Confirm
Smart Manufacturing

Confirm
Smart Manufacturing

ECML PKDD 2021 Tutorial

Machine Learning Meets Internet of Things: From Theory to Practice

Part III: Deep Optimizations of CNNs and Efficient Deployment on IoT Devices

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

Neural Networks on IoT Devices

Confirm
Smart Manufacturing

- TinyML is ultra-low-power machine learning - aka running NNs on MCUs. App landscape categorized based on data:
  - ✓ **Audio:** always-on inference for context recognition, spot keywords/wake-words/control-words
  - ✓ **Industry telemetry**: models deployed on MCUs monitor motor bearing vibrations, other sensors to detect anomalies and predict equipment faults
  - ✓ **Image**: object counting, text recognition, visual wake words
  - ✓ **Physiological/behavior:** activity recognition using IMU or EMG data

Neural Networks on IoT Devices

Confirm
Smart Manufacturing

- EdgeML: https://microsoft.github.io/EdgeML/
- Model Zoo: https://github.com/ARM-software/ML-zoo

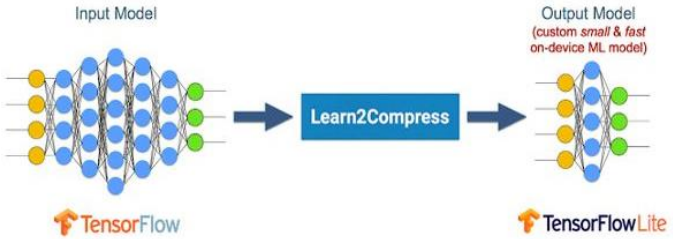
- Professional Certificate in TinyML
- TinyMLPerf

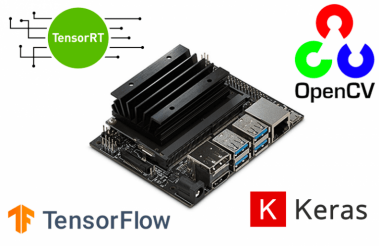
Who are practicing TinyML: Executing NNs on their MCU-based devices/products

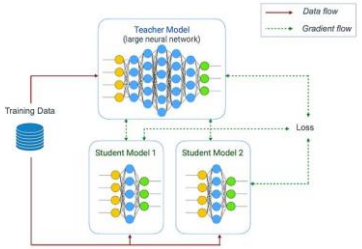
Challenge - Existing Frameworks

Confirm
Smart Manufacturing

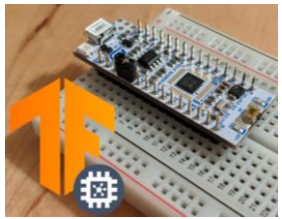
- Compression levels, speedups produced by generic optimization toolkits in ML frameworks (e.g., PyTorch, Tensorflow) are not sufficient since they target better-resourced edge hardware like Pi, Jetson Nano, Smartphones

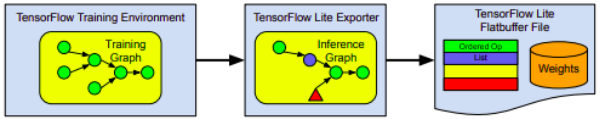






- TF Lite for MCUs, core runtime can fit in 16 KB on an ARM Cortex M3 - run basic NNs on MCUs without needing OS support or dynamic memory allocation





Model export workflow

- Before utilizing TF Lite Micro

✓ ***Need to optimize high memory, computation NNs in multiple aspects to produce small size, low latency, low-power consuming models***

Challenge - Performance Metrics Trade-offs

Confirm
Smart Manufacturing

- During runtime, IoT application that interacts with the loaded ML model may demand high performance on one particular metric over others

Paintball gun on Boston Dynamic's $75,000 robot dog

- ✓ **High accuracy** predictions mandatory as memory sufficient and non-time critical
- ✓ **Highest model size reduction** for low-memory device
- ✓ **Ultra-fast inference** when edge application demands real-time

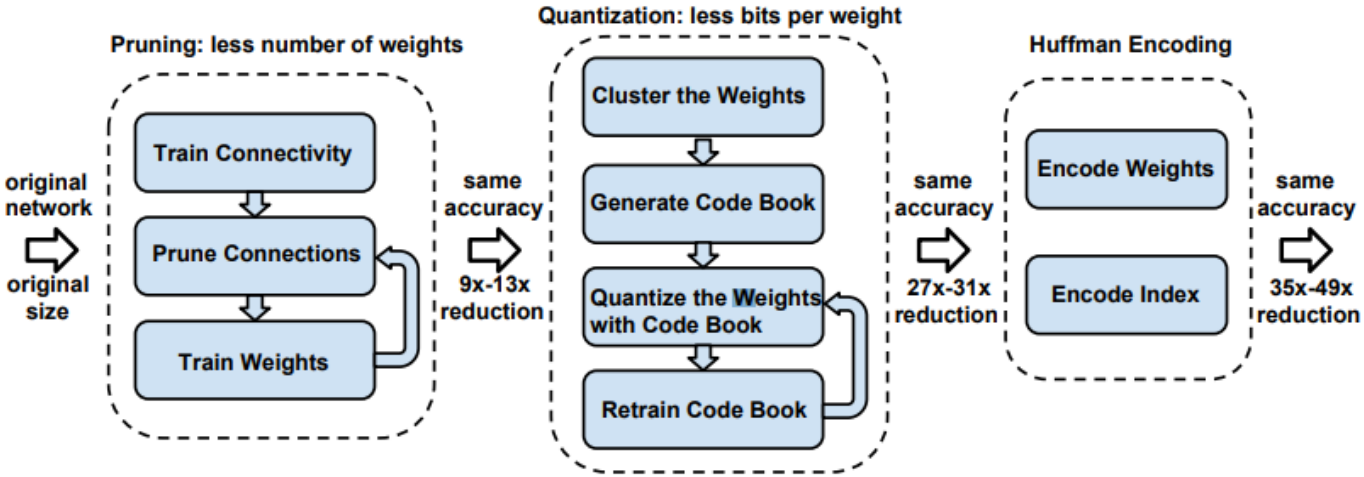
- ***How to perform optimization that favors particular metrics over others?***

