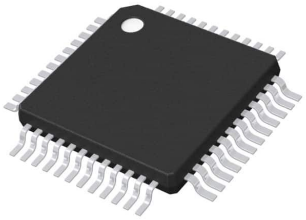
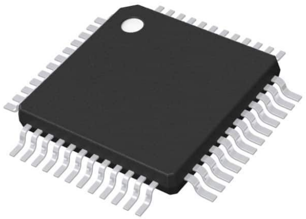


Algorithmic Optimizations

Jonathan Alvarez, Colby Banbury, Alicia Golden, Edgar Guzman

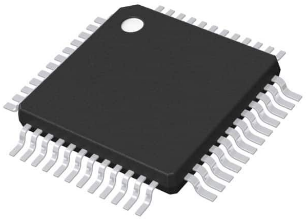
Microcontroller Image Source (All Slides): https://www.digikey.com/en/products/detail/stmicroelectronics/STM32L010C6T6/9770445?utm_adgroup=Integrated%20Circuits%20%28ics%29&utm_source=google&utm_medium=cpc&utm_campaign=Shopping_Supplier_STMicroelectronics&utm_term=&utm_content=Integrated%20Circuits%20%28ics%29&gclid=CjwKCAjwg5uZBhATEiwAhhRLHgzUltVvHUVxB2-_-ehqoi8xUiBPQ_6OZ2JkqSxNdAp_lVJiNtzAkxoCuwEQAvD_BwE

Algorithmic Optimizations



- Background of existing optimizations for TinyML
- Contributions of MCUNet
- MicroNets
- Future work / directions in the field

Algorithmic Optimizations



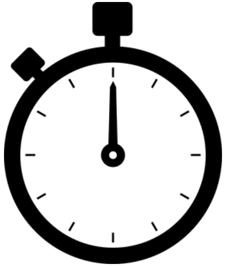
- **Background of existing optimizations for TinyML**
- Contributions of MCUNet
- MicroNets
- Future work / directions in the field

TinyML Constraints

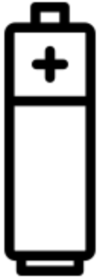
SRAM

Flash

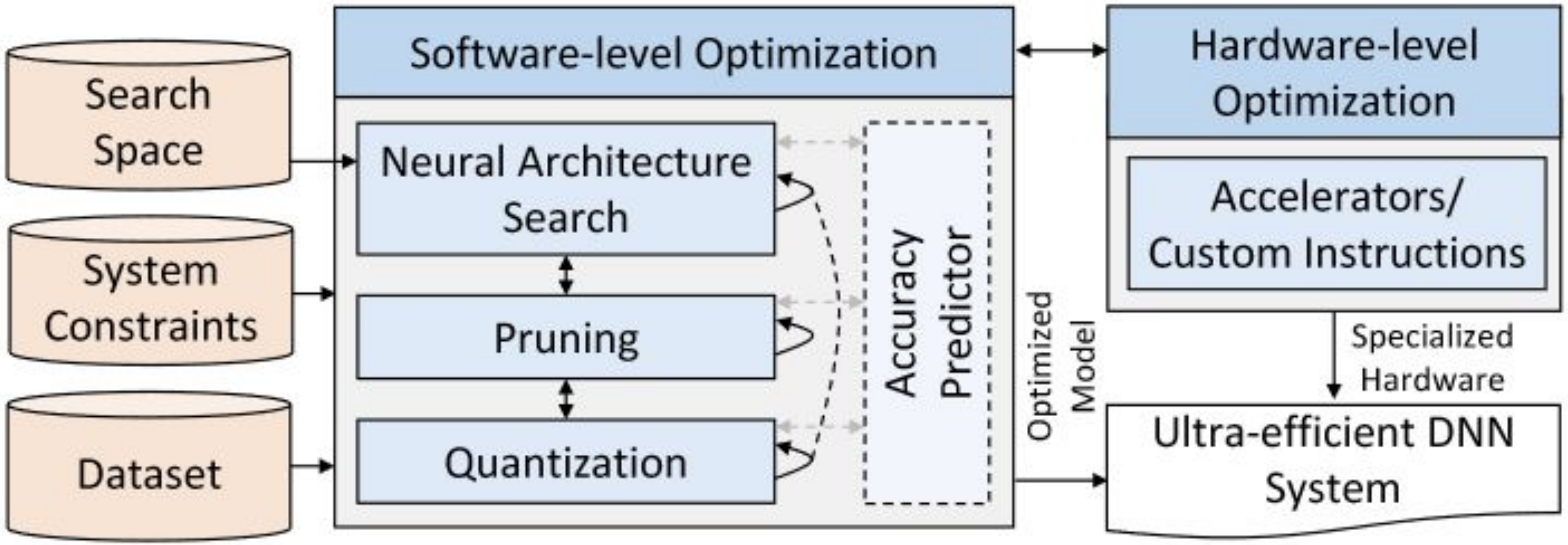
Latency



Energy

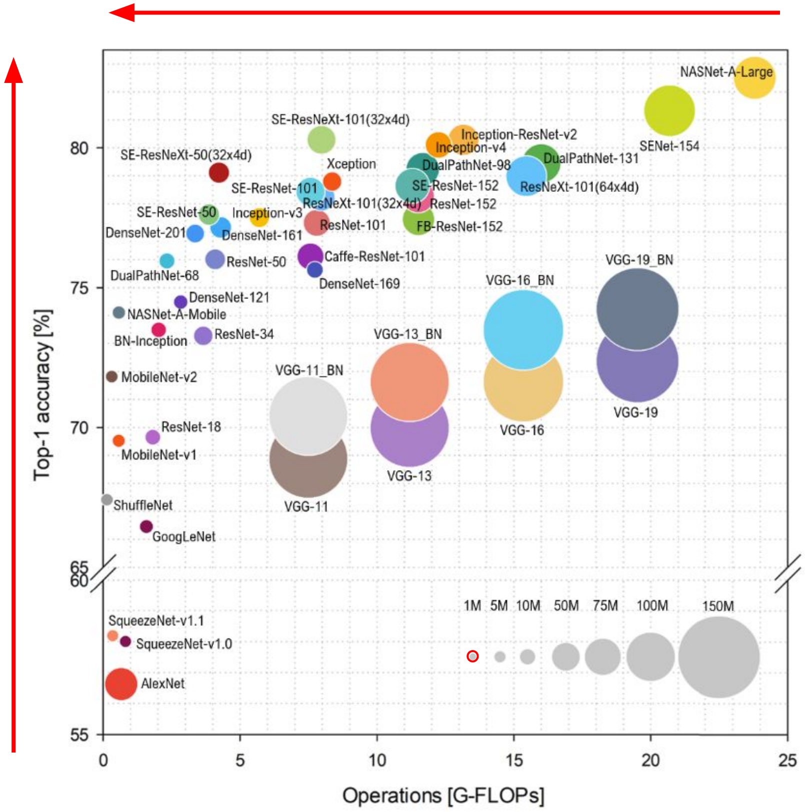


How can we tackle these constraints?



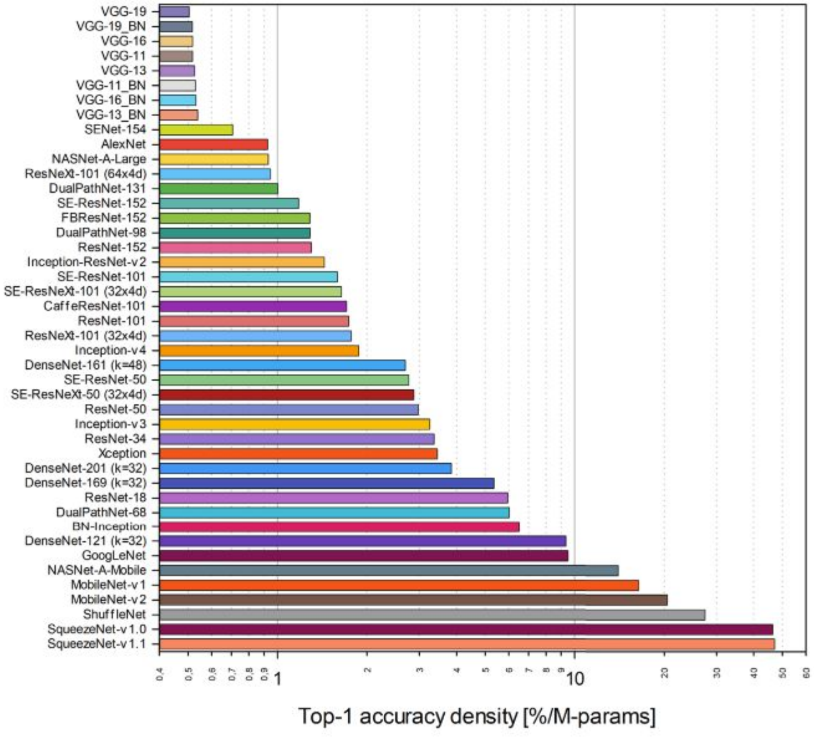
Shafique et al., “TinyML: Current Progress, Research Challenges, and Future Roadmap” (2017)

Why do software-level optimizations make sense?

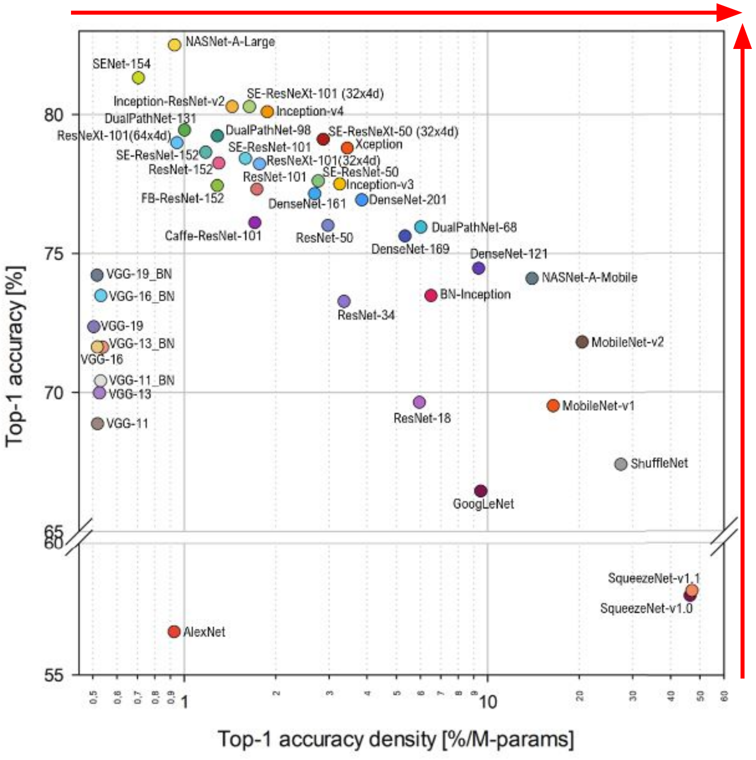


Bianco et al., Benchmark Analysis of Representative Deep Neural Network Architectures (2018)

Why do software-level optimizations make sense?



(a)

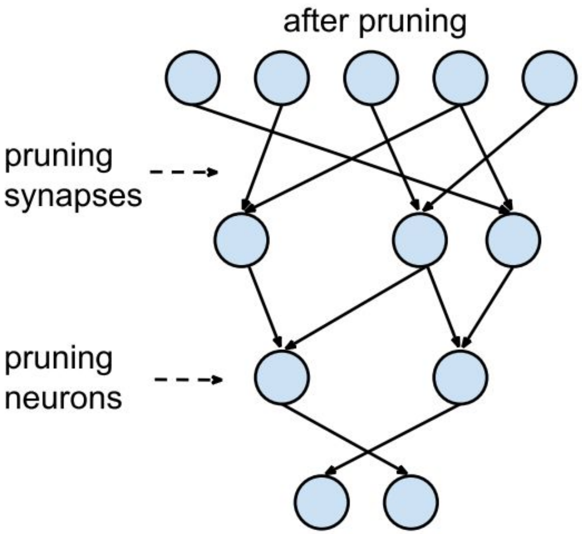


(b)

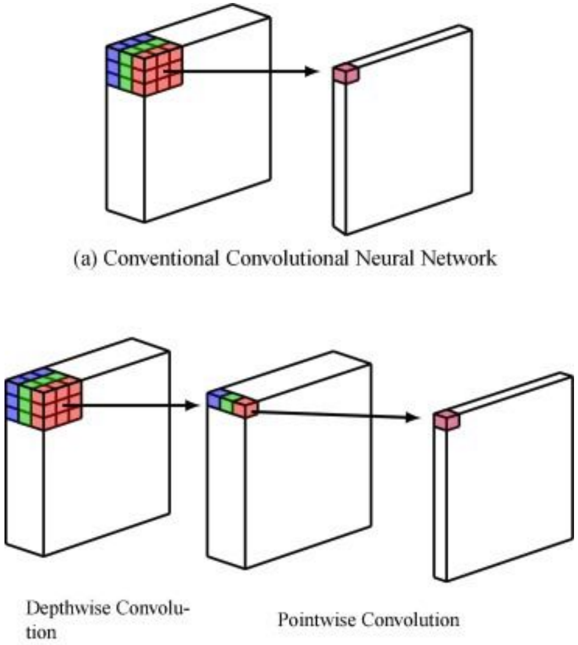
Bianco et al., Benchmark Analysis of Representative Deep Neural Network Architectures (2018)

Algorithmic Optimization

Model Compression



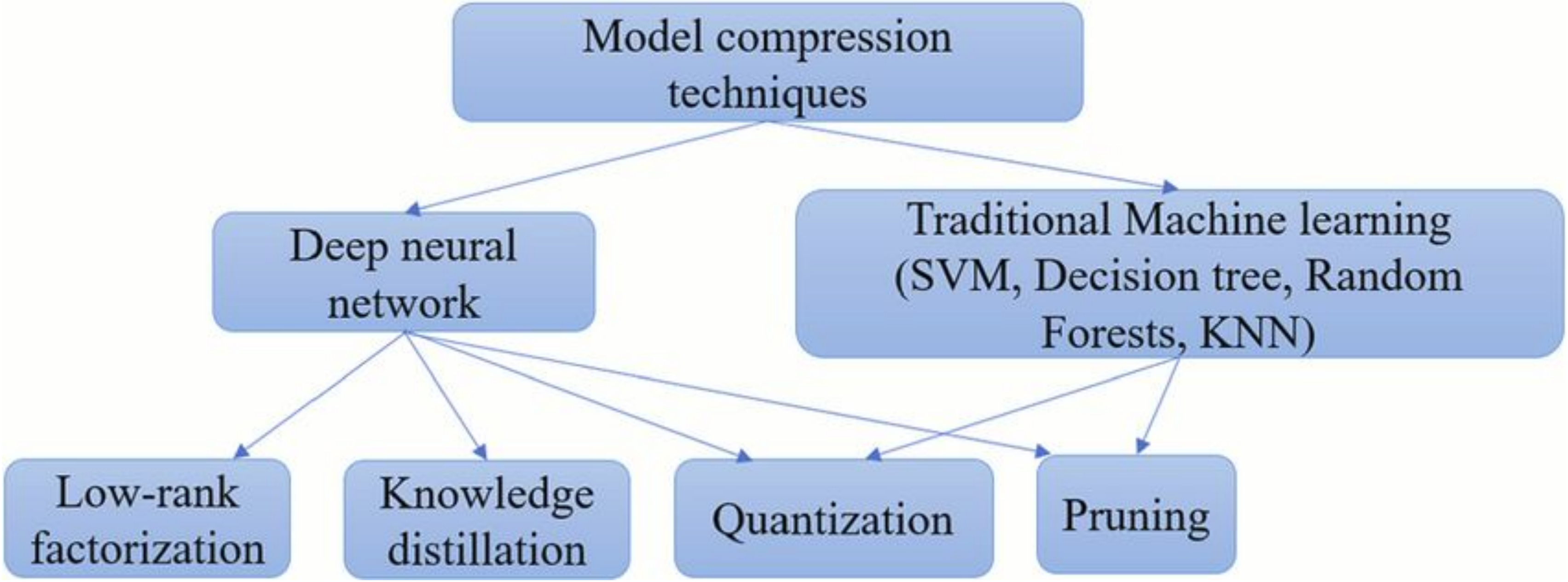
Model or Block Design



(a) Conventional Convolutional Neural Network

(b) Depthwise Separable Convolutional Neural Network

Model Compression



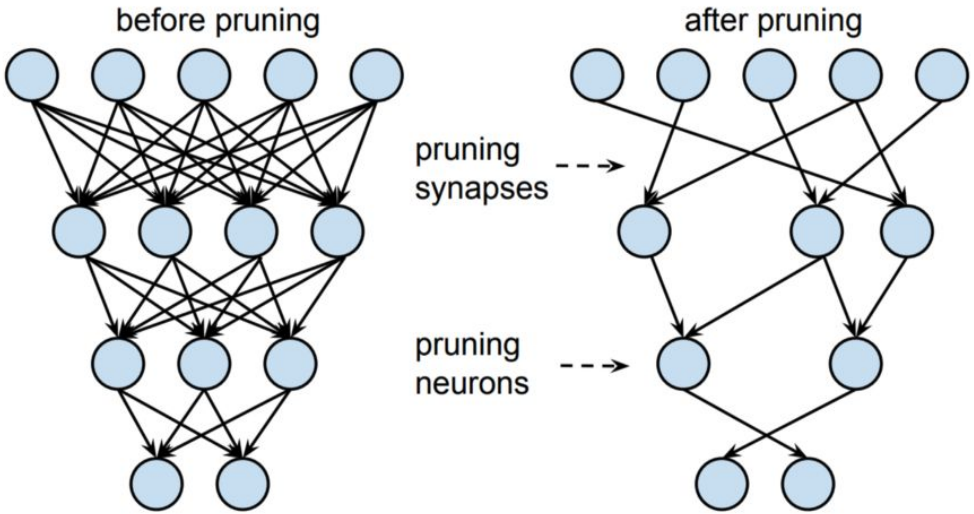
Choudhary et al. “A comprehensive survey on model compression and acceleration.” (2020)

