Neural Architecture Search for Efficient Edge AI

Willem Sanberg, PhD.

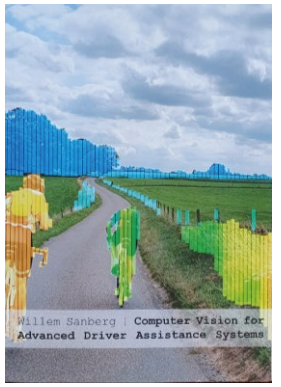
AI Systems R&D

NXP – CTO – AI Competence Center

March 10th, 2025

A little bit about me: Willem Sanberg

- **Academic Background:**
  - BSc. & MSc. in Electrical Engineering
  - PhD. “Computer Vision for Advanced Driver Assistance Systems”
    - Environment perception algorithms using a stereo camera with self-supervised AI and physics models



- **Work Experience:**
  - Sr. Research Scientist AI Systems
    - Scouting & analysing AI research (self & via university collaborations etc.)
    - Translate to NXP requirements, strategy & research projects
    - Execute such a project in a team (small, but cross NXP)
    - Current topics and domain:
      - Automated DNN optimization, Modular Hybrid/Physics-enhanced DNNs, Automotive radar processing

NXP headquarters
(High Tech Campus, Eindhoven)

Agenda for today

1. **The AI design vs. deployment dilemma**

2. **Neural Architecture Search**
   - Effective search space design
   - Search strategy selection
   - Performance estimation strategy selection

3. **Case studies:**
   - Neural Architecture Search for optimal computer vision on NXP HW
   - Bringing other optimizations into the loop: Quantization-aware NAS

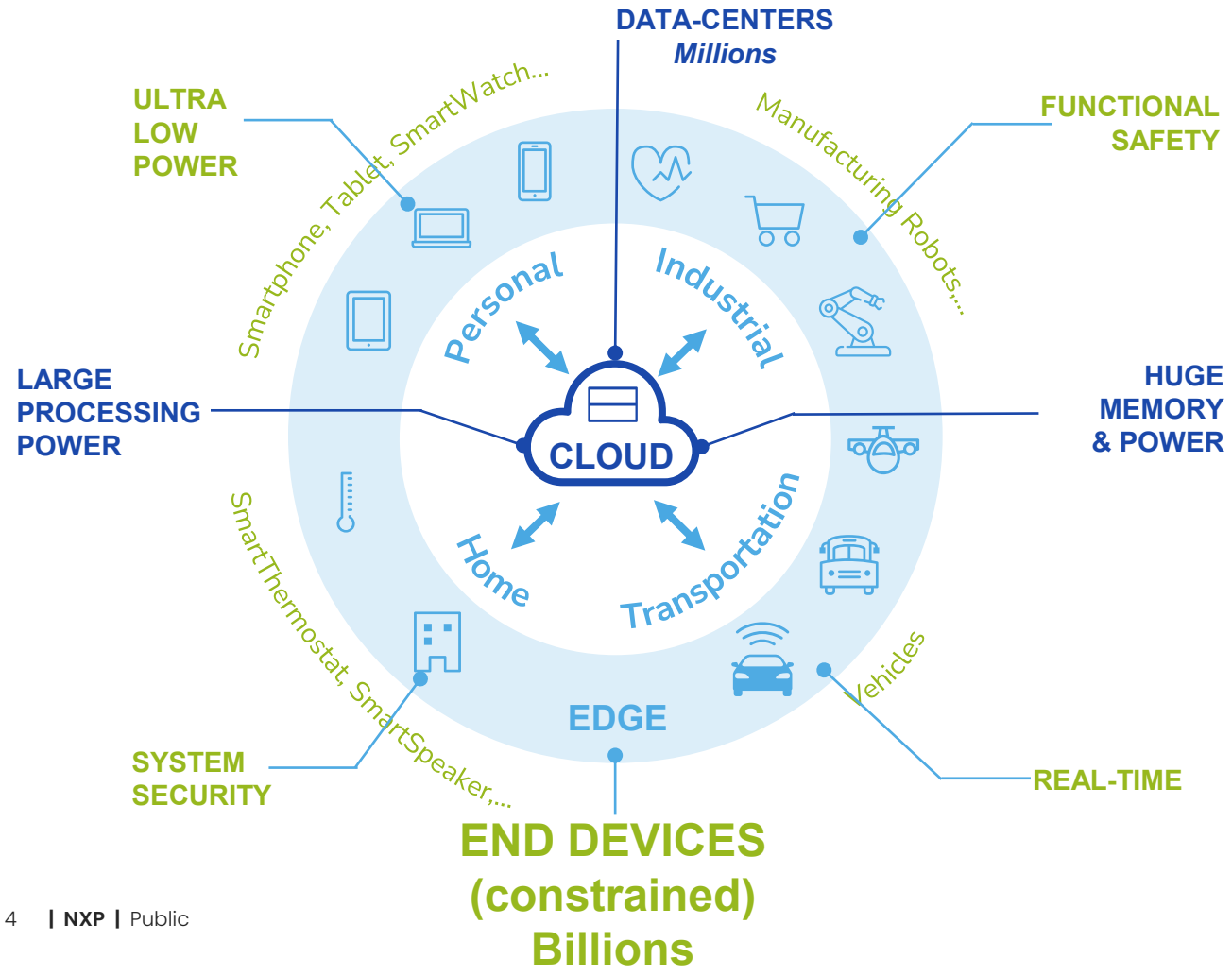
4. **Research outlook & Closing**

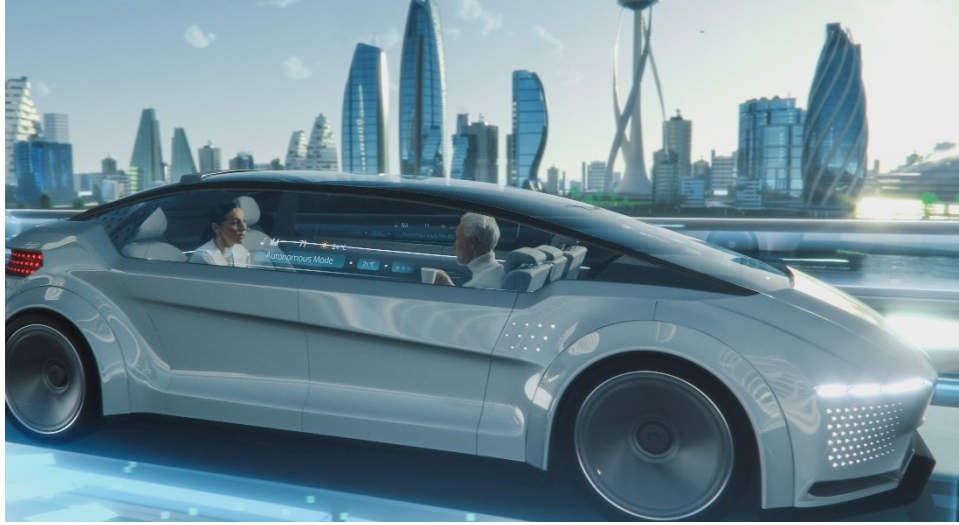
Note: Content and slides build on our bigger team! Especially much thanks to Hiram Rayo Torres Rodriguez, Sebastian Vogel, Yao Lu and Nick van de Waterlaat!

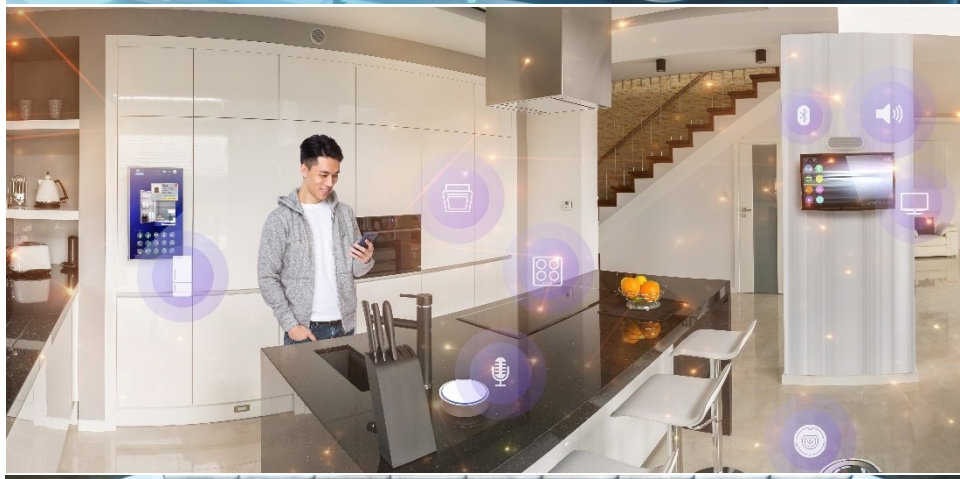
NXP's broad product portfolio serves markets with highly varying requirements and constraints

… but more and more applications want to exploit AI.

Challenge: "Designing AI with multiple conflicting(!?) objectives"

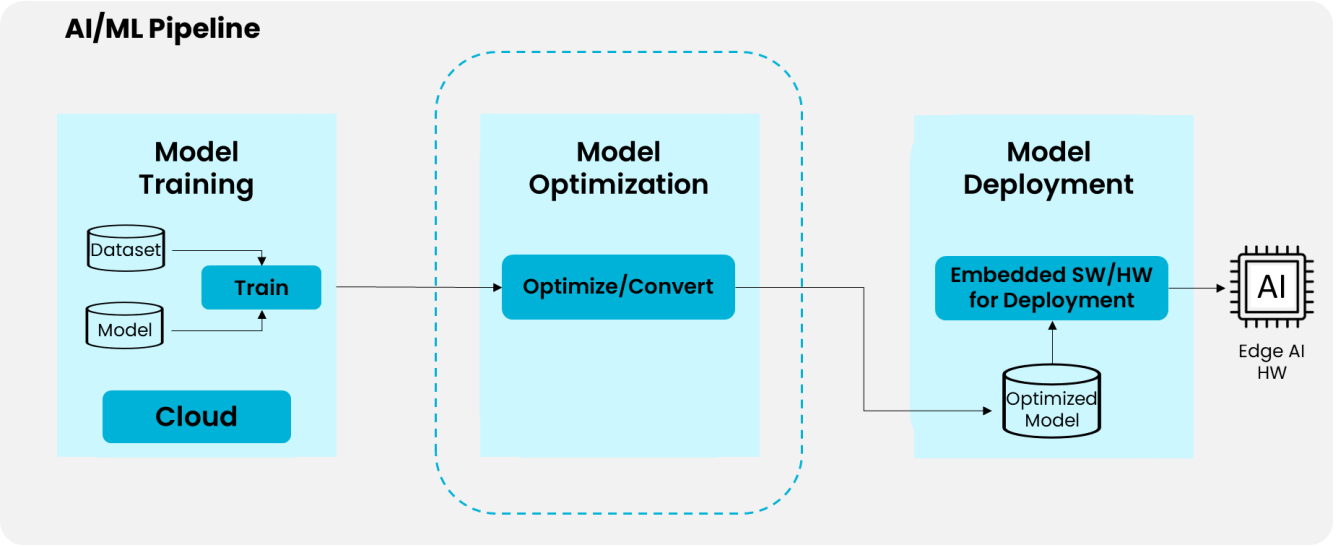






Neural network optimization is key for effectively and efficiently deploying AI models on edge HW

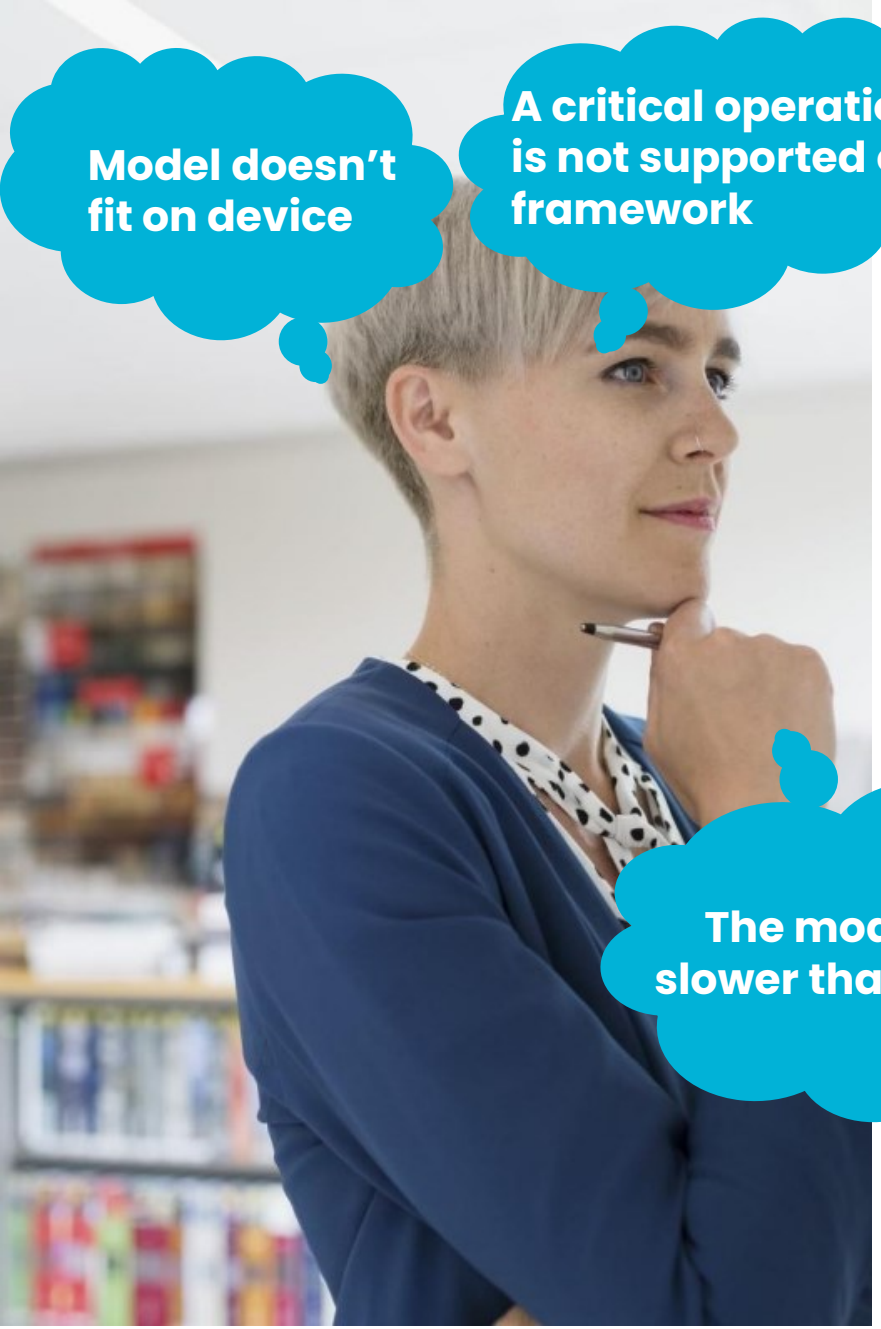
Model optimization aims to improve QoS metrics (e.g., model size, inference latency, energy) of AI/ML models on edge devices