Challenge - Optimization Compatibility

Confirm
Smart Manufacturing

- ✓ NNs optimized using one state-of-the-art method may exceed the target IoT device hardware's memory capacity by just a few bytes
- ✓ Need to spend days on unproductive to find a compatible optimizer, then implement it to check if the new compression and accuracy levels are satisfactory
- ✓ Models cannot be optimized further if failed to find a method that matches the previous optimizer. So, either have to tune the model network architecture and re-train from scratch
- ✓ To speed up the R&D phase (going from idea to product) of AI-powered IoT devices, ***we need a comprehensive guideline to optimize NN models that can readily be deployed on resource-constrained MCUs-based hardware***

Top Deep Optimization of NNs



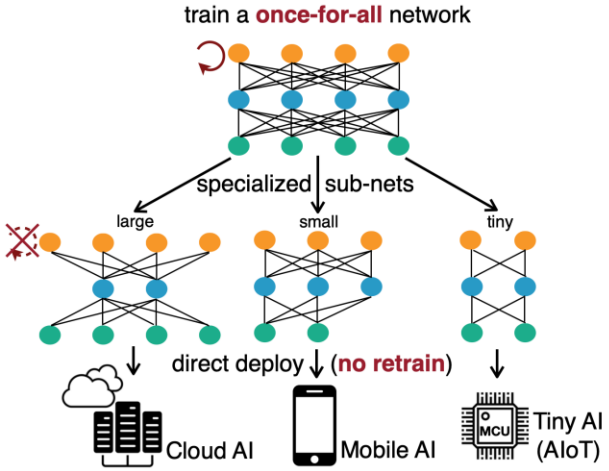
- Three-stage compression pipeline paper with Pruning + Quantization + Huffman coding

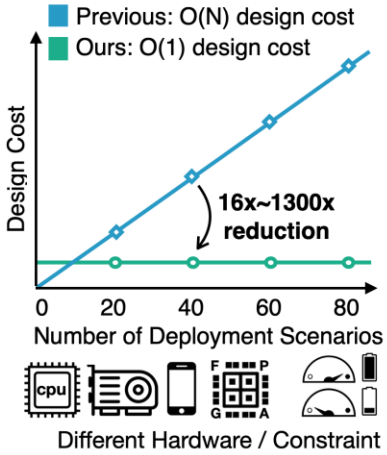
  - ✓ Pruning reduces the number of weights by 10×. Quantization further improves the compression rate between 27× and 31×

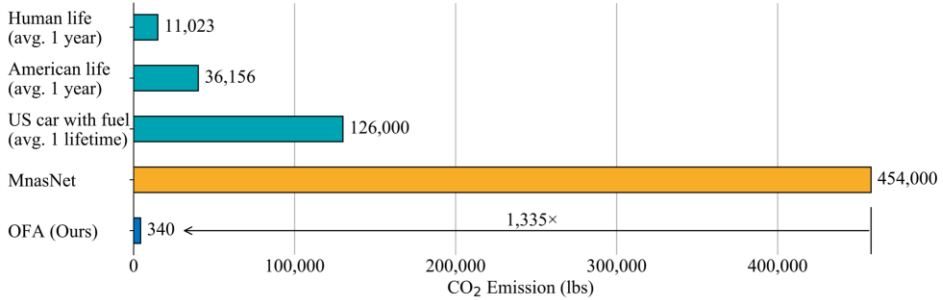
  - ✓ Huffman coding gives more compression between 35× and 49x. The compression scheme doesn't incur any accuracy loss

Top Deep Optimization of NNs

Confirm
Smart Manufacturing



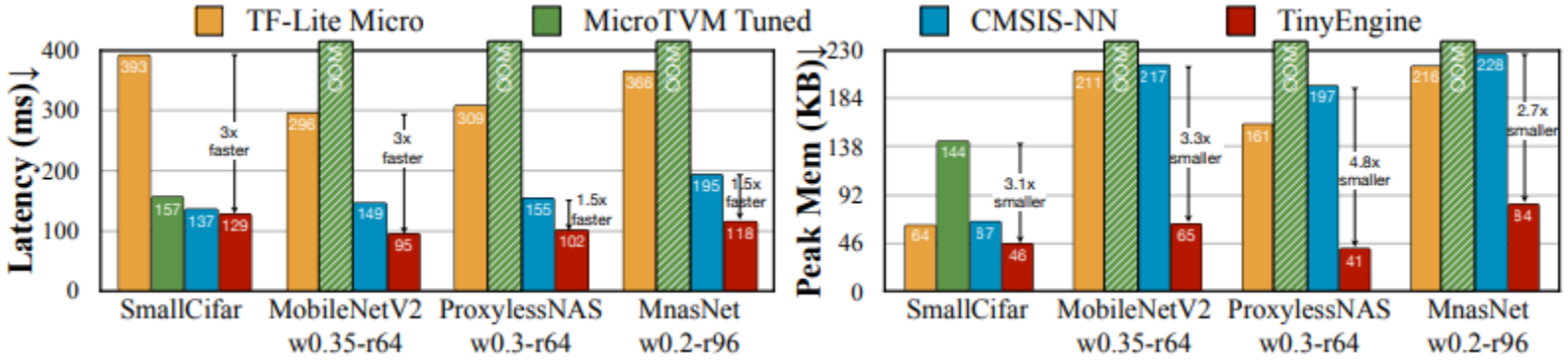




- Once for all paper
  - ✓ A single network is trained to support versatile architectural configurations including depth, width, kernel size, and resolution
  - ✓ Given a deployment scenario, a specialized subnetwork is directly selected from the once-for-all network without training
  - ✓ This approach reduces the cost of specialized deep learning deployment from O(N) to O(1)

Top Deep Optimization of NNs

Confirm
Smart Manufacturing



- MCUNet paper:
  - ✓ TinyEngine achieves higher inference efficiency while reducing the memory usage. 3× and 1.6× faster than TF-Lite Micro (Google) and CMSIS-NN (ARM) respectively
  - ✓ By reducing the memory usage TinyEngine can run various model designs with tiny memory, enlarging the design space for TinyNAS under the limited memory of MCU
  - ✓ Outperforms existing libraries by eliminating runtime overheads, specializing each optimization technique, and adopting in-place depth-wise convolution

