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

Igor Fedorov – Arm ML Research
June 9, 2020
tinyML Talks Sponsors

EDGE IMPULSE

www.edgeimpulse.com

Additional Sponsorships available – contact Bette@tinyML.org for info
Next tinyML Talk

<table>
<thead>
<tr>
<th>Date</th>
<th>Presenter</th>
<th>Topic / Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuesday</td>
<td>Yung-Hsiang Lu Professor, Purdue University</td>
<td>Low-Power Computer Vision</td>
</tr>
<tr>
<td>June 16</td>
<td>Jon Tapson CSO, GrAI Matter Labs</td>
<td>Saving 95% of your edge power with Sparsity to enable tiny ML</td>
</tr>
</tbody>
</table>

Webcast start time is 8 am Pacific time
Each presentation is approximately 30 minutes in length

Please contact talks@tinyml.org if you are interested in presenting
Igor Fedorov is a member of the ARM Machine Learning Lab, working on neural network optimization for ARM hardware. His work covers network pruning, quantization, and architecture search methods. Prior to ARM, Igor completed a PhD in Electrical Engineering at the University of California San-Diego, working on Bayesian learning algorithms for signal processing.
SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers

Igor Fedorov
June 9, 2020
Arm ML Research Lab

- Pruning

- Quantization

- Noise-robust ML
  - C. Zhou et al., “Noisy Machines: Understanding Noisy Neural Networks and Enhancing Robustness to Analog Hardware Errors Using Distillation,” Arxiv ‘20

- RNN compression
  - U. Thakker et al., “Compressing rnns for iot devices by 15-38x using kronecker products,” Arxiv ’19

- Efficient HW for ML application
  - P. N. Whatmough et al., “FixyNN: Efficient Hardware for Mobile Computer Vision via Transfer Learning,” MLSys ’19

https://www.arm.com/resources/research/ml
SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers

Igor Fedorov  
Arm ML Research  
igor.fedorov@arm.com

Ryan P. Adams  
Princeton University  
rpadmiral@princeton.edu

Matthew Mattina  
Arm ML Research  
matthew.mattina@arm.com

Paul N. Whatmough  
Arm ML Research  
paul.whatmough@arm.com

R. Adams  
M. Mattina  
P. N. Whatmough

Abstract

The vast majority of processors in the world are actually microcontroller units (MCUs), which find widespread use performing simple control tasks in applications ranging from automobiles to medical devices and office equipment. The Internet of Things (IoT) promises to inject machine learning into many of these everyday objects via tiny, cheap MCUs. However, these resource-impoverished hardware platforms severely limit the complexity of machine learning models that can be deployed. For example, although convolutional neural networks (CNNs) achieve state-of-the-art results on many visual recognition tasks, CNN inference on MCUs is challenging due to severe memory limitations. To circumvent the memory challenge associated with CNNs, various alternatives have been proposed that do fit within the memory budget of an MCU, albeit at the cost of prediction accuracy. This paper challenges the idea that CNNs are not suitable for deployment on MCUs. We demonstrate that it is possible to automatically design CNNs which generalize well, while also being small enough to fit onto memory-limited MCUs. Our Sparse Architecture Search method combines neural architecture search with pruning in a single, unified approach, which learns superior models on four popular IoT datasets. The CNNs we find are more accurate and up to 7.4× smaller than previous approaches, while meeting the strict MCU working memory constraint.

1 Introduction

The microcontroller unit (MCU) is a truly ubiquitous computer. MCUs are self-contained single-chip processors which are small (~ 1.5in²), cheap (~ $1), and power efficient (~ 1μW). Applications are extremely broad, but often include seemingly banal tasks such as simple control and sequencing operations for everyday devices like washing machines, microwave ovens, and telephones. The key advantage of MCUs over application specific integrated circuits is that they are programmed with software and can be readily updated to fix bugs, change functionality, or add new features. The short time to market and flexibility of software has led to the staggering popularity of MCUs. In the developed world, a typical home is likely to have around four general purpose microprocessors. In contrast, the number of MCUs in is around three dozen [46]. A typical mid-range car may have about 30 MCUs. Public market estimates suggest that around 50 billion MCU chips will ship in 2019 [1], which far eclipses other chips like graphics processing units (GPUs), whose shipments totalled roughly 100 million units in 2018 [2].

