5
ssdresnet101v1fpn	Tensorflow	69.7
ssdresnet152v1fpn	Tensorflow	85.3
yolov5x	Pytorch	87.1

All solutions



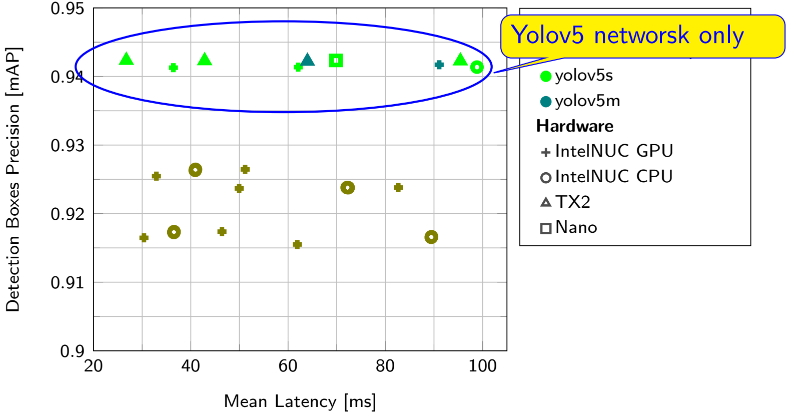
Solutions under cost constraints

latency
$\leq$ 100 ms
and
mAP50
$\geq$ 0.9.



Solutions under cost constraints

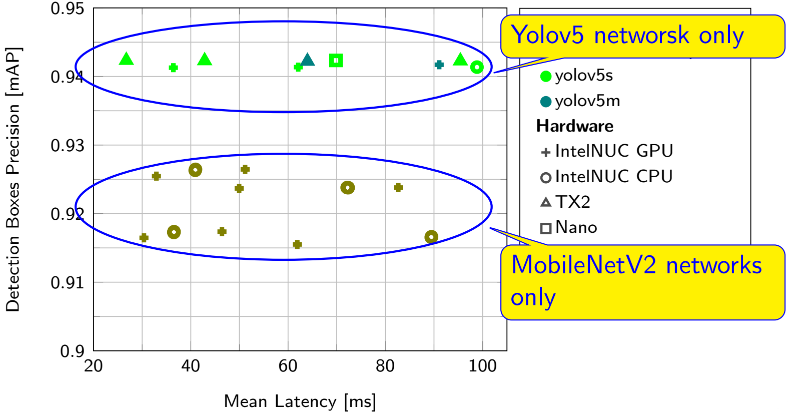
latency
$\leq$ 100 ms
and
mAP50
$\geq$ 0.9.



Yolov5 networsk only

Solutions under cost constraints

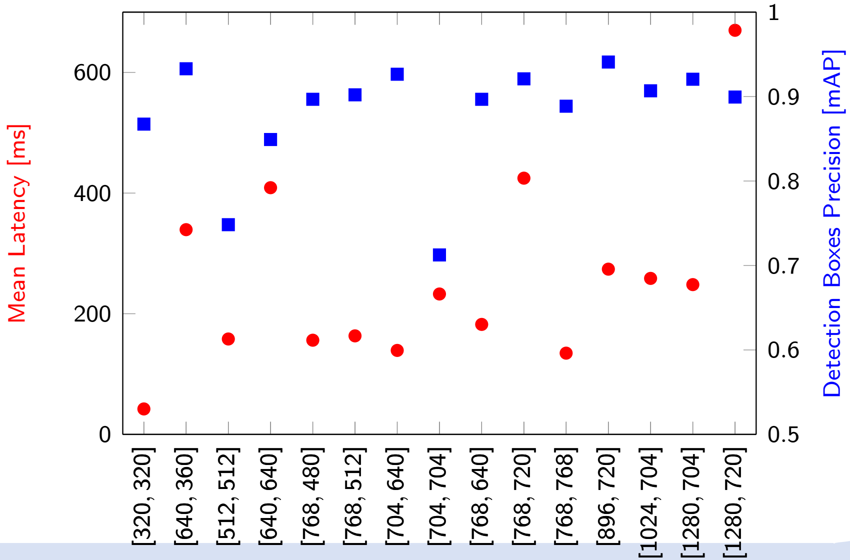
latency
$\leq$ 100 ms
and
mAP50
$\geq$ 0.9.



