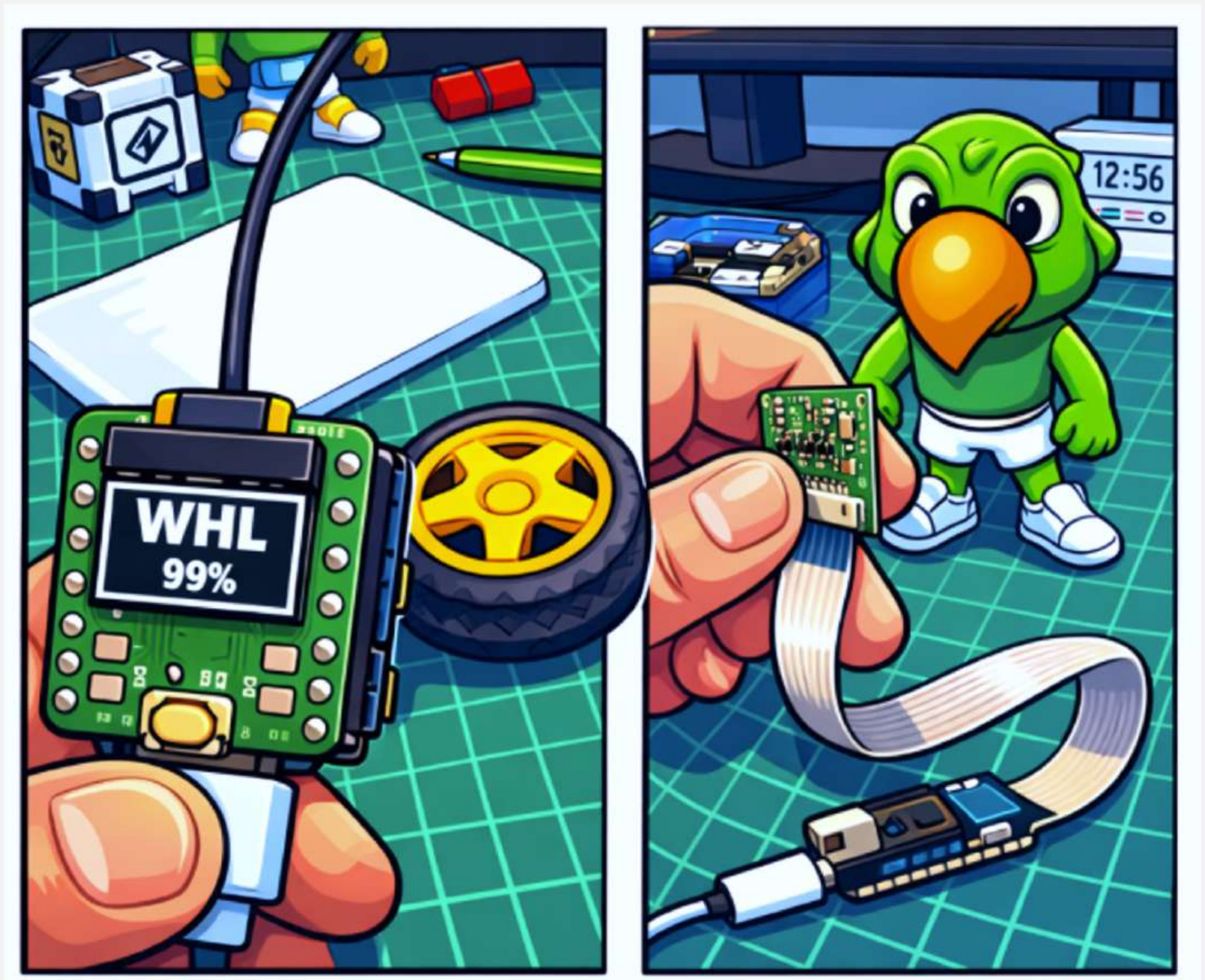
TinyML Made Easy

Hands-On with the Seeed Studio Devices

XIAOML Kit and Grove Vision AI V2



Marcelo Rovai

2026

TinyML Made Easy

Hands-On with Seeed Studio Devices

Marcelo Rovai

2026-02-05

Table of contents

Preface

Finding the right info used to be the big issue for people involved in technical projects. Nowadays, there is a deluge of information, but it is not easy to determine what can and what cannot be trusted. A good pointer is to look for the credentials and the affiliation of the author of a document or that of the associated institutions. Since 1992 EsLaRed has been providing training in different aspects of Information Technologies in Latin America and the Caribbean, always striving to provide accurate and timely training materials, which have evolved accordingly to shifts in the technology focus. IoT and Machine Learning are poised to play a pivotal role, so the work of Professor Marcelo Rovai addressing Tiny Machine Learning is both timely and authoritative, in this rapidly changing field.

TinyML Made Easy series is exactly what the title means. Written in a clear and concise style, with emphasis on practical applications and examples, drawing from many years of experience and overwhelming enthusiasm in sharing his knowledge, there is no doubt that Marcelo's work is a very significant contribution to this very interesting field. The knowledge acquired from the exercises will enable the readers to undertake other projects that might interest them.

Ermanno Pietrosemoli, President Fundación Escuela Latinoamericana de Redes

November 2023.

Acknowledgments

I extend my deepest gratitude to the entire TinyML4D Academic Network, comprised of distinguished professors, researchers, and professionals. Notable contributions from Marco Zennaro, Ermanno Petrosemoli, Brian Plancher, José Alberto Ferreira, Jesus Lopez, Diego Mendez, Shawn Hymel, Dan Situnayake, Pete Warden, and Laurence Moroney have been instrumental in advancing our understanding of Embedded Machine Learning (TinyML) and Edge AI.

Special commendation is reserved for Professor Vijay Janapa Reddi of Harvard University. His steadfast belief in the transformative potential of open-source communities, coupled with his invaluable guidance and teachings, has been a beacon and a cornerstone of our efforts from the beginning.

Acknowledging these individuals, we pay tribute to the collective wisdom and dedication that have enriched this field and our work.

Google Nano Banana and OpenAI's GPT were used to generate some of the images in the book. Claude Sonnet and Perplexity helped with code and text reviews.

Introduction

The convergence of artificial intelligence and embedded systems is revolutionizing how we interact with technology. TinyML—Machine Learning on microcontrollers—brings intelligent decision-making directly to edge devices, enabling applications from smart homes to industrial automation, all while maintaining privacy, reducing latency, and minimizing power consumption.

This book demystifies TinyML through hands-on projects using Seeed Studio's powerful yet affordable development platforms: the XIAOML Kit based on the XIAO ESP32S3 Sense and the Grove Vision AI V2. Whether you're classifying images, detecting objects, recognizing keywords, or identifying motion patterns, you'll learn by building real, working applications.

The XIAO ESP32S3 Sense, despite its thumb-sized form factor (21 x 17.5mm), packs remarkable capabilities: a dual-core ESP32-S3 processor running at 240 MHz, 8MB of PSRAM, an OV2640 camera, and a digital microphone. Combined with the XIAOML Kit's expansion board, which features an OLED display and an IMU sensor, this platform provides everything needed for comprehensive TinyML exploration.

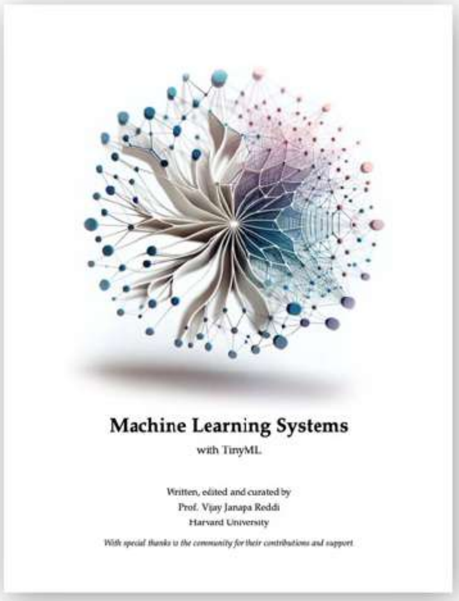
The Grove Vision AI V2 offers a complementary approach: a dedicated AI vision module with built-in hardware acceleration, ideal for no-code and low-code computer vision applications. Together, these platforms demonstrate the spectrum of TinyML deployment strategies, from fully custom models to pre-trained solutions.

Throughout this book, you'll use Edge Impulse Studio as your primary development platform, gaining practical experience in the complete machine learning workflow: data collection, preprocessing, model training, optimization, and deployment. Each project systematically builds your understanding, from foundational concepts to advanced techniques such as digital signal processing and feature engineering.

This isn't just a technical manual—it's your practical guide to building intelligent edge devices that can see, hear, and understand the world around them.

About this Book

This book is part of the open book Machine Learning Systems, which we invite you to



Machine Learning Systems

with TinyML

Written, edited and curated by

Prof. Vijay Janapa Reddi

Harvard University

With special thanks to the community for their contributions and support

read.

Who This Book Is For

This book is designed for makers, students, engineers, and anyone curious about bringing artificial intelligence to embedded devices. Whether you're new to machine learning or experienced with traditional ML but unfamiliar with embedded systems, the hands-on approach will guide you from setup to deployment.

You should have basic familiarity with Arduino programming and the Arduino IDE. Prior machine learning experience is helpful but not required—each concept is introduced with clear explanations and practical examples.

What You'll Learn

The book is organized into three main sections, each building on the previous:

XIAOML Kit Projects - The heart of the book, featuring comprehensive hands-on projects:

- **Setup**: Complete configuration of the XIAO ESP32S3 Sense and XIAOML Kit, testing all sensors and peripherals
- **Image Classification**: Build a computer vision system to classify objects using convolutional neural networks (CNNs)
- **Object Detection**: Deploy FOMO (Faster Objects, More Objects) for real-time multi-object detection
- **Keyword Spotting (KWS)**: Create a voice-activated system that recognizes specific spoken commands
- **Motion Classification and Anomaly Detection**: Use IMU data to recognize gestures and detect unusual patterns

Preprocessing Deep Dive - Understanding the signal processing that makes TinyML work:

- **DSP Spectral Features**: Master digital signal processing techniques for extracting meaningful features from sensor data
- **KWS Feature Engineering**: Learn specialized audio processing methods, including MFCC (Mel-frequency cepstral coefficients) and spectrograms

Grove Vision AI V2 - Exploring no-code and accelerated computer vision:

- **Setup and No-Code Applications**: Deploy pre-trained models without writing code using SenseCraft AI
- **Image Classification**: Custom model deployment with hardware acceleration
- **Object Detection**: High-performance object detection on a dedicated AI accelerator

Tools and Platforms

You'll work extensively with:

- **Arduino IDE**: Primary development environment for the XIAO ESP32S3 Sense
- **Edge Impulse Studio**: Cloud-based platform for ML model development, from data collection through deployment
- **SenseCraft AI**: Seeed Studio's platform for no-code AI model deployment
- **Python notebooks**: For advanced preprocessing and data analysis

Hardware Requirements

To follow along with all projects, you'll need:

- XIAOML Kit (includes XIAO ESP32S3 Sense and expansion board with OLED and an IMU)
- Grove Vision AI V2 (for the final section)
- USB-C cable for programming
- MicroSD card (optional, for data logging)

Project-Based Approach

Every chapter follows the same proven workflow:

1. **Problem definition**: Understanding the application and its constraints
2. **Data collection**: Gathering training data from sensors or datasets
3. **Feature extraction**: Processing raw sensor data into meaningful features
4. **Model training**: Building and training neural networks optimized for edge deployment
5. **Evaluation**: Testing model performance and understanding trade-offs
6. **Deployment**: Uploading trained models to your device and running inference
7. **Real-world testing**: Validating your application with actual sensor data

Learning Philosophy

This book emphasizes learning by doing. Rather than lengthy theoretical discussions, you'll build working projects that demonstrate core concepts. Code examples are complete and tested, with detailed explanations of key sections. When theory is necessary—like understanding how convolutional layers work or why we use Mel-frequency features for audio—it's presented in context, connected directly to the project you're building.

Beyond the Book

By the end, you won't just have completed several projects—you'll understand the fundamental principles that apply to any ML application. You'll know how to:

- Choose appropriate sensors and platforms for your application
- Collect and prepare training data effectively
- Select and configure neural network architectures for embedded constraints
- Optimize models for memory, latency, and power consumption
- Deploy and validate ML models on resource-constrained devices

These skills transfer directly to your own TinyML projects, whether you're building smart home devices, industrial sensors, wearables, or environmental monitors.

