Tiny ML: Machine learning for embedded systems

"The Future of Machine Learning is Tiny and Bright. We're excited to see what you'll do!"

Chaigneau Yanis

GreenAI U.P.P.A.

05-17-2022

1. TinyML
2. Micro-controllers
3. Fit models to micro-controllers
4. Applications
5. Patch-Based Inference
6. Conclusion

Remainder: Impact of Machine Learning

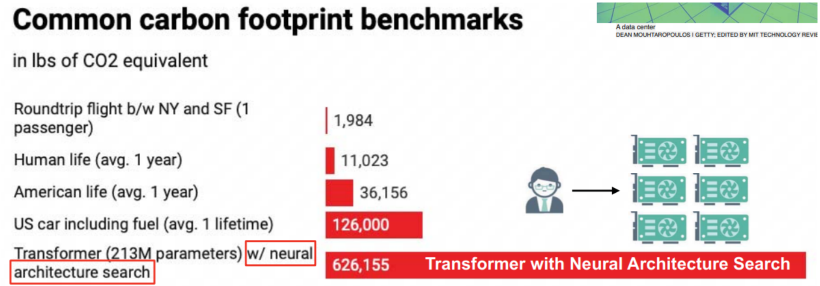


Figure 1: Carbon footprint comparative study between a neural network and activities [Han, 2021].

The impact of Machine Learning is non neglictible → Reduce the size of models !

Bound effect

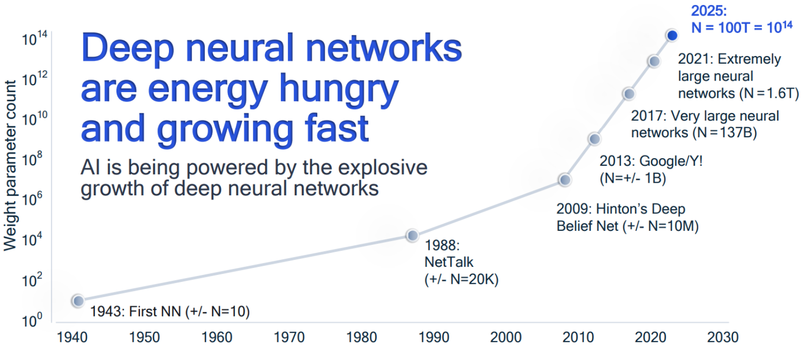


Figure 2: New Moore Law for Deep Learning [Fournarakis, 2021]

What is Tiny ML?

- What is tinyML ?
  TinyML: When a neural network model can be run at an energy cost of below 1 mW

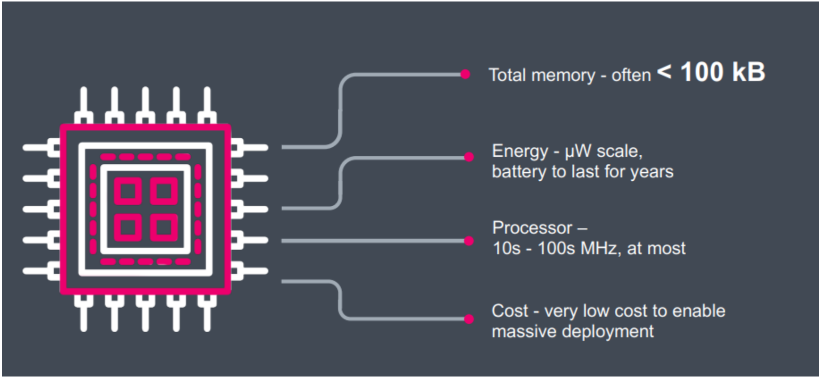


Figure 3: Definition of TinyML by Pete Warden[NEUTON.AI, 2022]

Tiny ML ?

- This presentation is highly inspired by the book *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers by Warden and Situnayake [Pete Warden, 2019].*

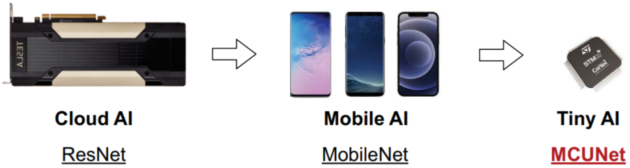


Figure 4: Different models for different capacities [Han, 2021]

It is also inspired by many presentations yielded at the TinyML Summit every year (March 2022)

Why ?