Research Question

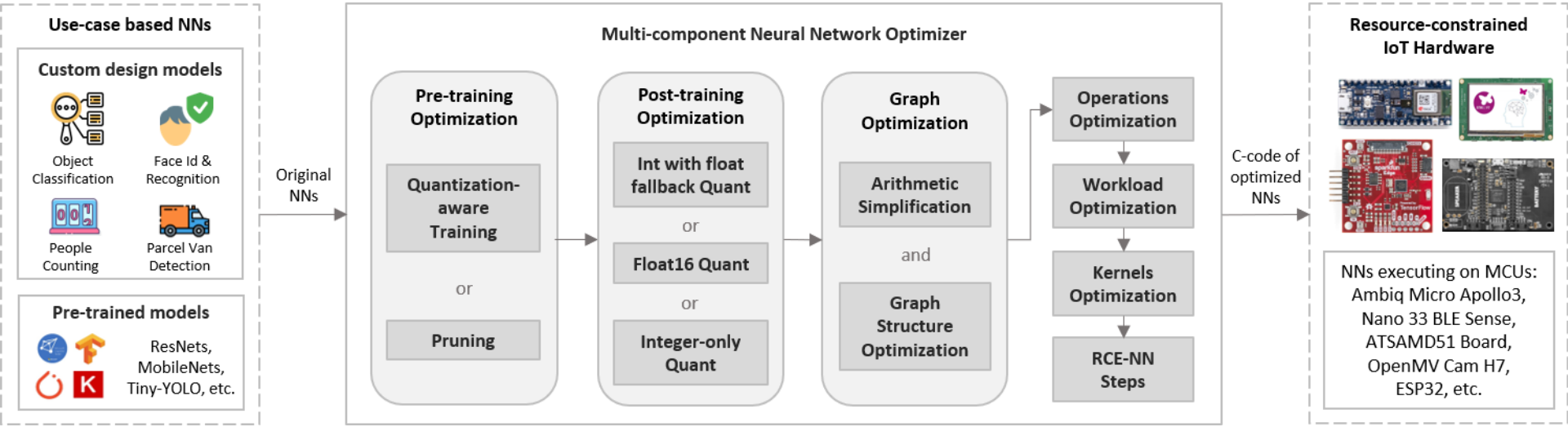
Confirm
Smart Manufacturing

- Is it possible to combine more than one state-of-the-art optimization technique to achieve deeper optimization levels of ML models? If possible, what is the maximum achievable

  - ✓ Size reduction
  - ✓ Inference speedup
  - ✓ Which combination shows the highest accuracy preservation

Multi-component NN Optimizer: Architecture

Confirm
Smart Manufacturing

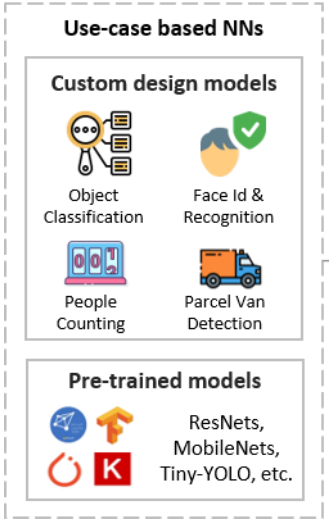


The architecture of our multi-component model optimizer

- Sequence to follow for optimizing Neural Networks to enable its execution on resource-constrained AIoT boards, small CPUs, MCUs based IoT devices
- In the upcoming slides each optimizer component shall be presented

Use-case based NNs: Input

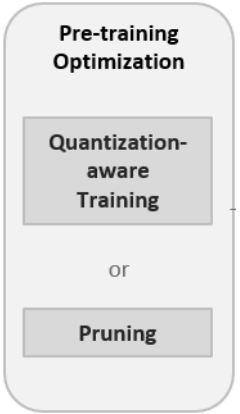
Confirm
Smart Manufacturing



- The optimizer takes a NN as an input and produces a highly optimized version of the input NN that can run on low resource, cost and power MCU-based IoT hardware. The input network can be
  - ✓ Pre-trained models such as Inception, Xception, Mobilenet, Tiny-YOLO, Resnet, etc
  - ✓ Or the custom-designed, yet to be trained networks

Pre-training Optimization

Confirm
Smart Manufacturing



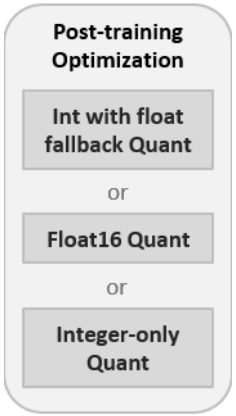
- The custom-designed, yet to be trained networks should pass through the pre-training optimization component

  - ✓ **Pruning:** We implemented the magnitude-based weight pruning, where the model's weights are gradually zeroed out during the training to achieve model sparsity. Thus obtained sparse models are easier to compress, and the zeroes can be skipped during inference, resulting in latency improvements

  - ✓ **Quantization-aware Training:** When quantizing a CNN, the parameters and computations go from a higher to lower precision, resulting in improved execution efficiency at a cost of information loss. This loss is because the model weights can only take a small set of values, thus losing the minute differences between them. To reduce the loss and maintain the model accuracy, we introduce quantization error as noise during the model training, as part of the overall loss, which the optimization algorithm in use tries to minimize

Post-training Optimization

Confirm
Smart Manufacturing



- Quantize the models by reducing the precision of their weights and activations to save memory and simplify calculations often without much impact on accuracy. Following are the popular techniques we implemented