#

XIAOML Kit

Setup

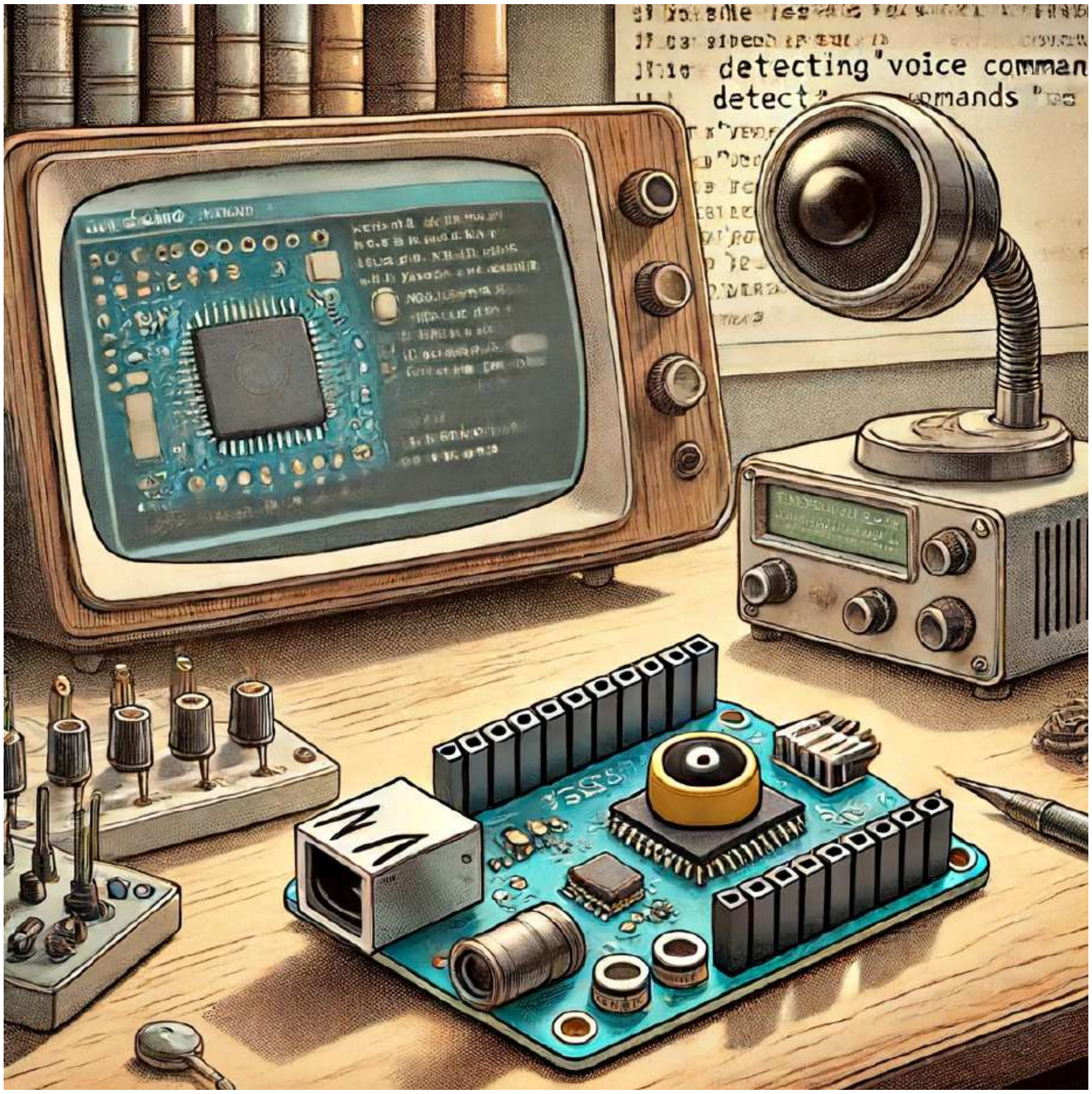


Figure 1: *DALL · E prompt - 1950s cartoon-style drawing of a XIAO ESP32S3 board with a distinctive camera module, as shown in the image provided. The board is placed on a classic lab table with various sensors, including a microphone. Behind the board, a vintage computer screen displays the Arduino IDE in muted colors, with code focusing on LED pin setups and machine learning inference for voice commands. The Serial Monitor on the IDE showcases outputs detecting voice commands like 'yes' and 'no'. The scene merges the retro charm of mid-century labs with modern electronics.*

Overview

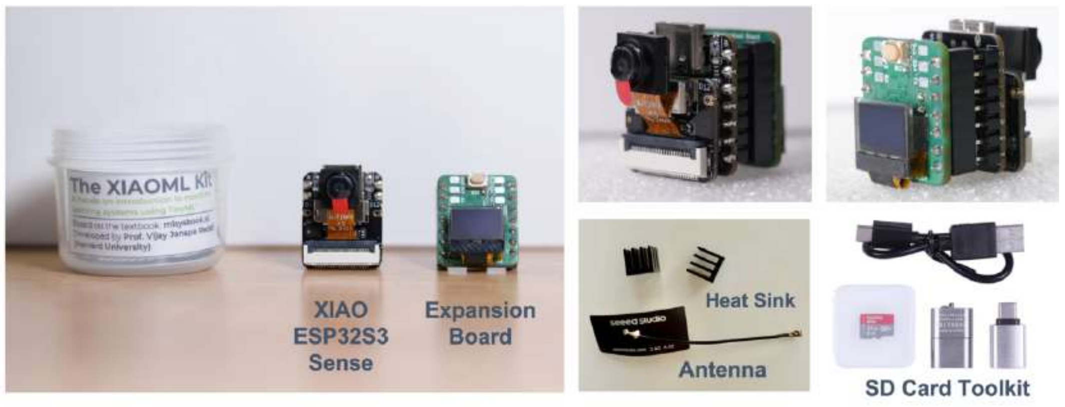
The XIAOML Kit is designed to provides hands-on experience with TinyML applications. The kit includes the powerful XIAO ESP32S3 Sense development board and an expansion board that adds essential sensors for machine learning projects.

Complete XIAOML Kit Components:

- **XIAO ESP32S3 Sense**: Main development board with integrated camera sensor, digital microphone, and SD card support
- **Expansion Board**: Features a 6-axis IMU (LSM6DS3TR-C) and 0.42” OLED display for motion sensing and data visualization
- **SD Card Toolkit**: Includes SD card and USB adapter for data storage and model deployment
- **USB-C Cable**: For connecting the board to your computer
- **Antenna and Heat Sinks**

Attention

Do not install the heat sinks (or carefully, remove them) on/from the XIAO ESP32S3 if you want to use the XIAO ML Kit Expansion Board. See Appendix for more information.

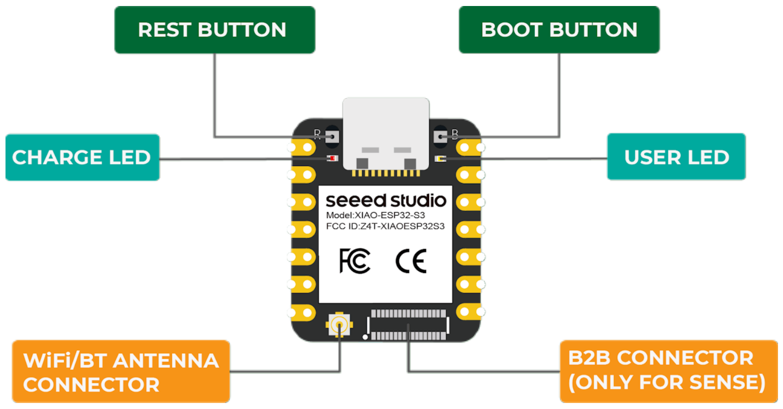


XIAO ESP32S3 Sense - Core Board Features

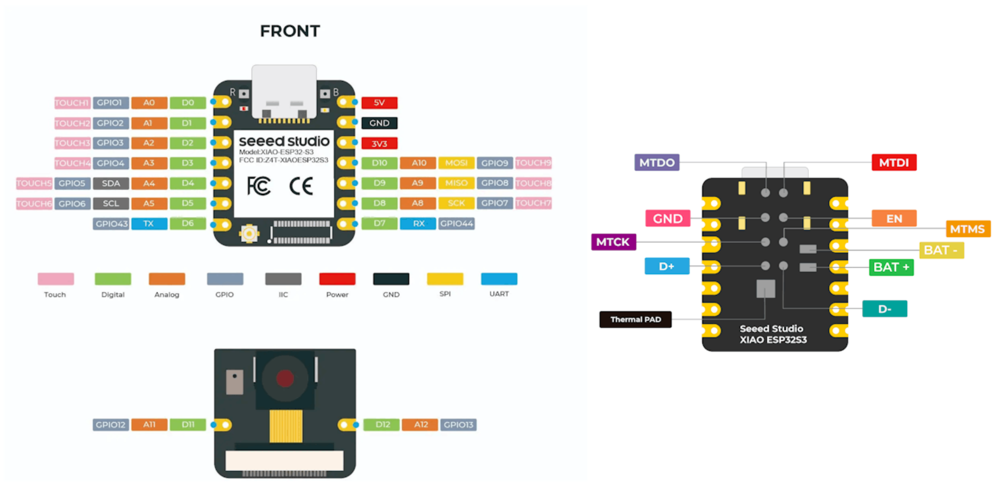
The XIAO ESP32S3 Sense serves as the heart of the XIAOML Kit, integrating embedded ML computing power with photography and audio capabilities, making it an ideal platform for TinyML applications in intelligent voice and vision AI.

Key Features

- **Powerful MCU**: ESP32S3 32-bit, dual-core, Xtensa processor operating up to 240 MHz, with Arduino / MicroPython support
- **Advanced Functionality**: Detachable OV2640 camera sensor for 1600 × 1200 resolution, compatible with OV5640 camera sensor, plus integrated digital microphone
- **Elaborate Power Design**: Lithium battery charge management with four power consumption models, deep sleep mode with power consumption as low as 14 A
- **Great Memory**: 8 MB PSRAM and 8 MB FLASH, supporting SD card slot for external 32 GB FAT memory
- **Outstanding RF Performance**: 2.4 GHz Wi-Fi and BLE dual wireless communication, supports 100m+ remote communication with U.FL antenna
- **Compact Design**: 21 × 17.5 mm, adopting the classic XIAO form factor, suitable for space-limited projects



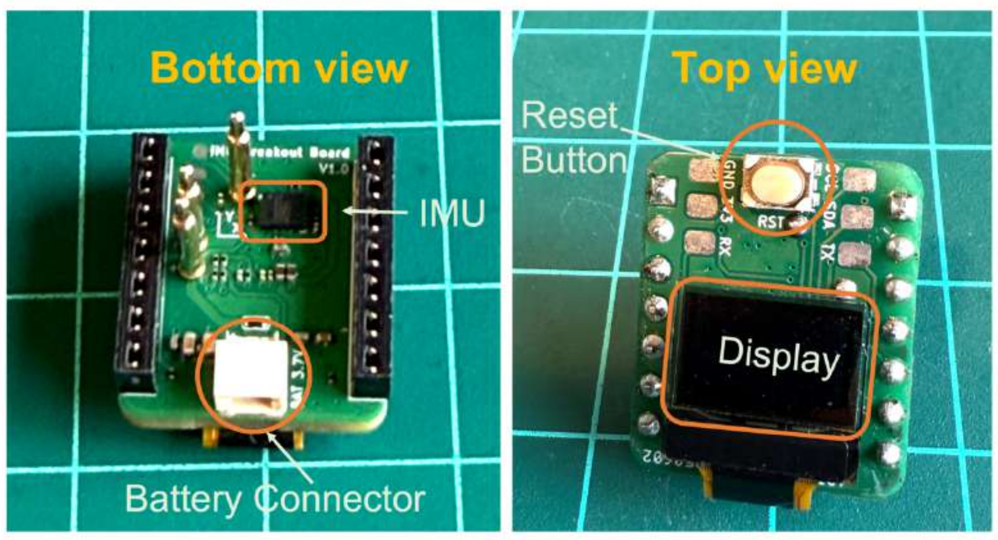
Below is the general board pinout:



For more details, please refer to the Seeed Studio Wiki page

Expansion Board Features

The expansion board extends the XIAOML Kit's capabilities for motion-based machine learning applications:



Components:

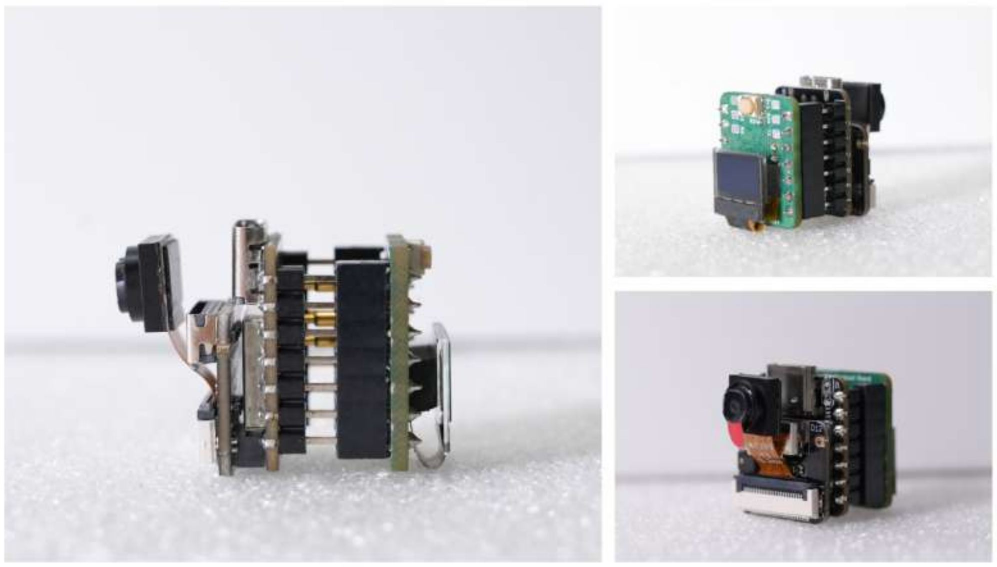
- 6-axis IMU (LSM6DS3TR-C):

  - 3-axis accelerometer and 3-axis gyroscope for motion detection and classification
    - Accelerometer range: ±2/±4/±8/±16 g
    - Gyroscope range: ±125/±250/±500/±1000/±2000 dps
    - I2C interface (address: 0x6A)

