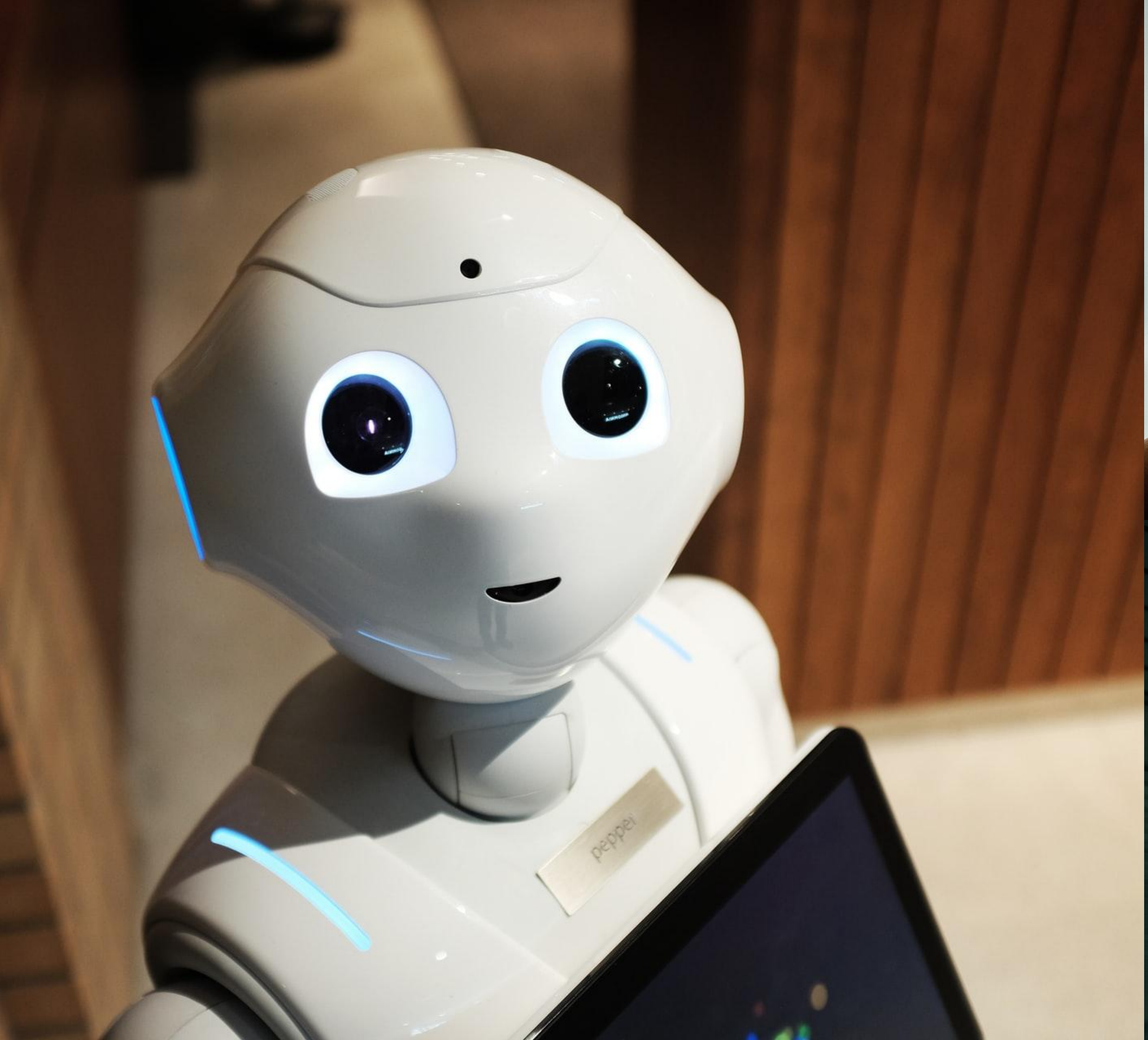
AI@EDGE

EMBEDDED AI

vives hogeschool

KU LEUVEN

eavise


pepper

WORKSHOP TEAM

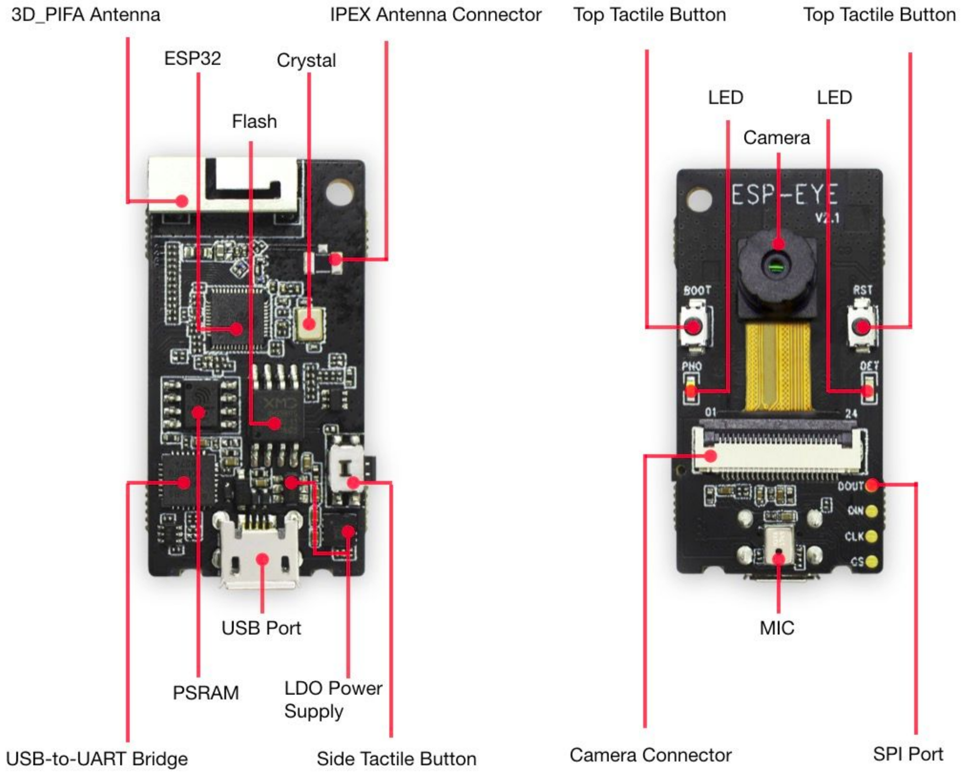
Tanguy
Ophoff

Maarten
Vandersteegen

Kristof
Van Beeck

Toon
Goedemé

ESP – EYE



ESP32 MCU

Xtensa Dual-Core 32-bit LX6
240 MHz Clock
512 kB RAM
36 GPIO
WIFI stack
Bluetooth stack

2 MP color camera
Microphone
4 MB External SPI Flash