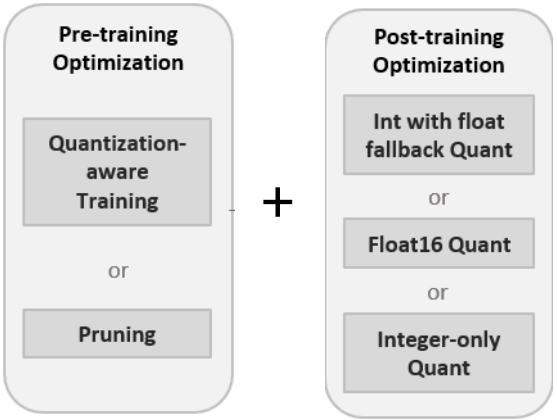
  - ✓ **Int with float fallback quantization:** Fully integer quantize a model, but use float operators when they don't have an integer implementation (to ensure conversion occurs smoothly)

  - ✓ **Float 16 quantization:** Reduce the size of floating point models by quantizing the weights to IEEE standard for 16-bit floating point numbers. Float16 models run on small CPUs without modification

  - ✓ **Integer-only quantization:** To improve the NN compatibility with integer only hardware devices or accelerators by making sure all model math is in integer

Joint Pre and Post-training Optimization

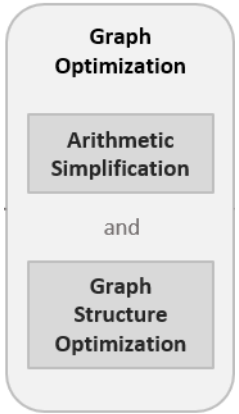
Confirm
Smart Manufacturing



- First, any of the pre-training optimization methods has to be applied to the model, followed by its Int-8 post-training quantization
  - ✓ When more than 11x size reduction is required
  - ✓ Example, when we aim to execute Inception v3 (23.9 MB after quantization) on a AIoT boards, which only has only 16 MB Flash memory

Graph Optimization

Confirm
Smart Manufacturing

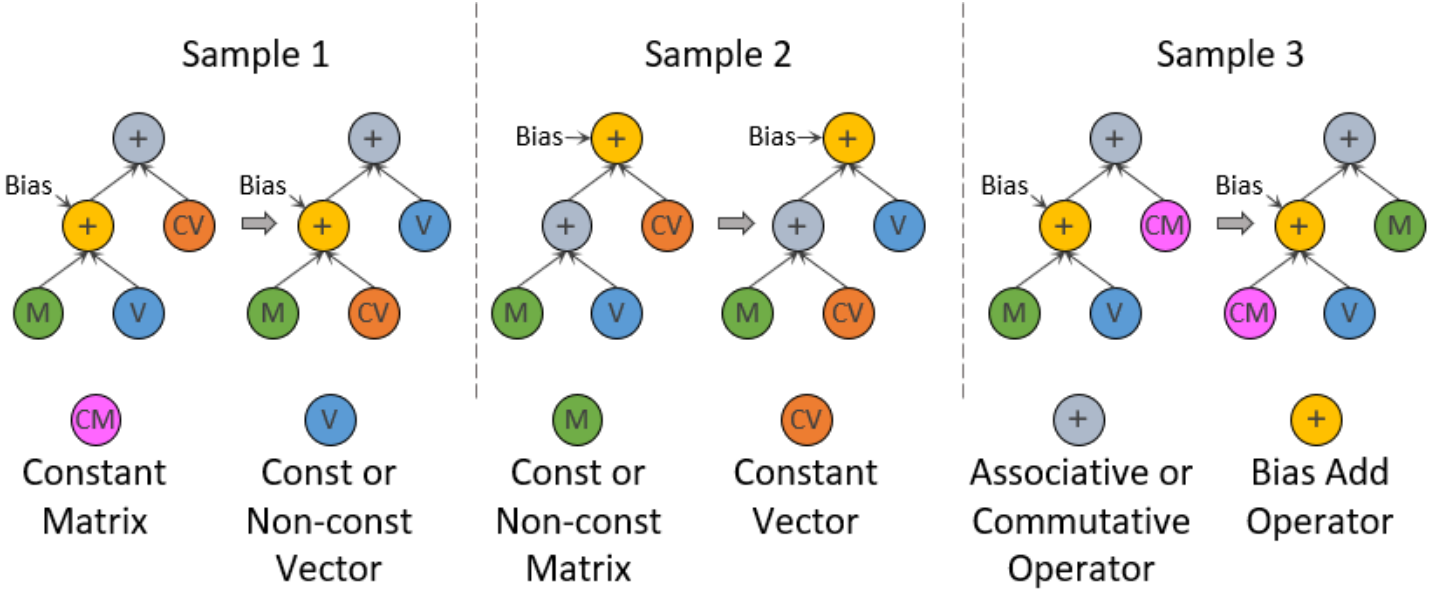


- The interior of trained models is a graph with defined data flow patterns
  - ✓ Graph contains an arrangement of nodes and edges
  - ✓ Nodes represent the operations of a model
  - ✓ Graph edges represent the flow of data between the nodes
- We present techniques to optimize graphs of NN to improve the computational performance of NN while reducing peak SRAM usage on MCUs
- We perform graph optimization in sequential steps
  - ✓ Arithmetic Simplification
  - ✓ Graph Structure Optimization

Graph Optimization: Arithmetic simplification

Confirm
Smart Manufacturing

- Arithmetic re-writes rely on known inputs. As shown, the known constant vector is grouped with an unknown vector that might be constant or non-constant. After performing such re-writes, if the unknown vector turns out to be a constant, then graph performance improves