Various optimizations available:

- Graph optimizations
  - E.g., Batch norm and constant folding, layer fusion, etc. (typically done automatically by AI frameworks)
- Quantization
- Pruning

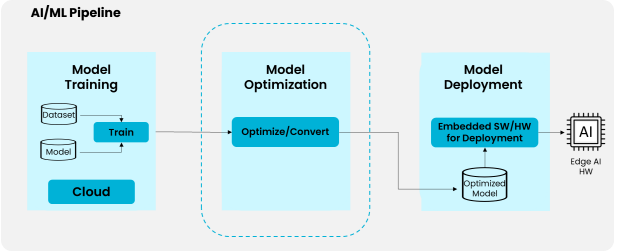
Q: Is this sufficient ?



The AI design vs. deployment dilemma

HW Deployment is infrequently considered when designing AI models

- Often leads to sub-optimal performance, even after applying optimizations (e.g., quantization and/or pruning)



When deployment is not considered during model design, problems occur

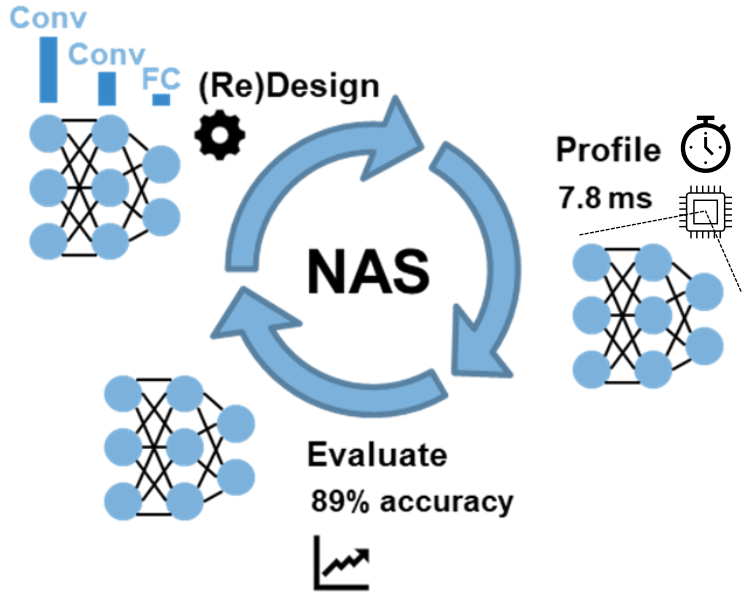
Solution

Neural Architecture Search (NAS)

NAS automatically derives networks highly optimized for a given edge hardware

NAS allows for optimizing multiple objectives,
to **automatically**

- find **effective solutions** for **edge hardware** and reveal the **trade-off between** different **objectives** (e.g., accuracy and model size)



Benefits:

- **Ease-of-Use** → easier to develop **optimal AI**
- **Quality of Service gain** at equal or better accuracy → 'smaller' HW can beat 'bigger' HW
- **Differentiator** → in **SW tooling**

What can we expect of Neural architecture search?
complex yet efficient solutions may 'easily' be found

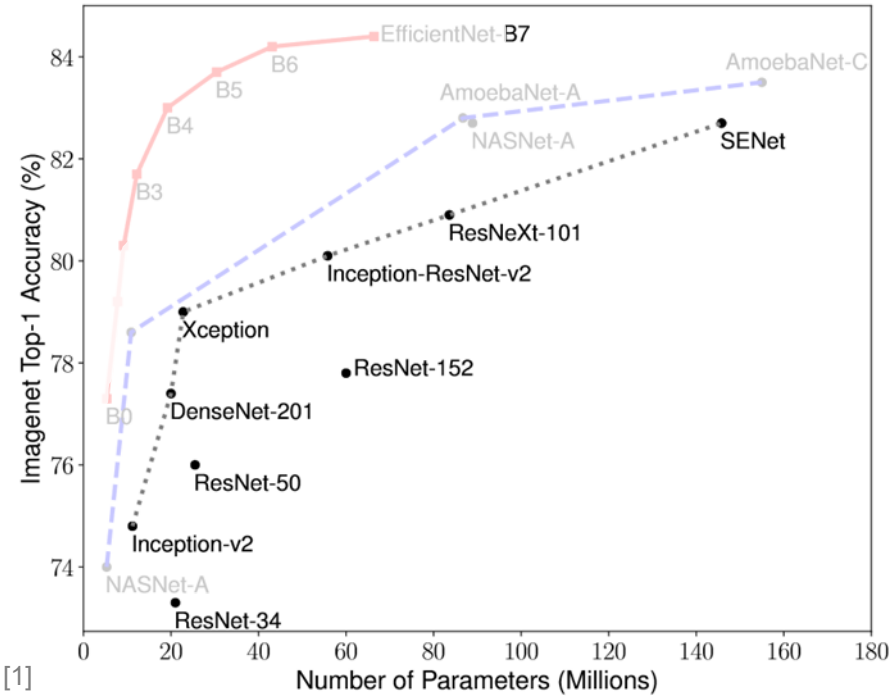
NAS finds significantly more accurate yet efficient solutions than expert data scientist have created

- Hand-crafted architectures (most competitive solutions)
- Early Neural Architecture Search (2017)
- Recent Neural Architecture Search (2019)

Found solutions may incorporate complex structures human experts would not intuitively design

NAS offers an automated approach especially suitable for multi-objective optimization of DNNs

- Interesting hardware-aware secondary objectives: *#params, #ops,* latency, energy, etc.



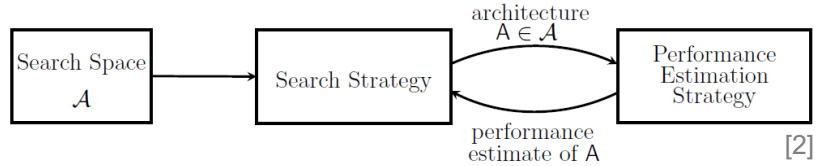
[1]

[1] M. Tan, et al., "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,"

How does NAS work? NAS is coarsely defined by three aspects: 1) Search Space, 2) Search Strategy, 3) Performance Estimation

Methods differ based on three main aspects of NAS

1) Search Space → Which archs. & training-related HPs* can be found?
2) Search Strategy → How to explore the space of possible solutions?
3) Performance Estimation → How does a solution perform in terms of accuracy, secondary hardware-cost, etc.?



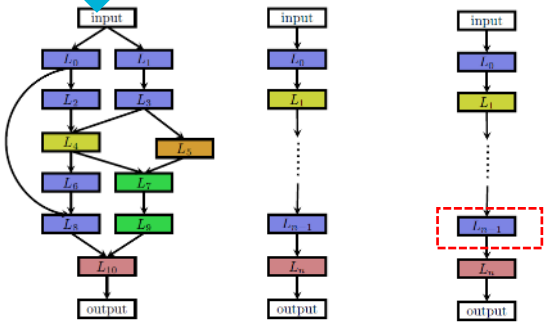
[2]

Design decisions w.r.t these 3 aspects impact on NAS resource requirements and evaluation time

1) Search Space

Smaller space

- Multi-branch networks incl. HPs*
- Chain structured DNNs**
- Only certain layers

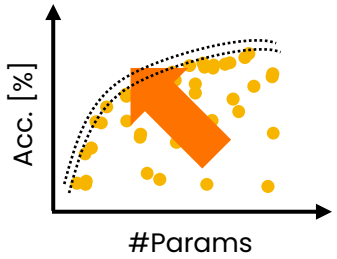


[2]

2) Search strategy

Faster convergence

- Random search
- Reinforcement learning
- Evolutionary/Genetic Alg.
- Bayesian optimization
- Differentiable architecture search



3) Performance Estimation

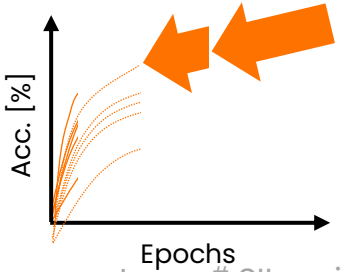
Faster estimation

Primary objectives (task performance):

- Full training
- Low-fidelity training (e.g., early stopping)
- Weight sharing
- Zero-cost proxies

Secondary objectives (resource reqs: memory, latency,…):

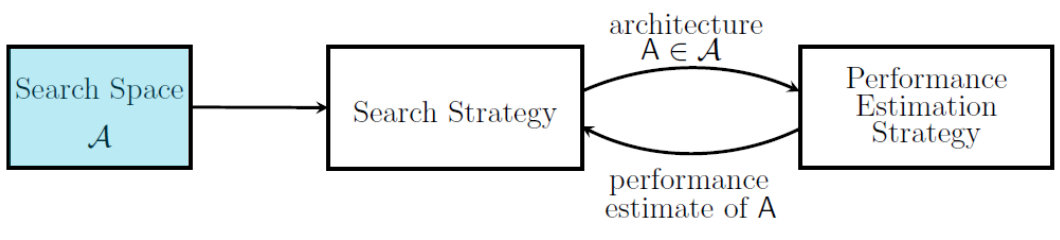
- Measuring (i.e., profiling), SIL#, HIL##



* HP – hyperparameter
** DNN – deep neural network
# SIL – simulator in the loop
HIL – hardware in the loop

[2] T. Elsken, et al., "Neural architecture search: A survey," JMLR '19

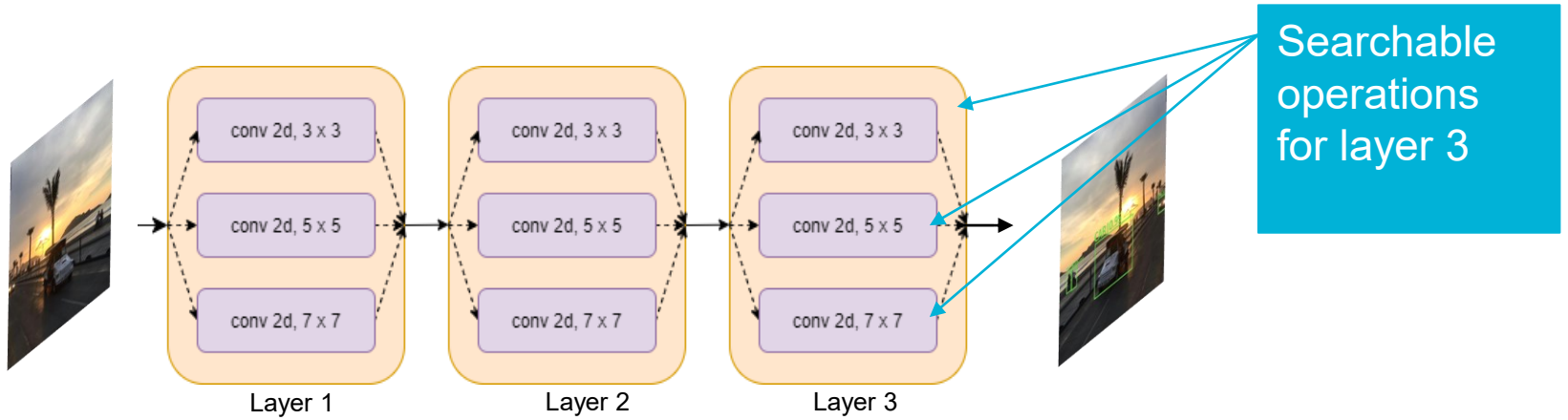
NAS: Search Space Design



Neural Architecture Search: Search Space Design

Most NAS works simply overparameterize a reference model (i.e., the seed network) introducing:

- Searchable kernel sizes, # channels, # layers, etc.



