“A hardware-aware neural architecture search algorithm targeting ultra-low-power microcontrollers”

Andrea Mattia Garavagno – Sant'Anna School of Advanced Studies of Pisa and University of Genoa

August 29, 2023
Thank you, tinyML Strategic Partners, for committing to take tinyML to the next Level, together
Executive Strategic Partners
Advancing AI research to make efficient AI ubiquitous

Power efficiency
- Model design, compression, quantization, algorithms, efficient hardware, software tool

Personalization
- Continuous learning, contextual, always-on, privacy-preserved, distributed learning

Efficient learning
- Robust learning through minimal data, unsupervised learning, on-device learning

Perception
- Object detection, speech recognition, contextual fusion

Reasoning
- Scene understanding, language understanding, behavior prediction

Action
- Reinforcement learning for decision making

A platform to scale AI across the industry

Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.
Accelerate Your Edge Compute

SYNTIANT

Making Edge AI A Reality

www.syntiant.com
Platinum Strategic Partners
DEPLOY VISION AI
AT THE EDGE AT SCALE
Gold Strategic Partners
Witness potential made possible at analog.com.

Where what if becomes what is.
Build the Future of tinyML on ARM
EDGE IMPULSE

The Leading Development Platform for Edge ML

edgeimpulse.com
Driving decarbonization and digitalization. Together.

Infineon serving all target markets as Leader in Power Systems and IoT

www.infineon.com
NEUROMORPHIC INTELLIGENCE FOR THE SENSOR-EDGE

www.innatera.com
Renesas is enabling the next generation of AI-powered solutions that will revolutionize every industry sector.
STMicroelectronics provides extensive solutions to make tiny Machine Learning easy
ENGINEERING
EXCEPTIONAL
EXPERIENCES

We engineer exceptional experiences for consumers in the home, at work, in the car, or on the go.

www.synaptics.com
Join Growing tinyML Communities:

tinyML - Enabling ultra-low Power ML at the Edge

The tinyML Community
https://www.linkedin.com/groups/13694488/

16.5k members in
49 Groups in 41 Countries

4k members &
12.7k followers
Subscribe to tinyML YouTube Channel for updates and notifications (including this video)
www.youtube.com/tinyML
tinyML Asia
Technical Forum

November 16, 2023
Seoul, South Korea

Call for Presentations and Posters – Deadline August 7
https://www.tinyml.org/event/asia-2023/
2023 Edge AI Technology Report

The guide to understanding the state of the art in hardware & software in Edge AI.

https://www.wevolver.com/article/2023-edge-ai-technology-report
Reminders

Slides & Videos will be posted tomorrow

tinyml.org/forums  youtube.com/tinyml

Please use the Q&A window for your questions
Andrea Mattia Garavagno was born in Rome (Italy) in 1996. He received his BSc in Electronic Engineering from the University of Genoa, and the MSc in Embedded Computing Systems from Scuola Superiore Sant’Anna and the University of Pisa, Italy. He is currently a PhD student at the Scuola Superiore Sant’Anna and the University of Genoa. Together with Giuliano Donzellini e Luca Oneto, he co-authored the Italian book "Introduzione al Progetto di Sistemi a Microprocessore", and the international book “Introduction to Microprocessor-Based Systems Design” published by Springer in 2021 and 2022. Currently he's working on hardware-aware neural architecture search targeting microcontrollers.
A hardware-aware neural architecture search algorithm targeting low-end microcontrollers

Andrea Mattia Garavagno

Email: AndreaMattia.Garavagno@{edu.unige.it, santannapisa.it}

Department of Electrical, Electronic, Telecommunication Engineering and Naval Architecture, DITEN, University of Genoa, Genoa 16145, Italy

Department of Excellence of Robotics and AI, Institute of Mechanical Intelligence, Scuola Superiore Sant’Anna, Pisa 56124, Italy
The aim

• Bring **convolutional neural networks (CNNs)** to **low-end microcontrollers** units (MCUs)

High-end Microcontroller:
• Thousand-ish CoreMark score
• Thousands of kB of RAM
• Multiple cores