Model Compression - Quantization

Takes an existing model and compresses its parameters by changing from floating-point numbers to low-bit width numbers, thus avoiding costly floating-point multiplications.

https://developer.nvidia.com/blog/achieving-fp32-accuracy-for-int8-inference-using-quantization-aware-training-with-tensorrt/

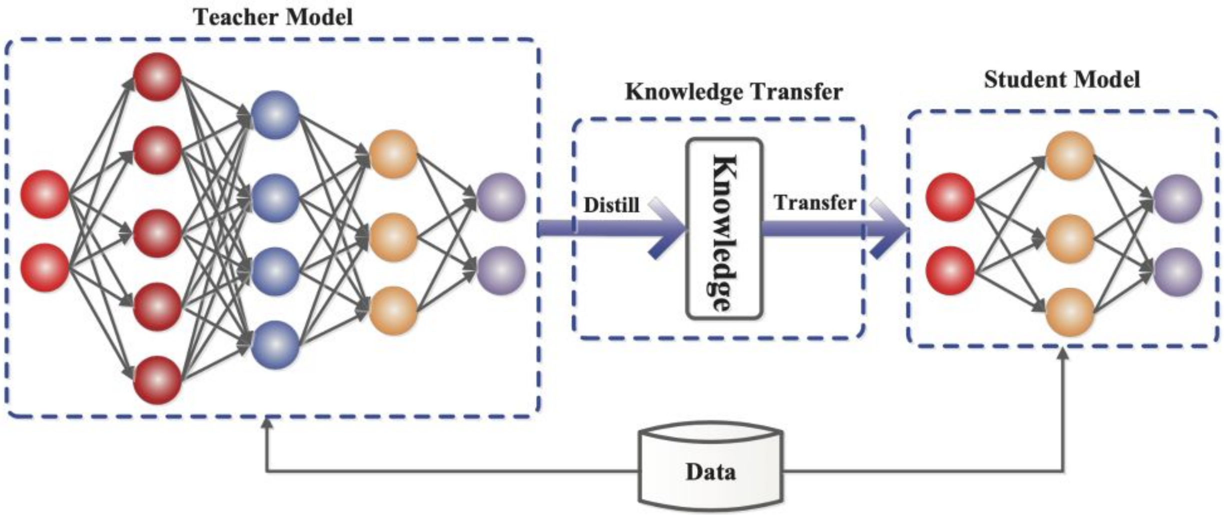
Model Compression - Pruning



Pruning involves removing the least important parameters (e.g., those that are close to 0).

Han et al., "**Learning both Weights and Connections for Efficient Neural Networks (2015)**

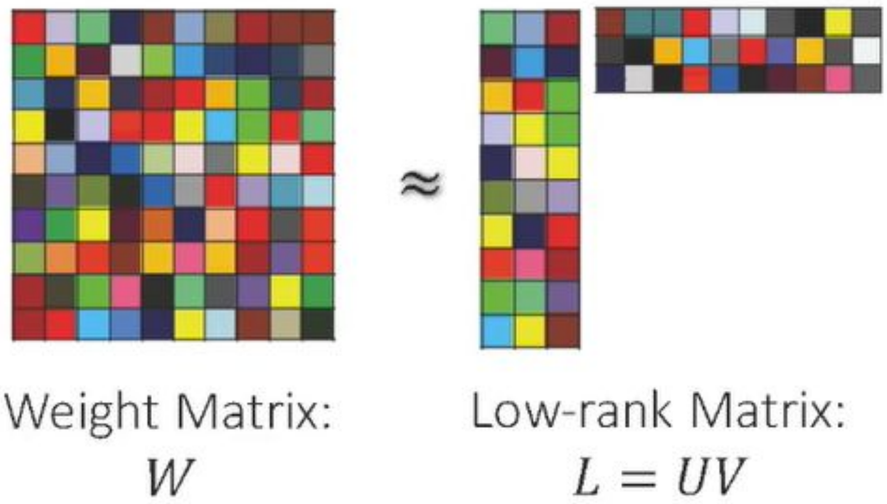
Model Compression - Knowledge Distillation



Knowledge distillation involves creating a smaller model that imitates the behavior of a larger, more powerful model. This is done by training the smaller model using the output predictions produced from the larger model.

Gou et al., “Knowledge Distillation: A Survey.” (2021)

Model Compression - Low-Rank Factorization

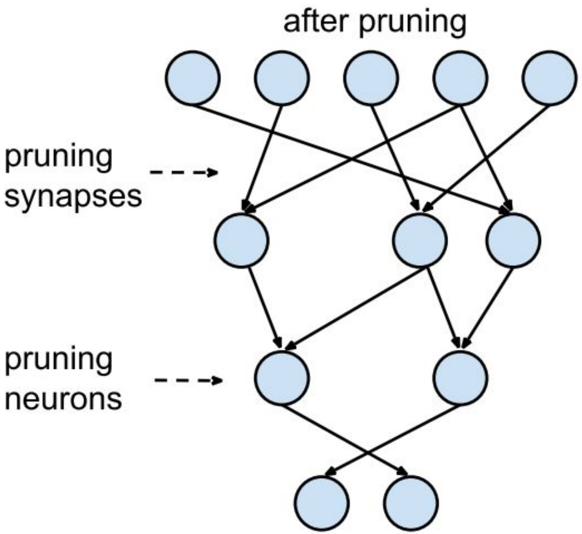


Low-rank factorization involves approximating the weight matrix by an approximating matrix subject to the constraint that the approximating matrix has reduced rank.

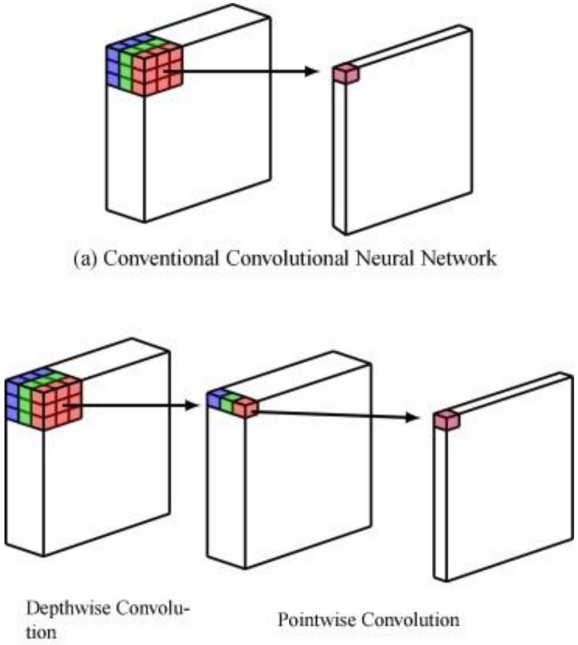
Yu et al., "**On Compressing Deep Models by Low Rank and Sparse Decomposition" (2017)**"

Algorithmic Optimization

Model Compression



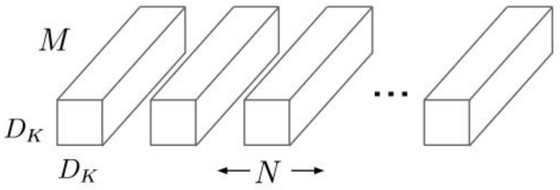
Model or Block Design



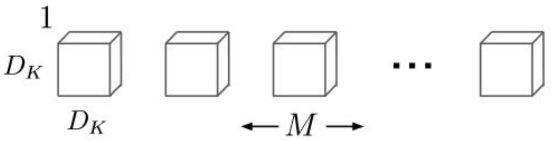
(a) Conventional Convolutional Neural Network

(b) Depthwise Separable Convolutional Neural Network

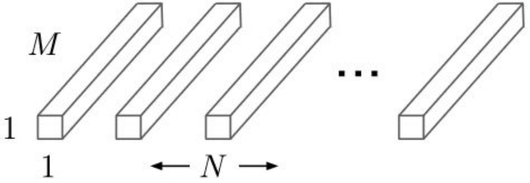
Model Design - MobileNet



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

D_F x D_F is feature map size

$$\frac{\boxed{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F} + \boxed{M \cdot N \cdot D_F \cdot D_F}}{\boxed{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F}}$$

$$= \frac{1}{N} + \frac{1}{D_K^2}$$

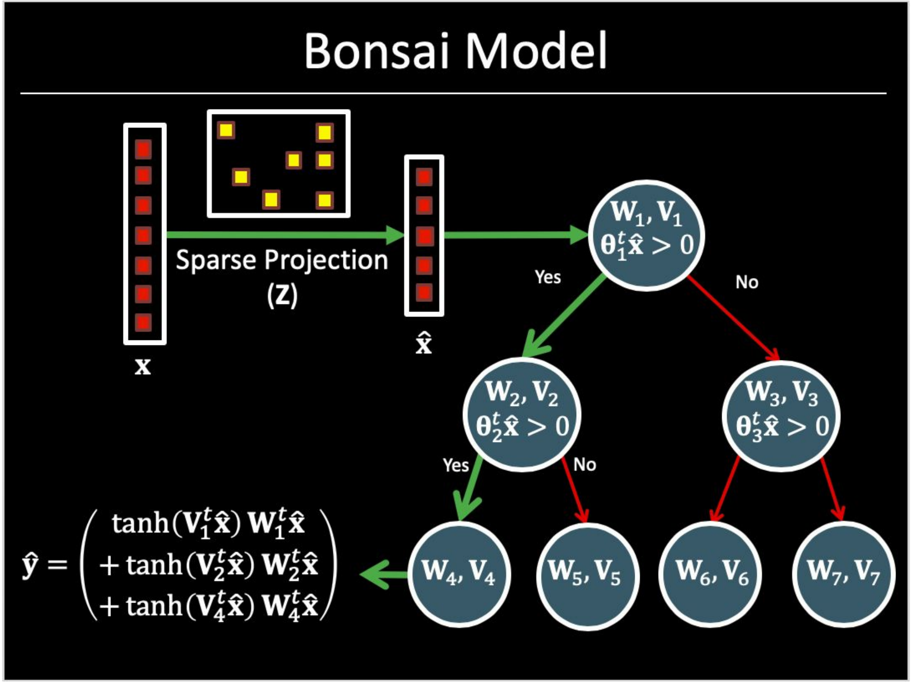
small reduction in accuracy with 8 to 9 times less computations*

Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Howard et al., "**MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications**" (2017)

Model Design - Bonsai



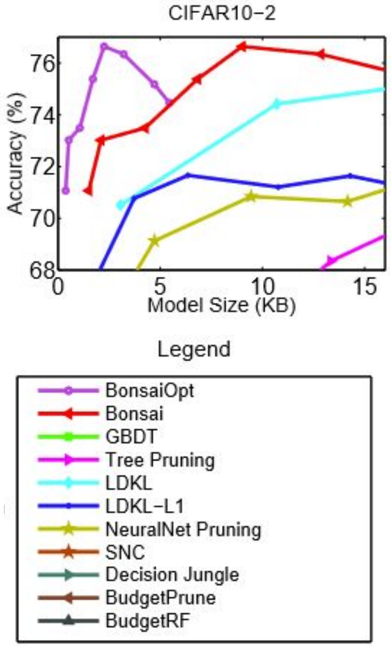
https://ashish-kmr.github.io/

(1) learns single, shallow, sparse tree

(2) non-linear predictions from nodes

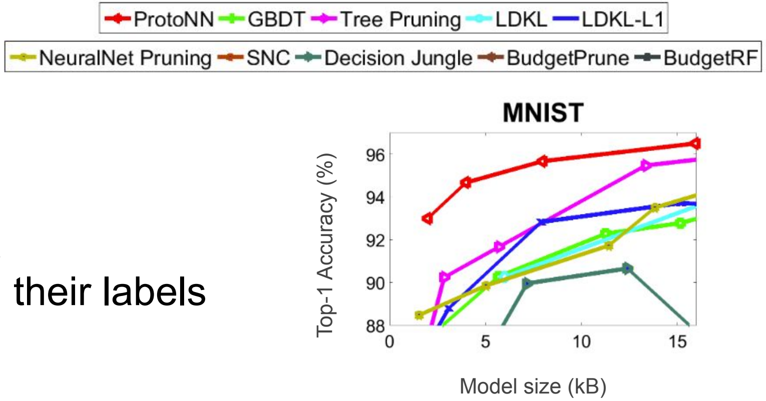
(3) projects data into low-D space

(4) nodes and projection learnt jointly



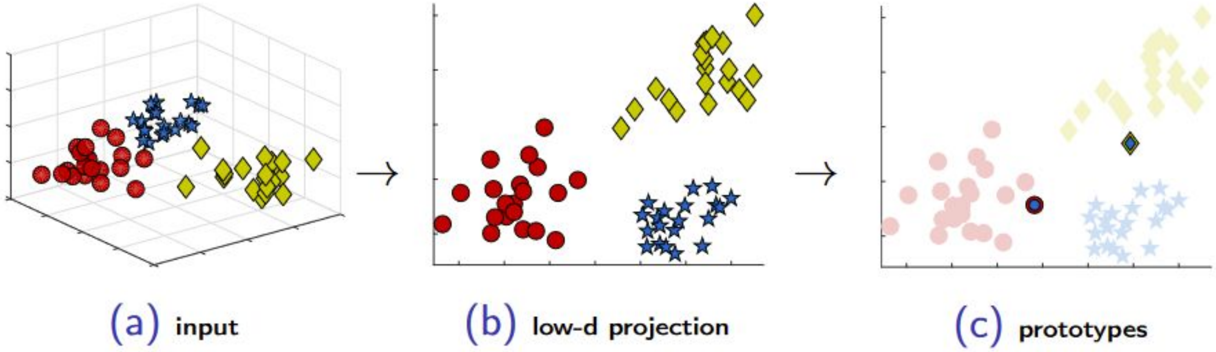
Kumar et al., "Resource-efficient Machine Learning in 2 KB RAM for the Internet of Things" (2016)

Model Design - ProtoNN



ProtoNN *jointly* optimizes:

- A sparse low-dimensional projection of the input data
- A set of *prototypes* in the low dimensional space with their labels

