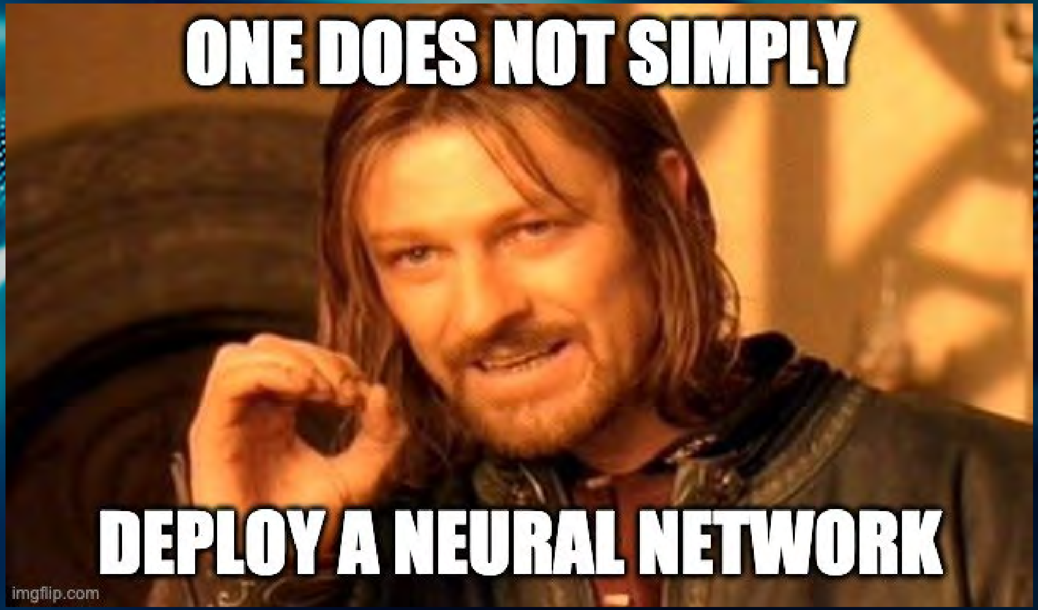
TinyML Model Design

2nd On-Device Intelligence Workshop @ MLSys 2021



Igor Fedorov
Arm ML Research Lab

April 9, 2021

Tiny Hardware

- ~50 billion MCU chips shipped in '19
  - ~100 million GPUs in '18
- Severe memory limitations
  - Limited flash memory —> limited model size
  - Limited SRAM —> limited feature map size
- LeNet for MNIST
  - 420 KB flash
  - 12 KB SRAM

Table 1: Processors for ML inference: estimated characteristics to indicate the relative capabilities.

Processor	Usecase	Compute	Memory	Power	Cost
Nvidia 1080Ti GPU	Desktop	10 TFLOPs/Sec	11 GB	250 W	$700
Intel i9-9900K CPU	Desktop	500 GFLOPs/Sec	256 GB	95 W	$499
Google Pixel 1 (Arm CPU)	Mobile	50 GOPs/Sec	4 GB	~5 W	–
Raspberry Pi (Arm CPU)	Hobbyist	50 GOPs/Sec	1 GB	1.5 W	–
Micro Bit (Arm MCU)	IoT	16 MOPs/Sec	16 KB	~1 mW	$1.75
Arduino Uno (Microchip MCU)	IoT	4 MOPs/Sec	2 KB	~1 mW	$1.14

- "Bonsai is not compared to deep convolutional neural networks as they have not yet been demonstrated to fit on such tiny IoT devices" [2]
- "Consider a typical IoT device that has ≤ 32kB RAM and a 16MHz processor. Most existing ML models cannot be deployed on such tiny devices" [3]

[1] Fedorov et al., SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers. NeurIPS '19
[2] Kumar et al. Resource-efficient machine learning in 2 kb ram for the internet of things. ICML '17
[3] Gupta et al. Protonn: Compressed and accurate knn for resource-scarce devices. ICML '17

Can we design the model *for* the device?