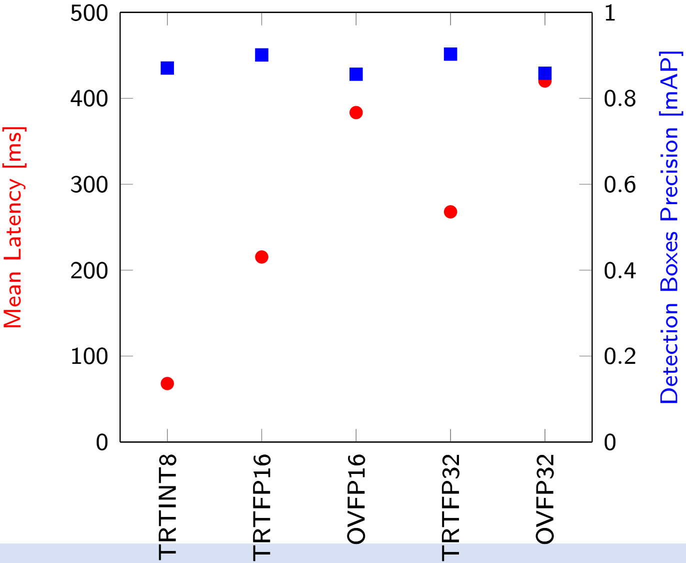
Yolov5 networsk only

MobileNetV2 networks only

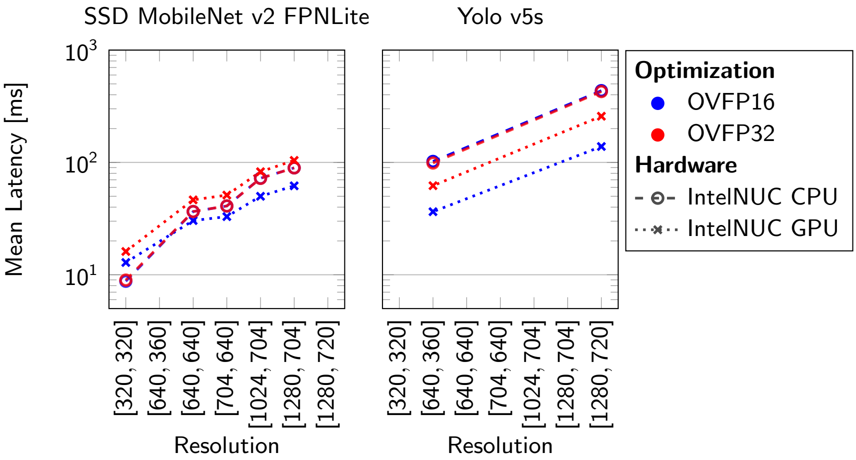
Impact of Image Resolution



Impact of Quantization

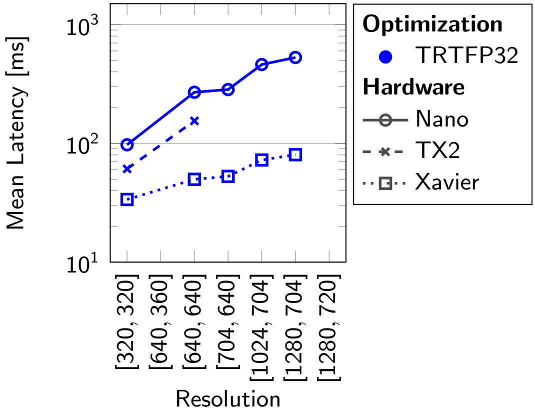


Impact of resolution and quantization on the Intel NUC platform

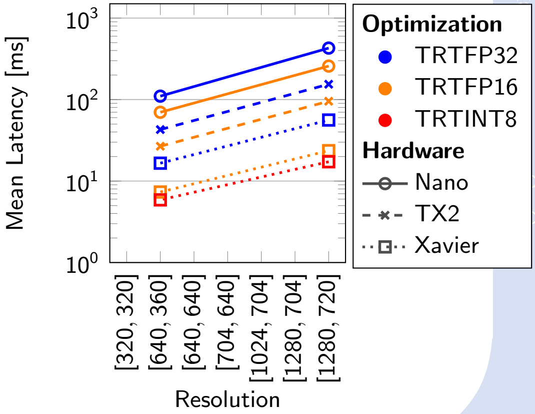


Impact of resolution and quantization on the Nvidia platform

SSD MobileNet v2 FPNLite



Yolo v5 small



Traffic Light Case Study - Summary

Summary

- **Yolo v5s** is the most suitable network;

Traffic Light Case Study - Summary

Summary

- **Yolo v5s** is the most suitable network;
- **Nvidia Jetson Nano and TX2** are most suitable platforms

Traffic Light Case Study - Summary

Summary

- **Yolo v5s** is the most suitable network;
- **Nvidia Jetson Nano and TX2** are most suitable platforms
- Yolo v5m and MobileNetV2 are reasonable networks;

Traffic Light Case Study - Summary

Summary

- **Yolo v5s** is the most suitable network;
- **Nvidia Jetson Nano and TX2** are most suitable platforms
- Yolo v5m and MobileNetV2 are reasonable networks;
- IntelNUC GPU, IntelNUC CPU are reasonable platforms.

Traffic Light Case Study - Summary

Summary

- **Yolo v5s** is the most suitable network;
- **Nvidia Jetson Nano and TX2** are most suitable platforms
- Yolo v5m and MobileNetV2 are reasonable networks;
- IntelNUC GPU, IntelNUC CPU are reasonable platforms.
- Latency depends linear on image resolution

Traffic Light Case Study - Summary

Summary

- **Yolo v5s** is the most suitable network;
- **Nvidia Jetson Nano and TX2** are most suitable platforms
- Yolo v5m and MobileNetV2 are reasonable networks;
- IntelNUC GPU, IntelNUC CPU are reasonable platforms.
- Latency depends linear on image resolution
- FP16 quantization is a sweet spot compared to FP32 and INT8

Outline

1. Challenge and Motivation
2. Convolutional Neural Networks
   - HW Friendly Optimizations
   - CNN Accelerator Architectures
3. Quantization
4. Profiling
5. Estimation
   - Enhanced Roofline Model
   - Step-wise Linear Model
6. Traffic Light Controller Case Study
7. **Summary**

