Confirm
Smart Manufacturing

Confirm
Smart Manufacturing

Enabling Machine Learning on the Edge using SRAM Conserving Efficient Neural Networks Execution Approach

Bharath Sudharsan

A World Leading SFI Research Centre

Science Foundation Ireland sfi For what's next

University of Limerick, Ollscoil Luimnigh
Tyndall National Institute
UCC University College Cork, Ireland, Coláiste na hOllscoile Corcaigh
NUI Maynooth, Ollscoil na hÉireann Má Nuad
OÉ Gaillimh NUI Galway
CIT Cork Institute of Technology, Institiúid Teicneolaíochta Chorcaí
AIT
LIT

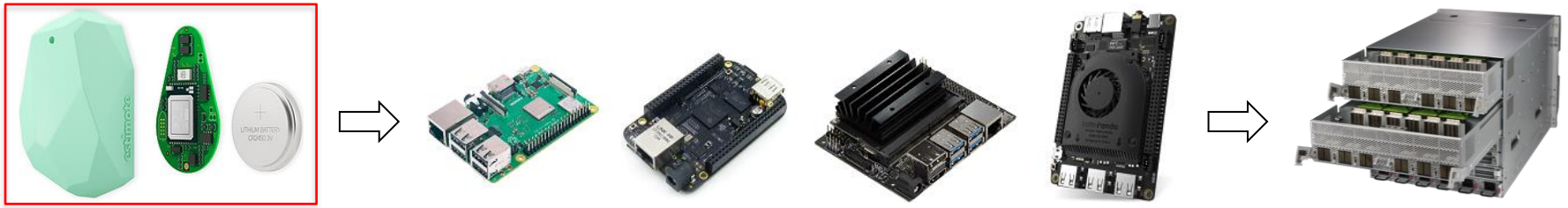
Edge Computing - Hardware View

Confirm
Smart Manufacturing

ARM Cortex-M0 MCU based BLE beacon

Powerful CPU + basic GPU based SBCs (single board computers)

Edge gateway with GPUs and SSDs



Edge computing hardware: highly resource constrained -> high resource (left to right)

- MCUs and small CPUs: BLE beacons, smart bulbs, smart plugs, TV remotes, fitness bands
- SBCs: Raspberry Pis, BeagleBones, NVIDIA Jetsons, Latte Pandas, Intel NUCs, Google Coral
- GPU accelerated: AWS snowball, Digi gateways, Dell Edge Gateways for IoT, HPE Edgeline

- Why MCU? Roughly **50 billion** MCU chips were shipped in 2020 (market estimates), which far exceeds other chips like GPUs & CPUs (only 100 million units sold)

Edge Computing - Hardware Spec

Confirm
Smart Manufacturing

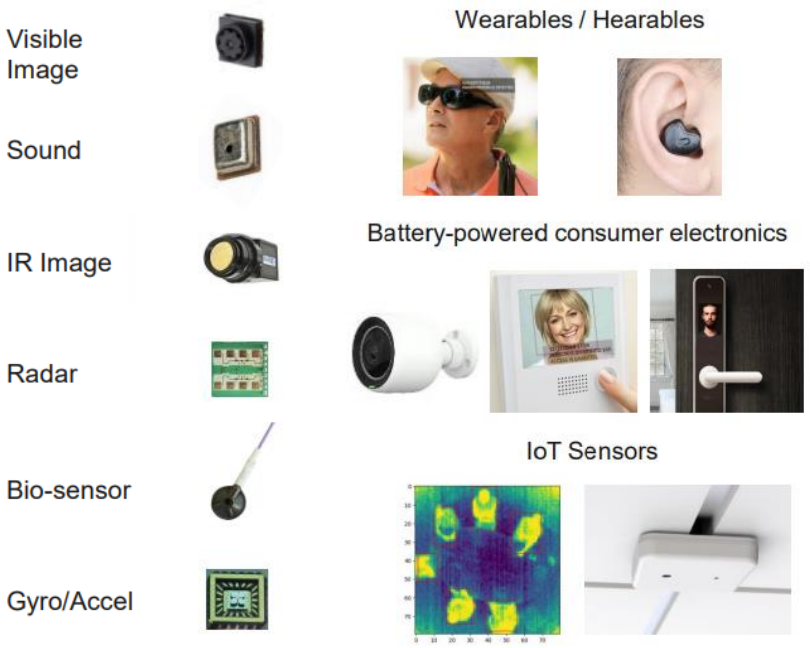
Board	MCU & Board Name	Specification					
		Bits	EEP ROM	SRAM	Flash	Clock (MHz)	FP
#1 #2	ATmega328P Arduino Pro Mini, Nano	8	1kB	2kB	32kB	16	✗
#3	nRF52840 Adafruit Feather	32	-	256kB	1MB	64	✓
#4	STM32f103c8 Blue Pill	32	-	20kB	128kB	72	✗
#5 #6	Adafruit HUZZAH32, Generic ESP32	32	-	520kB	4MB	240	✓
#7	ATSAMD21G18 Adafruit METRO	32	-	32kB	256kB	48	✗
#8	ATmega2560 Arduino Mega	8	4kB	8kB	256kB	16	✗