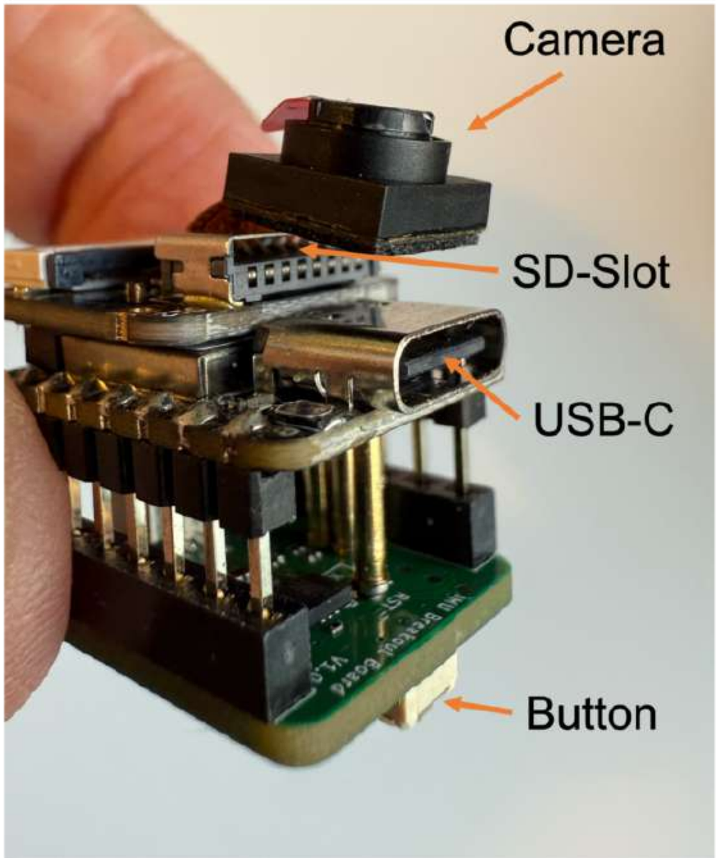
- 0.42" OLED Display
  - Monochrome display (72×40 resolution) for real-time data visualization
    - Controller: SSD1306
    - I2C interface (address: 0x3C)
- Restart Button (EN)
- Battery Connector (BAT+, BAT-)

Complete Kit Assembly

The expansion board connects seamlessly to the XIAO ESP32S3 Sense, creating a comprehensive platform for multimodal machine learning experiments covering vision, audio, and motion sensing.



Please pay attention to the mounting orientation of the module:



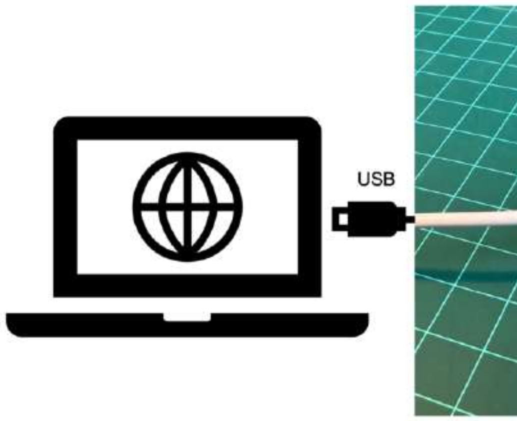
Note that

- The `EN` connection, shown at the bottom of the ESP32S3 Sense, is available on the expansion board via the `RST` button.
- The `BAT+` and `BAT-` connections are also available through the `BAT3.7V` white connector.

XIAOML Kit Applications:

- **Vision**: Image classification and object detection using the integrated camera
- **Audio**: Keyword spotting and voice recognition with the built-in microphone
- **Motion**: Activity recognition and anomaly detection using the IMU sensors
- **Multi-modal**: Combined sensor fusion for complex ML applications

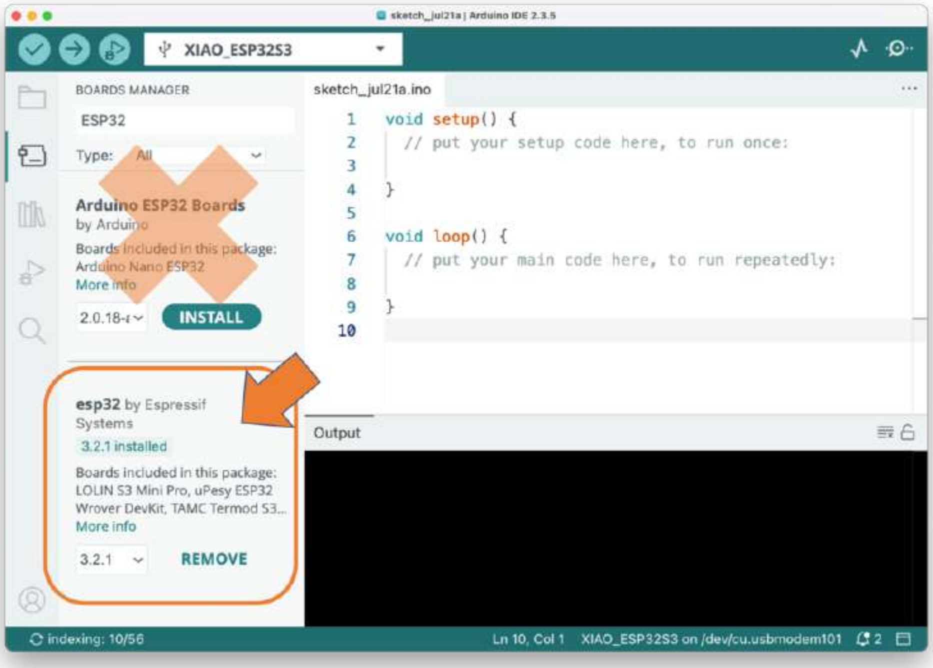
Installing the XIAO ESP32S3 Sense on Arduino IDE



1. Connect the XIAOML Kit to your computer via the USB-C port.
2. Download and Install the stable version of Arduino IDE according to your operating system.

   [Download Arduino IDE]
3. Open the **Arduino IDE** and select the Boards Manager (represented by the `UNO Icon`).
4. Enter "*ESP32*", and select"**esp32 by Espressif Systems**." You can `install` or `update` the board support packages.

   Do not select "**Arduino ESP32 Boards** by Arduino", which are the support package for the Arduino Nano ESP32 and not our board.



ESP32

Type: All

Arduino ESP32 Boards
by Arduino
Boards included in this package:
Arduino Nano ESP32
More info
2.0.18-a INSTALL

esp32 by Espressif Systems
3.2.1 installed
Boards included in this package:
LOLIN S3 Mini Pro, uPesy ESP32 Wrover DevKit, TAMC Termod S3...
More info
3.2.1 REMOVE

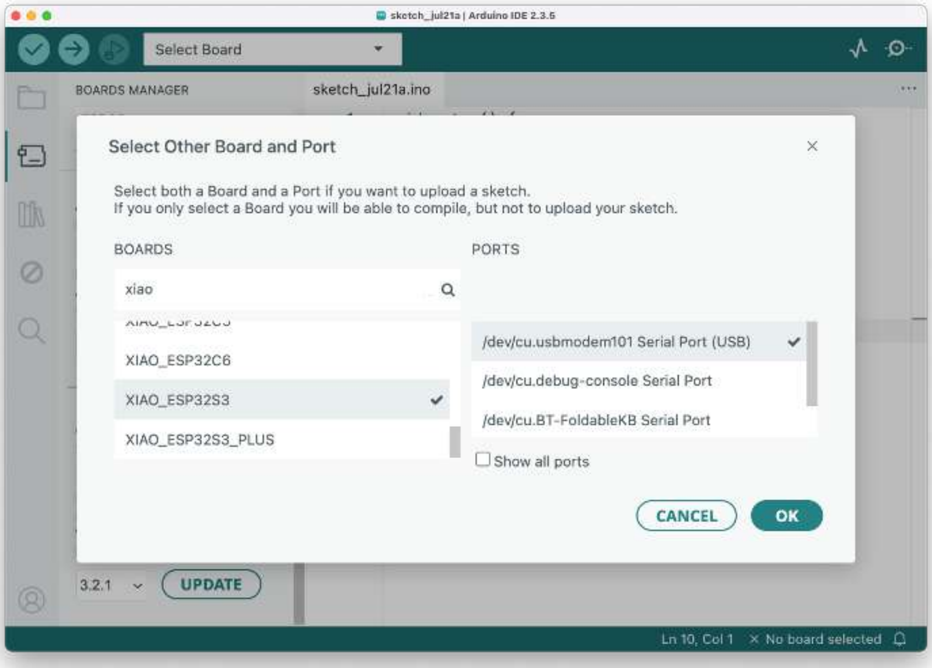
```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

indexing: 10/56   Ln 10, Col 1   XIAO_ESP32S3 on /dev/cu.usbmodem101   2

Attention

Versions 3.x may experience issues when using the XIAO ESP32S3 Sense with Edge Impulse deploy codes. If this is the case, use the last 2.0.x stable version (for example, 2.0.17) instead.

5. Click `Select Board`, enter with *xiao* or *esp32s3*, and select the `XIAO_ESP32S3` in the boards manager and the corresponding PORT where the ESP32S3 is connected.



Select Other Board and Port

Select both a Board and a Port if you want to upload a sketch.
If you only select a Board you will be able to compile, but not to upload your sketch.

BOARDS

xiao

XIAO_ESP32C6
XIAO_ESP32S3 ✓
XIAO_ESP32S3_PLUS

PORTS

/dev/cu.usbmodem101 Serial Port (USB) ✓
/dev/cu.debug-console Serial Port
/dev/cu.BT-FoldableKB Serial Port

☐ Show all ports

CANCEL
OK

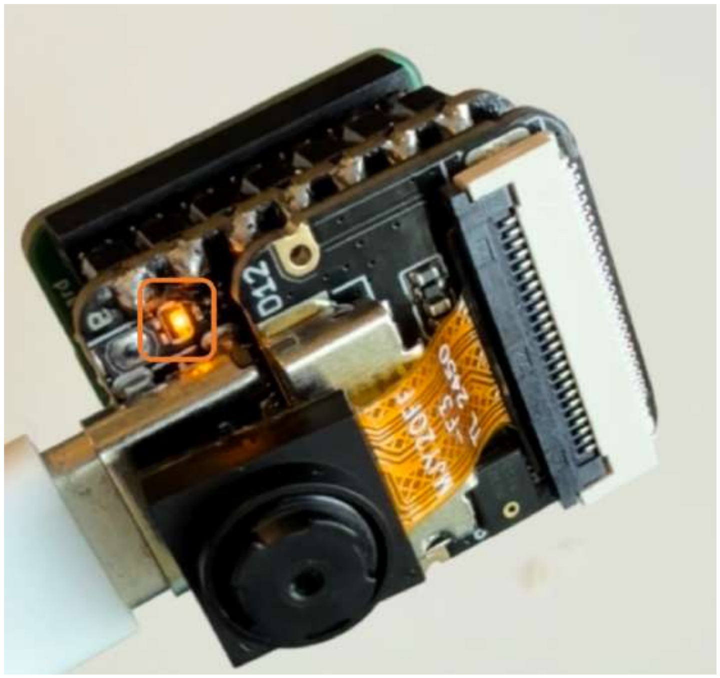
That is it! The device should be OK. Let's do some tests.

Testing the board with BLINK

The XIAO ESP32S3 Sense features a built-in LED connected to GPIO21. So, you can run the blink sketch (which can be found under `Files/Examples/Basics/Blink`. The sketch uses the `LED_BUILTIN` Arduino constant, which internally corresponds to the LED connected to pin 21. Alternatively, you can change the Blink sketch accordingly.