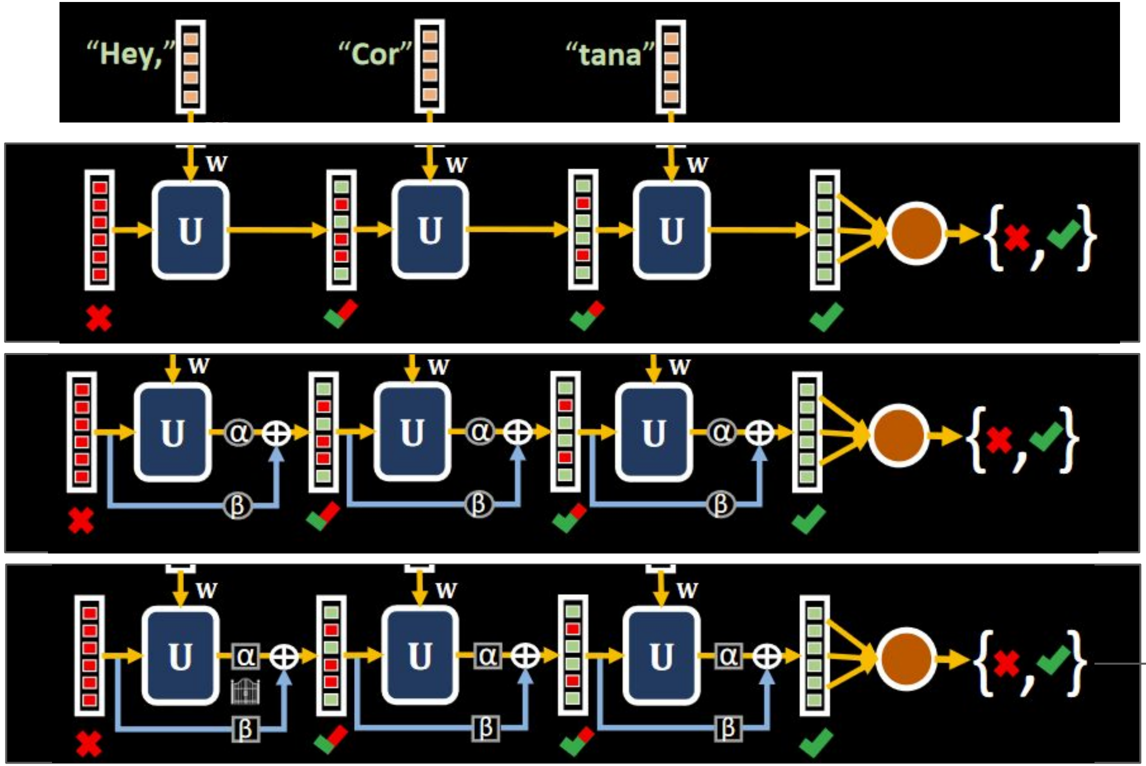


reducing model size, lowering prediction time, and improving accuracy

Gupta et al., "**ProtoNN: Compressed and Accurate kNN for Resource-scarce Devices**" (2017)

Model Design - FastGRNN

Compared to GRU / LSTM
~20-80x smaller
~Within 1% accuracy
~25-132x faster prediction on Arduino MKR1K



RNN

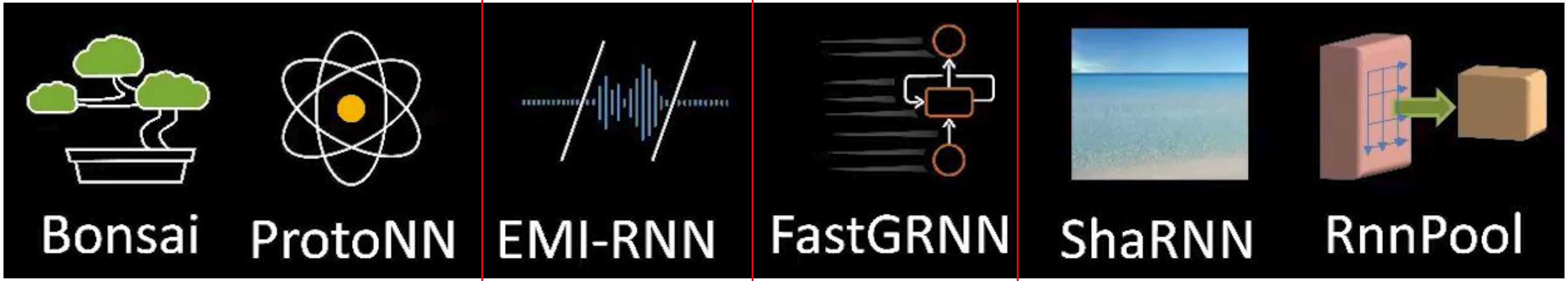
Fast RNN
Added residual connection.

Fast GRNN
Converted residual connection to gate.

For both Fast RNN architectures, the authors also imposed constraints for size (low-rank, spase, quantized).

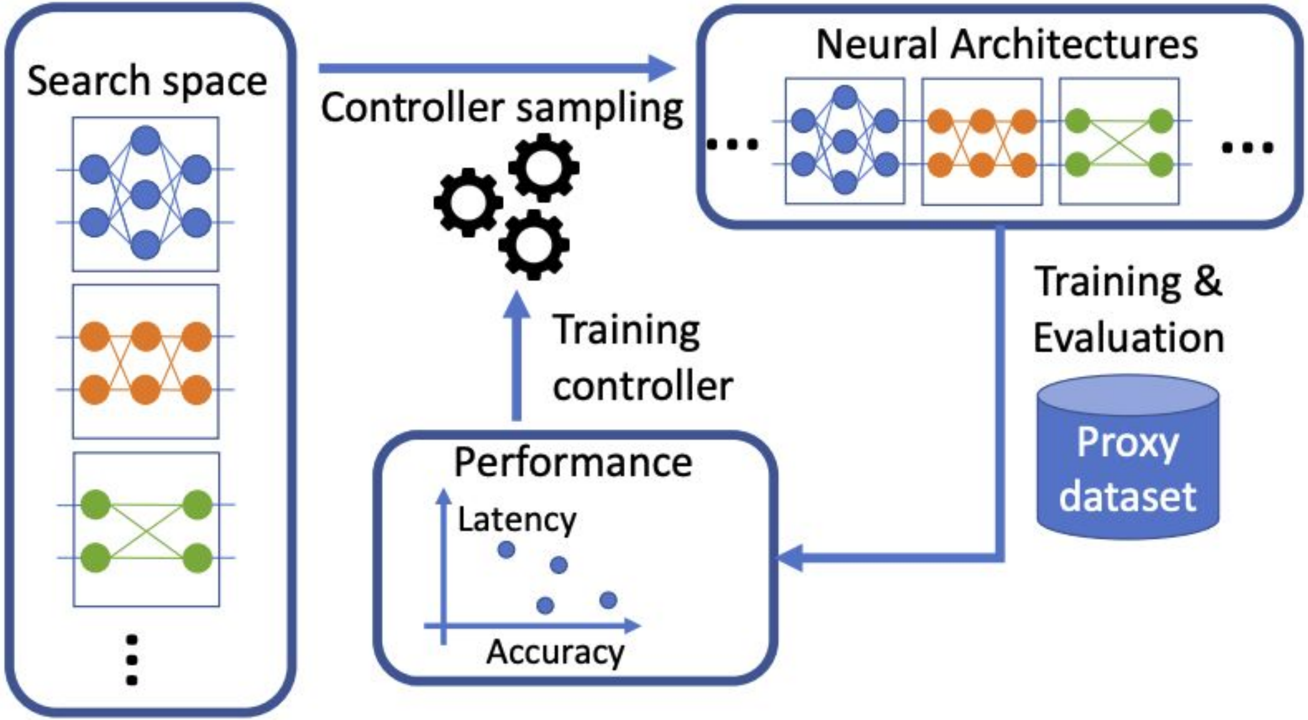
Kusupat et al., "FastGRNN: A Fast, Accurate, Stable and Tiny Kilobyte Sized Gated Recurrent Neural Network," (2019)

Model Design - Microsoft EdgeML



https://github.com/microsoft/EdgeML

Model Design - Neural Architecture Search

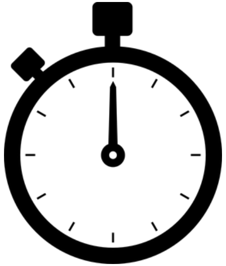


TinyML Constraints

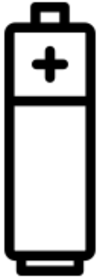
SRAM

Flash

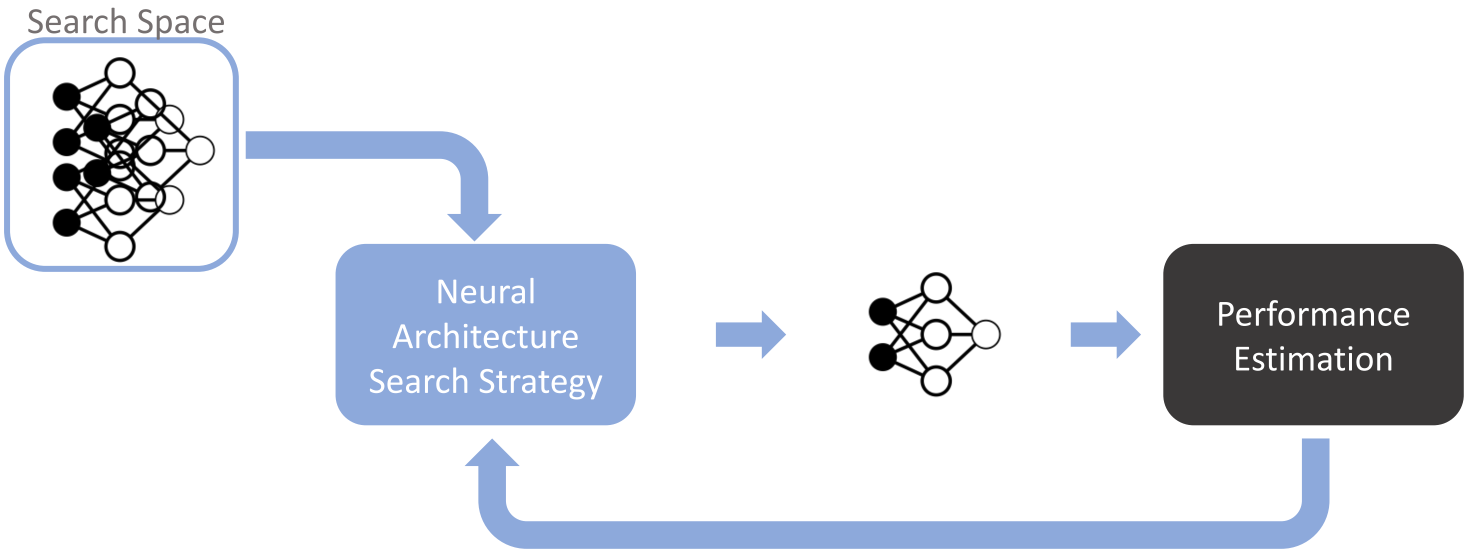
Latency



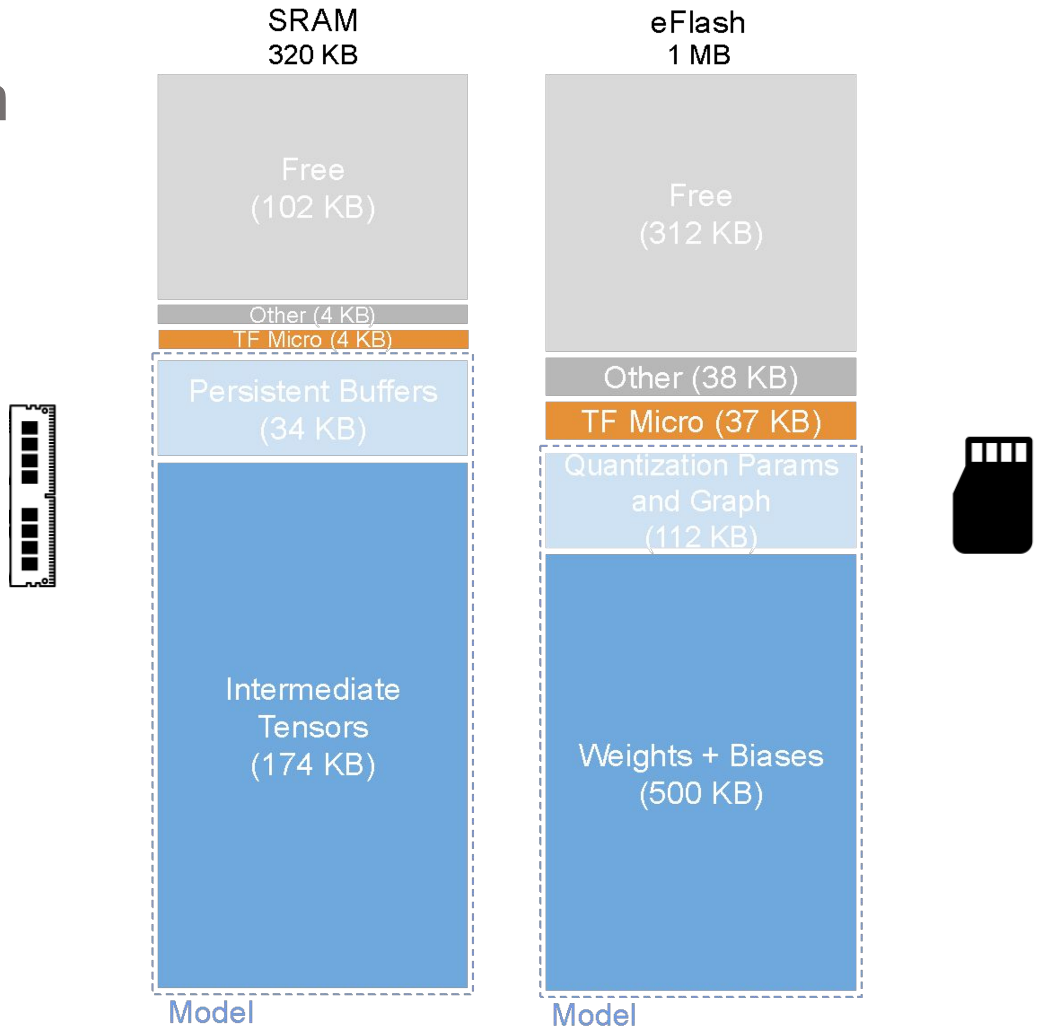
Energy



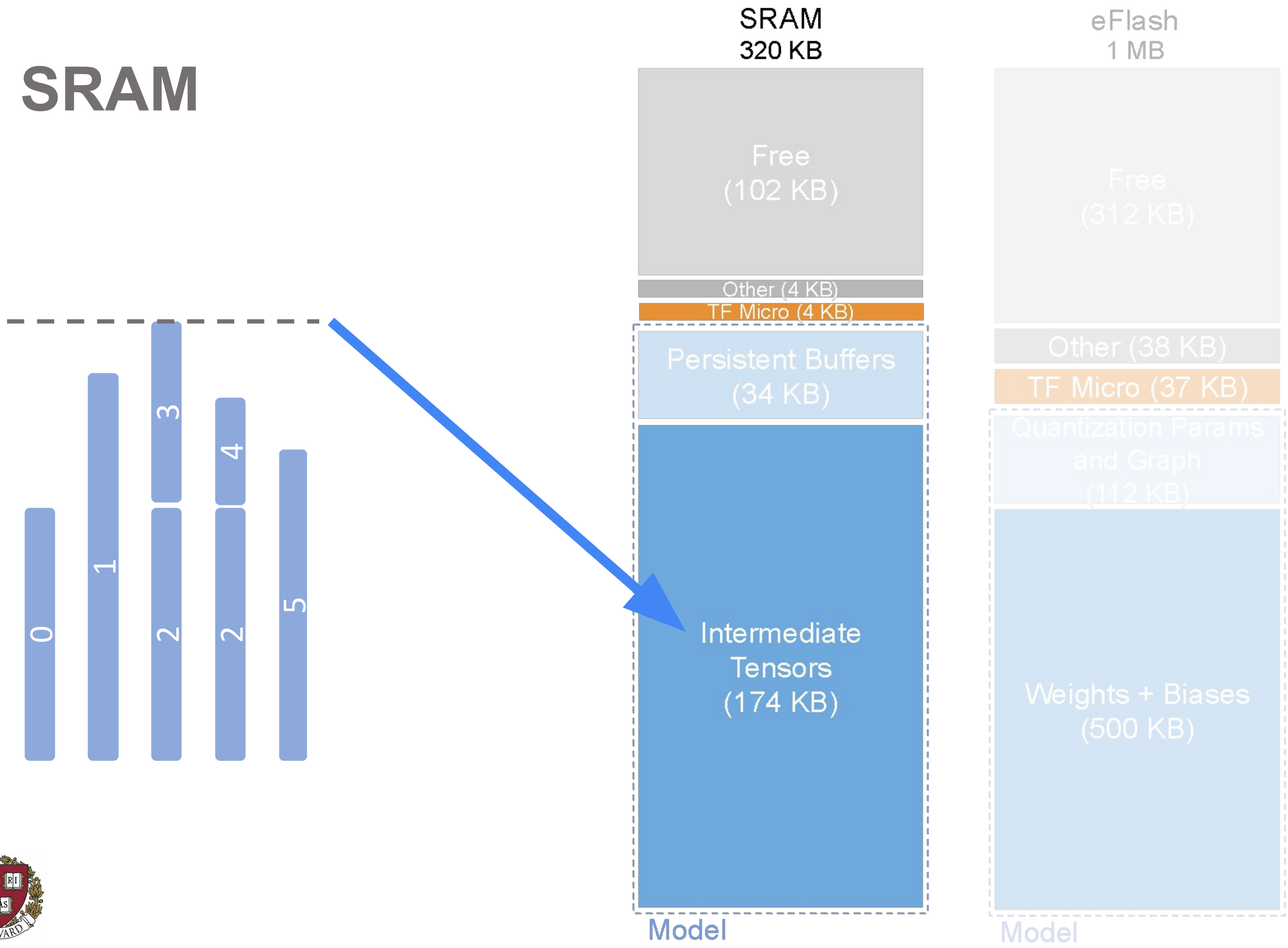
Neural Architecture Search (NAS)