- Function – wanting a smart device to act quickly and locally (independent of the Internet).
- Cost – accomplishing this with simple, lower cost hardware.
- Privacy – not wanting to share all sensor data externally.
- Efficiency – smaller device form-factor, energy-harvesting or longer battery life.

Limitations in terms of hardware

- Decrease in energy consumption → limitations in sRAM memory, flash memory, microprocessor capacities

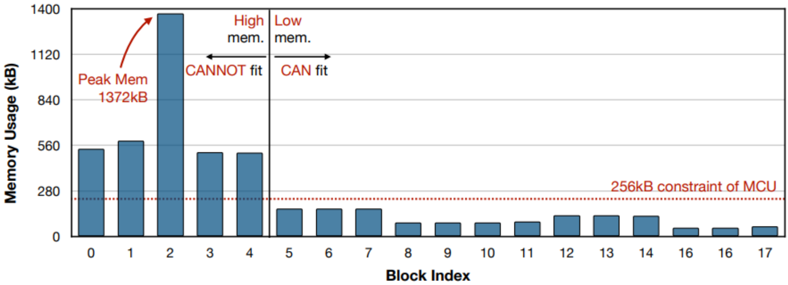


Figure 5: Per-block memory usage of MobileNetV2 [Ji Lin, 2021]

→ What is important is the **Peak memory** !

ML field

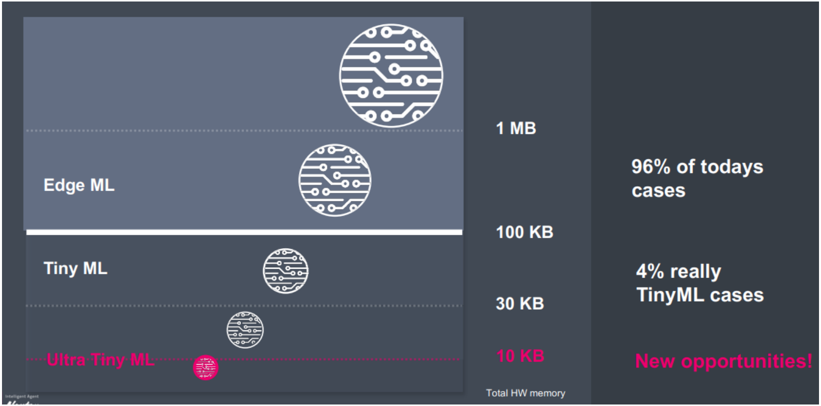


Figure 6: Benchmark of the tiny ML field [NEUTON.AI, 2022]

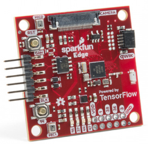
Micro-controllers

- A typical microcontroller system consists of a processor core, an on-chip SRAM block and an on-chip embedded flash
- Constraints
  - Peak memory usage of the model computations $<$ memory usage.
  - Number of parameters in the model $<$ flash memory storage
  - Model size and the peak memory $<$ 250 KB each;
  - CNN computation $<$ 60 million multiply-adds per inference at high accuracy

Comparison between hardwares

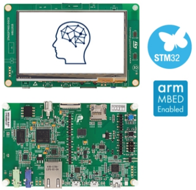
Micro-controller	Price	Memory	Specificities
Arduino Nano 33 BLE Sense	29,70€	256 kB	
SparkFun Edge	$16.50	384kB	
ST Microelectronics STM32F746G Discovery kit	$54.0	340 kB	Screen / included camera

Table 1: Main micro-controllers on the market for tinyML



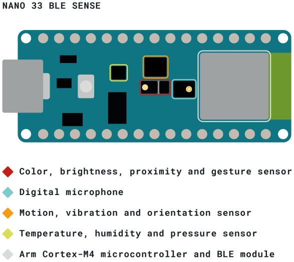
(a) SparkFun Edge

(b) Arduino Nano



(c) ST

Hardware: Arduino Nano 33 BLE Sense



32-bit ARM® Cortex®-M4 CPU running at 64 MHz, 256kB RAM

Arduino Nano 33 BLE Sense: Components

Sensor	Power
IMU	$1mW$
Weather (humidity, and temperature)	$5\mu W$
barometric sensor	10 μW
microphone	$300\mu W$
Gesture, proximity, light	?
Bluetooth® Low Energy connectivity	40 mW

Table 2: Components integrated

Possibility to connect many sensors such as cameras for recognition (1 mW at 30 FPS for 320 $\times$ 320-pixel monochrome image sensor).

