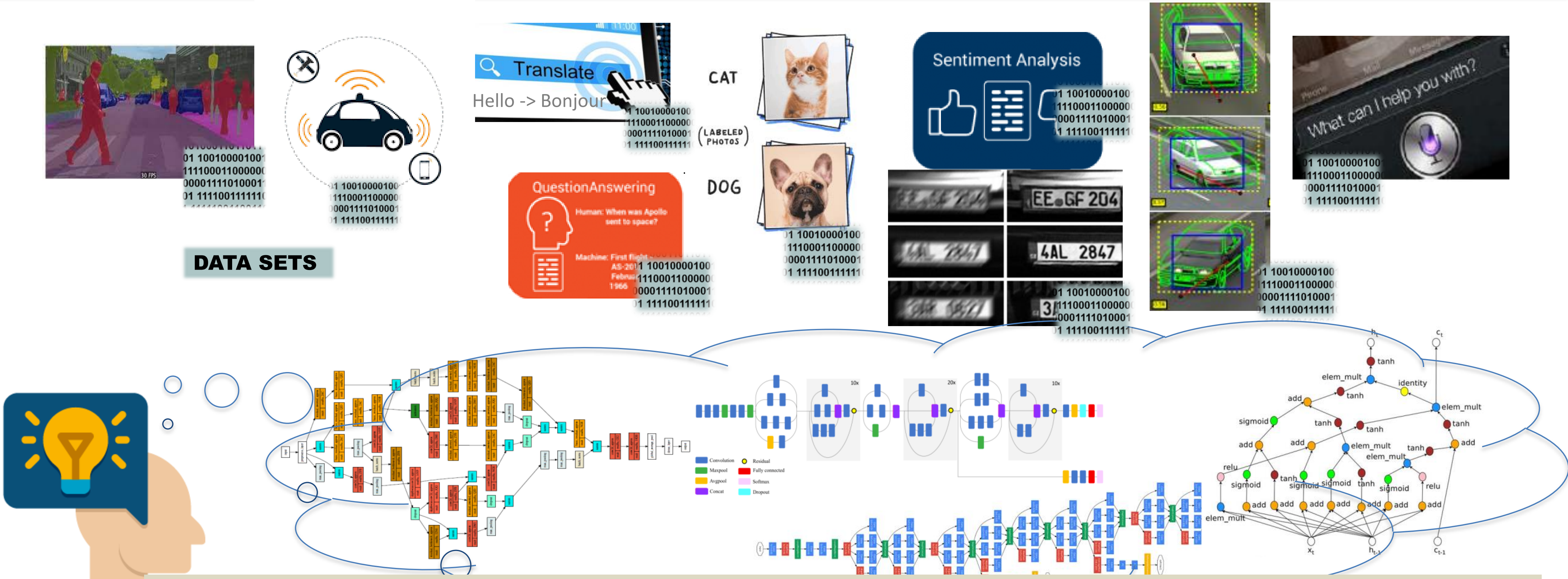
Evolutionary Hardware-Aware Neural Architecture Search

Lukáš Sekanina

Faculty of Information Technology
Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
sekanina@fit.vutbr.cz
https://www.fit.vut.cz/person/sekanina

ScaDS.AI ∴ January 18, 2023 ∴ Leipzig

Deep neural networks everywhere

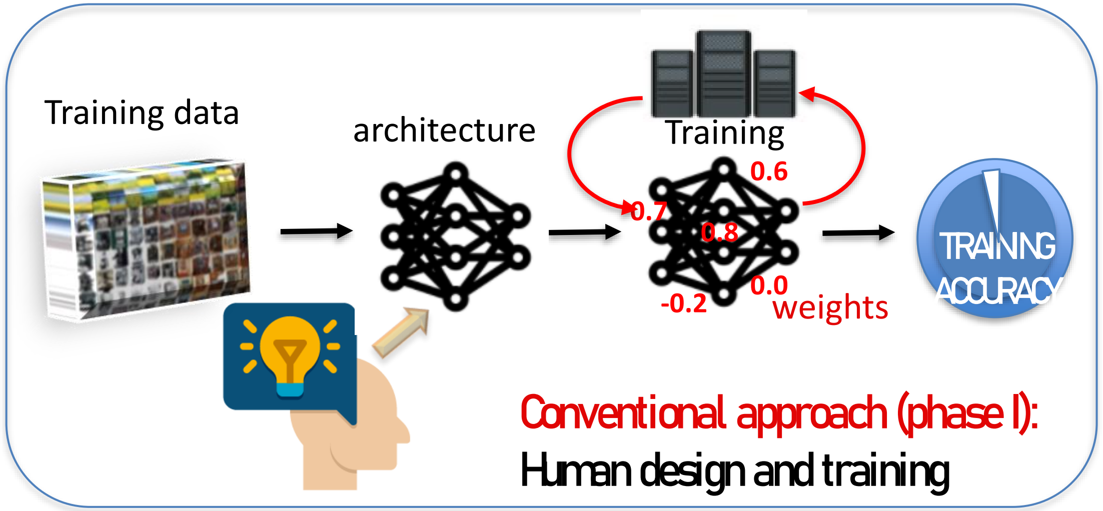


1. What is the best NN model for a given task and hardware?

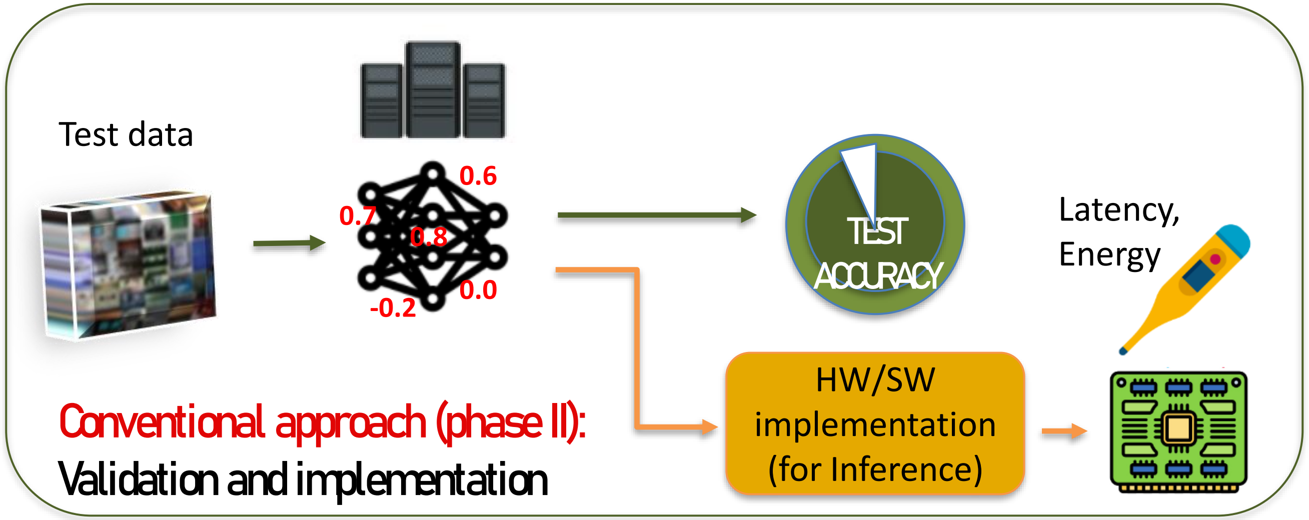
2. What is the best NN model and HW configuration for a given task?

Images taken from Internet.

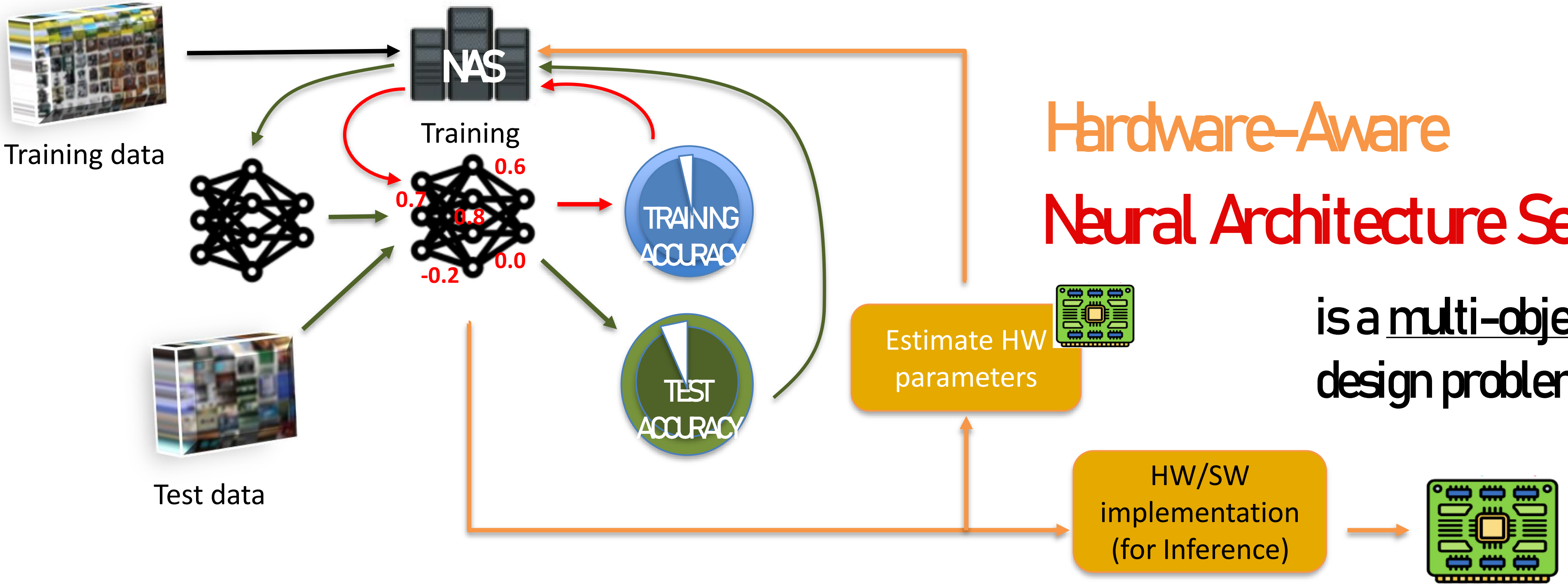
Neural Network Design and Neural Architecture Search (NAS)



Conventional approach (phase I):
Human design and training



Conventional approach (phase II):
Validation and implementation



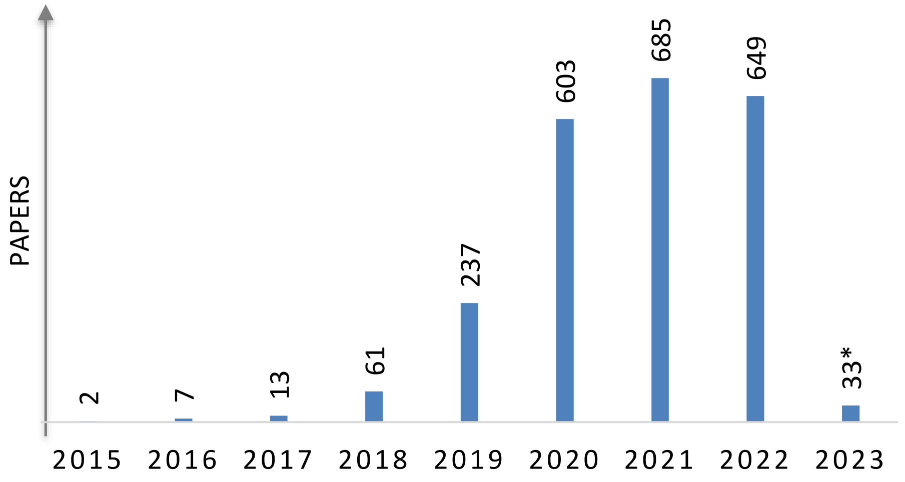
Hardware-Aware

Neural Architecture Search

is a <u>multi-objective</u> design problem!

Outline

- Efficient processing of CNNs
- Hardware accelerators for CNNs
- Single-objective Neural Architecture Search (NAS)
  - Search spaces and Search algorithms
  - Performance predictors
  - Classification of NAS methods
- Multi-objective Neural Architecture Search
  - Hardware-aware NAS
  - NAS with HW Co-design
  - Classification of NAS methods
- Benchmarking of NAS methods
- EvoApproxNAS
- Conclusions



The number of papers on NAS according to automl.org (January 13, 2023)

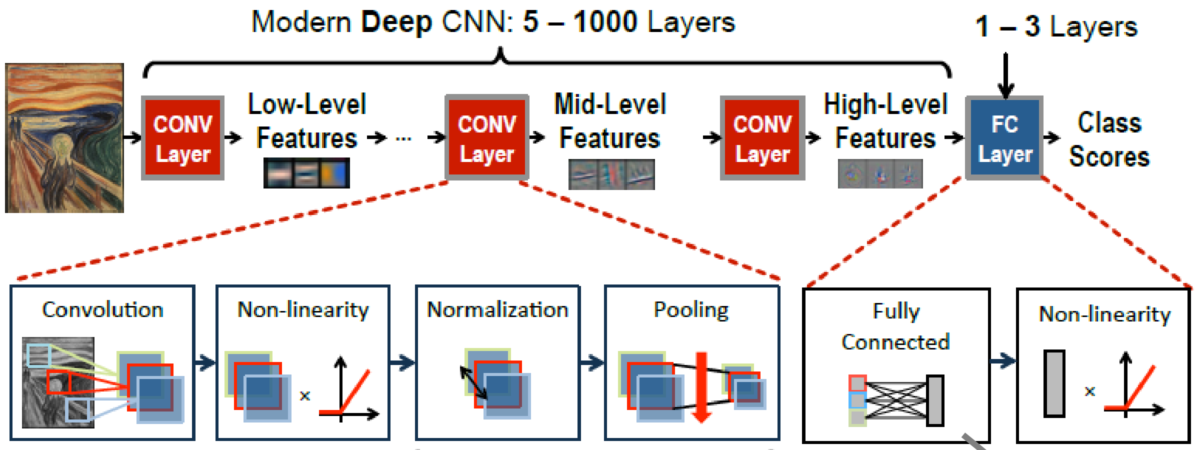
This talk is mostly based on:

Sekanina L.: Neural Architecture Search and Hardware Accelerator Co-Search: A Survey. IEEE Access, Vol. 9, 2021

Sze V. et al. Efficient processing of deep neural networks, Morgan & Claypool, 2020

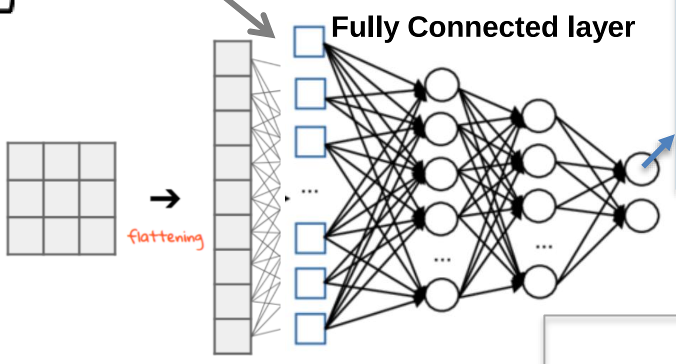
https://www.rle.mit.edu/eems/publications/tutorials/

Convolutional neural networks



Optional

Normalizing to (μ=0, σ=1).



Pooling

2x2 pooling, stride 2

9	3	5	3
10	32	2	2
1	3	21	9
2	6	11	7

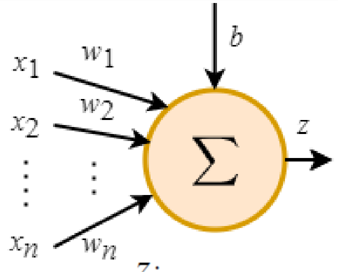
Max pooling

32	5
6	21

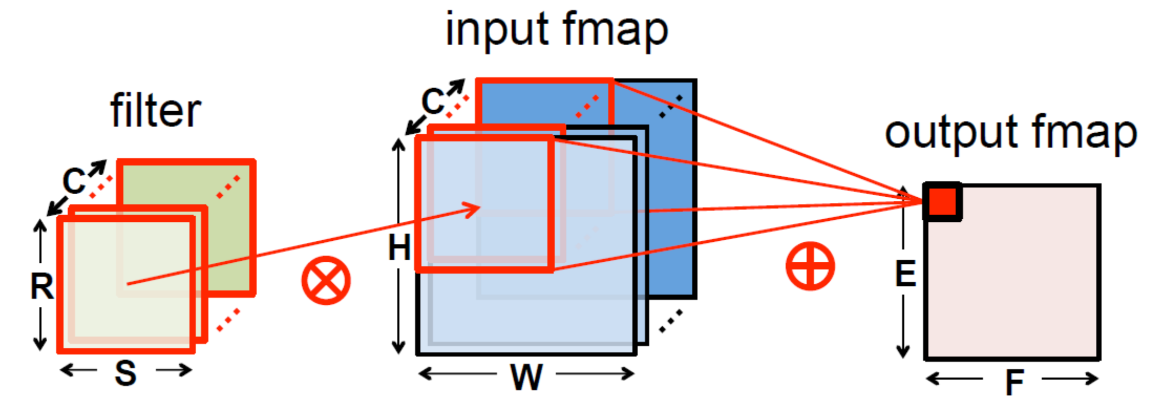
Average pooling

18	3
3	12

The output layer has k **such neurons** when classifying into k classes. The **softmax** gives the probability of i-th class:

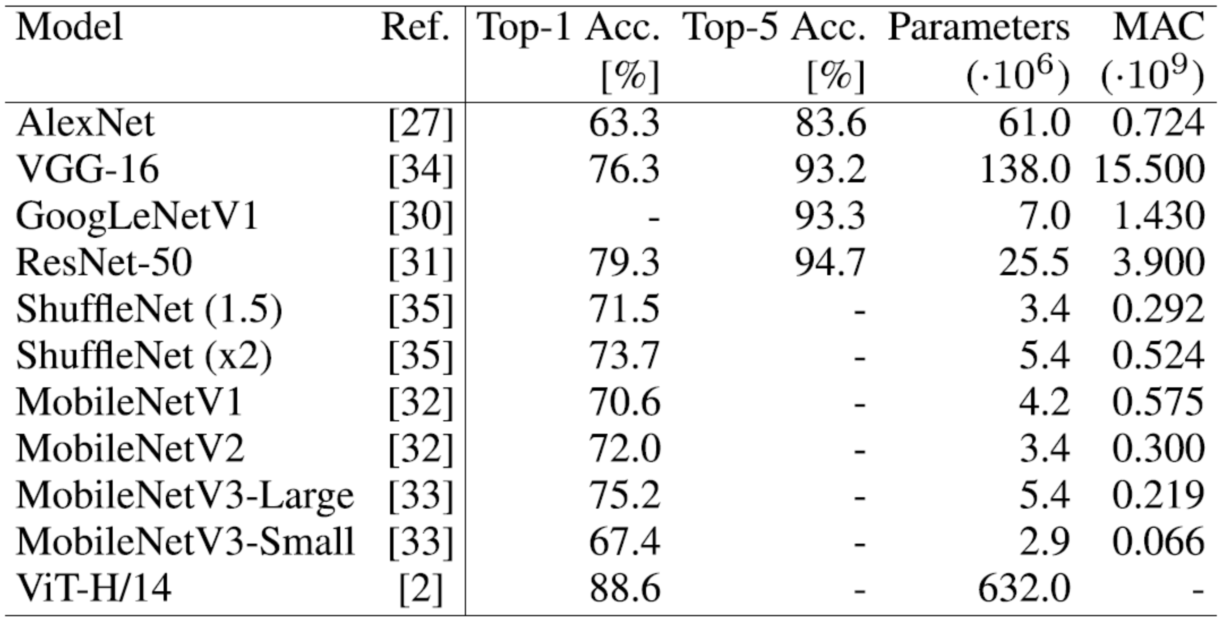


$$p(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$



Adopted from Sze et al. 2020

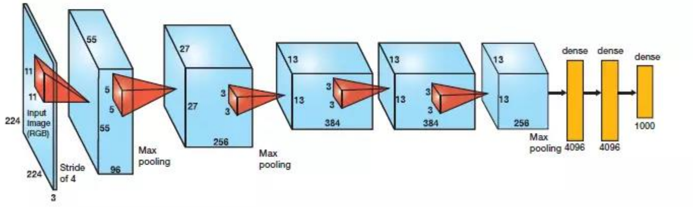
Parameters of popular hand-crafted CNNs (ImageNet)



Model	Ref.	Top-1 Acc. [%]	Top-5 Acc. [%]	Parameters $(\cdot 10^6)$	MAC $(\cdot 10^9)$
AlexNet	[27]	63.3	83.6	61.0	0.724
VGG-16	[34]	76.3	93.2	138.0	15.500
GoogLeNetV1	[30]	-	93.3	7.0	1.430
ResNet-50	[31]	79.3	94.7	25.5	3.900
ShuffleNet (1.5)	[35]	71.5	-	3.4	0.292
ShuffleNet (x2)	[35]	73.7	-	5.4	0.524
MobileNetV1	[32]	70.6	-	4.2	0.575
MobileNetV2	[32]	72.0	-	3.4	0.300
MobileNetV3-Large	[33]	75.2	-	5.4	0.219
MobileNetV3-Small	[33]	67.4	-	2.9	0.066
ViT-H/14	[2]	88.6	-	632.0	-

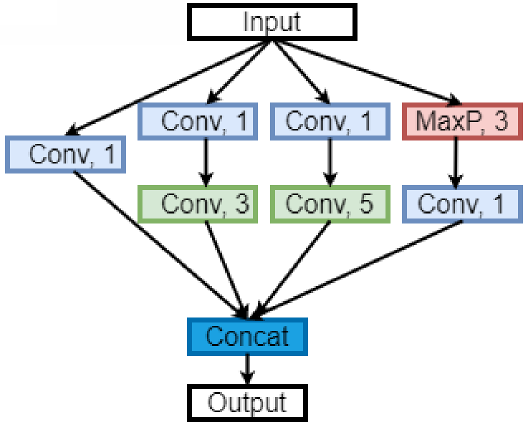
Rank	Model	Top-1 Accuracy	Parameters	Year
1	ViT-G/14	91.78%	1843M	2022
2	ViTAE-H	91.20%	644M	2022
7	FixEfficientNet-L2	90.90%	480M	2020
12	CvT-W24	90.60%	277M	2021
13	EfficientNet-L2	90.55%	480M	2019
14	ViT-L/16	90.54%	307M	2020
15	BiT-L	90.54%	928M	2019
18	Mixer-H/14- 448	90.18%	409M	2021
19	FixEfficientNet-B8	90.00%	87M	2020
21	FixResNeXt-101	89.73%	829M	2019
22	DeiT-B-384	89.30%	86M	2020
29	NASNet-A Large	87.56%		2017
41	ResNet-152	84.79%		2015
45	ResNet-50	82.94%	25M	2015