Graph Optimization: Graph Structure

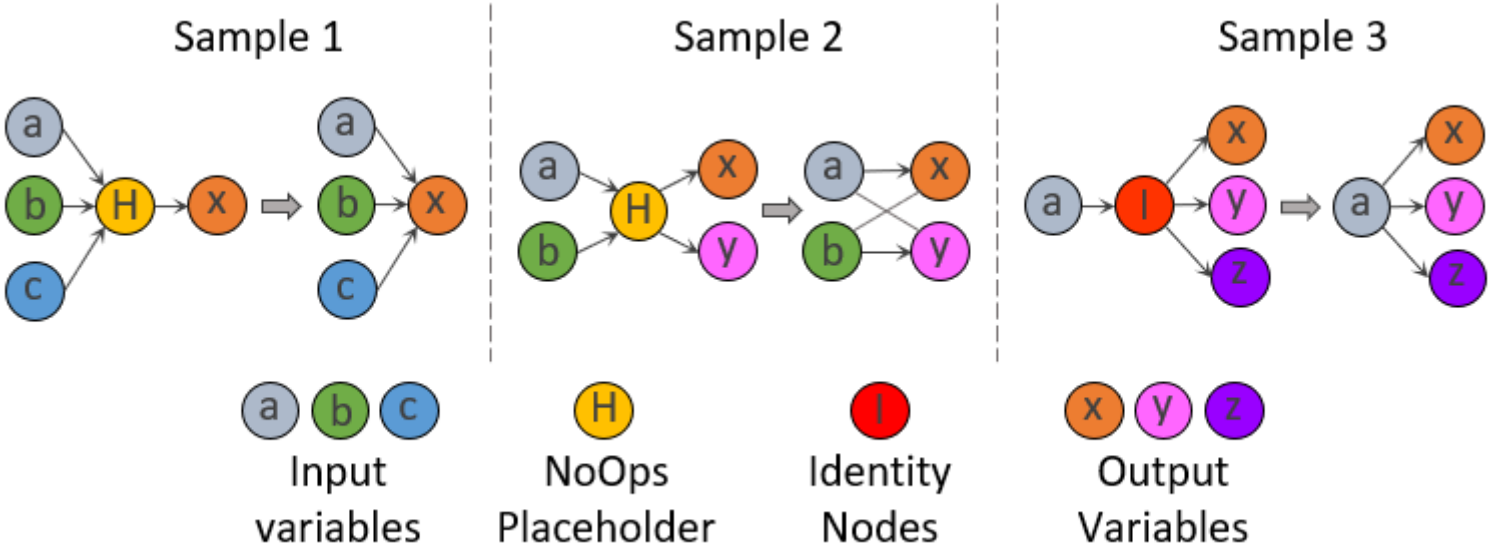
Confirm
Smart Manufacturing

- We propose tasks that when realized will optimize the graph for efficiency
  - ✓ Task1: Remove loop-invariant sub-graphs. i.e., remove the loops that are true both before and after iterations
  - ✓ Task2: Remove dead branches/ends from the graphs. So, during execution on MCUs, backtracing from the dead-end is not required to progress in the graph
  - ✓ Task3: Use a transitive reduction algorithm on entire graph to remove redundant control edges. Shorten the critical path of a model step by rearranging control dependencies
  - ✓ Task4: Replace recurrent subgraphs with optimized kernels/executable modules

Graph Optimization: Graph Structure

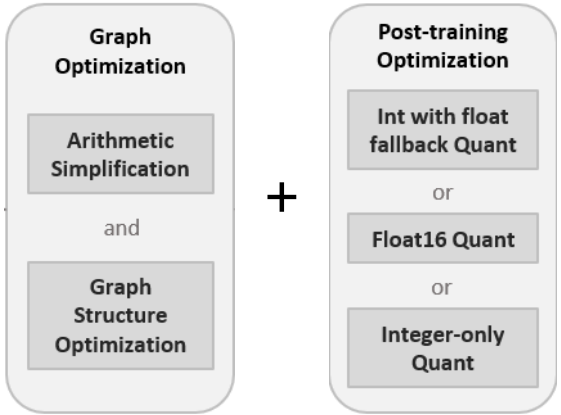
Confirm
Smart Manufacturing

- ✓ Task5: This task when realized, reduces the graph's size resulting in processing speedups. Here, we identify and remove nodes that are effectively NoOps (a placeholder of control edges) and Identity nodes (outputs data with the same content and shape of input)



Joint Graph and Post-Training Optimization

Confirm
Smart Manufacturing

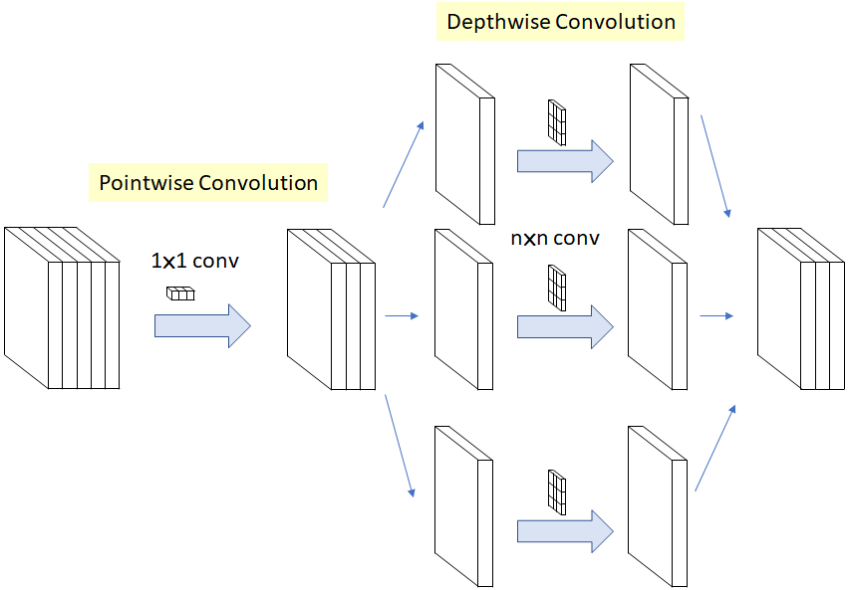


- First, the two Arithmetic simplification and Graph Structure Optimization steps needs to be applied to the original un-optimized model
- Then any of the post-training model optimizers need to be applied

Operations Optimization

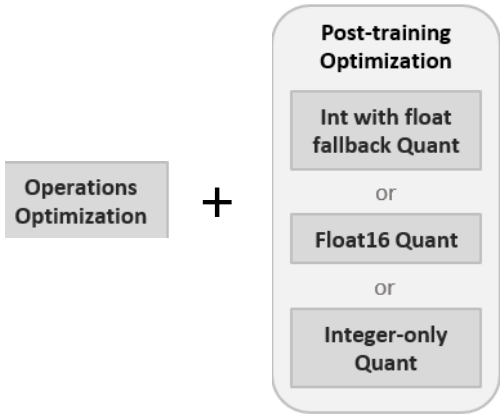
Confirm
Smart Manufacturing

- When designing ML models for tiny IoT hardware, only limited operations can be used to keep the cost low
- Over 90% arithmetic operations are used by convolutional (CONV) layers. So, we already convert floating-point operations into int-8 (fixed point) during post-training quantization