```
#define LED_BUILT_IN 21 // This line is optional

void setup() {
  pinMode(LED_BUILT_IN, OUTPUT); // Set the pin as output
}

// Remember that the pins work with inverted logic
// LOW to turn on and HIGH to turn off
void loop() {
  digitalWrite(LED_BUILT_IN, LOW); //Turn on
  delay (1000); //Wait 1 sec
  digitalWrite(LED_BUILT_IN, HIGH); //Turn off
  delay (1000); //Wait 1 sec
}
```

Note that the pins operate with inverted logic: LOW turns on and HIGH turns off.



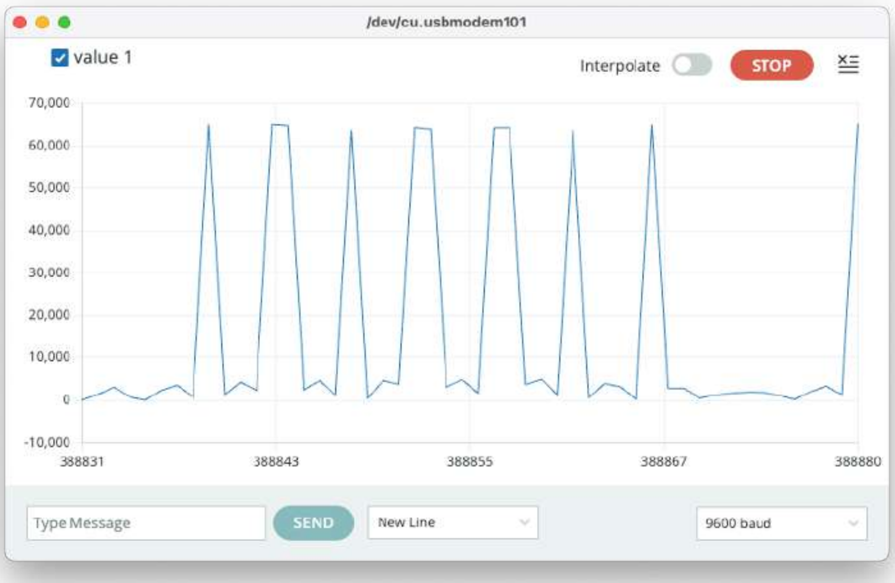
Microphone Test

Let's start with sound detection. Enter with the code below or go to the GitHub project and download the sketch: XIAOML_Kit_Mic_Test and run it on the Arduino IDE:

```
/*
  XIAO ESP32S3 Simple Mic Test
  (for ESP32 Library version 3.0.x and later)
*/

#include <ESP_I2S.h>
I2SClass I2S;
```

```
void setup() {
  Serial.begin(115200);
  while (!Serial) {
    }

  // setup 42 PDM clock and 41 PDM data pins
  I2S.setPinsPdmRx(42, 41);

  // start I2S at 16 kHz with 16-bits per sample
  if (!I2S.begin(I2S_MODE_PDM_RX,
                 16000,
                 I2S_DATA_BIT_WIDTH_16BIT,
                 I2S_SLOT_MODE_MONO)) {
    Serial.println("Failed to initialize I2S!");
    while (1); // do nothing
  }
}


void loop() {
  // read a sample
  int sample = I2S.read();

  if (sample && sample != -1 && sample != 1) {
    Serial.println(sample);
  }
}
```

Open the **Serial Plotter,** and you will see the loudness change curve of the sound.



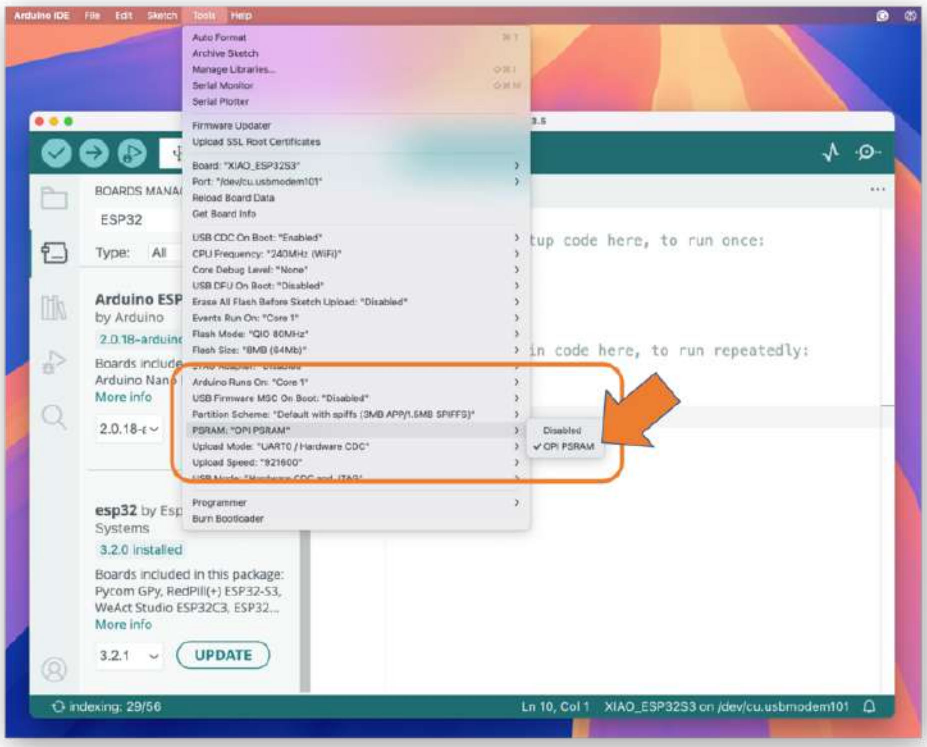
When producing sound, you can verify it on the Serial Plotter.

Save recorded sound (.wav audio files) to a microSD card.

Now, using the onboard SD Card reader, we can save .wav audio files. To do that, we need first to enable the XIAO PSRAM.

> ESP32-S3 has only a few hundred kilobytes of internal RAM on the MCU chip. This can be insufficient for some purposes, so up to 16 MB of external PSRAM (pseudo-static RAM) can be connected with the SPI flash chip (The XIAO has 8 MB of PSRAM). The external memory is incorporated in the memory map and, with certain restrictions, is usable in the same way as internal data RAM.

- To turn it on, go to `Tools->PSRAM:"OPI PSRAM"->OPI PSRAM`



Board: "XIAO_ESP32S3"
Port: "/dev/cu.usbmodem101"
USB CDC On Boot: "Enabled"
CPU Frequency: "240MHz (WiFi)"
Core Debug Level: "None"
USB DFU On Boot: "Disabled"
Erase All Flash Before Sketch Upload: "Disabled"
Events Run On: "Core 1"
Flash Mode: "QIO 80MHz"
Flash Size: "8MB (64Mb)"
Arduino Runs On: "Core 1"
USB Firmware MSC On Boot: "Disabled"
Partition Scheme: "Default with spiffs (3MB APP/1.5MB SPIFFS)"
PSRAM: "OPI PSRAM"
Upload Mode: "UART0 / Hardware CDC"
Upload Speed: "921600"

Disabled
✓ OPI PSRAM

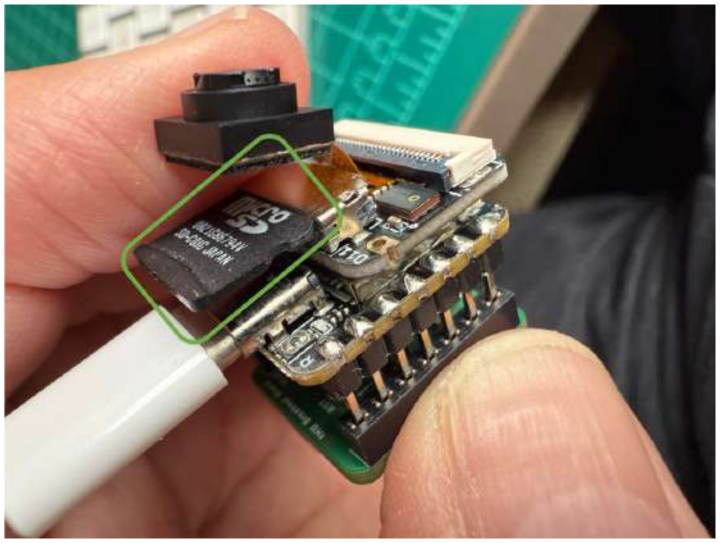
Arduino ESP32 by Arduino 2.0.18-arduino

esp32 by Espressif Systems 3.2.0 installed
Boards included in this package: Pycom GPy, RedPill(+) ESP32-S3, WeAct Studio ESP32C3, ESP32...
3.2.1

```
tup code here, to run once:
in code here, to run repeatedly:
```

XIAO ESP32S3 Sense supports microSD cards up to **32GB**. If you are ready to purchase a microSD card for XIAO, please refer to the specifications below. Format the microSD card to **FAT32 format** before using it.

Now, insert the FAT32 formatted SD card into the XIAO as shown in the photo below

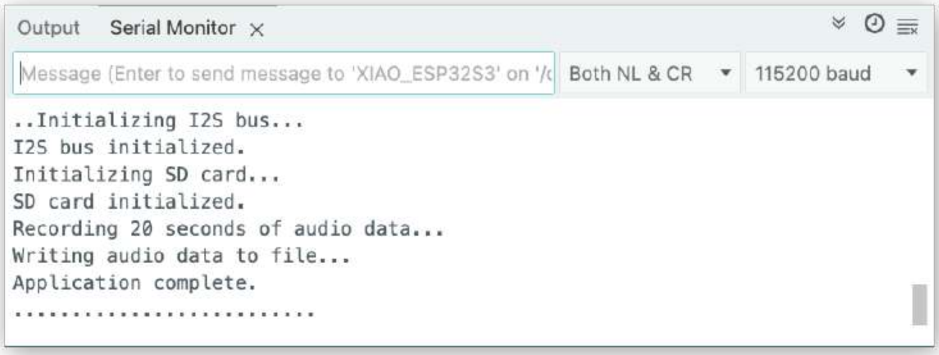


```
/*
 * WAV Recorder for Seeed XIAO ESP32S3 Sense
 * (for ESP32 Library version 3.0.x and later)
*/

#include "ESP_I2S.h"
#include "FS.h"
#include "SD.h"

void setup() {
  // Create an instance of the I2SClass
  I2SClass i2s;

  // Create variables to store the audio data
  uint8_t *wav_buffer;
  size_t wav_size;

  // Initialize the serial port
  Serial.begin(115200);
  while (!Serial) {
    delay(10);
  }

  Serial.println("Initializing I2S bus...");

  // Set up the pins used for audio input
```

```
i2s.setPinsPdmRx(42, 41);

// start I2S at 16 kHz with 16-bits per sample
if (!i2s.begin(I2S_MODE_PDM_RX,
               16000,
               I2S_DATA_BIT_WIDTH_16BIT,
               I2S_SLOT_MODE_MONO)) {
  Serial.println("Failed to initialize I2S!");
  while (1); // do nothing
}

Serial.println("I2S bus initialized.");
Serial.println("Initializing SD card...");

// Set up the pins used for SD card access
if(!SD.begin(21)){
  Serial.println("Failed to mount SD Card!");
  while (1) ;
}
Serial.println("SD card initialized.");
Serial.println("Recording 20 seconds of audio data...");

// Record 20 seconds of audio data
wav_buffer = i2s.recordWAV(20, &wav_size);

// Create a file on the SD card
File file = SD.open("/arduinor_rec.wav", FILE_WRITE);
if (!file) {
  Serial.println("Failed to open file for writing!");
  return;
}

Serial.println("Writing audio data to file...");

// Write the audio data to the file
if (file.write(wav_buffer, wav_size) != wav_size) {
  Serial.println("Failed to write audio data to file!");
  return;
}

// Close the file
```

```
  file.close();

  Serial.println("Application complete.");
}

void loop() {
  delay(1000);
  Serial.printf(".");
}
```

- Save the code, for example, as `Wav_Record.ino`, and run it in the Arduino IDE.
- This program is executed only once after the user turns on the serial monitor (or when the `RESET` button is pressed). It records for 20 seconds and saves the recording file to a microSD card as "arduino_rec.wav."
- When the "." is output every second in the serial monitor, the program execution is complete, and you can play the recorded sound file using a card reader.