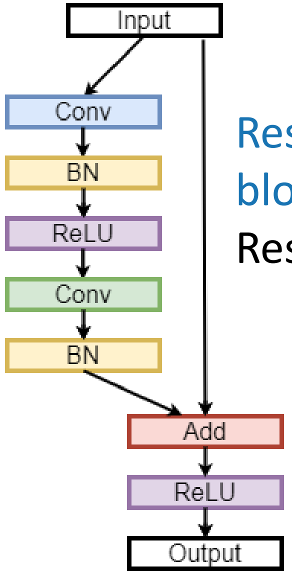
paperswithcode.com, April 4, 2022



Linear structure of AlexNet

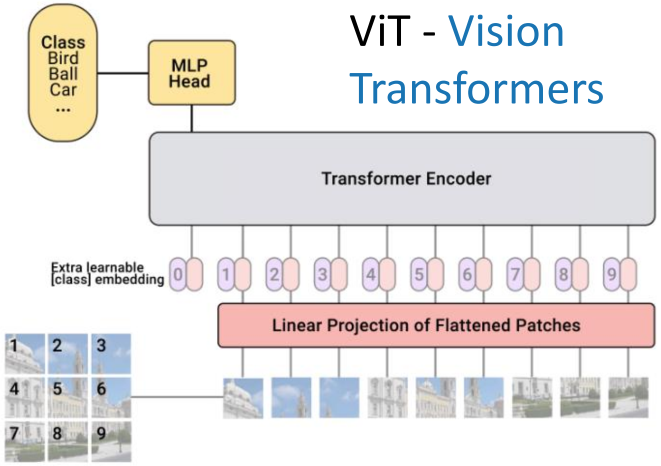
Inception block from GoogleNet





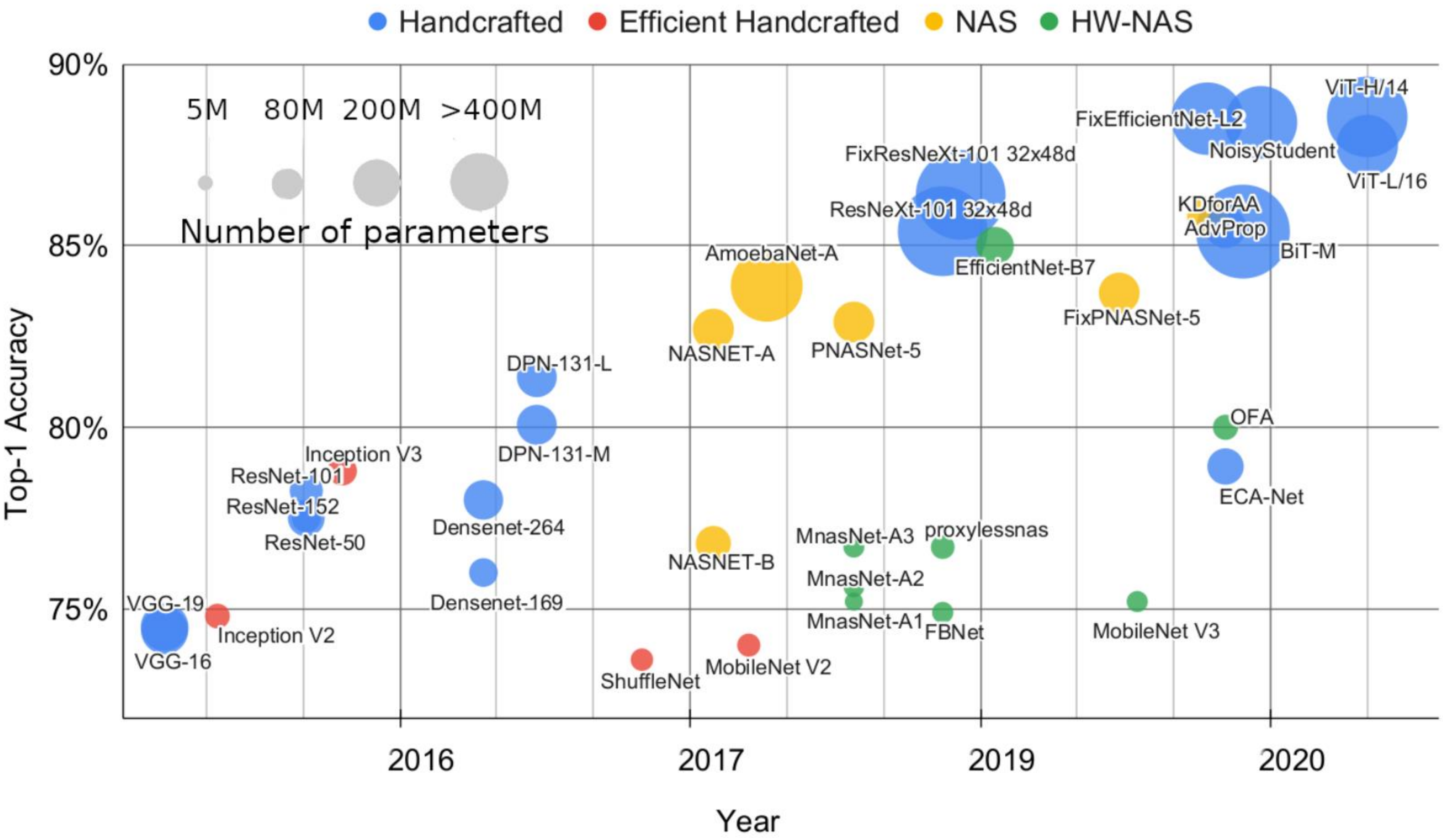
Residual block from ResNet

ViT - Vision Transformers


Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Extra learnable [class] embedding
0
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches
1
2
3
4
5
6
7
8
9

Development of CNN-based image classifiers

ImageNet:
256x256 pixel images
1000 classes
1.2M training images
100k test images



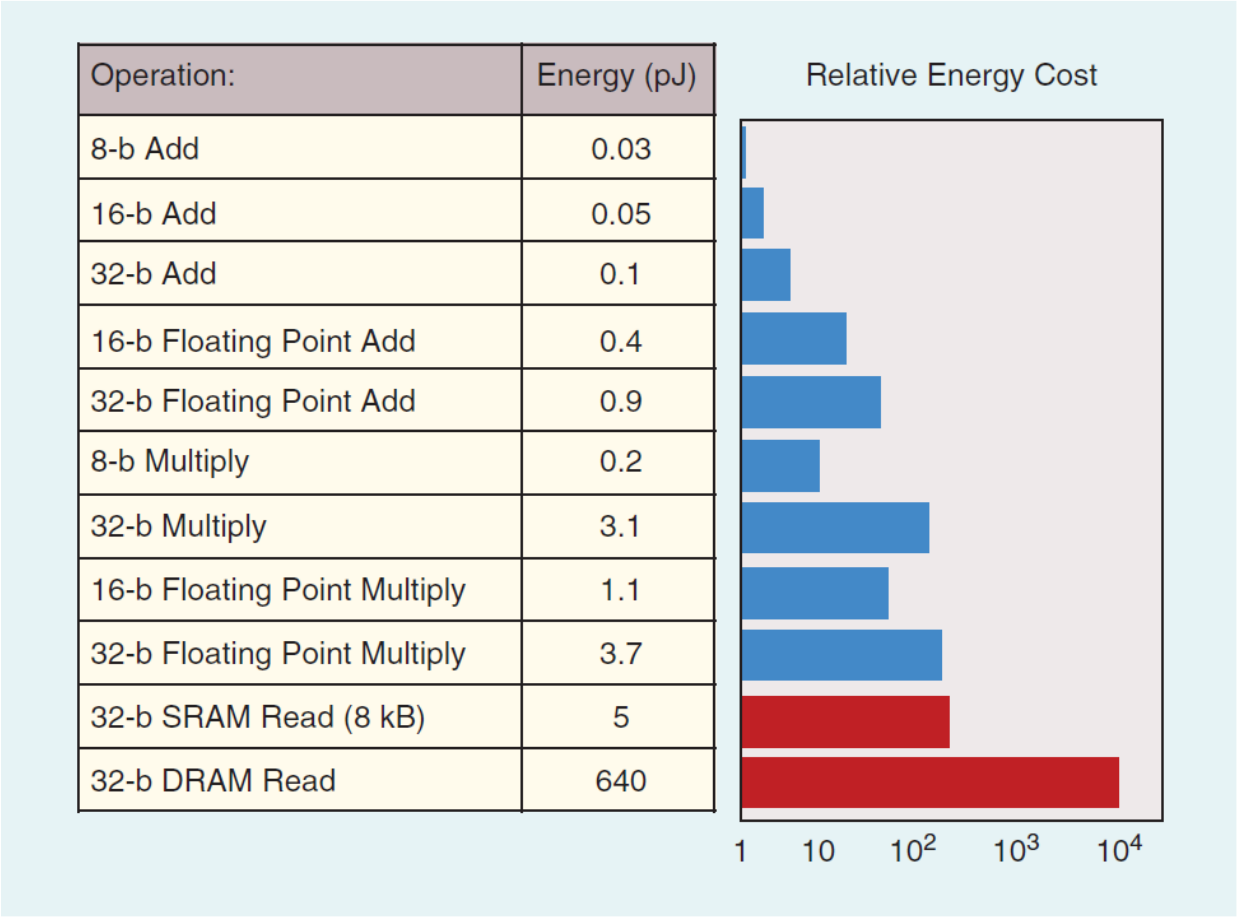
https://arxiv.org/abs/2101.09336

Efficient processing of CNNs (for inference)

Objective: Starting with a CNN model, develop its implementation(s) showing best tradeoffs between Accuracy and Latency/Energy/Memory Size for a given hardware.

Selected approaches

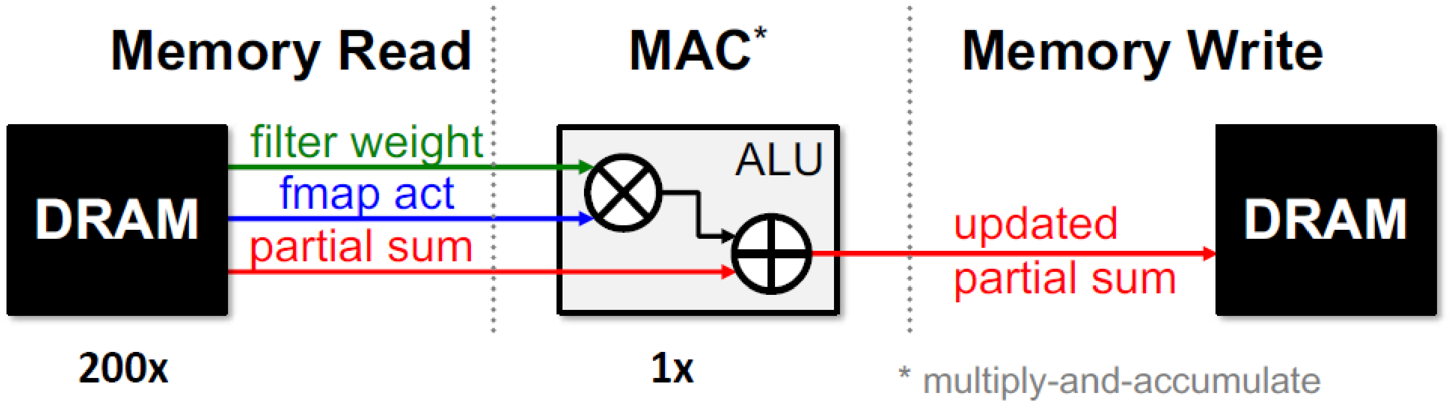
- Structure refinement
  - Optimized matrix multiplication, reducing the number of MACs, …
  - Compute reuse
- Data-oriented refinement
  - Quantization
  - Weight sharing
  - Pruning
- Operator refinement
  - Dedicated operators
  - Approximate operators



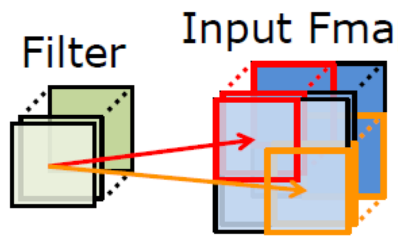
Operation:	Energy (pJ)
8-b Add	0.03
16-b Add	0.05
32-b Add	0.1
16-b Floating Point Add	0.4
32-b Floating Point Add	0.9
8-b Multiply	0.2
32-b Multiply	3.1
16-b Floating Point Multiply	1.1
32-b Floating Point Multiply	3.7
32-b SRAM Read (8 kB)	5
32-b DRAM Read	640

FIGURE 8: The energy consumption for various arithmetic operations and memory accesses in a 45-nm process. The relative energy cost (computed relative to the 8-b add) is shown on a log scale. The energy consumption of data movement (red) is significantly higher than arithmetic operations (blue). SRAM: static random-access memory. (Source: Adapted from [30].)

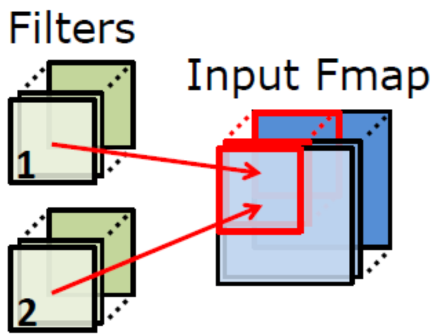
Memory Access is the Bottleneck



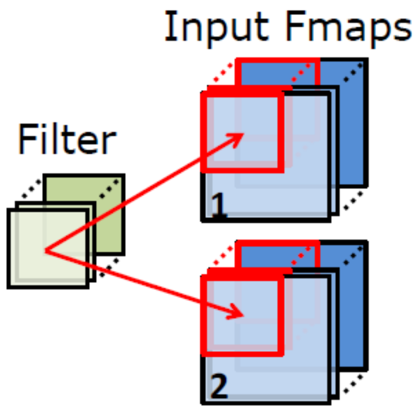
Example:
No reuse: **2896M** DRAM accesses required for AlexNet (724M MACs).



Convolutional Reuse
(Activations, Weights)
CONV layers only
(sliding window)



Fmap Reuse
(Activations)
CONV and FC layers



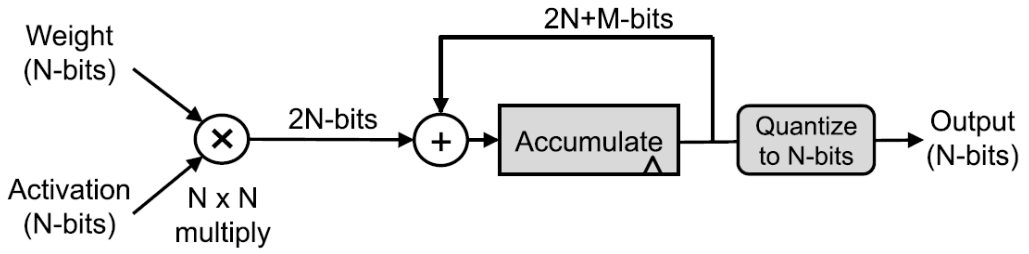
Filter Reuse
(Weights)
CONV and FC layers
(batch size > 1)

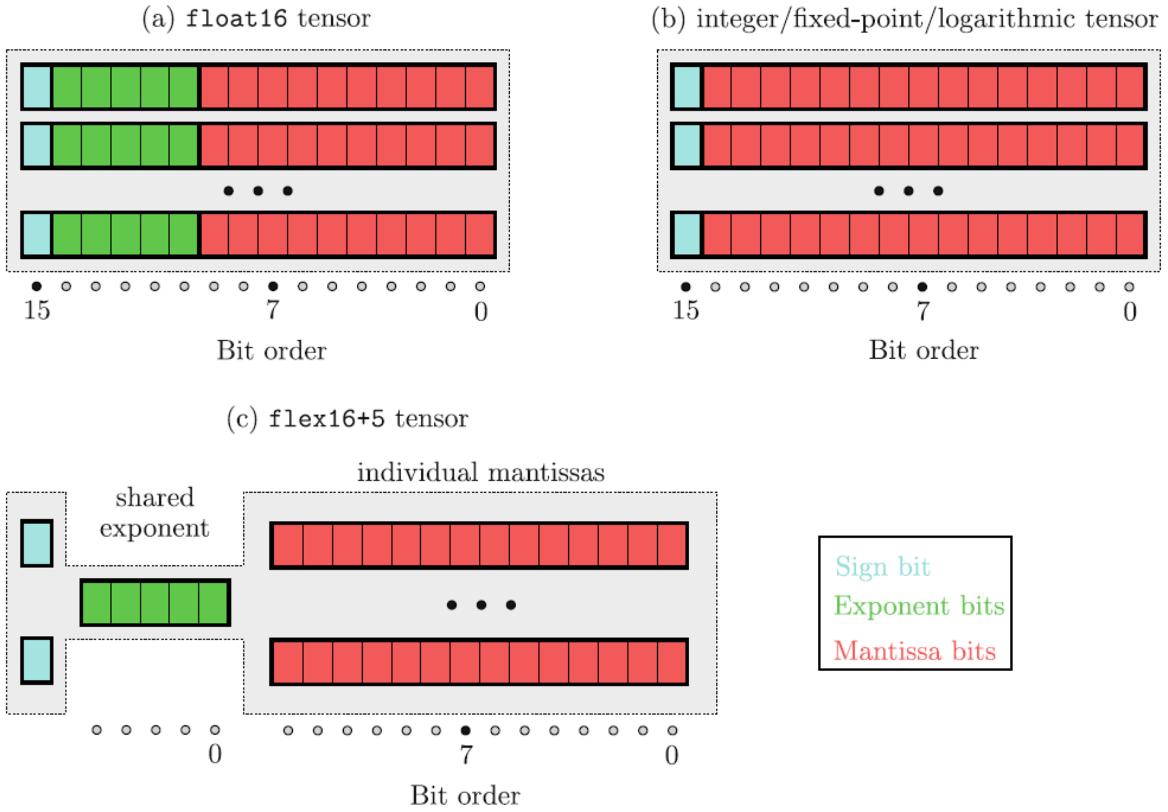
Input data reuse: **61M** DRAM accesses for AlexNet by exploiting low-cost local memory ~ 500x reduction

Adopted from Sze et al. 2020

Quantization and bit widths

- What to quantize
  - Inference: weights, activations, partial sums
  - Training: weights, activations, partial sums, gradients, weight update
- When
  - quantization-aware training
  - fine-tuning (training data required)
  - post-training quantization without any fine tuning
- Where
  - uniform (the entire NN)
  - non-uniform (layer level, channel level, …)
- Data formats
  - Floating point 8/16/32 bit; dynamic floating point
  - Fixed point/integer 4/8/16 bit
  - Binary and ternary {-c, 0, +c}, where c is a learnable parameter
  - Log representation
- Choosing quantized values
  - uniform
  - non-uniform (powers of 2, heuristically determined)
- Typical bit widths
  - 8 bit for inference (FX)
  - 16 bit for training (FP)



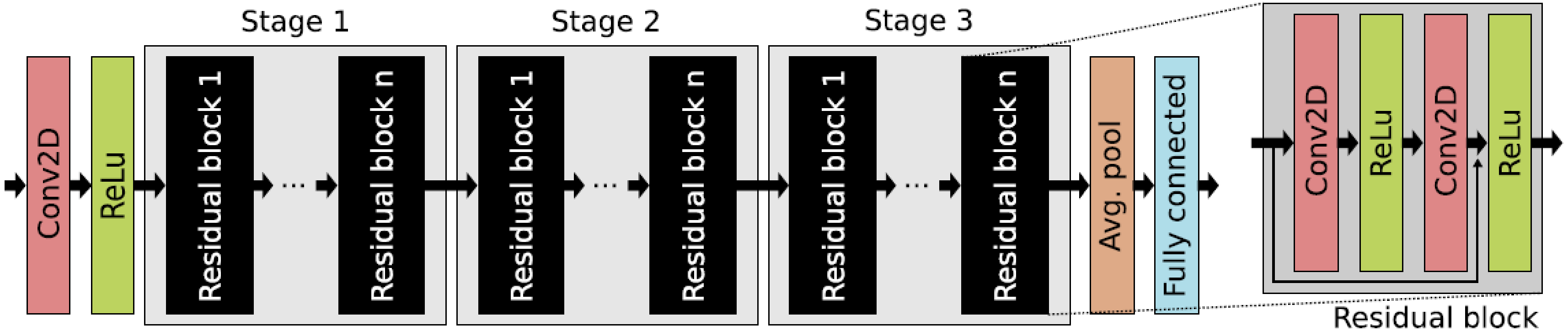

(a) float16 tensor

(b) integer/fixed-point/logarithmic tensor

(c) flex16+5 tensor

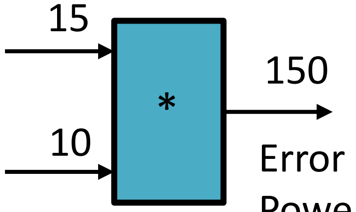
BFP – Block Floating Point

Quantization and approximation in ResNet



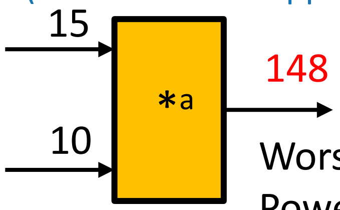
	ResNet instance	# conv. layers	mults $\times 10^6$	accuracy [%] (floating-point)	accuracy [%] (qint-8)
CIFAR-10	ResNet-8	7	21.1	83.42	82.85
	ResNet-14	13	35.3	85.93	85.81
	ResNet-20	19	49.5	88.32	88.09
	ResNet-26	25	63.6	90.05	89.70
CIFAR-100	ResNet-164 v2	163	592.6	74.46	74.27

Standard 8-bit multiplier



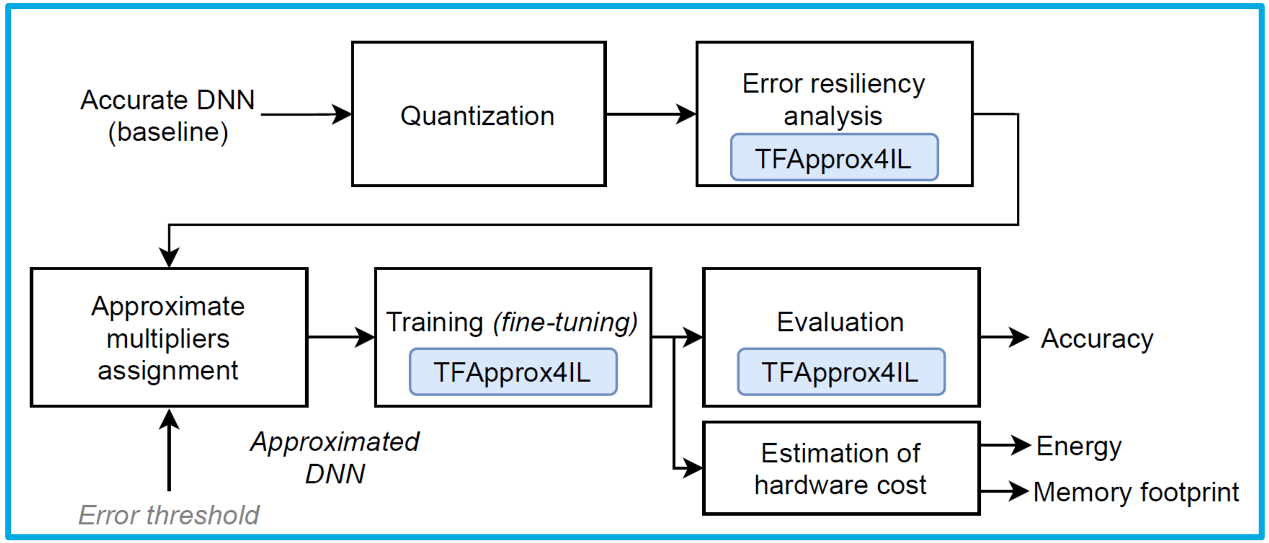
Error = 0
Power = 100%
Latency = 100%

Approximate 8-bit multiplier
(taken from EvoApproxLib)



Worst Error = 5%
Power = 61%
Latency = 82%

TFApprox enables the use of approx. multipliers in TensorFlow (GPU)