Research papers

- Is the source code publicly available?
- What are the compute resources required?

Deep learning training software
- Tensorflow
- Pytorch
- etc.

Deployment tool
- TFlite-micro [4]
- TinyEngine [5]

- Are all of the operators supported?
- All of compute graphs / structures?

Hardware

- What features are supported?

[4] David et al. TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems. MLSys ’21
[5] Lin et al. MCUNet: Tiny Deep Learning on IoT Devices. NeurIPS ’21

Algorithmic Tools

- Quantization
  - Int8 cheaper than float (storage + compute)
  - Sub int8 not supported by HW
- Pruning
  - Structured pruning
    - HW friendly
    - Reduces ops
    - Limited compression benefits
  - Unstructured / random pruning
    - Not HW friendly unless extreme
    - Large compression benefits

- Neural architecture search
  - Operators, connectivity, layer width, resolution, etc.
  - Computationally demanding
  - Not all computational graphs supported by deployment tools
- Learn from the HW directly
  - HW model, or
  - HW interface

Quantization

- Float —> int8 (weights + activations)
  - 4x reduction in model size
  - 4x reduction in feature map size
  - Cheaper computation

- Sub 8-bit and non-uniform not supported by HW

- $Q(w) = s \times round\left(\frac{clip(w, -w_{max}, w_{max})}{s}\right)$

  $s = \frac{w_{max}}{2^{8-1} - 1}$ [6]

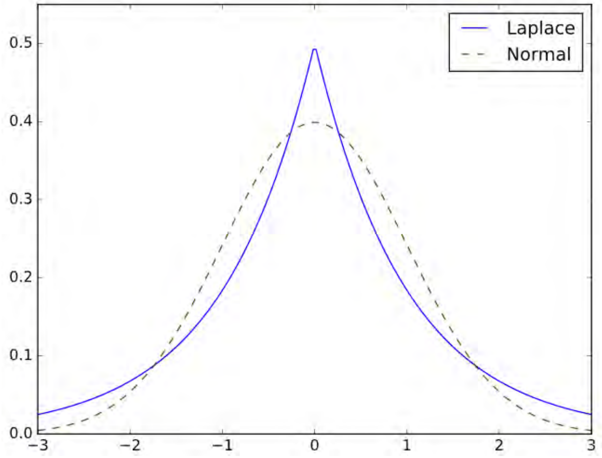
- How to select w_{max} ? Depends…

- If training data is available
  - Post-training calibration [7]
  - Quantization aware training [8]
    - Treat w_{max} as a variable and optimize by GD on the task objective
    - Two dependencies on w_{max}
    - Straight through estimator

- If no training data, still have options [9]

[6] https://intellabs.github.io/distiller/algo_quantization.html
[7] https://www.tensorflow.org/lite/performance/post_training_quantization
[8] Jain et al., Trained Quantization Thresholds for Accurate and Efficient Fixed-Point Inference of Deep Neural Networks. MLSys '20
[9] Nagel et al., Data-Free Quantization Through Weight Equalization and Bias Correction. ArXiv '19

