Applying Artificial Intelligence on Edge devices using Deep Learning with Embedded optimizations

VLAIO TETRA HBC.2019.2641

User group meeting 3 29-06-2021

ai-edge.be

iot-incubator.be www.eavise.be

Agenda

1. Introduction
2. Overview of frameworks
3. Academic use case
4. Use cases by the user group
    1. E.D.&A.
    2. TML & Digipolis
    3. Melexis
5. Workshop
6. VLAIO
7. Networking

Neural network deployment

- Overview
- Design space model
- Examples

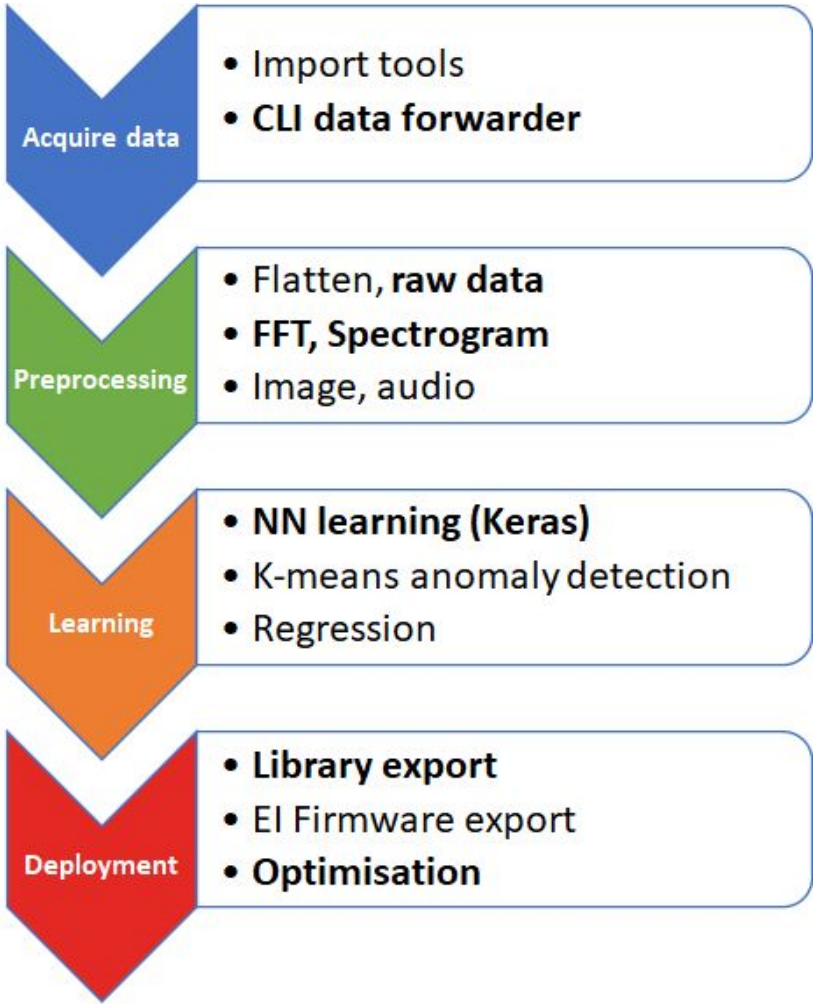
Neural network deployment

Edge Impulse flow:

Preprocessing:

- Example Academic use case

Deployment



Neural network deployment

- Deployment possibilities in Edge Impulse
- 2 options:
  - Choose “library” (framework)

    → Download library for your hardware
  - Choose “firmware” (device)

    → Acquires data via data forwarder (CLI tool from Edge Impulse), runs network on device, reports back to online tool

Neural network deployment

Library deployment in Edge Impulse

- Frameworks:
  - C++ & Arduino (TFLite)
  - Cube.MX CMSIS Pack
  - WebAssembly (node.js)
  - TensorRT (Nvidia GPU)
- Our Target:
  - Sensortile
  - STM32L476JGY (Cortex M4)
  - 80 MHz
  - 128 KB RAM
  - 1 MB Flash

Neural network deployment

Library deployment in Edge Impulse

- Model:
  - TFLite model (2D conv)
- Possible optimizations:
  - Quantizing 32 bit float → 8 bit int
  - EON Compiler (Edge Optimized Neural)
  - Compared to TFLite for µC compiler:
    - 25-55% less RAM, 35% less Flash
    - Same accuracy
    - Compiles model to C++
    - No interpreter necessary
    - Not loading model @ runtime

Arduino library

Neural network deployment

Library deployment in Edge Impulse

- Results with (inference only)
  - Quantization: 32 bit float → 8 bit int
  - EON Compiler

C++ library

Cube.MX CMSIS-PACK

Arduino library

Compiler	Quantisation	Accuracy (%)	Latency (ms)	RAM usage (kB)	Flash usage (kB)
Normal	32 bit float	85.5	165	90.4	65.4
	8 bit int	85.5	35	28.3	53.2
EON	32 bit float	85.5	165	73.0	46.6
	8 bit int	85.5	35	20.6	33.5

User group interaction

Questions / remarks?

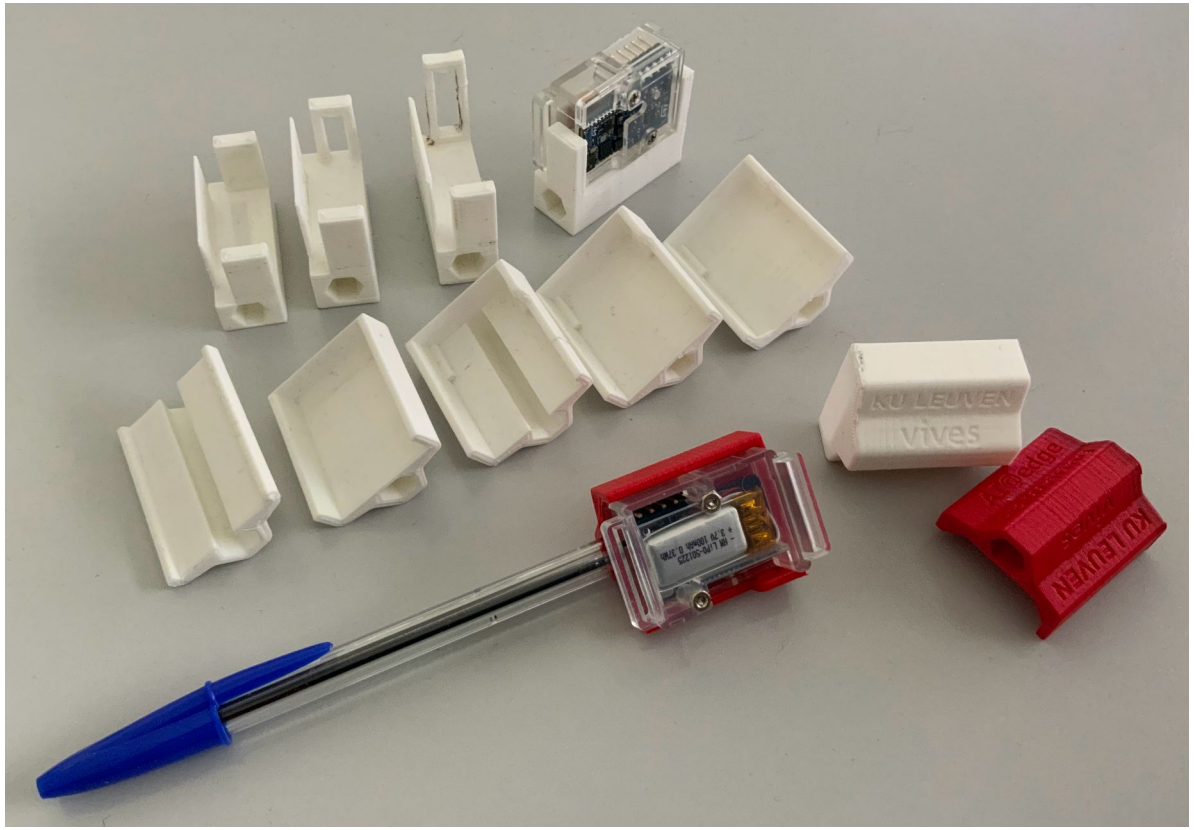
Update Academic use-case

Detection of writing A or B by using the 3-axis accelerometers of an STM Sensortile attached to a pen

Update Academic use-case: Overview

1. Hardware update
2. Data acquisition update
3. Analysis on optimal inference model
   a. No pre-processing (raw data)
   b. Spectral analysis - FFT length
   c. Spectral analysis - LPF
   d. Spectral analysis - HPF
   e. Spectrogram - 1D conv
   f. Spectrogram - 2D conv
4. Conclusion & overall best

Update Academic use-case: Hardware



- Design of Sensortile & pen holder
- Several iterations
- Click & slide mechanism
- Final 3D printed product

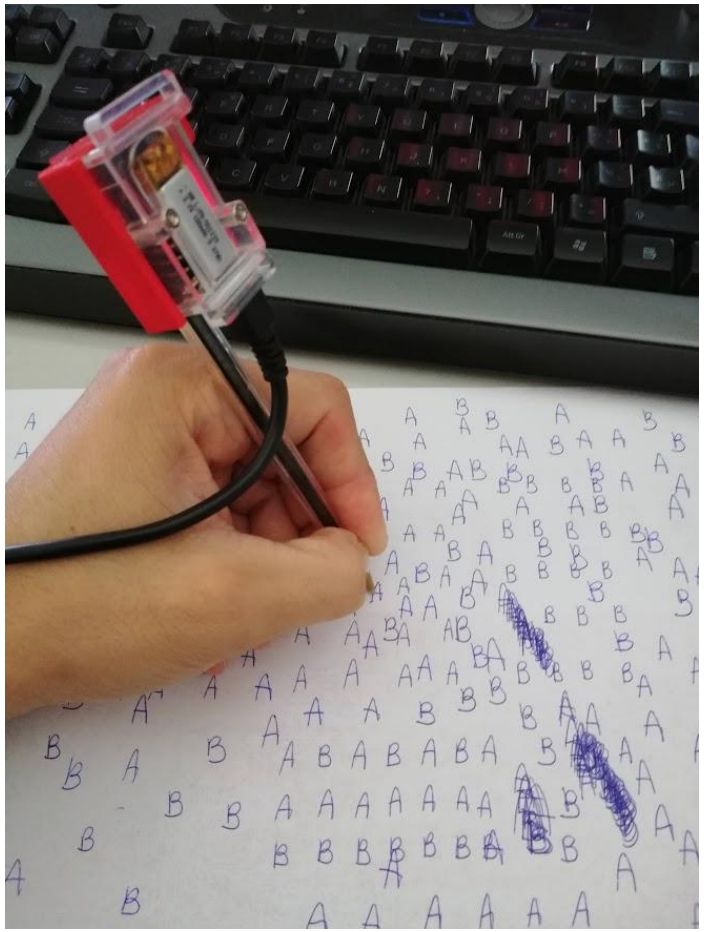
- Allows access to:
  - On/off switch
  - Micro-USB for charging & UART
  - SWD-pins (Serial Wire Debug)
  - SD-card slot

Update Academic use-case: Data

- Acquiring more training & test data
- During student project-week
  - Colleagues
  - Students

- Old dataset:
  - A-Left
  - B-Left
  - Idle

- New dataset:
  - A-Left
  - A-Right
  - B-Left
  - B-Right
  - Idle



Update Academic use-case: Analysis

- Analysis on optimal inference model

- Preprocessing: YES/NO?
- Fully interconnected (dense) layer: how many neurons?