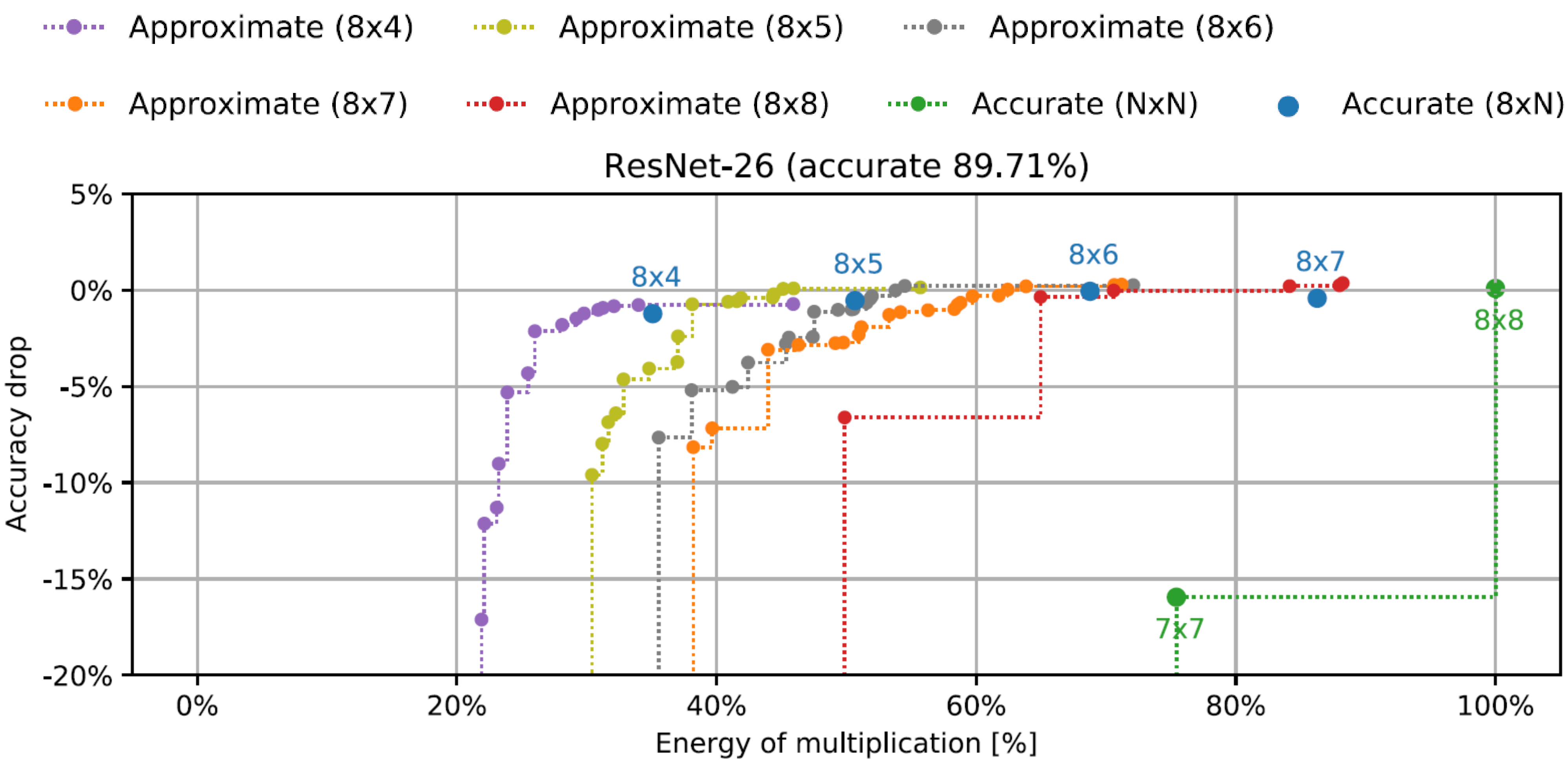


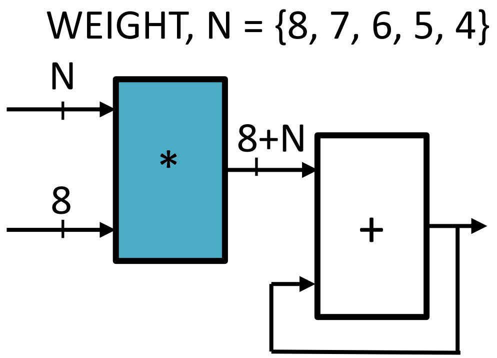
https://github.com/ehw-fit/tf-approximate

Mrázek, Sekanina et al. JETCAS 2020

Approximate multipliers in ResNet

- All exact 8-bit multiplications of all convolutional layers of different ResNet CNNs were replaced with one approximate implementation. Repeated for 35 different approximate 8-bit multipliers (from EvoApproxLib) to find Pareto fronts (Accuracy on CIFAR-10 vs. Energy).





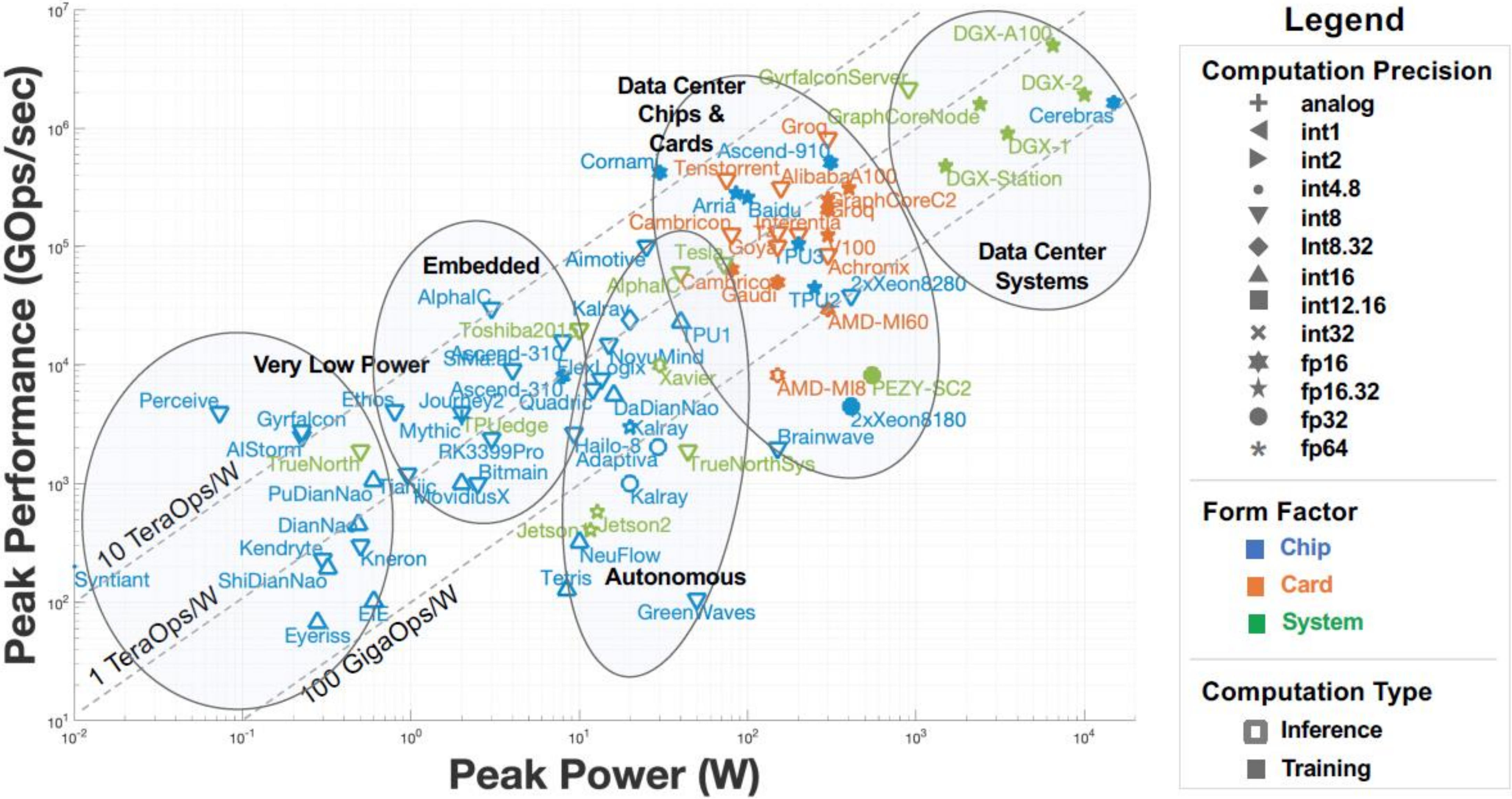
1. Improved the Accuracy-Energy tradeoffs
2. Reduced size of weight memory!

Mrázek, Sekanina et al. JETCAS 2020

Outline

- Efficient processing of CNNs
- Hardware accelerators for CNNs
- Single-objective Neural Architecture Search (NAS)
  - Search spaces and Search algorithms
  - Performance predictors
  - Classification of NAS methods
- Multi-objective Neural Architecture Search
  - Hardware-aware NAS
  - NAS with HW Co-design
  - Classification of NAS methods
- Benchmarking of NAS methods
- Conclusions

Accelerators for Machine Learning

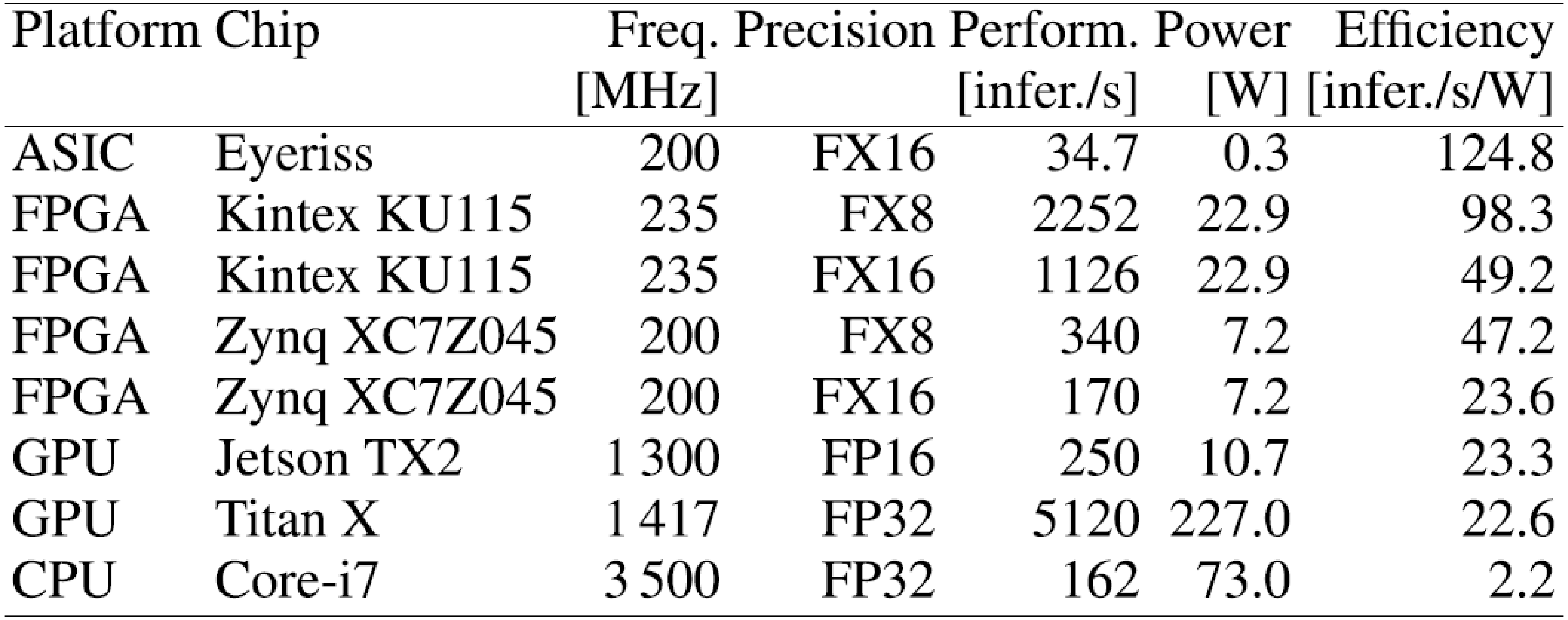


Reuther A. et al. arxiv.org/pdf/1908.11348.pdf

Accelerators for inference processing

Performance: the number of inferences per second

Energy-efficiency: the number of inferences per Watt/s



Platform	Chip	Freq. [MHz]	Precision	Perform. [infer./s]	Power [W]	Efficiency [infer./s/W]
ASIC	Eyeriss	200	FX16	34.7	0.3	124.8
FPGA	Kintex KU115	235	FX8	2252	22.9	98.3
FPGA	Kintex KU115	235	FX16	1126	22.9	49.2
FPGA	Zynq XC7Z045	200	FX8	340	7.2	47.2
FPGA	Zynq XC7Z045	200	FX16	170	7.2	23.6
GPU	Jetson TX2	1 300	FP16	250	10.7	23.3
GPU	Titan X	1 417	FP32	5120	227.0	22.6
CPU	Core-i7	3 500	FP32	162	73.0	2.2

AlexNet on various platforms

Unconventional platforms: in-memory computing, stochastic computing, memristive, RRAM, …

Two types of DNN accelerators

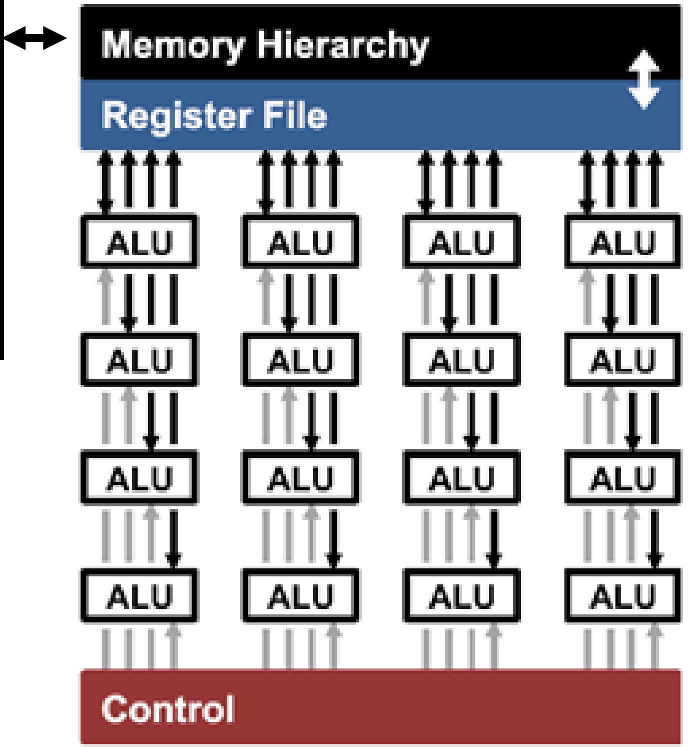
External memory (DRAM):

weights

inputs

results

Temporal Architecture



CPU, GPU

(for training & inference)

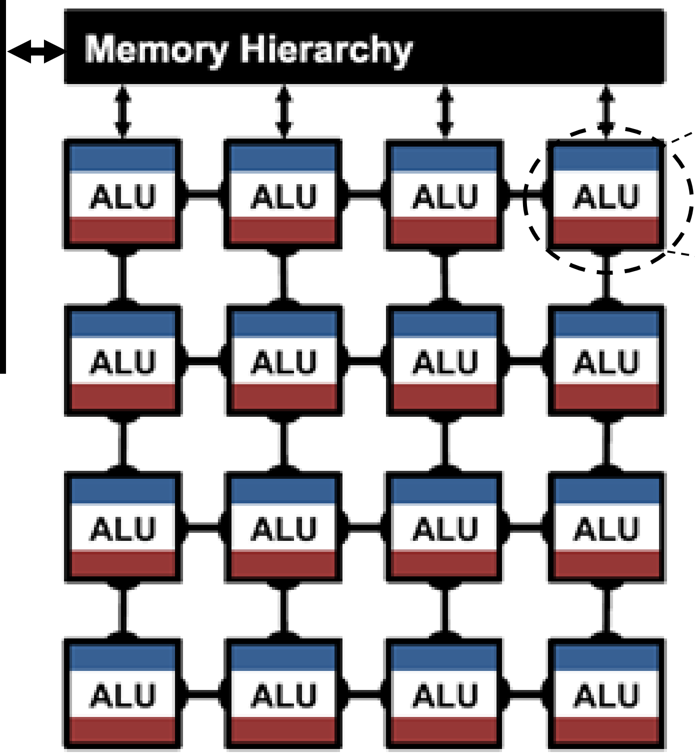
External memory (DRAM):

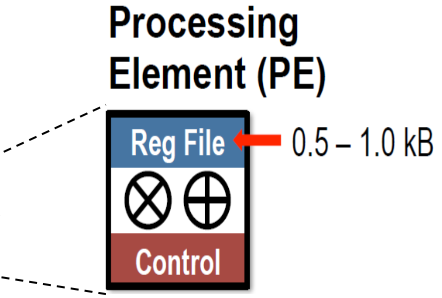
weights

inputs

results

Spatial Architecture





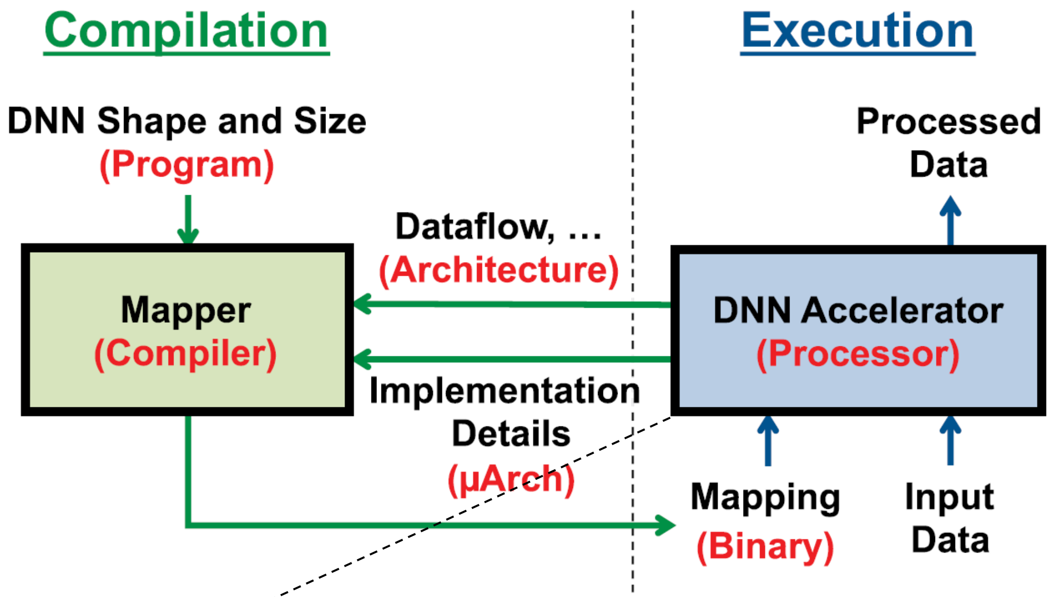
<u>Data flow strategy:</u> how reusing of input data (weights and activations) and local partial sums accumulation is implemented.

ASIC, FPGA

(usually for inference only)

Adopted from Sze et al. 2020

Example of an ASIC accelerator: Eyeriss (MIT, v1 2016; v2 2019)



Compilation

Execution

DNN Shape and Size (Program)

Mapper (Compiler)

Dataflow, … (Architecture)

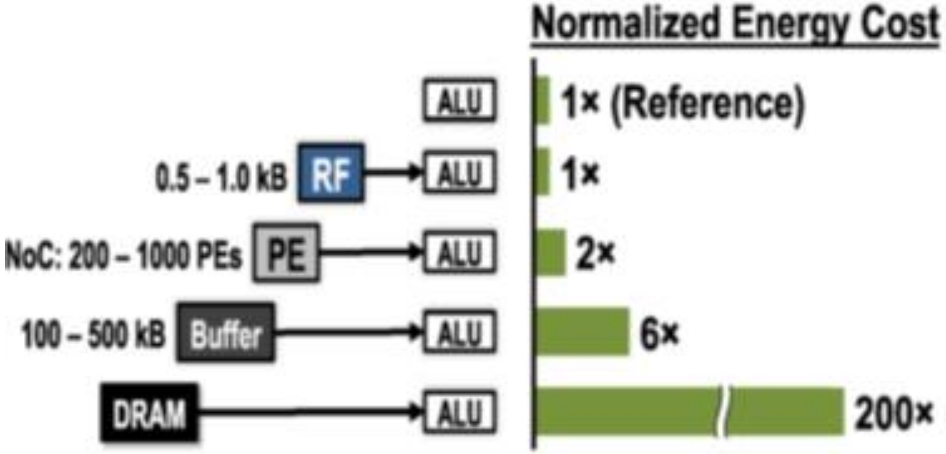
Implementation Details (µArch)

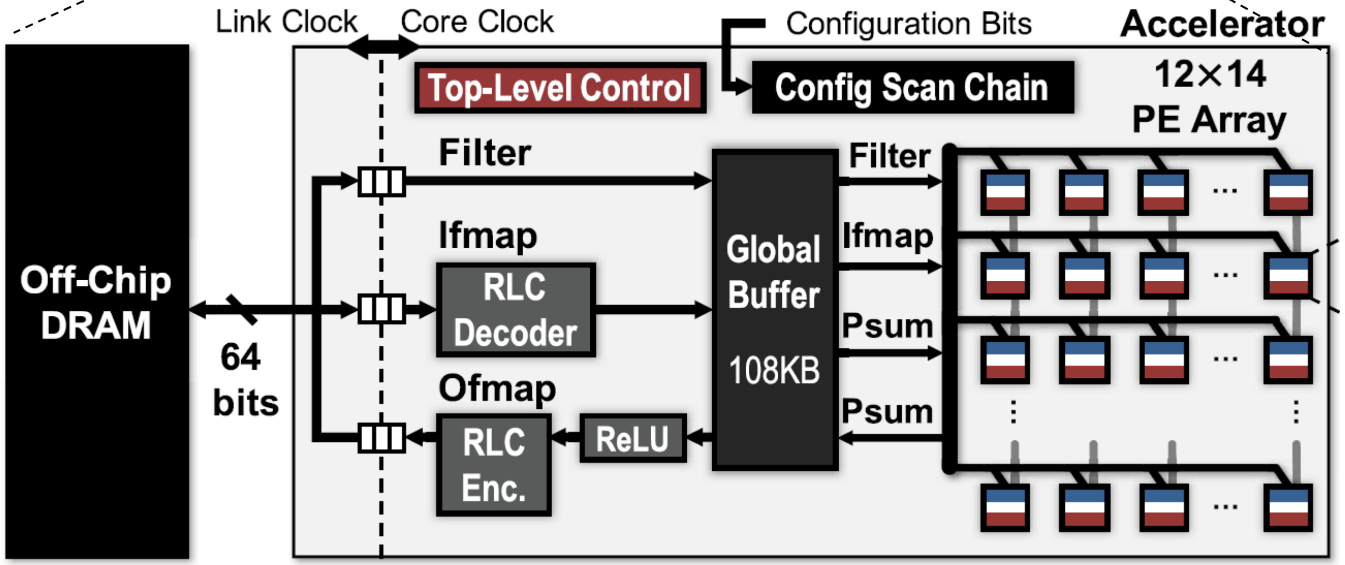
DNN Accelerator (Processor)

Processed Data

Mapping (Binary)

Input Data





Link Clock

Core Clock

Configuration Bits

Accelerator

Top-Level Control

Config Scan Chain

12×14 PE Array

Off-Chip DRAM

64 bits

Filter

Ifmap

RLC Decoder

Ofmap

RLC Enc.