MCUs can be highly resource constrained; Table 1 compares MCUs with bigger processors. The broad proliferation of MCUs relative to desktop GPUs and CPUs stems from the fact that they are

Overview

• Motivation
  • Pervasiveness and limitations of microcontroller units (MCUs)
  • Implications for CNNs on MCUs

• Black-box multi-objective optimization
  • Neural architecture search (NAS)
  • Pruning
  • Quantization
  • Lamarckian evolution and weight sharing

• Results
Motivation

• Around 50 billion MCU chips shipped in ‘19, compared to 100 million GPUs shipped in ‘18

• Challenges to deployment on MCUs: power consumption, latency, memory
  • Limited read-only (flash) memory
  • Limited RAM memory
  • Example: Arduino Uno has 2KB Flash and 2KB RAM

• CNNs are great at classification tasks, but:
  • Large model size
  • Large feature maps
  • LeNet requires: 420 KB flash & 391 KB RAM

• Some have proposed moving away from CNNs on MCUs
  • Pruned decision trees (Kumar et al., ICML ‘17)
  • Compressed KNN (Gupta et al., ICML ‘17)
CNN design as multi-objective optimization

- Leverage network optimization tools to find CNNs that satisfy HW constraints
- Neural architecture search
  - Depth, width, resolution, connectivity, operators
- Pruning
  - Random weight pruning
  - Channel/neuron pruning
- Quantization

\[
\begin{align*}
\min_{\Omega} & \quad f_1(\Omega) = 1 - \text{VALIDATION ACCURACY}(\Omega) \\
\text{subject to} & \quad f_2(\Omega) = \text{MODEL SIZE}(\omega) \\
& \quad f_3(\Omega) = \max_{l \in 1, \ldots, L} \text{WORKING MEMORY}_l(\Omega)
\end{align*}
\]

\[
\text{MODEL SIZE}(\omega) \approx \|\omega\|_0
\]

\[
\text{WORKING MEMORY}_1(\Omega) \approx \|x_l\|_0 + \|\omega_l\|_0
\]

\[
\text{WORKING MEMORY}_2(\Omega) \approx \|x_l\|_0 + \|y_l\|_0
\]

- Pareto optimality
Multi-objective optimization using random scalarizations

• We want to find the Pareto frontier for $f_1(x), \ldots, f_K(x)$

• Tchebychev scalarization: $\arg\min_x g(\lambda, x)$

  $$g(\lambda, x) = \max_{k \in [K]} \lambda_k f_k(x)$$

• Each choice of $\lambda$ corresponds to a different point on the frontier

• At each iteration, sample $\lambda$ randomly

• Preferences encoded by changing support of the distribution over $\lambda$

• In practice, we don’t know $f_k(x)$

• Use a Bayesian model!

System overview

- Accuracy
- Configs
- Number of parameters
- Accuracy Gaussian Process
- Distance function
- Parameters Gaussian Process
- Job scheduling
- Network
- Training
- Pruning
- Search space
- Multi-objective optimizer -- random scalarization

Technologies:
- ConfigSpace
- GPFlow
- Tensorflow
- Bayesian optimization + Hyperband (Pyro)
- Learnable
- Data
- Algorithm
Speeding up the search

• Network morphism
• Generate new proposals by slightly perturbing sampled configurations
• Inherit weights
• Less epochs
• 2-8x faster search

• Generate proposals only from the current Pareto frontier
### Search space

Table 7: Search space details. For discrete variables, ranges are listed in format [lower-bound:increment:upperbound].

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>downsample-input-in-depth</td>
<td>True/False</td>
<td>If True, max pool the input across the 3rd dimension</td>
</tr>
<tr>
<td>downsample-input</td>
<td>True/False</td>
<td>If True, max pool the input in spatial dimensions</td>
</tr>
<tr>
<td>input-downsampling-rate</td>
<td>[2 : 1 : 4]</td>
<td>Active only if downsample-input = True. The amount by which to downsample the input.</td>
</tr>
<tr>
<td>zero-regularization-epochs</td>
<td>[5:1:30]</td>
<td>Number of epochs for which VI inference is performed before the effect of the sparsity promoting prior is introduced.</td>
</tr>
<tr>
<td>annealing-epochs</td>
<td>[0:1:25]</td>
<td>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</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>[1e-2:1e-2:1]</td>
<td>Final value for the coefficient of the regularization term in the VI objective</td>
</tr>
<tr>
<td>pretraining</td>
<td>True/False</td>
<td>Only active if pretraining=False. If True, pretrain the CNN before pruning</td>
</tr>
<tr>
<td>batch-norm</td>
<td>True/False</td>
<td>Only used for random weight pruning experiments. If True, apply batch-normalization to the output of each layer</td>
</tr>
<tr>
<td>num-conv-blocks</td>
<td>[1:1:2]</td>
<td>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</td>
</tr>
<tr>
<td>num-fc-layers</td>
<td>[0:1:1]</td>
<td>Number of FC layers in the main branch following the convolution blocks</td>
</tr>
<tr>
<td>pruning-thresholds-block-k-layer-l</td>
<td>[1e-6:1e-1:3]</td>
<td>Thresholds for pruning weights in block $k$ layer $l$</td>
</tr>
<tr>
<td>total-fc-layer-weights</td>
<td>[1:1:800]e3</td>
<td>Number of weights in the FC layers comprising the main, left, and right branches</td>
</tr>
<tr>
<td>weight-fraction-main-branch</td>
<td>[0:1]</td>
<td>Percentage of total-fc-layer-weights that go into the FC layer in the main branch</td>
</tr>
<tr>
<td>num-conv-layers-block-k</td>
<td>[1:1:4]</td>
<td>Number of convolutional layers in block $k$</td>
</tr>
</tbody>
</table>
Models optimized for number of parameters