- Which combination yields the best result?
  - Minimal loss
  - Maximal accuracy
  - Latency/RAM usage

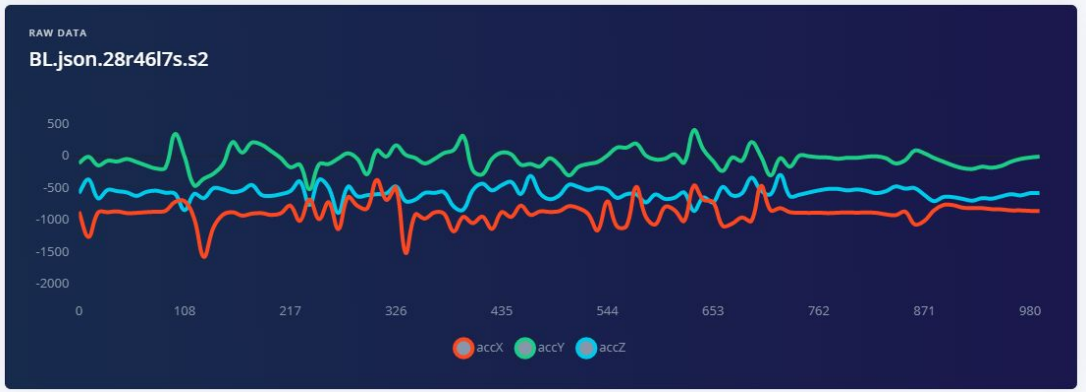
Update Academic use-case

Recap of the acquired data: (input)

One letter equals:

- Measurement for 1 second @ 100 Hz (101 datapoints)
- 3-axis accelerometer data (3x101 datapoints)
- Total = 303 datapoints (inputs)

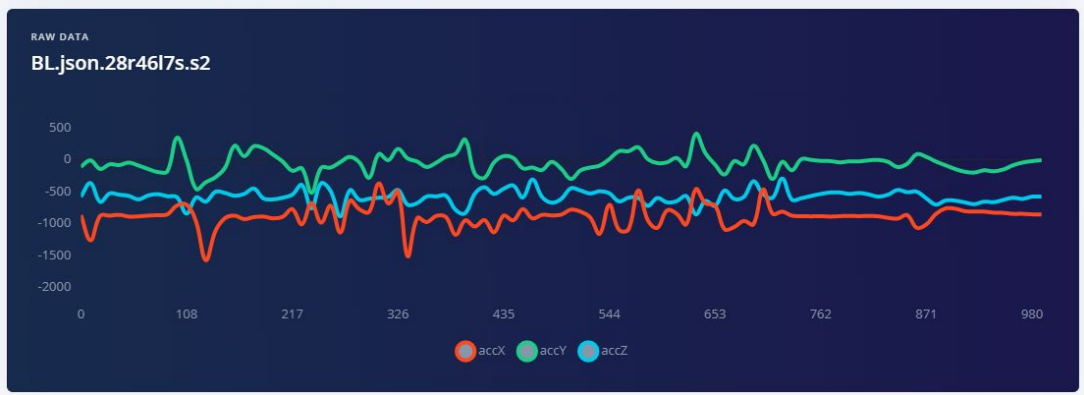
Goal: classify to 5 labels (outputs)

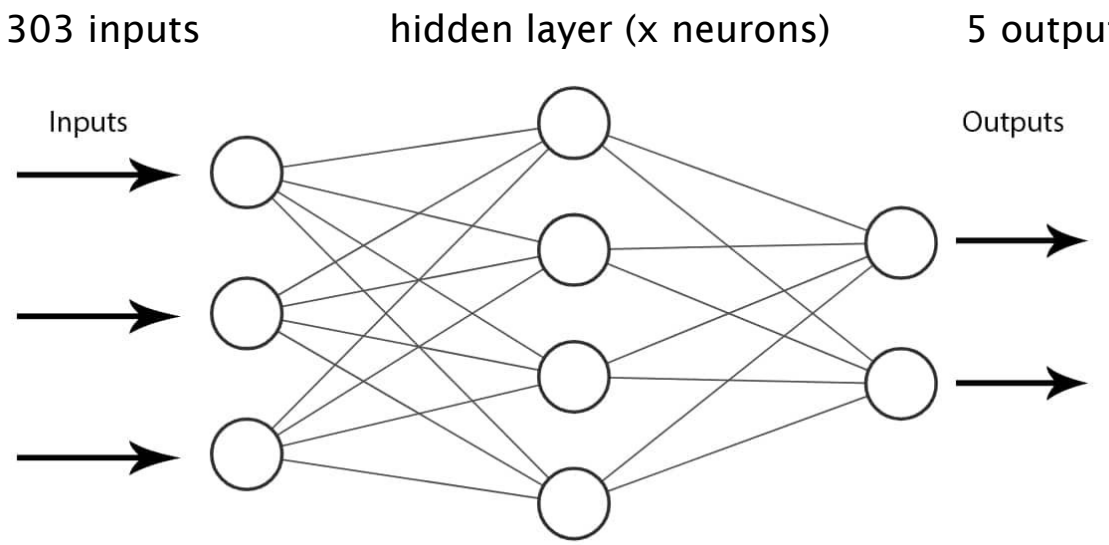


Update Academic use-case: Raw data

1. Raw Data (no preprocessing)

Network topology: FFNN





Update Academic use-case: Raw data

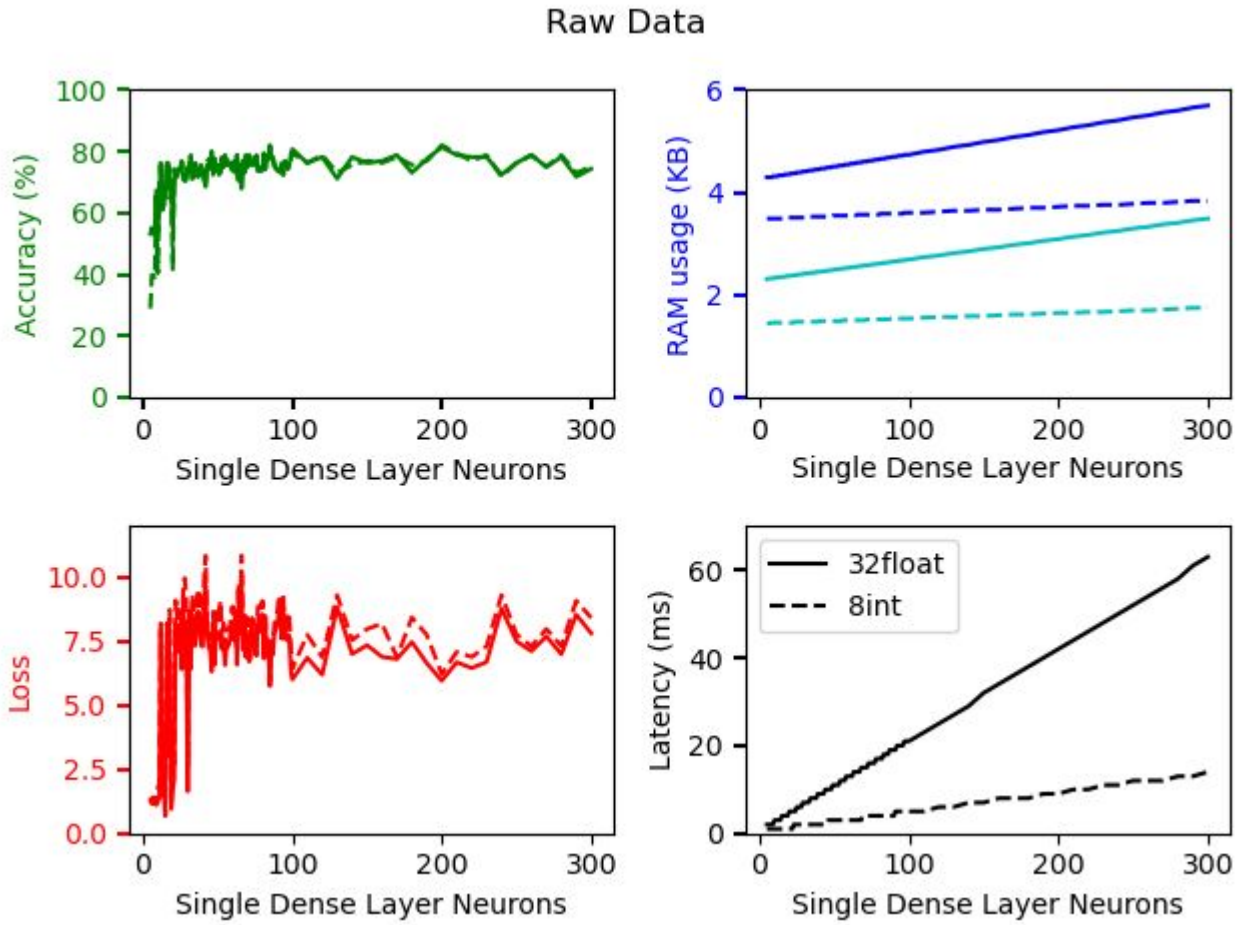
x-axis: number of neurons

y-axes: performance metrics

Full lines: 32 bit float

Dashed lines: 8 bit int (quantized)