8 MB External SPI PSRAM
€ 20 - 25

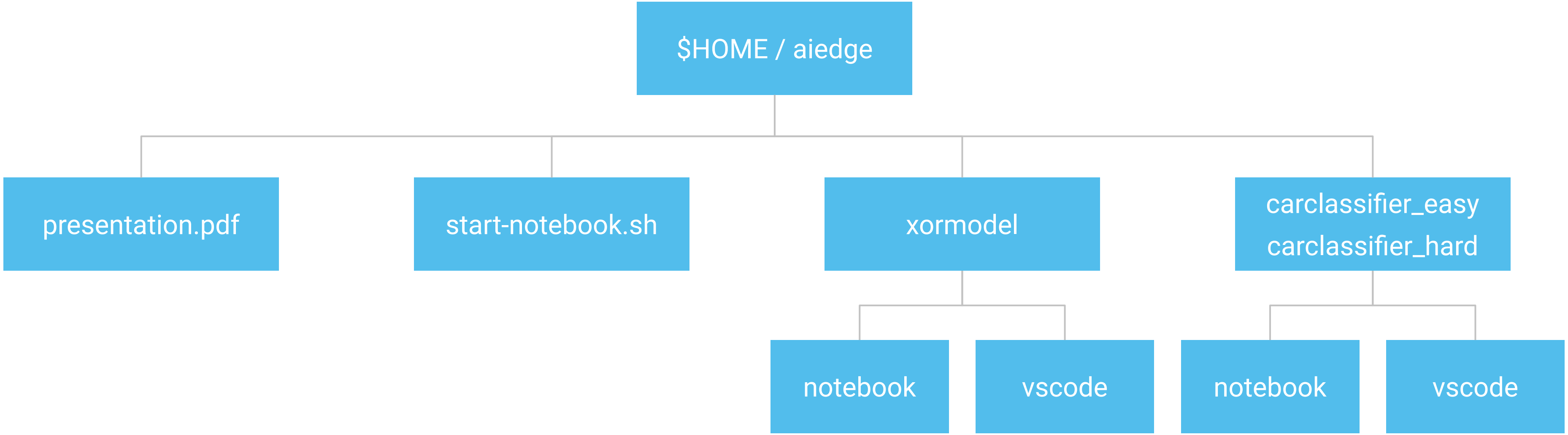
OPTIMIZATION PIPELINE

Train Model	Optimize Model	Embedded Framework	Compiler Optimization
TensorFlow	TensorFlow TFLite	TFLite Micro ESP IDF FreeRTOS	TFLite Micro ESP IDF FreeRTOS

AGENDA

09:30	Introduction
09:45	Project I - XOR Model
10:30	Coffee Break
10:45	Project II - Car Classifier
12:30	Lunch & Networking

