Tiny decomposition of complex neural networks for heterogeneous microcontrollers

Biagio Montaruli, Andrea Santamaria, Danilo Pau
STMicroelectronics

Introduction

The deployment of neural network (NN) models on low-power and resource-constrained devices represents a critical bottleneck in the development of intelligent and autonomous Internet of Things (IoT) systems due to their strict memory and computing capabilities.

This problem has been addressed by the tinyML research and industry communities using different approximation and optimization techniques, such as parameters pruning and sharing, quantization and knowledge distillation. Unfortunately, these approaches often require to re-design or modify the models' topology, which implies significant effort and time, and may lead to a reduction in terms of accuracy.

To overcome those issues, we propose a novel design methodology based on a distributed approach, which aims at automatically partitioning the execution of a NN over multiple heterogeneous tiny devices (see Figure 1).
Such a methodology is formalized as an optimization problem considering two objectives:

- minimize the inference latency, which is the total time to perform a single inference and it takes into account both the communication and computation time.
- maximize the throughput, which represents the inverse of the waiting time between two consecutive inferences.

The proposed work has been evaluated over different NN architectures and microcontrollers (MCUs) using two algorithms: Full Search (FS) and Dichotomic Search (DS).

The implementation has been carried out with X-CUBE-AI v7.1.0 [1] in order to profile the chosen NN models, as well as to automatically deploy and validate the obtained sub-models on the target devices.

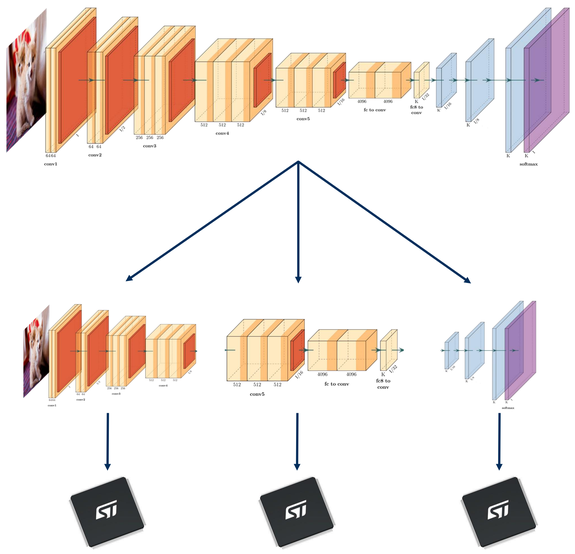

Figure 1

Full Search

It explores all the candidate solutions, one after the other, by checking at each step whether the current candidate is feasible and is better than the best solution found so far. If so, it updates the current best solution with that candidate.
PROS: it always finds the best solution
CONS: time complexity is exponential in the number of layers

Algorithm 1 Full Search Algorithm

Input: N, D, f
Output: optimal $P_{\overline{p}}$
Initialization: $P_{\overline{p}} = \emptyset$

for $p \in [1, d^n]$ **do**
  $P_p \leftarrow new\_configuration(p, N, D)$
  if $check\_feasibility(P_p)$ **then**
    if $better\_solution(P_p, P_{\overline{p}}, f)$ **then**
      $P_{\overline{p}} \leftarrow P_p$
    end if
  end if
end for
return $P_{\overline{p}}$

Dichotomic Search

It is a recursive algorithm that produces a bisection tree, which is explored in a depth-first search (DFS) fashion.
Moreover, for each new candidate, the DS checks whether it is feasible and better than the best solution found so far.
PROS: time complexity is linear in the number of layers
CONS: by construction, finding the optimum is not guaranteed

Algorithm 2 Dichotomic Search Algorithm

Input: N, D, f
Output: $optimal P_{\overline{p}}$
Initialization: $P_{\overline{p}} = \emptyset$

for $dev \in D$ **do**
  $P_p \leftarrow initial\_configuration(dev, N)$
  $recursion(P_p, P_{\overline{p}}, f)$
end for
return $P_{\overline{p}}$

Algorithm 3 recursion pseudo-code

Input: $P_p, P_{\overline{p}}, f$

if $check\_feasibility(P_p)$ **then**
  if $better\_solution(P_p, P_{\overline{p}}, f)$ **then**
    $P_{\overline{p}} \leftarrow P_p$
  end if
end if
if $is\_leaf(P_p)$ **then**
  return
else
  $C \leftarrow create\_child\_nodes(P_p)$
  for $c \in C$ **do**
    $recursion(c, P_{\overline{p}}, f)$
  end for
end if
return

Experimental evaluation

To evaluate the FS and DS algorithms a detailed experimental campaign has been carried out considering seven NN models (see Table 1) and nine STM32 MCUs (see Table 2) characterized by heterogeneous memory and computational properties.