SRAM and Flash



SRAM



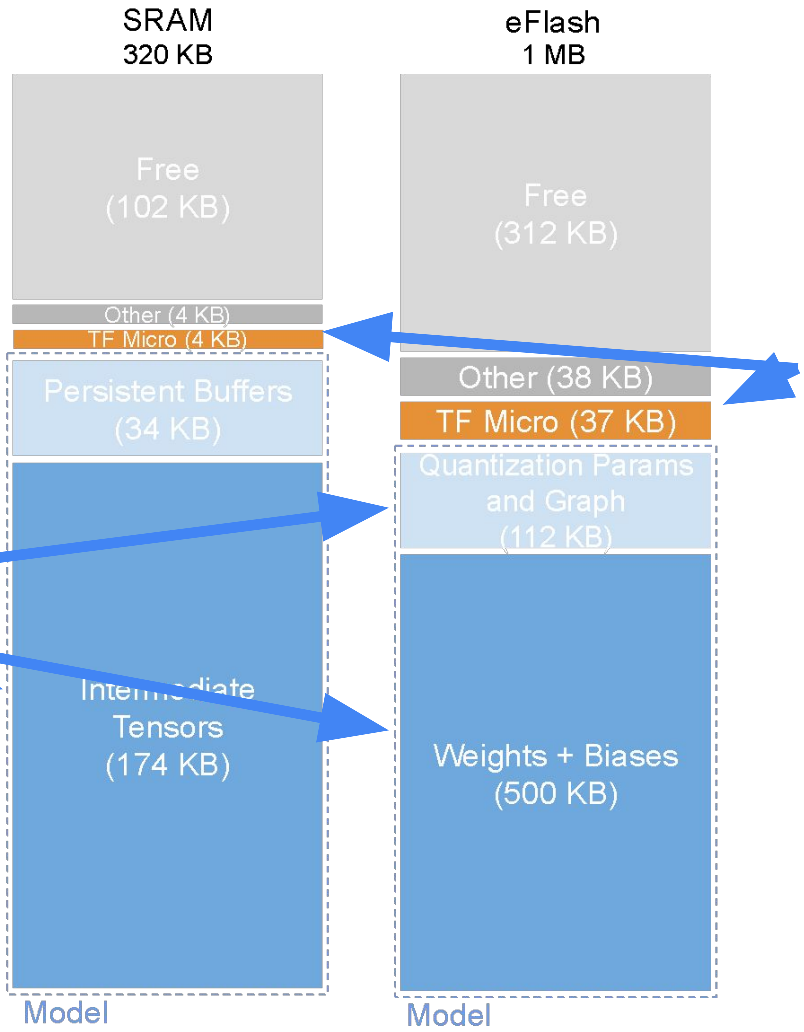
SRAM
320 KB
Free
(102 KB)
Other (4 KB)
TF Micro (4 KB)
Persistent Buffers
(34 KB)
Intermediate
Tensors
(174 KB)
Model
eFlash
1 MB
Free
(312 KB)
Other (38 KB)
TF Micro (37 KB)
Quantization Params
and Graph
(112 KB)
Weights + Biases
(500 KB)
Model
0
1
3
2
4
2
5

Flash


SRAM
320 KB
Free
(102 KB)
Other (4 KB)
TF Micro (4 KB)
Persistent Buffers
(34 KB)
Intermediate
Tensors
(174 KB)
Model
eFlash
1 MB
Free
(312 KB)
Other (38 KB)
TF Micro (37 KB)
Quantization Params
and Graph
(112 KB)
Weights + Biases
(500 KB)
Model

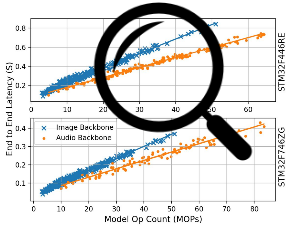
SRAM and Flash

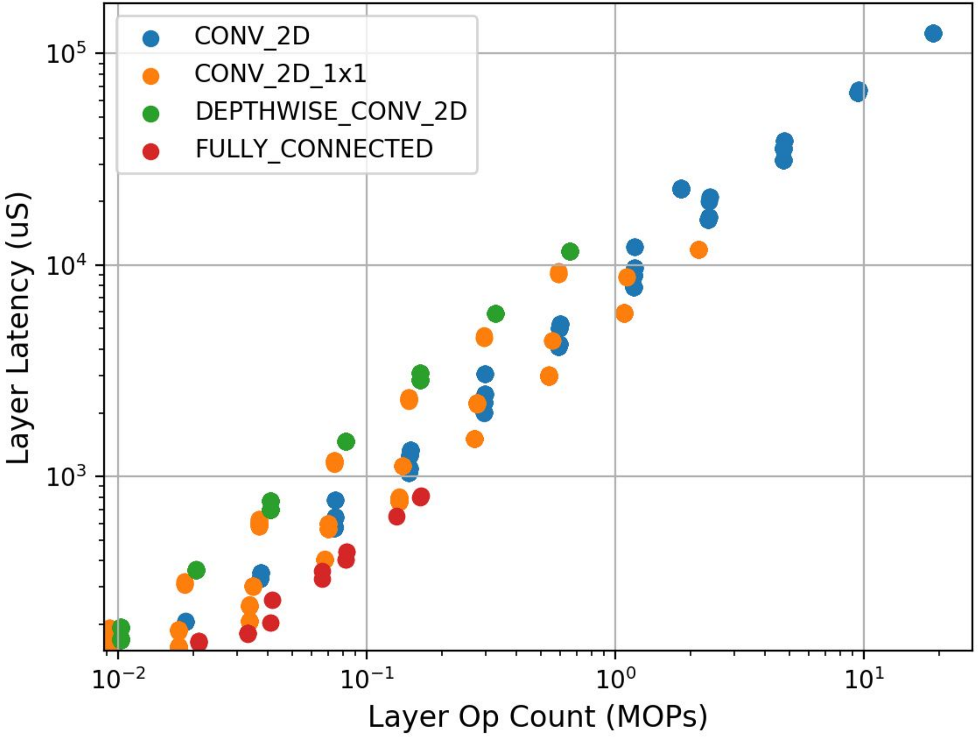
Determined by the Model Architecture



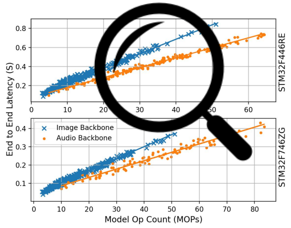
Overhead

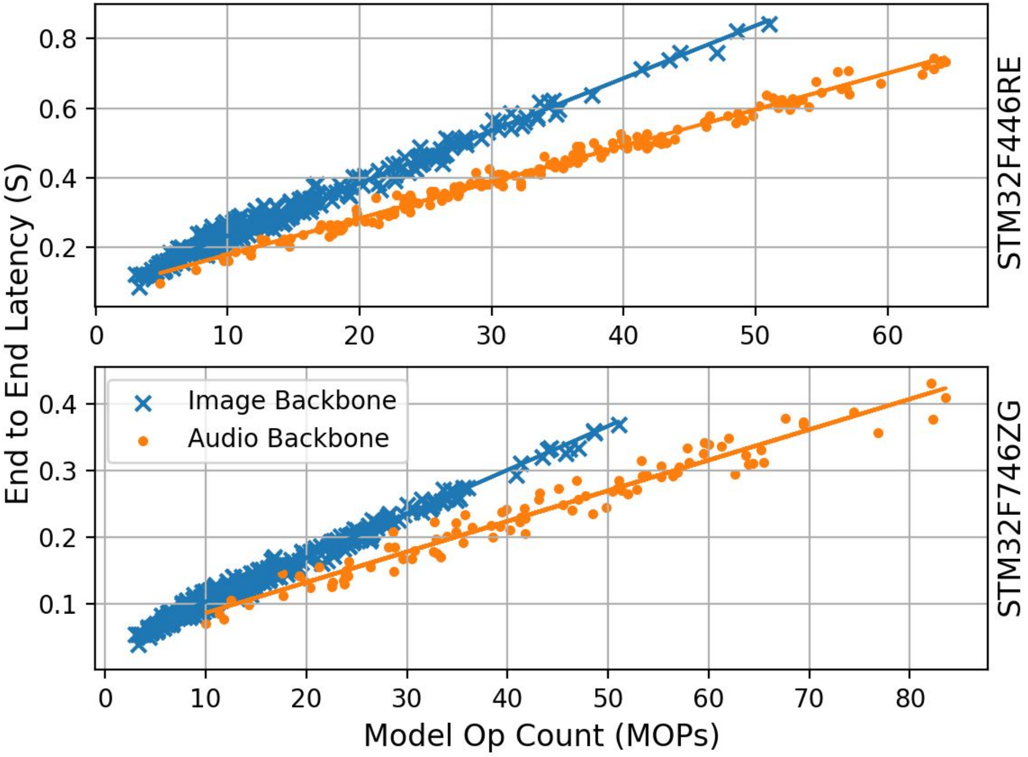
Per Layer Latency



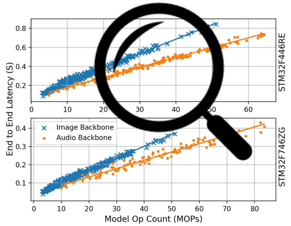


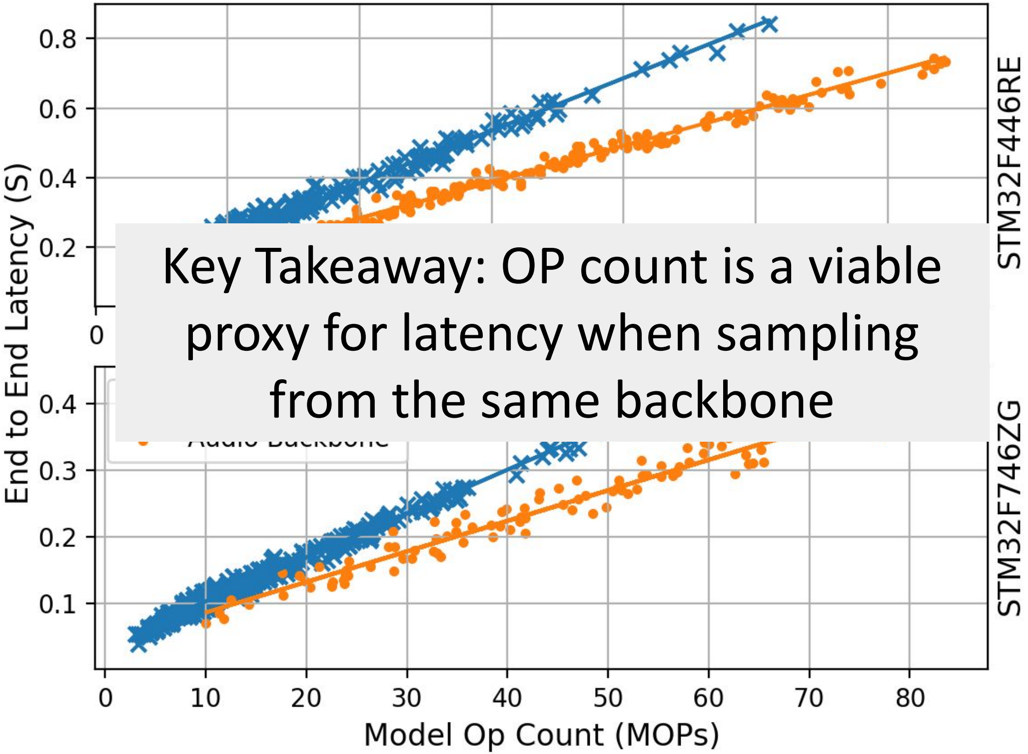
Model Latency





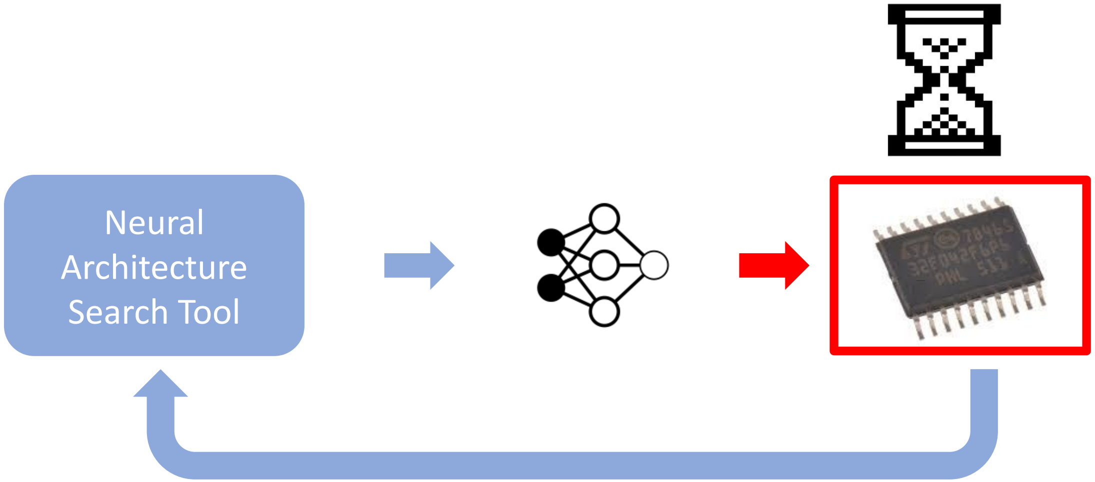
Model Latency



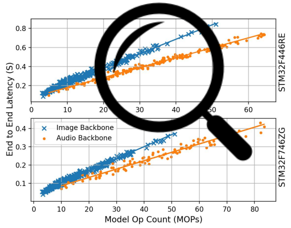


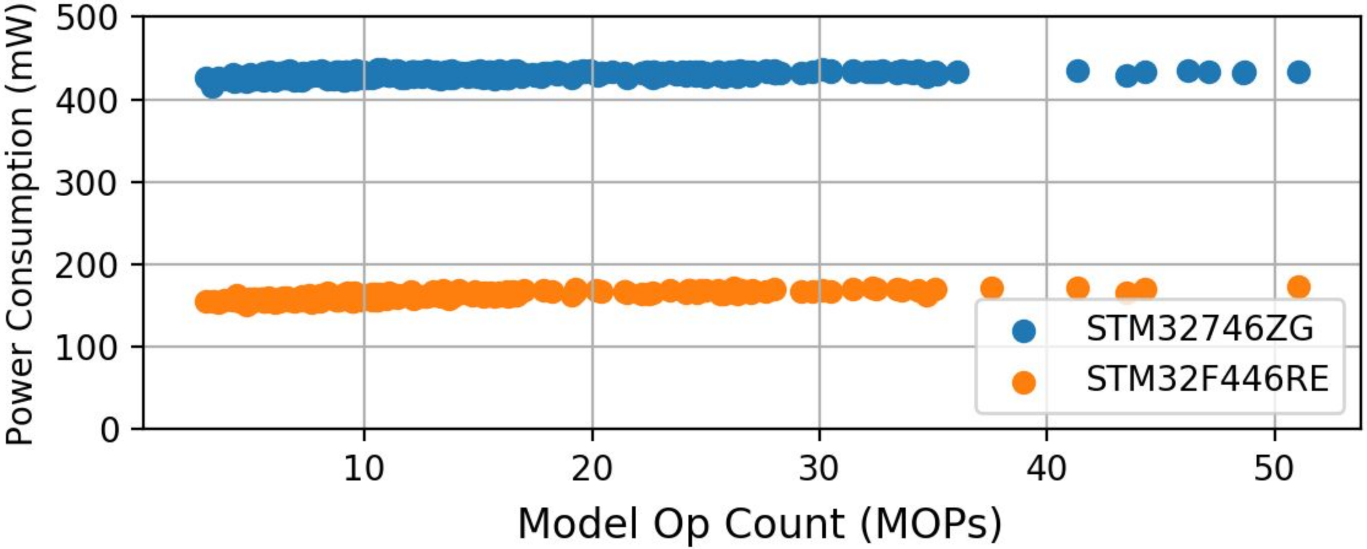
Key Takeaway: OP count is a viable proxy for latency when sampling from the same backbone