FOLDER STRUCTURE



AI@EDGE

XOR MODEL

vives hogeschool

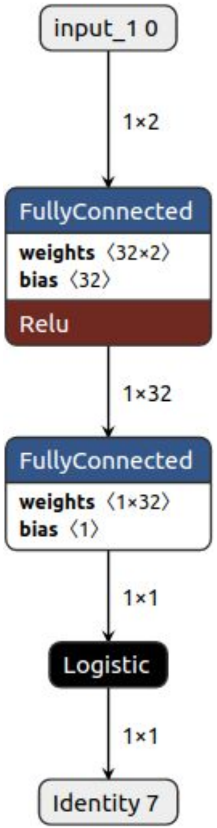
KU LEUVEN

eavise

XOR MODEL

We will **create**, **train** and **test** a simple deep learning network that models an XOR gate.
Afterwards we will **export** the model to TFLite and **run** it on the ESP-EYE.

INPUT 1	INPUT 2	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	0

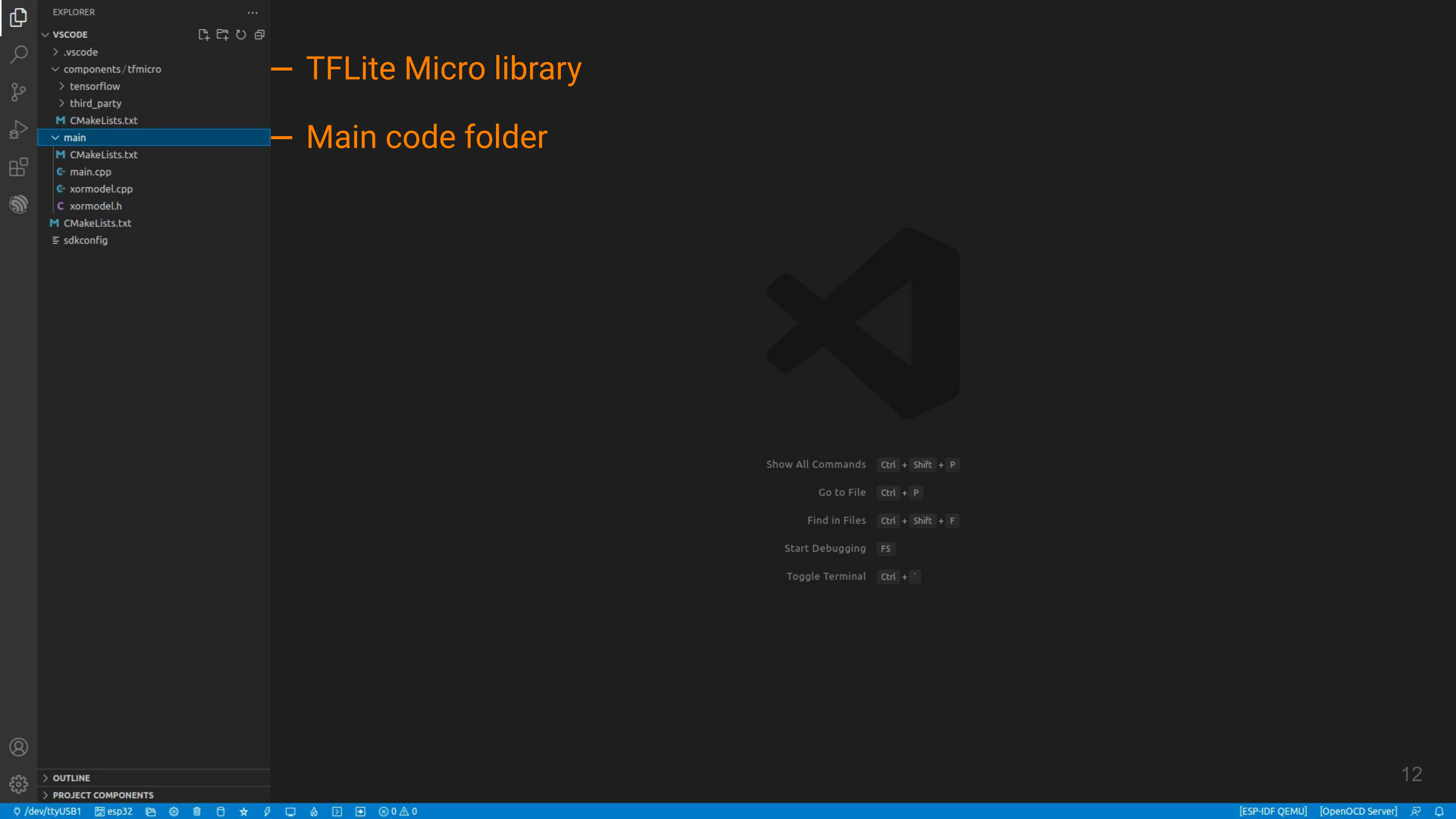


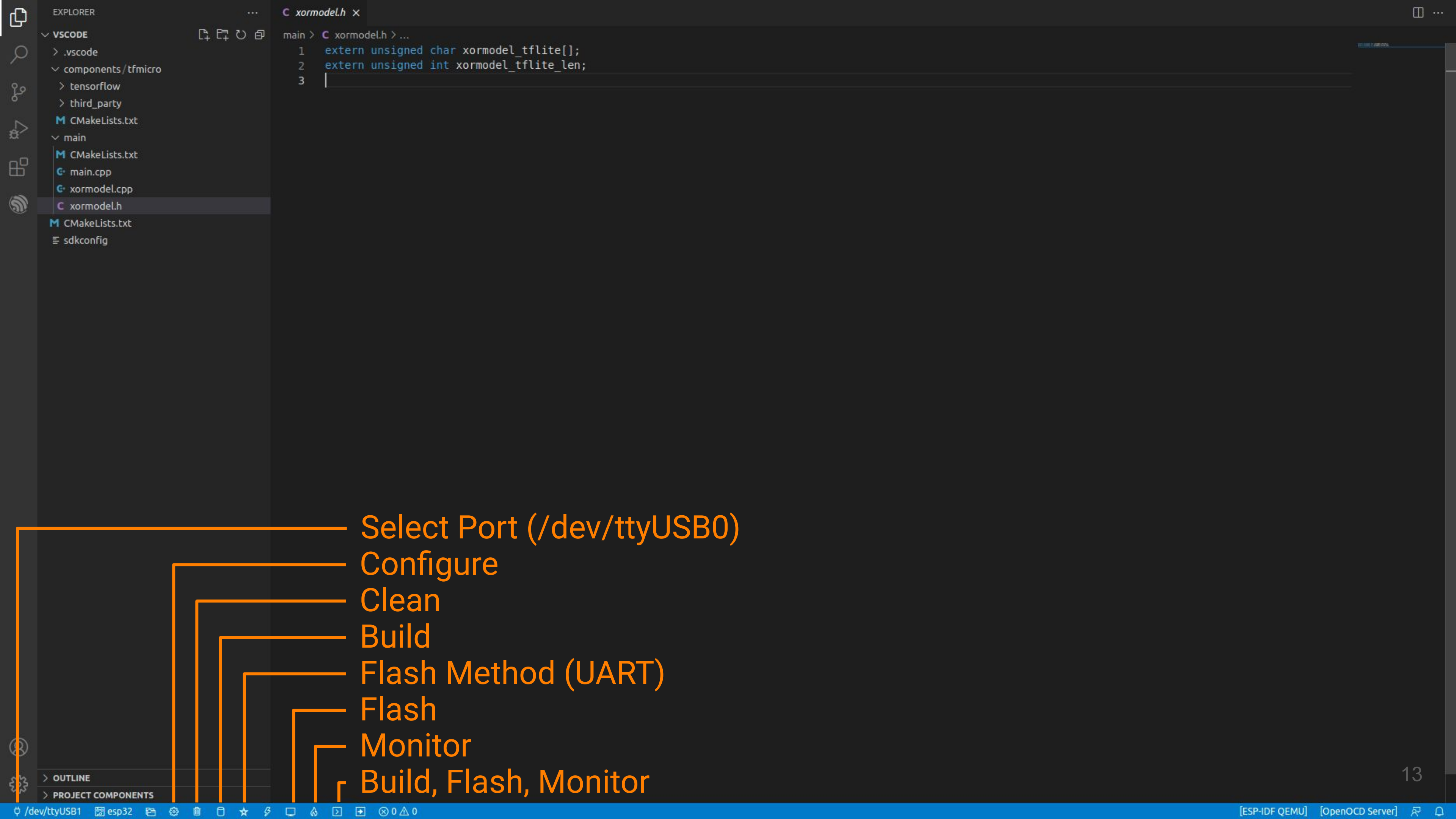
NOTEBOOK

1. Open a terminal
2. Go to the workshop folder: **cd ~/aiedge**
3. Start notebook for the XOR model: **./start-notebook.sh xormodel**
4. Open the notebook in the browser and complete the TODOs

VS CODE

1. Open a new terminal
2. Go to the correct folder: **cd ~/aiedge/xormodel**
3. Copy the generated cpp file: **cp notebook/xormodel.cpp vscode/main/xormodel.cpp**
4. Start visual studio code: **code vscode**
5. Open main.cpp and complete the TODOs
6. *OPTIONAL:* Modify the code to run the model 4 times and complete the XOR truth table
7. *OPTIONAL:* What happens when we add noise to the input data ?