ReLU

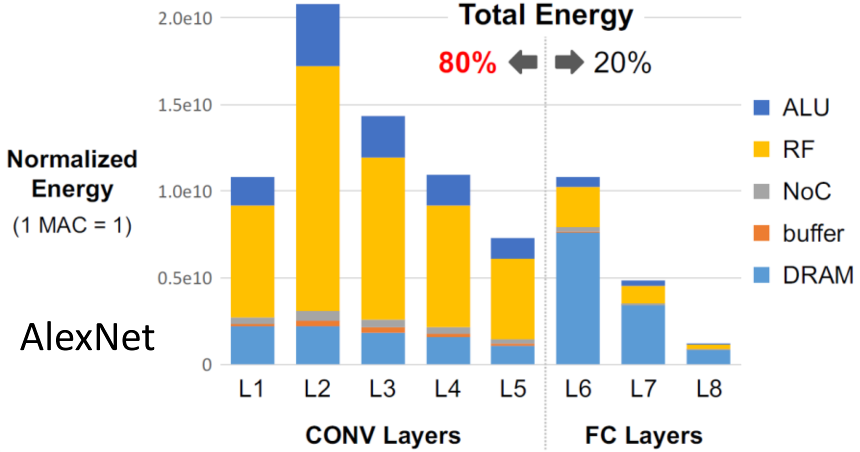
Global Buffer 108KB

Filter

Ifmap

Psum

Psum



Adopted from Sze et al. 2020

Outline

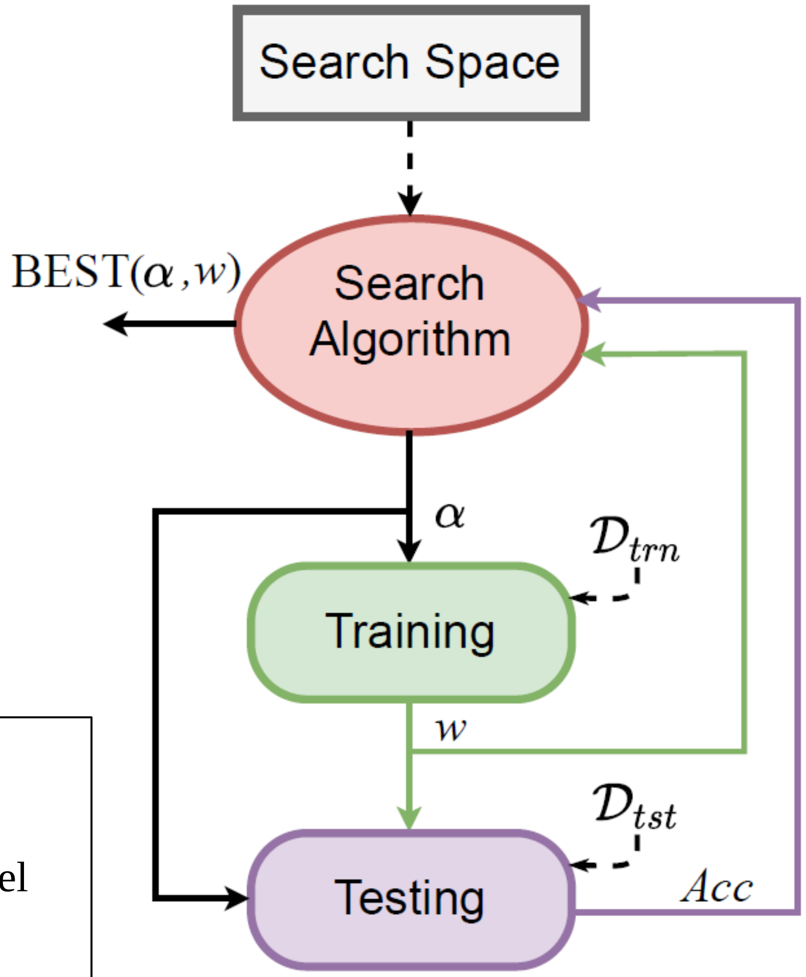
- Efficient processing of CNNs
- Hardware accelerators for CNNs
- Single-objective Neural Architecture Search (NAS)
  - Search spaces and Search algorithms
  - Performance predictors
  - Classification of NAS methods
- Multi-objective Neural Architecture Search
  - Hardware-aware NAS
  - NAS with HW Co-design
  - Classification of NAS methods
- Benchmarking of NAS methods
- Conclusions

Single-objective NAS

- The aim of NAS is to automate the process of finding the most suitable NN architecture for a given dataset. The single-objective NAS has one objective - maximizing the Accuracy.
  - Neuro-evolution has been performed in the Evolutionary Algorithms community since the mid-1980.
  - NAS has been connected with DNNs since 2016.
- Key components of NAS methods
  - Search space
  - Search algorithm
  - Performance estimation/evaluation
- Target hardware: usually GPU

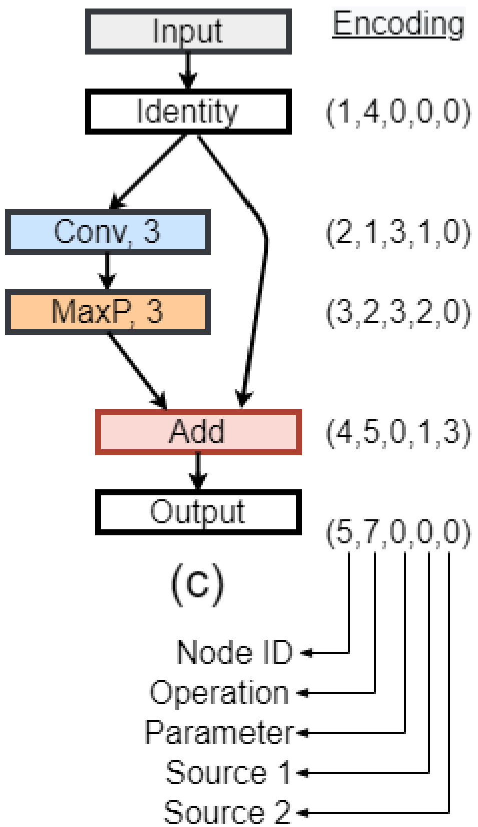
D_{trn} – training data
D_{tst} – test data
α – a candidate NN model
w – weights
Acc - Accuracy

Single-objective NAS (basic version)



Search Spaces and CNN encoding

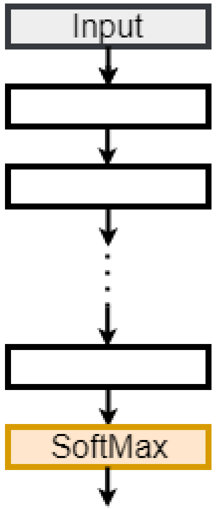
Candidate CNN ~ string of integers
Search space ~ all feasible strings



(Node ID; Operation; Parameter; Source ID 1; Source ID 2). Set of operations: (1) convolution, (2) max. pooling, (3) average pooling, (4) identity, (5) add, (6) concatenation, (7) terminal node [87].

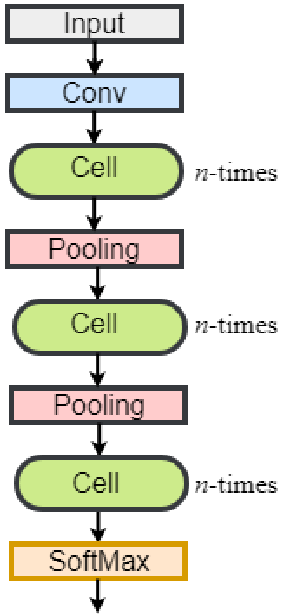
Macro search space

- The entire CNN is encoded.
- Some parts can be fixed by the designer.



Micro search space

- A subgraph (cell, block) or subgraphs is/are encoded and reused.



Hierarchical search space

- Recursive construction using a set of small graphs.

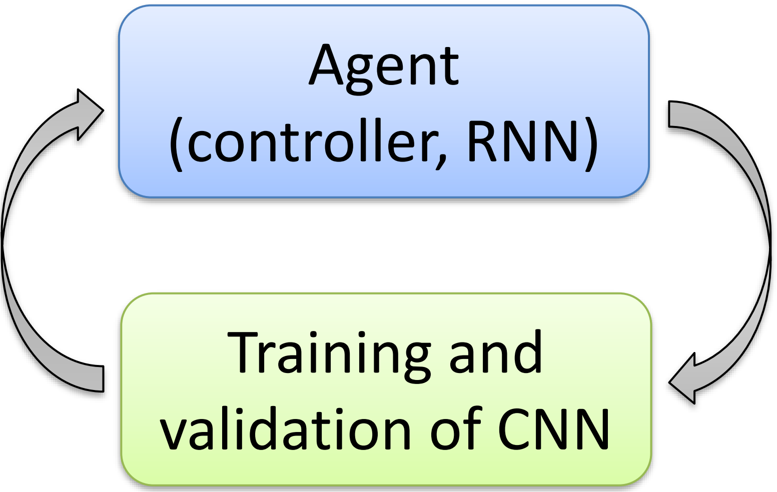
Indirect encoding

- A construction program is encoded.
- The program is executed to build a NN.

Recent survey:

Vargas-Hákim G.A. et al. A Review on Convolutional Neural Network Encodings for Neuroevolution. IEEE Tr. On Evol. Comp. 26(1), 2022

Search Algorithms: Reinforcement learning

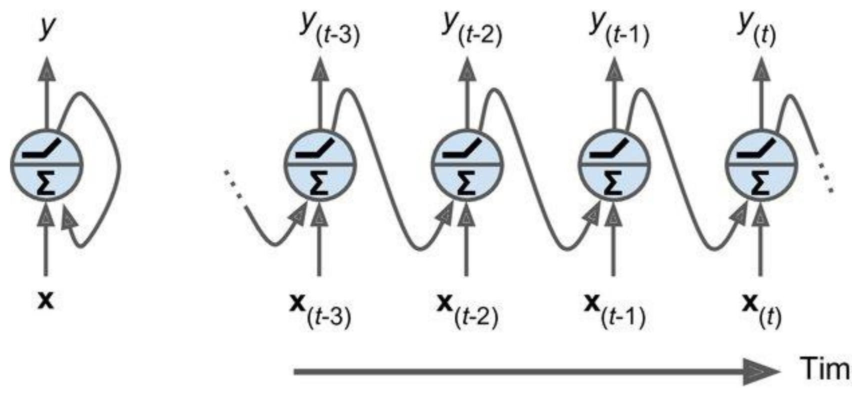


Agent (controller, RNN)

Training and validation of CNN

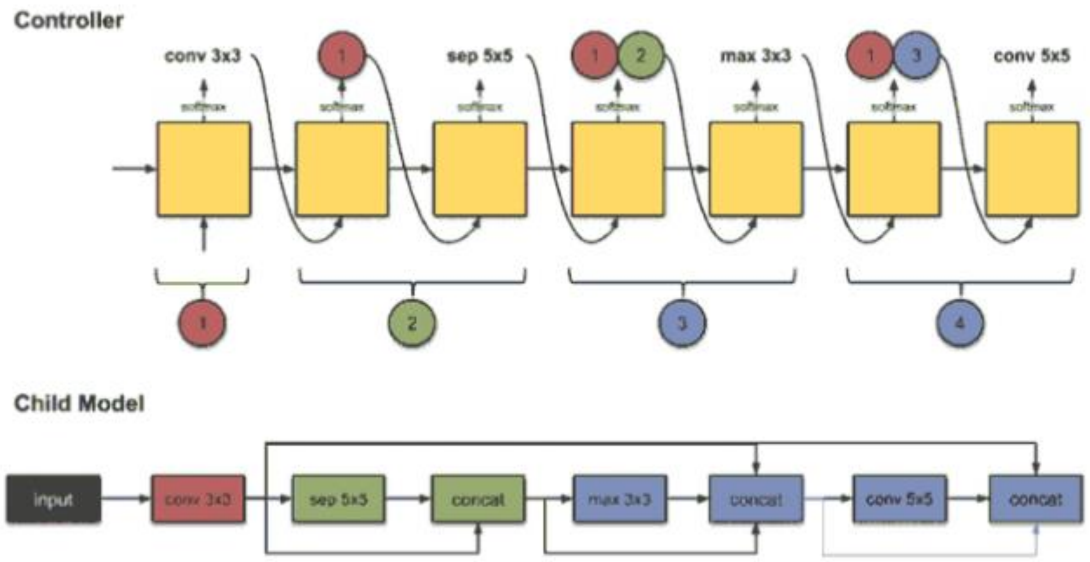
Reward: Accuracy (CNN) + Update RNN

Action: Sample CNN architecture (5, 7, 15, 6, …)

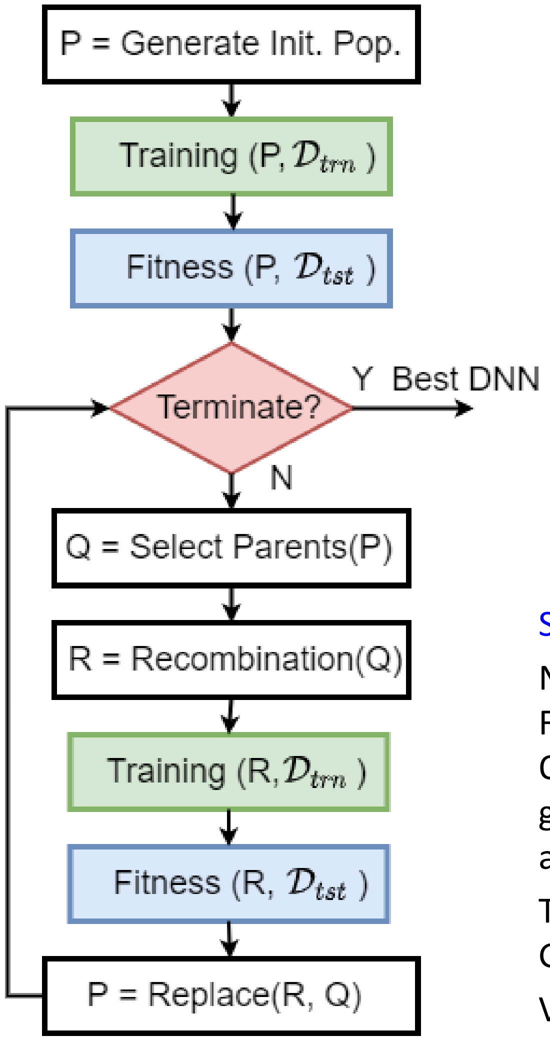


A recurrent neuron (left) unrolled through time (right)

- The agent’s action is the generation of a CNN architecture.
- The agent’s reward is the accuracy of CNN (obtained after training and validation on the test set).
- The agent (controller) is typically implemented as a recurrent neural network (RNN).
- The parameters of the RNN are optimized (using policy gradients techniques such as REINFORCE) in order to maximize the expected validation accuracy.



Search Algorithms: Genetic Algorithm



P = Generate Init. Pop.

Training (P, $\mathcal{D}_{trn}$)

Fitness (P, $\mathcal{D}_{tst}$)

Terminate?

Y Best DNN

N

Q = Select Parents(P)

R = Recombination(Q)

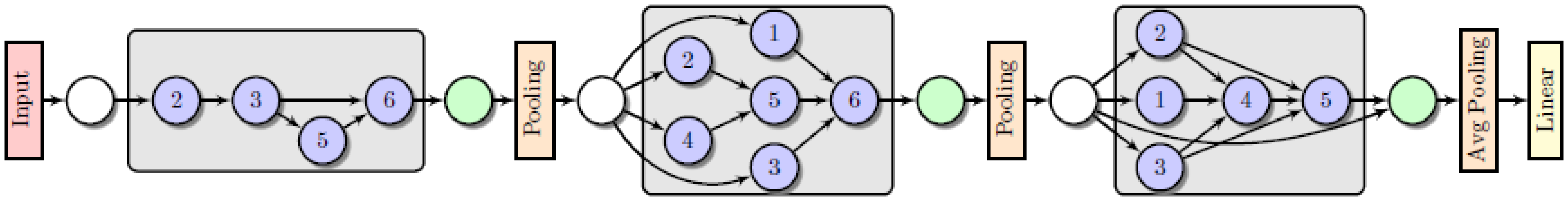
Training (R, $\mathcal{D}_{trn}$)

Fitness (R, $\mathcal{D}_{tst}$)

P = Replace(R, Q)

Example: Zhichao Lu et al. NSGA-Net, GECCO 2019

Encoding: CNN is a set of phases; max. 6 nodes in each phase encoded using a bit string. A node can be convolution, pooling, batch-normalization…



$\boldsymbol{x}_o^{(1)}$ =0-01-000-0010-00101-0

$\boldsymbol{x}_o^{(2)}$ =0-00-000-0101-10101-0

$\boldsymbol{x}_o^{(3)}$ =0-00-111-0111-00000-1

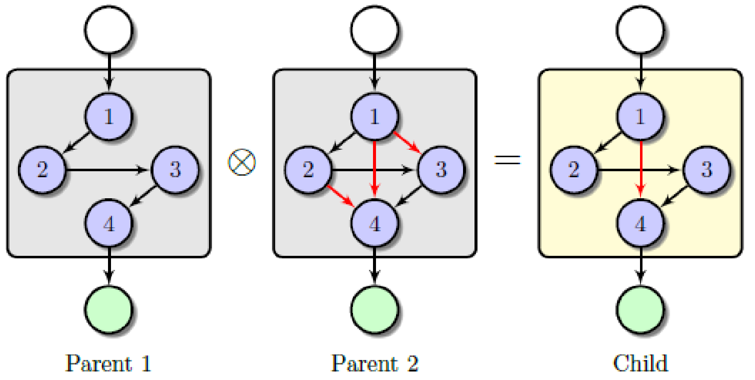
Search method:

NSGA-II (classification error vs the number of FLOPs), crossover, bit flip mutation, Bayesian Optimization Algorithm, population size = 40, generations = 20+10, i.e. 1200 network architectures are created in a single run

Training (during the evolution): SGD (Stochastic Gradient Descent) for 25 epochs

Validation of evolved CNNs: 600 epochs, batch size 96, data preprocessing, regularization …

Crossover:



(Parent 1) VGG:	1-01-001-0
(Parent 2) DenseNet:	1-11-111-0
(Child) ResNet:	1-01-101-0

⬛ : common

🟦 : VGG

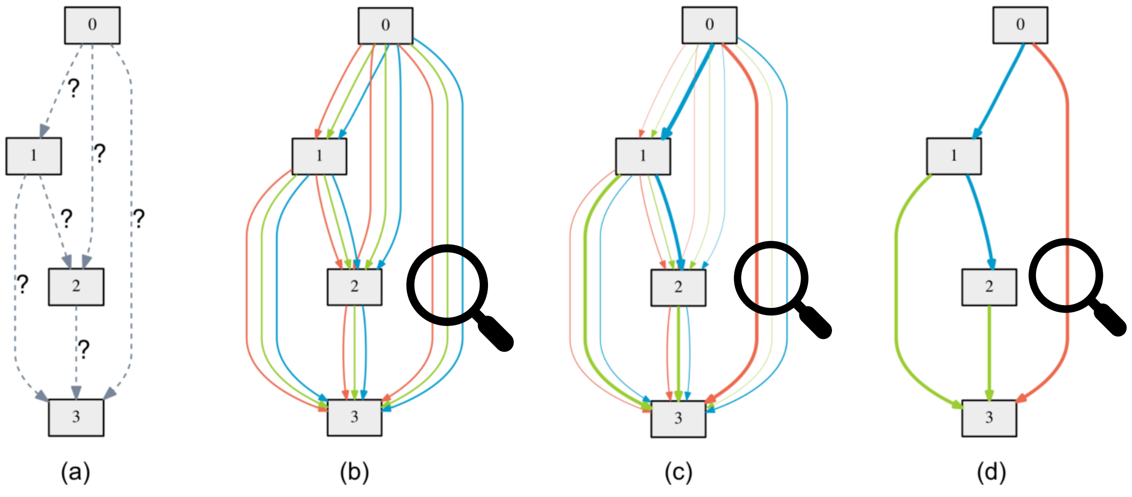
🟥 : DenseNet

Search Algorithms: Differentiable Neural Architecture Search

The weights w and continuous parameters α representing the NN architecture are jointly optimized by a gradient method.

Node $x^{(i)}$: Feature map

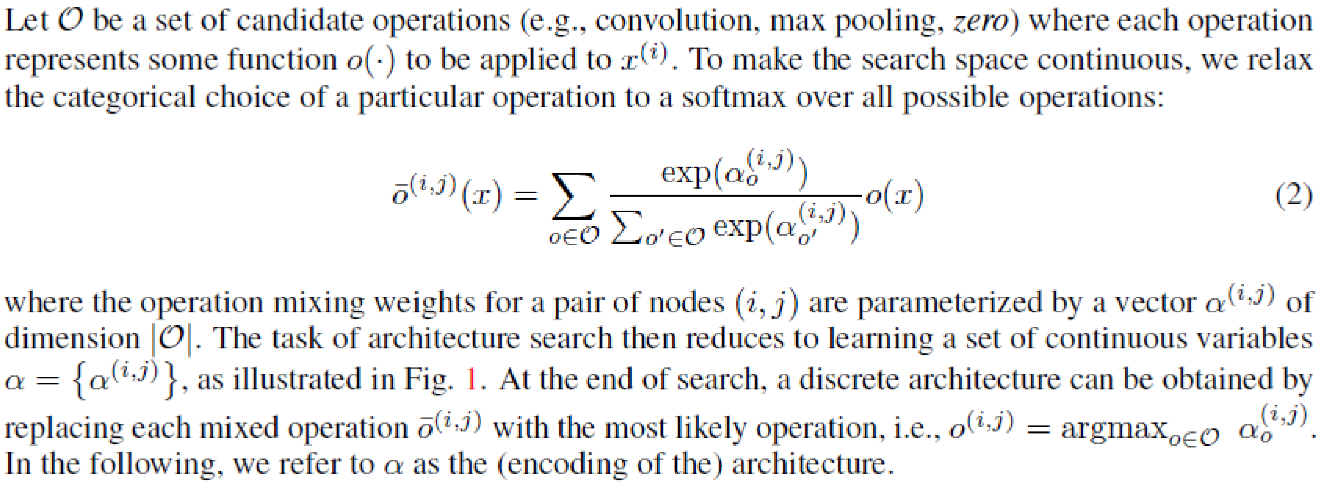
Edge $o^{(i,j)}$: Operation, e.g. Conv3, Conv5, AvgPool



1 discrete variable for 3 options

3 continuous variables α_i

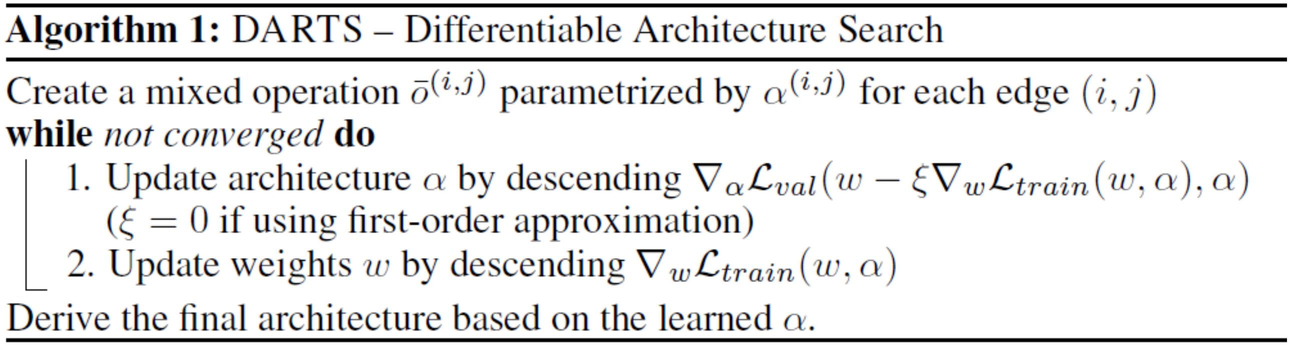
the most likely one is selected



Let $\mathcal{O}$ be a set of candidate operations (e.g., convolution, max pooling, *zero*) where each operation represents some function $o(\cdot)$ to be applied to $x^{(i)}$. To make the search space continuous, we relax the categorical choice of a particular operation to a softmax over all possible operations:

$$\bar{o}^{(i,j)}(x) = \sum_{o\in\mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o'\in\mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x) \tag{2}$$

where the operation mixing weights for a pair of nodes (i,j) are parameterized by a vector $\alpha^{(i,j)}$ of dimension $|\mathcal{O}|$. The task of architecture search then reduces to learning a set of continuous variables $\alpha = \{\alpha^{(i,j)}\}$, as illustrated in Fig. 1. At the end of search, a discrete architecture can be obtained by replacing each mixed operation $\bar{o}^{(i,j)}$ with the most likely operation, i.e., $o^{(i,j)} = \mathrm{argmax}_{o\in\mathcal{O}}\ \alpha_o^{(i,j)}$. In the following, we refer to α as the (encoding of the) architecture.



```
Algorithm 1: DARTS – Differentiable Architecture Search
Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge $(i,j)$
while not converged do
    1. Update architecture $\alpha$ by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi\nabla_w \mathcal{L}_{train}(w,\alpha),\alpha)$
       ($\xi = 0$ if using first-order approximation)
    2. Update weights $w$ by descending $\nabla_w \mathcal{L}_{train}(w,\alpha)$
Derive the final architecture based on the learned $\alpha$.
```

DARTS reduced the search time 10x-500x on ImageNet.