Low-end Microcontroller:
• Tens-ish CoreMark score
• Tens of KB of RAM
• Just one core
The problem

• It’s **not so easy** to design **CNN** able to fit the constraints of **low-end MCUs**
• Typically, people involved in software for low-end MCUs are not confident in the machine learning (ML) domain
• It would be useful to have an **automatic** way to **design CNN**
A possible solution

- Hardware-aware Neural Architecture Search (HW NAS)
  - a technique for automating the design of artificial neural networks (ANNs), in our case CNNs, taking into consideration hardware constraints

- As of today:
  - Gives state-of-the-art results in several Tiny-ML benchmarks
  - Targets high-performance MCUs
  - Requires from 200 to 40,000 GPU hours

<table>
<thead>
<tr>
<th>Model</th>
<th>GPU hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCUNet</td>
<td>300</td>
</tr>
<tr>
<td>ProxylessNAS</td>
<td>200</td>
</tr>
<tr>
<td>MNASNET</td>
<td>40,000</td>
</tr>
</tbody>
</table>
The reasons behind a so high search cost

• **Huge search spaces** which contains few good candidate solutions able to perform well on MCUs
• **Long evaluation methods** of candidate solutions which often imply a complete training of each architecture
• **Computationally intensive search strategies** which often requires the computation of a huge number of derivatives or the usage reinforcement learning or gradient descent methods
Our solution

• Does not require any GPU to obtain results in an acceptable amount of time

• Targets low-end MCUs

• Achieves state-of-the-art results on the Visual Wake Word dataset, in just 3:37 hours on a laptop mounting an 11th Gen Intel(R) Core(TM) i7-11370H CPU @ 3.30GHz equipped with 16 GB of RAM and 512 GB of SSD, without using a GPU
How

• A refined search space, crafted explicitly for occupying few RAM while providing acceptable performances on low-end microcontrollers, reduces the number of candidate solutions

• A novel derivative-free search strategy, inspired by Occam’s razor, which starts from the smallest admissible solution and tries to generate larger candidates until the evaluation score increases, avoiding unnecessary multiplication of resources

• A fast evaluation method, based on an extremized version of the early stopping criterion, avoids spending a lot of time in the training of candidates
Refined search space

• The proposed search space is built by staking cells composed of fixed architectural elements (yellow dashed lines) upon a pre-processing pipeline (green dashed lines). The number of kernels, $k$, used in the first convolutional layer (red dashed lines) sets the number of kernels used in the cells according to the following equation.

$$n_c = \begin{cases} 
  k & \text{if } c = 0 \\
  \left[ (2 - \sum_{i=1}^{c-1} 2^{-i}) \cdot n_{c-1} \right] & \text{if } c \geq 1 
\end{cases}$$  \hspace{1cm} (1)

• Candidate architectures can be conveniently represented by the tuple $(k, c)$ where $k$ is the number of kernels used in the first convolutional layer and $c$ is the number of cells used by the architecture.
Search strategy

- The proposed search strategy starts with the lowest number of kernels (k=1) and searches for the best number of cells to stake (c), starting from zero (c=0). Then, it repeats itself, trying with larger values of k until the performance of the network found continues to increase. Doing so, resources are only added when the performance increases, thus respecting Occam’s razor (entities should not multiplied beyond necessity).

**Algorithm 1** search strategy pseudocode

```plaintext
Algorithm 1 search strategy pseudocode

k ← 1  ▷ Minimum number of kernels of the first layer
c ← 0  ▷ No cells added

while (k, c) is feasible and f(k, c) increases do
  c ← 0 ▷ Reset cells
  while (k, c) is feasible and f(k, c) increases do
    c ← c + 1 ▷ Try with one more cell
  end while
  k ← k + 1 ▷ Try with more kernels
end while

return (k, c) : max f(k, c)
```
Search strategy - example

• We start with (k=1) and search for the best number of cells to stake (c), starting from zero (c=0)
• Note that (k=1,c=0) is the smallest feasible solution
Search strategy - example

- We try to add one cell (c=1)
- We find out that this solution is better than the previous one, so we mark it with a green arrow (the evaluation phase will be discussed later)
Search strategy - example