- ✓ Depth-separation of 3D filters
- ✓ Also Decompose 2-D CONVs and 1-D CONVs to reduce parameters and operations count
- ✓ 3D convolution uses C * A * B multiplications, whereas a depth-separable 3D convolution only requires C + A + B multiplications

Joint Operations & Post-training Optimization



- Even if the models such as MobileNet and SqueezeNet are manually designed to execute within a tight memory budget, it would exceed the AIoT board's capacity by over 5 x times
- Hence, we propose to first optimize the operations of any model, then perform post-training model optimization

Workload Optimization

Confirm
Smart Manufacturing

- The complexity and size of the model have an impact on the workload
  - ✓ Larger and denser models lead to increased processor workload
  - ✓ Hardware spends more time working and less time idle resulting in elevated power consumption and heat output
- We present techniques that can be used to reduce workloads. The methods we recommend apply globally, i.e., not biased towards local performance optimizations for a single operation, as in many previous works

Workload Optimization

Confirm
Smart Manufacturing

- **Input data reduction** Use computationally inexpensive low pass filters
- **Hardware accelerators** Resource-constrained devices are not capable to utilize off-the-shelf accelerators. For successful offloads, we recommend

  ✓ Storing the C code of the optimized model to be executed in a shared memory location
  ✓ Perform parallel offloading for internal data reuse
  ✓ Offload the processing to the inbuilt KPU, FFT units

- **Number of threads** Limit the number of threads initialized by NNs for computation
- **Low-level optimization** Perform low-level optimization of convolution to eliminate unnecessary data layout transformation overheads
- **Linear algebraic properties** Analyze the linear algebraic properties of models and apply algorithms such as Strassen Gaussian elimination, Winograd's minimal filtering

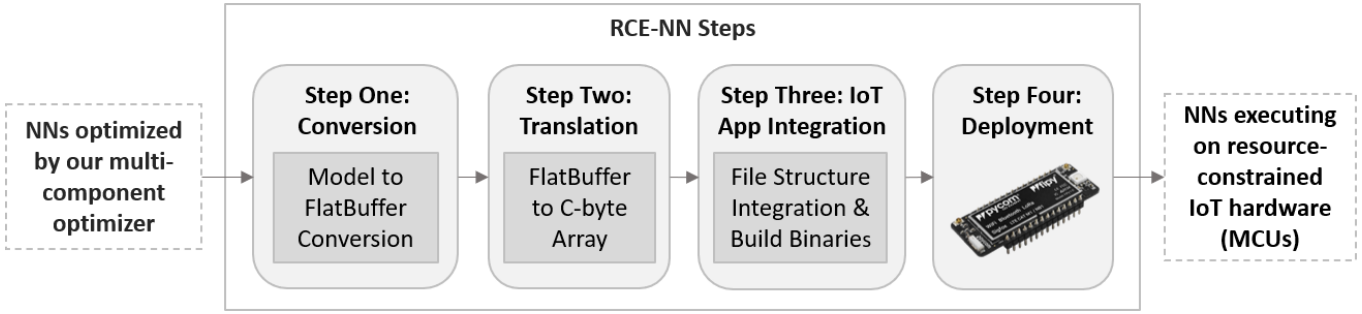
Kernels Optimization

Confirm
Smart Manufacturing

- The general C/C++ implemented kernels for MCUs need hardware-specific optimizations
- We present library independent kernel optimization techniques that are generic across a wide range of resource-constrained hardware for guaranteeing no runtime performance bottlenecks
  - ✓ Remove excess modules, components in project directory before building a project
  - ✓ Group multiple operators together within a single kernel
  - ✓ Go deep into the assembly code level for improving small matrix multiplication tasks. Implement the matrix multiplication kernel with 2x2 kernels to save on load instructions
  - ✓ Convolutions should be partitioned into disjoint pieces to achieve parallelism
  - ✓ Self-customized thread pooling techniques should be used to reduce overheads while launching and suppressing threads to reduce performance jitters while adding threads

RCE-NN Steps

Confirm
Smart Manufacturing



- **Step1: Model to FlatBuffer Conversion**
  - ✓ Creates flat buffers of the CNNs
  - ✓ Memory-mapped and can be utilized directly from disk/flash
  - ✓ Zero additional memory requirements for data access

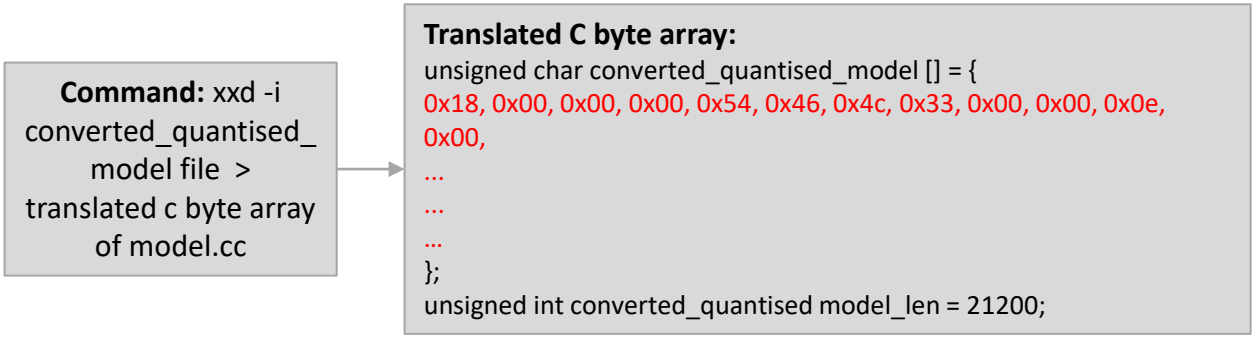
RCE-NN Steps

Confirm
Smart Manufacturing

- **Step2: FlatBuffer to C-byte Array**

  - ✓ MCUs in edge devices lack native filesystem support. Hence, we cannot load and execute the regular format (“.h5”, “.pb”, “.tflite”, etc.) trained CNNs