Challenges

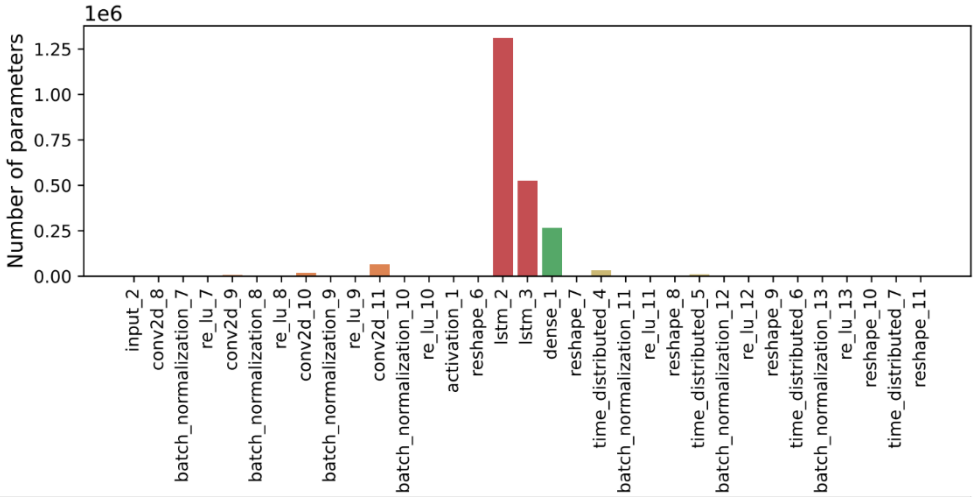
- The bigger the search space, the more exploration may be needed to find good solutions
- Certain operations might not contribute to one or more objectives
- Operations can differ widely in impact on secondary metrics (e.g. inference latency), leading to a poor search space

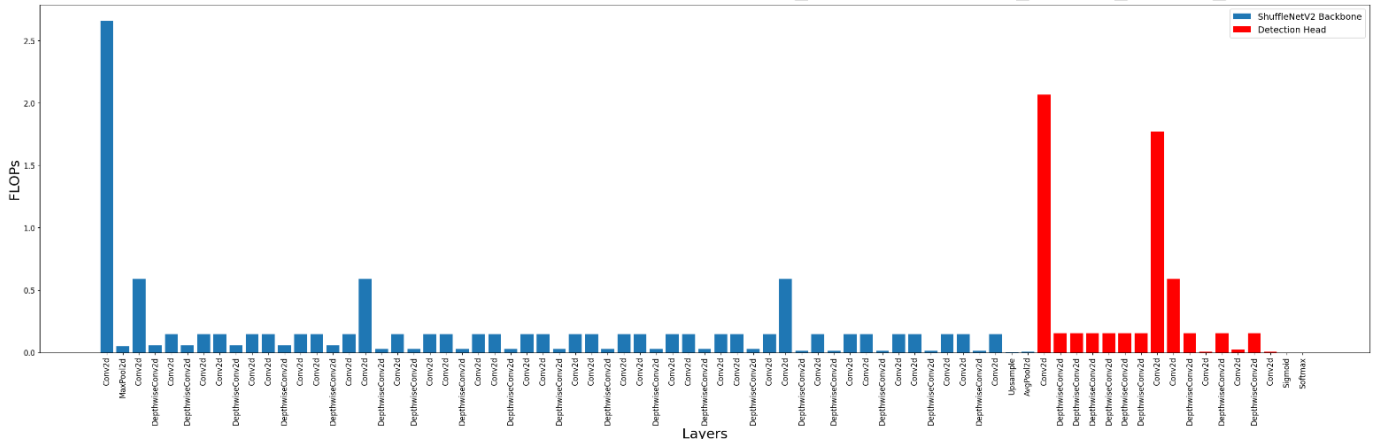
Q: How to address these challenges ?

Analyzing the seed network layer-wise statistics w.r.t secondary metrics can be an effective strategy to design a search space to optimize bottlenecks

In order to find networks that outperform the seed network, carefully consider the the search space design.

- Collect the layer-wise statistics corresponding to the secondary metric of interest (e.g., memory, latency) to find bottlenecks.
- Introduce searchable operations to optimize these bottlenecks





Q: What operations to introduce ?

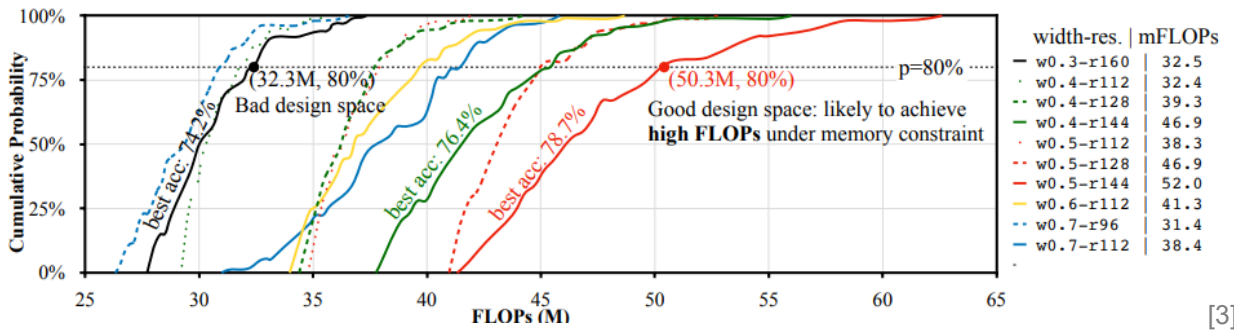
Using proxies for task performance while considering secondary metrics can be an effective strategy to identify what operations to introduce to the search space

Introducing searchable operations using proxies for task performance

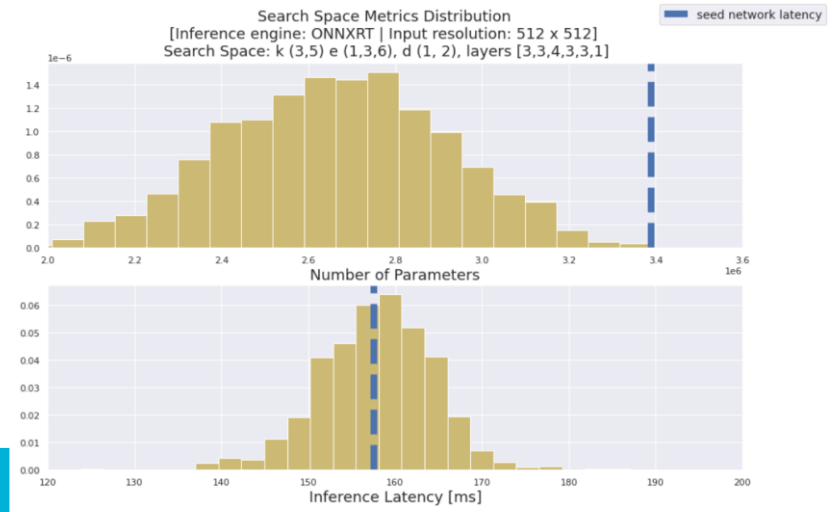
- # FLOPs
- # Params

Consider the impact of these proxies on the secondary metrics

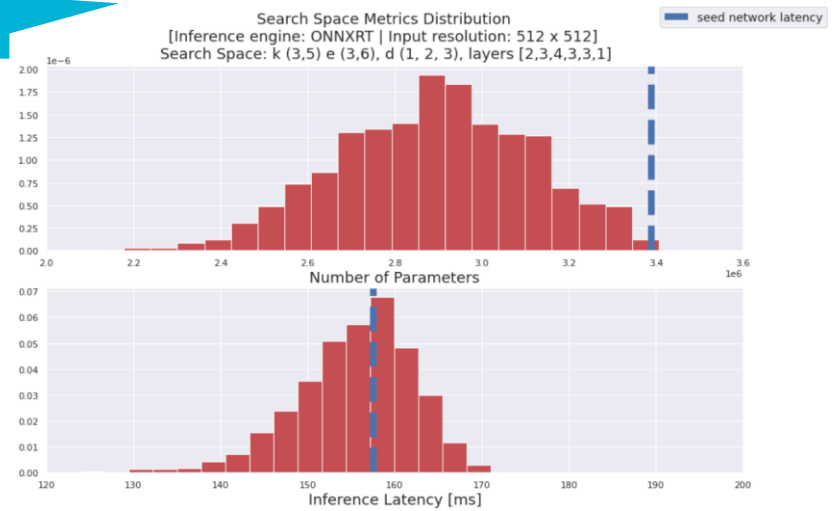
- **idea:** Ensure a search space with a high density of architectures that maximize the proxy, while minimizing the secondary metric of interest



[3]

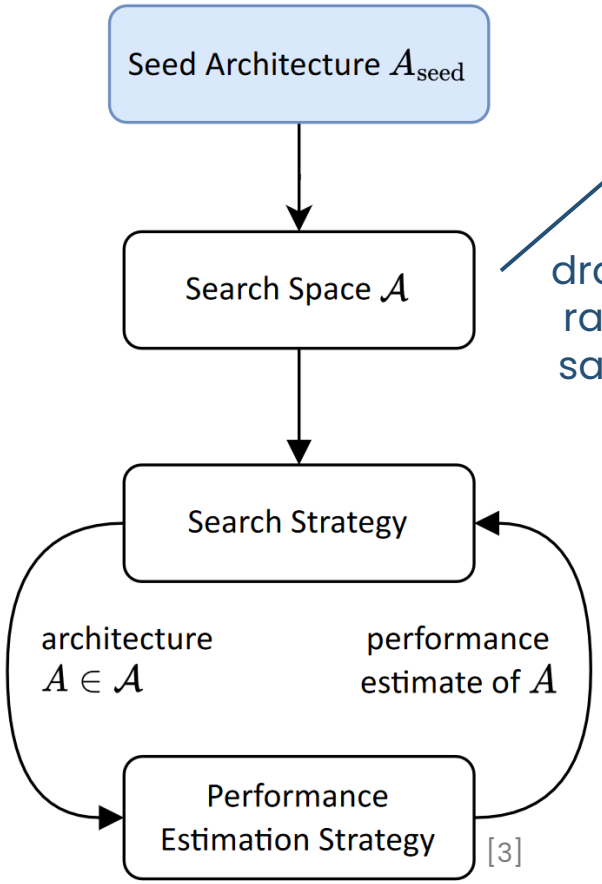


Which one do you prefer?



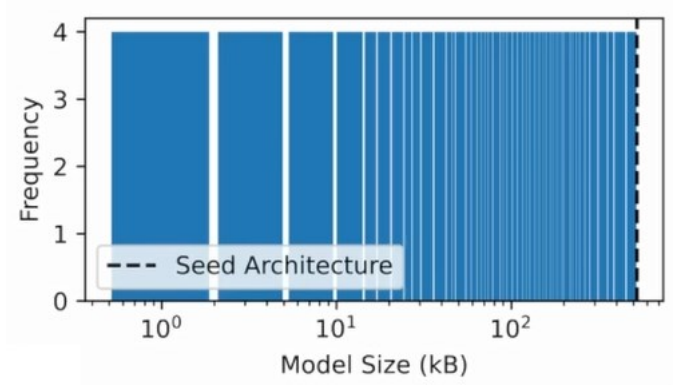
[3] J. Lin, et al., "MCUNet: Tiny Deep Learning on IoT Devices," NeurIPS '20

What about training-related hyperparemeters? Including HPO alongside NAS can lead to substantially smaller networks at equivalent levels of performance



[3]

draw 200 random samples



- **Variation in model size** (*e.g.*, 1 kB, 10 kB, 100 kB)
- **Smaller models** may **benefit** from a **different training paradigm**

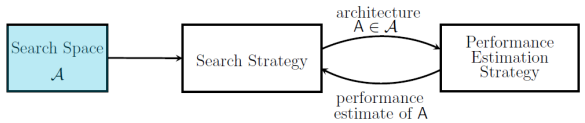
⇒ **Include training-related hyper-parameters** in NAS (*i.e.*, **HPO**)

NAS

Parameter	A_{seed}	Search Space $\mathcal{A}$
#LSTMs	2	[1, 2, 3]
#units in LSTM x	100	[1, 2, ..., 100]

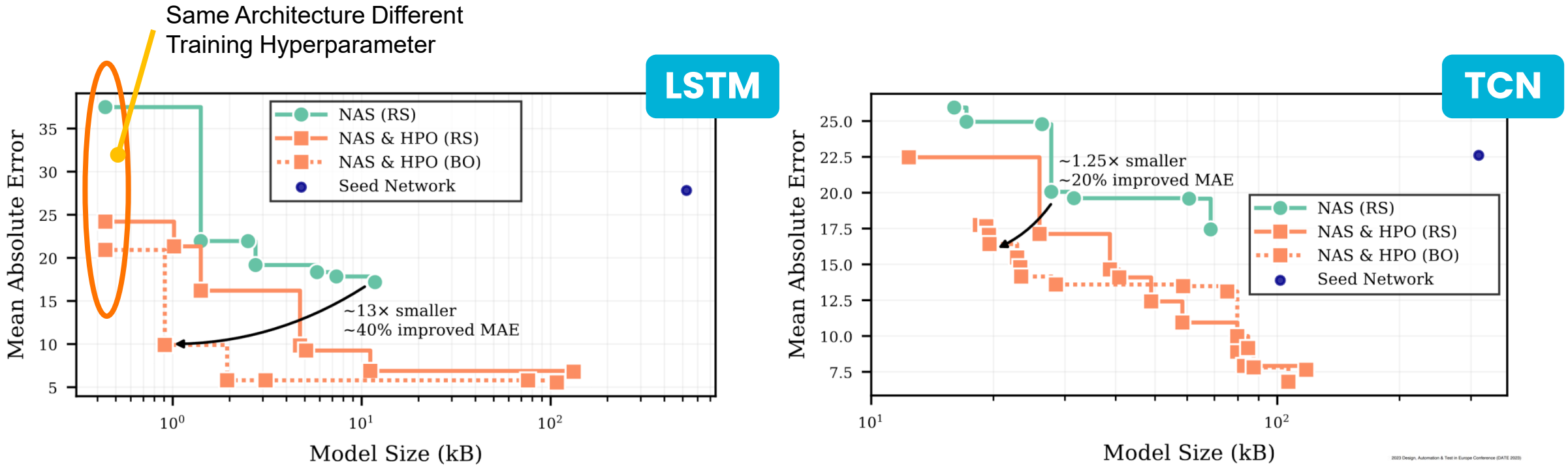
HPO

Parameter	A_{seed}	Search Space $\mathcal{A}$
Learning Rate (LR)	0.0025	[0.01, 0.025, ...]
LR Scheduler	None	[None, Cosine D., Exp. D.]
Optimizer	RMSProp	[RMSProp, Adam]
{kernel, recurrent, bias} regularization for LSTM x	None	[None, L1, L2]
{kernel, recurrent, bias} initialization for LSTM x	...	[glorot_uniform, glorot_normal, ...]



