SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers

Igor Fedorov[*], Ryan P. Adams[#], Matthew Mattina[*], Paul N. Whatmough[*]

igor.fedorov@arm.com, rpa@princeton.edu, matthew.mattina@arm.com, paul.whatmough@arm.com

[*]Arm ML Research, [#]Princeton University

Introduction

- The microcontroller unit (MCU) is a ubiquitous computer. A typical home has ~40 MCUs. Total annual shipments estimated at 50 billion units, compared to 100 million GPUs

  MCUs have particularly stringent memory limitations. Researchers have moved away from CNNs to models specialized for MCU deployment. MCU hardware induces the following model constraints:
  - Maximum size of intermediate feature maps cannot exceed RAM capacity
  - Model parameters must not exceed the ROM (flash memory) capacity

- We perform Sparse Architecture Search (SpArSe), using Bayesian optimization to traverse the space of neural architectures and pruning strategies in search of models that satisfy MCU constraints.

Processor	Usecase	Compute	Memory	Power	Cost
Nvidia 1080Ti GPU	Desktop	10 TFLOPs/sec	11 GB	250 W	$700
Google Pixel 1 (Arm CPU)	Mobile	50 GOPs/sec	4 GB	~5 W	
Micro Bit (ARM MCU)	IoT	16 MOPs/sec	16 KB	~1 mW	$1.75

SpArSe: CNN design as multi-objective optimization

- Ω: network, training, and pruning hyperparameters
- The multi-objective optimization problem:

$$f_1(\Omega) = 1 - ValidationAccuracy(\Omega)$$
$$f_2(\Omega) = ModelSize(\Omega)$$
$$f_3(\Omega) = \max_{i \in 1,\cdots,L} WorkingMemory_i(\Omega)$$

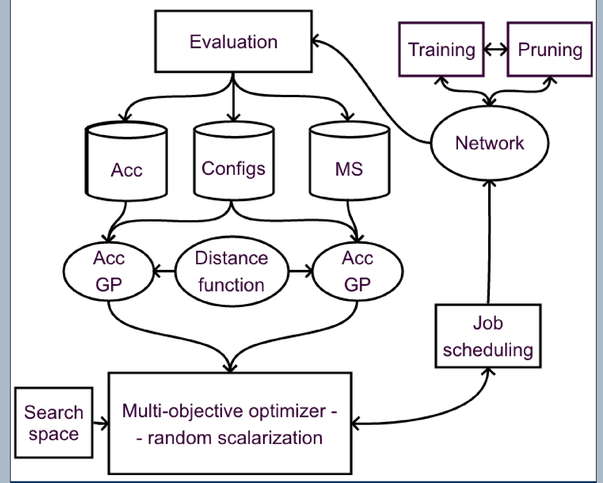
- We use analytical memory models:

$$ModelSize(\Omega) \approx \|\omega\|_0$$
$$WorkingMemory_l(\Omega) \approx \|x_l\|_0 + \|y_l\|_0$$

where x_l, y_l, w_l are layer inputs, outputs, and weights

- Unstructured/Structured pruning to reduce MS, WM
- Bayesian optimizer w/ random scalarization: $g(\Omega) = \max_{k \in 1,\cdots,K} \lambda_k f_k(\Omega)$
- Use morphisms to reduce evaluation cost
- Models run on Micro Bit and STM32F413 MCUs to evaluate hardware WM, MS, latency, and energy per inference