EXPLORER
VSCODE
.vscode
components / tfmicro
tensorflow
third_party
CMakeLists.txt
main
CMakeLists.txt
main.cpp
xormodel.cpp
xormodel.h
CMakeLists.txt
sdkconfig
TFLite Micro library
Main code folder
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P
Find in Files Ctrl + Shift + F
Start Debugging F5
Toggle Terminal Ctrl + `
OUTLINE
PROJECT COMPONENTS
/dev/ttyUSB1
esp32
[ESP-IDF QEMU]
[OpenOCD Server]


EXPLORER
VSCODE
.vscode
components / tfmicro
tensorflow
third_party
CMakeLists.txt
main
CMakeLists.txt
main.cpp
xormodel.cpp
xormodel.h
CMakeLists.txt
sdkconfig
OUTLINE
PROJECT COMPONENTS
xormodel.h
main > C xormodel.h > ...
extern unsigned char xormodel_tflite[];
extern unsigned int xormodel_tflite_len;
Select Port (/dev/ttyUSB0)
Configure
Clean
Build
Flash Method (UART)
Flash
Monitor
Build, Flash, Monitor
/dev/ttyUSB1
esp32
0 0
[ESP-IDF QEMU]
[OpenOCD Server]

AI@EDGE

CAR CLASSIFIER

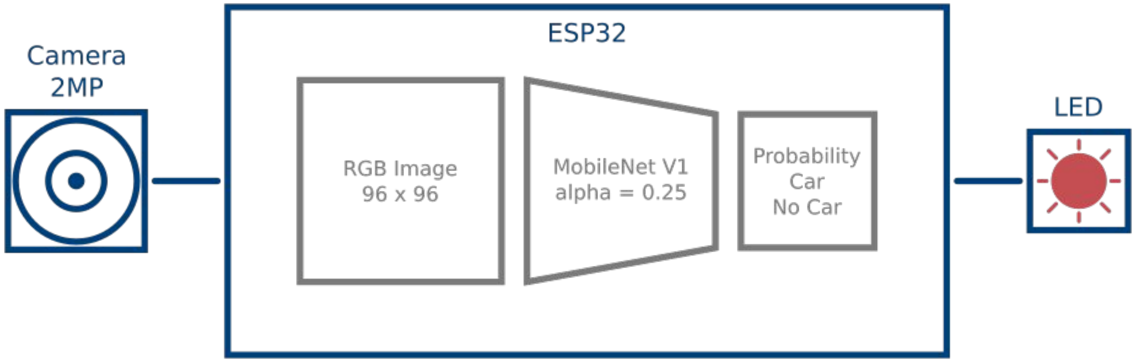
vives hogeschool

KU LEUVEN

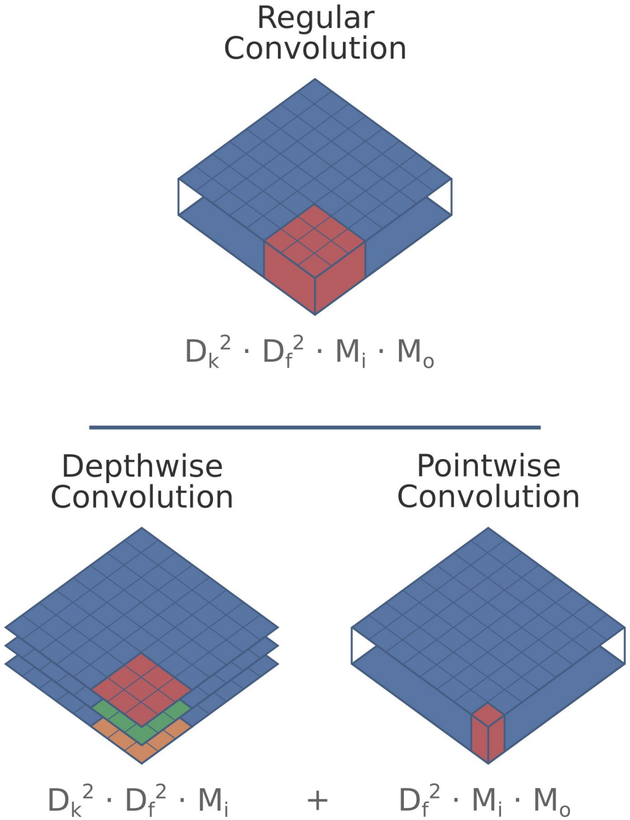
eavise

CAR CLASSIFIER

We will **test** and **quantize** a deep learning network that detects cars.
Afterwards we will **export** the model to TFLite and **run** it on the ESP-EYE.
Finally, we run a **benchmark** with the test dataset on the ESP-EYE.