SUMMARY

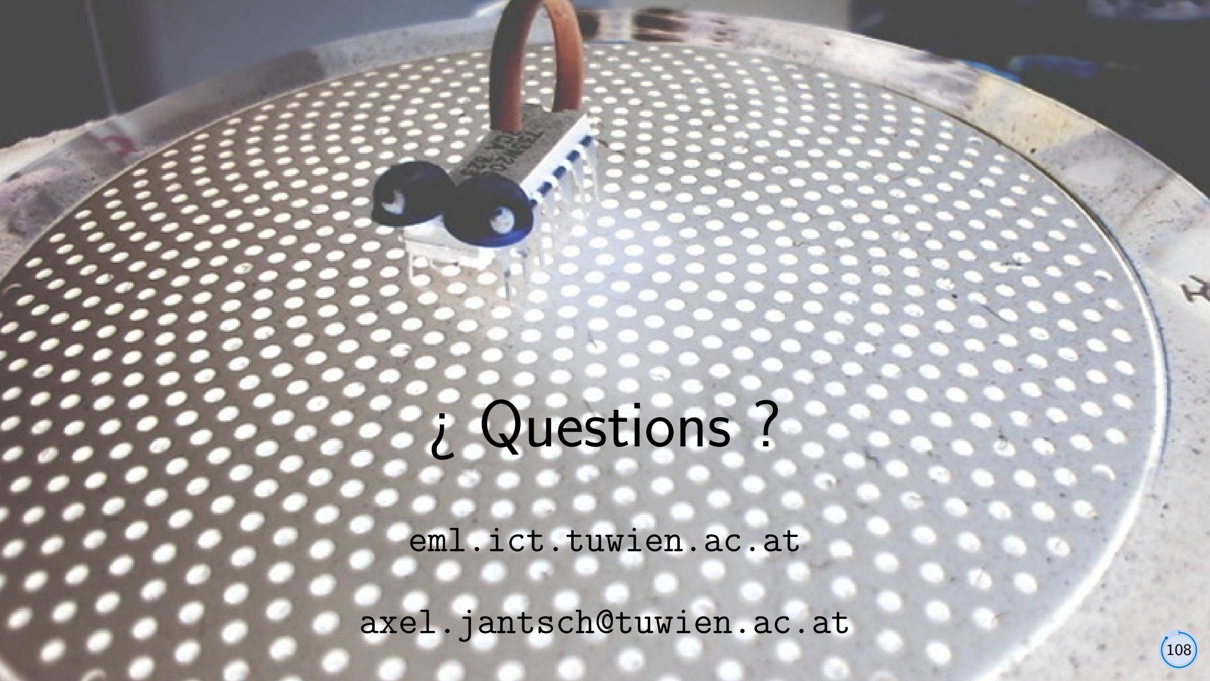
Summary

Summary

- Embedded Machine Learning has many applications;
  - Bandwidth limitations;
  - Delay constraints;
  - Privacy;
  - Security;

Summary

- Embedded Machine Learning has many applications;
  - Bandwidth limitations;
  - Delay constraints;
  - Privacy;
  - Security;
- There are distinct challenges:
  - Limited resources;
  - Specialized HW platforms;
  - Huge design space for optimization and mapping.

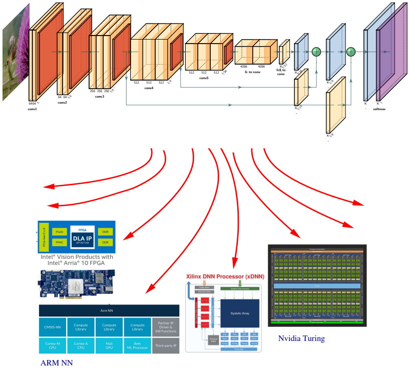


¿ Questions ?

eml.ict.tuwien.ac.at

axel.jantsch@tuwien.ac.at

Summary Results at eml@tuwien



Mapping

- Quantization aware training
- Platform aware pruning
- Post-training quantization
- Time-series DNNs on Microcontrollers

Applications

- Image: Yolo, MobilNet, ResNet, VGG, DeepLabv3+, EfficientNet
- Time series: InceptionTime, LSTM

DNN Optimization

- Shunt connections
- DNN Fusion
- Automated pruning
- Partitioning

Platforms

- GPU
- ARM
- FPGA
- TPU, NCS, ...

Estimation

- ANNETTE
- Blackthorn
- Profiling and benchmarking
- Estimation confidence

Output

- Design contests
- Papers
- Tools, scripts, and flows

List of Results -I

Mapping	Platform specific	Quantization aware training [DSD 2023]
		Traffic sign classification [IGSC 2019]
		Platform aware pruning [Under submission]
		Post-training quantization for FPGAs [BSc 2021]
		ECG on MCUs [ICCAD 2023]
		Energy aware TinyML [ISLPED 2023]
		Object tracking [MSc 2023]
	Distributed	Pruning and distributed mapping [SAS 2023]
		Waist tightening [Electronics 2021, RAGE 2023]
		Design space exploration [ACCESS 2021]
DNN Optimization		Shunt connections [AIAI 2021]
		Multispectral Feature Fusion [DATE 2023]
		Automated Pruning [INDIN 2021]
		Post-training quantization for FPGAs [BSc 2021]