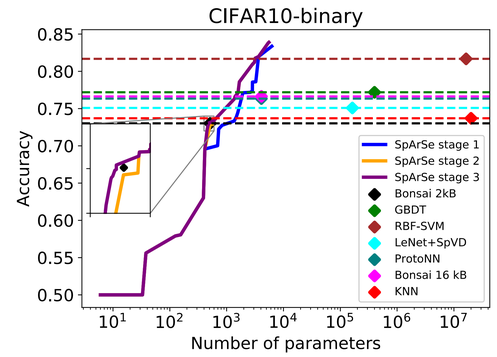
System overview

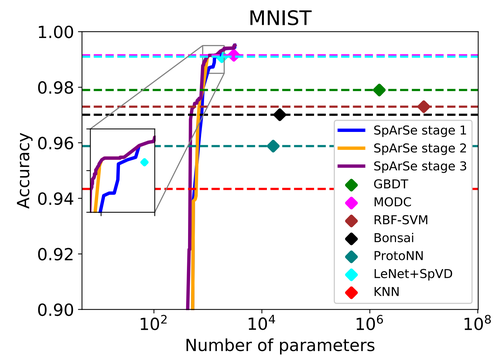


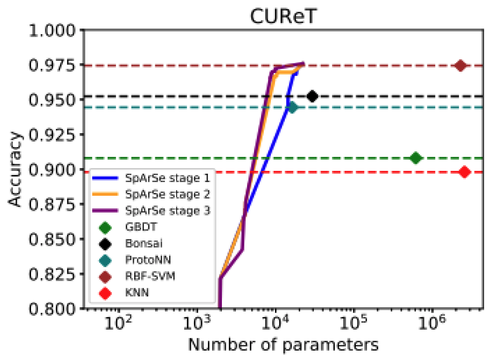
Search space

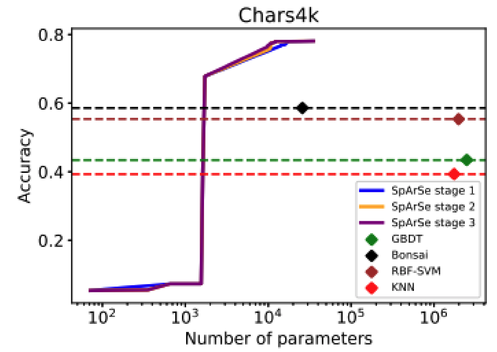
Name	Range	Description
downsample-input-in-depth	True/False	If True, max pool the input across the 3rd dimension
downsample-input	True/False	If True, max pool the input in spatial dimensions
input-downsampling-rate	[2 : 1 : 4]	Active only if downsample-input = True. The amount by which to downsample the input.
zero-regularization-epochs	[5:1:30]	Number of epochs for which VI inference is performed before the effect of the sparsity promoting prior is introduced.
annealing-epochs	[15:1:25]	Only active if pretraining=False. Number of epochs over which the coefficient in front of the regularization term in the VI objective is annealed from 0 to its final value
α	[1e-2:1e-2:1]	Final value for the coefficient of the regularization term in the VI objective
pretraining	True/False	Only active if pretraining=False. If True, pretrain the CNN before pruning
batch-norm	True/False	Only used for random weight pruning experiments. If True, apply batch-normalization to the output of each layer
num-conv-blocks	[1:1:2]	Number of convolution blocks in the CNN, where each block consists of a series of convolutional layers. The output of each block is downsampled through max pooling
num-fc-layers	[0:1:1]	Number of FC layers in the main branch following the convolution blocks
pruning-thresholds-block-k-layer-l	[-6:1e-1:3]	Thresholds for pruning weights in block k layer l
total-fc-layer-weights	[1:1:800]e3	Number of weights in the FC layers comprising the main, left, and right branches
weight-fraction-main-branch	[0:1]	Percentage of total-fc-layer-weights that go into the FC layer in the main branch
num-conv-layers-block-k	[1:1:3]	Number of convolutional layers in block k
layer-type-block-k-layer-l	[Conv2D, DownsampledConv2D, SeparableConv2D]	Layer type for convolutional block k layer l
kernel-size-block-k-layer-l	[2:1:5]	Convolutional kernel size of block k layer l
num-filters-block-k-layer-l	[1:1:100]	Number of output feature maps for block k layer l
downsample-block-k-layer-l	[0:0.5]	Active only if layer-type-block-k-layer-l=DownsampledConv2D. If True, the input feature maps are first passed through a $1 \times 1 \times (downsample - block - k - layer - l \times num - filters - block - k - layer - (l-1))$ convolutional layer
left-branch	True/False	If True, a branch is added to the feed-forward architecture. The branch takes the output of the first convolution block, sends it to an FC layer, sends the result to a merge operation, whose output is sent to a final FC layer
right-branch	True/False	If True, a branch is added to the feed-forward architecture. The branch takes the input to the first convolution block, sends it to an FC layer, sends the result to a merge operation, whose output is sent to a final FC layer
weight-fraction-left-branch	[0.01:1]	Active only if left-branch=True. Percentage of total-fc-layer-weights that go into the left branch FC layer
weight-fraction-left-branch	[0.01:1]	Active only if right-branch=True. Percentage of total-fc-layer-weights that go into the right branch FC layer
merge-type	Sum/Concatenate	Active only if at least one of left-branch or right-branch are True. How the main, left, and right branches are to be combined

Results: model size optimization









Results: model size & working memory optimization

Bonsai vs SpArSe for analytical MS, WM models (KB). For MNIST, SpArSe evaluated on ten-class dataset while Bonsai reports on two-class problem.

		Sparse	Sparse	Bonsai
MNIST	Acc	**97**	95.8	94.4[*]
	WM	1.38	**0.62**	<2
	MS	15	**1.76**	1.96
CIFAR10-binary	Acc	**73.66**	71.76	73
	WM	**1.13**	1.4	<2
	MS	3.95	**1.88**	1.98
CUReT-binary	Acc	**73.22**	**73.22**	
	WM	**1.9**	**1.9**	
	MS	**0.14**	**0.14**	
Chars4k-binary	Acc	**76.8**	74.9	74.71
	WM	**0.39**	1.64	<2
	MS	20.1	**0.16**	2
USPS-binary	Acc	**97.56**	96.21	94.42
	WM	1.81	**0.98**	<2
	MS	31.8	**1.48**	2

arm Research

References

Fedorov, I., et al. "SpArSe: Sparse Architecture Search for CNNs on Resource-constrained Microcontrollers." *Advances in Neural Information Processing Systems*. 2019.