```
..Initializing I2S bus...
I2S bus initialized.
Initializing SD card...
SD card initialized.
Recording 20 seconds of audio data...
Writing audio data to file...
Application complete.
........................
```

The sound quality is excellent!

> The explanation of how the code works is beyond the scope of this lab, but you can find an excellent description on the wiki page.

To know more about the File System on the XIAO ESP32S3 Sense, please refer to this link.

Testing the Camera

For testing (and using the camera, we can use several methods:

- The SenseCraft AI Studio
- The CameraWebServer app on Arduino IDE (See the next section)

- Capturing images and saving them on an SD card (similar to what we did with audio)

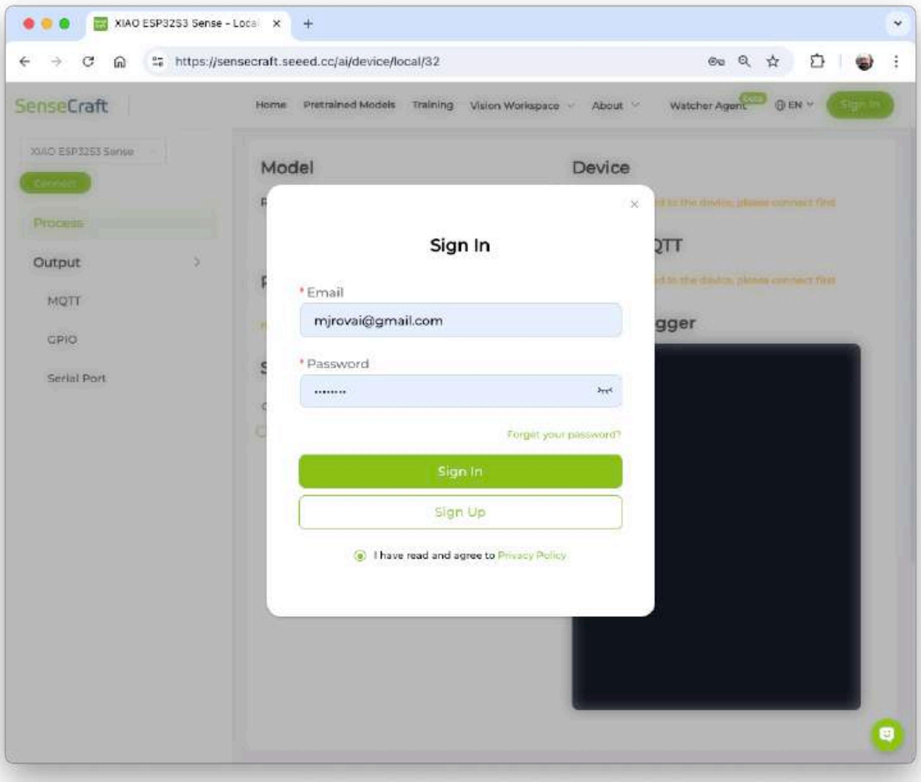
Testing the camera with the SenseCraft AI Studio

The easiest way to see the camera working is to use the SenseCraft AI Studio, a robust platform that offers a wide range of AI models compatible with various devices, including the **XIAO ESP32S3 Sense** and the Grove Vision AI V2.

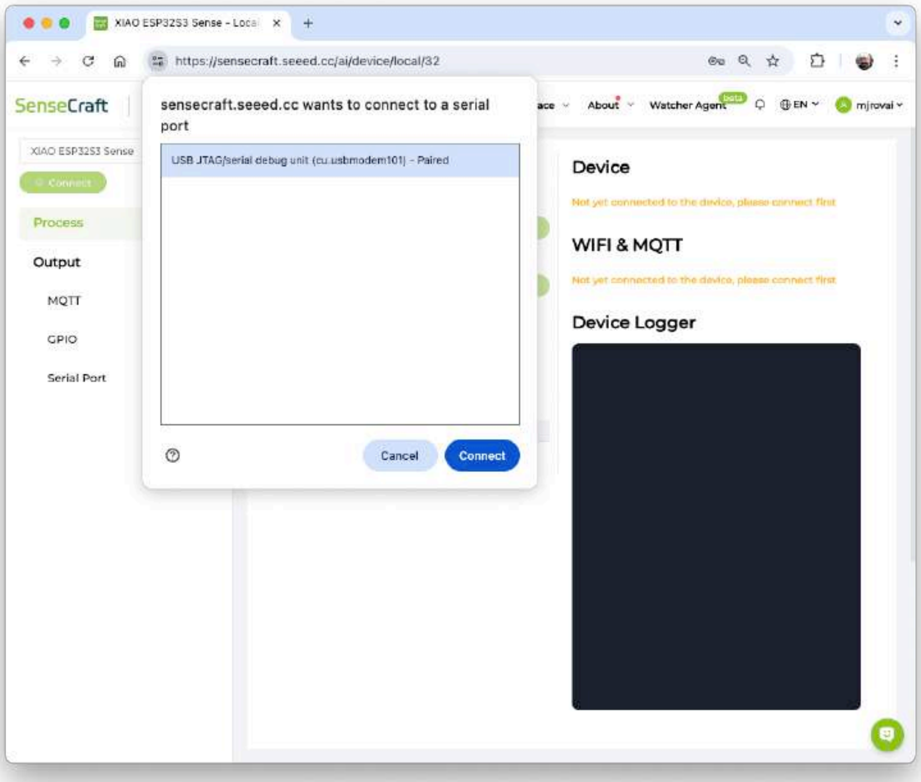
> We can also use the **SenseCraft Web Toolkit**, a simplified version of the SenseCraft AI Studio.

Let's follow the steps below to start the **SenseCraft AI**:

- Open the SenseCraft AI Vision Workspace in a web browser, such as **Chrome**, and sign in (or create an account).

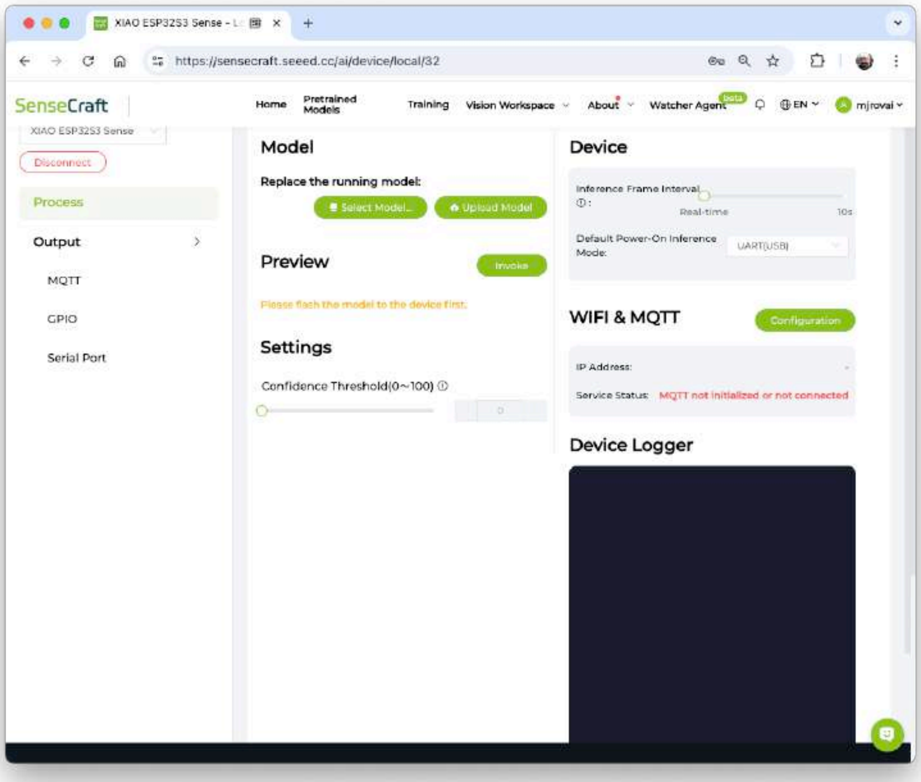


- Having the XIAOML Kit physically connected to the notebook, select it as below:


XIAO ESP32S3 Sense - Local
https://sensecraft.seeed.cc/ai/device/local/32
SenseCraft
sensecraft.seeed.cc wants to connect to a serial port
USB JTAG/serial debug unit (cu.usbmodem101) - Paired
Cancel
Connect
XIAO ESP32S3 Sense
Connect
Process
Output
MQTT
GPIO
Serial Port
About
Watcher Agent
EN
mjrovai
Device
Not yet connected to the device, please connect first
WIFI & MQTT
Not yet connected to the device, please connect first
Device Logger

> Note: The **WebUSB tool** may not function correctly in certain browsers, such as Safari. Use Chrome instead. Also, confirm that the Arduino IDE or any other serial device is not connected to the XIAO.

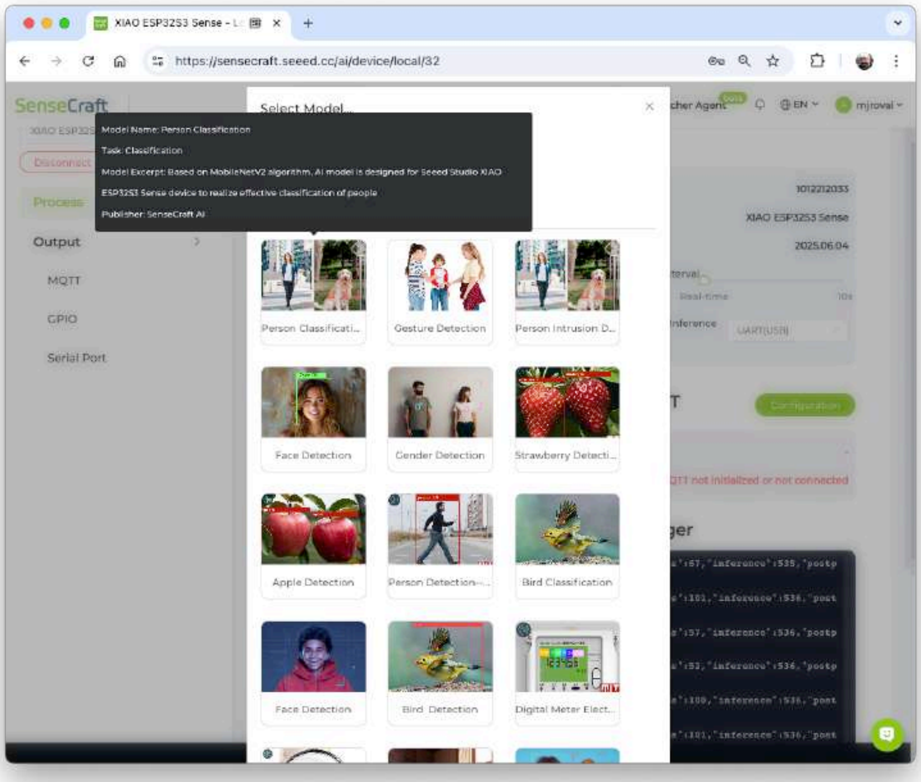
To see the camera working, we should upload a model. We can try several Computer Vision models previously uploaded by Seeed Studio. Use the button `[Select Model]` and choose among the available models.



Please flash the model to the device first.

Service Status: MQTT not initialized or not connected

Passing the cursor over the AI models, we can have some information about them, such as name, description, **category** or **task** (Image Classification, Object Detection, or Pose/Keypoint Detection), the **algorithm** (like YOLO V5 or V8, FOMO, MobileNet V2, etc.) and in some cases, **metrics** (Accuracy or mAP).



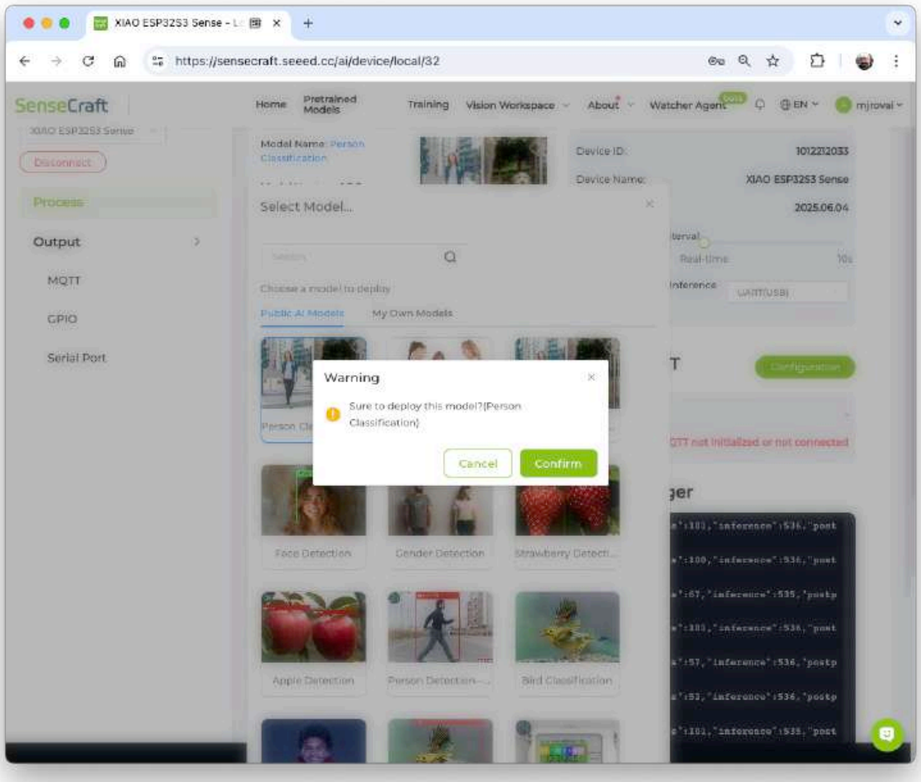
Model Name: Person Classification

Task: Classification

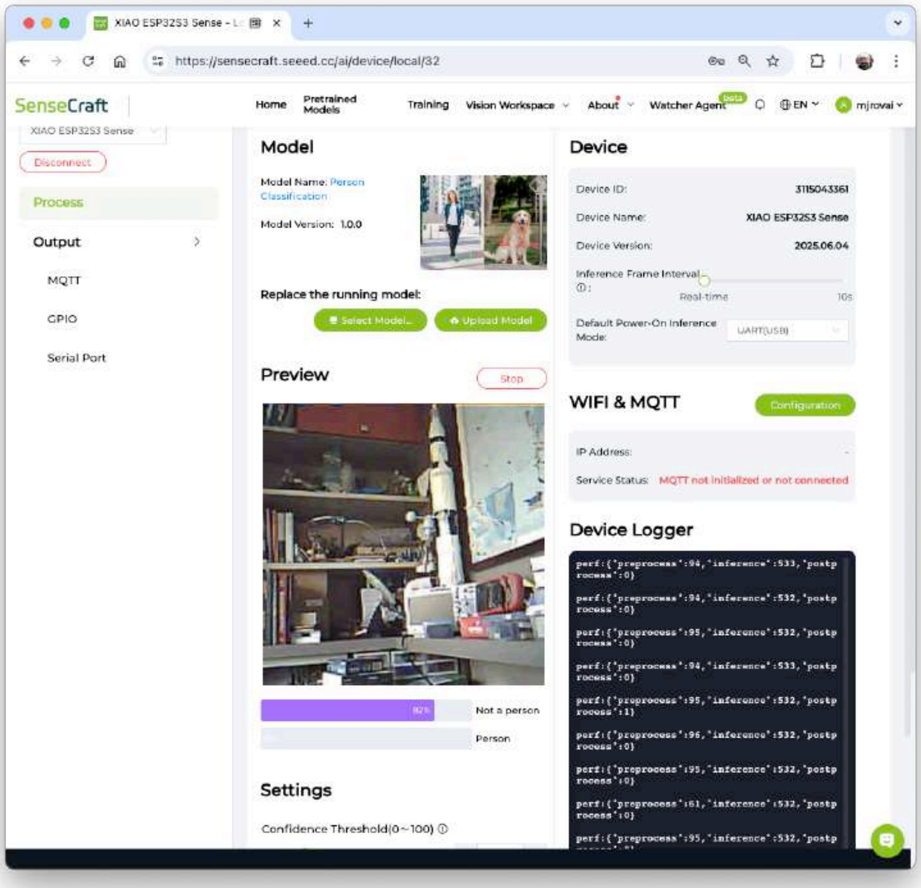
Model Excerpt: Based on MobileNetV2 algorithm, AI model is designed for Seeed Studio XIAO ESP32S3 Sense device to realize effective classification of people

Publisher: SenseCraft AI

We can choose one of the ready-to-use AI models, such as "Person Classification", by clicking on it and pressing the [Confirm] button, or upload our own model.



In the **Preview** Area, we can see the streaming generated by the camera.



Model

Model Name: Person Classification

Model Version: 1.0.0

Replace the running model:

Preview

82% Not a person

Person

Settings

Confidence Threshold(0~100)

Device

Device ID:	3115043361
Device Name:	XIAO ESP32S3 Sense
Device Version:	2025.06.04
Inference Frame Interval:	Real-time … 10s
Default Power-On Inference Mode:	UART(USB)

WIFI & MQTT

IP Address:	-
Service Status:	MQTT not initialized or not connected

Device Logger