TensorFlow Lite

- Memory constraints → How to reduce the size of neural networks (storage and memory usage)?
- TensorFlow Lite → represents the model in the FlatBuffers format (Access to serialized data without parsing/unpacking)
- TensorFlow Lite Converter: converts TensorFlow models to TensorFlow Lite, applies optimizations to reduce the model size
- TensorFlow Lite Interpreter This runs an appropriately converted TensorFlow Lite model using the most efficient operations for a given device.

Quantization

- Quantization: Store weights and compute calculations with fewer bits (INT8)

Power consumption

Significant reduction in energy for both computations and memory access

Add energy (pJ)	
INT8	FP32
0.03	0.9
30X energy reduction	

Mult energy (pJ)	
INT8	FP32
0.2	3.7
18.5X energy reduction	

Mem access energy (pJ)	
Cache (64-bit)	
8KB	10
32KB	20
1MB	100
DRAM	1300-2600
Up to 4X energy reduction	

Latency

With less memory access and simpler computations, latency can be reduced

Silicon area

Integer math or less bits require less silicon area compared to floating point math and more bits

Add area (μm²)	
INT8	FP32
36	4184
116X area reduction	

Mult area (μm²)	
INT8	FP32
282	7700
27X area reduction	

Figure 8: Quantization efficiency [Fournarakis, 2021]

Quantization in practice

FP32 tensor $\rightarrow$ $X \approx s_X X_{\text{int}} = \hat{X}$ $\leftarrow$ scaled quantized tensor

$$W = \begin{pmatrix} 0.97 & 0.64 & 0.74 & \boxed{1.00} \\ 0.58 & 0.84 & 0.84 & 0.81 \\ \boxed{0.00} & 0.18 & 0.90 & 0.28 \\ 0.57 & 0.96 & 0.80 & 0.81 \end{pmatrix} \approx \frac{1}{255} \begin{pmatrix} 247 & 163 & 189 & \boxed{255} \\ 148 & 214 & 214 & 207 \\ \boxed{0} & 46 & 229 & 71 \\ 145 & 245 & 204 & 207 \end{pmatrix} = s_W W_{\text{uint8}}$$

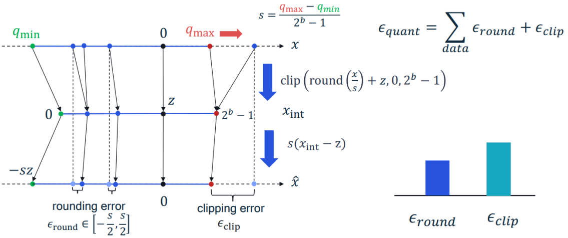


Figure 9: Quantization error [Fournarakis, 2021]

Pruning

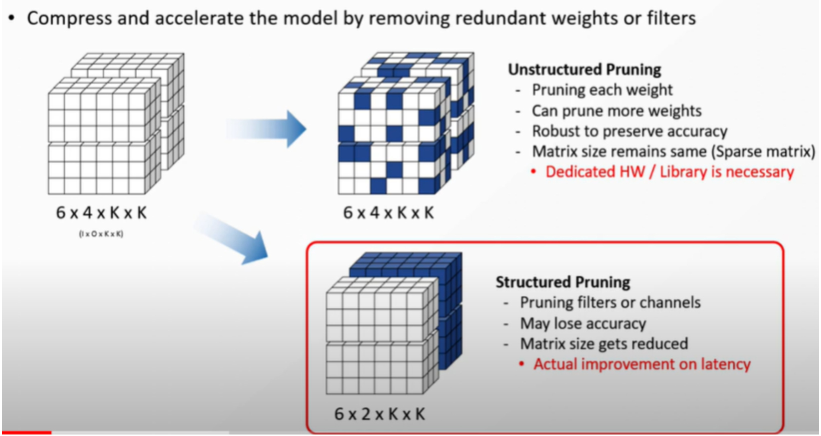


Figure 10: Another method to reduce the size of neural networks is to use pruning [Shim, 2021]

Optimizations

To resume, before sending the model to the micro-controller, three main optimizations must be done:

- Optimizing latency
  - Quantization
  - Hardware changes
  - Optimizing operation
- Optimizing energy usage
  - Measuring Real Power Usage
  - Improving Power Usage (Duty Cycling, Cascading Design)
  - Quantization, pruning...
- Optimizing model and binary size
  - Reducing the size of the executable
  - Pruning, quantization...
  - Code optimization