List of Results - II

Estimation	Latency	ANNETTE [ACCESS 2021] Blackthorn [ACCESS 2021]
	Power	Power analysis [ISLPED 2023]
	Confidence	Estimation quality assessment [ACCESS 2024]
Application	Image	Object identification and segmentation [SAMOS 2022] Object identification [Bookchapter 2023] Classification [IGSC 2019] Object identification [RAGE 2023, Electronics 2021, SAS 2023] Anomaly detection [DSD 2023]
	Time series	Prediction [NeurIPS 2023] Anomaly detection [DSD 2023] Classification [ISLPED 2023] Prediction [ICMLA 2023]

List of Results - III

Case studies

Traffic light controller [Bookchapter 2023]
Ragweed detection on drones [SAMOS 2022]
Garbage segmentation network
Climate modeling [NeurIPS 2023]
ECG [ICCAD 2023]
Smart grids [ICMLA 2023]
Rail track monitoring [DSD 2023]

Platforms and DNNs

Platforms

GPU	Nvidia Jetson TX2, Jetson Nano, Jetson AGX Xavier, Orin
ARM	i.MX8M, i.MX93, GAP8, GAP9, Hailo, STM32
FPGA	ZCU102, Xilinx Zync UltraScale+
Others	Edge TPU, Intel NUC, NCS2

DNNs

Image	Yolo, MobilNet, ResNet, VGG, DeepLabv3+, EfficientNet
Timeseries	InceptionTime, LSTM

Design Contests

Contest	Date	Result
DAC System Design Contest	June 2021	25. place
Evergreen Innovation Camp Hackathon	April 2022	1. Place
DAC System Design Contest	June 2022	19. place
ACM/IEEE TinyML Design Contest at ICCAD	November 2022	17. place
ACM/IEEE TinyML Design Contest at ICCAD	November 2023	1. place

Toolbox I

Quantization framework: Fast, Quantization Aware DNN Training for Efficient HW Implementation [DSD 2023].
Repository: `FastQATforPOTRescaler`

ANNETTE: Accurate Neural Network Execution Time Estimation [ACCESS 2021].
Repository: `annette`

mmdnn graph utils: Package that allows importing mmdnn-ir files, onnx-files and stores them in ANNETTE format Network Architecture description files.
Repository: `mmdnn-graph-utils`

Inference modules for ANNETTE: Tools to optimize, execute and process neural networks for ANNETTE [ACCESS 2021].
Repository: `inference_modules`

Blackthorn: The Balackthorn latency estimation tools for Nvidia platforms [ACCESS 2021].
Repository: `blackthorn`

Toolbox II

tf2oda: TF2 Object Detection API Models on NVIVIDA Devices.
Repository: `hwmodule-tf2oda-nvidia`

C++ Framework for μC: Framework to support lean implementations of Deep Neural Networks (DNNs) on micro controller platforms [ICCAD 2023].
Repository: To be published.

ICCAD TinyML 2023 Contest: The project that won the ICCAD TinyML 2023 Contest [ICCAD 2023].
Repository: `ICCAD-TinyML2023-1st-Place`

powerutils: Utility Tools for DNN port analysis [DSD 2023].
Repository: `powerutils`

Embedded Machine Learning Toolbox: Scripts for automated and simplified training and inference [Bookchapter 2023].
Repository: `eml-tools`

Toolbox III

EML Object Detection Android App: App for object detection that can be downloaded to Android phones.
Repository: `eml-mobile-photo-app`

Bibliography I

[Bit+23a] Matthias Bittner, Domink Dallinger, Matthias Wess, Daniel Schnöll **and** Axel Jantsch. *Energy Aware Time Series Classification for low-power microcontrollers*. Design contest at the International Symposium of Low Power Electronic Design (ISLPED). Vienna, Austria, **august** 2023.

[Bit+23b] Matthias Bittner **andothers**. "An LSTM-based Downscaling Framework for Australian Precipitation Projections". In: *NeurIPS 2023 Workshop: Tackling Climate Change with Machine Learning at the Conference on Neural Information Processing Systems*. December, 2023.

[Bit+23c] Matthias Bittner, Daniel Hauer, Christian Stippel, Katharina Scheucher, Robin Sudhoff **and** Axel Jantsch. "Forecasting Critical Overloads based on Heterogeneous Smart Grid Simulation". In: *Proceedings of the International Conference on Machine Learning and Applications (ICMLA*. IEEE **and** AMLA. Jacksonville, Florida, USA, **december** 2023.