```
perf:{'preprocess':94,'inference':533,'postp
rocess':0}
perf:{'preprocess':94,'inference':532,'postp
rocess':0}
perf:{'preprocess':95,'inference':532,'postp
rocess':0}
perf:{'preprocess':94,'inference':533,'postp
rocess':0}
perf:{'preprocess':95,'inference':532,'postp
rocess':1}
perf:{'preprocess':96,'inference':532,'postp
rocess':0}
perf:{'preprocess':95,'inference':532,'postp
rocess':0}
perf:{'preprocess':61,'inference':532,'postp
rocess':0}
perf:{'preprocess':95,'inference':532,'postp
```

We will return to the SenseCraft AI Studio in more detail during the Vision AI labs.

Testing WiFi

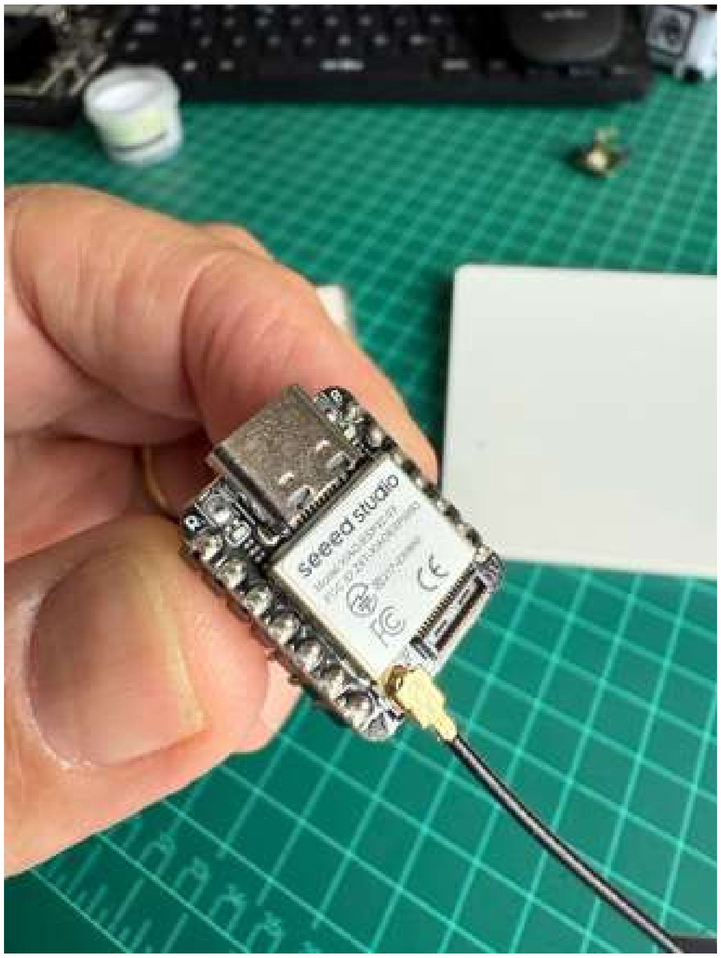
Installation of the antenna

The XIAOML Kit arrived fully assembled. First, remove the Sense Expansion Board (which contains the Camera, Mic, and SD Card Reader) from the XIAO.

On the bottom left of the front of XIAO ESP32S3, there is a separate "WiFi/BT Antenna Connector". To improve your WiFi/Bluetooth signal, remove the antenna from the package and attach it to the connector.

There is a small trick to installing the antenna. If you press down hard on it directly, you will find it very difficult to press and your fingers will hurt! The correct way to install the antenna is to insert one side of the antenna connector into the connector block first, then gently press down on the other side to ensure the antenna is securely installed.

Removing the antenna is also the case. Do not use brute force to pull the antenna directly; instead, apply force to one side to lift, making the antenna easy to remove.

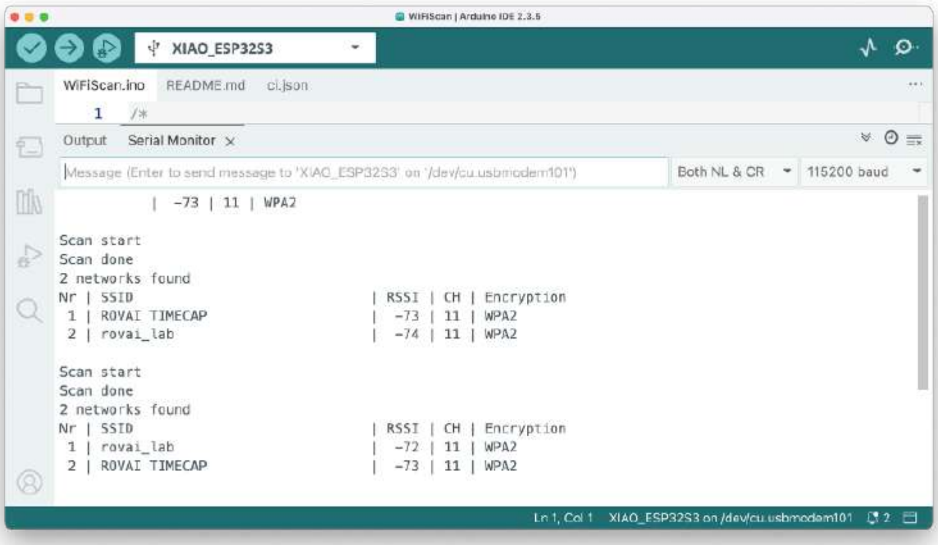


Reinstalling the expansion board is very simple; you just need to align the connector on the expansion board with the B2B connector on the XIAO ESP32S3, press it hard, and hear a "click." The installation is complete.

One of the XIAO ESP32S3's differentiators is its WiFi capability. So, let's test its radio by scanning the Wi-Fi networks around it. You can do this by running one of the code examples on the board.

Open the Arduino IDE and select our board and port. Go to Examples and look for **WiFI ==> WiFIScan** under the "Examples for the XIAO ESP32S3". Upload the sketch to the board.

You should see the Wi-Fi networks (SSIDs and RSSIs) within your device's range on the serial monitor. Here is what I got in the lab:


WiFiScan | Arduino IDE 2.3.5
XIAO_ESP32S3
WiFiScan.ino
README.md
ci.json
Output
Serial Monitor
Message (Enter to send message to 'XIAO_ESP32S3' on '/dev/cu.usbmodem101')
Both NL & CR
115200 baud
Ln 1, Col 1 XIAO_ESP32S3 on /dev/cu.usbmodem101

```
          |  -73 | 11 | WPA2

Scan start
Scan done
2 networks found
Nr | SSID                             | RSSI | CH | Encryption
 1 | ROVAI TIMECAP                    |  -73 | 11 | WPA2
 2 | rovai_lab                        |  -74 | 11 | WPA2

Scan start
Scan done
2 networks found
Nr | SSID                             | RSSI | CH | Encryption
 1 | rovai_lab                        |  -72 | 11 | WPA2
 2 | ROVAI TIMECAP                    |  -73 | 11 | WPA2
```

Simple WiFi Server (Turning LED ON/OFF)

Let's test the device's capability to behave as a Wi-Fi server. We will host a simple page on the device that sends commands to turn the XIAO built-in LED ON and OFF.

Go to Examples and look for **WiFI ==> SimpleWiFIServer** under the "Examples for the XIAO ESP32S3".

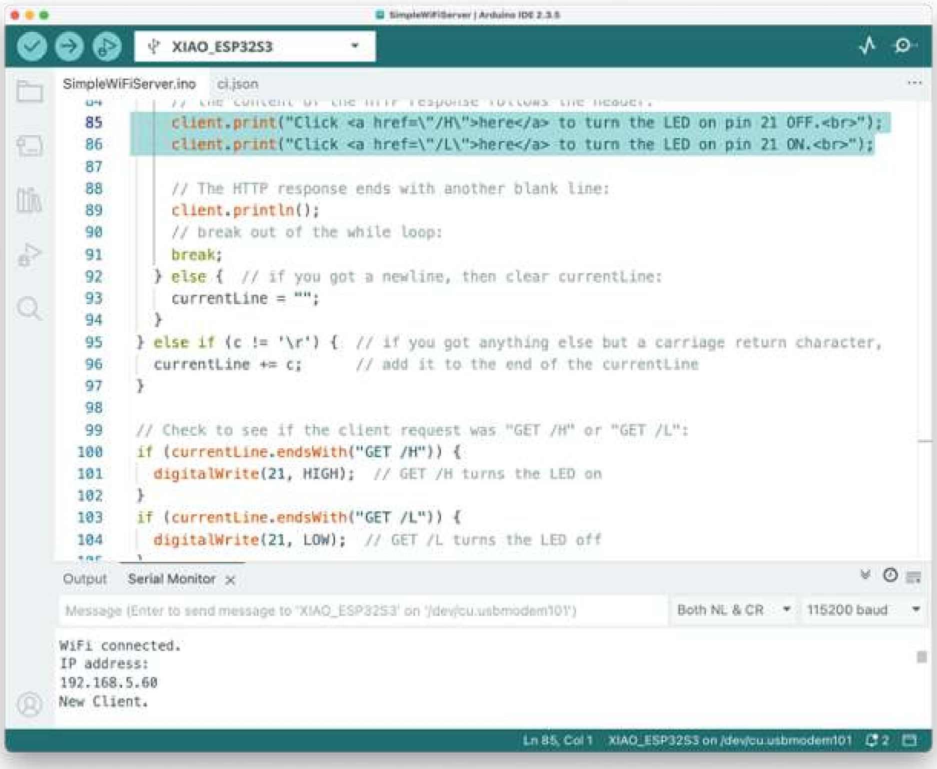
Before running the sketch, you should enter your network credentials:

```
const char* ssid     = "Your credentials here";
const char* password = "Your credentials here";
```

And modify **pin 5** to **pin 21**, where the built-in LED is installed. Also, let's modify the webpage (lines 85 and 86) to reflect the correct LED Pin and that it is active with LOW:

```
client.print("Click <a href=\"/H\">here</a> to turn the LED on pin 21 OFF.<br>");
client.print("Click <a href=\"/L\">here</a> to turn the LED on pin 21 ON.<br>");
```

You can monitor your server's performance using the Serial Monitor.

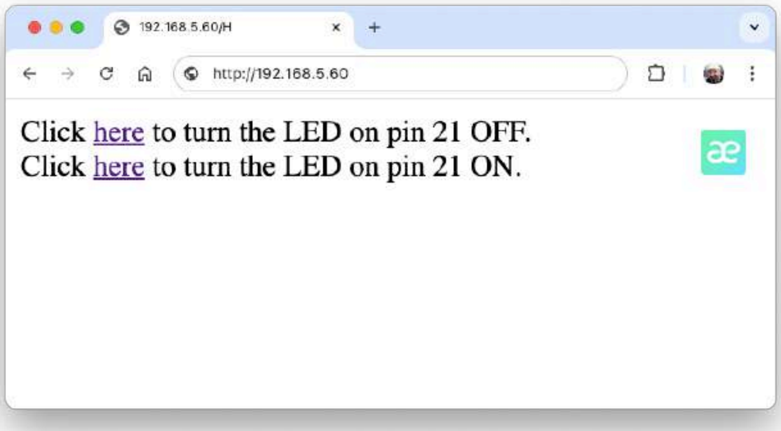


```
    // the content of the HTTP response follows the header:
    client.print("Click <a href=\"/H\">here</a> to turn the LED on pin 21 OFF.<br>");
    client.print("Click <a href=\"/L\">here</a> to turn the LED on pin 21 ON.<br>");

    // The HTTP response ends with another blank line:
    client.println();
    // break out of the while loop:
    break;
  } else {  // if you got a newline, then clear currentLine:
    currentLine = "";
  }
} else if (c != '\r') {  // if you got anything else but a carriage return character,
  currentLine += c;      // add it to the end of the currentLine
}

// Check to see if the client request was "GET /H" or "GET /L":
if (currentLine.endsWith("GET /H")) {
  digitalWrite(21, HIGH);  // GET /H turns the LED on
}
if (currentLine.endsWith("GET /L")) {
  digitalWrite(21, LOW);  // GET /L turns the LED off
```

Output
Serial Monitor
Message (Enter to send message to 'XIAO_ESP32S3' on '/dev/cu.usbmodem101')
Both NL & CR
115200 baud