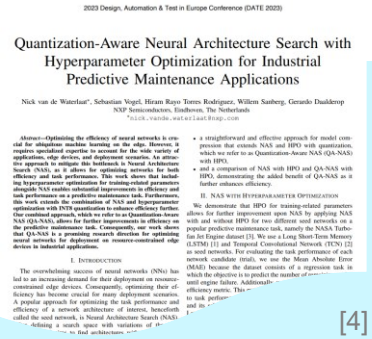
* HPO – hyperparameter optimization

What about training-related hyperparemeters? Including HPO alongside NAS can lead to substantially smaller networks at equivalent levels of performance



Including HPO alongside NAS leads to substantially smaller networks at equivalent levels of performance

Include HPO alongside NAS if a considerable variation w.r.t. efficiency measures (e.g., model size, latency) is present across architectures in the search space


Quantization-Aware Neural Architecture Search with Hyperparameter Optimization for Industrial Predictive Maintenance Applications

[4]

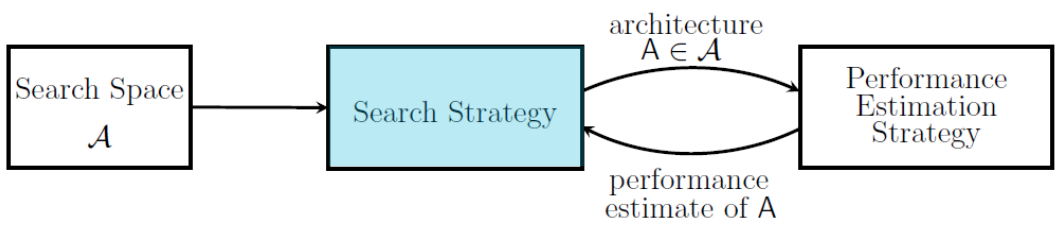
RS – Random Search BO – Bayesian Optimization

Slide courtesy of Nick van de Waterlaat

[4] N. van de waterlaat, et al., "Quantization-Aware Neural Architecture Search with Hyperparameter Optimization for Industrial Predictive Maintainance Applications," DATE '23

NAS:
Search Strategy

(one slide only…)



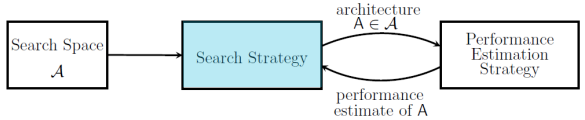
Neural Architecture Search: selecting a search strategy

Consider the search space design when choosing a search strategy, since:

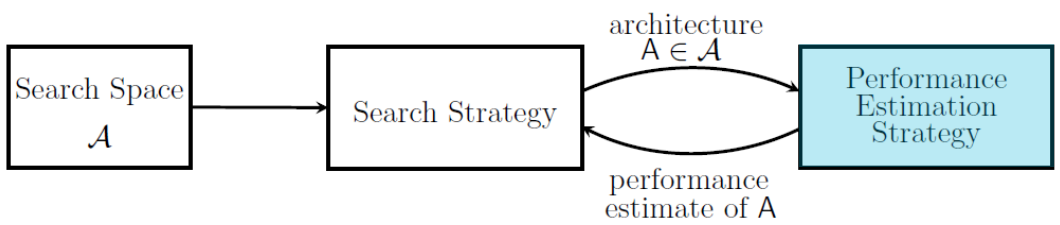
- Bayesian optimization (builds a PDF and hence) may struggle with categorical search spaces
- Evolutionary algorithms (builds a family and hence) can be greedy, which is dangerous in non-homogeneous spaces with local optima
- Differentiable architecture search may be fast but…
  - Limits search space flexibility
  - Hard to enable (engineering-wise) (since it needs to be integrated within the model architecture & training code).

Recommendations:

- Balance the size and 'smartness' over all three blocks – they are heavily intertwined
  - Small search space?
    - Random search may be sufficient!
  - Huge search space and/or costly performance evaluation?
    - Smart, selective search strategy might be required for timely convergence (or… maybe design a multi-stage approach)
- E.g. only rely on differentiable architecture search if the use-case under exploration is highly compute intensive, even after relying on highly efficient performance estimation strategies (more information next)



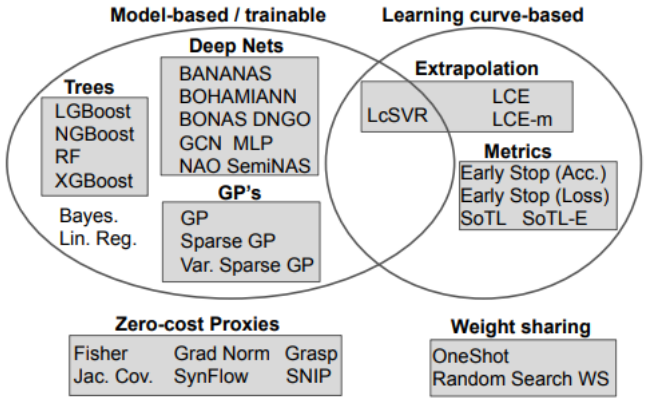
NAS: Performance Estimation



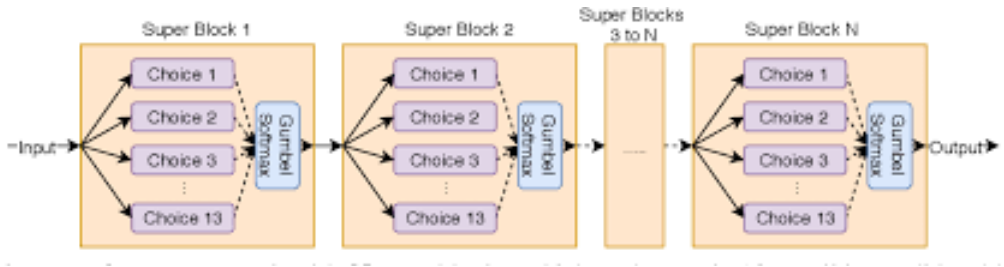
Neural Architecture Search: selecting an effective performance estimation strategy

NAS can be computationally expensive. Thus, a good performance estimation strategy is highly relevant for NAS scalability, **but selecting an optimal one for a given use case is non-trivial**

- **Learning Curve Methods (e.g., Early Stopping):** Can be highly sensitive to the # epochs and/or training stability
- **Model-based Predictors (e.g., DNNs):** May require many training samples + HPO*
- **Weight-sharing:** may exhibit a low correlation between subnet and ground truth performance while limiting search space flexibility and introducing complications in terms of SW enablement
- **Zero-cost Proxies**[5] **(e.g., # FLOPs):** May exhibit wildly different correlations between estimated and ground truth performance across tasks



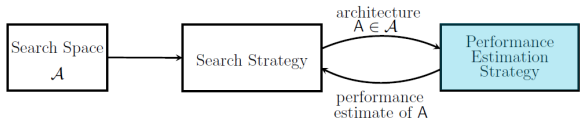
[5]



[6]

‡ FLOP: Floating point operations

* HPO: Hyperparameter Optimization



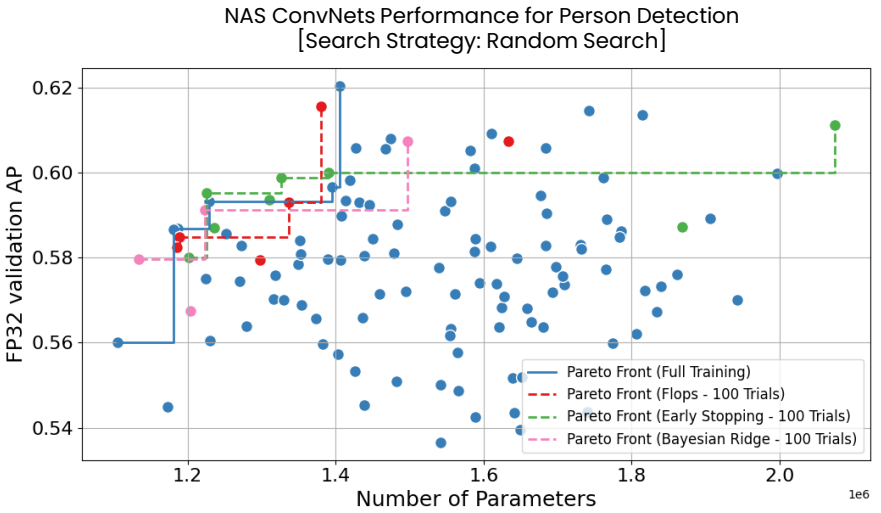
[5] C. White, et al., "How Powerful are Performance Predictors in Neural Architecture Search?," NeurIPS '21

[6] A. Shaw, et al., "SqueezeNAS: Fast neural architecture search for faster semantic segmentation," ICCV '19

Neural Architecture Search: selecting an effective performance estimation strategy

Exploring various performance estimation strategies can be an effective way to identify the most promising strategy for a given use case

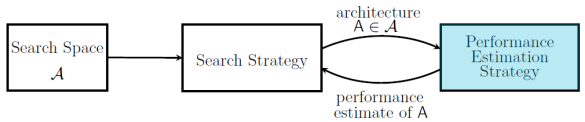
Despite being imperfect, performance estimation strategies can achieve competitive performance compared to full training, while achieving substantial speedups



	Search Time (Seconds)	Search Speedup	Post Search Training Time (Seconds)	Total (seconds)	Overall Speedup
Full training	96,000	1.0	N/A	96,000 (26.6 Hours)	1.0
Early stopping	20,000	**x4.8**	14,400	34,400 (9.55 Hours)	**x2.79**
Bayesian Ridge	30	**x3,200**	6,400	6,430 (1.78 Hours)	**x14.93**
FLOPs	6	**x16,000**	9,600	9,606 (2.66 Hours)	**x10**

‡ FLOP: Floating point operations

* HPO: Hyperparameter Optimization



What about performance estimation for secondary metrics?

How to assess deployment cost (e.g., inference latency) in a scalable manner ?

- **Hardware-in-the-loop (HIL):** accurate but slow, and requires access to physical resources
- **Lookup Tables (LUT):** Fast and accurate, but set to a fixed set of options
- **Surrogate models:** Fast and unconstrained but additional problems to solve…
  - Model selection
  - Architecture encoding
  - # samples required for fitting the surrogate

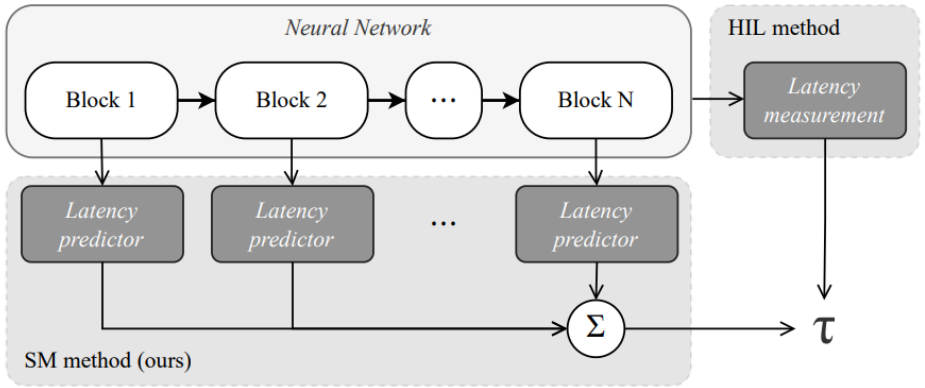


Fig. 2. Schematic showing the assessment methodolgy of full-network latency τ using HIL profiling and our proposed block-level based estimate.