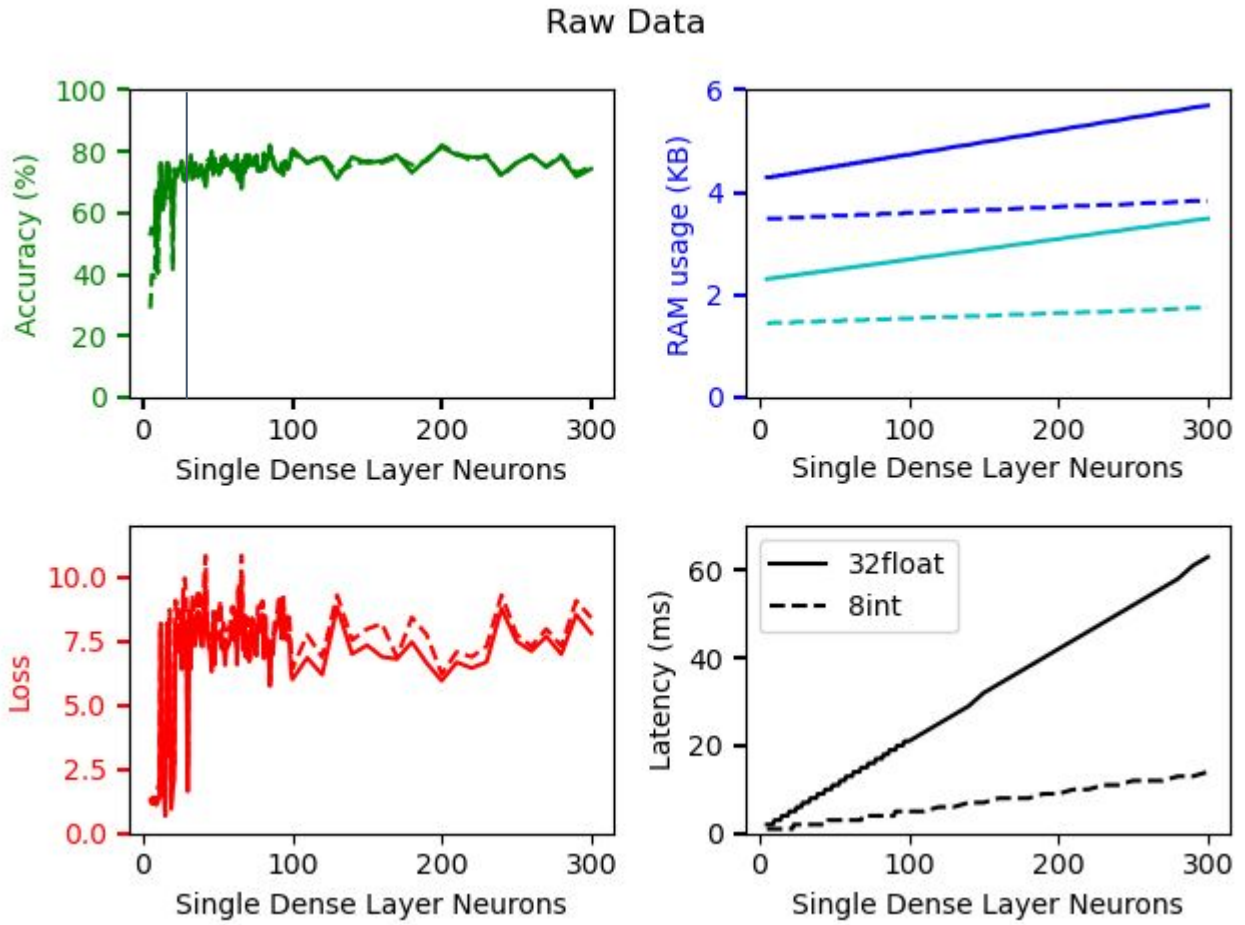
Cyan: RAM with EON compiler (accuracy, loss, latency equal)



Update Academic use-case: Raw data

Raw Data Conclusions

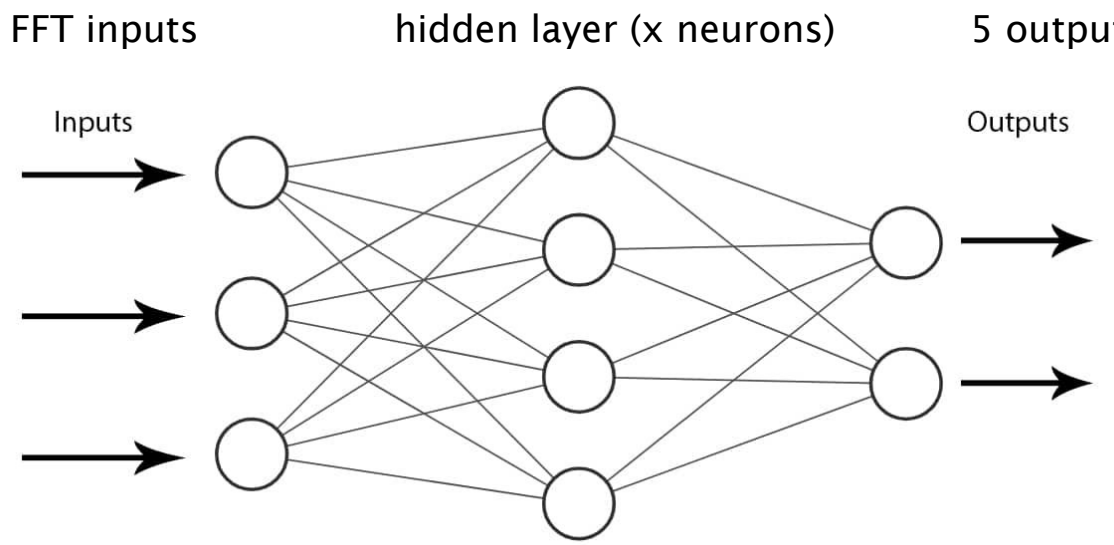
- Min. 30 neurons
- Accuracy ~80%
- Loss = high
- RAM: within specs
  - EON compiler: < 2 kB
- Latency (7 ⟶ 2 ms)

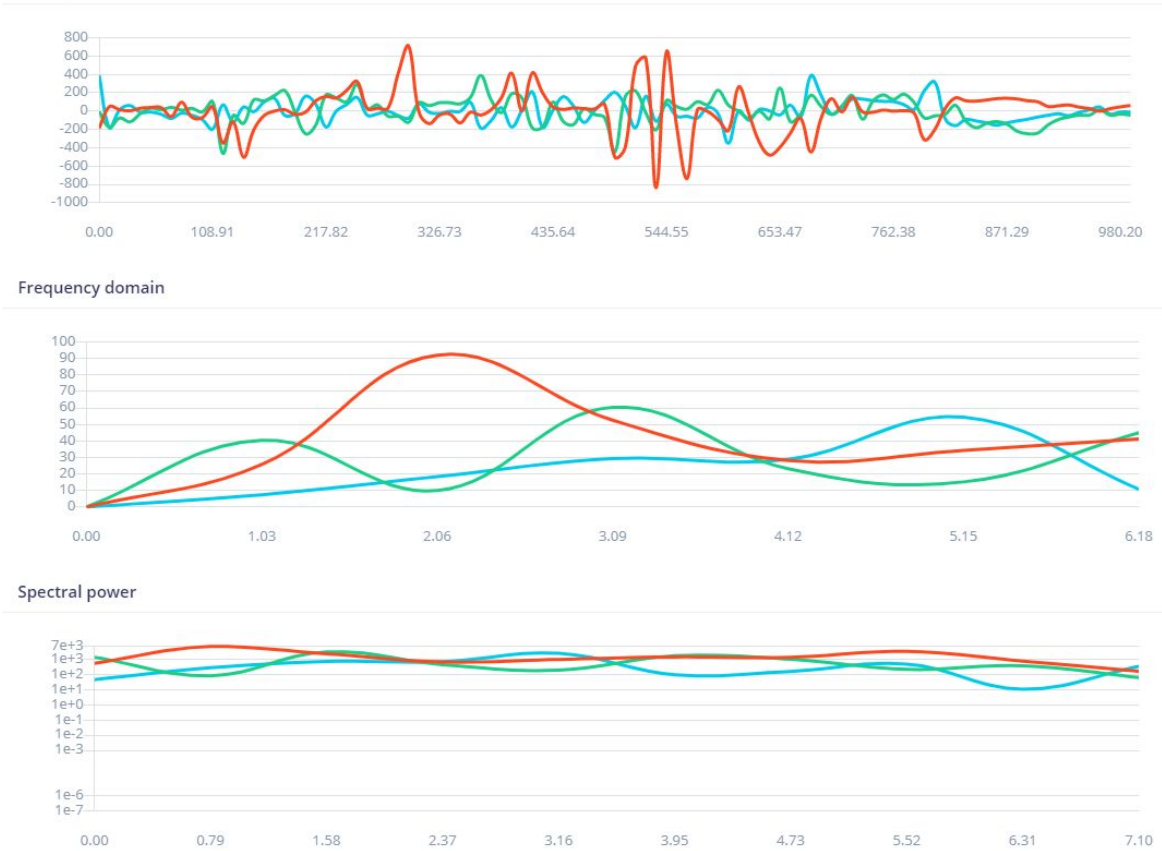


Update Academic use-case: Spectrum

2. Spectral Analysis (FFT)

Network topology: FFNN





Update Academic use-case: Spectrum

2. Spectral Analysis

<u>Options in Edge Impulse:</u>

- Apply filter: None, LPF, HPF
- Cut-off frequency
- Filter order
- FFT length (power of 2)

<u>Extracted features:</u>

- RMS value after filter
- N peaks (value & frequency)
- (M-1) Power buckets
  - with M edges

Total features: 1+2N+(M-1)

Update Academic use-case: Spectrum

2. Spectral Analysis

<u>Options in Edge Impulse:</u>

- Apply filter: **None**, LPF, HPF
- Cut-off frequency
- Filter order
- **FFT length (power of 2)**

<u>Extracted features:</u>

- RMS value after filter
- N peaks (value & frequency)
- (M-1) Power buckets
  - with M edges

Total features: 1+2N+(M-1)

Update Academic use-case: Spectrum

2. Spectral Analysis

No filter, 5 peaks, 5 buckets

⟶ 48 features (inputs)

FFT length 64

x-axis: number of neurons

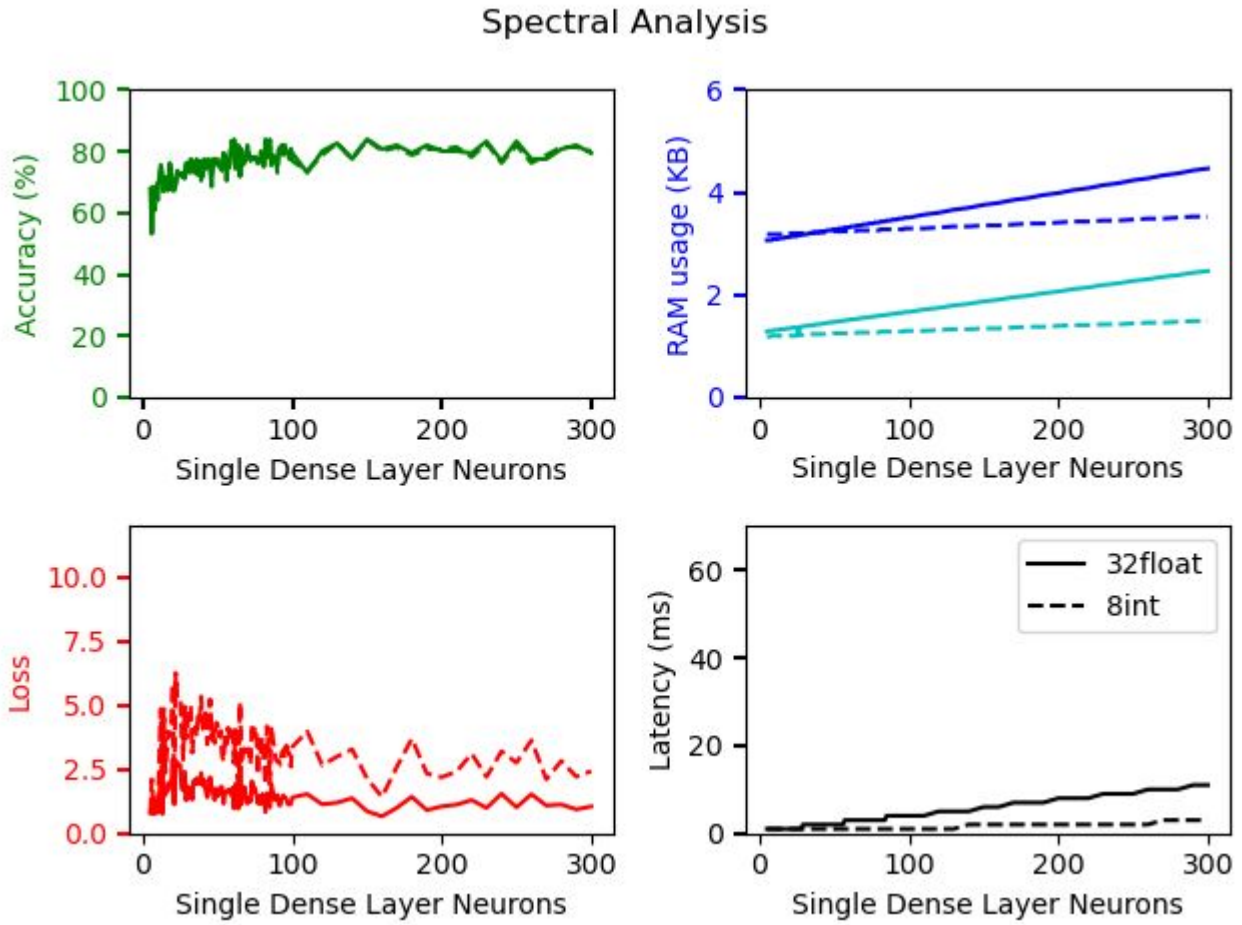
y-axes: performance metrics

Full lines: 32 bit float

Dashed lines: 8 bit int (quantized)