Direct Latency Benchmarking

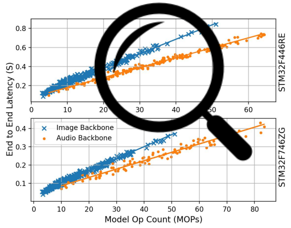


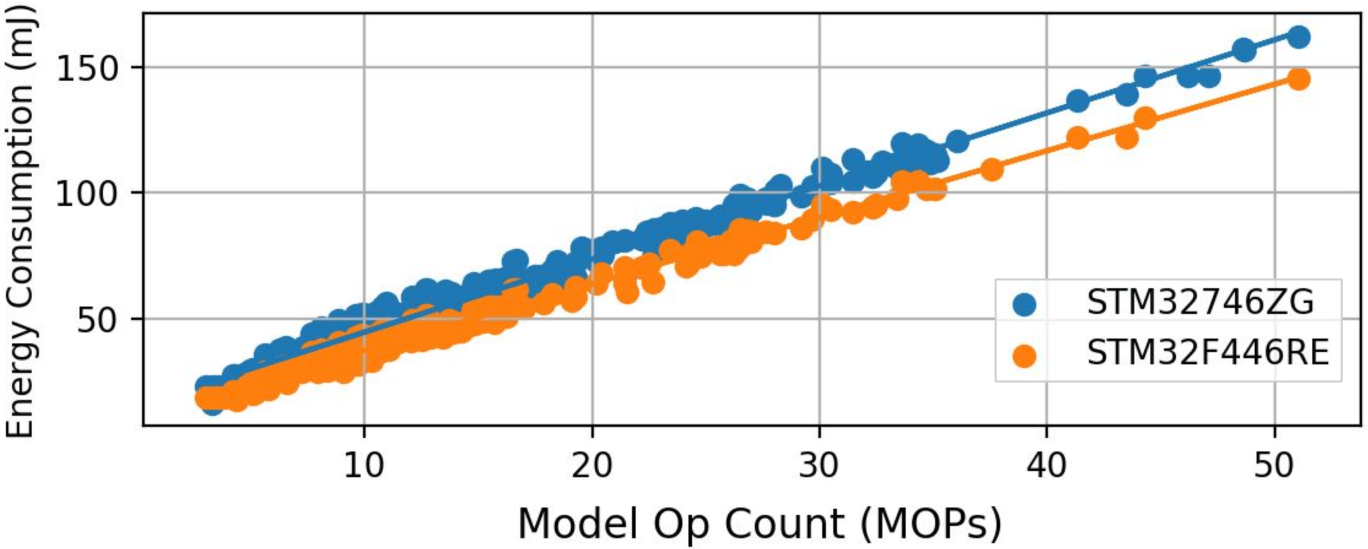
Model Energy



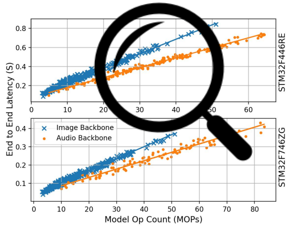


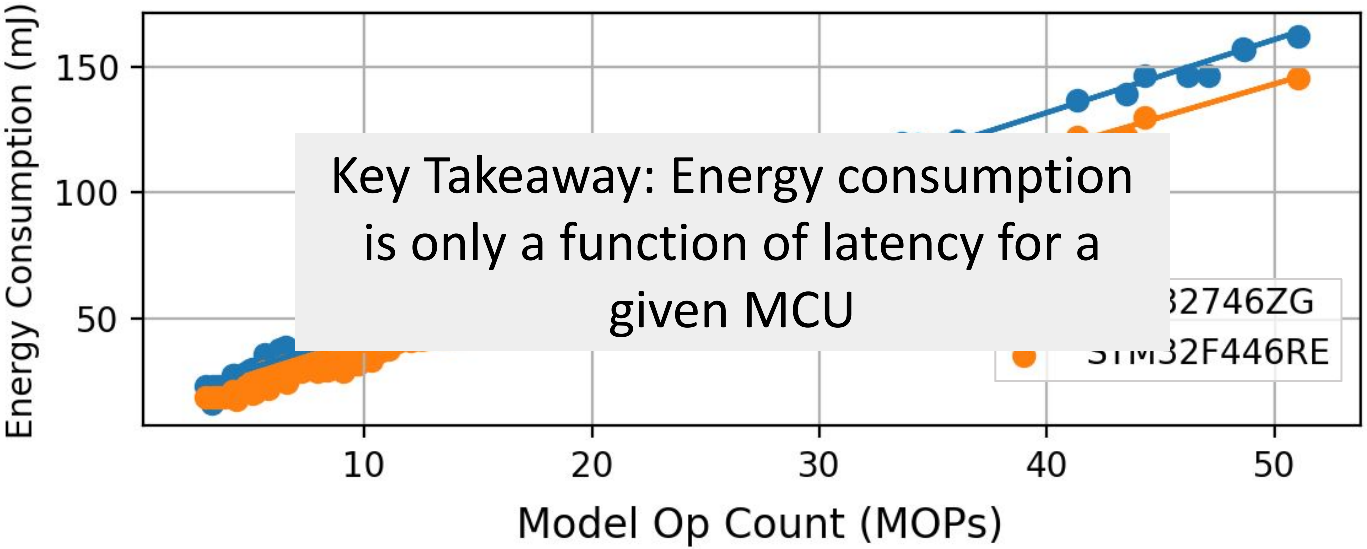
Model Energy





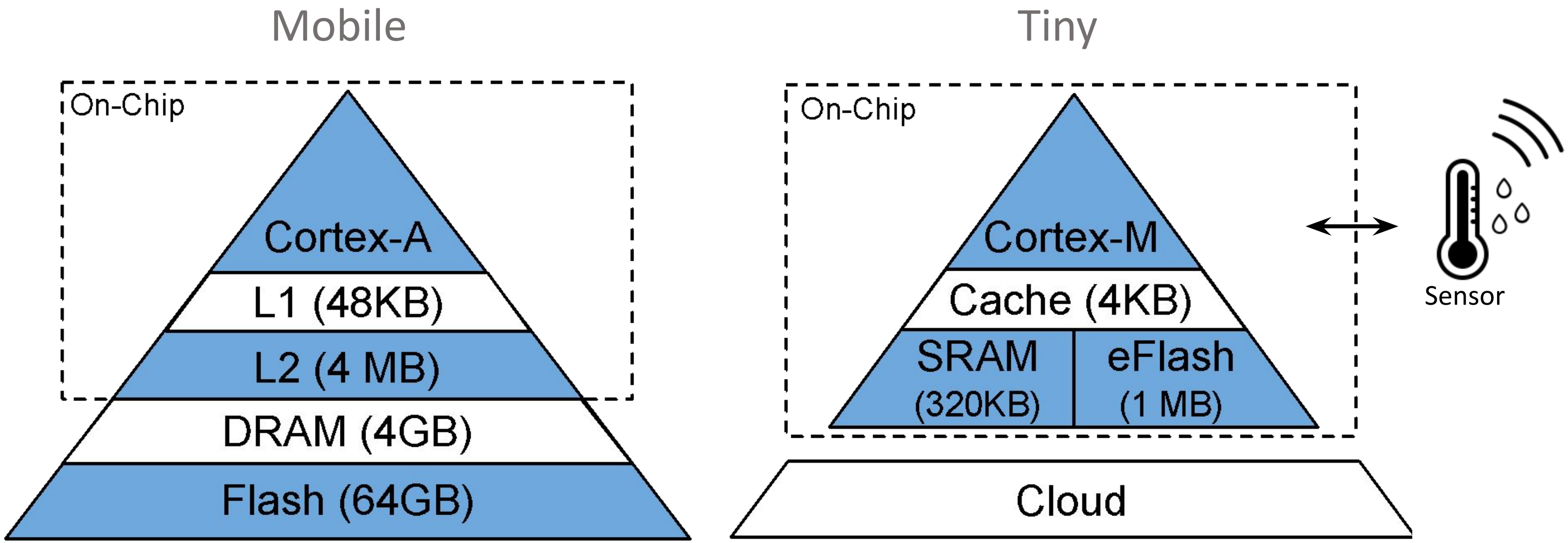
Model Energy



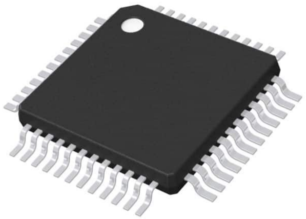


Key Takeaway: Energy consumption is only a function of latency for a given MCU

Memory Hierarchy

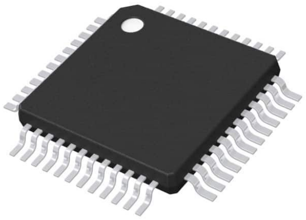


Algorithmic Optimizations



- Background of existing optimizations for TinyML
- **Contributions of MCUNet**
- MicroNets
- Future work / directions in the field

MCUNet: Tiny Deep Learning on IoT Devices



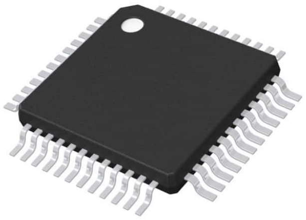
TinyNAS

1. Automated search space optimization
2. Resource-constrained model specification

TinyEngine

1. Reducing overhead with separated compilation and runtime
2. In-place depth-wise convolution

MCUNet: Tiny Deep Learning on IoT Devices



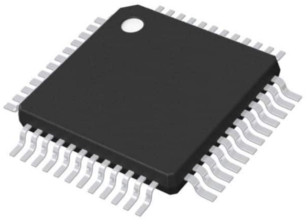
TinyNAS

1. Automated search space optimization
2. Resource-constrained model specification

TinyEngine

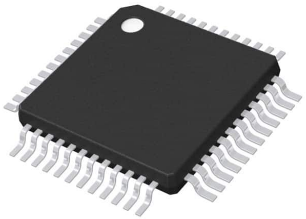
1. Reducing overhead with separated compilation and runtime
2. In-place depth-wise convolution

TinyNAS: Neural Architecture Search



How should one implement Neural Architecture Search?

TinyNAS: Neural Architecture Search

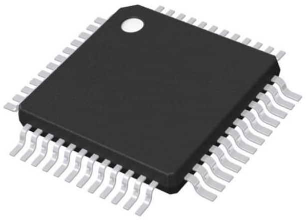


Neural Architecture Search = a method to find a neural network architecture automatically

3 main parts:

- **Search space**
- **Optimization algorithm (ie Gradient descent, random search, etc)**
- **Performance evaluation (ie accuracy, or approximation of accuracy)**

MCUNet: Tiny Deep Learning on IoT Devices



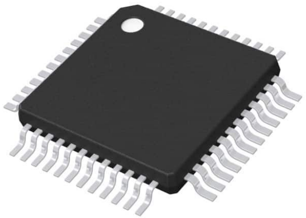
TinyNAS

1. **Automated search space optimization**
2. Resource-constrained model specification

TinyEngine

1. Reducing overhead with separated compilation and runtime
2. In-place depth-wise convolution

TinyNAS - Automated Search Space Optimization



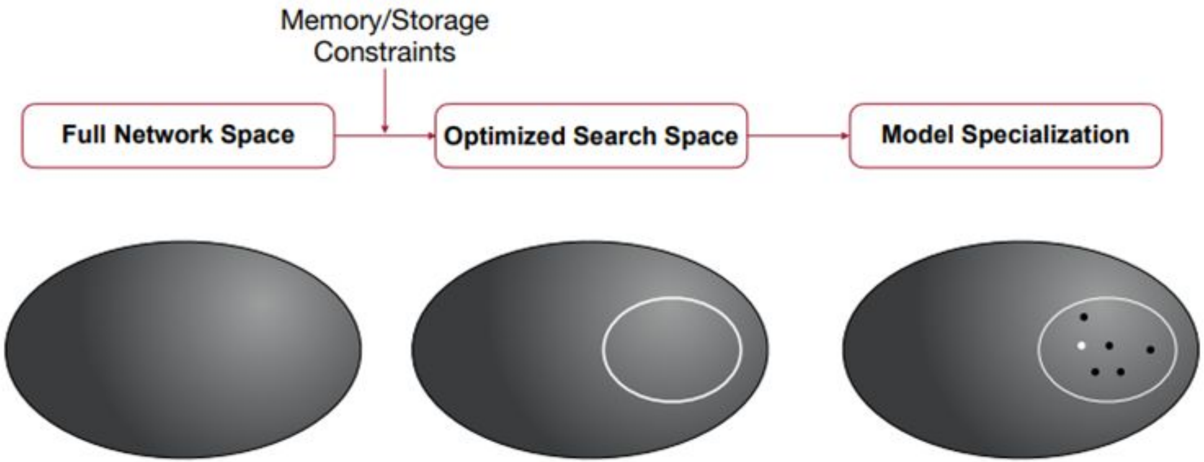
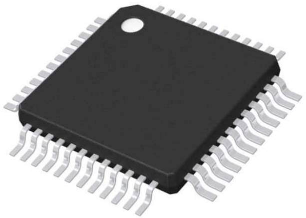


Image source: MCUNet (mit.edu)

TinyNAS - Automated Search Space Optimization



$$S = W \times R$$

Search Space

108 possible
search spaces

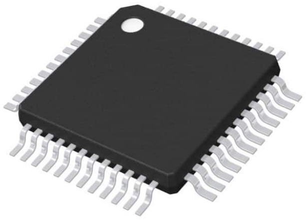
Width Multiplier

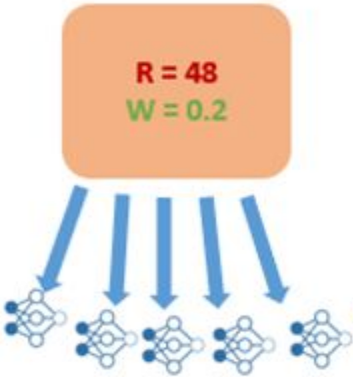
$W = \{0.2, 0.3, 0.4, \ldots, 1.0\}$

Image Resolution

$R = \{48, 64, 80, \ldots, 192, 208, 224\}$

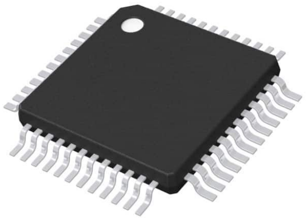
TinyNAS - Automated Search Space Optimization