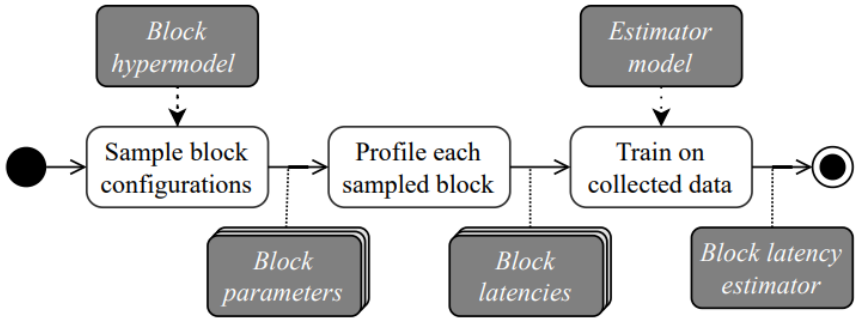
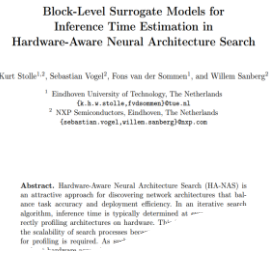


Fig. 3. UML activity diagram showing the end-to-end process of sampling configurations of a block, profiling these samples, and fitting a predictor model on this data. This yields a latency predictor that is specifically designed for a single block type and the hardware platform it was profiled on.


Block-Level Surrogate Models for Inference Time Estimation in Hardware-Aware Neural Architecture Search

[7]

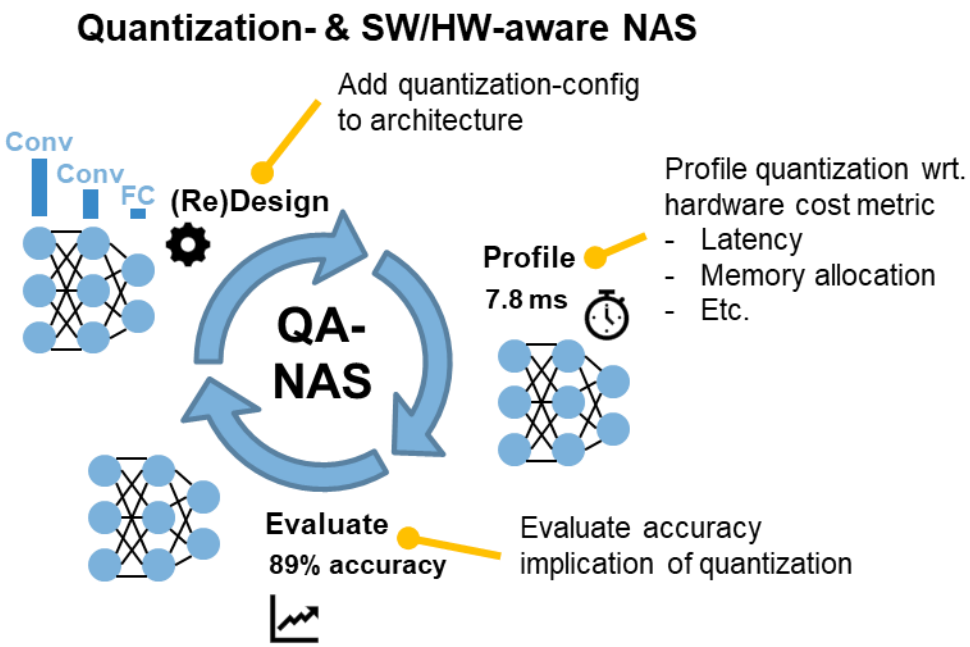
[7] K. Stolle, et al., "Block-Level Surrogate Models for Inference Time Estimation in Hardware-Aware Neural Architecture Search," ECML '22

Bringing optimization methods together:
Quantization-Aware Neural Architecture search

Problems:

- Quantization typically leads to reduced task performance. Furthermore, optimal full precision (FP32) models might be sub-optimal (e.g., in terms of inference latency) when quantized due to SW/HW specific kernel implementations.

- Quantizing all layers of a Neural Network to 8 bits can be suboptimal, hence, recent work explores few-bit mixed-precision quantization (FB-MP) to quantize certain layers to fewer bits to further optimize NN models. How to derive the FB-MP policy ?



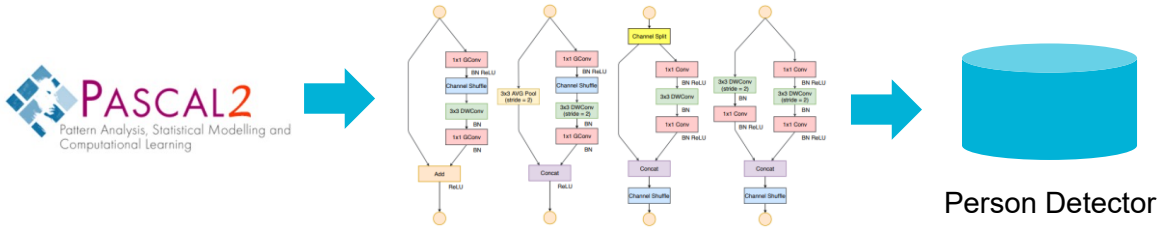
Solution: Quantization-Aware NAS

Case study:
Enabling optimal Person Detection on i.MXRT117x via QA-NAS

Person Detection Use-Case

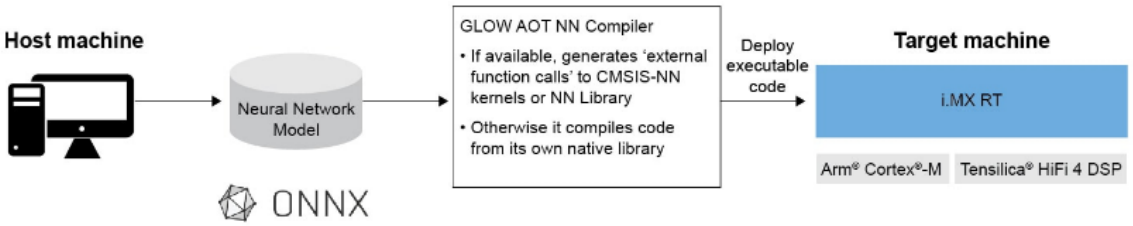
Algorithm Development

- **FastestDet***: A lightweight object detection network for person detection based on a **ShuffleNetv2**[8] backbone and trained on **PASCAL VOC** dataset.



Algorithm Deployment

- **eIQ® Inference with Glow neural network (NN) compiler:** The model is converted to the executable code through NXP's eIQ Glow NN compiler and deployed on the i.MXRT117x MCU.



Baseline Network

Real-time performance requirement:

100ms (10 FPS) on i.MXR1170

*: Open-source repository: https://github.com/dog-qiuqiu/FastestDet

[8] N. Ma, et al., "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design," ECCV'18

i.MX RT1170: 1 GHz Crossover MCU with ARM Cortex Cores

i.MX RT1170* Crossover MCUs are **dual-core devices featuring an Arm® Cortex®-M7 and Arm® Cortex®-M4** for real-time microcontroller (MCU) performance and high integration for **automotive, industrial and IoT applications**

Features:

- Arm Cortex-M7 at 1 Ghz and Arm Cortex-M4 at 400 MHz
- 2 MB SRAM with 512 KB of TCM for Arm Cortex-M7 and 256 KB of TCM for Cortex-M4
- Zephyr RTOS support
- Advanced security, including secure boot and crypto engines, and is part of the EdgeLock® Assurance program
- 2 x Gb ENET with AVB and TSN
- 2D GPU
- MIPI® CSI/DSI

*:https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/i-mx-rt-crossover-mcus/i-mx-rt1170-1-ghz-crossover-mcu-with-arm-cortex-cores:i.MX-RT1170

We start by investigating the seed network training to identify a suitable NAS method

We analyze the seed network training to select a suitable NAS method

- The relatively little training time allows us to work with **sampled based NAS** → more flexible search space

We further improve the training pipeline to speedup NAS and/or allow further exploration

- By carefully selecting appropriate training HP
  - ➢ Learning rate schedule, #epochs

	Total Number of Epochs	Training Time [GPU Hours]	Memory [MiB]	AP [%]
FastestDet	300	0.410	3535	61.4

2x faster convergence

	Total Number of Epochs	Training Time [GPU Hours]	Memory [MiB]	mAP [%]
FastestDet	120	**0.205**	3535	61.4

Model is evaluated on 256x320 input resolution on the Pascal VOC dataset

AP is evaluated at 0.5 IoU

We then investigate the best quantization setting and their real-time performance on the target device (imxrt1170)

We evaluate the seed network performance across various quantization settings to identify the right setting before performing NAS.

FP32 AP (0.5 IoU)	Quantization Schema	Per Tensor/channel	INT8 AP (0.5 IoU) [%]	Latency [ms]
0.614	Asymmetric	Per Tensor	61.0	769
0.614	**Asymmetric**	**Per Channel**	**60.6 (+0.4)**	**807**
0.614	Symmetric	Per Tensor	60.2	727
0.614	**Symmetric**	**Per Channel**	**60.5 (+0.3)**	**747**
0.614	Symmetric with Uint8	Per Tensor	60.4	728
0.614	**Symmetric with uint8**	**Per Channel**	**60.4**	**793**
0.614	Symmetric with power2 scale	Per Tensor	58.9	237 (w/ cmsis-nn)
0.614	**Symmetric with power2 scale**	**Per Channel**	**60.2 (+1.3)**	**229** (w/ cmsis-nn*)

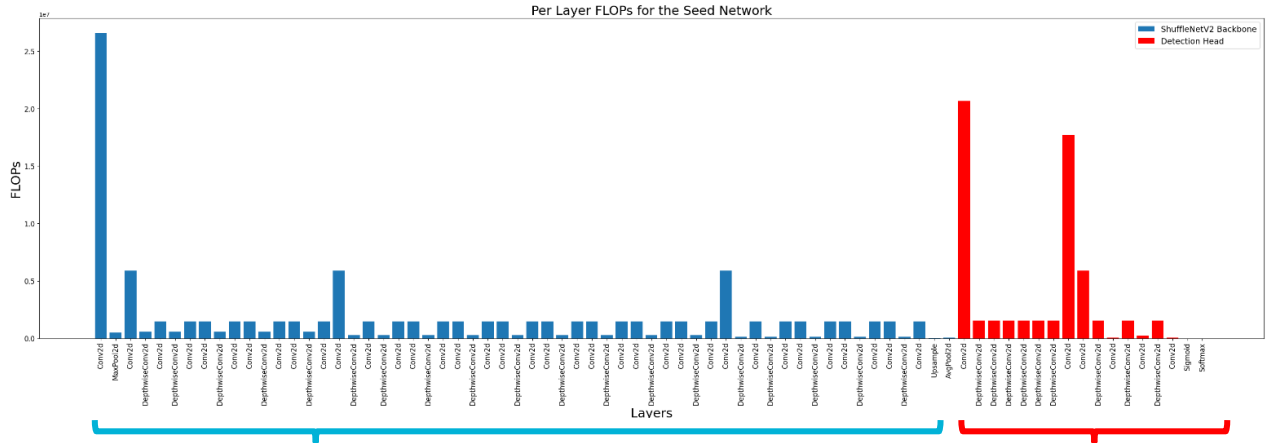
Models are evaluated on 256x320 input resolution on the Pascal VOC dataset

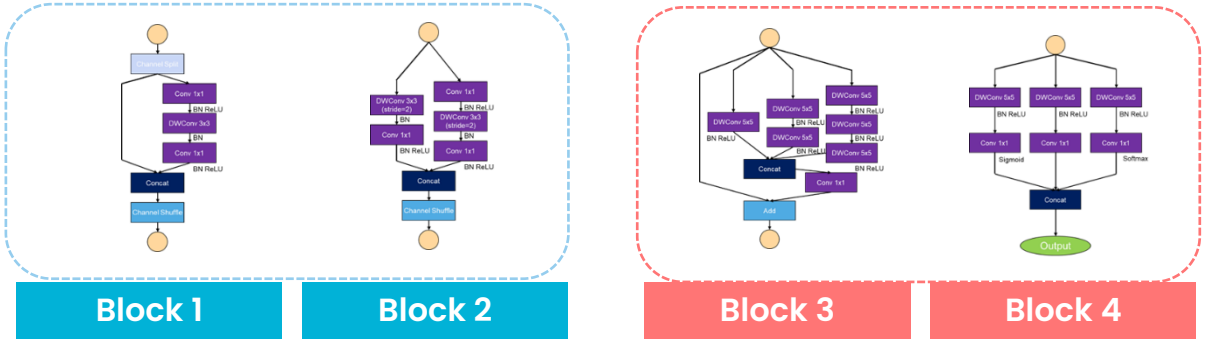
*https://github.com/ARM-software/CMSIS-NN

We analyze the seed network layer-wise distribution to design a search space to minimize number of flops as proxy for inference latency

- **We find that 62.6% of compute is located in the backbone, with only 37.4% in the detection head.**
  - The majority is contributed by the first convolution, with the rest of the backbone being rather cheap.
  - The backbone is initialized with pre-trained weights which we dont know where they come from → it is not easy to search for it.