[Bre+23] David Breuss, Maximilian Götzinger, Jenny Vuong, Clemens Reisner **and** Axel Jantsch. "VADAR: A Vision-based Anomaly Detection Algorithm for Railroads". In: *Proceedings of the 26th Euromicro Conference on Digital System Design (DSD)*. Durres, Albania, **september** 2023.

Bibliography II

[Cha20] Frances S. Chance. "Interception from a Dragonfly Neural Network Model". In: *International Conference on Neuromorphic Systems 2020 (ICONS)*. Oak Ridge, TN, USA. ACM, New York, NY, USA, **july** 2020.

[Dal21] Dominik Dallinger. "FPGA optimized dynamic post-training quantization of TinyYoloV3". Bachelor's Thesis. TU Wien, 2021.

[GLW21] Andreas Glinserer, Martin Lechner **and** Alexander Wendt. "Automated Pruning of Neural Networks for Mobile Applications". In: *IEEE International Conference on Industrial Informatics (INDIN)*. 2021.

[Gud+19] Sumanth Gudaparthi, Surya Narayanan, Rajeev Balasubramonian, Edouard Giacomin, Hari Kambalasubramanyam **and** Pierre-Emmanuel Gaillardon. "Wire-Aware Architecture and Dataflow for CNN Accelerators". In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO '52. Columbus, OH, USA: Association for Computing Machinery, 2019.

[Haa+21] Bernhard Haas, Alexander Wendt, Axel Jantsch **and** Matthias Wess. "Neural Network Compression Through Shunt Connections and Knowledge Distillation for Semantic Segmentation Problems". In: *17th International Conference on Artificial Intelligence Applications and Innovations (AIAI)*. **june** 2021.

Bibliography III

[Kot+23] Thomas Kotrba, Martin Lechner, Omair Sarwar **and** Axel Jantsch. "Multispectral Feature Fusion for Deep Object Detection on Embedded Nvidia Platforms". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Antwerp, Belgium, **april** 2023.

[Lea+21] Isaac Sánchez Leal, Irida Shallari, Silvia Krug, Axel Jantsch **and** Mattias O'Nils. "Impact of Input Data on Intelligence Partitioning Decisions for IoT Smart Camera Nodes". In: *Electronics* 10.16 (2021).

[Lea+23] Isaac Sánchez Leal, Eiraj Saqib, Irida Shallari, Axel Jantsch, Silvia Krug **and** Mattias O'Nils. "Waist Tightening of CNNs: A Case study on Tiny YOLOv3 for Distributed IoT Implementations". In: *Proceedings of the Real-time And intelliGent Edge computing workshop (RAGE)*. San Antonio, Texas, **may** 2023.

[LJ21] Martin Lechner **and** Axel Jantsch. "Blackthorn: Latency Estimation Framework for CNNs on Embedded Nvidia Platforms". In: *IEEE Access* (2021).

[LJ24] Martin Lechner **and** Axel Jantsch. "Hardware-Aware Latency Pruning for Efficient Inference on Embedded GPUs". In: *Under submission* (2024).

Bibliography IV

[LJD19] Martin Lechner, Axel Jantsch **and** Sai M. P. Dinakarrao. “ResCoNN: Resource-Efficient FPGA-Accelerated CNN for Traffic Sign Classification”. In: *2019 Tenth International Green and Sustainable Computing Conference (IGSC)*. **october** 2019, **pages** 1–6.

[LJS22] Martin Lechner, Axel Jantsch **and** Lukas Steindl. “Study of DNN-based Ragweed Detection from Drones”. In: *Proceedings of International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*. Samos, Greece, **july** 2022.

[Lud23] Alexander Ludwig. “Optimization of Siamese Object Tracking Networks”. mathesis. Gusshausstrasse 27–29 / 384, 1040 Wien: TU Wien, **june** 2023.

[Rab22] Jan M. Rabaey. “Of Brains and Computers”. In: *Foundations and Trends in Integrated Circuits and Systems* 2.1–2 (2022), **pages** 1–192.