Comparison of single-objective NAS methods from 2018

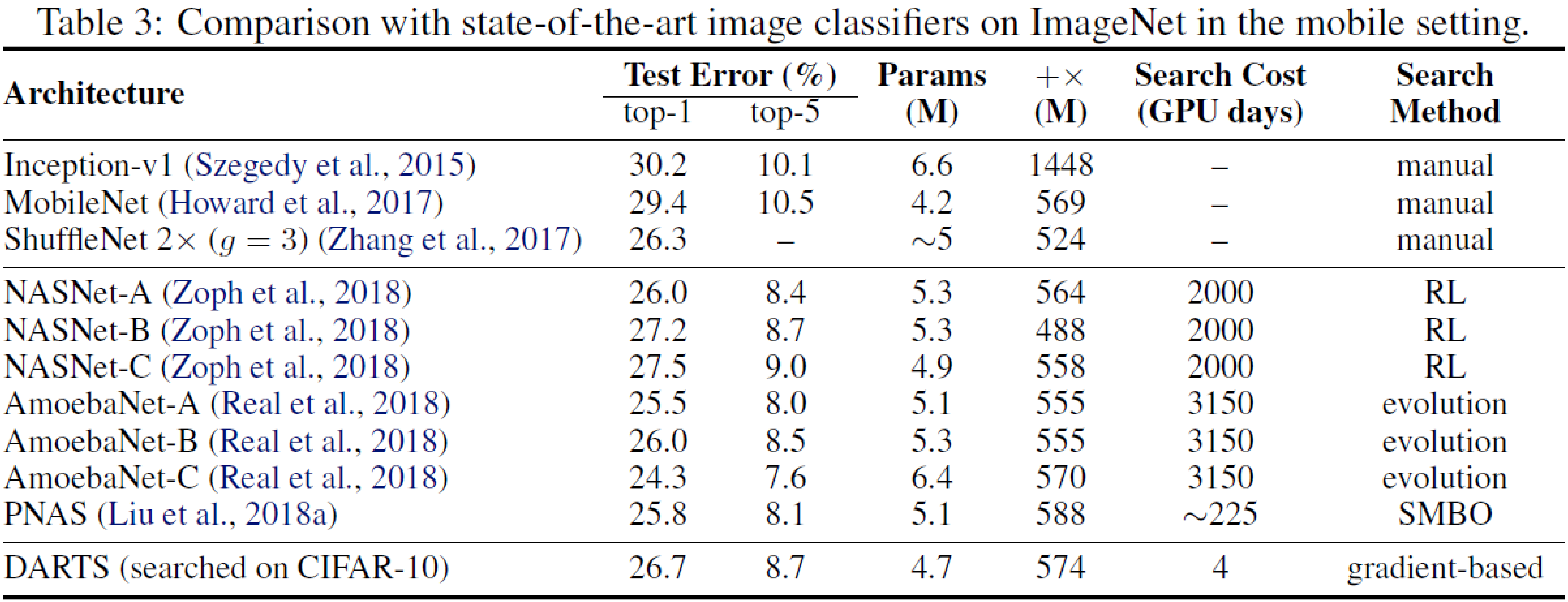


Table 3: Comparison with state-of-the-art image classifiers on ImageNet in the mobile setting.

Architecture	Test Error (%)		Params	+×	Search Cost	Search
	top-1	top-5	(M)	(M)	(GPU days)	Method
Inception-v1 (Szegedy et al., 2015)	30.2	10.1	6.6	1448	–	manual
MobileNet (Howard et al., 2017)	29.4	10.5	4.2	569	–	manual
ShuffleNet 2× ($g = 3$) (Zhang et al., 2017)	26.3	–	∼5	524	–	manual
NASNet-A (Zoph et al., 2018)	26.0	8.4	5.3	564	2000	RL
NASNet-B (Zoph et al., 2018)	27.2	8.7	5.3	488	2000	RL
NASNet-C (Zoph et al., 2018)	27.5	9.0	4.9	558	2000	RL
AmoebaNet-A (Real et al., 2018)	25.5	8.0	5.1	555	3150	evolution
AmoebaNet-B (Real et al., 2018)	26.0	8.5	5.3	555	3150	evolution
AmoebaNet-C (Real et al., 2018)	24.3	7.6	6.4	570	3150	evolution
PNAS (Liu et al., 2018a)	25.8	8.1	5.1	588	∼225	SMBO
DARTS (searched on CIFAR-10)	26.7	8.7	4.7	574	4	gradient-based

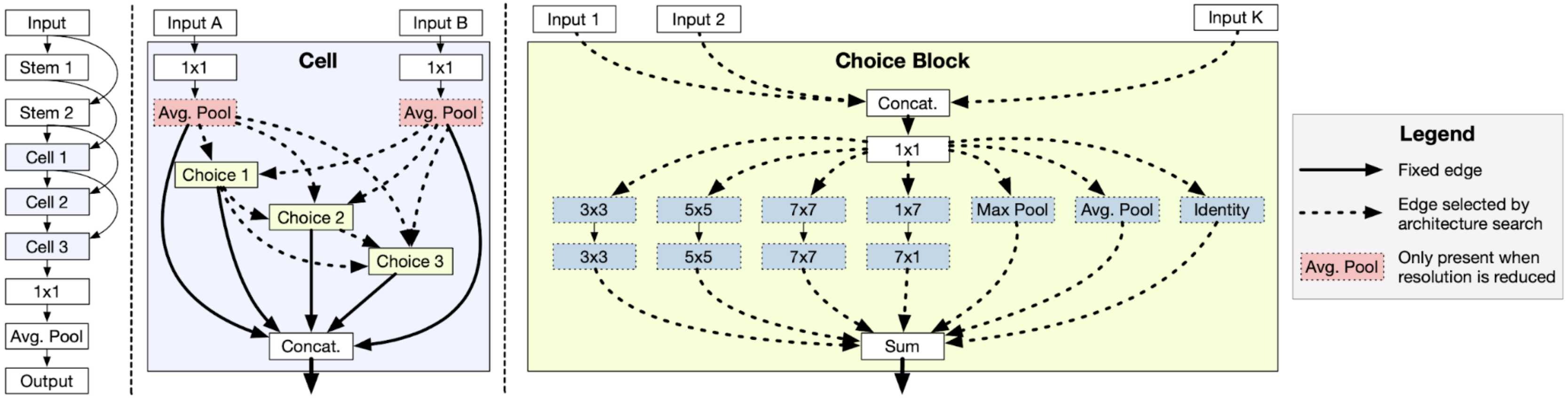
H. Liu, K. Simonyan, and Y. Yang, ``DARTS: Differentiable architecture search,'' ICLR 2019

SMBO – Sequential Model Based Optimization

Mobile setting: up to 600 million MACs per inference

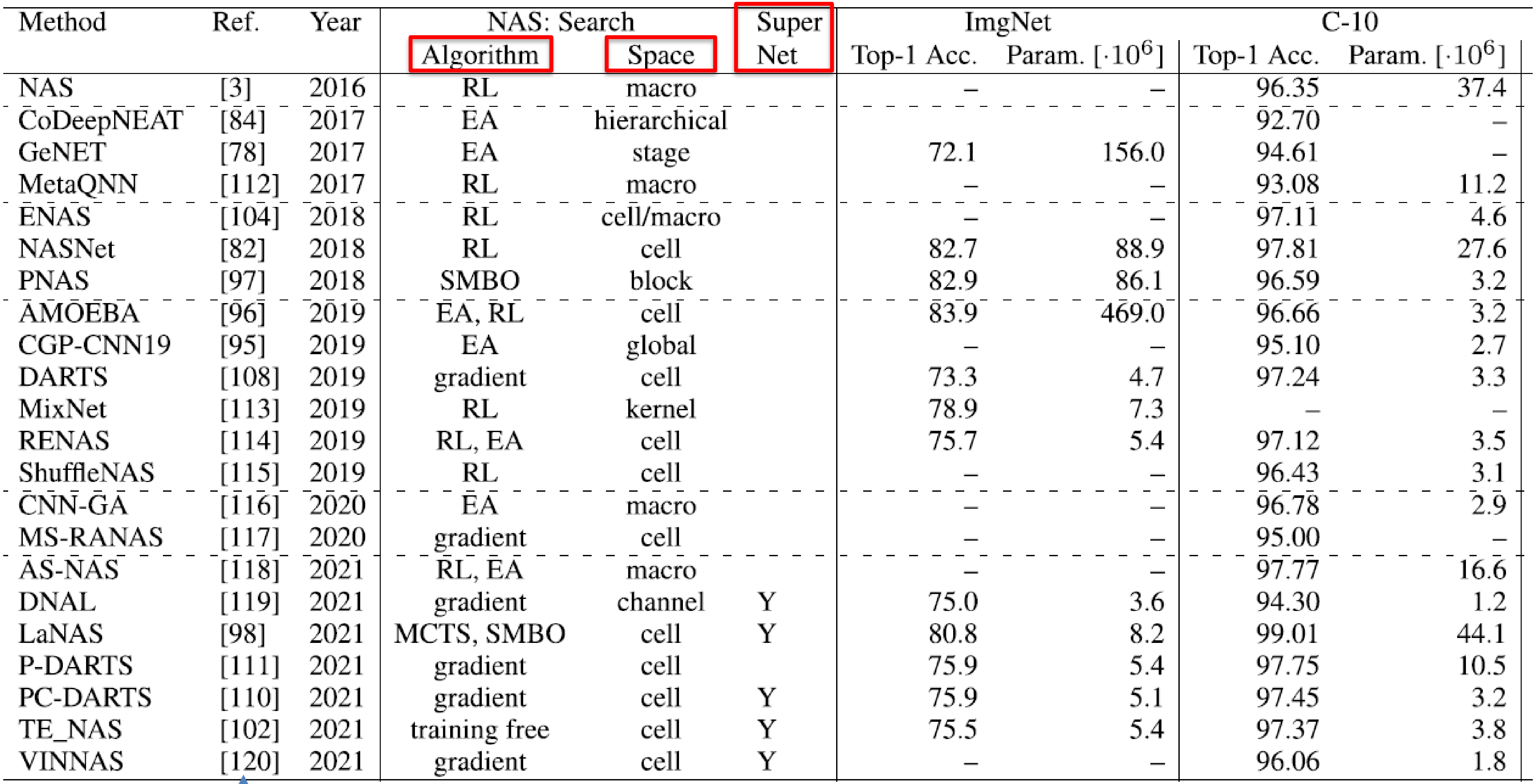
Supernet (one-shot network)

- **Idea**: Each candidate CNN could be seen as a **subnetwork** of a larger network.
- A single large over-parameterized network (**supernet**) is constructed such that it contains every possible operation in the search space.
- Once the supernet model is trained (which is very expensive!), it is used for evaluating the performance of many different architectures (subnetworks) sampled by zeroing out or removing some operations.
- The expensive design of supernet is amortized by reusing it for different target scenarios (chips).



Each cell has choice blocks and each choice block can select up to 2 operations. Solid edges are used in every architecture, where dash lines are optional (Bender et al 2018)

Selected single-objective NAS methods



Method	Ref.	Year	NAS: Search		Super	ImgNet		C-10	
			Algorithm	Space	Net	Top-1 Acc.	Param. [$\cdot 10^6$]	Top-1 Acc.	Param. [$\cdot 10^6$]
NAS	[3]	2016	RL	macro		–	–	96.35	37.4
CoDeepNEAT	[84]	2017	EA	hierarchical				92.70	–
GeNET	[78]	2017	EA	stage		72.1	156.0	94.61	–
MetaQNN	[112]	2017	RL	macro		–	–	93.08	11.2
ENAS	[104]	2018	RL	cell/macro		–	–	97.11	4.6
NASNet	[82]	2018	RL	cell		82.7	88.9	97.81	27.6
PNAS	[97]	2018	SMBO	block		82.9	86.1	96.59	3.2
AMOEBA	[96]	2019	EA, RL	cell		83.9	469.0	96.66	3.2
CGP-CNN19	[95]	2019	EA	global		–	–	95.10	2.7
DARTS	[108]	2019	gradient	cell		73.3	4.7	97.24	3.3
MixNet	[113]	2019	RL	kernel		78.9	7.3	–	–
RENAS	[114]	2019	RL, EA	cell		75.7	5.4	97.12	3.5
ShuffleNAS	[115]	2019	RL	cell		–	–	96.43	3.1
CNN-GA	[116]	2020	EA	macro		–	–	96.78	2.9
MS-RANAS	[117]	2020	gradient	cell		–	–	95.00	–
AS-NAS	[118]	2021	RL, EA	macro		–	–	97.77	16.6
DNAL	[119]	2021	gradient	channel	Y	75.0	3.6	94.30	1.2
LaNAS	[98]	2021	MCTS, SMBO	cell	Y	80.8	8.2	99.01	44.1
P-DARTS	[111]	2021	gradient	cell		75.9	5.4	97.75	10.5
PC-DARTS	[110]	2021	gradient	cell	Y	75.9	5.1	97.45	3.2
TE_NAS	[102]	2021	training free	cell	Y	75.5	5.4	97.37	3.8
VINNAS	[120]	2021	gradient	cell	Y	–	–	96.06	1.8

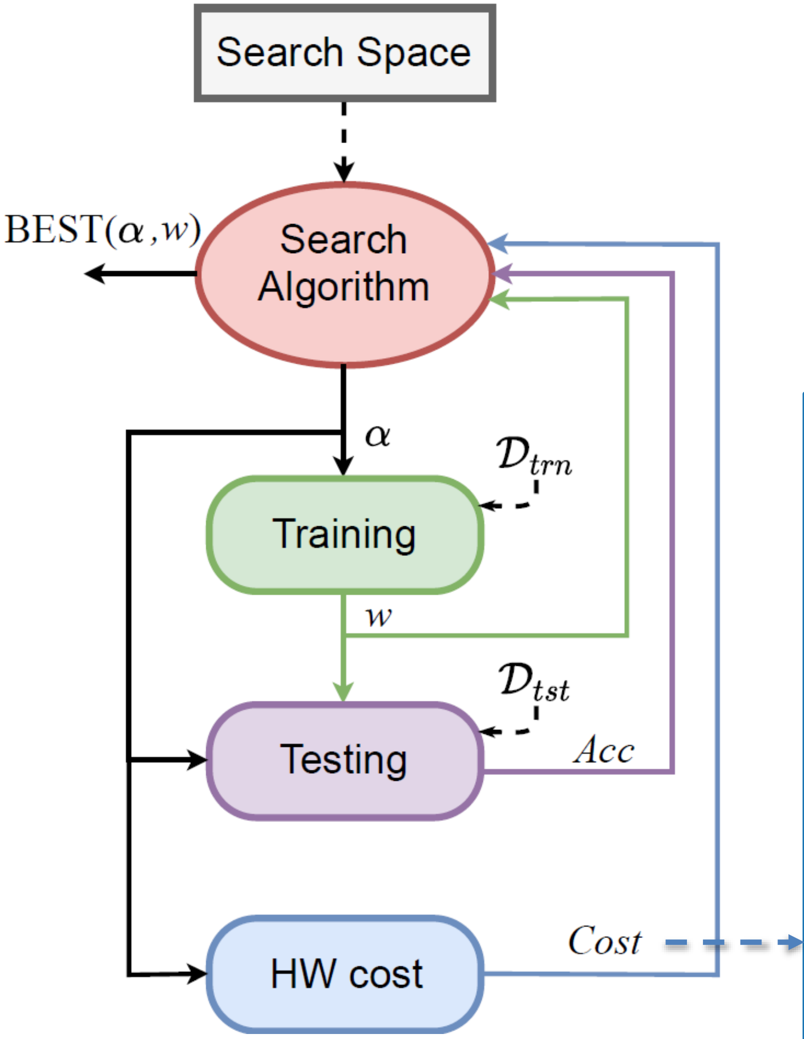
see Sekanina L.: IEEE Access, 2021

SMBO – Sequential Model Based Optimization

Outline

- Efficient processing of CNNs
- Hardware accelerators for CNNs
- Single-objective Neural Architecture Search (NAS)
  - Search spaces and Search algorithms
  - Performance predictors
  - Classification of NAS methods
- Multi-objective Neural Architecture Search
  - Hardware-aware NAS
  - NAS with HW Co-design
  - Classification of NAS methods
- Benchmarking of NAS methods
- Conclusions

Multi-objective NAS for a particular (fixed) hardware



Additional Objectives:

Latency

Area

Energy

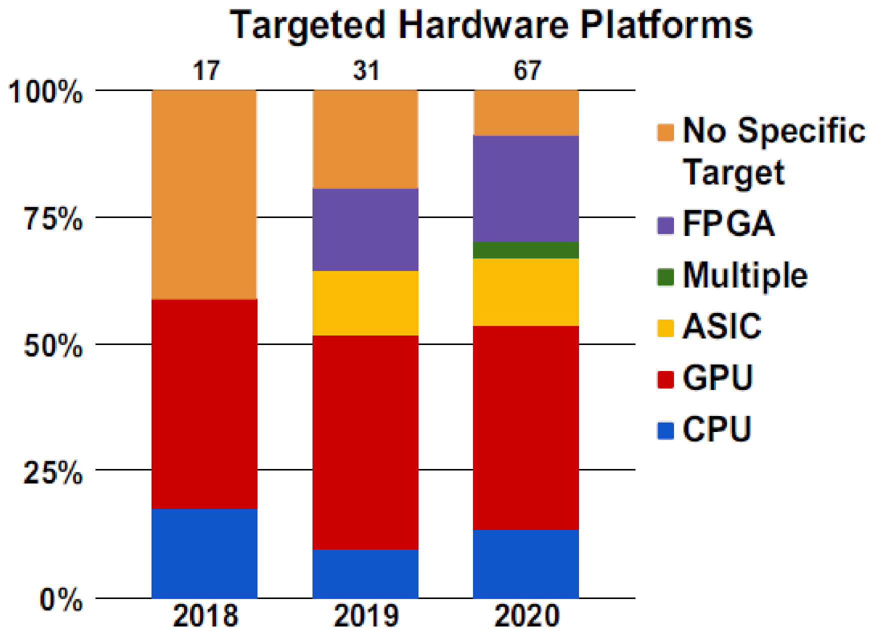
RAM size

Flash size

#MAC

Reliability

etc.

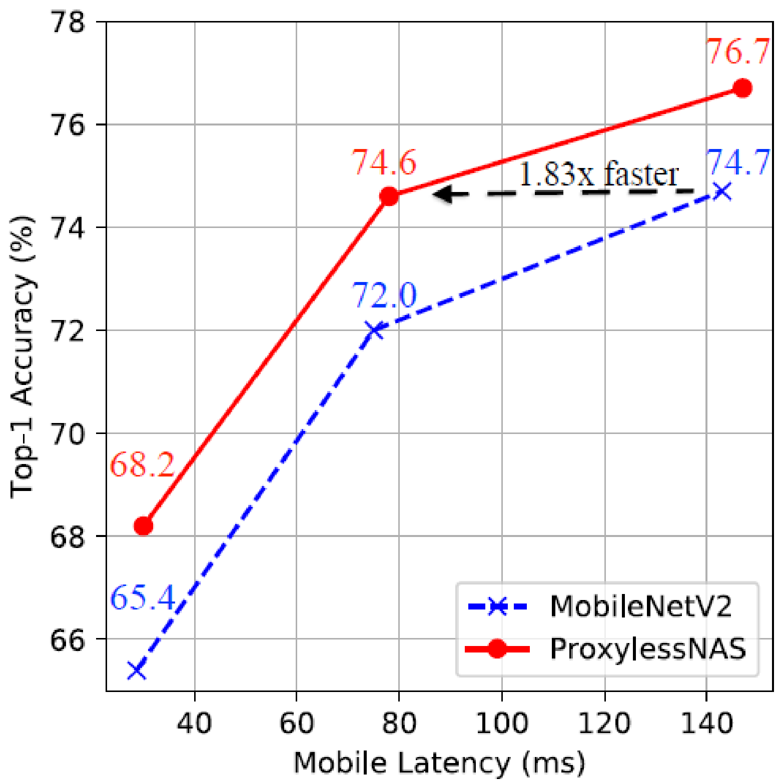


Benmeziane et al. 2021

Hardware-aware NAS is a NAS reflecting a given hardware executing the inference.

Important: Hardware itself **is not** optimized! There is no additional search space of hardware configurations.

Multi-objective NAS for a particular (fixed) hardware



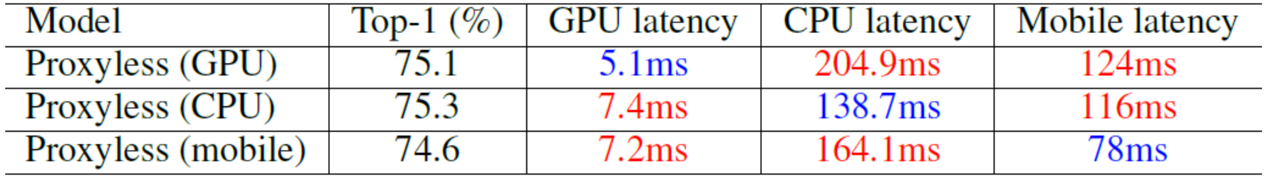
ProxylessNAS optimizing **Accuracy** and **Latency**

Possibilities when solving a multi-objective optimization problem with objective functions $f_1 \dots f_m$:

1. Transform it into a single-objective problem (using suitable constraints, prioritization, or aggregation techniques) and solve it with a common single-objective method

   $$f^A(a) = \sum_{i=1}^{m} v_i \cdot f_i(a)$$

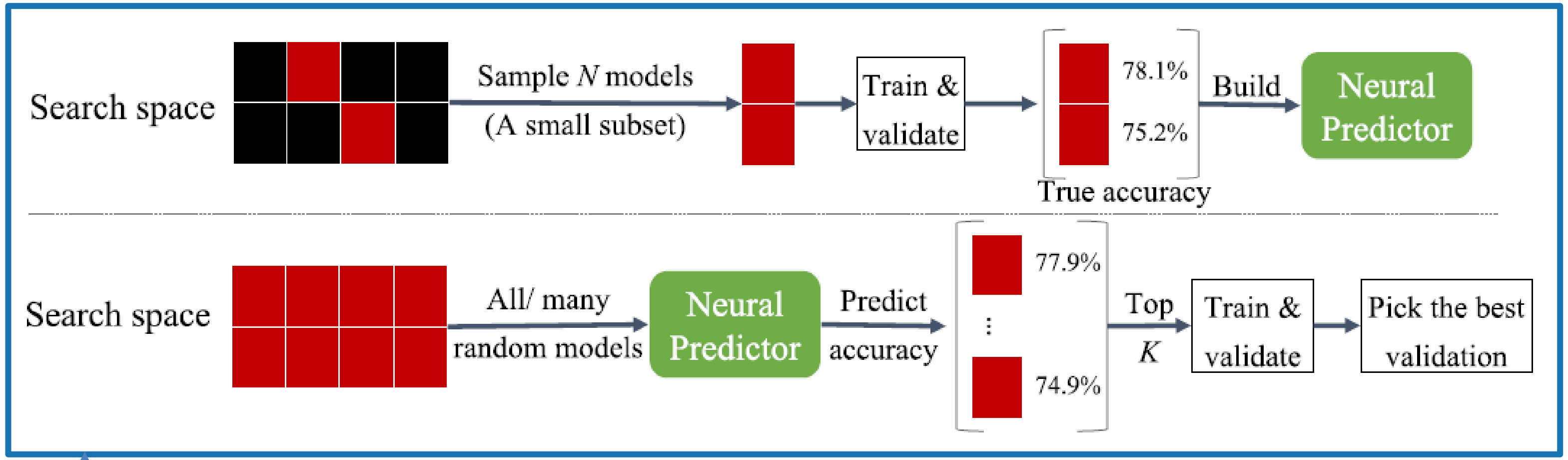
2. Employ a truly multi-objective approach, e.g. NSGA-II, that utilizes the concept of Pareto dominance during the search.