- **We focus on the detection head and design a search space introducing searchable parameters to minimize the number of FLOPs while maximizing task performance.**





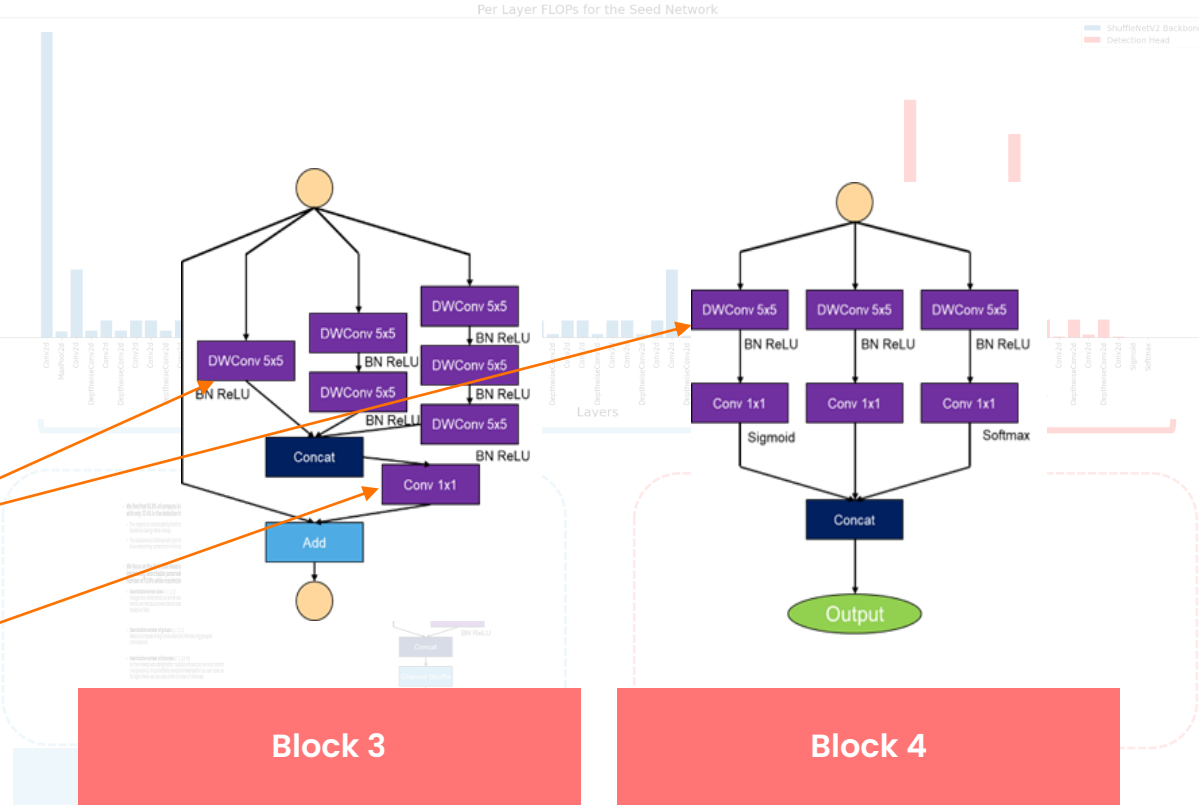
Metric	ShuffleNetV2 Backbone	Detection Head	Total
FLOPs	~97.9 M	~ 58.4 M	~ 156.4 M

FLOPs are calculated using an input resolution of 256 x 320 following the repository configuration

We analyze the seed network layer-wise distribution to design a search space to minimize number of flops as proxy for inference latency

- **We find that 62.6% of compute is located in the backbone, with only 37.4% in the detection head.**
  - The majority is contributed by the first convolution, with the rest of the backbone being rather cheap.
  - The backbone is initialized with pre-trained weights which we dont know where they come from → it is not easy to search for it.

- **We focus on the detection head and design a search space introducing searchable parameters to minimize the number of FLOPs while maximizing task performance.**
  - **Searchable Kernel sizes:** $k \in \{3, 5\}$
    Images are rather small, so kernel size 5 might be unnecessary. Hence, we introduce lower kernel sizes and search for an optimal receptive field.
  - **Searchable number of groups:** $g \in \{1, 2\}$
    Reduce compute of big convolutions by introducing grouped convolutions.
  - **Searchable number of channels:** $C \in [24, 96]$
    As the network was designed for multiple classes, but we only care for one (persons), it is potentially overparameterized for our use-case, so its highly likely we can reduce the number of channels.



Metric	ShuffleNetV2 Backbone	Detection Head	Total
FLOPs	~97.93 M	~ 58.429	~ 156.36 M

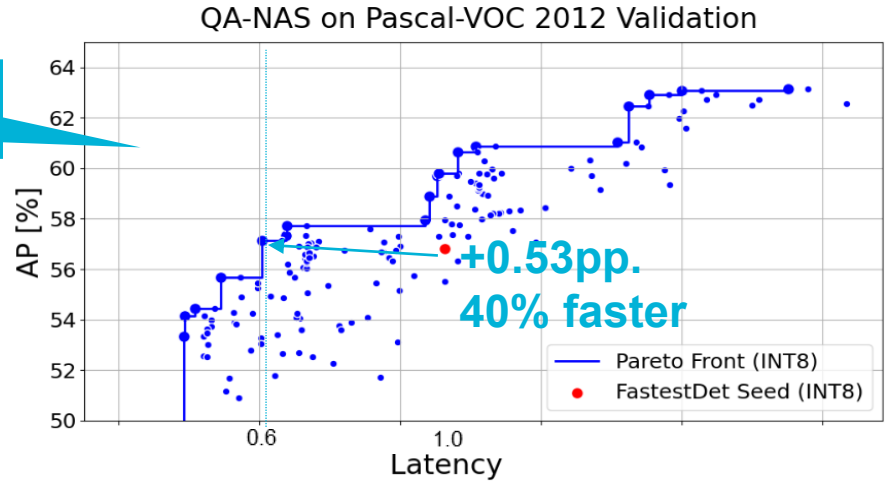
FLOPs are calculated using an input resolution of 256 x 320 following the repository configuration

Via NAS, we reduce inference latency by ~40% while keeping similar task performance compared to the seed network

Search space:

- Kernel size $\in \{2, 3, 4, 5\}$
- Groups $\in \{1, 2, 4, 8\}$
- Channels $\in [24, 96]$
- Input width $\in [220, 320]$
- Aspect ratio $\in [0.5625, 0.6, 0.75, 0.8, 1.0]$

By optimizing the task-head only (37% of original FLOPS)



Search strategy:

- Multi-objective Tree Parzen Estimation (MOTPE[10]).
  - 120 trials

Performance estimation:

- Full model training

Search time:

- ~2.5 GPU days

Baseline

NAS Version

[9] T. Akiba, et al., "Optuna: A next-generation hyperparameter optimization framework," KDD '19
[10] Y. Ozaki, et al., "Multiobjective tree-structured parzen estimator for computationally expensive optimization problems," GECCO '20

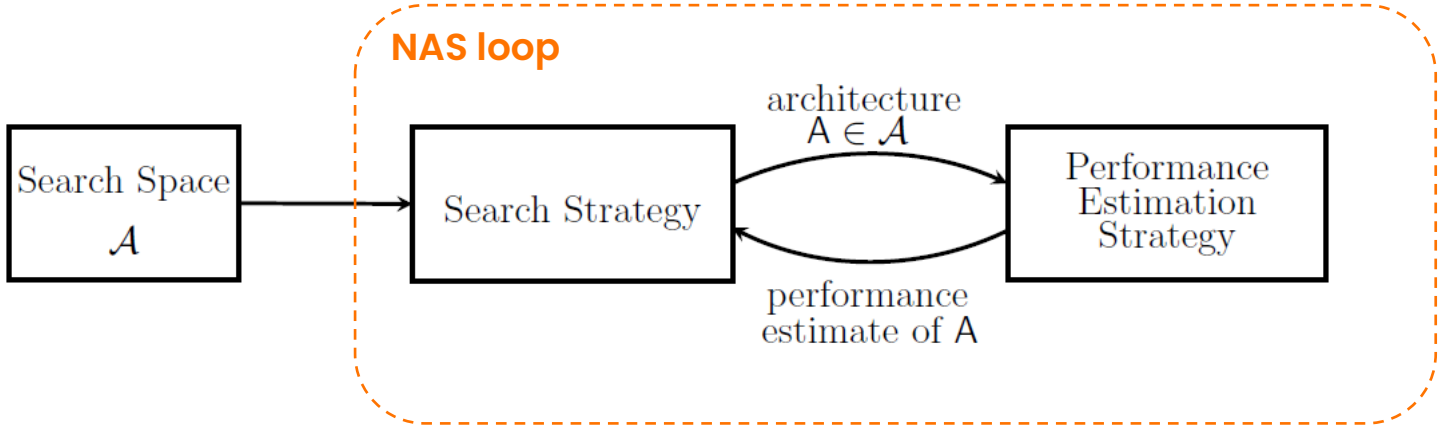
Case study:

Scaling up Quantization-Aware NAS for Efficient Deep Learning on the Edge

Traditional sample-based NAS can be unsuitable to large scale / compute intensive tasks

Sample-based NAS:

- Sample architectures from the search space and then train each of them from scratch to validate their performance
- Scaling NAS to large scale / compute intensive tasks (e.g., semantic segmentation) can quickly become intractable due to the long training time incurred by training each candidate solution in isolation



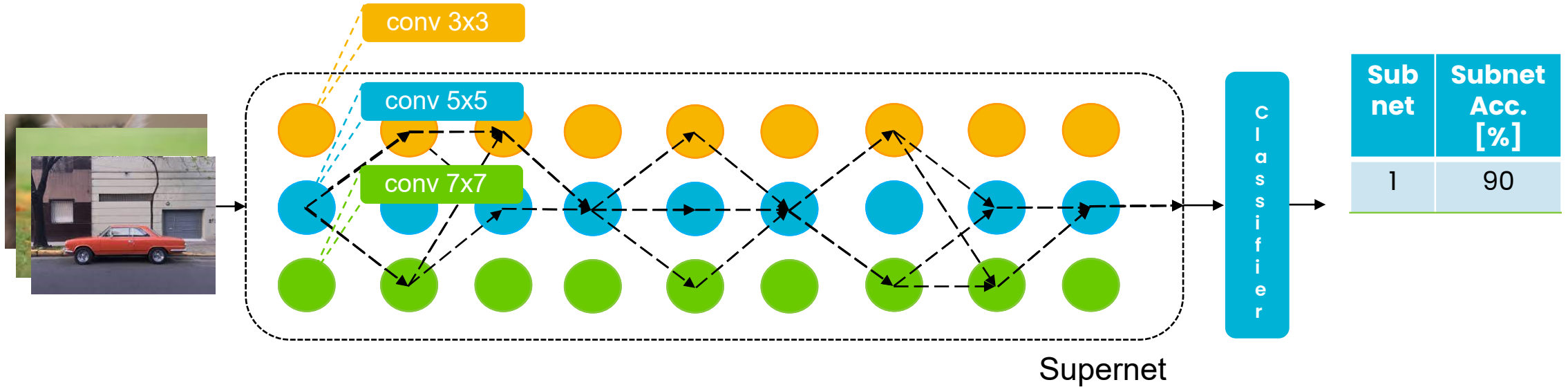
Weight-sharing-based NAS introduces a solution for large scale / compute intensive tasks

Weight-sharing-based NAS (Two-stage approach):

1. Train a **super-network** which encompasses all candidate architectures as subnets.
   - ➢ Subnets share weights during training → reduced training time, as only the supernet needs to be trained.
2. Evaluate and rank the performance of subnets to identify the best solutions.
   - ➢ Use subnet performance as a proxy for final performance → Select the subnet which exhibits the highest performance

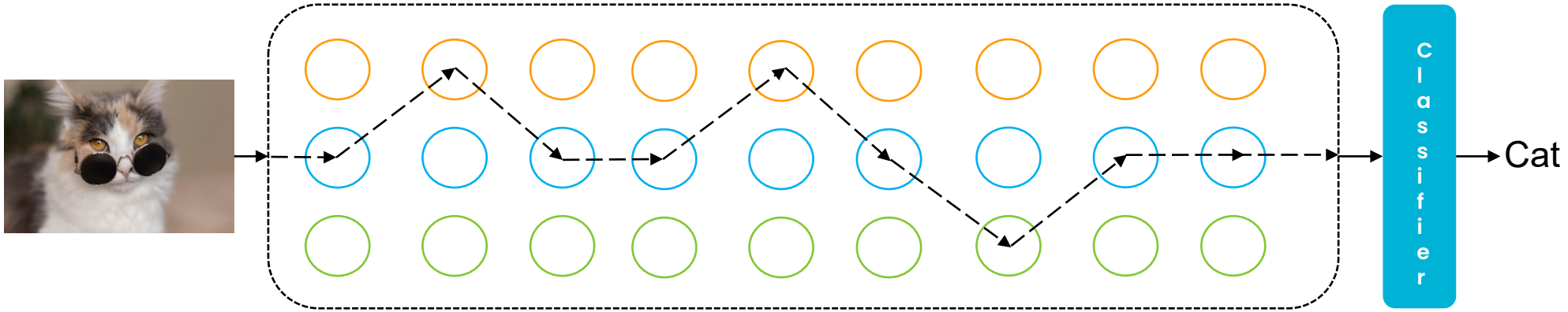
Finally, train the best performing solution(s) stand-alone to full convergence.

Challenge: Unproper/insufficient supernet training can lead to a low-ranking correlation between subnet and stand-alone performance, leading to selecting sub-optimal solutions.



Block-wise NAS introduces a solution to improve ranking correlation, resulting in better solutions compared to traditional supernet-based NAS

Block-wise NAS[11] (BWNAS) Key idea:
Divide the supernet into several blocks and search for these independently.