Figure 2: SpArSe results from minimization of \((1 - \text{VALIDATIONACCURACY}(\Omega)) \cdot \text{MODEL.SIZE}(\omega)\).
# Models optimized for number of parameters

Table 2: Dominating configurations for parameter minimization experiment. SpArSe models are listed on top and the competing method on bottom. SpArSe finds CNNs that are more accurate and have fewer parameters than competing methods. The amount of time spent obtaining each dominating configuration is reported in GPU days (GPUD).

<table>
<thead>
<tr>
<th></th>
<th>MNIST</th>
<th>CIFAR10-binary</th>
<th>CUReT</th>
<th>Chars4k</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc</td>
<td>( | |_3 )</td>
<td>GPUD</td>
<td>Acc</td>
</tr>
<tr>
<td>Bonsai</td>
<td>97.24</td>
<td>510</td>
<td>11</td>
<td>73.08</td>
</tr>
<tr>
<td></td>
<td>97.01</td>
<td>2.15e4</td>
<td>11</td>
<td>73.02</td>
</tr>
<tr>
<td>Bonsai (16 kB)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>76.66</td>
</tr>
<tr>
<td>ProtoNN</td>
<td>96.84</td>
<td>476</td>
<td>11</td>
<td>76.56</td>
</tr>
<tr>
<td></td>
<td>95.88</td>
<td>1.6e4</td>
<td>11</td>
<td>76.35</td>
</tr>
<tr>
<td>GBDT</td>
<td>98.78</td>
<td>804</td>
<td>11</td>
<td>77.90</td>
</tr>
<tr>
<td></td>
<td>97.90</td>
<td>1.5e6</td>
<td>11</td>
<td>77.19</td>
</tr>
<tr>
<td>kNN</td>
<td>96.84</td>
<td>476</td>
<td>11</td>
<td>76.34</td>
</tr>
<tr>
<td></td>
<td>94.34</td>
<td>4.71e7</td>
<td>11</td>
<td>73.70</td>
</tr>
<tr>
<td>RBF-SVM</td>
<td>97.42</td>
<td>569</td>
<td>10</td>
<td>81.77</td>
</tr>
<tr>
<td></td>
<td>97.30</td>
<td>1e7</td>
<td>10</td>
<td>81.68</td>
</tr>
<tr>
<td>LeNet + SpVD</td>
<td>99.16</td>
<td>1e3</td>
<td>8</td>
<td>75.35</td>
</tr>
<tr>
<td></td>
<td>99.10</td>
<td>1.8e3</td>
<td>8</td>
<td>75.09</td>
</tr>
<tr>
<td>MODC</td>
<td>99.17</td>
<td>1.45e3</td>
<td>1</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>99.15</td>
<td>3e3</td>
<td>1</td>
<td>–</td>
</tr>
</tbody>
</table>
Models optimized for total memory footprint

Table 3: Comparison of Bonsai with SpArSe for WM model (5). The first row shows the highest accuracy model for WM ≤ 2KB and the second row shows the highest accuracy model for WM, MS ≤ 2KB. For MNIST, SpArSe is evaluated on the full ten-class dataset whereas Bonsai reports on a reduced two-class problem. SpArSe finds models with smaller MS, less WM, and higher accuracy in all cases. WM, MS reported in KB. Best performance highlighted in bold.

<table>
<thead>
<tr>
<th></th>
<th>MNIST</th>
<th>CIFAR10-binary</th>
<th>CUREt-binary</th>
<th>Chars4K-binary</th>
<th>USPS-binary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc</td>
<td>WM</td>
<td>MS</td>
<td>GPUD</td>
<td>Acc</td>
</tr>
<tr>
<td>SpArSe</td>
<td>98.64</td>
<td>1.96</td>
<td>2.77</td>
<td>1</td>
<td>73.84</td>
</tr>
<tr>
<td>SpArSe</td>
<td>96.49</td>
<td>1.33</td>
<td>1.44</td>
<td>1</td>
<td>73.84</td>
</tr>
<tr>
<td>Bonsai</td>
<td>94.38*</td>
<td>&lt; 2</td>
<td>1.96</td>
<td></td>
<td>73.02</td>
</tr>
</tbody>
</table>

Table 4: SpArSe versus Bonsai for WM model (6). See Table 3 for details.