Model	Top-1 (%)	GPU latency	CPU latency	Mobile latency
Proxyless (GPU)	75.1	5.1ms	204.9ms	124ms
Proxyless (CPU)	75.3	7.4ms	138.7ms	116ms
Proxyless (mobile)	74.6	7.2ms	164.1ms	78ms

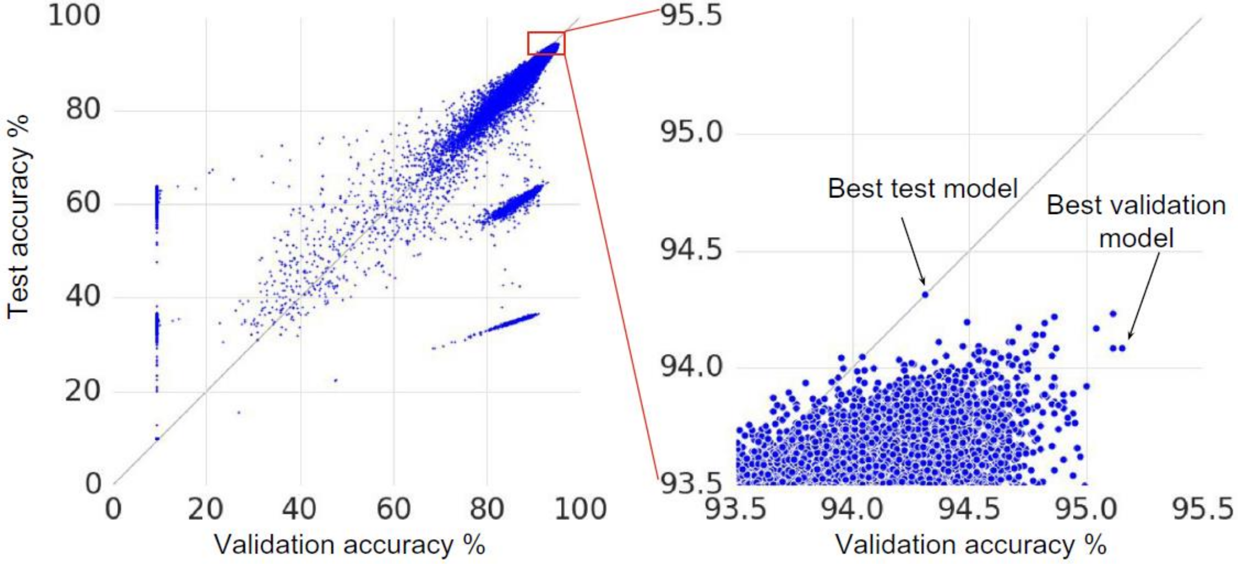
Shortening the evaluation time: Accuracy

- Simplify the common approach
  - Employ a proxy data set
  - Reduce the number of training epochs
  - Extrapolate the learning curve
  - etc.
- Build a surrogate model – Accuracy predictor
  - NN
  - regression trees
  - Gaussian process (GP)
  - etc.
- NAS needs only the rank of the performance values



Search space → Sample N models (A small subset) → Train & validate → 78.1% … 75.2% True accuracy → Build → Neural Predictor

Search space → All/ many random models → Neural Predictor → Predict accuracy → 77.9% ⋮ 74.9% → Top K → Train & validate → Pick the best validation



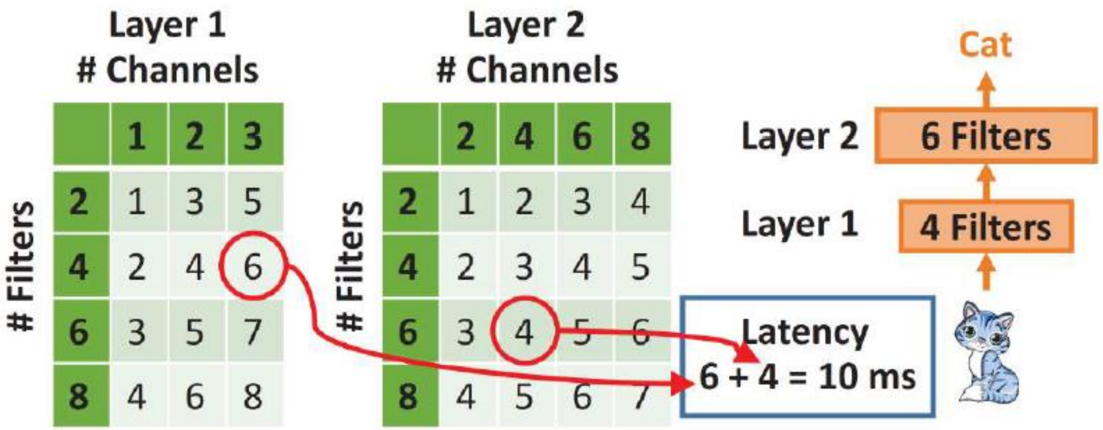
Results on NASBench-101 (CIFAR-10) by Wen W. et al. ECCV 2020

Shortening the evaluation time: Hardware metrics

Hardware metrics: Latency, Energy, Area, Memory etc.

Methods according to Benmeziane et al. 2021:

- **Baseline:** Real-time measurements on target hardware.
- **Analytical Estimation** - consists of analytical computing a rough estimate, e.g., using the processing time, the stall time, and the starting time.
- **Prediction Model** a ML model is built to predict the cost using architecture and dataset features.
- **Lookup Table** Models

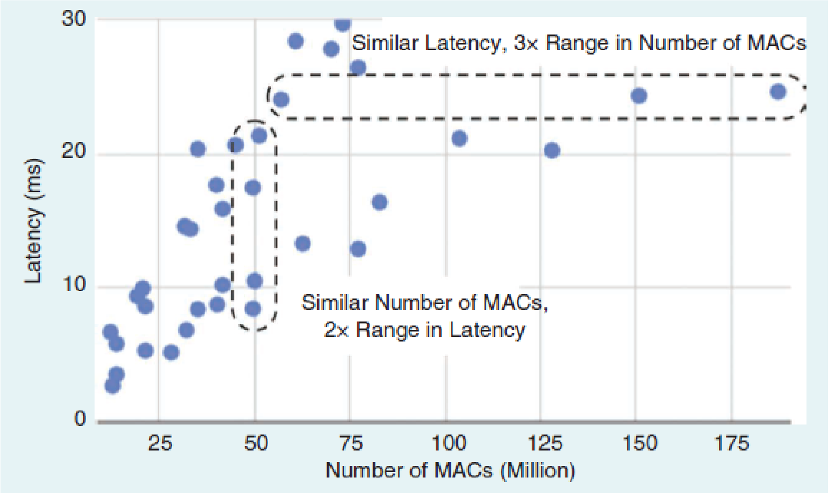


Layer 1 # Channels

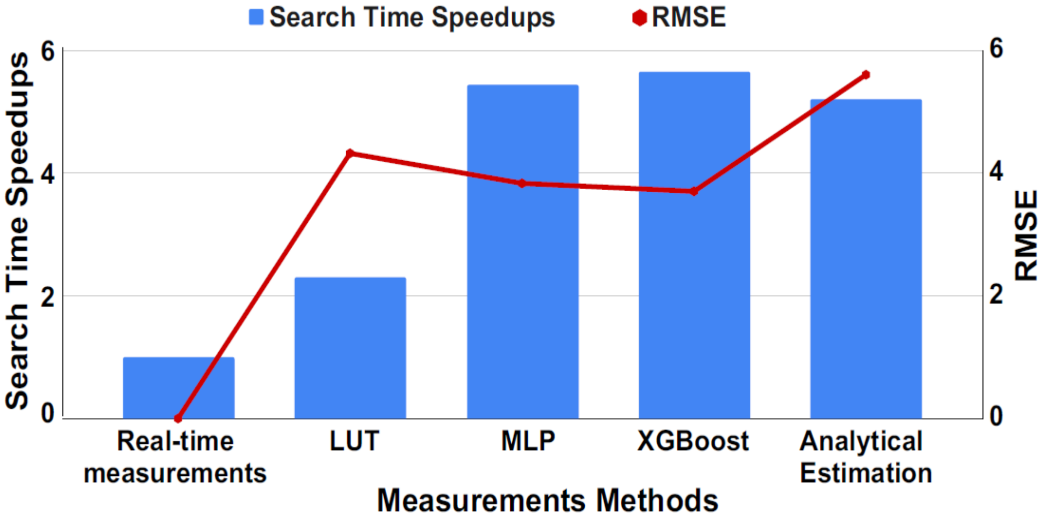
# Filters	1	2	3
2	1	3	5
4	2	4	6
6	3	5	7
8	4	6	8

Layer 2 # Channels

# Filters	2	4	6	8
2	1	2	3	4
4	2	3	4	5
6	3	4	5	6
8	4	5	6	7



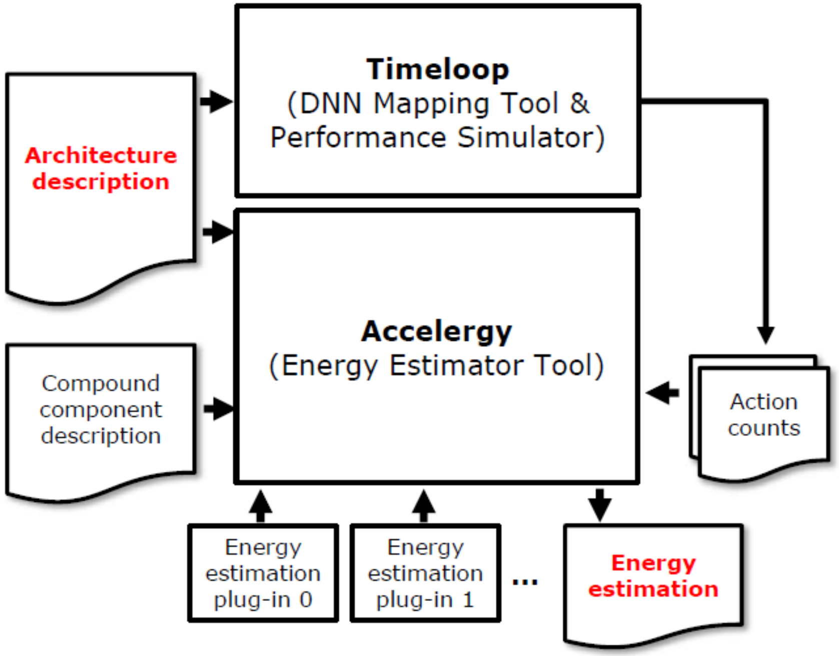
#MAC is not a good proxy for latency! Shown for various NN models on a Google Pixel phone.

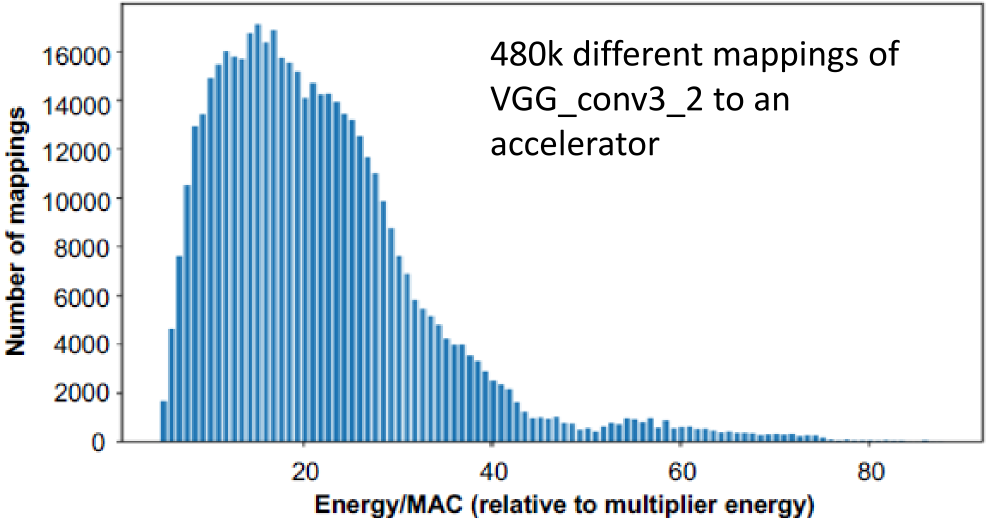


Benmeziane et al. 2021

DNN-to-Accelerator Mapping and Energy estimation SW tools

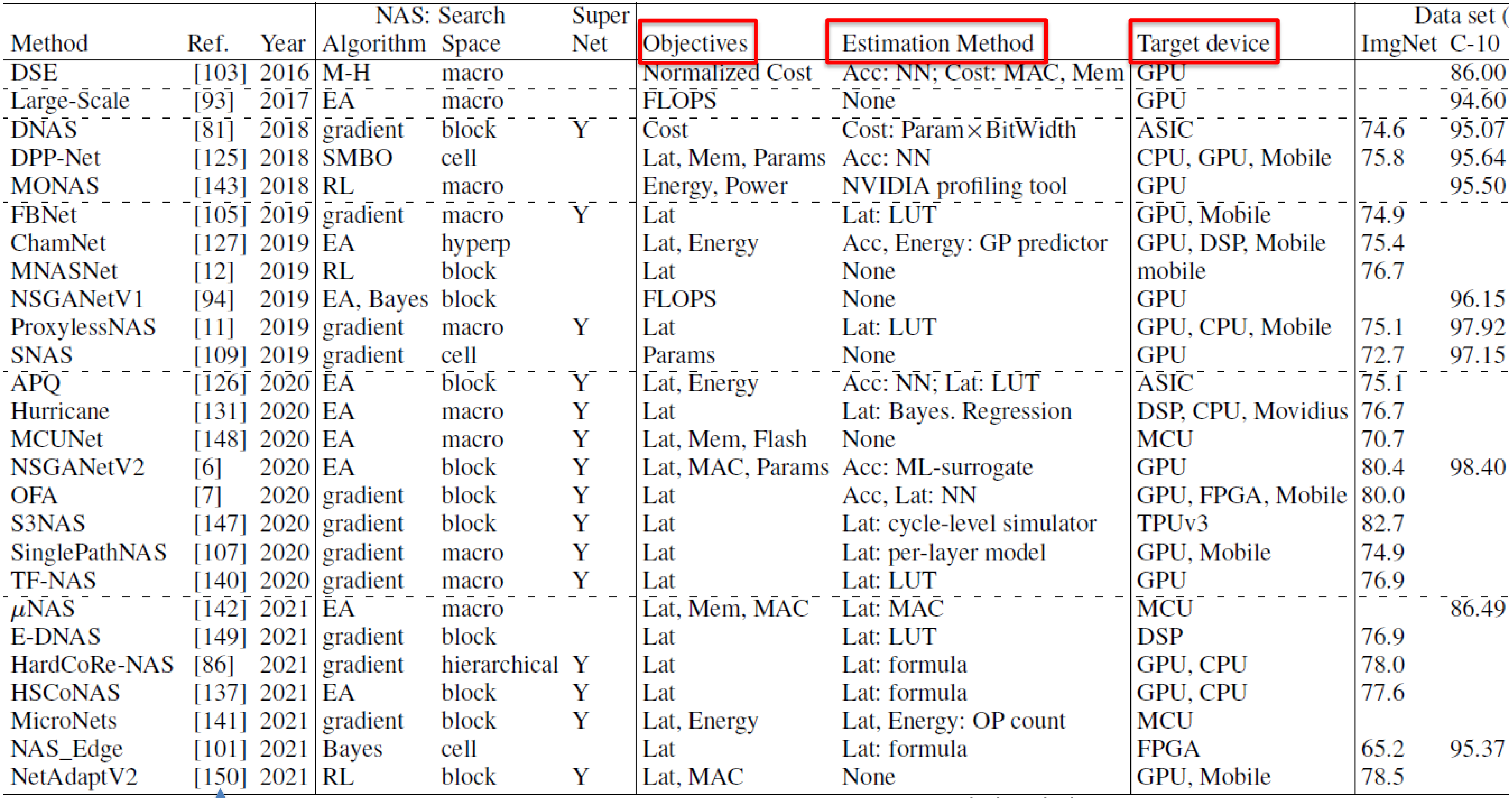
- **Timeloop** [**Parashar**, *ISPASS* 2019]
  - A tool searching for the most suitable mapping of DNN to HW accelerator (several search methods implemented)
    - Layer-wise data tiling reflecting the memory hierarchy of the accelerator
  - HW accelerator is described at the architecture level, incl. data flow organization
  - Performance Simulator -> Action counts
- **Accelergy** [**Wu**, *ICCAD* 2019]
  - Early stage energy estimation tool at the architecture level
    - Estimate energy consumption based on architecture level components (e.g., # of PEs, memory size, on-chip network)
    - Plug-ins for different technologies


Architecture description
Timeloop
(DNN Mapping Tool & Performance Simulator)
Accelergy
(Energy Estimator Tool)
Compound component description
Action counts
Energy estimation plug-in 0
Energy estimation plug-in 1
...
Energy estimation



480k different mappings of VGG_conv3_2 to an accelerator

Selected hardware-aware NAS methods



			NAS:	Search	Super				Data set (	
Method	Ref.	Year	Algorithm	Space	Net	Objectives	Estimation Method	Target device	ImgNet	C-10
DSE	[103]	2016	M-H	macro		Normalized Cost	Acc: NN; Cost: MAC, Mem	GPU		86.00
Large-Scale	[93]	2017	EA	macro		FLOPS	None	GPU		94.60
DNAS	[81]	2018	gradient	block	Y	Cost	Cost: Param×BitWidth	ASIC	74.6	95.07
DPP-Net	[125]	2018	SMBO	cell		Lat, Mem, Params	Acc: NN	CPU, GPU, Mobile	75.8	95.64
MONAS	[143]	2018	RL	macro		Energy, Power	NVIDIA profiling tool	GPU		95.50
FBNet	[105]	2019	gradient	macro	Y	Lat	Lat: LUT	GPU, Mobile	74.9	
ChamNet	[127]	2019	EA	hyperp		Lat, Energy	Acc, Energy: GP predictor	GPU, DSP, Mobile	75.4	
MNASNet	[12]	2019	RL	block		Lat	None	mobile	76.7	
NSGANetV1	[94]	2019	EA, Bayes	block		FLOPS	None	GPU		96.15
ProxylessNAS	[11]	2019	gradient	macro	Y	Lat	Lat: LUT	GPU, CPU, Mobile	75.1	97.92
SNAS	[109]	2019	gradient	cell		Params	None	GPU	72.7	97.15
APQ	[126]	2020	EA	block	Y	Lat, Energy	Acc: NN; Lat: LUT	ASIC	75.1	
Hurricane	[131]	2020	EA	macro	Y	Lat	Lat: Bayes. Regression	DSP, CPU, Movidius	76.7	
MCUNet	[148]	2020	EA	macro	Y	Lat, Mem, Flash	None	MCU	70.7	
NSGANetV2	[6]	2020	EA	block	Y	Lat, MAC, Params	Acc: ML-surrogate	GPU	80.4	98.40
OFA	[7]	2020	gradient	block	Y	Lat	Acc, Lat: NN	GPU, FPGA, Mobile	80.0	
S3NAS	[147]	2020	gradient	block	Y	Lat	Lat: cycle-level simulator	TPUv3	82.7	
SinglePathNAS	[107]	2020	gradient	macro	Y	Lat	Lat: per-layer model	GPU, Mobile	74.9	
TF-NAS	[140]	2020	gradient	macro	Y	Lat	Lat: LUT	GPU	76.9	
μNAS	[142]	2021	EA	macro		Lat, Mem, MAC	Lat: MAC	MCU		86.49
E-DNAS	[149]	2021	gradient	block		Lat	Lat: LUT	DSP	76.9	
HardCoRe-NAS	[86]	2021	gradient	hierarchical	Y	Lat	Lat: formula	GPU, CPU	78.0	
HSCoNAS	[137]	2021	EA	block	Y	Lat	Lat: formula	GPU, CPU	77.6	
MicroNets	[141]	2021	gradient	block	Y	Lat, Energy	Lat, Energy: OP count	MCU		
NAS_Edge	[101]	2021	Bayes	cell		Lat	Lat: formula	FPGA	65.2	95.37
NetAdaptV2	[150]	2021	RL	block	Y	Lat, MAC	None	GPU, Mobile	78.5	

Lat – Latency, Mem – RAM size; Flash – Flash size

see Sekanina L.: IEEE Access, 2021

NAS with HW Co-design: Three search spaces!