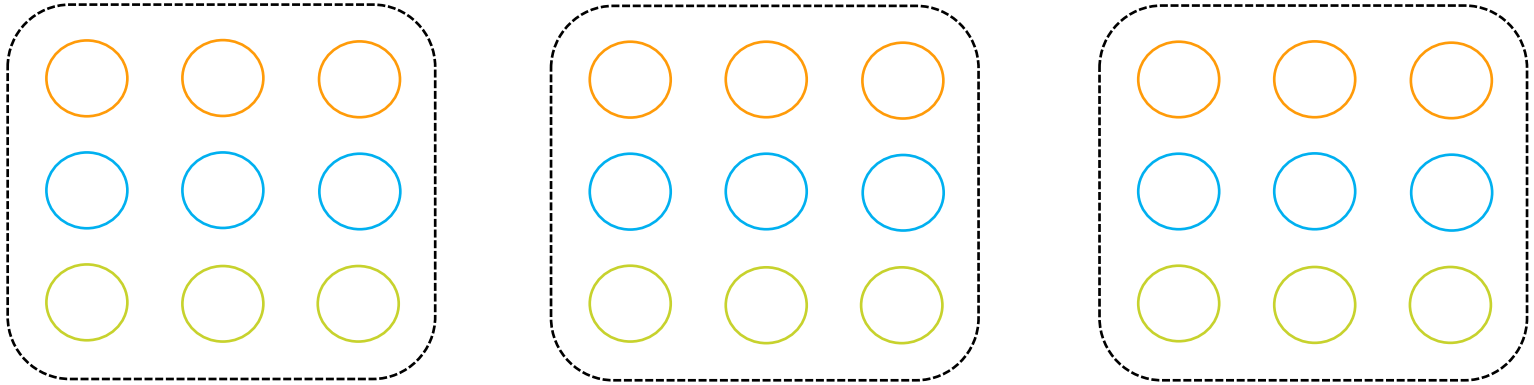
Search space: C^d

- C: candidate (searchable) operation
- d: supernet depth (i.e., total number of layers)

[11] C. Li, et al., "Blockwisely Supervised Neural Architecture Search with Knowledge Distillation," CVPR '20

Block-wise NAS introduces a solution to improve ranking correlation, resulting in better solutions compared to normal supernet-based NAS

Block-wise NAS[11] (BWNAS) Key idea:
Divide the supernet into several blocks and search for these independently.



Search space is reduced following: $C^{d_i}/(\prod_{i=0}^{N} C^{d_i})$

- C: candidate (searchable) operation
- d: **block** depth (i.e., total number of layers)

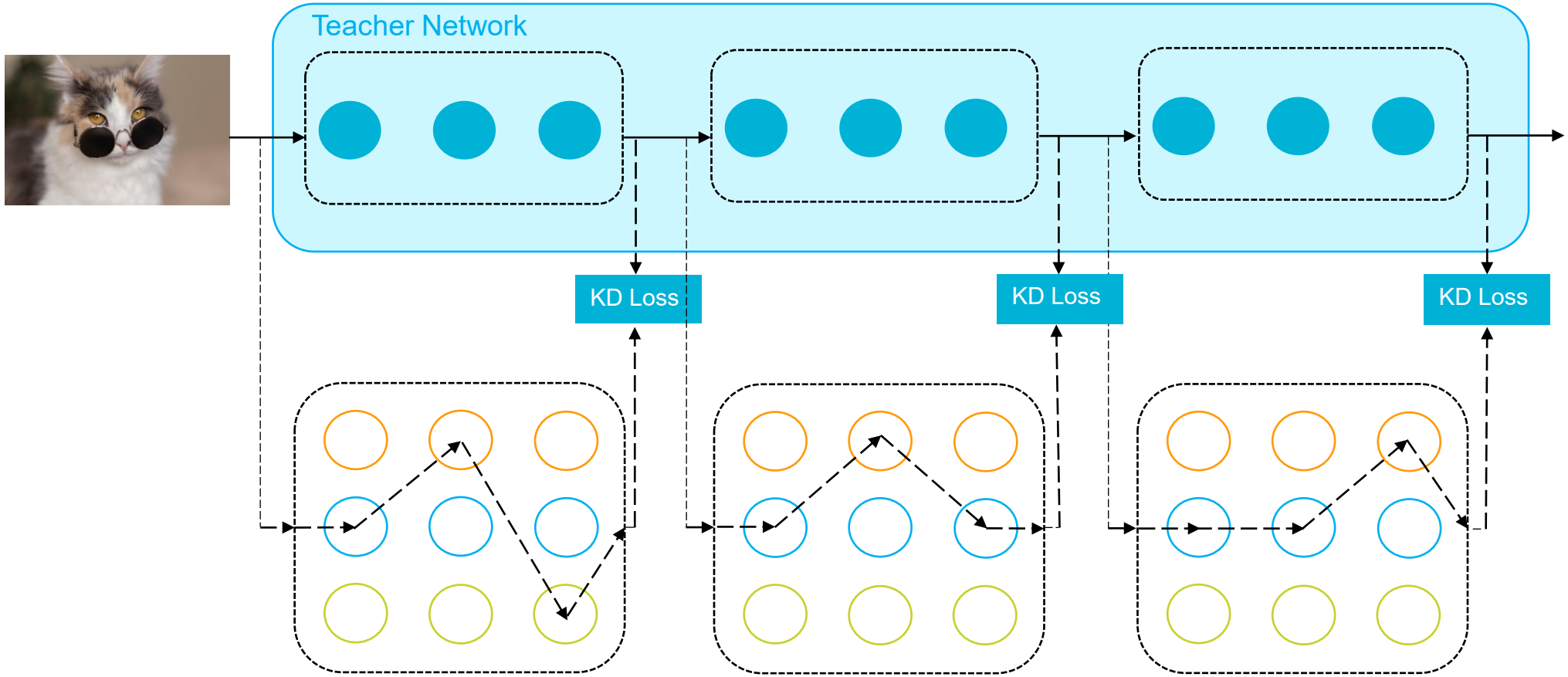
Optimizing blocks in isolation allows for subnets to better optimized, thus improving ranking accuracy.

Q: How to train the blocks ?

[11] C. Li, et al., "Blockwisely Supervised Neural Architecture Search with Knowledge Distillation," CVPR '20

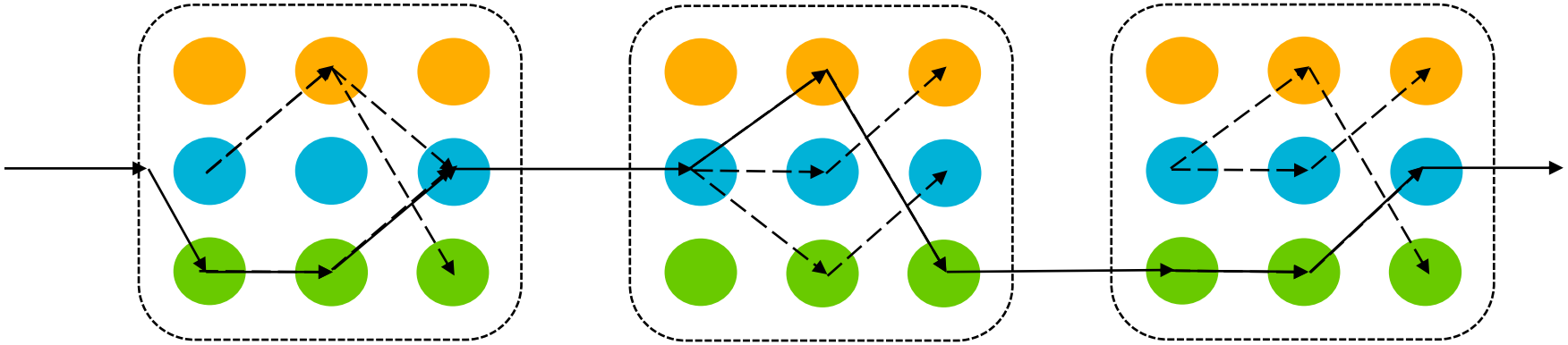
Block-wise NAS introduces a solution to improve ranking correlation, resulting in better solutions compared to normal supernet-based NAS

Blocks are trained independently with supervision from the teacher model using feature-based **Knowledge Distillation (KD)** with the objective of minimizing the loss between the teacher and the student feature maps.



Block-wise NAS introduces a solution to improve ranking correlation, resulting in better solutions compared to normal supernet-based nas, but fails to address quantization

After training, the KD loss of each student candidate in each block is recorded to populate a lookup table (LUT)



LUT(s)

Student	KD Loss
1	0.43
2	0.32
3	0.28

Student	KD Loss
1	0.40
2	0.33
3	0.24

Student	KD Loss
1	0.23
2	0.15
3	0.22

The highest performing architecture can be found by selecting the subnets which exhibit the lowest loss on each block

Finally, we train the found architecture from scratch on the target dataset

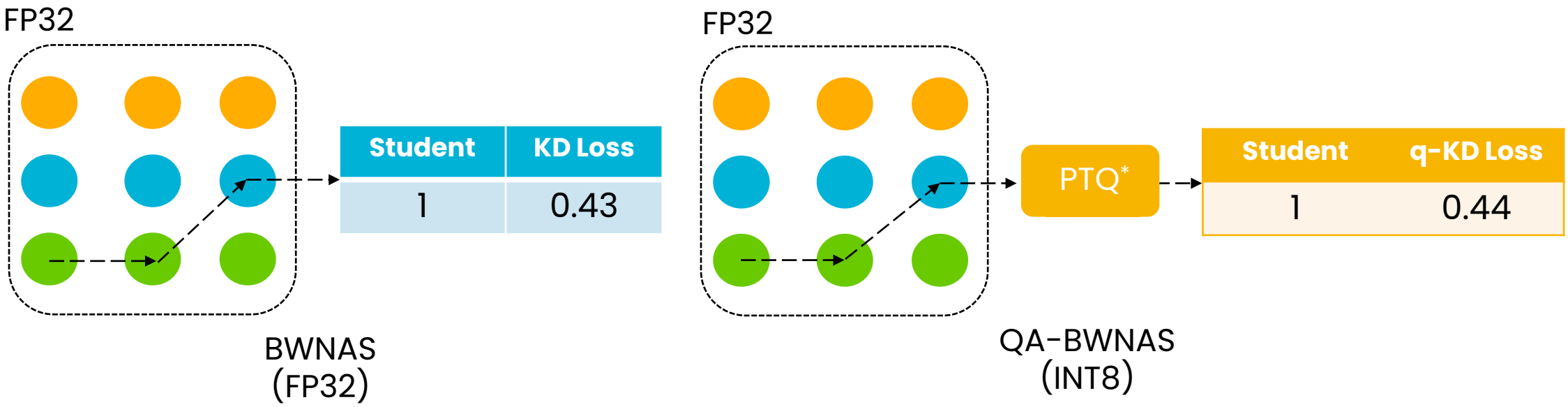
Key research question:

- **When and how to introduce quantization into block-wise NAS ?**

Quantization-Aware BWNAS: We quantize the block subnets and populate a quantized LUT to search for optimal quantized models

Idea: Quantize each subnet and populate a LUT with **q**uantized **KD** losses (q-KD) to model quantized performance and search for optimal quantized models

- **Hypothesis:** By introducing quantization during the search, we expect **Q**uantization-**A**ware **BWNAS** (QA-BWNAS) to be able to identify which models are less sensitive to quantization, thus, resulting in superior quantized performance



Afterwards, we search and train the best quantized model
Finally, we quantize the fully trained model via PTQ

Q: Can we do better ?

*: Post-Training Quantization

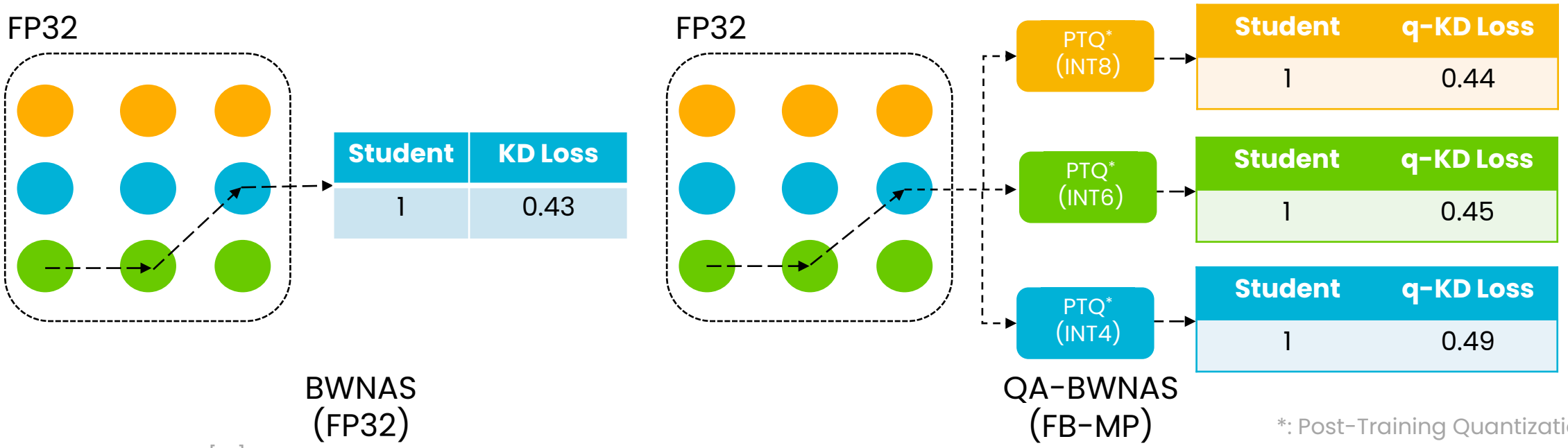
[12] Y. Lu, et al., "Scaling Up Quantization-Aware Neural Architecture Search for Efficient Deep Learning on the Edge," CODAI '23

We extend QA-BWNAS to search for few-bit mixed-precision networks

In order to achieve further compression, we introduce lower bit-widths into the space to search for highly compressed **F**ew-**B**it **M**ixed-**P**recision (FB-MP) networks.