Specification of popular MCU boards

- No file system
- Lack Python support. Only C, C++
- Only few hundred MHz clock speed and low SRAM and Flash memory
- Lack multiple cores. No parallel execution units
- Lack inbuilt hardware accelerators such as APU (Accelerated Processing Unit), KPU (convolution operation accelerator), FPU (Floating-point accelerator), and FFT (Fourier transform accelerator)

Neural Networks for Edge Computing

Confirm
Smart Manufacturing



Who are practicing TinyML

- Running NNs on MCUs is ultra-low-power machine learning, also known as TinyML

  ✓ **Audio** - always-on inference. **Industry telemetry** - monitor, detect anomalies in motor bearing vibrations. **Image** - object counting, visual wake words. **Physiological/behavior** - activity recognition using IMU or EMG

NNs on MCUs - Prevailing Challenges

Confirm
Smart Manufacturing

SRAM Overflow Challenge

- To deploy, when taking maximum deep compressed NN models
  - ✓ Can exceed the device's memory capacity just by a few bytes SRAM margin
  - ✓ Cannot apply additionally optimization as already max optimized
- Core algorithm of proposed SRAM optimized approach - makes such max compressed NNs consume **less SRAM during execution**

IoT Device Fail Challenge

- Devices running large NNs fail - cases reported
  - ✓ Due to overheating, fast battery wear, and run-time stalling
  - ✓ Prime reason for such failure is the exhaustion of device SRAM memory
- Proposed Tensor Memory Mapping (TMM) - **accurately** compute + visualize tensor memory requirement of each operator in NN computation graph during execution

Efficient NN Execution - Proposed Design

Confirm
Smart Manufacturing

- The trained model size and its peak SRAM need to be highly reduced due to the limited Flash and SRAM memory capacity of IoT devices or TinyML hardware

- Our approach can reduce the peak SRAM consumed by NNs, speedup inference, and save energy during model execution. Upcoming slides presents the parts of proposed approach

    i. Tensor Memory Mapping (TMM) method

    ii. Load fewer tensors and tensors re-usage

    iii. Cheapest NN graph execution sequence

    iv. Finally, the core algorithm that combines above three parts and presents the complete approach in the form of an implementable algorithm

Tensor Memory Mapping (TMM)