<table>
<thead>
<tr>
<th></th>
<th>MNIST</th>
<th>CIFAR10-binary</th>
<th>CUREt-binary</th>
<th>Chars4K-binary</th>
<th>USPS-binary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc</td>
<td>WM</td>
<td>MS</td>
<td>GPUD</td>
<td>Acc</td>
</tr>
<tr>
<td>SpArSe</td>
<td>97.03</td>
<td>1.38</td>
<td>15</td>
<td>1</td>
<td>73.66</td>
</tr>
<tr>
<td>SpArSe</td>
<td>95.76</td>
<td>0.62</td>
<td>1.76</td>
<td>2</td>
<td>71.76</td>
</tr>
<tr>
<td>Bonsai</td>
<td>94.38*</td>
<td>&lt; 2</td>
<td>1.96</td>
<td></td>
<td>73.02</td>
</tr>
</tbody>
</table>
Is it better to prune a large model or start with a small model? SpArSe can do both!

Figure 3: Fig. 3a shows the Pareto frontier of SpArSe with and without pruning, where both experiments sample the same number (325) of configurations. Fig. 3b-3c show scatter plots of $|E|$ versus $\|\tilde{\omega}\|_0$ for the best performing configurations from the experiment in Section 4.1. Fig. 3b: MNIST networks with $>95\%$ accuracy. Fig. 3c: CIFAR10-binary networks with $>70\%$ accuracy.
Ablation study

Table 5: Ablation on MNIST using WM model (6), searching for models with WM,MS ≤ 2KB on 250 configuration budget. SpArSe w/o pruning did not yield a model that satisfies the constraints.

<table>
<thead>
<tr>
<th></th>
<th>SpArSe</th>
<th>SpArSe w/o pruning</th>
<th>SpArSe w/ product scalarization</th>
<th>SpArSe w/o morphism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc</td>
<td>95.76</td>
<td>–</td>
<td>11.35</td>
<td>97.46</td>
</tr>
<tr>
<td>WM</td>
<td>0.62</td>
<td>–</td>
<td>0.01</td>
<td>0.68</td>
</tr>
<tr>
<td>MS</td>
<td>1.76</td>
<td>–</td>
<td>0.05</td>
<td>1.31</td>
</tr>
<tr>
<td>GPUD</td>
<td>2</td>
<td>–</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Latency and power measurements

Bonsai figures not directly comparable because it operates on features, not raw data

A recent related work, MODC [25], is considerably slower than SpArSe, at 684 ms for MNIST on the Arduino Uno

Although it may be too early to say if CNN latency/power consumption can meet application requirements, we hope this work provides much needed data to start to answer this question.

Table 6: Measurement of SpArSe models on Micro Bit and STM MCUs, compared with Bonsai on Arduino Uno. Latency in ms.

<table>
<thead>
<tr>
<th>Model</th>
<th>MNIST</th>
<th>CIFAR10-binary</th>
<th>CUREF-binary</th>
<th>Chars4k-binary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc</td>
<td>WM</td>
<td>Lat. μBit</td>
<td>Nulled μBit</td>
</tr>
<tr>
<td>SpArSe</td>
<td>96.97</td>
<td>1.32</td>
<td>15.86</td>
<td>285.82</td>
</tr>
<tr>
<td>SpArSe</td>
<td>95.76</td>
<td>0.71</td>
<td>2.35</td>
<td>115.40</td>
</tr>
<tr>
<td>Bonsai</td>
<td>94.38</td>
<td>&lt; 2</td>
<td>1.96</td>
<td>2.18</td>
</tr>
</tbody>
</table>
Conclusion

• While ubiquitous, MCUs have been largely ignored by ML researchers
• We target MCUs for deployment of ML, enabling future IoT products and use cases
• Contrary to previous assertions, it is in fact possible to design CNNs for MCUs with as little as 2KB RAM
• SpArSe optimizes CNNs for the multiple constraints of MCU hardware platforms, finding models that are both smaller and more accurate than previous SOTA non-CNN models across a range of standard datasets
• Thanks to Michael Bartling, Patrick Hansen, and Neil Tan for their help in model deployment.
Copyright Notice

This presentation in this publication was presented as a tinyML® Talks webcast. The content reflects the opinion of the author(s) and their respective companies. The inclusion of presentations in this publication does not constitute an endorsement by tinyML Foundation or the sponsors.

There is no copyright protection claimed by this publication. However, each presentation is the work of the authors and their respective companies and may contain copyrighted material. As such, it is strongly encouraged that any use reflect proper acknowledgement to the appropriate source. Any questions regarding the use of any materials presented should be directed to the author(s) or their companies.

tinyML is a registered trademark of the tinyML Foundation.

www.tinyML.org