Training and uploading the tinyML model on the MCU

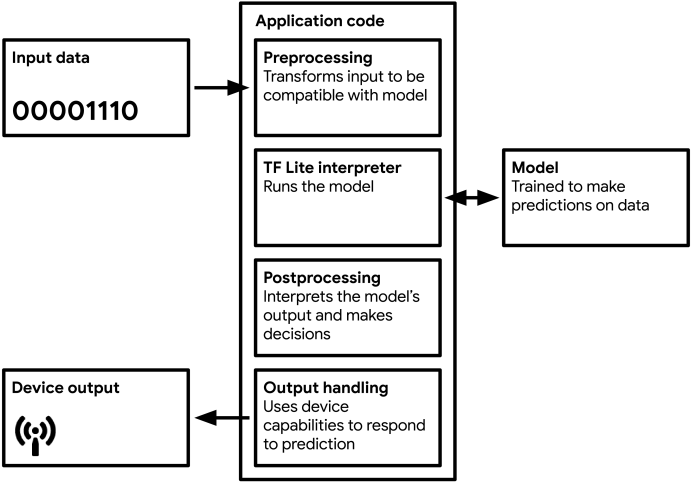
- Training the model before like a classical training with TensorFlow (and Keras). To do so, training data can be generated with Arduino sensors, depending on the use case.
- 1) Convert the model for TensorFlow Lite with the TensorFlow Lite Converter's Python API:
  - (writes Keras model to disk in the form of a FlatBuffer: space-efficient format)
  - Apply optimizations to the model: quantization, pruning...
- 2) Convert the model into a c source file (the extra code required to load a model from disk would be wasteful given our limited space) with xxd

C model

```
unsigned char sine_model_quantized_tflite[] = {
  0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x12,
  0x1c, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00, 0x10, 0x00, 0x14,
  // ...
  0x00, 0x00, 0x08, 0x00, 0x0a, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x04, 0x00, 0x00, 0x00
};
unsigned int sine_model_quantized_tflite_len = 2512;
```

Figure 11: Representation of the model in C [Pete Warden, 2019]

Prediction with TensorFlow Lite



Applications of tinyML

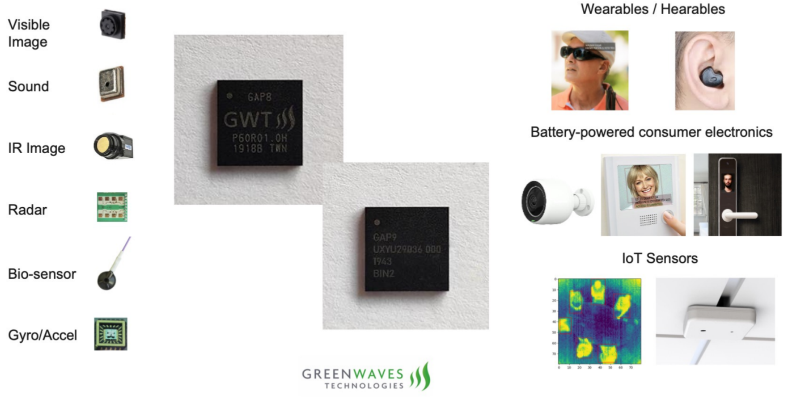


Figure 12: Different possible applications of tinyML [Fournarakis, 2021]

Wake word detection

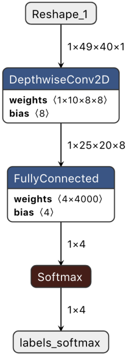


Figure 13: Convolutional Neural network [Pete Warden, 2019]

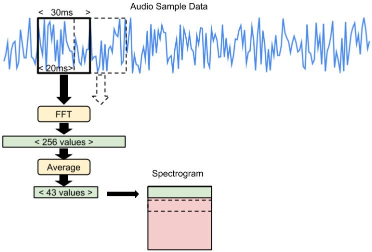


Figure 14: Trained on Speech Commands dataset (65,000 one-second-long utterances of 30 short words)
[Pete Warden, 2019]

Vision with nano controllers

- Visual WakeWords: Person/Not-Person, Object counting, Object localization:

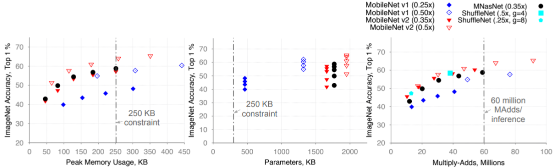


Figure 15: ImageNet dataset [Aakanksha Chowdhery, 2019]