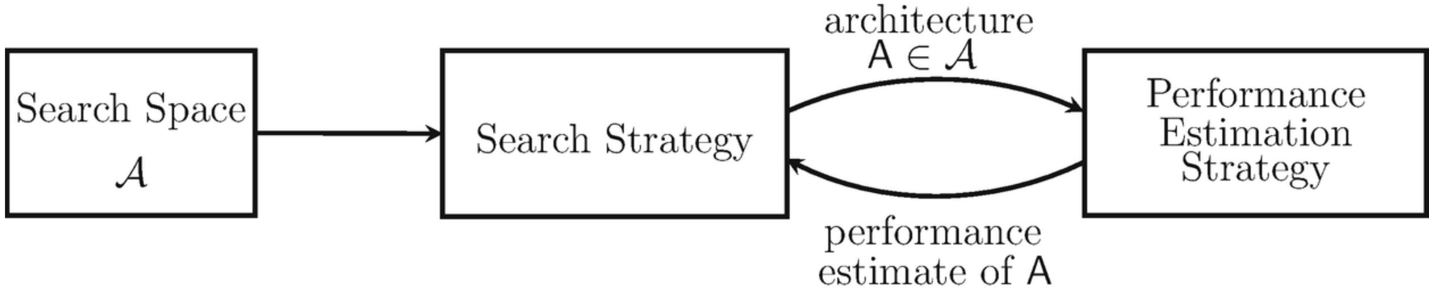
Pruning



- Unstructured [10] vs structured [11]
  - Can the HW benefit from highly compressed weights?
- Sparsity promoting regularization —> drive the weights to 0
  - How to pick number of non-zeros per layer?
  - Reinforcement learning [12]
- Gradient based [13]
  - $w \leftarrow w \times 1_{w>\tau}$,approximate indicator by sigmoid during backdrop
- Rank —> pruned —> retrain —> (repeat)
  - Magnitude [14], minimal influence on objective [15]

[10] Molchanov et al. Variational dropout sparsifies deep neural networks. ICML '17
[11] Louizos et al. Bayesian compression for deep learning. NeurIPS '17
[12] He et al. AMC: AutoML for Model Compression and Acceleration on Mobile Devices. ECCV '18
[13] Fedorov et al. TinyLSTMs: Efficient Neural Speech Enhancement for Hearing Aids. Interspeech '20
[14] Han et al. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. ICLR '16
[15] Molchanov et al. Importance Estimation for Neural Network Pruning. CVPR '19

Neural architecture search



- The search strategy
  - Black box (RL, Bayesian optimization, genetic alg., etc.) [16] vs gradient based [17]
  - Hardware model —> do we need it, or is there a viable proxy?

- Estimation strategy
  - Computationally expensive —> thousands of GPU days in some cases
  - Weight sharing
  - Morphisms

- The distinction between quantization / pruning and NAS is arbitrary [18]

[16] Elsken et al. Neural Architecture Search: A Survey. JMLR '19
[17] Liu et al. DARTS: Differentiable Architecture Search. ICLR '19
[18] Cai and Vasconcelos. Rethinking Differentiable Search for Mixed-Precision Neural Networks. CVPR '20

A few examples and lessons from our work

SpArSe [1]

- Multi-objective Bayesian optimization
- Architecture, channel / weight pruning thresholds, training hyperparameters
- Closed form memory model
  - SRAM usage modeled by sum of input and output tensors for each layer
- Able to deploy to devices previously thought too small for NNs
- The deployment tool (uTensor) only supported feed-toward graphs
- Bayesian optimization is slow, even with “tricks” to speed it up —> 10 GPU days on CIFAR10-binary

	MNIST			CIFAR10-binary			CUReT			Chars4k		
	Acc	$\|\|\omega\|\|_0$	GPUD	Acc	$\|\|\omega\|\|_0$	GPUD	Acc	$\|\|\omega\|\|_0$	GPUD	Acc	$\|\|\omega\|\|_0$	GPUD
Bonsai	**97.24** 97.01	**510** 2.15e4	11	**73.08** 73.02	**487** 512	1	**96.45** 95.23	**8.5e3** 2.9e4	1	**67.82** 58.59	**1.7e3** 2.6e4	1
Bonsai (16 kB)	–	–	–	**76.66** 76.64	**1.4e3** 4.1e3	9	–	–	–	–	–	–
ProtoNN	**96.84** 95.88	**476** 1.6e4	11	**76.56** 76.35	**1.4e3** 4.1e3	10	**96.45** 94.44	**8.5e3** 1.6e4	1	–	–	–
GBDT	**98.78** 97.90	**804** 1.5e6	11	**77.90** 77.19	**1.6e3** 4e5	8	**96.45** 90.81	**8.5e3** 6.1e5	1	**67.82** 43.34	**1.7e3** 2.5e6	1
kNN	**96.84** 94.34	**476** 4.71e7	11	**76.34** 73.70	**1.4e3** 2e7	10	**96.45** 89.81	**8.5e3** 2.6e6	2	**67.82** 39.32	**1.7e3** 1.7e6	1
RBF-SVM	**97.42** 97.30	**569** 1e7	10	**81.77** 81.68	**3.2e3** 1.6e7	3	**97.58** 97.43	**2.2e4** 2.3e6	2	**67.82** 48.04	**1.7e3** 2e6	1
LeNet + SpVD	**99.16** 99.10	**1e3** 1.8e3	8	**75.35** 75.09	**1.4e3** 1.6e5	10	–	–	–	–	–	–
MODC	**99.17** 99.15	**1.45e3** 3e3	1	–	–	–	–	–	–	–	–	–