- We try one more cell (c=2)
- For another time, we find a better solution for (k=1), hence we put another green arrow
Search strategy - example

- We continue adding cells (c=3)
- This time we find that the new candidate performs worse than the previous one, so we put a red arrow
- According to Occam’s razor, we must stop here to avoid unnecessary multiplications of resources
Search strategy - example

• We found out that (c=2) is the best solution for (k=1)
Search strategy - example

• Now we repeat the same process with (k=2)
Search strategy - example

• Now we repeat the same process with (k=2)
Search strategy - example

- Now we repeat the same process with \( k=2 \)
Search strategy - example

• Now we repeat the same process with \((k=2)\)
Search strategy - example

• Now we repeat the same process with \((k=2)\)
Search strategy - example

- This time we reached the maximum number of cells that can be staked (no more pixels to process) without having a performance degradation, so we stopped there.
- \((c=5)\) is the best solution for \((k=2)\).
Search strategy - example

• We find out that \((k=2, c=5)\) is better than \((k=1, c=2)\), so we proceed with \((k=3)\)
Search strategy - example

• We continue until we find better solutions
Search strategy - example

• We find that (k=10,c=4) is worse than (k=9,c=3) so, according to Occam’s razor, we stop there.
• The solution found is (k=9,c=3).
• Notice that we could also stop because of resource completion.
Search strategy – in short

- It is a sort of directional search method, inspired by Occam’s razor
- The c direction is explored in the inner loop, while the k direction is in the outer one
- Not requiring derivatives allows for a faster search

**Algorithm 1** search strategy pseudocode

```plaintext
k ← 1  ▶ Minimum number of kernels of the first layer
c ← 0  ▶ No cells added

while (k, c) is feasible and \( f(k, c) \) increases do
  c ← 0  ▶ Reset cells
  while (k, c) is feasible and \( f(k, c) \) increases do
    c ← c + 1  ▶ Try with one more cell
  end while
  k ← k + 1  ▶ Try with more kernels
end while

return (k, c) : max f(k, c)
```
Evaluation strategy

- Now let’s talk about how we pick the best model between two
- Candidates are evaluated by applying an extremized version of the early stopping criterion
  - Each candidate is trained for just three epochs
  - The best validation accuracy obtained during these epochs is used to pick the best candidate between two

How good is extremizing the early stopping criterion?
How good is extremizing early stopping?

Let’s compare it with a coin. On the left, we can see the probability of guessing the best performant model between two in the search space, using early stopping until epoch $n$. On the right, the same probability using a coin to decide which is the best model.
How good is extremizing early stopping?

Let’s compare it with a coin. On the left, we can see the probability of guessing the best performant model between two in the search space, using early stopping until epoch \( n \). On the right, the same probability using a coin to decide which is the best model.
To take away

Extremizing the early stopping criterion

- allows for a drastic reduction in the search cost, enabling GPU-less HW NAS
- Reduces the search’s precision and repeatability...
- ...but is consistently better than random guessing
ColabNAS

• Another HW NAS targeting low-end MCUs
• It can be run on free GPU programs like Google’s Colaboratory and Kaggle Kernel
• It is more repeatable than this NAS...
• ...but it still requires a GPU (even if you don’t have to own it)
Summing up

• We use:
  • a refined search space, crafted explicitly for occupying few RAM while providing acceptable performances on low-end microcontrollers, which reduces the number of candidate solutions
  • a novel derivative-free search strategy, inspired by Occam’s razor, which starts from the smallest admissible solution and tries to generate larger candidates until the evaluation score increases, avoiding unnecessary resource usage
  • a fast evaluation method, based on an extremized version of the early stopping criterion, which avoids spending a lot of time in the training of candidates
Hardware-awareness

• We evaluated our algorithm on three STM32 Ultra Low Power MCUs
• We used the Visual Wake Words datasets
• We set the resolution at 50x50 rgb