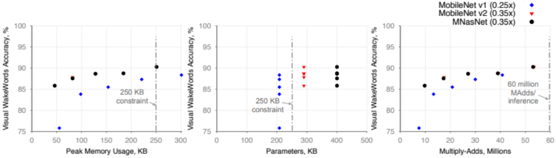


Figure 16: Visual Wake Words dataset [Aakanksha Chowdhery, 2019]

Person detection with Arduino

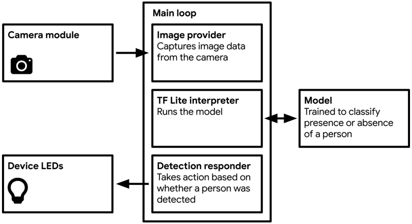


Figure 17: Example with Arduino [Pete Warden, 2019]

- Data acquisition with Arducam Mini 2MP Plus (25.99 dollars) (1920×1080)
- Resized to 160×120 pixels
- Converted into grayscale
- Model: mobilenet v1 (smallest amount of RAM at runtime)

Patch-Based Inference

- Constatation: Imbalanced Memory Distribution of CNNs → Needs for more memory efficient CNN
- Better method for inference with convolutional neural networks developed by Ji Lin et al (MIT) [Ji Lin, 2021]
- "Unlike conventional layer-by-layer execution, it operates on a small spatial region of the feature map at a time, instead of the whole activation"

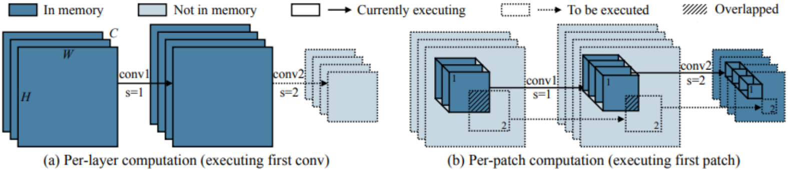


Figure 18: Patch-based vs Per-layer computation [Ji Lin, 2021]

Peak memory reduced: per-layer

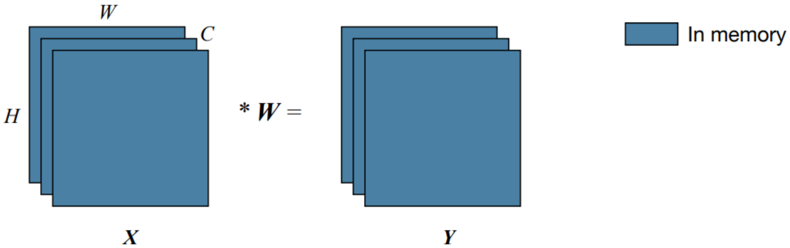


Figure 19: With a classical per-layer computation, the memory is filled with all the filters. [Ji Lin, 2021]

Peak Mem = 2 WHC

Peak memory reduced: per-batch

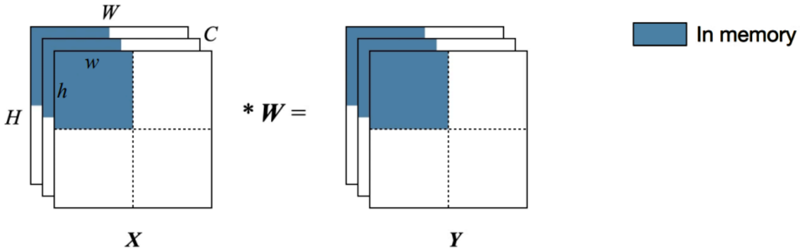


Figure 20: On the other hand, with a per-path computation, the peak memory is reduced as only a part of each filter is stored. [Ji Lin, 2021]

Peak Mem = 2 whC

Example with 2 layers

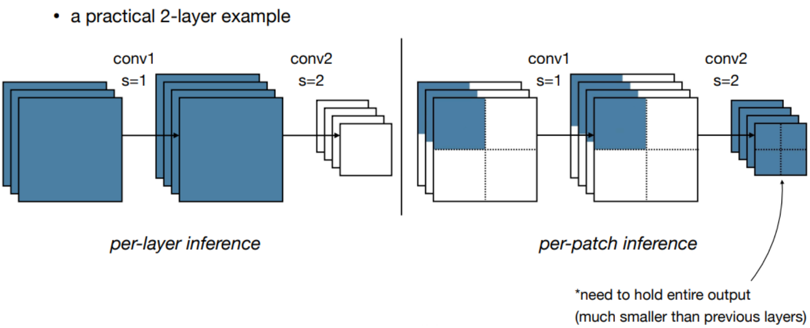


Figure 21: Comparison with 2 layers [Ji Lin, 2021]