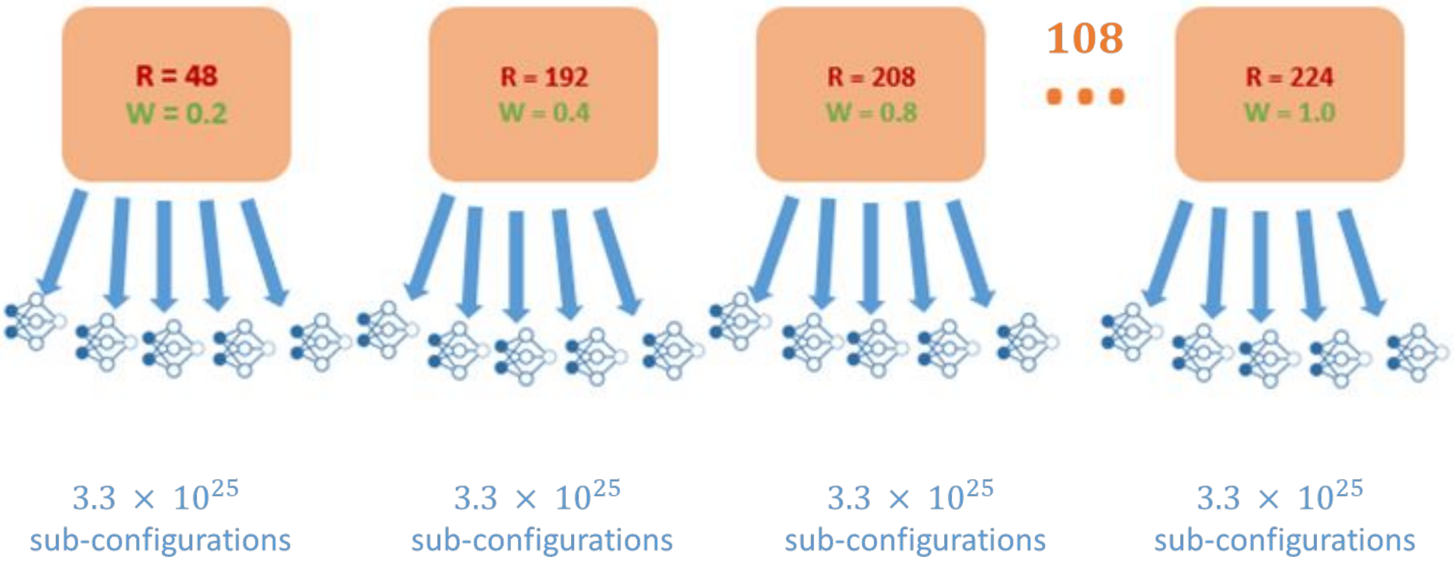




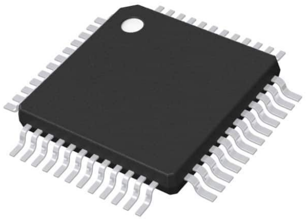
3.3×10^{25}
sub-configurations

TinyNAS - Automated Search Space Optimization





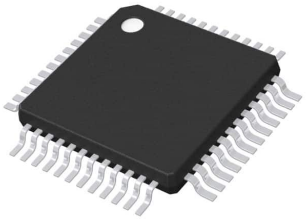
TinyNAS - Automated Search Space Optimization



Given all possible search spaces, how do we find the one that has the best model within it?

Simple Answer: Examine each individual search space to find the best model, and compare across all search spaces

TinyNAS - Automated Search Space Optimization

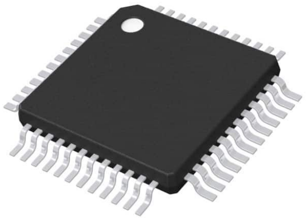


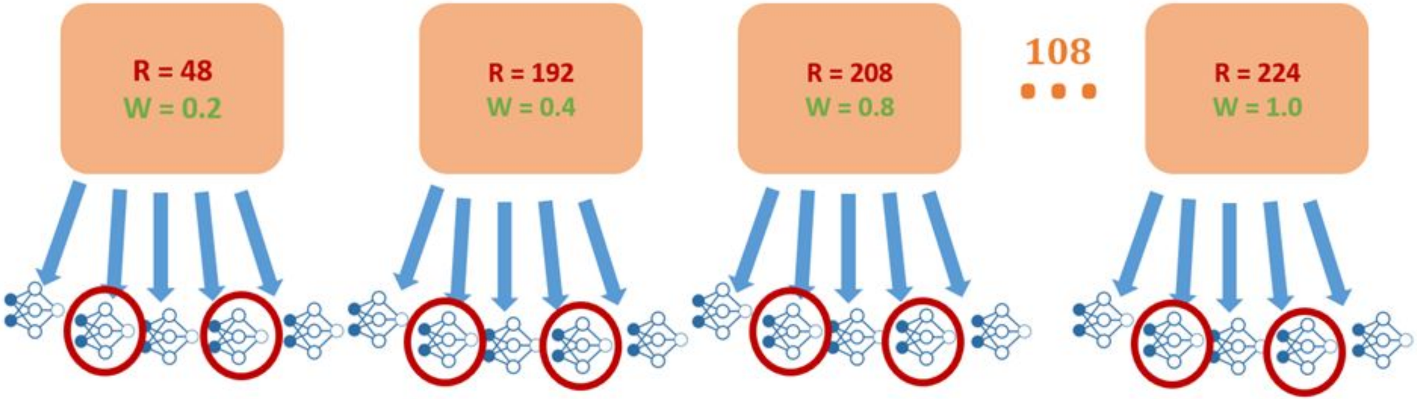
Given all possible search spaces, how do we find the one that has the best model within it?

Simple Answer: Examine each individual search space to find the best model, and compare across all search spaces

More Efficient Answer: Sample **m** networks in each search space

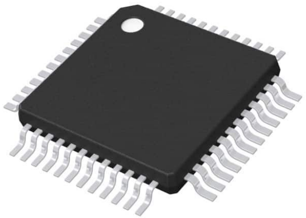
TinyNAS - Automated Search Space Optimization





But how do we compare models?

TinyNAS - Automated Search Space Optimization



Key Assumption:

Accuracy and computation are positively correlated within the same model family

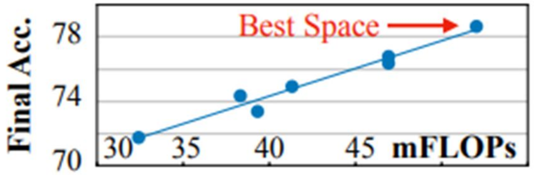
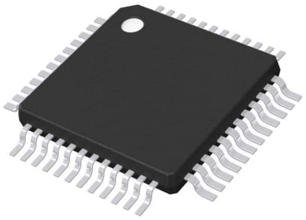


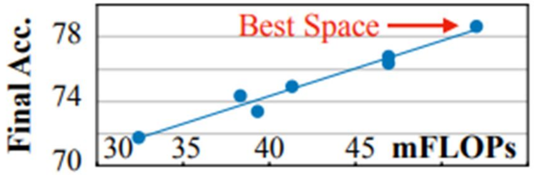
Image source: MCUNet (mit.edu)

TinyNAS - Automated Search Space Optimization



Key Assumption:

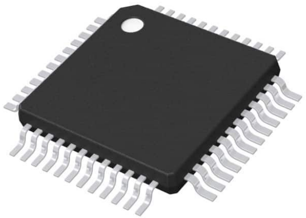
Accuracy and computation are positively correlated within the same model family



Use CDF of FLOPs as an approximation of the CDF of accuracy when comparing search spaces!

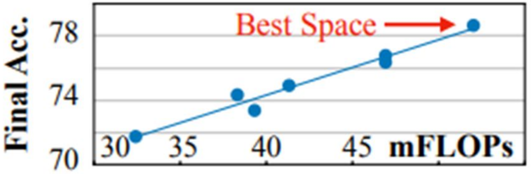
Image source: MCUNet (mit.edu)

TinyNAS - Automated Search Space Optimization



Key Assumption:

Accuracy and computation are positively correlated within the same model family



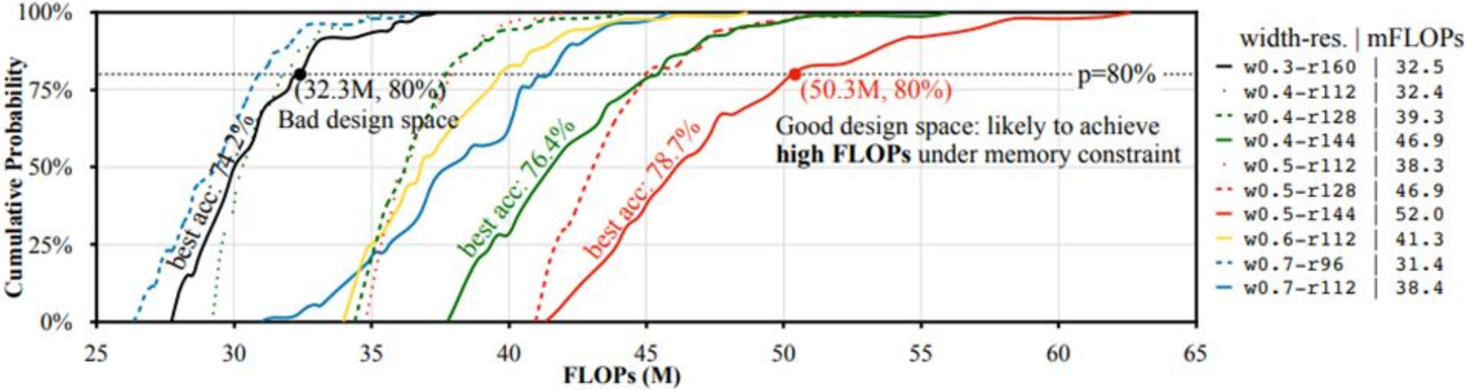
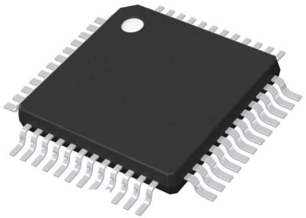
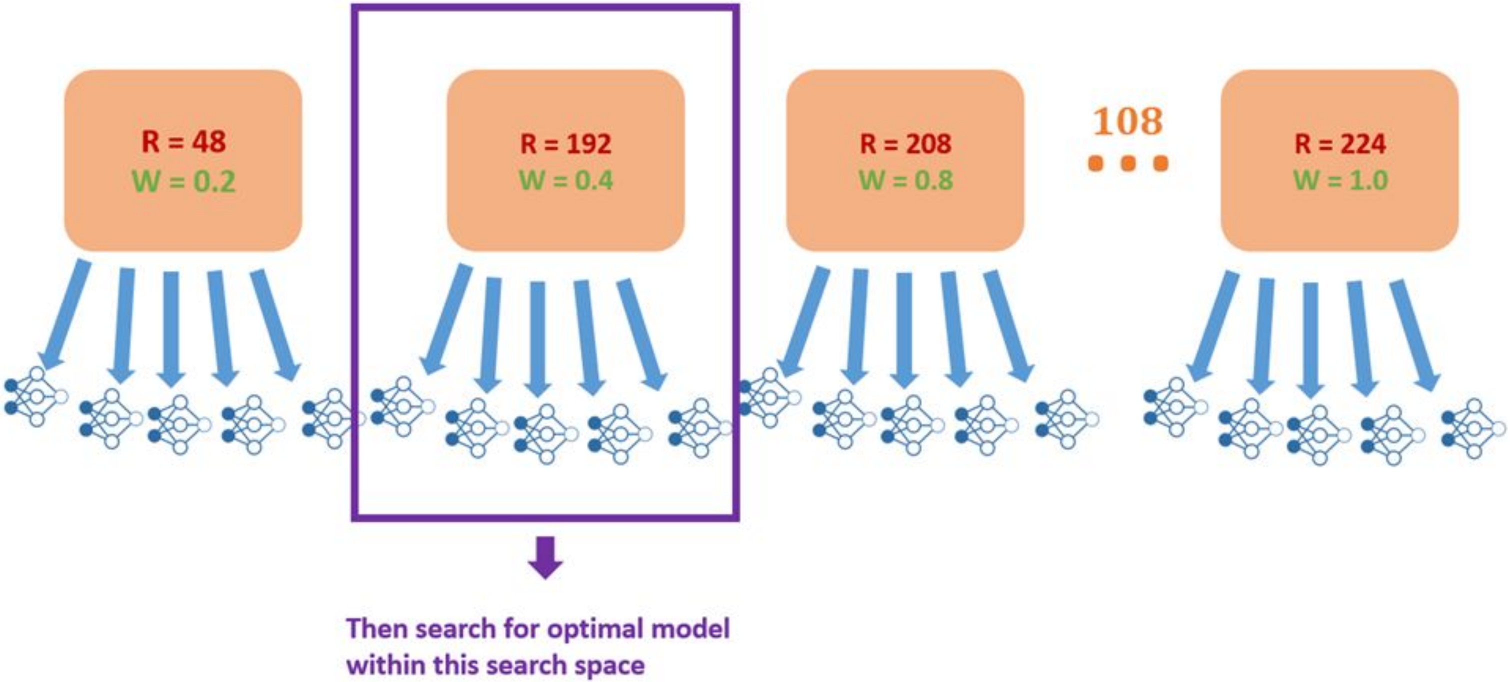


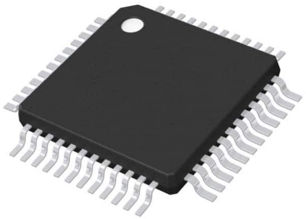
Image source: MCUNet (mit.edu)

TinyNAS - Automated Search Space Optimization





MCUNet: Tiny Deep Learning on IoT Devices



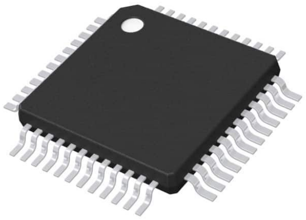
TinyNAS

1. Automated search space optimization
2. **Resource-constrained model specialization**

TinyEngine

1. Reducing overhead with separated compilation and runtime
2. In-place depth-wise convolution

TinyNAS - Resource Constrained Model Specialization



Model Configurations - Mobile Search Space

- **Kernel size** for depthwise convolution : 3, 5, 7
- **Expansion** ratios for inverted bottleneck : 3, 4, 6
- Stage **depths** : 2, 3, 4

Train largest network, use these weights to train all other subnetworks

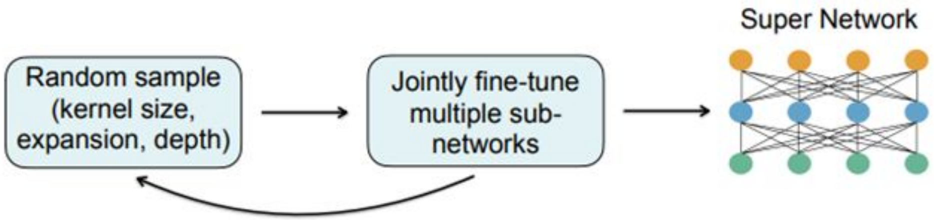
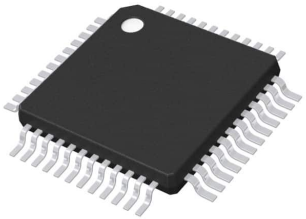


Image source: MCUNet (mit.edu)

MCUNet: Tiny Deep Learning on IoT Devices



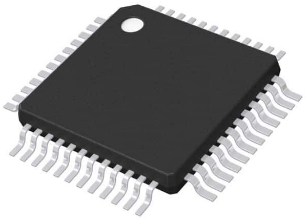
TinyNAS

1. Automated search space optimization
2. Resource-constrained model specification

TinyEngine

1. Reducing overhead with separated compilation and runtime
2. In-place depth-wise convolution

MCUNet: Tiny Deep Learning on IoT Devices



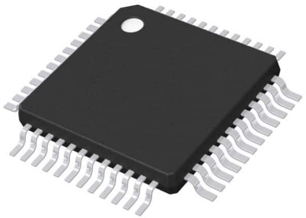
TinyNAS

1. Automated search space optimization
2. Resource-constrained model specification

TinyEngine

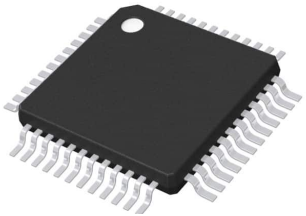
1. **Reducing overhead with separated compilation and runtime**
2. In-place depth-wise convolution

TinyEngine - Separation of Compilation and Runtime



TFLM	TinyEngine
Interpreter-based	Compiles code offline to reduce runtime
Suited for all models	Compiles operations for one specific model
Memory scheduling based on layer configuration only	Memory scheduling additionally takes into account statistics of the model