Cyan: RAM with EON compiler

(accuracy, loss, latency equal)



Update Academic use-case: Spectrum

2. Spectral Analysis

No filter, 5 peaks, 5 buckets

⟶ 48 features (inputs)

FFT length 128

x-axis: number of neurons

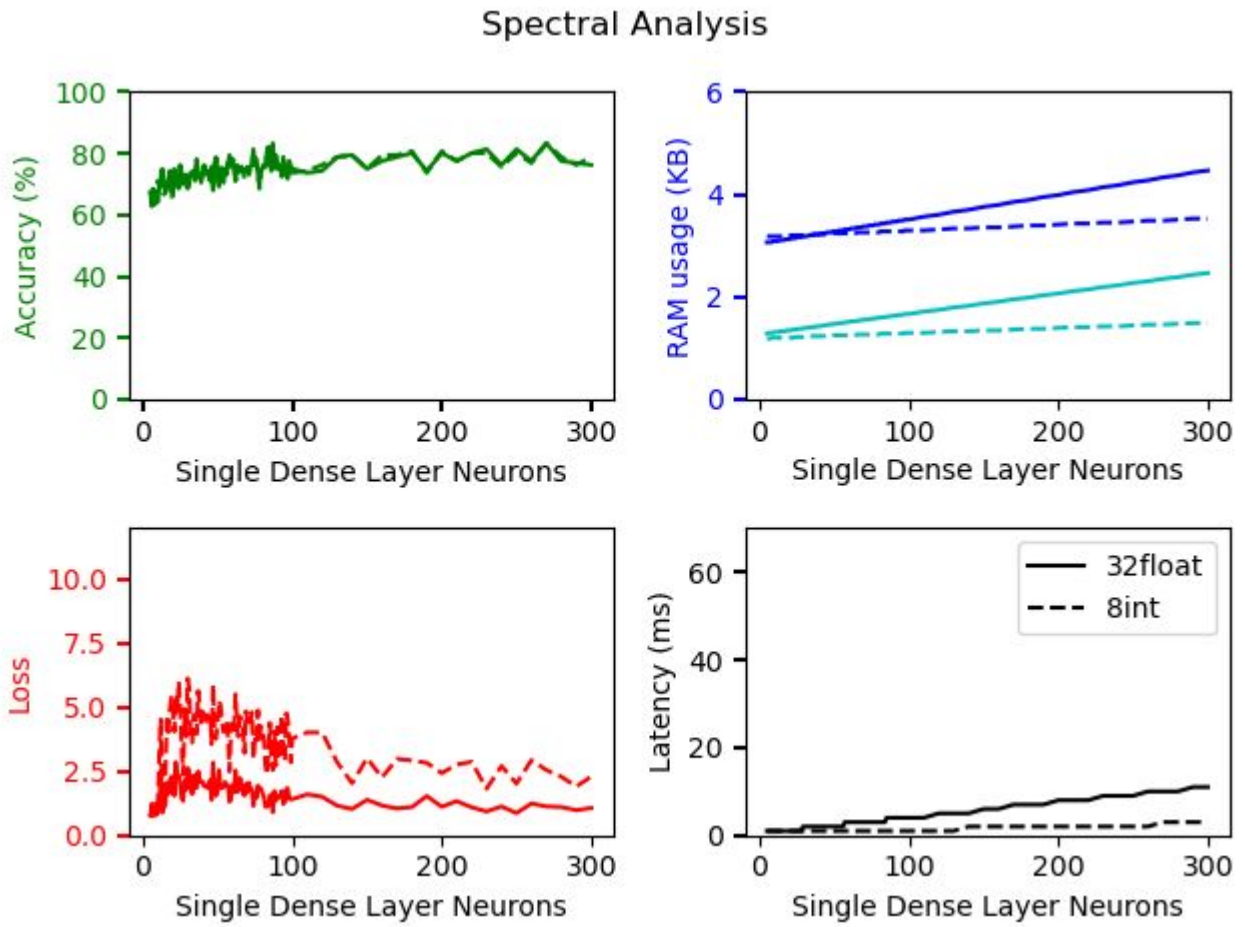
y-axes: performance metrics

Full lines: 32 bit float

Dashed lines: 8 bit int (quantized)

Cyan: RAM with EON compiler

(accuracy, loss, latency equal)



Update Academic use-case: Spectrum

2. Spectral Analysis

No filter, 5 peaks, 5 buckets

⟶ 48 features (inputs)

FFT length 1024

x-axis: number of neurons

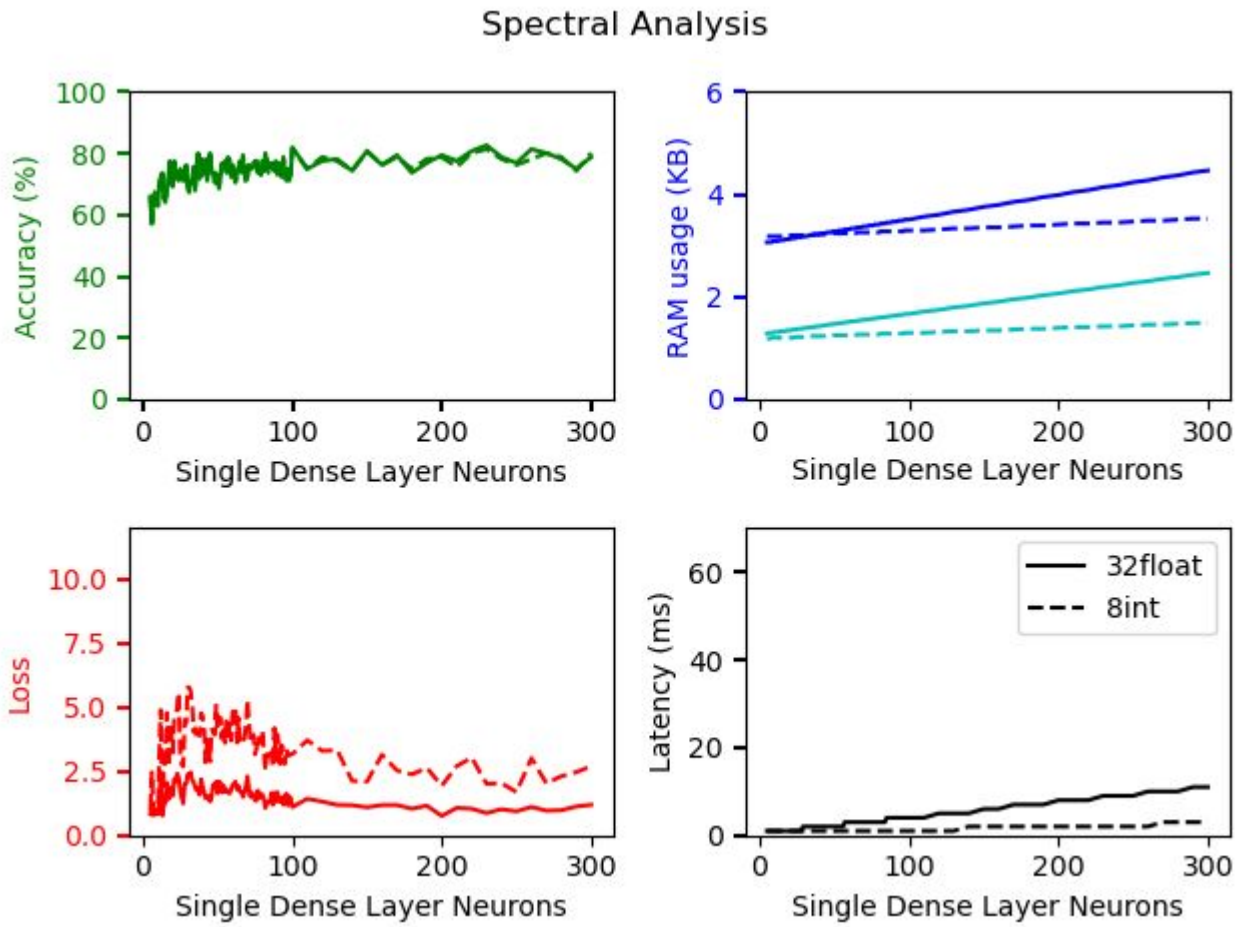
y-axes: performance metrics

Full lines: 32 bit float

Dashed lines: 8 bit int (quantized)