Problem with overlapping

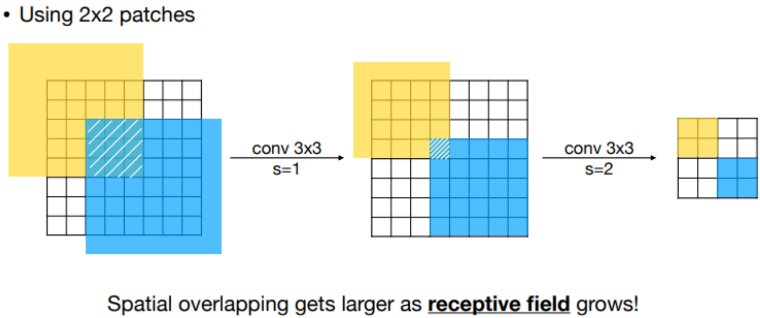


Figure 22: Overlapping increases the overall computation ($+10\%$ while reducing the peak memory [Ji Lin, 2021]

MCUNetV2

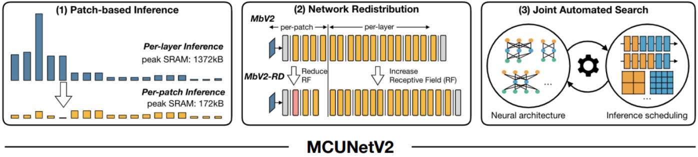


Figure 23: Final implementation of Patch-Based Learning [Ji Lin, 2021]

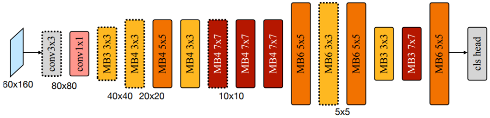


Figure 24: MCUNetV2 architecture [Ji Lin, 2021]

Results

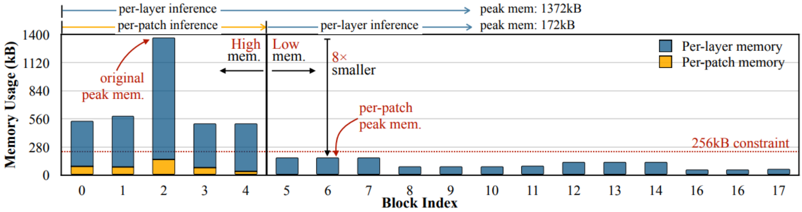
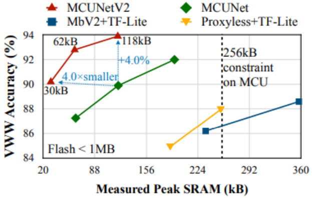


Figure 25: Memory usage results with patch-based inference [Ji Lin, 2021]



Conclusion

- TinyML: very dynamic field
- Relatively easy to deploy a ML algorithm on a micro-controller with TensorFlow lite (a lot of documentation)
- A lot of applications (for environmental studies also)
- Main problem: peak-memory overflow → ideas like MCUNETV2 and patch-based inference

Thanks for listening

Thanks for listening ! Any questions ?

References I

[Aakanksha Chowdhery, 2019] Aakanksha Chowdhery, Pete Warden, J. S. A. H. R. R. (2019).
Visual wake words dataset.

[Fournarakis, 2021] Fournarakis, M. (2021).
A practical guide to neural network quantization.

[Han, 2021] Han, S. (2021).
Putting ai on a diet: Tinyml and efficient deep learning.

[Ji Lin, 2021] Ji Lin, Wei-Ming Chen, H. C. C. G. S. H. (2021).
Mcunetv2: Memory-efficient patch-based inference for tiny deep learning.

References II

[NEUTON.AI, 2022] NEUTON.AI (2022).
A novel approach to building exceptionally tiny models without loss of accuracy.

[Pete Warden, 2019] Pete Warden, D. S. (2019).
Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers.

[Shim, 2021] Shim, K.-H. (2021).
Toward compact deep neural networks via energy-aware pruning.