Confirm
Smart Manufacturing

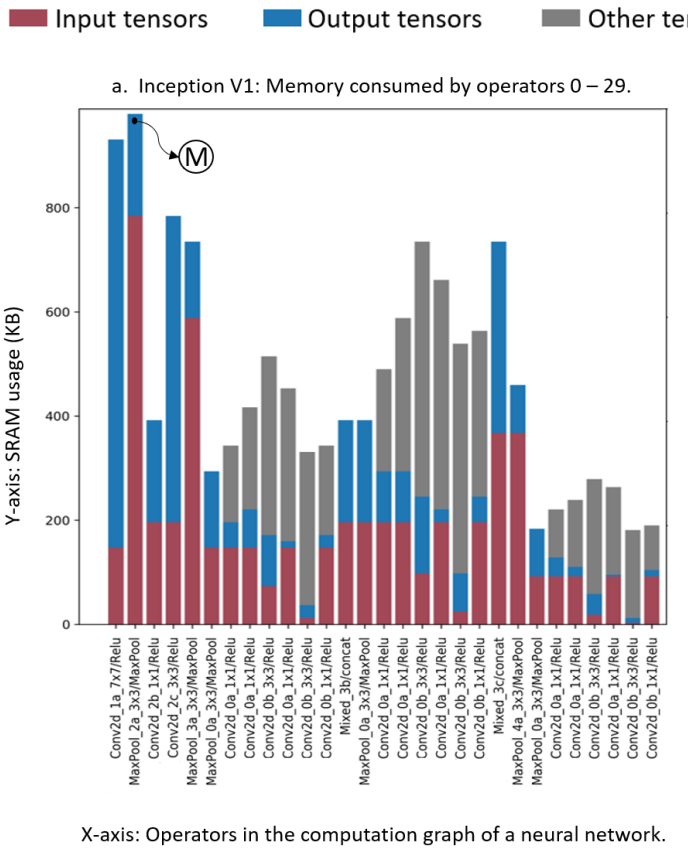
- Before deployment, the execution memory requirement of models is often unknown or calculated with less accuracy. i.e., there will exist a few MB of deviations in the calculations

- When the model is targeted to run on smartphones or edge GPUs, these few MB deviations do not cause any issues. But when target is MCUs, then the low-accuracy calculation causes run-time memory overflows and/or restrict flashing model on IoT devices due to SRAM peaks

- TMM can be realized to **accurately compute and visualize the tensor memory requirement of each operator** in any NN model computation graph. We use this high-accuracy calculation method in the core algorithm design of our efficient NN execution approach

Tensor Memory Mapping (TMM)