Cyan: RAM with EON compiler

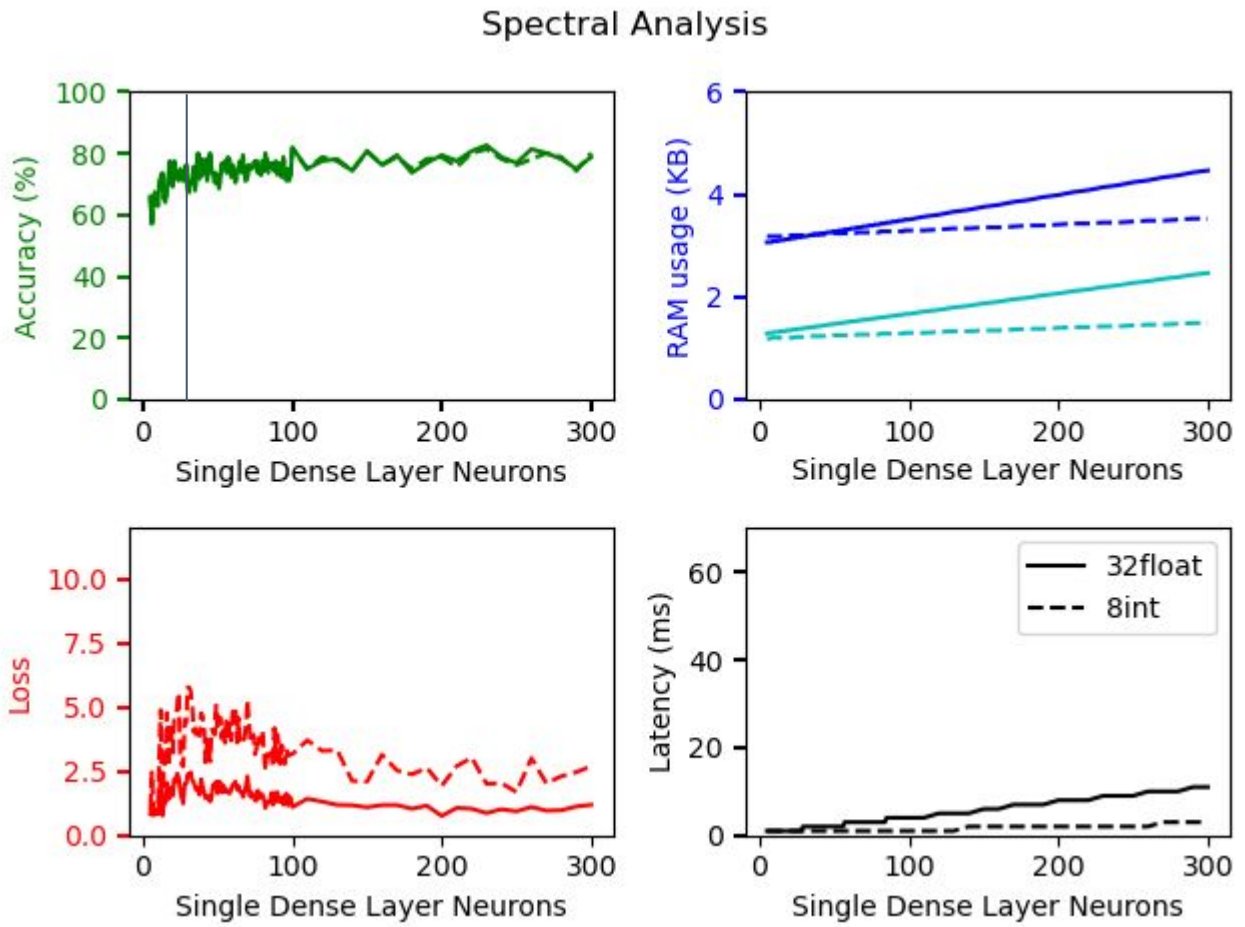
(accuracy, loss, latency equal)



Update Academic use-case: Spectrum

FFT Length Conclusions:

- Length: almost no effect
- Less inputs
  - ⟶ Lower latency
- 20-30 neurons
- Accuracy ~80%
- Loss = lower
  - Compared to raw data
- RAM: within specs
  - EON compiler: < 1.5 kB
- Latency (2 ⟶ 1 ms)



Update Academic use-case: Spectrum

2. Spectral Analysis

Options in Edge Impulse:

- Apply filter: None, **LPF, HPF**
- Cut-off frequency
- Filter order → **6**
- FFT length → **128**

Extracted features:

- RMS value after filter
- N peaks (value & frequency)
- (M-1) Power buckets
  - with M edges

Total features: 1+2N+(M-1)

Update Academic use-case: Spectrum

2. Spectral Analysis

LPF, 5 peaks, 5 buckets

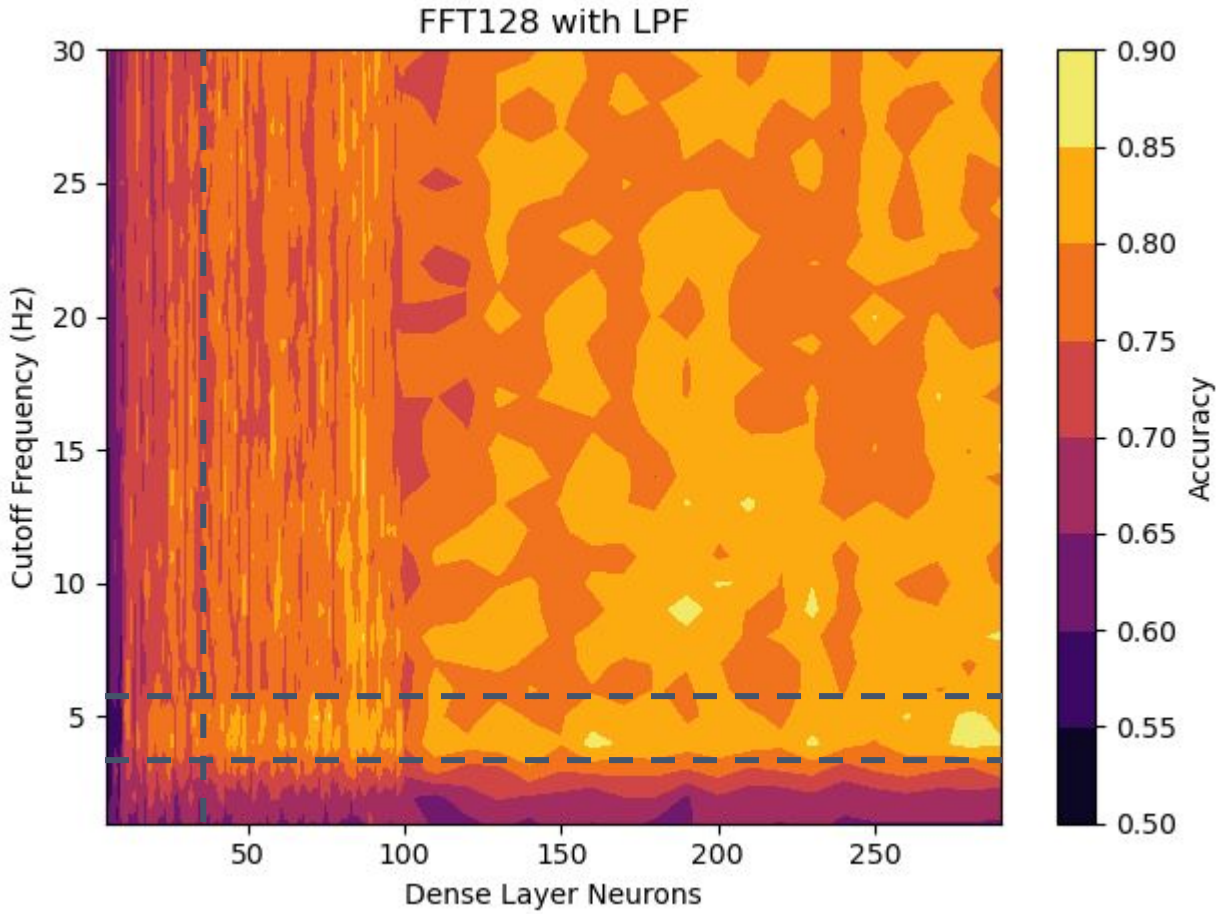
⟶ 48 features (inputs)

FFT length 128

32 bit float (not quantized)

x-axis: number of neurons

y-axes: cutoff frequency (Hz)



Update Academic use-case: Spectrum

2. Spectral Analysis

LPF, 5 peaks, 5 buckets

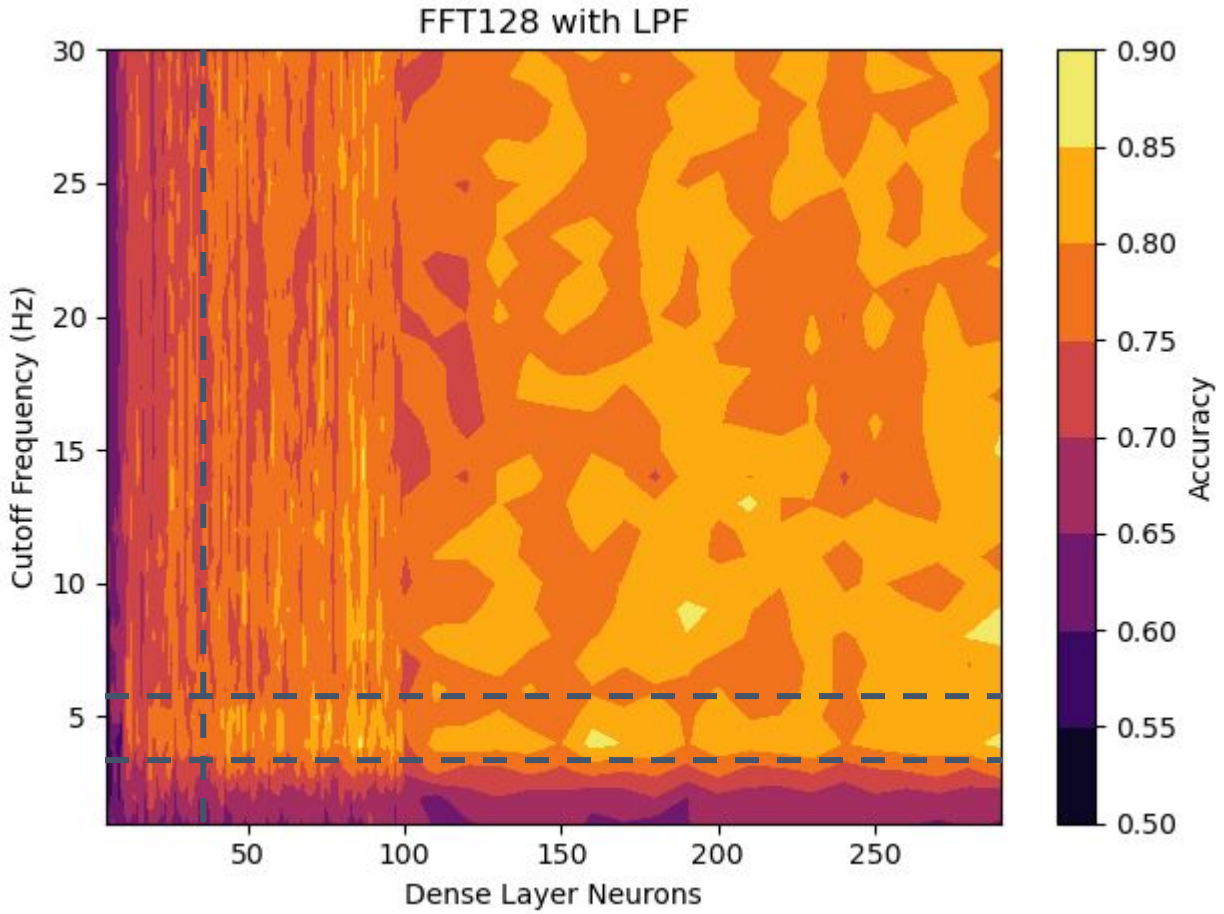
⟶ 48 features (inputs)

FFT length 128

8 bit int (quantized)

x-axis: number of neurons

y-axes: cutoff frequency (Hz)