- **Hypothesis:** By searching for both architectures and quantization policies, we expect QA-BWNAS to be able to discover mixed-precision architectures which can retain task performance, hopefully outperforming the results obtained by searching for homogeneous INT8 models.

Our previous approach can be used to easily introduce lower bit-widths (e.g., 4, 6) into the search space.



*: Post-Training Quantization

[12] Y. Lu, et al., "Scaling Up Quantization-Aware Neural Architecture Search for Efficient Deep Learning on the Edge," CODAI '23

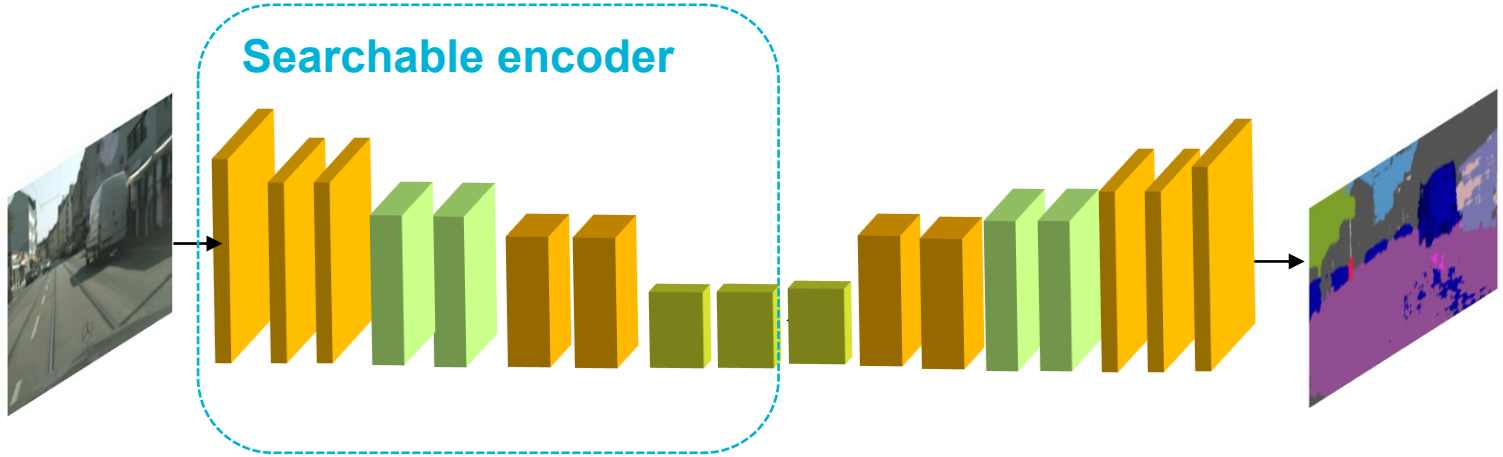
We investigate the effectiveness of QA-BWNAS to search for optimal quantized semantic segmentation networks

Task: Semantic Segmentation

Dataset: Cityscapes

Teacher model: DeepLabv3

- MobileNetV2 encoder.



Search Space:

- Kernel sizes $\in \{3, 5, 7\}$
- Expansion rates $\in \{3, 6\}$
- **Bit-width (weights only)** $\epsilon \{4, 6, 8\}^{\#}$

Optimization objectives:

- High task performance (mIoU*)
- Small model size ("params * bitwidths")

#: Activations are quantized to 8 bits

*: mean Intersection over Union

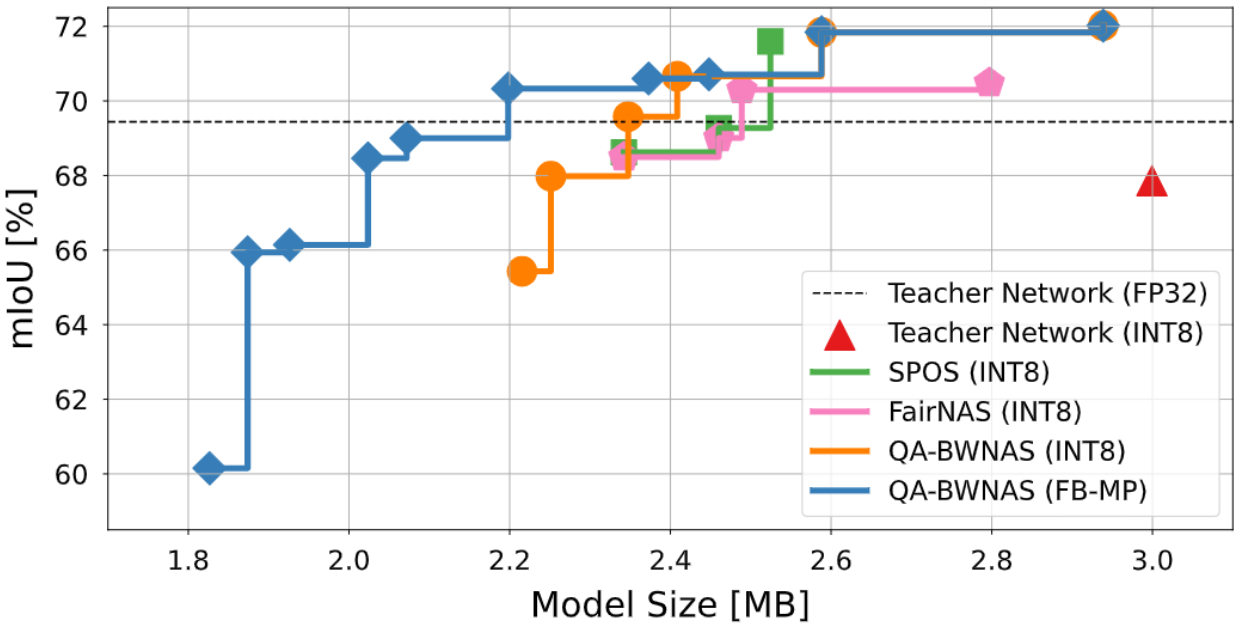
QA-BWNAS identifies quantized models that outperform the seed network, while reducing model size

QA-BWNAS (INT8) derives a Pareto front of solutions, substantially outperforming the teacher network

- Task performance improvement of up to 4.2 pp. compared to the quantized teacher network, while slightly reducing model size
- Model size reduction of up to 25%, with slightly higher accuracy as the INT8 quantized teacher network

QA-BWNAS (FB-MP) exhibits a pareto dominance over INT8 solutions, emphasizing the benefits over jointly searching for the optimal architecture and the quantization policy

- We reduce model size up to 33% while still outperforming the seed network (teacher model)




Scaling Up Quantization-Aware Neural Architecture Search for Efficient Deep Learning on the Edge

[12]

[12] Y. Lu, et al., "Scaling Up Quantization-Aware Neural Architecture Search for Efficient Deep Learning on the Edge," CODAI '23

Closing & Research Outlook

Take-home message: NAS does not magically solve everything out-of-the-box However, design NAS properly and it will always bring you benefits

Practical recommendations & takeaways

NAS does not magically solve everything out-of-the-box for you. Design it properly.

- First: ensure a proper setup for your baseline network
  - Optimize the training pipeline (data loading, preprocessing, training params)
  - Explore first what quantization setting works best for your baseline network. This can be used as a set configuration during NAS.
- Second: know what you need to optimize for.
  - Application requirements: latency, memory (model size or RAM usage?), …
  - System constraints: max memory, bitwidths, unsupported operations, …
  - Which metrics are relevant? How can I obtain or approximate those?
- Third: spend your resources wisely, know where to focus, keep it tractable
  - Profile the layer-wise metrics for the baseline network to identify bottlenecks.
  - Always consider Search space, Search Strategy & Performance Estimation jointly. Which part can be simple, which part needs to be designed smartly?

NN optimization today and future work

Neural Network optimization is a huge field with many challenges and opportunities

- HW vendors already offering solutions based on this technologies
  - NXP: eIQ Toolkit and eIQ Auto
  - STMicroelectronics: STM32Cube.AI
  - Qualcomm: AIMET

Many **Software-as-a-Service (SaaS) companies** providing optimization services based on these technologies have started to appear

- Deci (acquired by NVIDIA)
- Imagimob (acquired by Infineon)
- OmniML (MIT spin-off acquired by NVIDIA)
- Eta Compute (NXP partner)

Future work (…a lot to do for new researchers!):

- Optimizations for transformers and LLMs
- Agentic AI (AI making decisions and taking actions)
- Data availability: simulations, labeling, domain gaps, …
- EU AI Act compliance: Explainable AI, tracable designs, …

HOME › NEWS › EDGE & IOT

Nvidia acquires OmniML for AI Edge workloads

Quietly picked up the startup early this year

July 03, 2023 By: Sebastian Moss Have your say

EETimes

HOME NEWS PERSPECTIVES DESIGNLINES PODCASTS EDUCATION STORE

DESIGNLINES | AI & BIG DATA DESIGNLINE

Eta Compute Signs Up NXP for TinyML Platform

NXP Agrees to Acquire Edge AI Pioneer Kinara to Redefine the Intelligent Edge

February 10, 2025 8:00 AM EDT (UTC-4) by NXP Semiconductors Press Release

Transformers are in rising demand, but optimizing them for efficient deployment at the edge is challenging

Transformer-based architectures are on an exponential rise in industry and academia, powering many AI use cases:

- Speech recognition, Vision, Anomaly detection, etc.

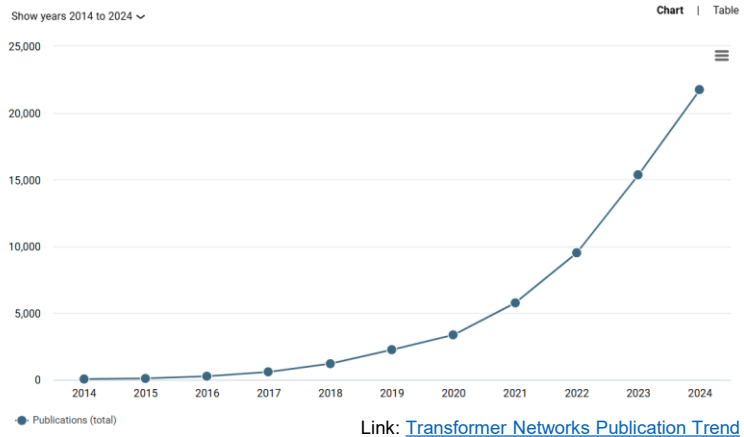
Transformers also starting to appear for edge AI applications:

- Conformer, Efficient Transfomer, etc.

Efficiently deploying transformers at the edge presents unique challenges[16]**:**

- Quantization: High dynamic activation ranges
  - Difficult to represent with a low bit fixed-point format
- SOTA investigates transformer-specific optimization: e.g., quantization, pruning, etc. [17, 18]

Number of publications per year on 'Transformer Networks"



Link: Transformer Networks Publication Trend

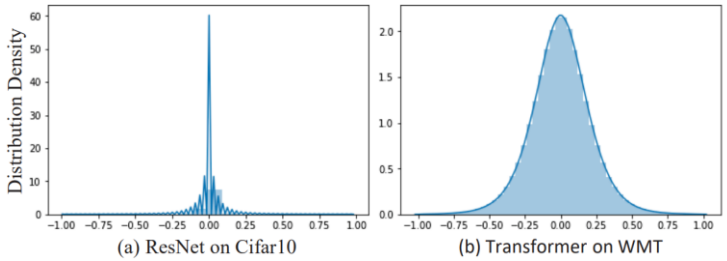


Fig. 2. Weight distribution of (a) ResNet and (b) Transformer

[16] Y. Bondarenko, et al., "Understanding and Overcoming the Challenges of Efficient Transformer Quantization," EMNLP '21

[17] G. Xiao, et al., "SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models," ICML '23

[18] W. Kwon, et al., "A Fast Post-Training Pruning Framework for Transformers," NeurIPS '22

Example student project topics (internships & graduation projects)

Design and automated optimization of DNNs for radar-based ADAS

- Leveraging radar-domain specifics and Physics-enhanced ML to improve DNN reliability or efficiency

Efficient LLMs, Transformers and Multimodal AI Agents at the Edge

- How to optimize and deploy highly optimized edge-ready transformers and LLMs?
- How to introduce additional modalities (e.g., sensor-based) into existing multimodal models?
- How to optimize small multimodal models towards 'agency': accurate function calling for a specific application?

Neural Networks for sensor signal improvement: denoising, infilling, artefact removal

- For several data or sensor domains, each with their own challenges and requirements, e.g. radar, audio, UWB, …

Domain Generalization via LLMs and unlabeled data

- How to use unlabeled data to improve the performance of models?
- Can we use the generalization capabilities of Foundation Models (e.g. LLMs) to improve the performance of a given model?

Automatic joint design and collaborative optimization of neural networks

- How to best combine two or more optimization techniques to maximize performance?

Autonomous tinyML systems – MCUs with novel NPUs

- How to enable efficient on-device learning?
- How to map tiny transformer models and attention operations?

Feel free to reach out!

- willem.sanberg@nxp.com
- selcuk.sandikci@nxp.com

(send us: CV, short motivation message, grade list, preferred start date & duration)

nxp.com