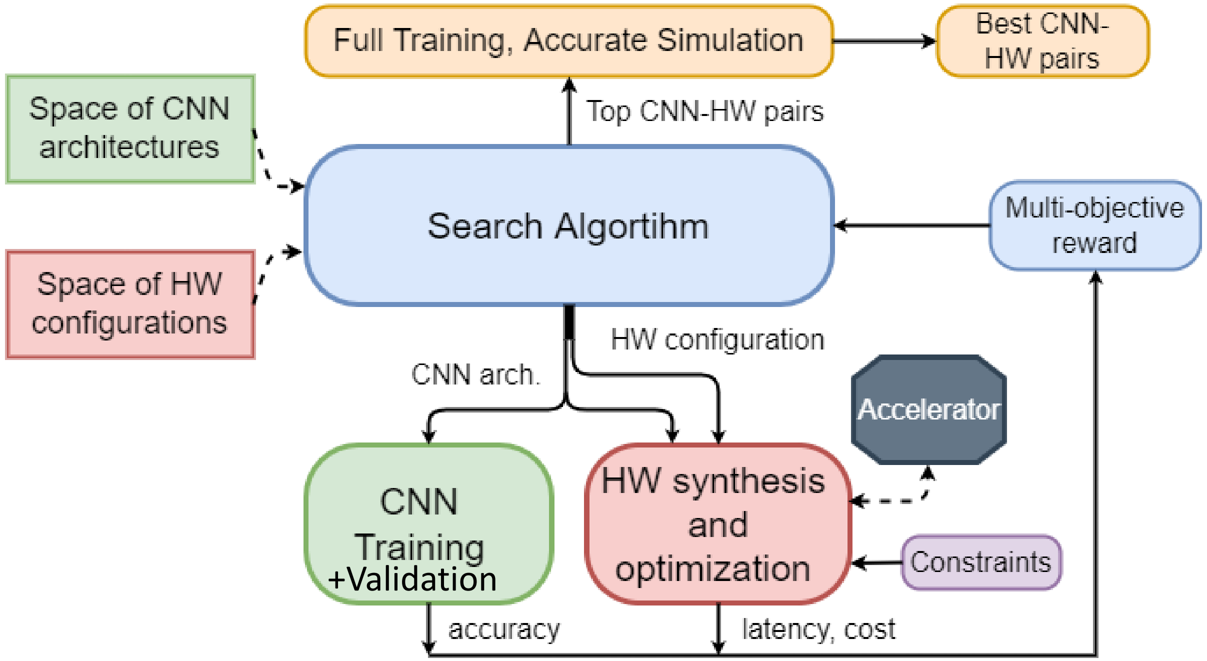
Search Spaces	Search Algorithms
NN models	RL, EA, SMBO, gradient …
NN weights	Gradient
HW configurations	EA, ILP, gradient, … join with

- Bit widths
- Quantization
- PE array size
- Buffer size
- MAC circuit configuration (approx. multipliers)
- Dataflow organization
- Tiling strategy
- Loop order
- Memory subsystem parameters
- Preferences for the high-level synthesis SW
- …

We can observe that the HW-aware NAS has a much narrower search space than the proposed co-exploration approach. Basically, HW-aware NAS will prune the architectures that violates hardware specifications on a fixed hardware design. However, by opening the hardware design space, it is possible to find a tailor-made hardware design for the pruned architectures to make them meet the hardware specifications. Therefore, compared with the HW-aware NAS, the co-exploration approach enlarges the search space. As a result, it can make better tradeoffs between accuracy and hardware efficiency. [Jiang et al. IEEE TCAD 2020]

NAS with HW Co-design: selected approaches

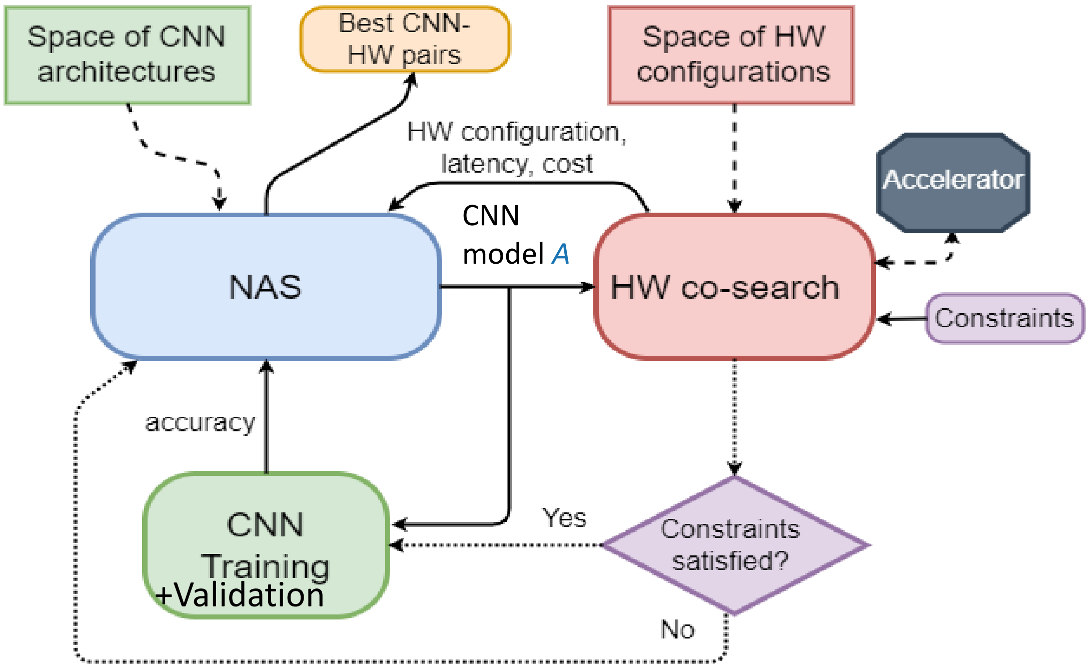
One search algorithm



```
Sample CNN & HW configuration
Evaluate Accuracy & Obtain HW parameters
REPEAT until not satisfactory
Report the best CNN-HW pair
```

Too time-consuming!

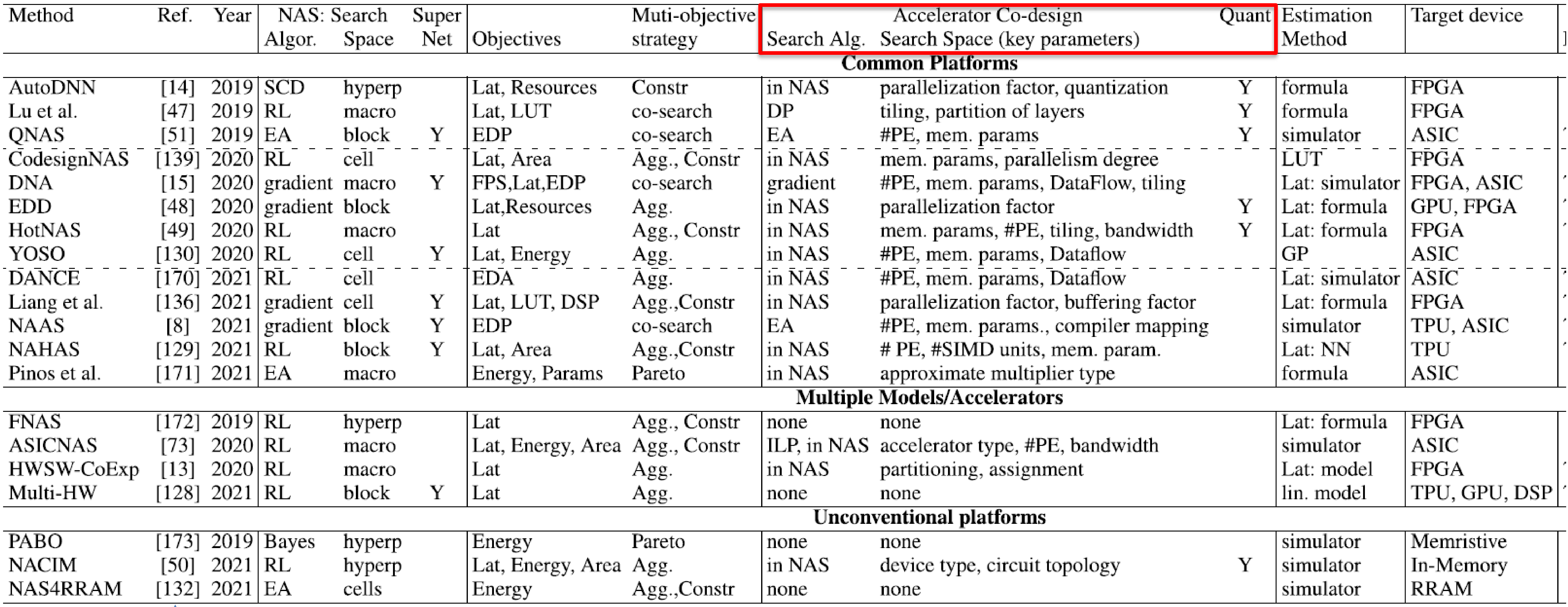
Two search algorithms



```
Sample a CNN model A
Optimize HW for A
IF the HW implementation of A is satisfactory
   THEN Train and test A to get Accuracy
   ELSE Discard A
REPEAT until not satisfactory
Report the best CNN-HW pair
```

Suitable if TIME(HW eval) << TIME(Train&Test)

Selected NAS methods with HW Co-design



Method	Ref.	Year	NAS: Search		Super		Muti-objective	Accelerator Co-design		Quant	Estimation	Target device
			Algor.	Space	Net	Objectives	strategy	Search Alg.	Search Space (key parameters)		Method	
Common Platforms												
AutoDNN	[14]	2019	SCD	hyperp		Lat, Resources	Constr	in NAS	parallelization factor, quantization	Y	formula	FPGA
Lu et al.	[47]	2019	RL	macro		Lat, LUT	co-search	DP	tiling, partition of layers	Y	formula	FPGA
QNAS	[51]	2019	EA	block	Y	EDP	co-search	EA	#PE, mem. params	Y	simulator	ASIC
CodesignNAS	[139]	2020	RL	cell		Lat, Area	Agg., Constr	in NAS	mem. params, parallelism degree		LUT	FPGA
DNA	[15]	2020	gradient	macro	Y	FPS,Lat,EDP	co-search	gradient	#PE, mem. params, DataFlow, tiling		Lat: simulator	FPGA, ASIC
EDD	[48]	2020	gradient	block		Lat,Resources	Agg.	in NAS	parallelization factor	Y	Lat: formula	GPU, FPGA
HotNAS	[49]	2020	RL	macro		Lat	Agg., Constr	in NAS	mem. params, #PE, tiling, bandwidth	Y	Lat: formula	FPGA
YOSO	[130]	2020	RL	cell	Y	Lat, Energy	Agg.	in NAS	#PE, mem. params, Dataflow		GP	ASIC
DANCE	[170]	2021	RL	cell		EDA	Agg.	in NAS	#PE, mem. params, Dataflow		Lat: simulator	ASIC
Liang et al.	[136]	2021	gradient	cell	Y	Lat, LUT, DSP	Agg.,Constr	in NAS	parallelization factor, buffering factor		Lat: formula	FPGA
NAAS	[8]	2021	gradient	block	Y	EDP	co-search	EA	#PE, mem. params., compiler mapping		simulator	TPU, ASIC
NAHAS	[129]	2021	RL	block	Y	Lat, Area	Agg.,Constr	in NAS	# PE, #SIMD units, mem. param.		Lat: NN	TPU
Pinos et al.	[171]	2021	EA	macro		Energy, Params	Pareto	in NAS	approximate multiplier type		formula	ASIC
Multiple Models/Accelerators												
FNAS	[172]	2019	RL	hyperp		Lat	Agg., Constr	none	none		Lat: formula	FPGA
ASICNAS	[73]	2020	RL	macro		Lat, Energy, Area	Agg., Constr	ILP, in NAS	accelerator type, #PE, bandwidth		simulator	ASIC
HWSW-CoExp	[13]	2020	RL	macro		Lat	Agg.	in NAS	partitioning, assignment		Lat: model	FPGA
Multi-HW	[128]	2021	RL	block	Y	Lat	Agg.	none	none		lin. model	TPU, GPU, DSP
Unconventional platforms												
PABO	[173]	2019	Bayes	hyperp		Energy	Pareto	none	none		simulator	Memristive
NACIM	[50]	2021	RL	hyperp		Lat, Energy, Area	Agg.	in NAS	device type, circuit topology	Y	simulator	In-Memory
NAS4RRAM	[132]	2021	EA	cells		Energy	Agg.,Constr	none	none		simulator	RRAM

see Sekanina L.: IEEE Access, 2021

in NAS – one search algorithm. The same algorithm is used to search for the NN model and HW configurations

NAS with HW Co-design: The NAAS method

- Normalized EDP and top-1 accuracy of ResNet (on ImageNet)
- Eyeriss accelerator

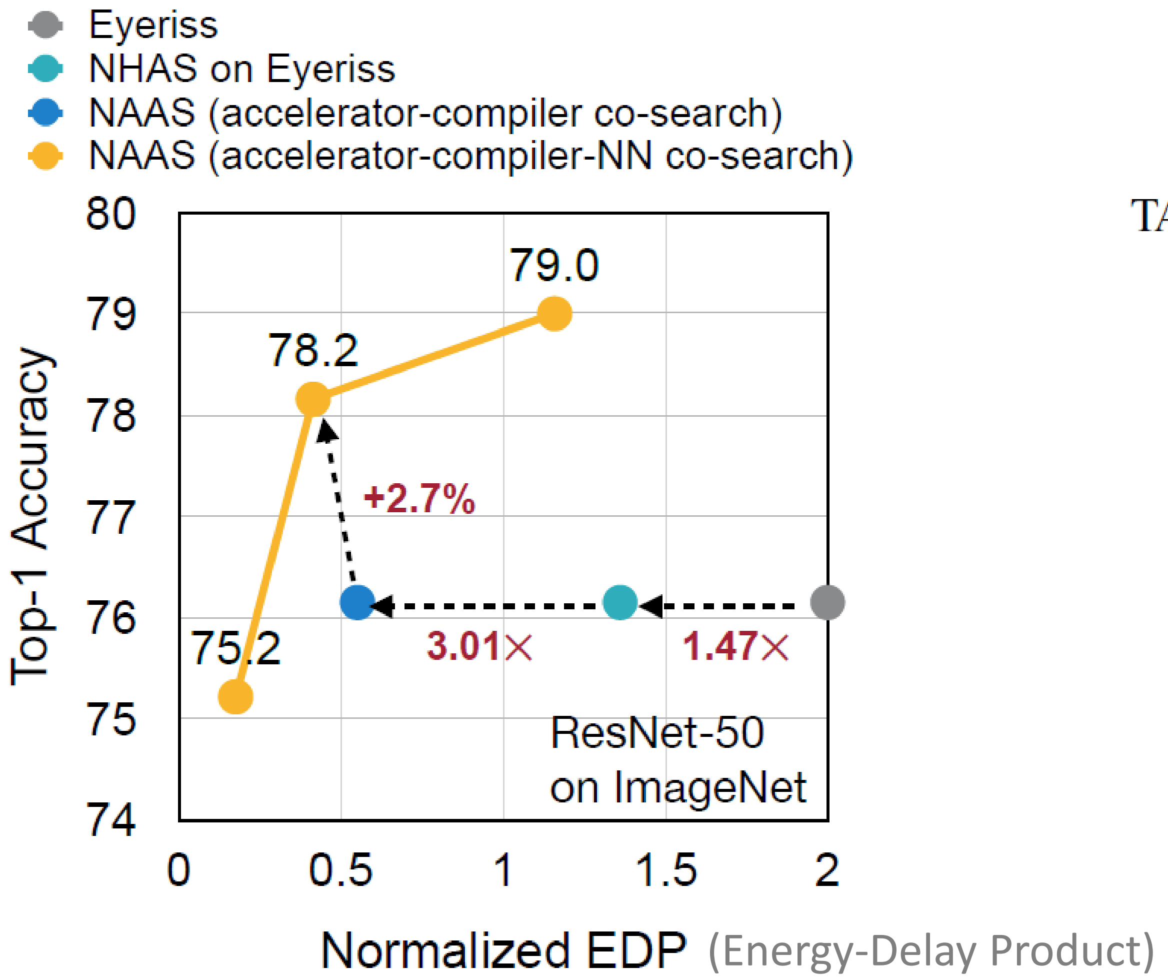
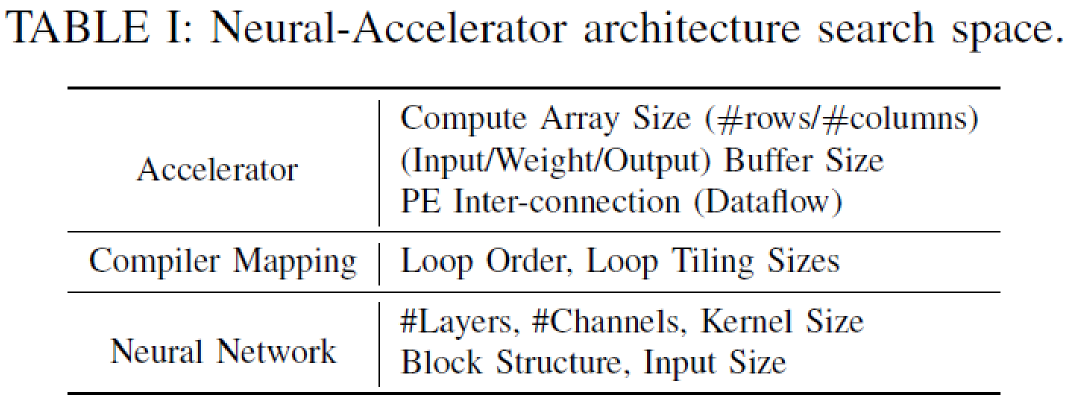


TABLE I: Neural-Accelerator architecture search space.



Accelerator	Compute Array Size (#rows/#columns) (Input/Weight/Output) Buffer Size PE Inter-connection (Dataflow)
Compiler Mapping	Loop Order, Loop Tiling Sizes
Neural Network	#Layers, #Channels, Kernel Size Block Structure, Input Size

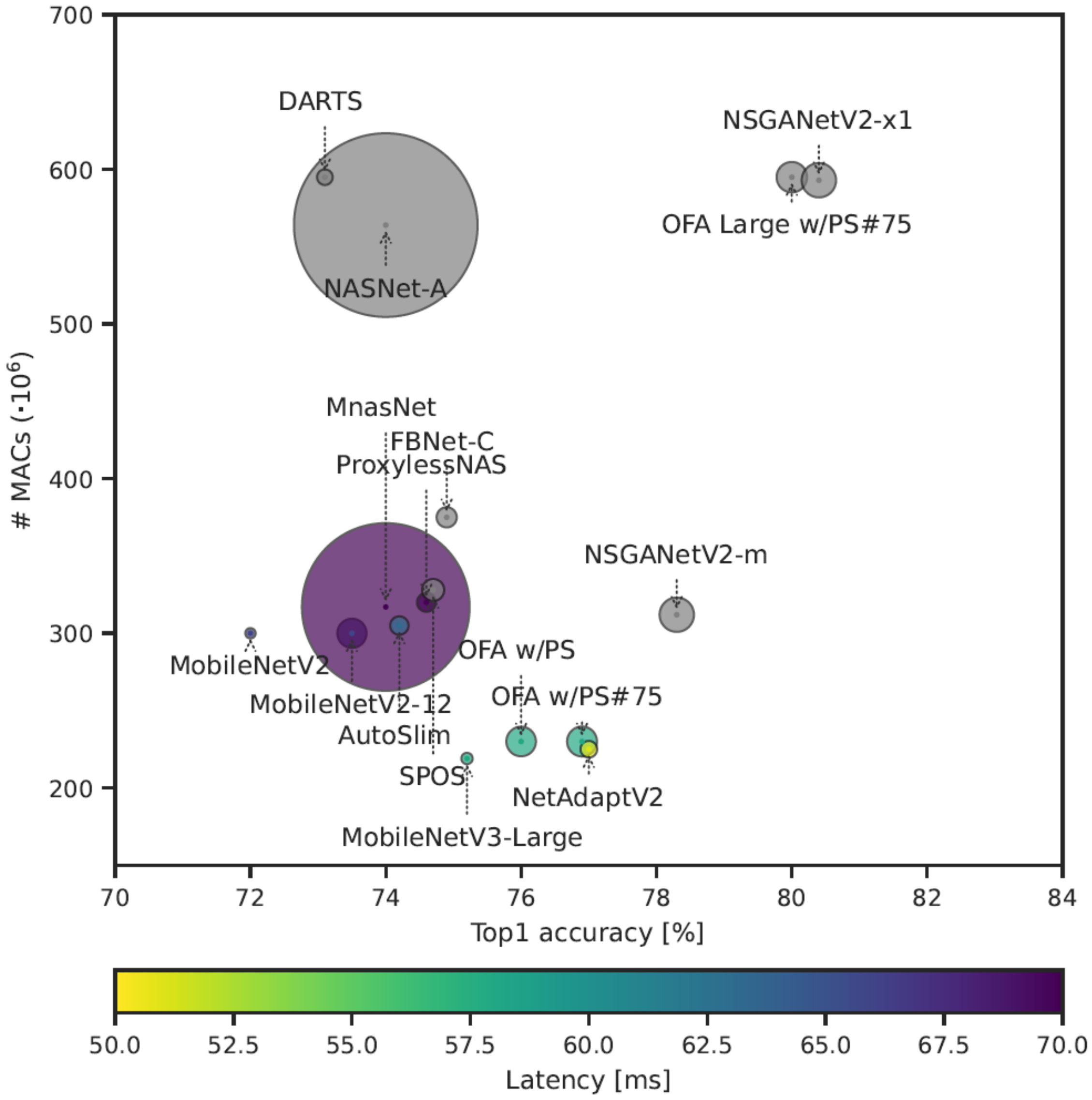
Y. Lin, M. Yang, and S. Han: NAAS: Neural accelerator architecture search, DAC 2021

Comparison of selected NAS methods

- NAS methods are evaluated with respect to the quality of produced CNNs and the resources needed to generate them.
- Fair benchmarking of an extensive collection of NAS methods (particularly the hardware-aware NAS methods) remains an open research problem. The difficulty is that too many aspects have to be considered during the comparison, and their deep cross-analysis is expensive to perform.

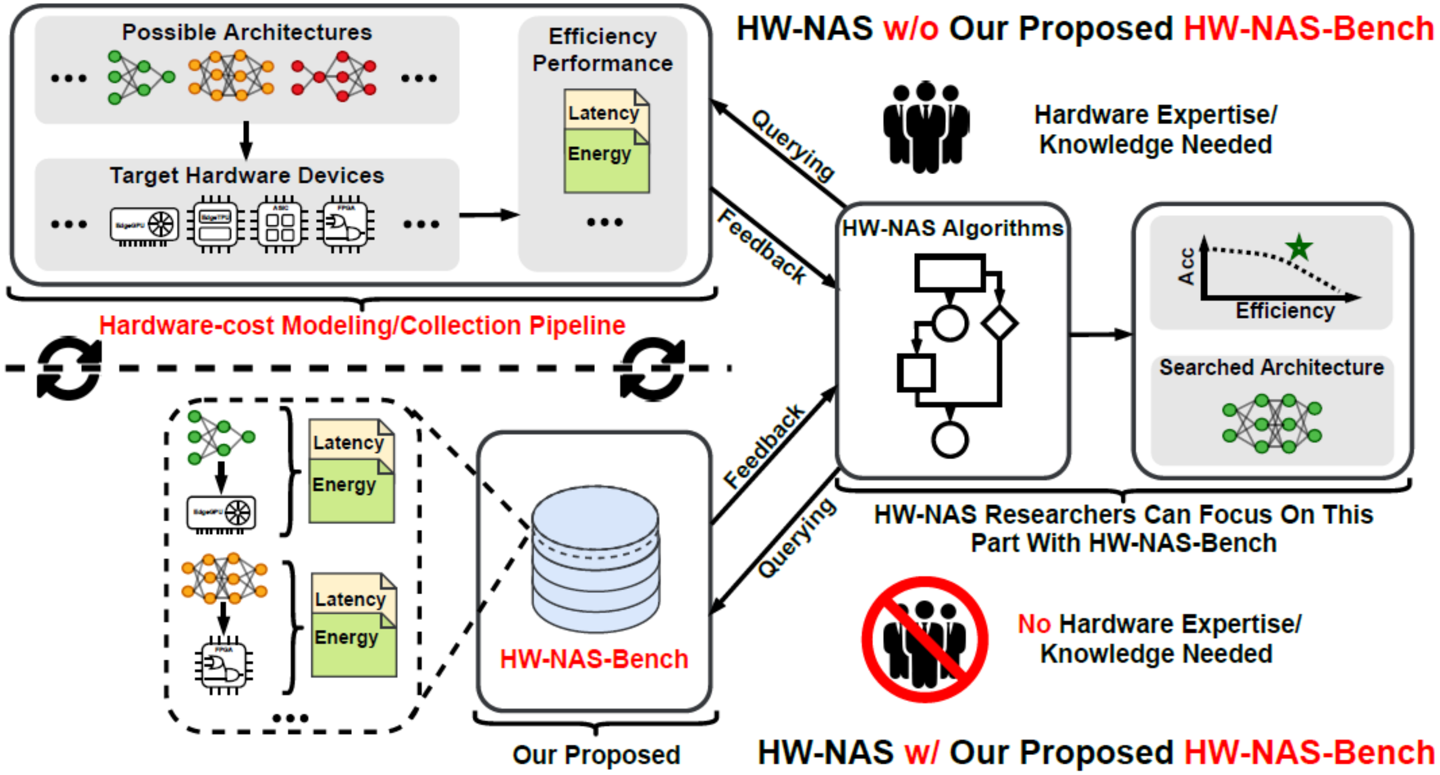
Figure:

- The top-1 accuracy (ImageNet), the number of MACs, and latency on Pixel 1 phone for CNNs obtained using selected NAS methods.
- An unknown latency is depicted using a grey color.
- The circle's area is proportional to the total design time (on a scale from 150 to 40 000 GPU hours).