Update Academic use-case: Spectrum

2. Spectral Analysis

HPF, 5 peaks, 5 buckets

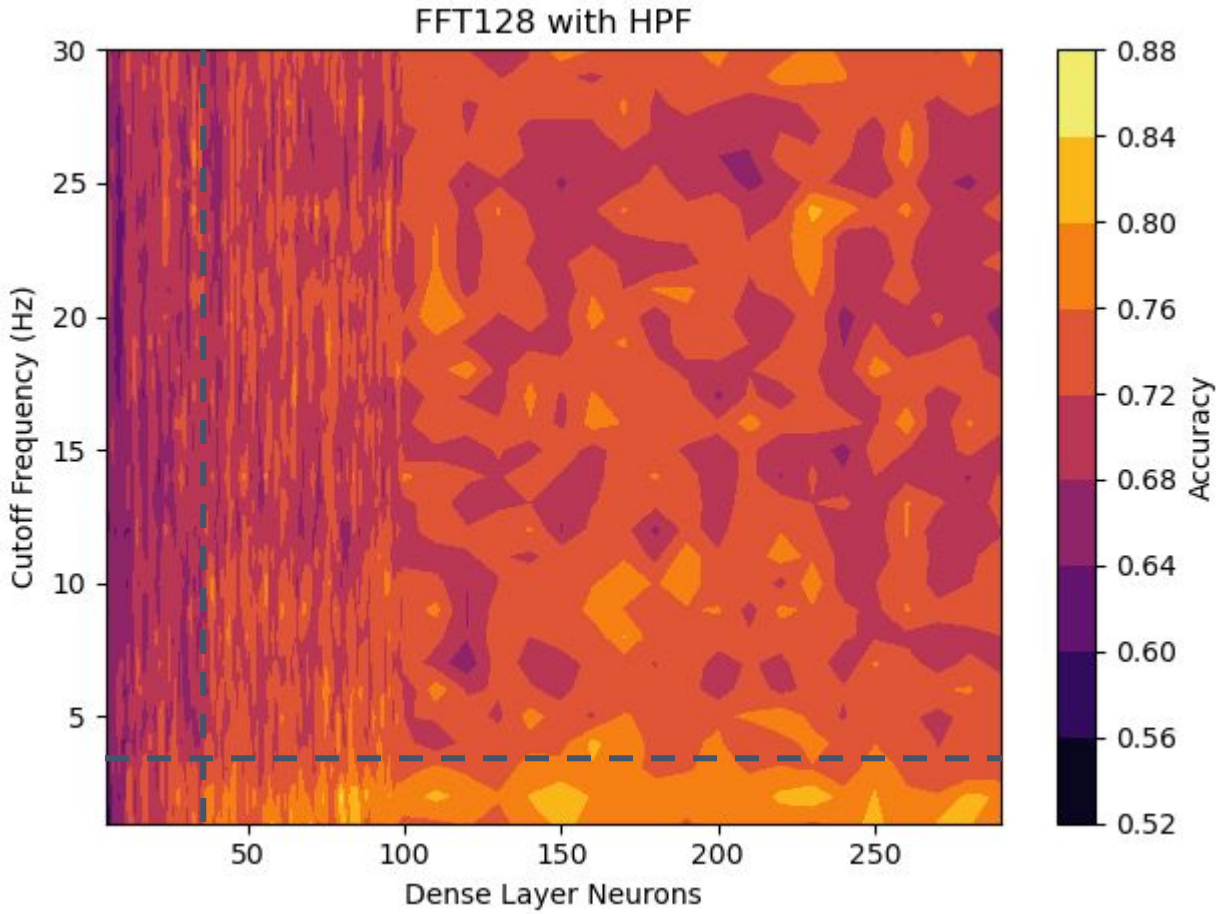
$\longrightarrow$ 48 features (inputs)

FFT length 128

32 bit float (not quantized)

x-axis: number of neurons

y-axes: cutoff frequency (Hz)



Update Academic use-case: Spectrum

2. Spectral Analysis

HPF, 5 peaks, 5 buckets

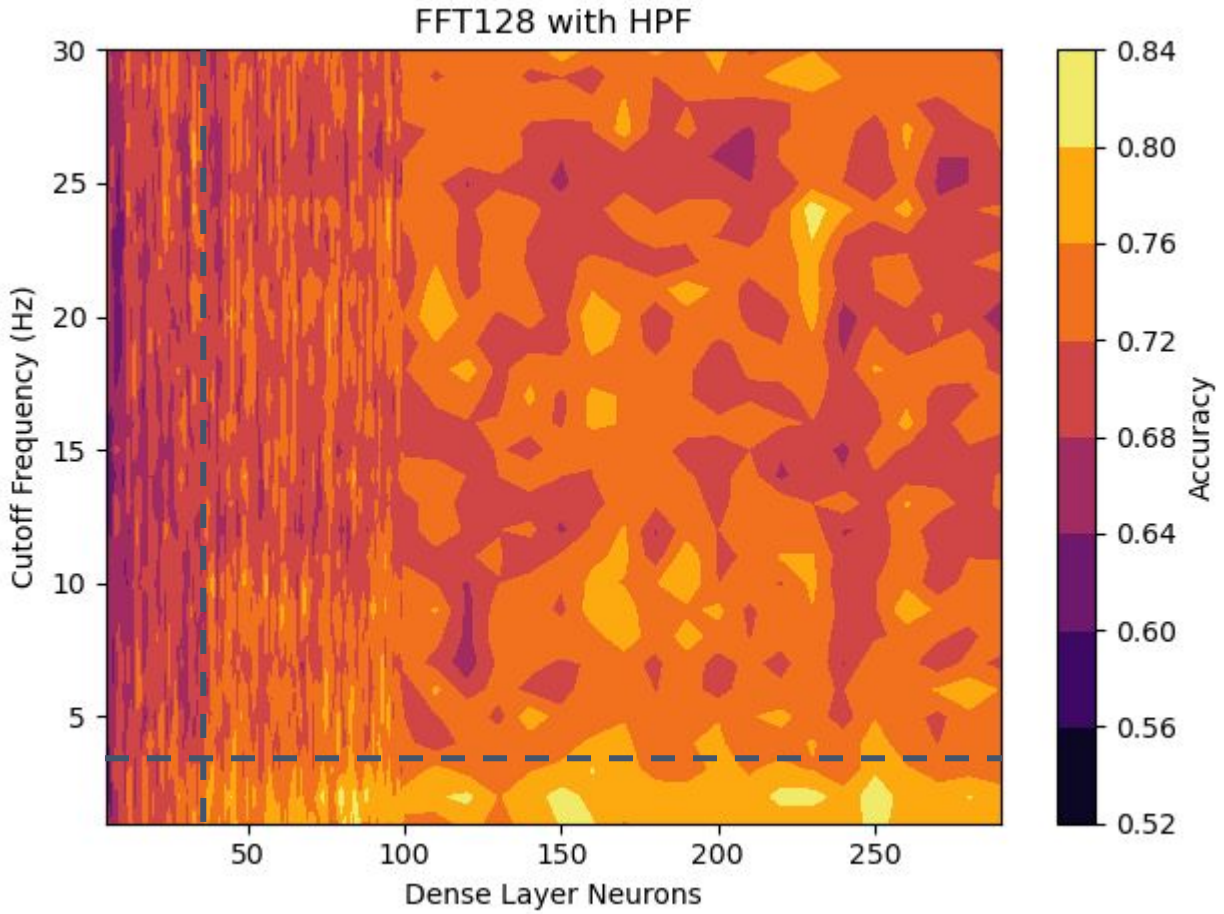
⟶ 48 features (inputs)

FFT length 128

8 bit int (quantized)

x-axis: number of neurons

y-axes: cutoff frequency (Hz)



Update Academic use-case: Spectrum

2. Spectral Analysis

Other performance metrics:

- Latency: Independent of cut-off frequency (~1-2 ms)
- Ram usage: similar as other spectral analysis performance (~2 kB)

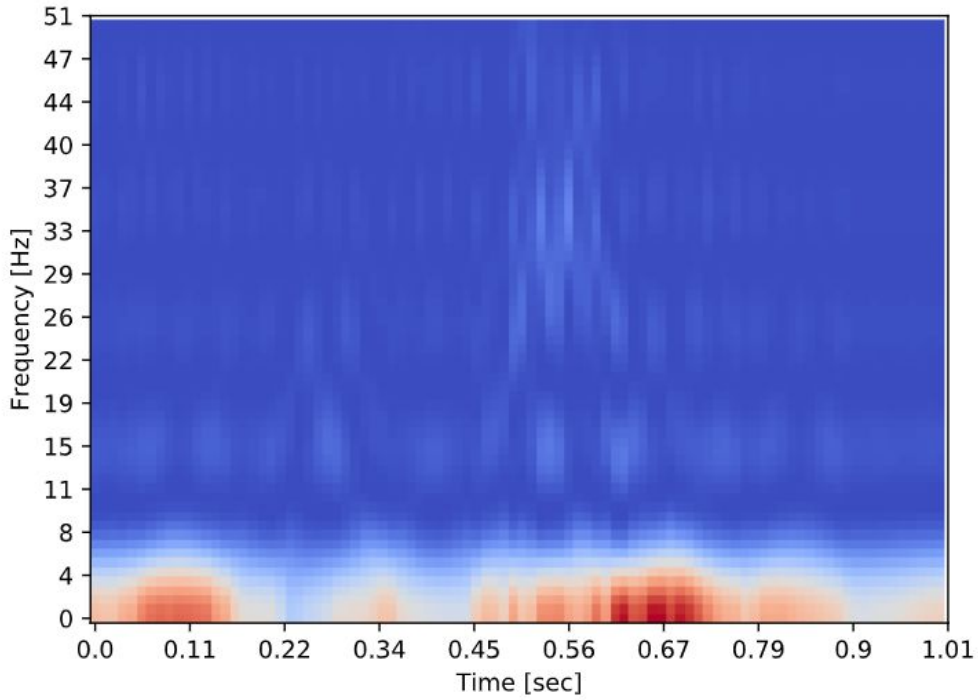
Conclusions:

- LPF better for this use-case (info in lower frequencies)
- Filter necessary to reach higher accuracies: > 80%

Update Academic use-case: Spectrogram

3. Spectrogram = time and frequency information

- Split up time domain in frames
- Take FFT per time frame



Number of network input features:

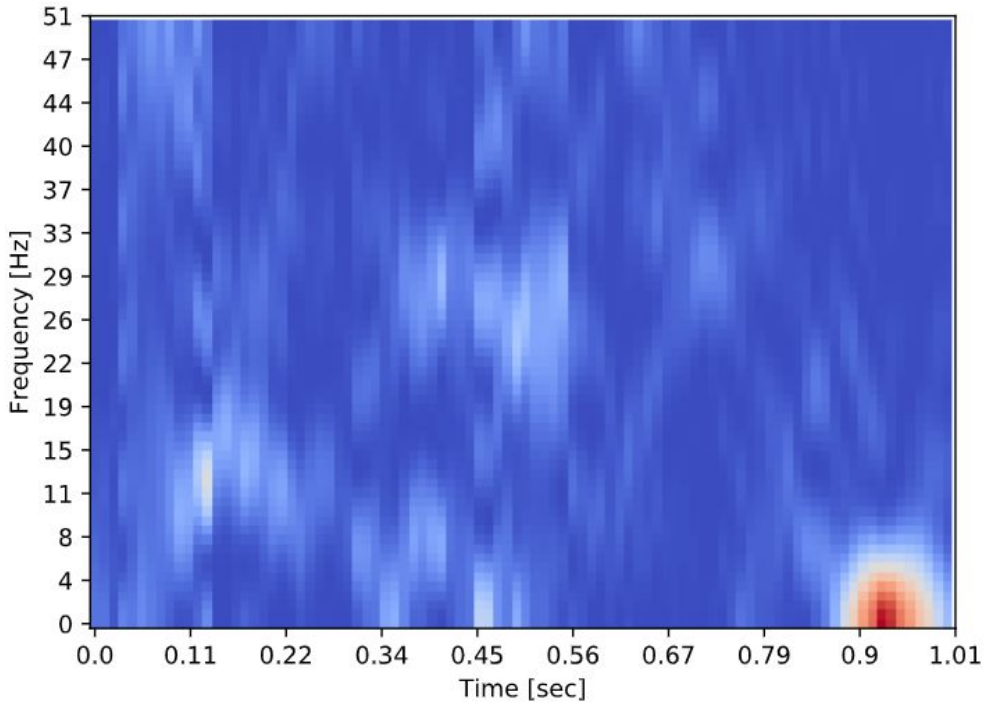
- FFT L:128, 92 frames
- 5980 inputs per accelerometer axis!
- 17940 inputs total

Update Academic use-case: Spectrogram

3. Spectrogram = time and frequency information

- Large number of inputs
- Single dense layer: not sufficient
- Solution
  - 1D convolutional architecture
  - 2D convolutional architecture

Commonly used on images



Update Academic use-case: Spectrogram

3. Spectrogram = time and frequency information

Results (with EON compiler, 8 bit quantisation):

Architecture	Accuracy (%)	Loss	Latency (ms)	RAM usage (kB)	Flash usage (kB)
1D convolutional	84.8	0.39	70	38.2	36.2
2D convolutional	84.2	0.82	948	178.2	121.3

Confusion matrix:

	AL	AR	BL	BR	IDLE
AL	61.1%	16.7%	16.7%	0%	5.6%
AR	0%	63.2%	5.3%	31.6%	0%
BL	9.5%	0%	81.0%	9.5%	0%
BR	5.9%	29.4%	5.9%	58.8%	0%
IDLE	0%	0%	0%	0%	100%
F1 SCORE	0.69	0.62	0.79	0.57	0.99

Update Academic use-case

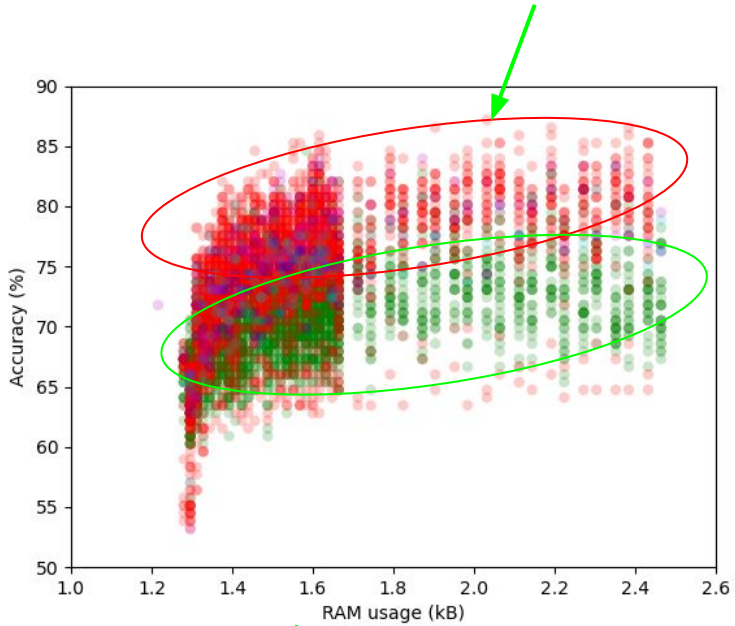
Overall comparison

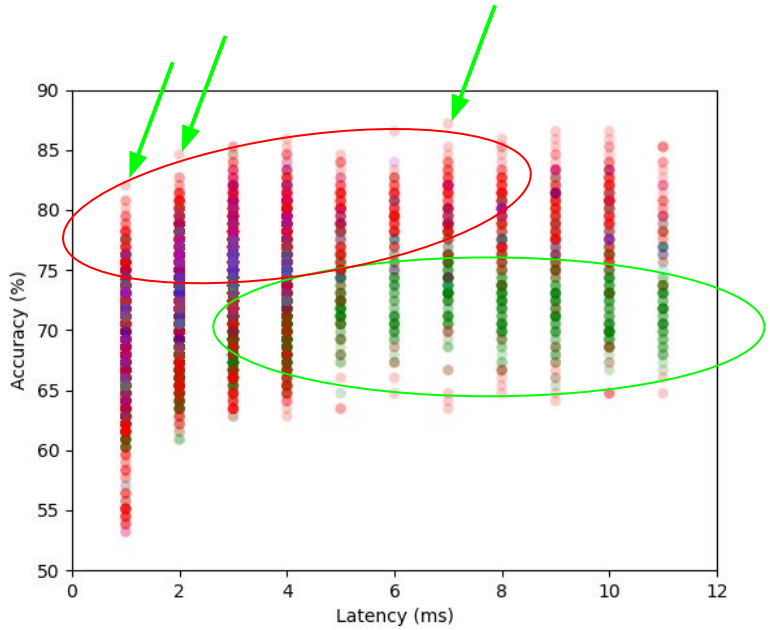
Y-axis: Accuracy (%)

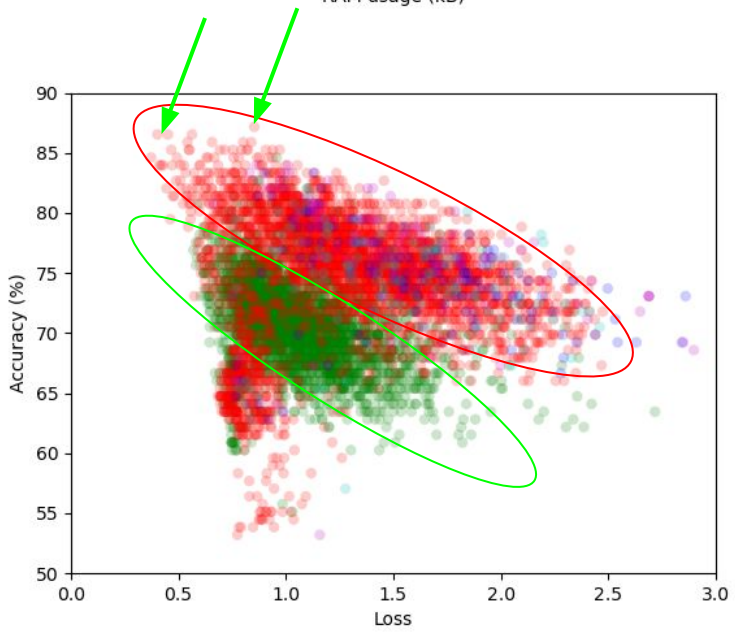
Versus: Latency, Loss, RAM

One of the best:

- LPF 4 Hz
- FFT 128
- 86 neurons







Update Academic use-case

Conclusions:

- All methods yield implementable result
- Used method is application-specific
- Used method depends on requirements
- A lot of variables, use prior knowledge to make the right decisions

User group interaction

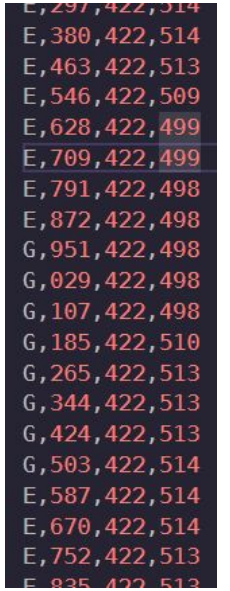
Questions / remarks?

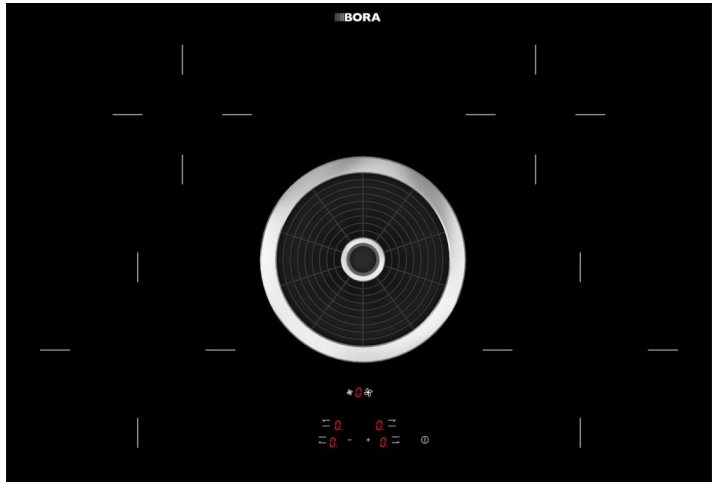
Use case by the user group: E.D.&A.

- Induction cooker with built-in ventilation unit
- Capacitive touch control
- Increasing/decreasing ventilation
  - 2 buttons
  - 4 states
  - 7 segment display
- Raw ADC values
  - 12 Hz sampling rate
  - arbitrary idle value
  - delta touch +/- 20
  - Published on UART @ 9600 baud

E.D.&A.: Induction cooker ventilation control

- Classical signal processing for touch detection
- Hardware and firmware setup fixed
- Setup developed by intern student (2019)
- Labeled data using mechanical finger
  - data from a single button
  - idle states
  - increasing / decreasing induction
  - wet touch buttons
  - …





Use case by the user group: E.D.&A.