<table>
<thead>
<tr>
<th>STM32 MCU</th>
<th>RAM</th>
<th>Flash</th>
<th>CoreMark</th>
</tr>
</thead>
<tbody>
<tr>
<td>L010RBT6</td>
<td>20 kiB</td>
<td>128 kiB</td>
<td>75</td>
</tr>
<tr>
<td>L151UCY6DTR</td>
<td>32 kiB</td>
<td>256 kiB</td>
<td>93</td>
</tr>
<tr>
<td>L412KBU3</td>
<td>40 kiB</td>
<td>128 kiB</td>
<td>273</td>
</tr>
</tbody>
</table>
Hardware-awareness

For reference:

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>RAM occupancy</th>
<th>FLASH occupancy</th>
<th>Search Cost</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>vww_l010r6t6</td>
<td>72.3%</td>
<td>20 kiB</td>
<td>10.66 kiB</td>
<td>1:50h</td>
<td>no</td>
</tr>
<tr>
<td>vww_l151ucy6dt</td>
<td>74.6%</td>
<td>26 kiB</td>
<td>19.73 kiB</td>
<td>2:01h</td>
<td>no</td>
</tr>
<tr>
<td>vww_l412kbu3</td>
<td>77.2%</td>
<td>31 kiB</td>
<td>28.48 kiB</td>
<td>3:53h</td>
<td>no</td>
</tr>
</tbody>
</table>
Performance comparison

• We compare our method with **MCUNet** (MIT) and **Micronets** (ARM) projects, two HW NAS offering **state-of-the-art** results for the **Visual Wake Words** dataset.

• They both **target** high-end MCUs of **STM’s high-performance** series.

• Given our target, which is low-end microcontrollers, we selected the largest target among the **lightest** of the two projects, and we ran the proposed algorithm on it.
Performance comparison

**Test Accuracy**
- MCUNet: 87.4%
- Micronets: 76.8%
- Ours: 77%

**Flash occupancy - TFLite Micro**
- MCUNet: 530.52 kiB
- Micronets: 273.81 kiB
- Ours: 23.65 kiB

**RAM occupancy - TFLite Micro**
- MCUNet: 168.5 kiB
- Micronets: 70.50 kiB
- Ours: 28.50 kiB

**MACC**
- MCUNet: 6 MM
- Micronets: 3.3 MM
- Ours: 1.4 MM
input_shape = (50,50,3)

#The path must point to a folder containing the dataset
#organised in subfolders, one for each class
path_to_training_set = './datasets/melanoma_cancer_dataset/train'
val_split = 0.3
path_to_test_set = './datasets/melanoma_cancer_dataset/test'

#whether or not to cache datasets in memory
#if the dataset cannot fit in the main memory, the application will crash
cache = True

#target: STM32L412KBU3
#273 CoreMark, 40 kiB RAM, 128 kiB Flash
ram_upper_bound = 40960
flash_upper_bound = 131072
MACC_upper_bound = 2730000 #CoreMark * 1e4

nanoNAS = NanoNAS(ram_upper_bound, flash_upper_bound, MACC_upper_bound,
                   path_to_training_set, val_split, cache, input_shape, save_path='./results')

#search
nanoNAS.search(save_search_history=False)

#train resulting architecture
nanoNAS.train(training_epochs=100, training_learning_rate=0.01, training_batch_size=128)

#apply uint8 post training quantization
nanoNAS.apply_uint8_post_training_quantization()

#evaluate post training quantization
nanoNAS.test_keras_model(path_to_test_set)
nanoNAS.test_tflite_model(path_to_test_set)
Conclusion

• It’s an easy way to obtain CNNs for low-end MCUs
  • does not require a GPU to obtain results in a reasonable amount of time
  • It achieves state-of-the-art performances on the Visual Wake Words dataset, a standard TinyML benchmark

• We hope it can foster the usage of HW NAS for the developing of IoT and wearable devices
Future works

• We’re working on a smaller implementation able to run on embedded devices

• It could preserve privacy by allowing the design of CNNs on the device itself
Thank you for the attention

Download and try it!
Copyright Notice

This multimedia file is copyright © 2023 by tinyML Foundation. All rights reserved. It may not be duplicated or distributed in any form without prior written approval.

tinyML® is a registered trademark of the tinyML Foundation.

www.tinyml.org
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.