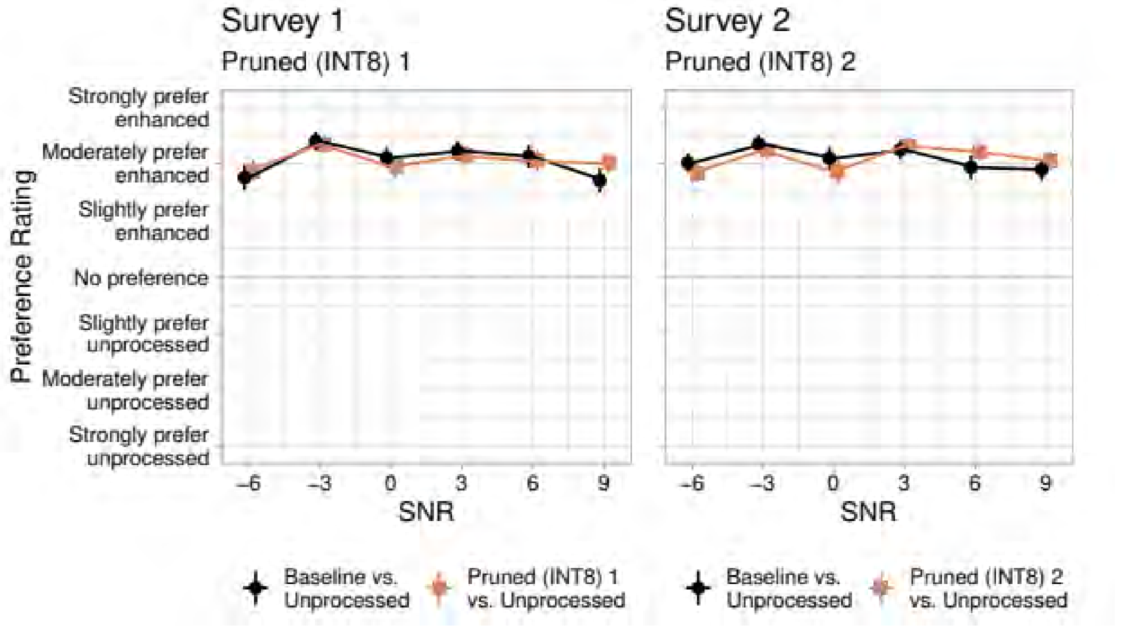
[1] Fedorov et al., SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers. NeurIPS ’19

A few examples and lessons from our work

TinyLSTMs [13]

- Speech denoising using LSTMs
- Latency constraint w/ ops proxy
- Neuron pruning to reduce ops
  - Gradient based threshold learning for efficiency
- Quantization to run w/ integer math
- LSTMs were not supported by TFlite-micro

	SISDR (dB)	BSS SDR (dB)	Params (M)	MS (MB)	WM (KB)	MOps/inf.	Latency (ms/inf.)	Energy (mJ/inf.)	GPUH
Baseline (FP32)	11.99	12.77	0.97	3.70	26.0	1.94	12.52*	6.76*	14
Pruned (FP32)	11.99	12.78	0.52	1.97	18.8	1.04	6.71*	3.62*	72
Pruned (INT8) 1	11.80	12.69	0.61	0.58	5.1	1.22	7.87	4.25	61
Pruned (INT8) 2	11.47	12.22	0.33	0.31	3.7	0.66	4.26	2.30	144
Pruned Skip RNN (INT8)	11.42	12.07	0.46	0.43	4.67	0.37†	2.39†	1.29†	275
Erdogan et al. [20]	-	13.36	0.96	3.65	25.9	1.92	12.39	6.69	-
Wilson et al. [6]	-	14.60	65	247.96	4472.6	130	839*	453	18360