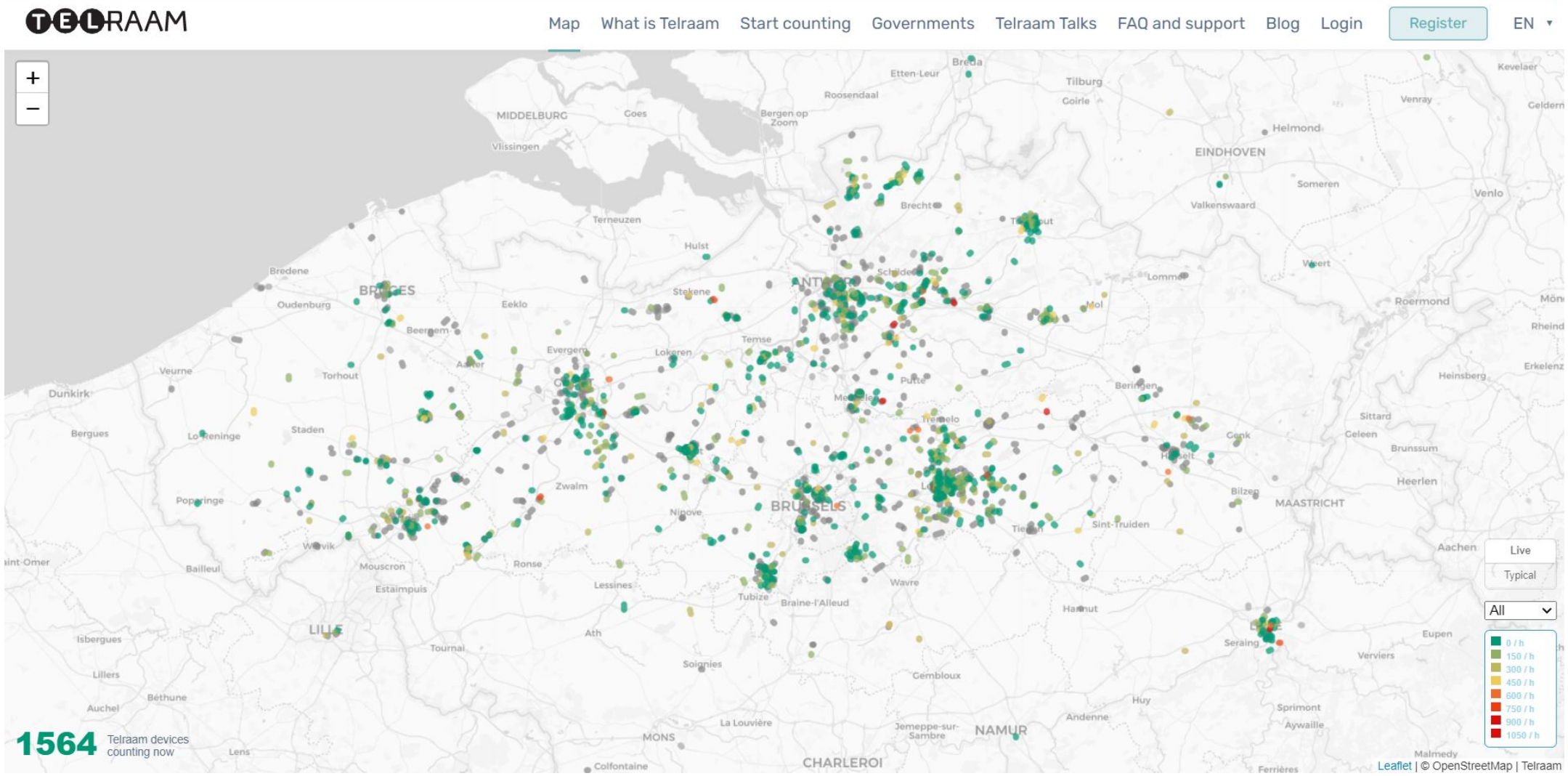
Goals

1. Find out if a NN on microcontroller can replace classical processing algorithm
   a. Best fit model
   b. Targets:
      i. Cortex M4
      ii. Cortex M0
2. Find out if a NN can improve immunity compared to classical processing algorithm
   a. Different levels of induction settings
   b. Wet touch buttons
   c. Different types of pots and locations

User group interaction

Questions / remarks?

TML: Introduction

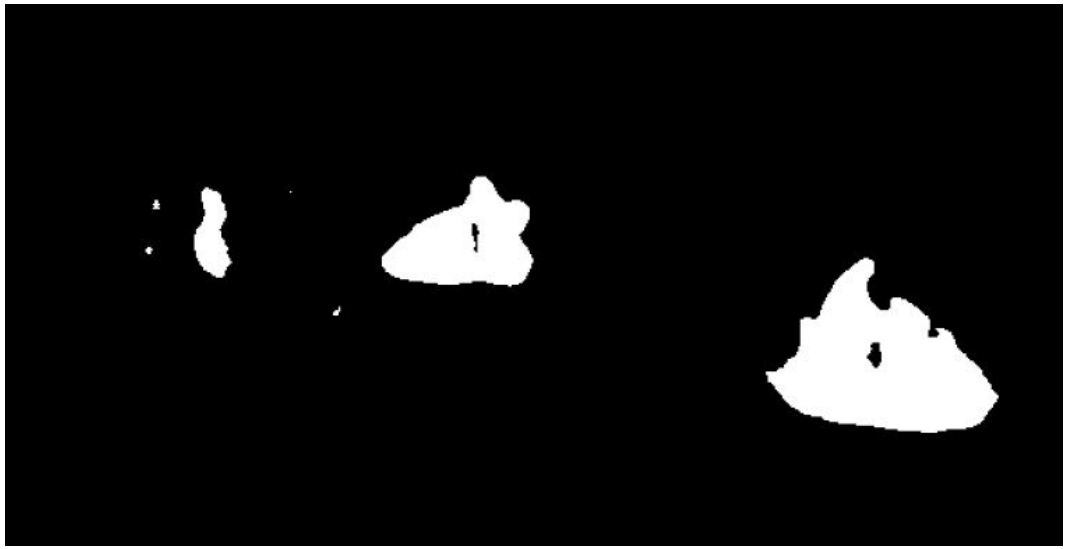


TML: Introduction

Hello am project

- Easy traffic counting
- Using Raspberry Pi with camera
- Insights about traffic density with user supplied data

- Classic ML & CV
- Background subtraction: slow
- Classifier for object blobs: difficult and inaccurate



TML: Use Case

Goals:

- Traffic counting at home
- Using Raspberry Pi with camera
- 2 labeled data sets available
- Detecting 5 different classes: pedestrian, bike, car, truck and other
- Frame rate of +/-5 fps

TML: Methodology

Use object detector to detect object class and location

➡ Slow (~ seconds/frame) in normal DL framework

TF Lite is perfect for low power devices!
Combine with Object Detection API

TML: Detection

Pre-trained (MS COCO) SSD+MobileNetV2 in TF

Train further on mix of data sets

160x160 resolution

Dynamic range post training quantiz

of weights (TF Lite default settings)

Export to TF Lite model

TML: Tracking

Passersby should only be counted **once**

➡ track them!

Using motpy library

Detect when in certain “zone”

TML: Setup

Raspberry Pi 4 4GB RAM with Raspberry Pi OS

Python code, includes:

- TF Lite interpreter
- motpy tracker
- OpenCV

Valid .tflite model

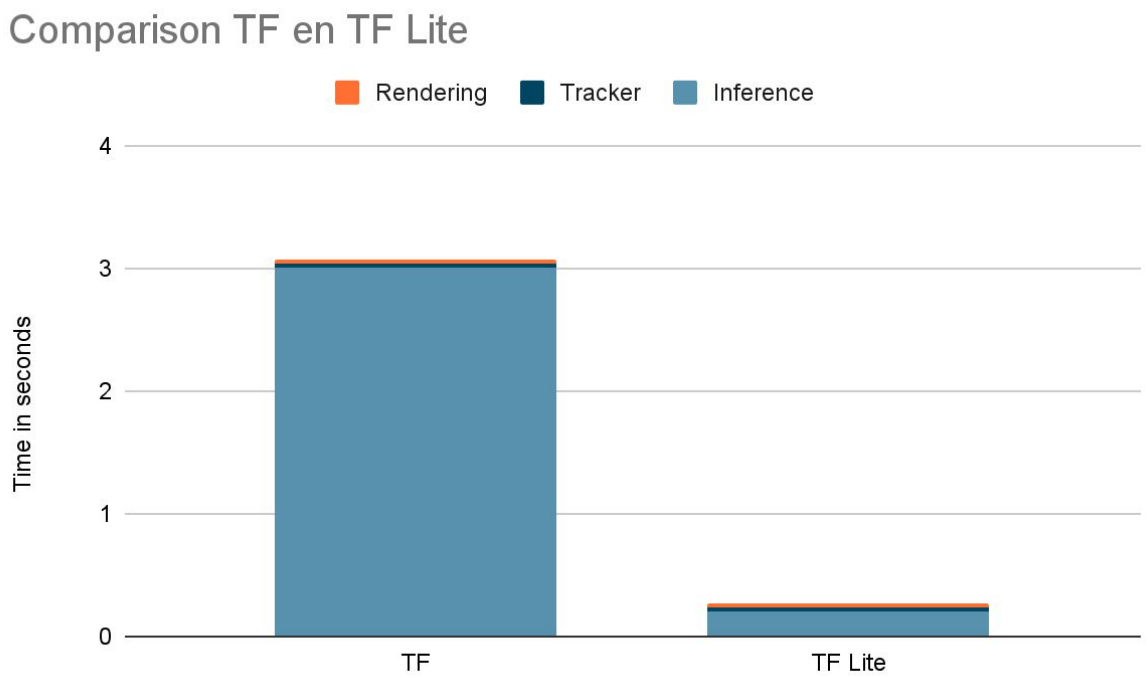
.tflite compatible labelmap file

TML: Results

59% COCO mAP

85% PASCAL VOC mAP

Average detection: 0.2 seconds → +/- 5 fps



TML: Improvements

- More data: better generalization!
  Retrain model for better results

- In-depth optimization using TF Lite:
  various quantization strategies + more to come!

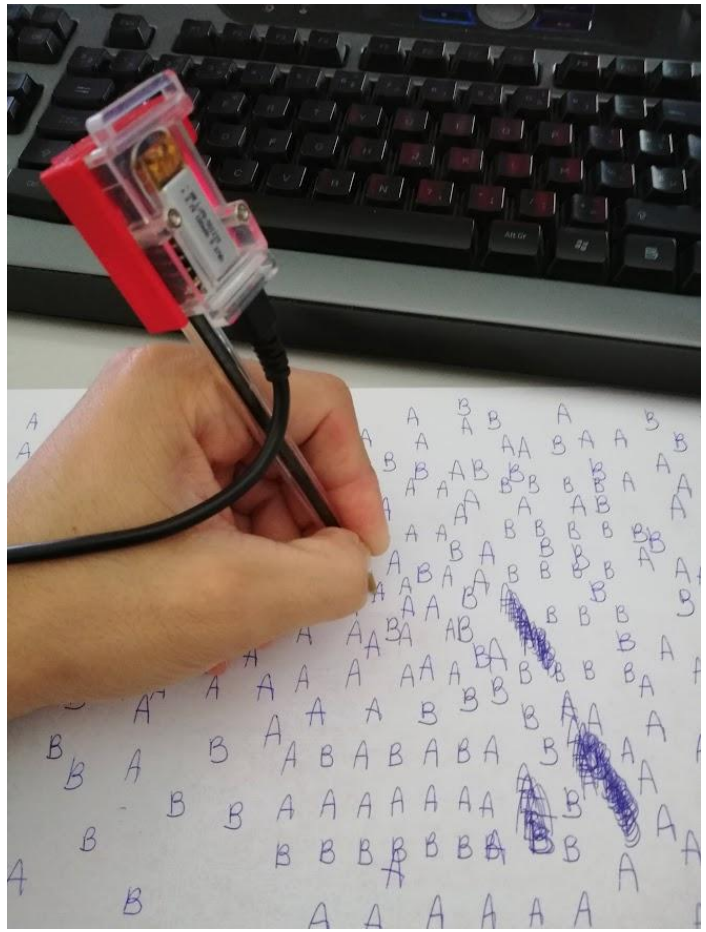
TML: Demo

User group interaction

Questions / remarks?

Workshop

- 14/09/2021 afternoon
- Based on the academic use case
- Edge Impulse



Networking