MOBILENET V1



	FLOAT	INT8
Alpha	0.25	0.25
Parameters	213 329	213 329
Operations	8.9 Million	8.9 Million
Model Size	840 kB	320 kB
Inference Time	3.1 sec	0.8 sec

The float model does **not** fit into **Internal RAM**

Placing the model into **External RAM** is **cumbersome**

QUANTIZATION BENEFITS

Model Weights

4x less storage
reduced loading times from FLASH

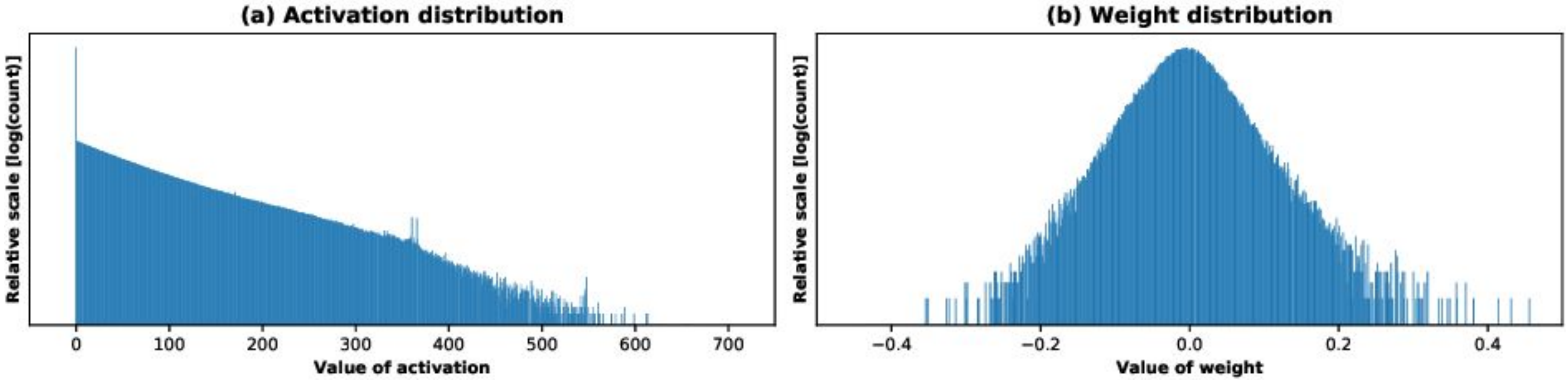
Output Activations

4x less RAM
reduced loading times from RAM
no floating ops needed

Quantization allows us to **reduce** the **memory** requirements
and **increase** the inference **speed** without sacrificing accuracy.

HOW TO QUANTIZE

$$q = round(x)$$



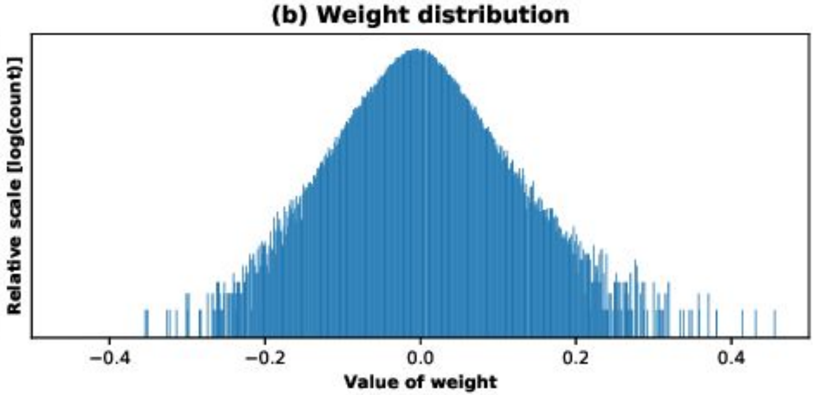
This **naive** approach results in a severe **degradation** of the model **accuracy**.

QUANTIZE WEIGHTS

PER CHANNEL QUANTIZATION

$$w \approx sq$$

w → float weight
q → int8 weight
s → float scale or int32 scale + bit-shift



Compute a separate scale s for each filter within a convolution and use it to quantize the kernel weights.