TinyEngine - Separation of Compilation and Runtime



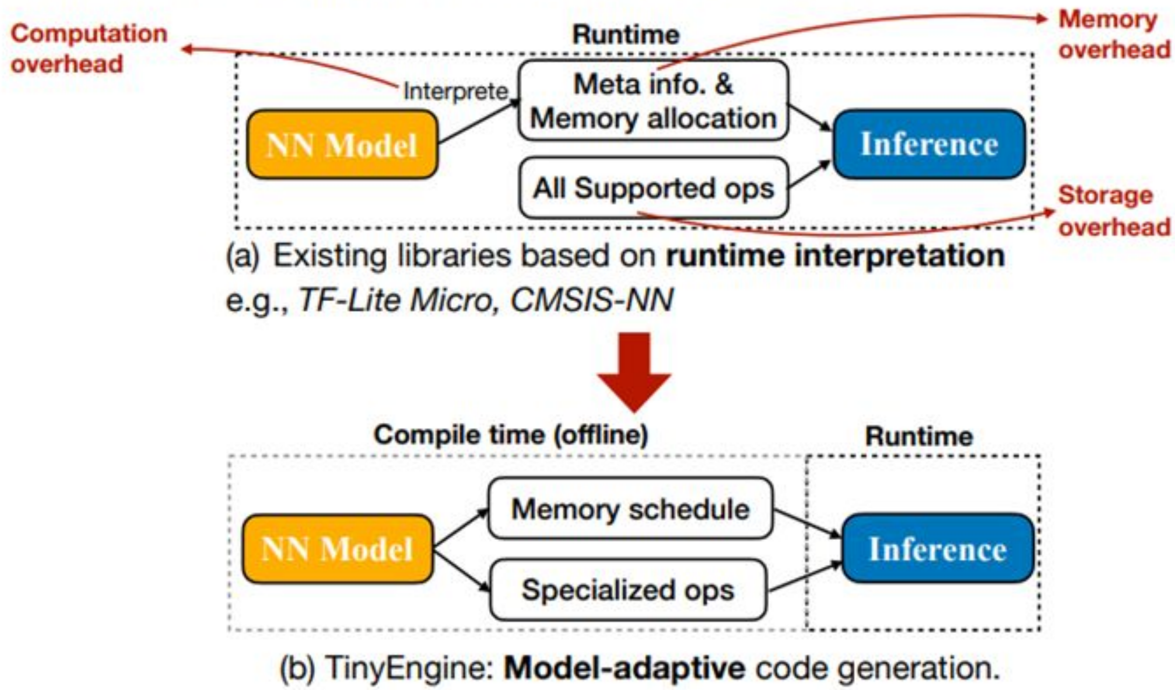
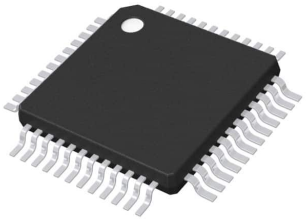


Image source: MCUNet (mit.edu)

MCUNet: Tiny Deep Learning on IoT Devices



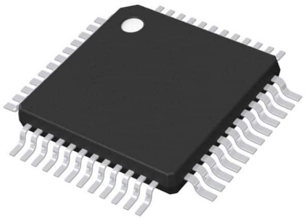
TinyNAS

1. Automated search space optimization
2. Resource-constrained model specification

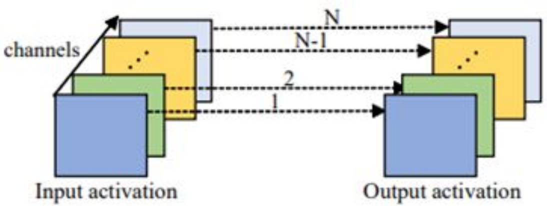
TinyEngine

1. Reducing overhead with separated compilation and runtime
2. **In-place depth-wise convolution**

TinyEngine - In-Place Depthwise Convolution

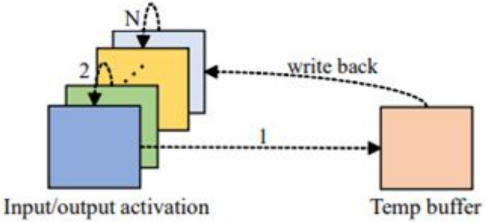


No filtering across channels = Reduced peak memory



(a) Depth-wise convolution

Peak Memory: 2N

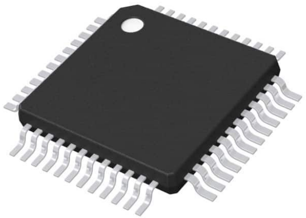


(b) In-place depth-wise convolution

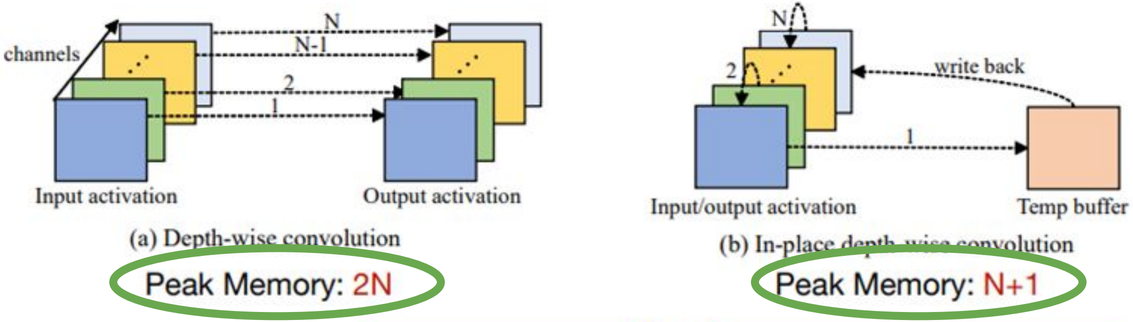
Peak Memory: N+1

Image source: MCUNet (mit.edu)

TinyEngine - In-Place Depthwise Convolution



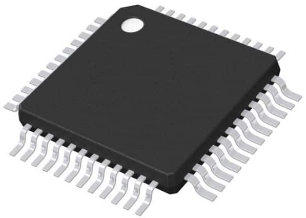
No filtering across channels = Reduced peak memory

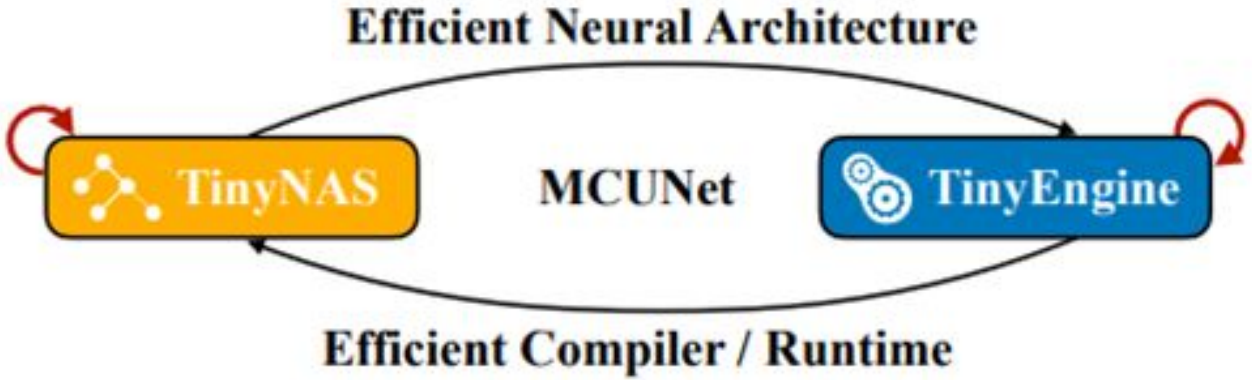


Other methods to reduce peak memory? … Future work

Image source: MCUNet (mit.edu)

MCUNet: Tiny Deep Learning on IoT Devices

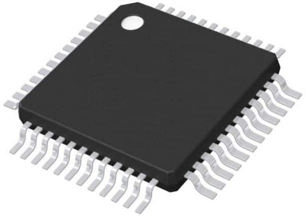


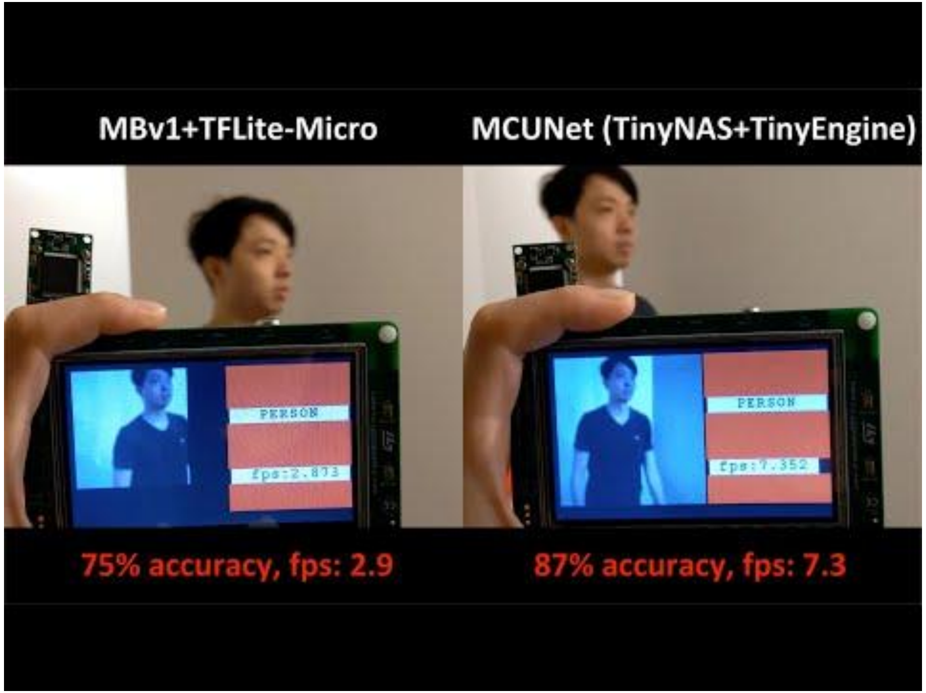


(c) *MCUNet*: system-algorithm co-design

Image source: MCUNet (mit.edu)

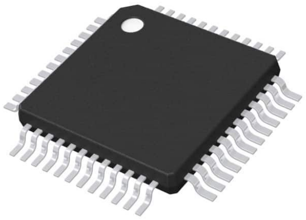
MCUNet: Tiny Deep Learning on IoT Devices





Video source: MCUNet (mit.edu)

Algorithmic Optimizations



- Background of existing optimizations for TinyML
- Contributions of MCUNet
- **MicroNets**
- Future work / directions in the field

NAS for TinyML

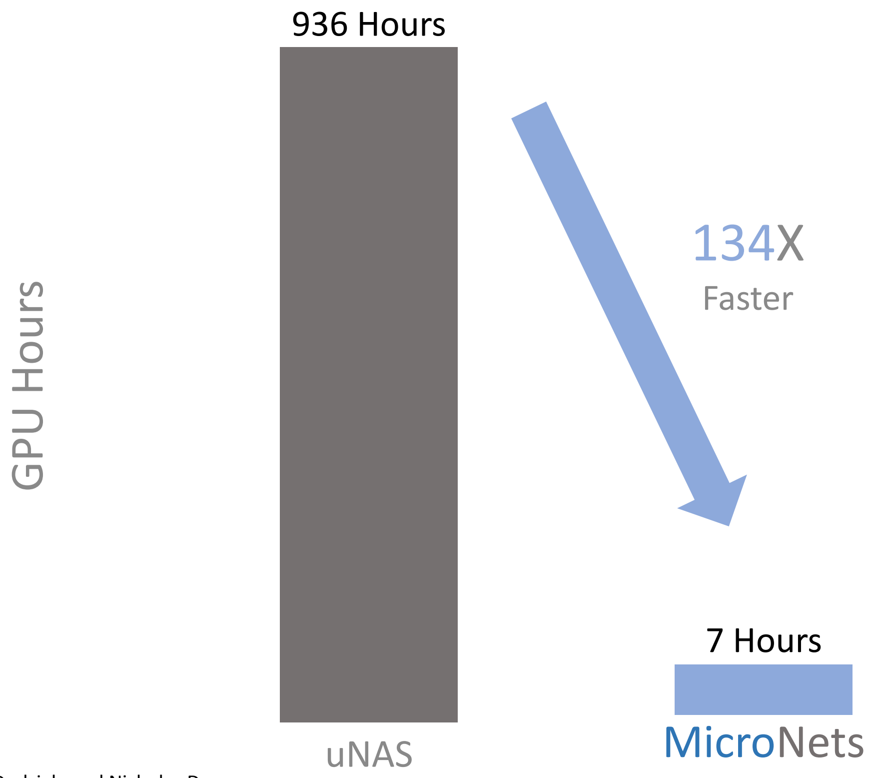
		SpArSe	MCUNet	uNAS	MicroNets
	Search Method	Bayesian	Evolutionary	Evolutionary	**Gradient Based**
Optimize	Latency	No	Yes	Yes	**Yes**
	SRAM	Yes	Restrict Search Space	Yes	**Yes**
	Flash	# of Parameters	Restrict Search Space	Yes	**Yes**

Fedorov, Igor, et al. "Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers." NeurIPS (2019).

Lin, Ji, et al. "Mcunet: Tiny deep learning on iot devices." NeurIPS (2020).

Liberis, Edgar, Łukasz Dudziak, and Nicholas D. Lane. "µNAS: Constrained Neural Architecture Search for Microcontrollers." *EuroMLSys*. 2021.

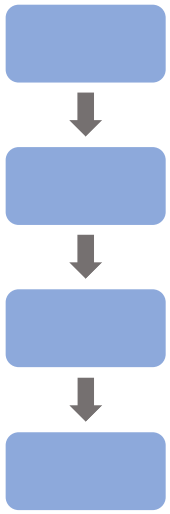
Differentiable Neural Architecture Search (DNAS)



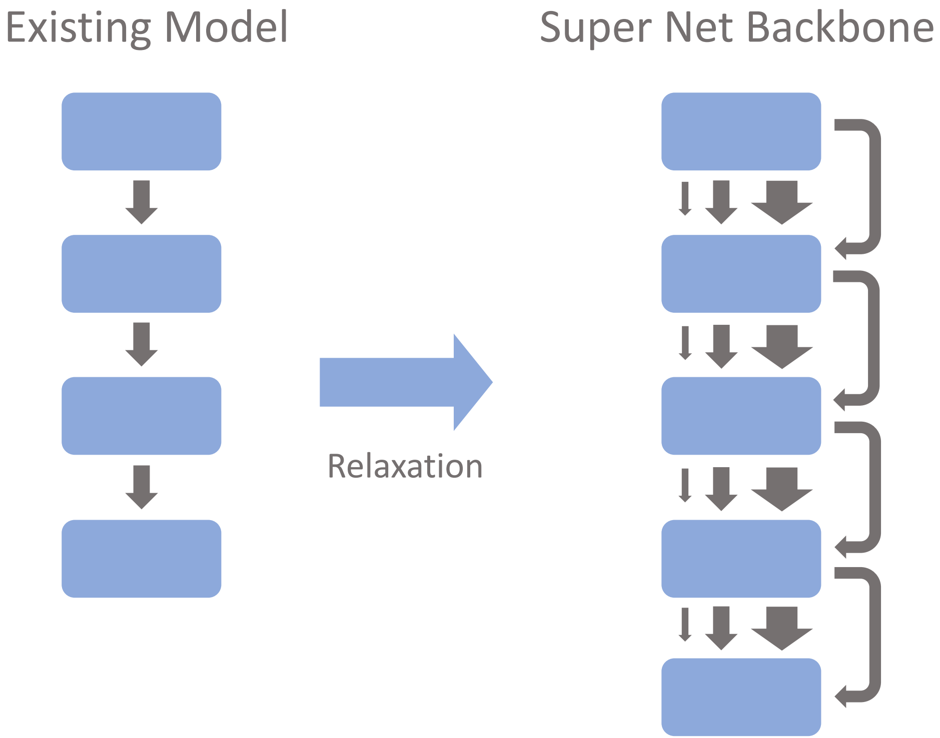
Liberis, Edgar, Łukasz Dudziak, and Nicholas D. Lane. "µNAS: Constrained Neural Architecture Search for Microcontrollers." *EuroMLSys*. 2021.