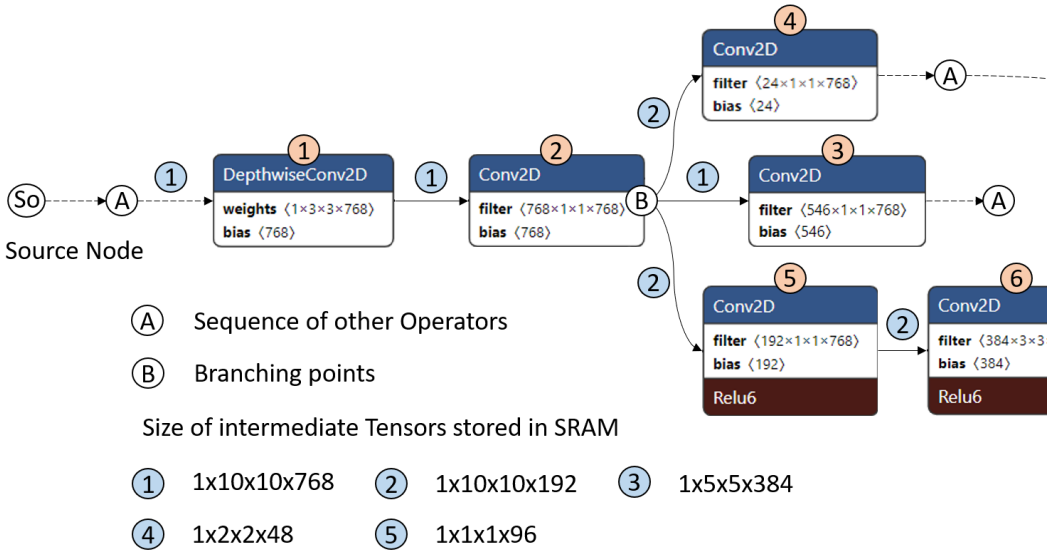
Confirm
Smart Manufacturing



- TMM is suitable for any pre-trained models like NASNet, Tiny-YOLO, SqueezeNet, etc.
  - ✓ Computes total required SRAM. i.e., the space required to store **input tensors + output tensors + other tensors**, and then exports the detailed report in CSV format
  - ✓ Can also produce images that show the tensor memory requirement of each operator
  - ✓ For example, when we feed the Inception V1 that contains 84 graph nodes/operators to TMM, it produces shown Fig (for brevity, we show only 0 - 29 operators) along with the detailed CSV report

Load Fewer Tensors, also Re-use

Confirm
Smart Manufacturing



A part of the COCO SSD MobileNet graph with its branched operators

- Intermediate tensors stored in SRAM during the NN graph execution
- First branching point, when the default model execution software is utilized, the two tensors are loaded. Then it executes all the branched operators
- This method of loading many tensors and executing many operators leads to the most critical SRAM overflow issue, especially in the scenarios where multiple branches are emerging from one branching point

Load Fewer Tensors, also Re-use

Confirm
Smart Manufacturing

- As explained, in traditional NN graph execution, multiple tensors of various sizes are loaded into the buffer. Such bulk loading is the reason for causing peak memory usage. In contrast, our approach

  - ✓ Executes many operators by just loading a minimum number of tensors. This part of our approach also aims to achieve **SRAM conservation by tensors re-usage**

  - ✓ Identifies and stores a particular set of tensors in the buffer (buffers are created within SRAM) and first executes the branch of the NN graph containing operators compatible with the stored tensors

  - ✓ Then in next iteration, it loads tensors that are suitable as input for the set of operators belonging to the next branch, then performs the execution. After each iteration, the buffers are reclaimed

Cheapest NN Graph Execution Sequence

Confirm
Smart Manufacturing

- Changing execution sequence of operators still produces a valid scheme?
  - ✓ Computation graph of NN model are DAG. We proved that in DAGs, execution of available nodes in any topological order will result in a valid execution sequence
  - ✓ Branched computation graphs provide freedom for the NN model execution software to alter the execution order/sequence of the operators
  - ✓ So, the proposed approach achieves its memory conservation goal by intelligently selecting the execution branch that when executed consumes less SRAM (reduces the peak memory consumption)

Core Algorithm

Confirm
Smart Manufacturing

- Discovering multiple topological orders of nodes in a computation graph. Uses the previous slides presented methods inside loops

  - ✓ Analyzes the complete network by running through each branch of the network and finally discovering the cheapest graph execution path/sequence

  - ✓ The time consumed by algorithm to produce the results depends on complexity $T_c = (|O|\ 2^{|O|})$, where $|O|$ is the total operators count

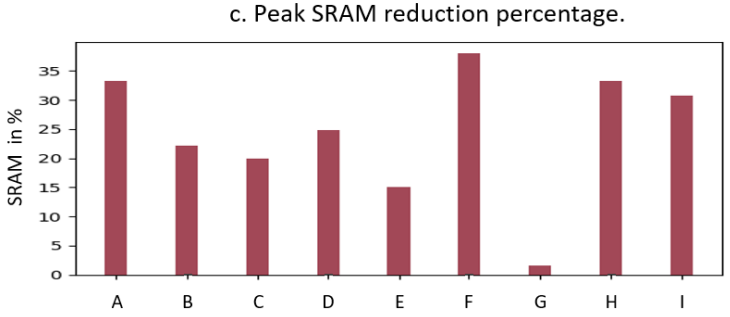
Experimental Evaluation

Confirm
Smart Manufacturing

- Downloaded popular pre-trained TensorFlow Lite models (.tflite format)
  - ✓ 9 models that range from image classification to text detection
  - ✓ Contain hundreds of operators so the complexity of our algorithm is high
  - ✓ Conduct evaluation on a standard laptop Intel (R) Core (TM) i7-5500 CPU @ 2.40 GHz
- We execute each model using default execution sequence, then using execution order altered sequence
  - ✓ We tabulate corresponding peak SRAM usage, unit inference time, and the energy consumed
  - ✓ In next slides we present the benefits achieved as a result of optimizing model graph execution using the proposed approach