[Saq+23] Eiraj Saqib, Isaac Sánchez Leal, Irida Shallari, Axel Jantsch, Silvia Krug **and** Mattias O’Nils. “Optimizing the IoT Performance: A Case Study on Pruning a Distributed CNN”. In: *Proceedings of the IEEE Sensors Applications Symposium (SAS)*. 2023.

Bibliography V

[Sch+23a] Daniel Schnöll, Matthias Wess, Matthias Bittner, Maximilian Götzinger **and** Axel Jantsch. "Fast, Quantization Aware DNN Training for Efficient HW Implementation". In: *Proceedings of the 26th Euromicro Conference on Digital System Design (DSD)*. Durres, Albania, **september** 2023.

[Sch+23b] Daniel Schnöll, Domink Dallinger, Matthias Bittner **and** Axel Jantsch. *TinyML Design Contest - Team CDL EML TU Wien*. First Place and Winner of the TinyML Design contest at the International Conference on Computer Aided Design (ICCAD). San Francisco, California, USA, **november** 2023.

[SGM19] Emma Strubell, Ananya Ganesh **and** Andrew McCallum. "Energy and Policy Considerations for Deep Learning in NLP". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, **july** 2019, **pages** 3645–3650.

[Sha+21] Irida Shallari, Isaac Sánchez Leal, Silvia Krug, Axel Jantsch **and** Mattias O'Nils. "Design space exploration on IoT node: Trade-offs in processing and communication". In: *IEEE Access* (2021).

Bibliography VI

[SKO20] Irida Shallari, Silvia Krug **and** Mattias O'Nils. "Communication and computation inter-effects in people counting using intelligence partitioning". In: *Real-Time Image Processing* 17 (2020), **pages** 1869–1882.

[SRC15] SIA - SRC. *Rebooting the IT Revolution: A Call to Action*. techreport. Semiconductor Industry Association **and** Semiconductor Research Corporation, **september** 2015.

[Tri+24] Charles Edison Tripp **andothers**. *Measuring the Energy Consumption and Efficiency of Deep Neural Networks: An Empirical Analysis and Design Recommendations*. 2024. arXiv: 2403.08151 [cs.LG].

[WDJ18] Matthias Wess, Sai Manoj Pudukotai Dinakarrao **and** Axel Jantsch. "Weighted Quantization-Regularization in DNNs for Weight Memory Minimization towards HW Implementation". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.10 (**october** 2018).

[Wen+23] Alexander Wendt **andothers**. "A Pedestrian Detection Case Study for a Traffic Light Controller". In: *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing - Software Optimizations and Hardware/Software Codesign*. **byeditor**Sudeep Pasricha **and** Muhammad Shafique. Springer, 2023, **pages** 75–96.

Bibliography VII

[Wes+21] Matthias Wess, Marco Ivanov, Christian Unger, Anvesh Nookala, Alexander Wendt **and** Axel Jantsch. "ANNETTE: Accurate Neural Network Execution Time Estimation With Stacked Models". In: *IEEE Access* 9 (2021), **pages** 3545–3556.

[Wes+23] Matthias Wess, Dominik Dallinger, Daniel Schnöll, Matthias Bittner, Maximilian Götzinger **and** Axel Jantsch. "Energy Profiling of DNN Accelerators". In: *Proceedings of the 26th Euromicro Conference on Digital System Design (DSD)*. Durres, Albania, **september** 2023.

[Wes+24] Matthias Wess, Daniel Schnöll, Dominik Dallinger, Matthias Bittner **and** Axel Jantsch. "Conformal Prediction based Confidence for Latency Estimation of DNN Accelerators: A Black-box Approach". In: *IEEE Access* (2024).

[WWP09] Samuel Williams, Andrew Waterman **and** David Patterson. "Roofline: an insightful visual performance model for multicore architectures". In: *Commun. ACM* 52.4 (**april** 2009), **pages** 65–76.

eml.ict.tuwien.ac.at

axel.jantsch@tuwien.ac.at