HW-NAS-Bench: Hardware-Aware Neural Architecture Search Benchmark

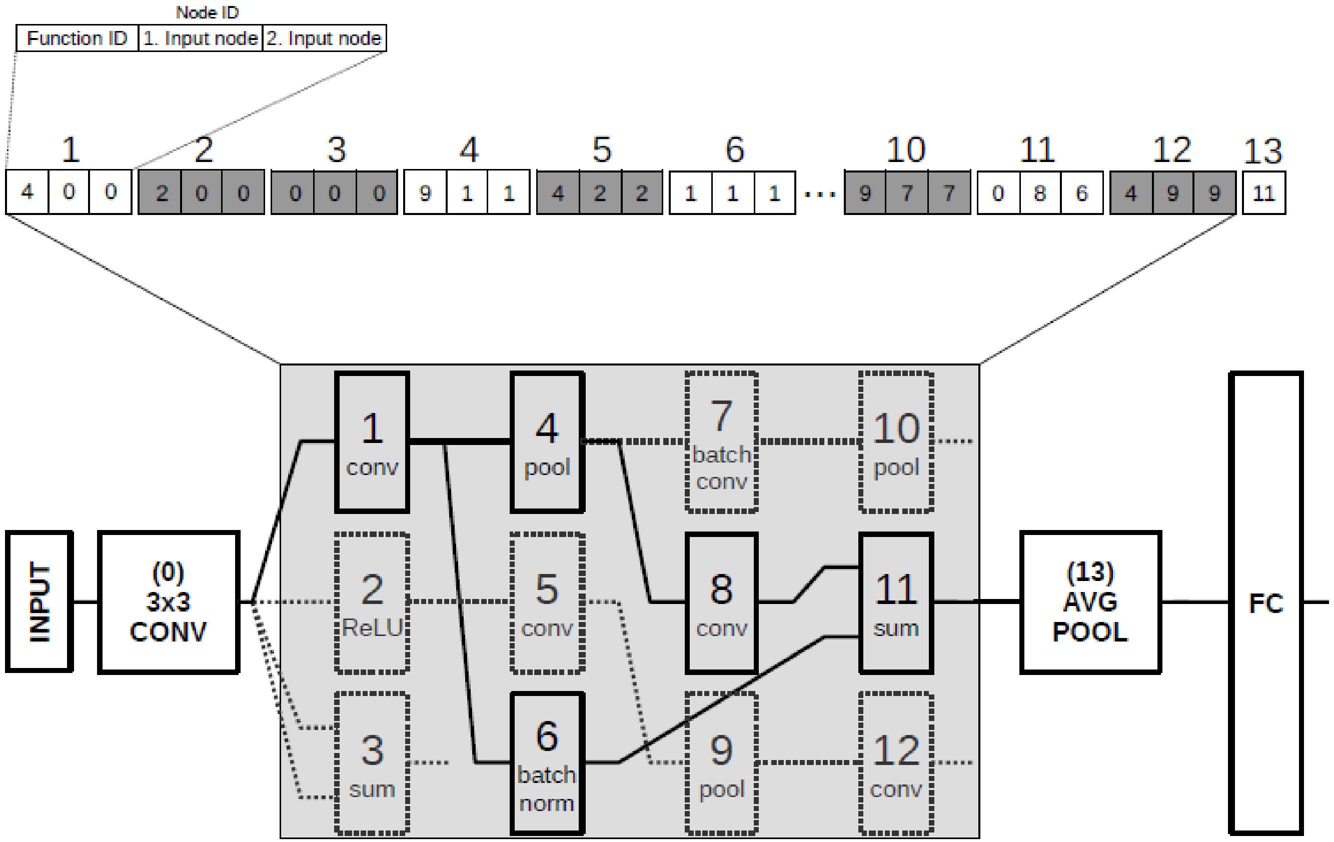
- A unified benchmark for HW-NAS to make HW-NAS research more reproducible and accessible.
- Search spaces
  - NAS-Bench-201: 46875 architectures (CIFAR-10, CIFAR-100, ImageNet16-120)
  - FBNet: 10^{21} architectures (CIFAR-100, ImageNet)
- HW: Edge GPU, Edge TPU, Raspberry Pi 4, Pixel 3, Eyeriss , FPGA
- Available information for each NN model on a given HW: Accuracy, Latency, Energy


Possible Architectures
...
...
Target Hardware Devices
...
...
Efficiency Performance
Latency
Energy
...
Hardware-cost Modeling/Collection Pipeline
HW-NAS w/o Our Proposed HW-NAS-Bench
Hardware Expertise/ Knowledge Needed
Querying
Feedback
HW-NAS Algorithms
Acc
Efficiency
Searched Architecture
Latency
Energy
Latency
Energy
...
HW-NAS-Bench
Our Proposed
Feedback
Querying
HW-NAS Researchers Can Focus On This Part With HW-NAS-Bench
No Hardware Expertise/ Knowledge Needed
HW-NAS w/ Our Proposed HW-NAS-Bench

Li Ch. et al. HW-NAS-Bench: Hardware-Aware Neural Architecture Search Benchmark, ICLR 2021

Example: NAS with approximate multipliers (1) [Pinos, Mrazek, Sekanina: GENP 2022]

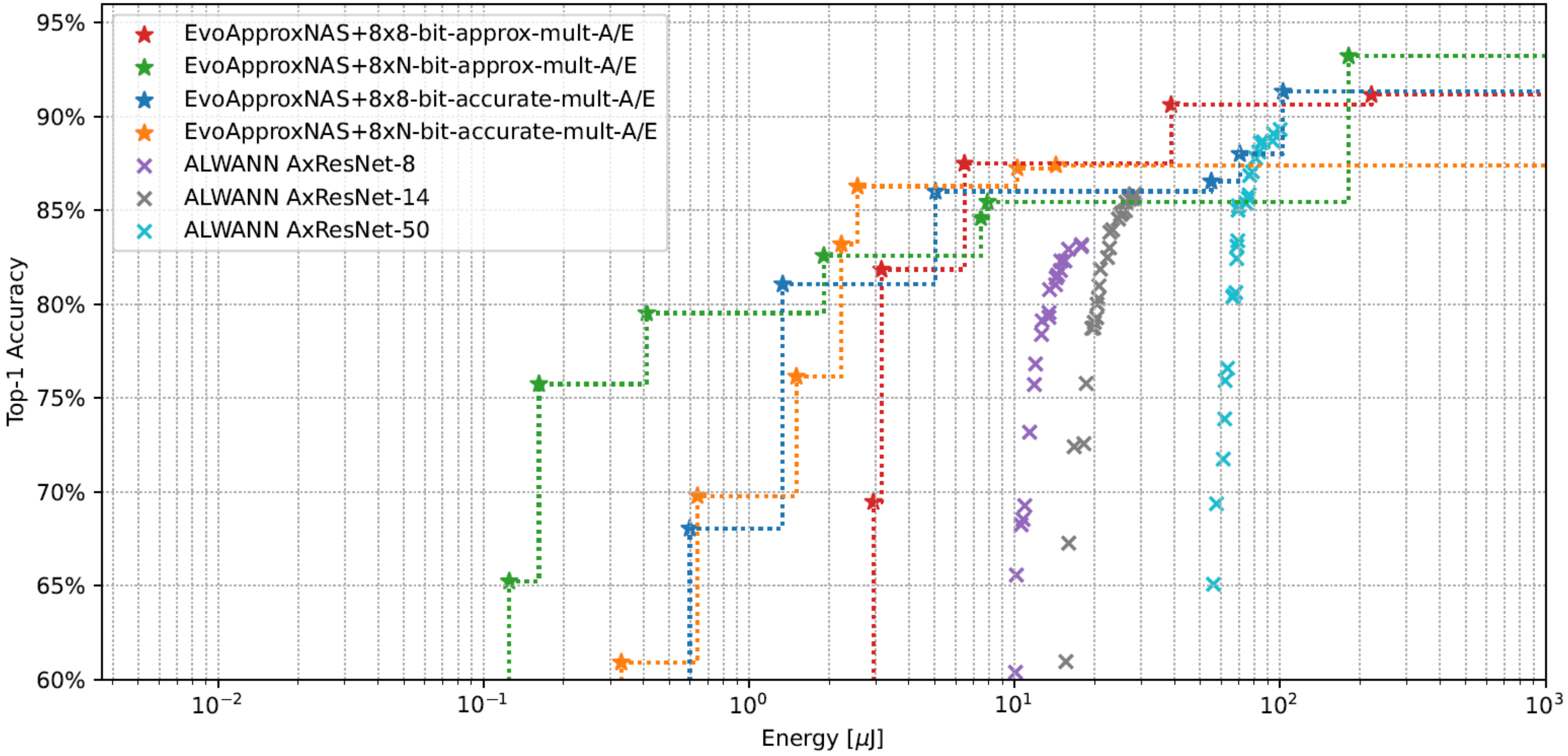
EvoApproxNAS: Cartesian genetic programming + NSGA-II used to design CNNs for image classification and select suitable approximate multipliers (from EvoApproxLib for convolutional layers). Approximate multipliers are employed to reduce power consumption of the on-chip inference

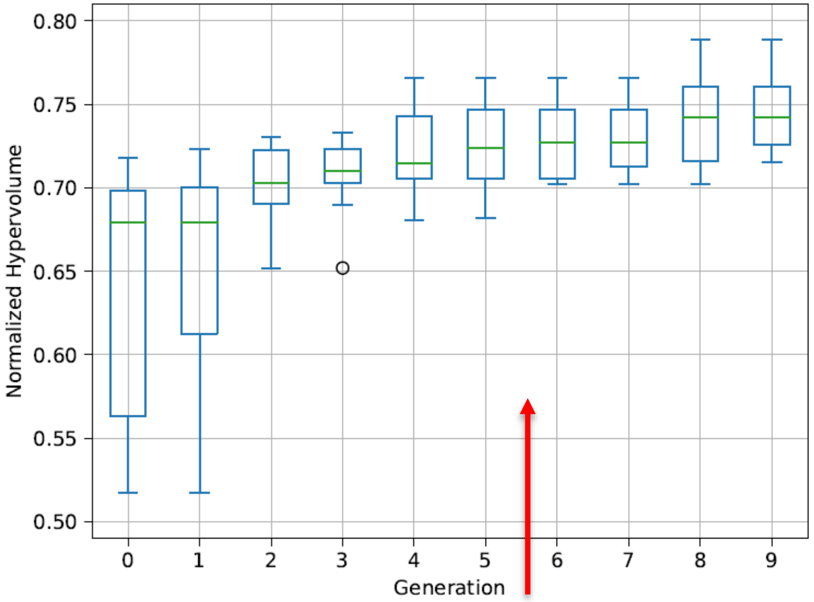


FUNCTION TABLE				
ID	Name	Arity	Parameters	Values
0	summation	2	-	-
1	batch_norm	1	-	-
2	activation	1	activation	$\{relu, elu, leakyrelu, prelu\}$
3	depthwise_conv2d	1	kernel_size strides activation use_bias approx_mul_file	$\{2\times 2, 3\times 3\}$ $\{1\times 1, 2\times 2\}$ $\{relu, elu\}$ $\{True, False\}$ $\{list\_of\_ax\_mults\}$
4	conv2d	1	kernel_size channels strides activation use_bias approx_mul_file	$\{2\times 2, 3\times 3\}$ $\{32, 40, 48, 64, 80, 96, 112, 128\}$ $\{1\times 1, 2\times 2\}$ $\{relu, elu\}$ $\{True, False\}$ $\{list\_of\_ax\_mults\}$
5	basic_residual	1	kernel_size channels strides preact approx_mul_file	$\{3\times 3, 4\times 4\}$ $\{32, 40, 48, 64, 80, 96, 112, 128\}$ $\{1\times 1, 2\times 2\}$ $\{True, False\}$ $\{list\_of\_ax\_mults\}$
6	residual_bottleneck	1	kernel_size factor strides approx_mul_file	$\{3\times 3, 4\times 4\}$ $\{1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0\}$ $\{1\times 1, 2\times 2\}$ $\{list\_of\_ax\_mults\}$
7	residual_inverted	1	kernel_size factor strides approx_mul_file	$\{3\times 3, 4\times 4\}$ $\{1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0\}$ $\{1\times 1, 2\times 2\}$ $\{list\_of\_ax\_mults\}$
8	dropout	1	min max	$\{0.05, 0.10, 0.15\}$ $\{0.20, 0.25, 0.30\}$
9	max_pooling	1	-	-

Typical setup of CGP: 10 x 30 nodes, population_size = 8, generations = 10, mutation-based search.

NAS with approximate multipliers (2)

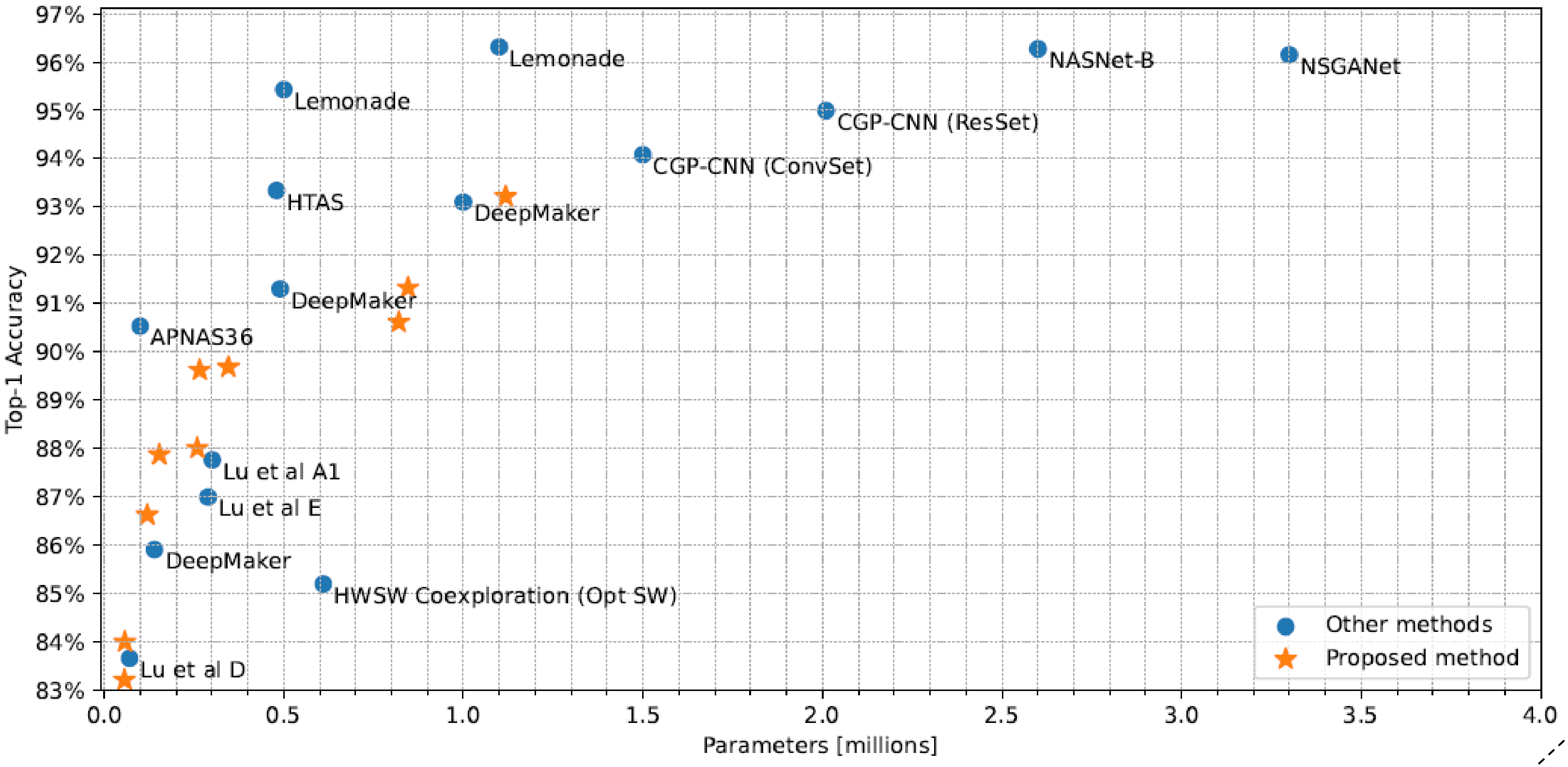




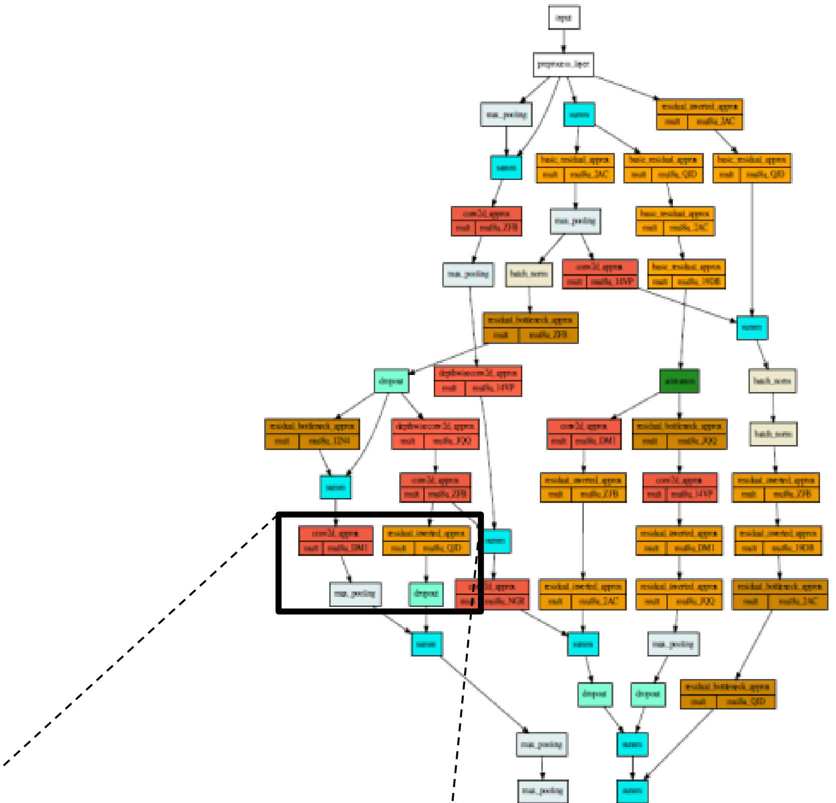
Hypervolume calculated from 10 runs.

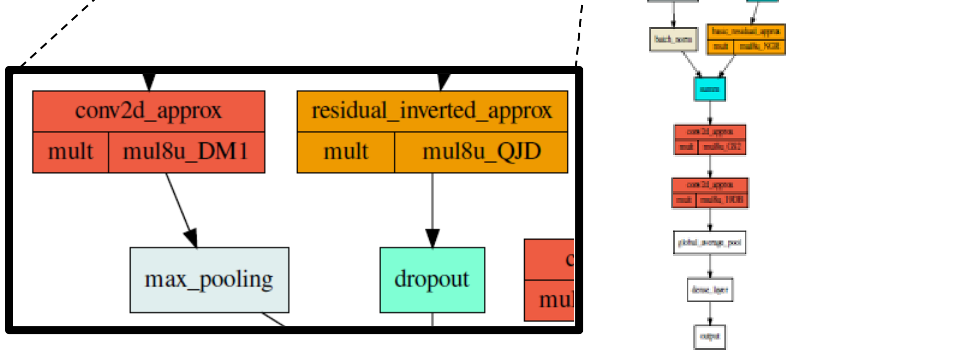
Classification accuracy on CIFAR-10 vs. power consumption (of all multiplications in convolutional layers).

NAS with approximate multipliers (3)



Classification accuracy on CIFAR-10 vs. CNN parameters

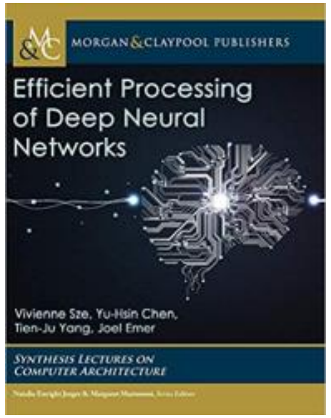




Conclusions

- We surveyed the key elements of recent NAS methods that - to various extents - consider hardware implementation of the resulting CNN. We classified these NAS methods into three major classes:
  - single-objective NAS (no hardware is considered)
  - hardware-aware NAS
  - NAS with hardware co-optimization.
- NAS methods improve design productivity and enable the designer to automatically obtain competitive CNNs for various hardware platforms and data sets.
- The original NAS approach was significantly accelerated by using pre-trained supernets, adopting surrogate models, and incorporating the differentiable architecture search.
- Introducing the hardware search space has led to more efficient implementations of CNNs on particular hardware platforms. However, several search algorithms working in the space of weights, neural architectures, and hardware configurations have to be coordinated, making the entire method complicated.

References: CNNs and their efficient processing


MORGAN & CLAYPOOL PUBLISHERS
Efficient Processing of Deep Neural Networks
Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer
SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE

- Sze V., Chen Y.H., Yang T.J., Emer J.S.: Efficient Processing of Deep Neural Networks. Morgan & Claypool Publishers, 2020
- Sze, V., Chen, Y., Yang, T., Emer, J.S.: Efficient processing of deep neural networks: A tutorial and survey. Proc. of the IEEE, Vol. 105, No. 12, 2295-2329, 2017
- Yanjiao Chen et al.: Deep Learning on Mobile and Embedded Devices: State-of-the-art, Challenges, and Future Directions. ACM Comput. Surv. 53, 4, Article 84, 2020, 37 pages
- Venkataramani S. et al.: Efficient AI System Design With Cross-Layer Approximate Computing. Proc. of the IEEE, Vol. 108, No. 12, 2020
- Mittal S., A survey of FPGA-based accelerators for convolutional neural networks, Neural Comput. Appl., vol. 32, no. 4, pp. 1109-1139, 2020

References: NAS

- NAS
  - T. Elsken, J. H. Metzen, and F. Hutter, Neural architecture search: A survey, J. Mach. Learn. Res., vol. 20, pp. 55:1-55:21, 2019
  - K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, Designing neural networks through neuroevolution, Nature Mach. Intell., vol. 1, pp. 24-35, 2019
  - P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, A comprehensive survey of neural architecture search: Challenges and solutions, ACM Comput. Surveys, vol. 54, no. 4, pp. 1-34, 2021
  - E.-G. Talbi, Automated design of deep neural networks: A survey and unified taxonomy, ACM Comput. Surv., vol. 54, no. 2, pp. 1-37, 2021
  - Vargas-Hákim G.A. et al. A Review on Convolutional Neural Network Encodings for Neuroevolution. IEEE Tr. On Evol. Comp. 26(1), 2022

- HW-Aware NAS:
  - Sekanina L.: Neural Architecture Search and Hardware Accelerator Co-Search: A Survey. IEEE Access, Vol. 9, 2021, p. 151337-151362
  - Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, Naigang Wang: Hardware-Aware Neural Architecture Search: Survey and Taxonomy. Proc. of the Thirtieth International Joint Conference on Artificial Intelligence Survey Track. 2021, Pages 4322-4329

Acknowledgements

Zdeněk Vašíček

Vojtěch Mrázek

Michal Piňos

Filip Vaverka

…

https://ehw.fit.vutbr.cz

Thank you!

This work was supported by the Czech Science Foundation projects:

Automated design of hardware accelerators for resource-aware machine learning, 21-13001S

AppNeCo: Approximate Neurocomputing, 22-02067S