As for the NN models, we used three MobileNets v1 [2] trained with different values for the α parameter (0.25, 0.30 and 0.35), YAMNet 256, which is a modified version of the original YAMNet model [3] obtained by taking the first 6 convolutional blocks, a proprietary Convolutional Neural Network trained on the VoxCeleb dataset [4], as well as the CNN and KWS-CNN models for keyword spotting presented in [5].

Obtained results, summarized in Table 3, show that, the DS algorithm achieved the best results in terms of computational complexity in all cases. However, as for the YAMNet 256, it was not able to find the optimal solution.

Model	NN depth	FLASH size (KB)	RAM size (KB)	MACC (10^6)
MobileNet 025	30	1825.53	262.06	14.4
MobileNet 030	30	2366.15	311.32	19.6
MobileNet 035	30	2976.80	360.34	26.0
YAMNet 256	13	526.25	396.25	24.4
VoxCeleb	7	926.84	39.75	12.1
KWS CNN	8	270.91	31.21	2.53
KWS DS-CNN	17	155.80	56.25	4.83

Table 1

MCU	FLASH (KB)	RAM (KB)	CPU Freq. (MHz)	CpM
STM32H743ZI	2048	1024	480	6
STM32H723ZG	1024	564	550	6
STM32F446RE	512	128	180	9
STM32F401RE	512	96	84	9
STM32F401RB	128	64	84	9
STM32L4R5ZI	2048	640	120	9
STM32L452RE	512	128	80	9
STM32L433RC	256	64	80	9
STM32L412KB	128	40	80	9

Table 2

Model	MCUs	Latency (s)	Throughput (s^{-1})	Num sub-models	FS steps	DS steps
MobileNet v1 025	STM32H743ZI, STM32L4R5ZI	0.268	4.034	2	2^{30}	236
MobileNet v1 030	STM32H743ZI, STM32F401RE	1.839	0.544	3	2^{30}	236
MobileNet v1 035	STM32H743ZI, STM32L4R5ZI	0.448	2.379	2	2^{30}	236
YAMNet 256	STM32H743ZI, STM32L4R5ZI	4.331	0.278	2	2^{13}	100
VoxCeleb	STM32L452RE, STM32F446RE	0.684	1.492	2	2^{7}	52
VoxCeleb	STM32F446RE, STM32H723ZG	0.208	4.955	2	2^{7}	52
KWS CNN	STM32L433RC, STM32L412KB	0.822	1.216	3	2^{8}	46
KWS DS-CNN	STM32F401RB, STM32F401RB	2.74	0.431	2	2^{17}	132

Table 3

Implementation

On-device deployment of MobileNet v1 030 (shown in Figure 2) and validation using X-CUBE-AI (see Figure 3).

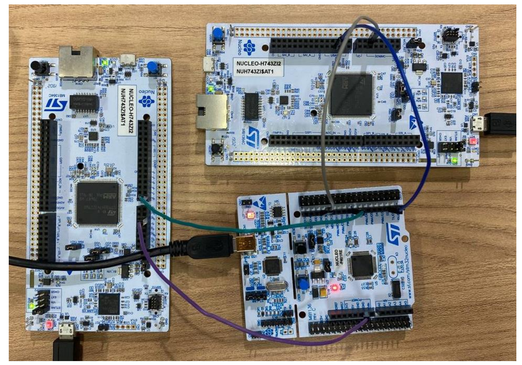

Figure 2

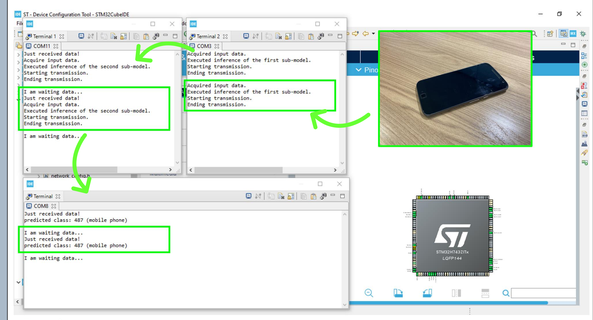

Figure 3

Conclusions

The proposed methodology has been evaluated on several NN models and MCUs using two different algorithms, FS and DS, whose involve a trade-off between optimality of the solution and computational complexity.

As future works, we plan to extend the experimental campaign with additional state-of-the-art NN models, as well as to implement algorithms that achieve even lower computational complexity regarding DS and find the optimal solution as FS.

References

[1] X-CUBE-AI - AI expansion pack for STM32CubeMX – STMicroelectronics. Available online: www.st.com/en/embedded-software/x-cube-ai.html
[2] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications“, 2017.
[3] YAMNet source code and tutorial. Available online: www.tensorflow.org/hub/tutorials/yamnet.
[4] Arsha Nagrani et al. “VoxCeleb: a large-scale speaker identification dataset”, 2017.
[5] Yundong Zhang et al. “Hello Edge: Keyword Spotting on Microcontrollers”, 2017.