SRAM Usage

Confirm
Smart Manufacturing



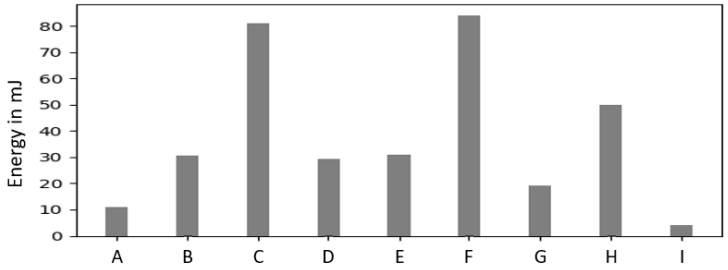
A – MobileNet V1, B – Squeezenet, C – Inception V1, D – MnasNet,
E – NASNet mobile, F – DenseNet, G – DeepLabv3, H – PoseNet, I – EAST

- We take quantized DenseNet with its default execution sequence and feed it to TMM. From the computed memory requirement for each operator in default graph, 24th operator showed peak SRAM usage of 8429.568 KB
- Next, after applying proposed Algorithm on DenseNet, the resultant memory-friendly graph execution sequence, when evaluated by TMM showed the peak memory of only 5221.264 KB (peak reduced by 38.06%)
- We plot thus calculated peak SRAM reduction percentage for all models (A - I)
- The maximum peak SRAM reduction of **38.06% for DenseNet** and least of **1.61% for DeepLabv3**

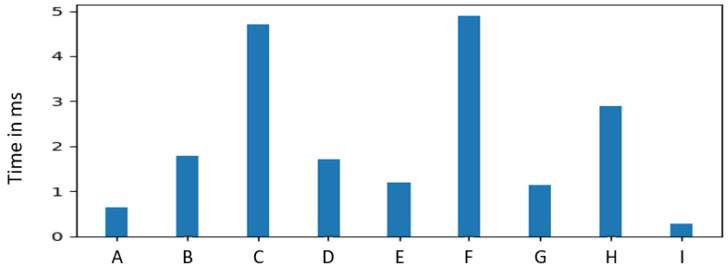
Inference Time and Energy Consumption

Confirm
Smart Manufacturing

a. Reduction in consumed energy.



b. Reduction in inference time.



A – MobileNet V1, B – Squeezenet, C – Inception V1, D – MnasNet,
E – NASNet mobile, F – DenseNet, G – DeepLabv3, H – PoseNet, I – EAST

- Execute each model (A - I) first with their default execution sequence, then with the memory peak reduced sequence produced by the proposed approach
  - ✓ DenseNet shows max inference time reduction of 4.9 ms and the least of 0.28 ms reduction for EAST
  - ✓ 4 - 84 mJ less energy to perform unit inference since executing model using optimized sequence is 0.28 - 4.9 ms faster than the default sequence

Conclusion

- We presented an approach to efficiently execute (with reduced SRAM usage) deeply optimized (maximally compressed) ML models on resource-constrained devices. As shown with evaluations, when users apply the presented approach, they can

  - ✓ Execute large-high-quality models on their IoT devices/products **without needing to upgrade** the hardware **or alter the model architecture** and re-train to produce a smaller model

  - ✓ Devices can control real-world applications by making **timely predictions/decisions**

  - ✓ Devices can perform high **accuracy offline analytics** without affecting the operating time of battery-powered devices

Confirm
Smart Manufacturing

Confirm
Smart Manufacturing

Contact: Bharath Sudharsan
Email: bharath.sudharsan@insight-centre.org

www.confirm.ie

Science Foundation Ireland sfi For what's next