$$s = \frac{max(|w|)}{127}$$

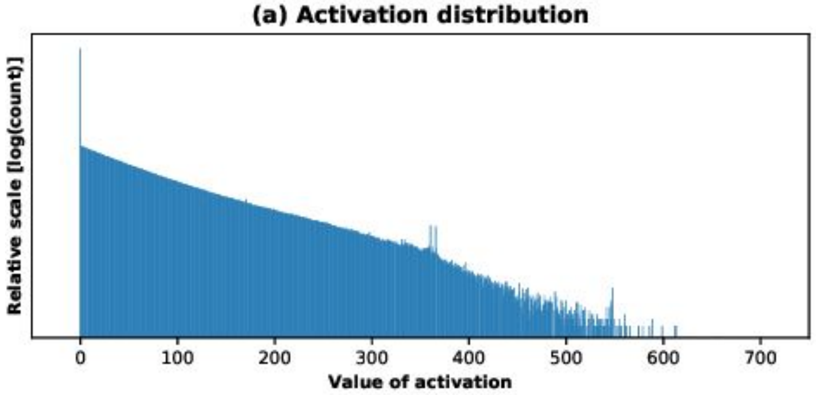
$$q = round\left(\frac{w}{s}\right)$$

QUANTIZE ACTIVATIONS

PER AXIS QUANTIZATION

$$a \approx s(q - z)$$

a → float activation
q → int8 activation
s → float scale or int32 + bit-shift
z → int8 zero point

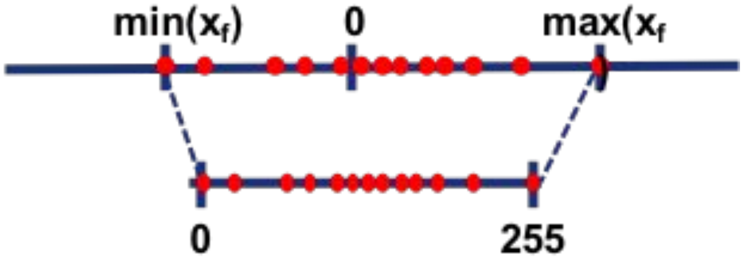


Compute a separate scale s and zero point z for each layer
and use it to quantize the output feature map activations.

$$s = \frac{max(a) - min(a)}{255} \qquad z = -128 - \frac{min(a)}{s}$$

CALIBRATION

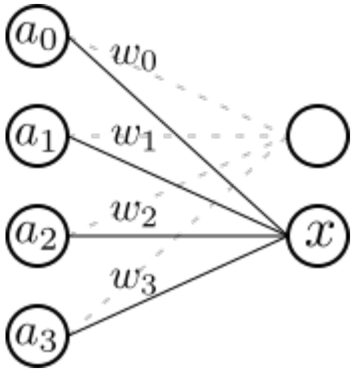
Run some **image** samples through the network
and **record** the **minimal** and **maximal** activation values for **every layer**.
These can then be used to **compute** the quantization **values**.



$$s = \frac{max(a) - min(a)}{255} \qquad z = -128 - \frac{min(a)}{s}$$

QUANTIZED INFERENCE

$$x = \sum_{i}^{N} a_i w_i$$



$$s_x(q_x - z_x) = \sum_{i}^{N} \left(s_a(q_a^i - z_a)\right)\left(s_w q_w^i\right)$$

Additional multiplication

Additional summations

$$q_x = z_x + \frac{s_a s_w}{s_x} \sum_{i}^{N} (q_a^i - z_a) q_w^i$$

NOTEBOOK

1. Open a terminal
2. Go to the workshop folder: **cd ~/aiedge**
3. Start notebook for the car classifier model:
   ./start-notebook.sh carclassifier_easy
   ./start-notebook.sh carclassifier_hard
4. Open the notebook in the browser and complete the TODOs