  - ✓ We convert the quantized version of the trained model into a C array and compile it along with the program for the IoT application which is to be executed on the edge device



Method to translate the trained model into a C byte array

RCE-NN Steps

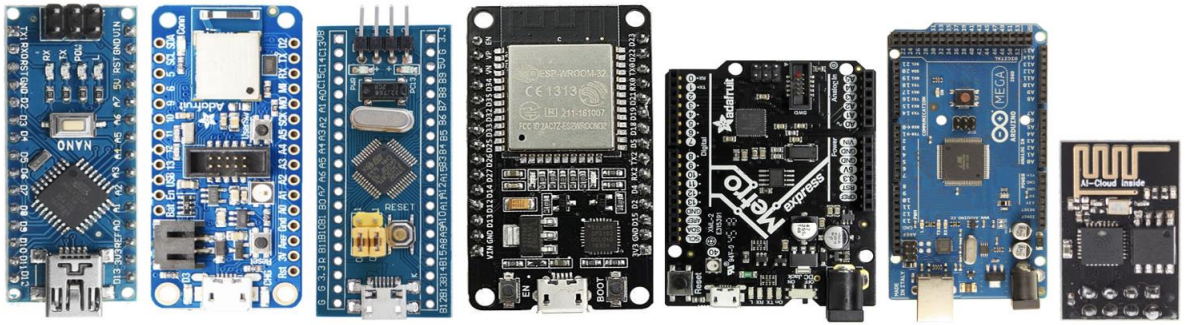
Confirm
Smart Manufacturing

- **Step3: IoT App Integration**
  - ✓ We fuse the trained CNN (translated c-byte array) with the main program of an IoT use-case
  - ✓ We build binaries that will be loaded on MCUs for execution
- **Step4: Deployment**
  - ✓ The generated binaries of a NN model are flashed via serial port on the MCU-based devices
  - ✓ To flash, use an external In-System Programmer (ISP) or a ParallelProgrammer, which does not occupy bootloader space, also avoids the bootloader delay

Resource-constrained IoT Hardware: Output

Confirm
Smart Manufacturing

- **Output:** A highly optimized version of the input NN that can run on low resource, cost and power IoT hardware such as the below shown MCUs

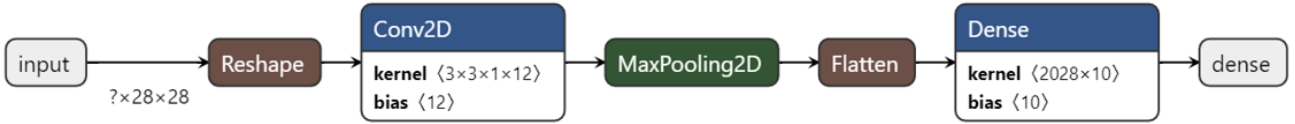


MCU1	MCU2	MCU3	MCU4	MCU5	MCU6	MCU7
ATmega328P	**nRF52840**	**STM32f103c8**	**ESP32**	**ATSAMD21G18**	**ATmega2560**	**ESP8266**
8 kB SRAM	256 kB SRAM	20 kB SRAM	520 kB SRAM	32 kB SRAM	8 kB SRAM	32 kB SRAM
32 kB Flash	1 MB Flash	128 kB Flash	4 MB Flash	256 kB Flash	256 kB Flash	1 MB Flash
@ 16 MHz	@ 64 MHz	@ 72 MHz	@ 240 MHz	@ 48 MHz	@ 16 MHz	@ 80 MHz

Evaluation Setup

Confirm
Smart Manufacturing

- We implement each of the optimizer component on CNNs and present the experimental results
  - ✓ We define a basic CNN whose architecture is shown below



  - ✓ CNN1: When above network is trained using the standard MNIST Fashion dataset
  - ✓ CNN2: When above network is trained using the MNIST Digits datasets

Evaluation: Pre-training Optimization Components

Confirm
Smart Manufacturing

- Below, we show the architecture of the **QAT CNN1**

  ✓ 2.49x smaller in size
  ✓ Can Infer 10.76 x times faster

- The **Pruned CNN1** architecture is same as the Original CNN1

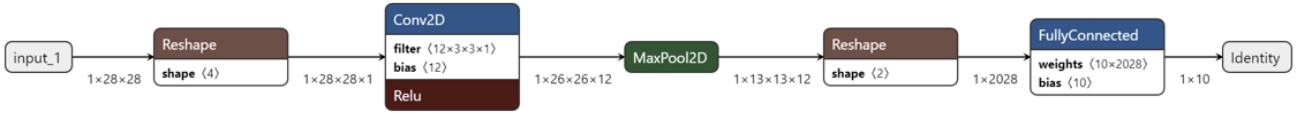
  ✓ 2.76x smaller in size
  ✓ Can Infer 1.85 x times faster

Evaluation: Post-training Optimization Components

Confirm
Smart Manufacturing

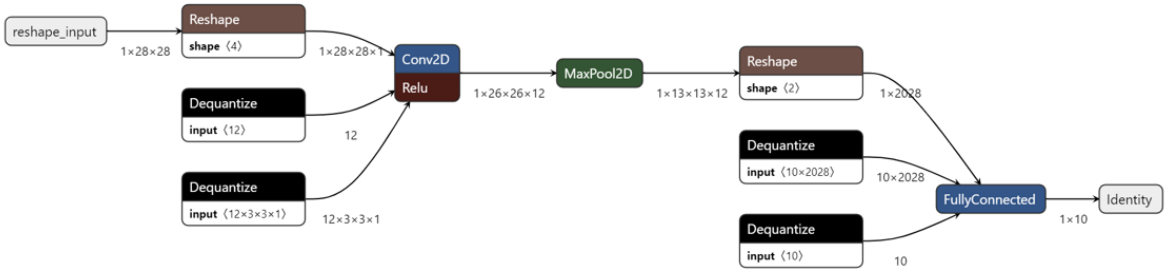
- Below, we show the architecture of **Int with float-fallback quantized CNN1**