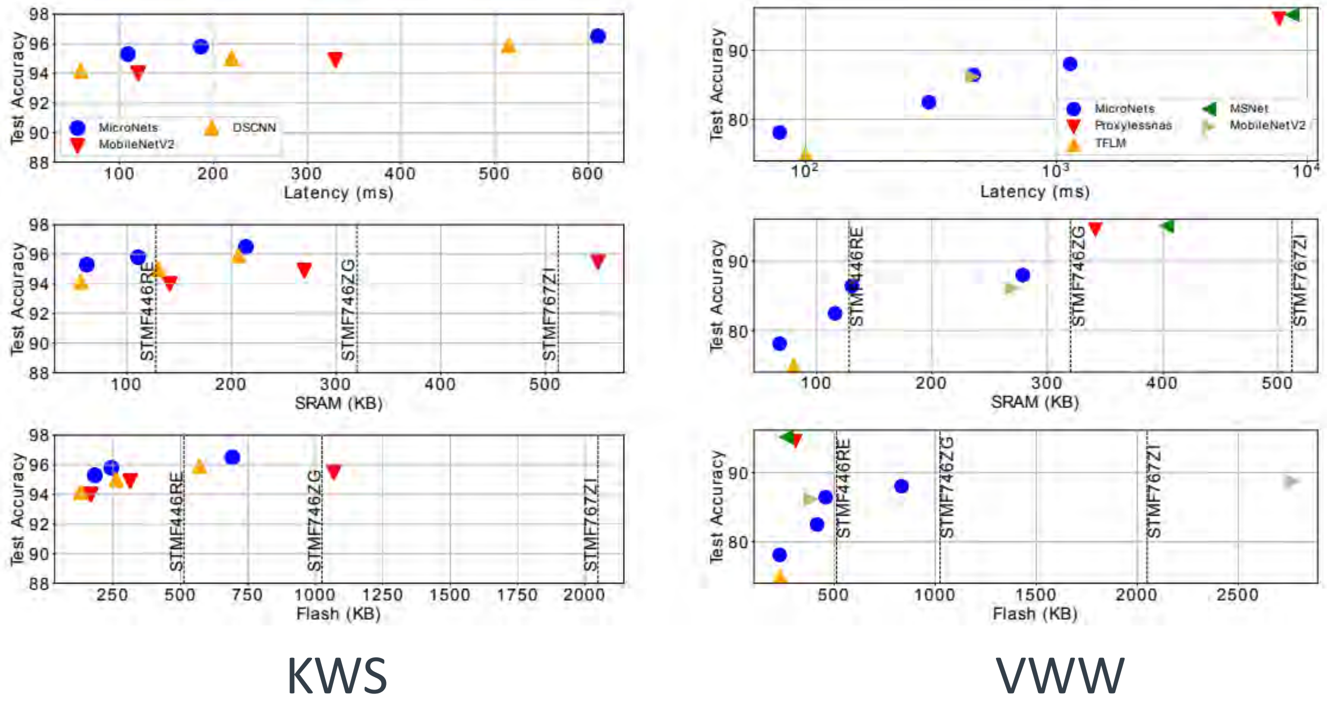


[13] Fedorov et al. TinyLSTMs: Efficient Neural Speech Enhancement for Hearing Aids. Interspeech '20

A few examples and lessons from our work

MicroNets [19]

- Differentiable NAS
- Commodity MCU target
- 3 TinyMLperf datasets
- Optimize for Flash, SRAM, ops —> good proxy for latency on MCUs
- Search cost on the order of hours
- TFlite-micro deployment tool
- Memory overheads difficult to predict



[19] Banbury et al. MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers. MLsys ’21