Differentiable Neural Architecture Search (DNAS)

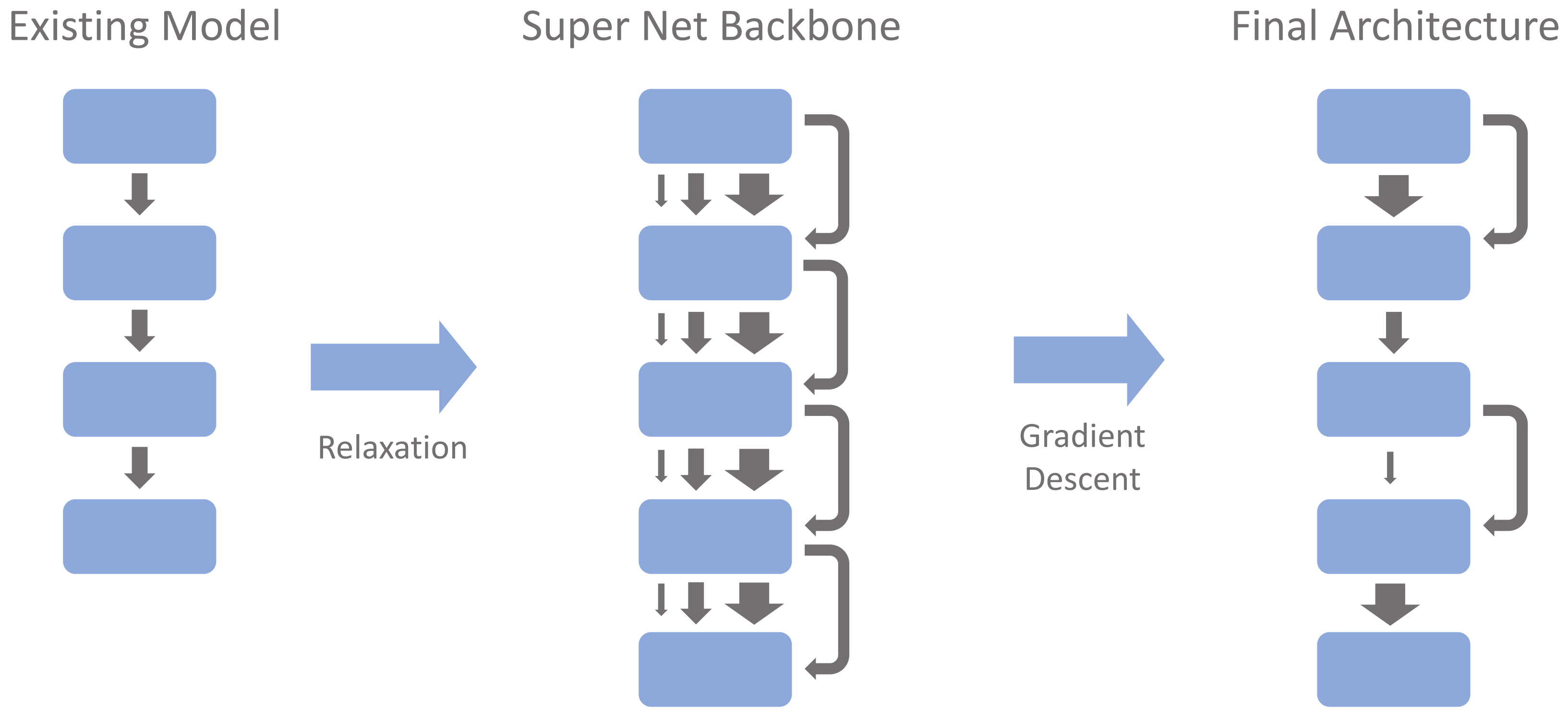
Existing Model



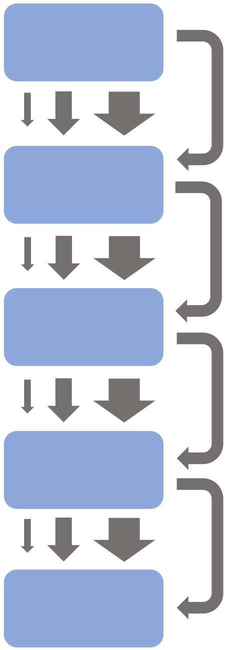
Differentiable Neural Architecture Search (DNAS)



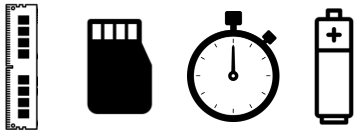
Differentiable Neural Architecture Search (DNAS)



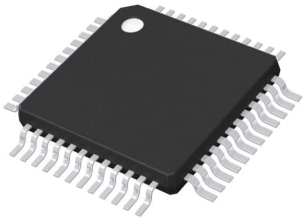
Differentiable Neural Architecture Search (DNAS)



DNAS is **fast** to solution but needs **continuous functions** for all of the objectives



Algorithmic Optimizations



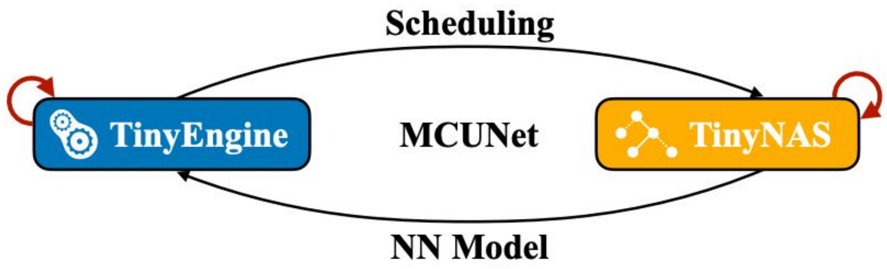
- Background of existing optimizations for TinyML
- Contributions of MCUNet
- MicroNets
- **Future work / directions in the field**

MCUNet Recap

MCUnet was designed to combine TinyNAS and the interface library TInyEngine, enabling deep learning on tiny hardware resources.

They achieved a record ImageNet accuracy (70.7%) on off the shelf microcontroller, and accelerated the inference wake word application by 2.4 - 3.2x.

People have brought down the cost of deep learning inference from $5,000 workstation GPU to $500 mobile phones. MCUNet brought it down to $5 or less



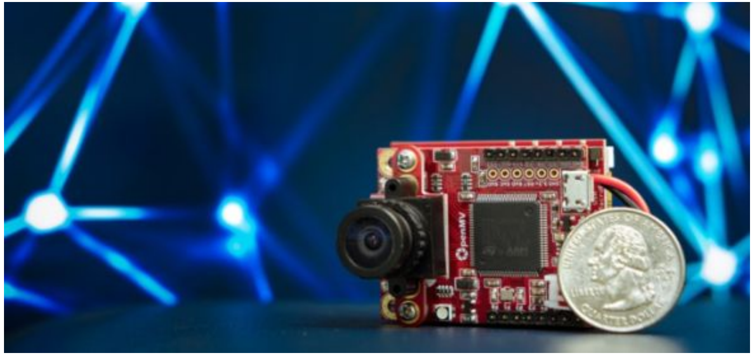
MCUNet: system-algorithm co-design

MCUNet v2

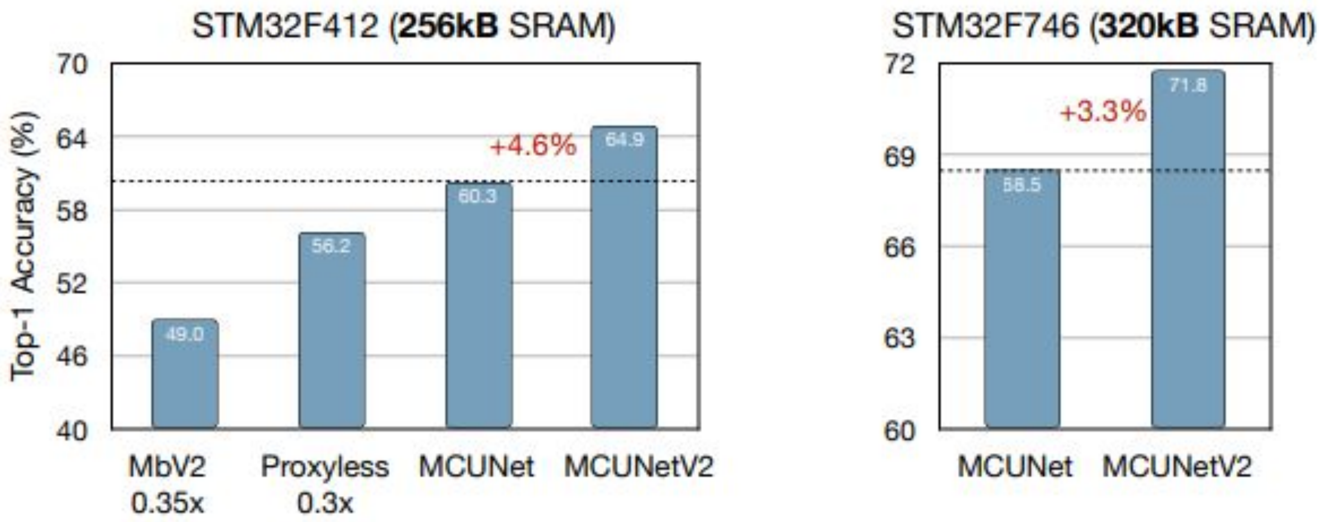
Patch by patch inference scheduling, which operates only on a small spatial region of the feature map and significantly cuts down the peak memory

Reduces peak memory usage of existing networks by 2 - 8x.

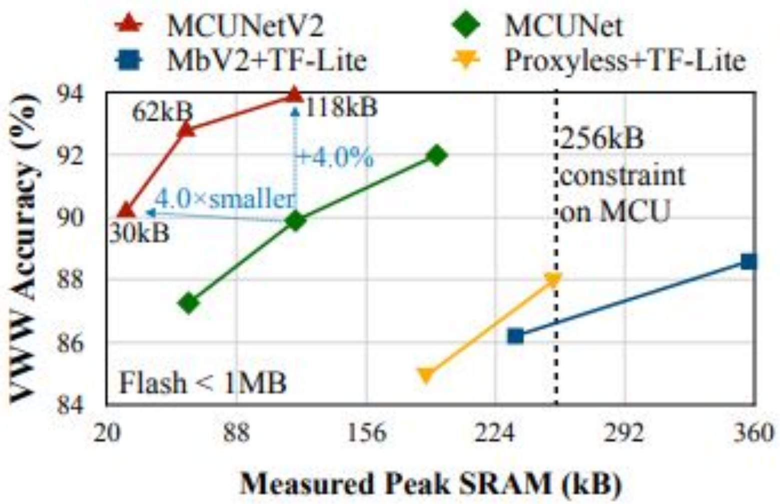
Automate the process with neural architecture search to jointly optimize the neural architecture and inference scheduling

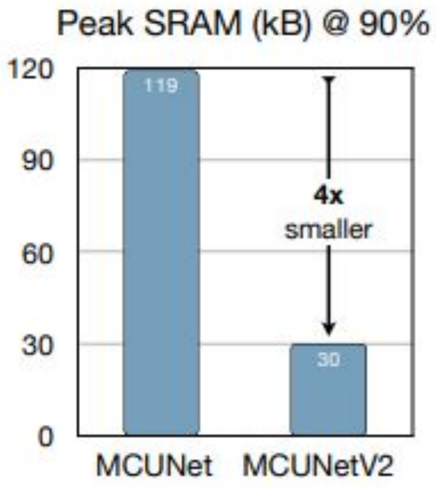


ImageNet Classification



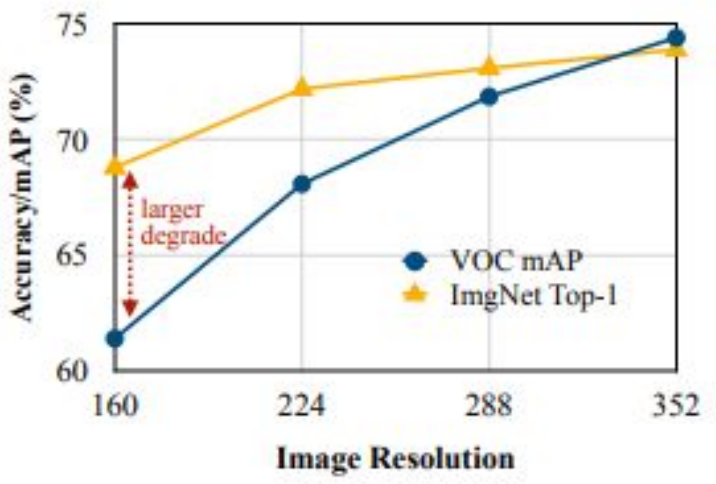
Visual Wake Words (VWW)

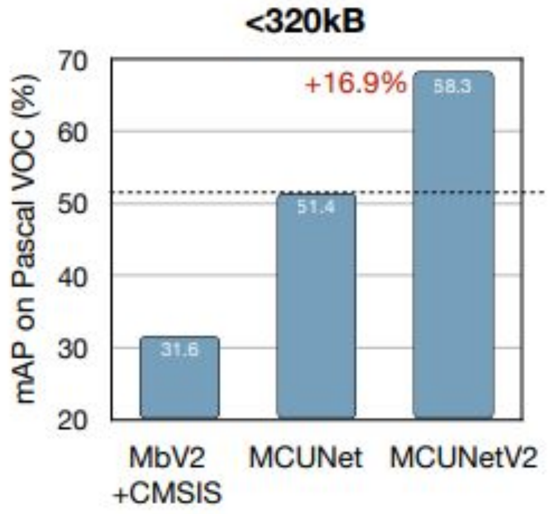




Object Detection

Object detection is more sensitive to input resolution





(a) RNNPool-Face-Quant (b) MCUNetV2


Demo
MCUNetV2: Memory-Efficient Patch-based
Inference for Tiny Deep Learning
Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, Song Han
mcunet.mit.edu

MCUNet v2 Recap

Patch based inference effectively reduces the peak memory usage of existing network by 4-8x.

On imagenet, they achieve a record accuracy of 71.8% on MCU

On visual wake words dataset, they achieved higher than 90% accuracy under only 32kB SRAM (4.0x smaller than MCUNetV1)

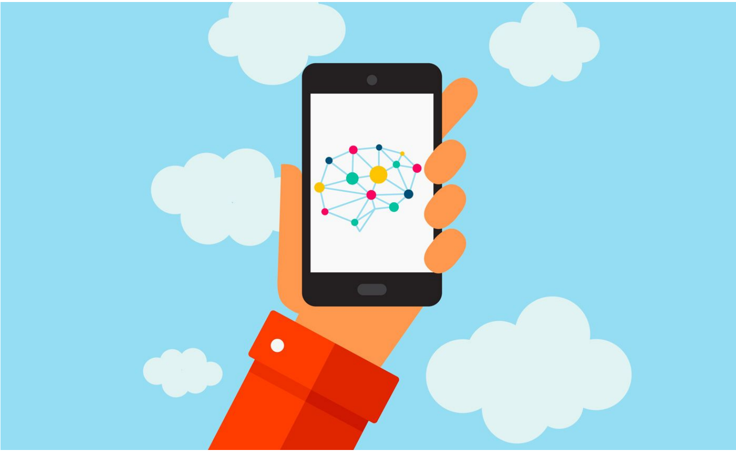
Object detection on tiny devices, achieving 16.9% higher mAP of Pascal VOC compared to the state of the art.

MCUNet v3

On device training enables the model to adapt to new data collected from the sensor by fine tuning a pre trained model

On device training faces two unique challenges

1.) The quantized graph of neural network are hard to optimize due to mixed bit precision and the lack of normalization
2.) The limited hardware resource (memory and computation) does not allow full backward computation

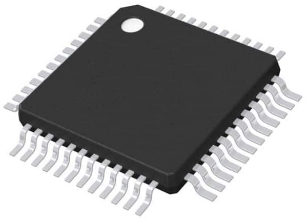


MCUNet v3

Quantization Aware Scaling - Automatically scale the gradient of tensors with different bit - precision, which effectively stabilizes the training and matches the accuracy of the floating point counterpart (No tuning is required)

To reduce the memory footprint of the full backward computation, Sparse Update is proposed to skip the gradient computation of less important layers and subtensors.

First solution to actually enable tiny on device training of convolutional neural network under 256KB



Thank You!

Questions?