  ✓ 12.06x smaller in size. Can Infer 960.85 x times faster



- Below, we show the architecture of **Float16 quantized CNN1**

  ✓ 6.31x smaller in size. Can Infer 1256.5 x times faster



Evaluation: Post-training Optimization Components

Confirm
Smart Manufacturing

- Below, we show the architecture of **Int-only quantized CNN1**

  ✓ 11.69x smaller in size. Can Infer 882.94 x times faster

- Analyzing the post-training optimization results

  ✓ Float16 quantization for reasonable compression rates (we obtain approx. 6x compression), without loss of precision (we experience only 0.01 % loss in accuracy)

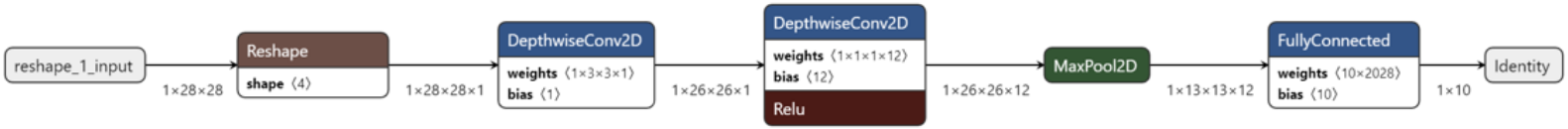
  ✓ Int with float-fallback quantization for smallest model size and fastest inference on MCUs

  ✓ Float16 quantization for fastest inference on CPUs

Evaluation: Operation, Graph Optimization Components

Confirm
Smart Manufacturing

- **Graph optimized CNN1:** We implemented and performed all applicable arithmetic simplification rewrites and graph structure optimization tasks on CNNs
  - ✓ 2.76x smaller in size, 13.12x times faster
- Below, we show the architecture of the **Operations optimized CNN1**
  - ✓ 6.36x times faster



Optimization Results Analysis

Confirm
Smart Manufacturing

We performed analysis based on the evaluation results

- **Best Optimization Sequence for Smallest Model Size:**
  - ✓ Graph optimized then integer with float fallback quantized version is only 22.5 KB
  - ✓ 12.06 x times smaller than original CNN

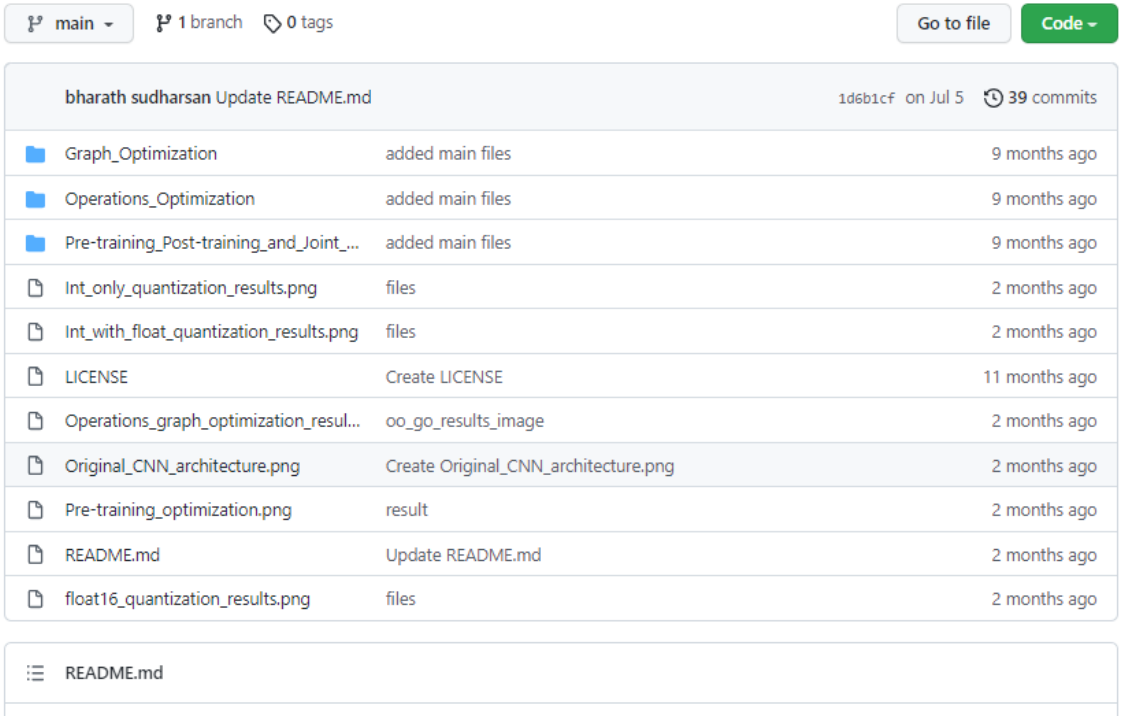
- **Best Optimization Sequence for Accuracy Preservation:**
  - ✓ Graph optimized then integer only quantized version
  - ✓ For MNIST Fashion, the accuracy increased by 0.27 %, and by 0.13 % for MNIST Digits

- **Best Optimization Sequence for Fast Inference**:
  - ✓ Operations optimized then float16 quantized version
  - ✓ Produces the fastest unit inference results in 0.06 ms

Demo

Confirm
Smart Manufacturing

- Walkthrough Jupyter Notebook @ https://github.com/bharathsudharsan/CNN_on_MCU



Summary

Confirm
Smart Manufacturing

- To enable ultra-fast and accurate ML-based offline analytics on resource-constrained IoT devices, we presented an end-to-end multi-component ML model optimization sequence
- Researchers and engineers can use our optimization sequence to optimize high memory, computation demanding models in multiple aspects to produce small size, low latency, low-power consuming models
- Our optimization components can produce models that are; (i) 12.06 x times compressed; (ii) 0.13% to 0.27% more accurate; (iii) orders of magnitude faster unit inference at 0.06 ms

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