VS CODE

1. Open a new terminal

2. Go to the correct folder:
   cd ~/aiedge/carclassifier_easy
   cd ~/aiedge/carclassifier_hard

3. Copy the generated cpp file:
   cp notebook/carclassifier.cpp vscode/main/carclassifier.cpp

4. Start visual studio code: **code vscode**

5. Open main.cpp and complete the TODOs

carclassifier.h

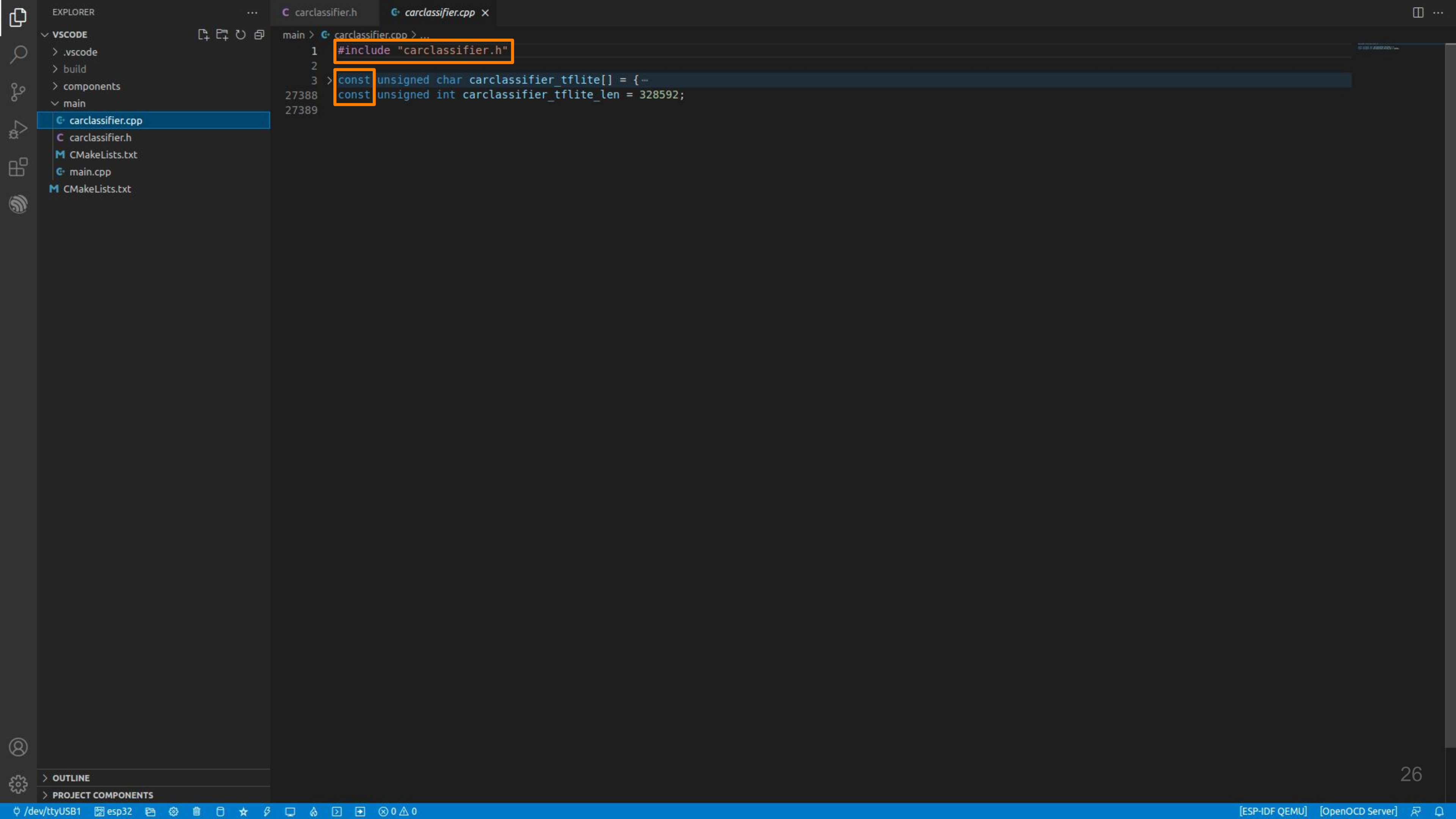
main > carclassifier.h > ...

```
#ifndef CARCLASSIFIER_H
#define CARCLASSIFIER_H

// Model input dimensions
constexpr int kNumCols = 96;
constexpr int kNumRows = 96;
constexpr int kNumChannels = 3;
constexpr int kMaxImageSize = kNumCols * kNumRows * kNumChannels;

extern const unsigned char carclassifier_tflite[];
extern const unsigned int carclassifier_tflite_len;

#endif /* CARCLASSIFIER_H */
```


EXPLORER
VSCODE
.vscode
build
components
main
carclassifier.cpp
carclassifier.h
CMakeLists.txt
main.cpp
CMakeLists.txt
OUTLINE
PROJECT COMPONENTS
carclassifier.h
carclassifier.cpp
main > carclassifier.cpp > ...
#include "carclassifier.h"
const unsigned char carclassifier_tflite[] = {...
const unsigned int carclassifier_tflite_len = 328592;
/dev/ttyUSB1
esp32
0 0
[ESP-IDF QEMU]
[OpenOCD Server]

COMPILER OPTIMIZATIONS

Click on the configure button (cogwheel) and **enable** some compiler **optimizations**.
Run your code again to verify that the **inference** is indeed **faster**.

1. Set the *CPU Frequency* to “240 MHz”
2. Set the *Optimization Level* to “Optimize for performance”
3. Set the *Bootloader Optimization Level* to “Optimize for performance”
4. Set the *Assertion Level* to “disabled”

OPTIONAL BENCHMARK

1. Replace the *CameraImageProvider* with the *SerialImageProvider*
   Look at the header files for the correct arguments
   You can use a baud rate of 115 200

2. Enable the debug assertions again.
   Reading from UART does not work without the debug assertions.

3. Send the output and inference time back
   You can use the *EspSerialPort::write_result* method
   The output should be 1 if there is a car and 0 otherwise

4. Compile and flash your program on the ESP-EYE, but do not open the monitoring terminal
   Afterwards, run the following commands in a terminal to start sending images:
   cd ~/aiedge
   source /opt/venv/embeddedai/bin/activate
   ./run_serial.py /dev/ttyUSB0

Questions