```
WiFi connected.
IP address:
192.168.5.60
New Client.
```

Take the IP address shown in the Serial Monitor and enter it in your browser. You will see a page with links that can turn the built-in LED of your XIAO ON and OFF.



Click here to turn the LED on pin 21 OFF.
Click here to turn the LED on pin 21 ON.

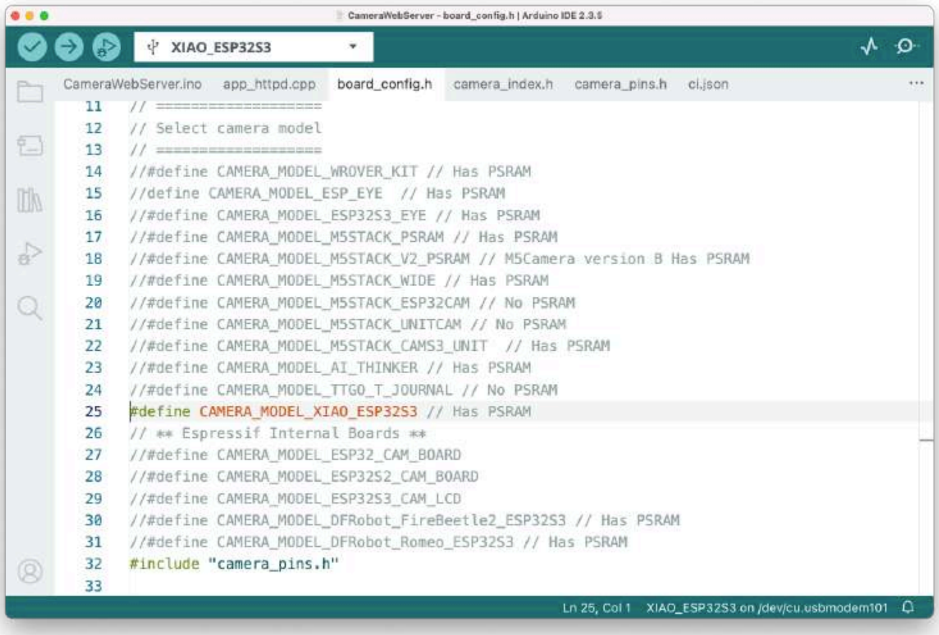
Using the CameraWebServer

In the Arduino IDE, go to `File > Examples > ESP32 > Camera`, and select `CameraWebServer`

On the `board_config.h` tab, comment on all cameras' models, except the XIAO model pins:

`#define CAMERA_MODEL_XIAO_ESP32S3 // Has PSRAM`

Do not forget to check the `Tools` to see if PSRAM is enabled.


CameraWebServer - board_config.h | Arduino IDE 2.3.5
XIAO_ESP32S3
CameraWebServer.ino
app_httpd.cpp
board_config.h
camera_index.h
camera_pins.h
ci.json
Ln 25, Col 1   XIAO_ESP32S3 on /dev/cu.usbmodem101

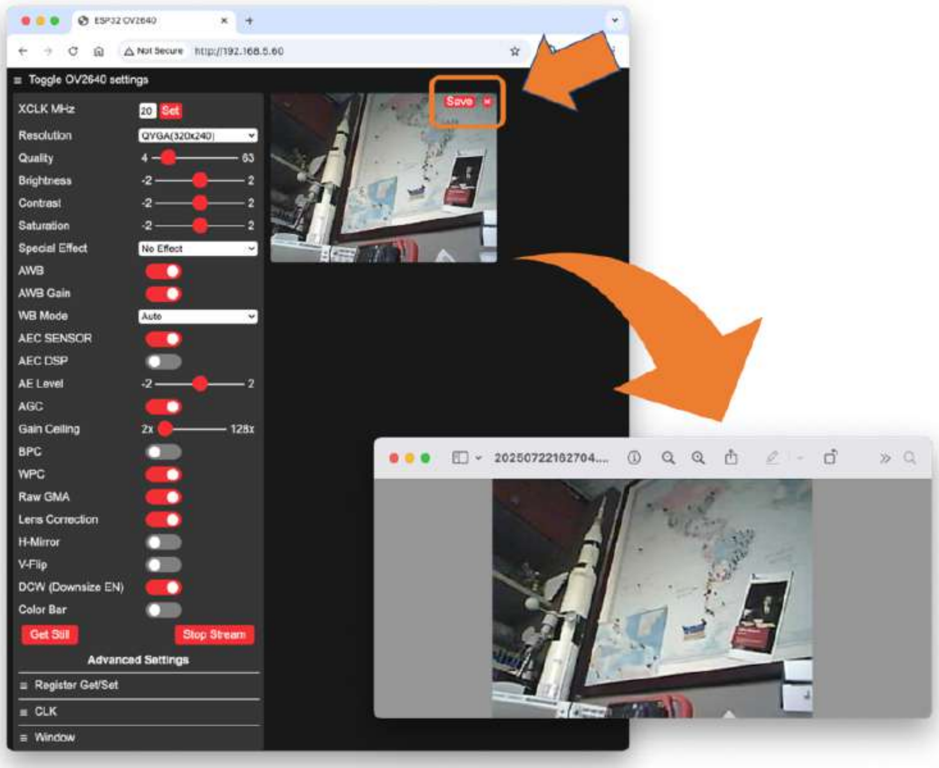
```
// ===================
// Select camera model
// ===================
//#define CAMERA_MODEL_WROVER_KIT // Has PSRAM
//define CAMERA_MODEL_ESP_EYE  // Has PSRAM
//#define CAMERA_MODEL_ESP32S3_EYE // Has PSRAM
//#define CAMERA_MODEL_M5STACK_PSRAM // Has PSRAM
//#define CAMERA_MODEL_M5STACK_V2_PSRAM // M5Camera version B Has PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE // Has PSRAM
//#define CAMERA_MODEL_M5STACK_ESP32CAM // No PSRAM
//#define CAMERA_MODEL_M5STACK_UNITCAM // No PSRAM
//#define CAMERA_MODEL_M5STACK_CAMS3_UNIT  // Has PSRAM
//#define CAMERA_MODEL_AI_THINKER // Has PSRAM
//#define CAMERA_MODEL_TTGO_T_JOURNAL // No PSRAM
#define CAMERA_MODEL_XIAO_ESP32S3 // Has PSRAM
// ** Espressif Internal Boards **
//#define CAMERA_MODEL_ESP32_CAM_BOARD
//#define CAMERA_MODEL_ESP32S2_CAM_BOARD
//#define CAMERA_MODEL_ESP32S3_CAM_LCD
//#define CAMERA_MODEL_DFRobot_FireBeetle2_ESP32S3 // Has PSRAM
//#define CAMERA_MODEL_DFRobot_Romeo_ESP32S3 // Has PSRAM
#include "camera_pins.h"
```

As done before, in the `CameraWebServer.ino` tab, enter your wifi credentials and upload the code to the device.

If the code is executed correctly, you should see the address on the Serial Monitor:

```
WiFi connecting....
WiFi connected
Camera Ready! Use 'http://192.168.5.60' to connect
```

Copy the address into your browser and wait for the page to load. Select the camera resolution (for example, QVGA) and select `[START STREAM]`. Wait for a few seconds, depending on your connection. Using the `[Save]` button, you can save an image to your computer's download area.



XCLK MHz 20 Set
Resolution QVGA(320x240)
Quality 4 — 63
Brightness -2 — 2
Contrast -2 — 2
Saturation -2 — 2
Special Effect No Effect
AWB
AWB Gain
WB Mode Auto
AEC SENSOR
AEC DSP
AE Level -2 — 2
AGC
Gain Ceiling 2x — 128x
BPC
WPC
Raw GMA
Lens Correction
H-Mirror
V-Flip
DCW (Downsize EN)
Color Bar

That's it! You can save the images directly on your computer for use on projects.

Testing the IMU Sensor (LSM6DS3TR-C)

An **Inertial Measurement Unit (IMU)** is a sensor that measures motion and orientation. The LSM6DS3TR-C on your XIAOML kit is a **6-axis IMU**, meaning it combines:

- **3-axis Accelerometer**: Measures linear acceleration (including gravity) along X, Y, and Z axes
- **3-axis Gyroscope**: Measures angular velocity (rotation rate) around X, Y, and Z axes

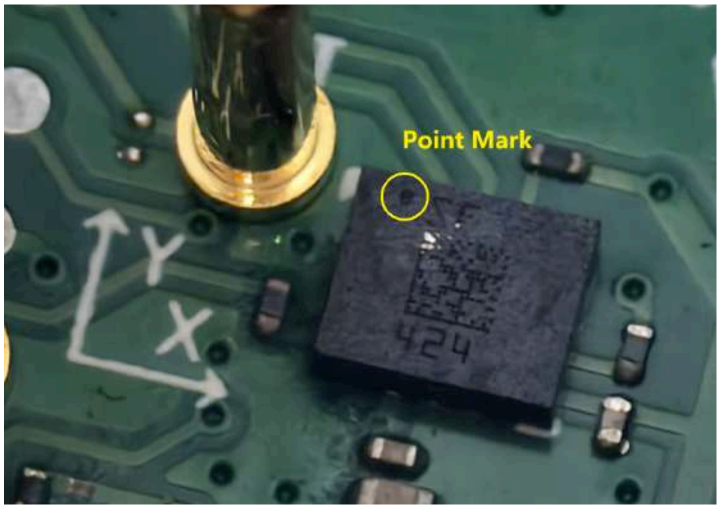
Technical Specifications:

- **Communication**: I2C interface at address `0x6A`
- **Accelerometer Range**: ±2/±4/±8/±16 g (we use ±2g by default)
- **Gyroscope Range**: ±125/±250/±500/±1000/±2000 dps (we use ±250 dps by default)
- **Resolution**: 16-bit ADC

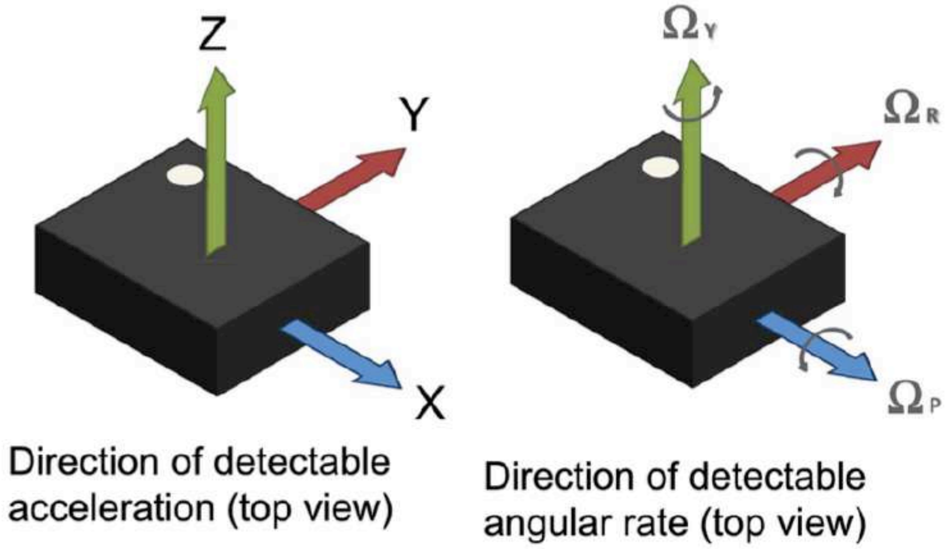
- **Power Consumption**: Ultra-low power design

Coordinate System:

The sensor follows a right-hand coordinate system. When looking at the IMU sensor with the point mark visible (Expansion Board bottom view):



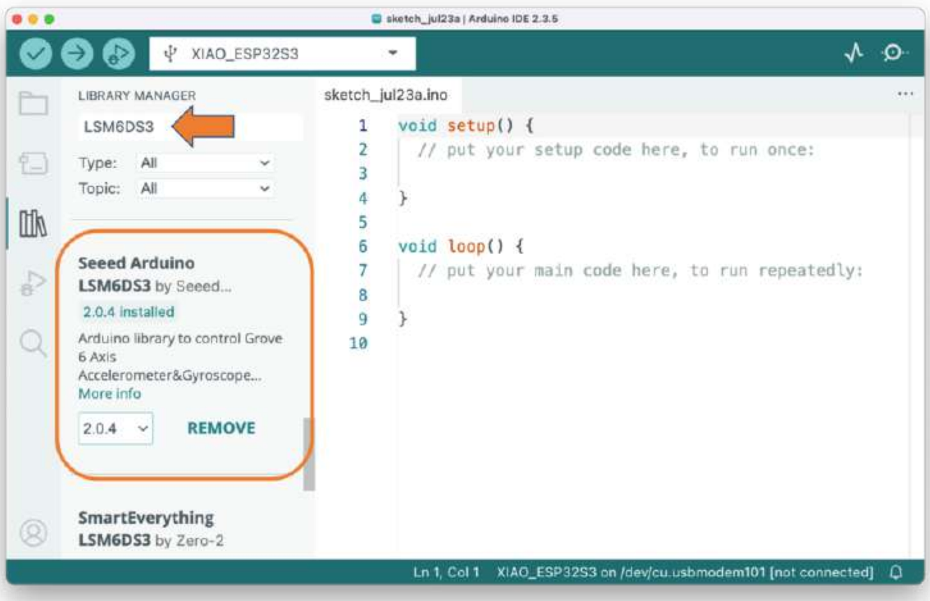
- **X-axis**: Points to the right
- **Y-axis**: Points forward (away from you)
- **Z-axis**: Points upward (out of the board)


Z
Y
X
Direction of detectable acceleration (top view)
Ω Y
Ω R
Ω P
Direction of detectable angular rate (top view)

Required Libraries

Before uploading the code, install the required library:

1. Open the **Arduino IDE** and select **Manage Libraries** (represented by the `Books Icon`).
2. For the IMU library, enter “*LSM6DS3*”, and select”**Seeed Arduino LSM6DS3 by Seeed**”. You can `INSTALL` or `UPDATE` the board support packages.


sketch_jul23a | Arduino IDE 2.3.5
XIAO_ESP32S3
LIBRARY MANAGER
LSM6DS3
Type: All
Topic: All
Seeed Arduino LSM6DS3 by Seeed...
2.0.4 installed
Arduino library to control Grove 6 Axis Accelerometer&Gyroscope...
More info
2.0.4
REMOVE
SmartEverything LSM6DS3 by Zero-2
sketch_jul23a.ino
Ln 1, Col 1   XIAO_ESP32S3 on /dev/cu.usbmodem101 [not connected]

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Important: Do NOT install “Arduino_LSM6DS3 by Arduino” - that’s for different boards!

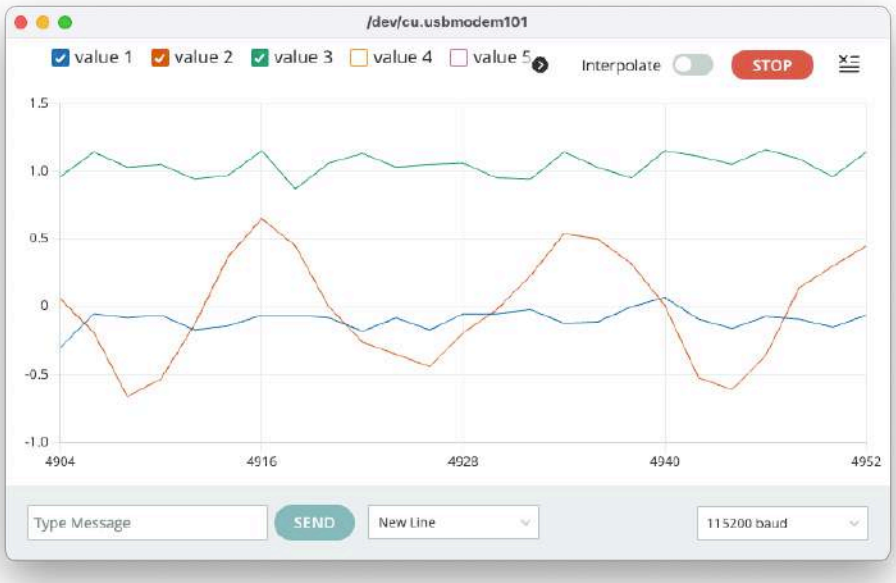
Test Code

Enter with the code below at the Arduino IDE and uploaded it to Kit:

```
#include <LSM6DS3.h>
#include <Wire.h>

// Create IMU object using I2C interface
// LSM6DS3TR-C sensor is located at I2C address 0x6A
LSM6DS3 myIMU(I2C_MODE, 0x6A);
```

```
// Variables to store sensor readings
float accelX, accelY, accelZ;  // Accelerometer values (g-force)
float gyroX, gyroY, gyroZ;     // Gyroscope values (degrees per second)

void setup() {
  // Initialize serial communication at 115200 baud rate
  Serial.begin(115200);

  // Wait for serial port to connect (useful for debugging)
  while (!Serial) {
    delay(10);
  }

  Serial.println("XIAOML Kit IMU Test");
  Serial.println("LSM6DS3TR-C 6-Axis IMU Sensor");
  Serial.println("=============================");

  // Initialize the IMU sensor
  if (myIMU.begin() != 0) {
    Serial.println("ERROR: IMU initialization failed!");
    Serial.println("Check connections and I2C address");
    while(1) {
      delay(1000); // Halt execution if IMU fails to initialize
    }
  } else {
    Serial.println("  IMU initialized successfully");
    Serial.println();

    // Print sensor information
    Serial.println("Sensor Information:");
    Serial.println("- Accelerometer range: ±2g");
    Serial.println("- Gyroscope range: ±250 dps");
    Serial.println("- Communication: I2C at address 0x6A");
    Serial.println();

    // Print data format explanation
    Serial.println("Data Format:");
    Serial.println("AccelX,AccelY,AccelZ,GyroX,GyroY,GyroZ");
    Serial.println("Units: g-force (m/s²), degrees/second");
    Serial.println();
    delay(2000); // Brief pause before starting measurements
  }
}

void loop() {
  // Read accelerometer data (in g-force units)
  accelX = myIMU.readFloatAccelX();
  accelY = myIMU.readFloatAccelY();
  accelZ = myIMU.readFloatAccelZ();

  // Read gyroscope data (in degrees per second)
  gyroX = myIMU.readFloatGyroX();
  gyroY = myIMU.readFloatGyroY();
  gyroZ = myIMU.readFloatGyroZ();

  // Print readable format to Serial Monitor
  Serial.print("Accelerometer (g): ");
  Serial.print("X="); Serial.print(accelX, 3);
  Serial.print(" Y="); Serial.print(accelY, 3);
  Serial.print(" Z="); Serial.print(accelZ, 3);

  Serial.print(" | Gyroscope (°/s): ");
  Serial.print("X="); Serial.print(gyroX, 2);
  Serial.print(" Y="); Serial.print(gyroY, 2);
  Serial.print(" Z="); Serial.print(gyroZ, 2);
  Serial.println();

  // Print CSV format for Serial Plotter
  Serial.println(String(accelX) + "," + String(accelY) + "," +
                 String(accelZ) + "," + String(gyroX) + "," +
                 String(gyroY) + "," + String(gyroZ));

  // Update rate: 10 Hz (100ms delay)
  delay(100);
}
```

The Serial monitor will show the values, and the plotter will show their variation over time. For example, by moving the Kit over the **y-axis**, we will see that value 2 (red line) changes accordingly. Note that **z-axis** is represented by value 3 (green line), which is near 1.0g. The blue line (value 1) is related to the **x-axis**.



You can select the values 4 to 6 to see the Gyroscope behavior.

Testing the OLED Display (SSD1306)

OLED (Organic Light-Emitting Diode) displays are self-illuminating screens where each pixel produces its own light. The XIAO ML kit features a compact 0.42-inch monochrome OLED display, ideal for displaying sensor data, status information, and simple graphics.

Technical Specifications:

- **Size**: 0.42 inches diagonal
- **Resolution**: 72 × 40 pixels
- **Controller**: SSD1306
- **Interface**: I2C at address `0x3C`
- **Colors**: Monochrome (black pixels on white background, or vice versa)
- **Viewing**: High contrast, visible in bright light
- **Power**: Low power consumption, no backlight needed

Display Characteristics:

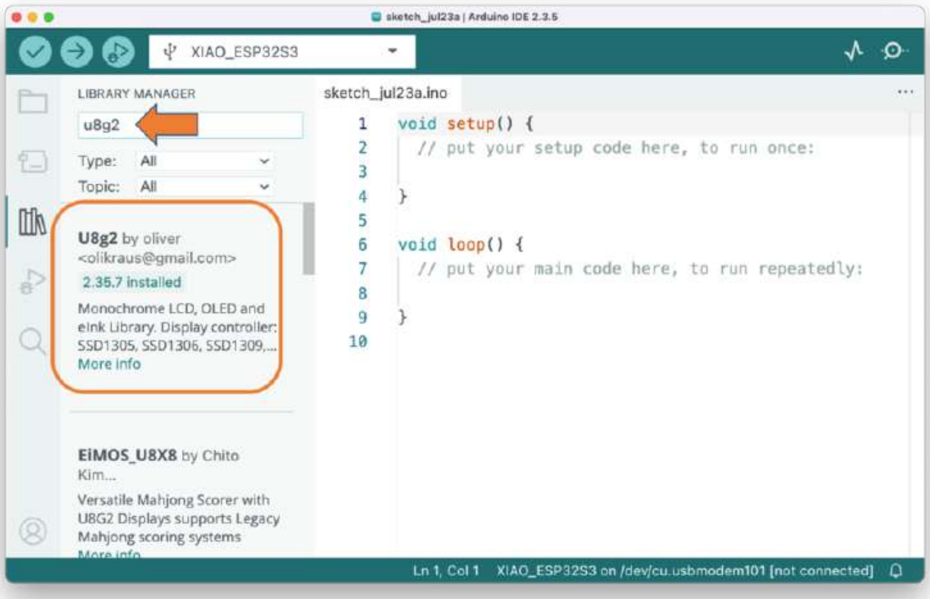
- **Pixel-perfect**: Each of the 2,880 pixels (72×40) can be individually controlled
- **Fast refresh**: Suitable for animations and real-time data
- **No ghosting**: Instant pixel response
- **Wide viewing angle**: Clear from multiple viewing positions

Required Libraries

Before uploading the code, install the required library:

1. Open the **Arduino IDE** and select the "Manage Libraries" (represented by the `Books Icon`).
2. Enter *u8g2* and select **U8g2 by oliver**. You can `install` or `update` the board support packages.

   Note: U8g2 is a powerful graphics library supporting many display types



sketch_jul23a | Arduino IDE 2.3.5

XIAO_ESP32S3

LIBRARY MANAGER

u8g2

Type: All

Topic: All

U8g2 by oliver
<olikraus@gmail.com>
2.35.7 installed
Monochrome LCD, OLED and eInk Library. Display controller: SSD1305, SSD1306, SSD1309,...
More info

EiMOS_U8X8 by Chito Kim...
Versatile Mahjong Scorer with U8G2 Displays supports Legacy Mahjong scoring systems
More info

sketch_jul23a.ino

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Ln 1, Col 1   XIAO_ESP32S3 on /dev/cu.usbmodem101 [not connected]

The **U8g2** library is a monochrome graphics library with these features:

- Support for many display controllers (including SSD1306)
- Text rendering with various fonts
- Drawing primitives (lines, rectangles, circles)
- Memory-efficient page-based rendering

- Hardware and software I2C support

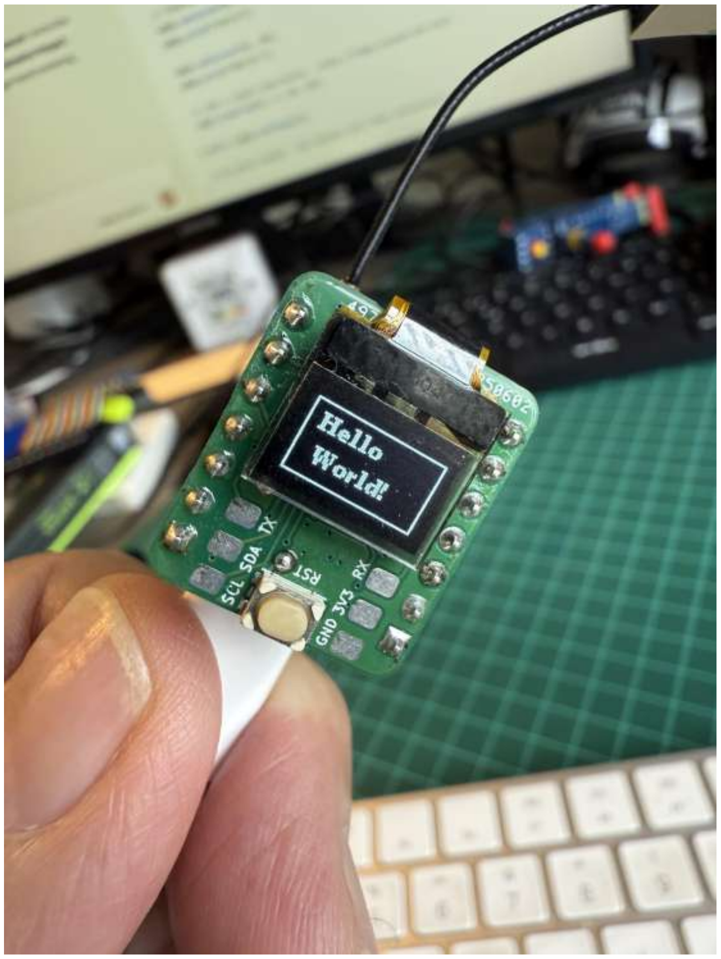
Test Code

Enter with the code below at the Arduino IDE and uploaded it to Kit:

```
#include <U8g2lib.h>
#include <Wire.h>

// Initialize the OLED display
// SSD1306 controller, 72x40 resolution, I2C interface
U8G2_SSD1306_72X40_ER_1_HW_I2C u8g2(U8G2_R2, U8X8_PIN_NONE);

void setup() {
  Serial.begin(115200);

  Serial.println("XIAOML Kit - Hello World");
  Serial.println("==========================");

  // Initialize the display
  u8g2.begin();

  Serial.println("  Display initialized");
  Serial.println("Showing Hello World message...");

  // Clear the display
  u8g2.clearDisplay();
}

void loop() {
  // Start drawing sequence
  u8g2.firstPage();
  do {
    // Set font
    u8g2.setFont(u8g2_font_ncenB08_tr);

    // Display "Hello World" centered
    u8g2.setCursor(8, 15);
    u8g2.print("Hello");

    u8g2.setCursor(12, 30);
```

```
      u8g2.print("World!");

      // Add a simple decoration - draw a frame around the text
      u8g2.drawFrame(2, 2, 68, 36);

    } while (u8g2.nextPage());

    // No delay needed - the display will show continuously
  }
```

If everything works fine, you should see at the display, "Hello World" inside a rectangle.



OLED - Text Sizes and Positioning

- Note that the text is positioned with `setCursor(x, y)`, in this case centered:

```
u8g2.setCursor(8, 15);
```

- The font used in the code was medium.

  ```
  u8g2.setFont(u8g2_font_ncenB08_tr);
  ```

  But other font sizes are available:

  - `u8g2_font_4x6_tr`: Tiny font (4×6 pixels)
  - `u8g2_font_6x10_tr`: Small font (6×10 pixels)
  - `u8g2_font_ncenB08_tr`: Medium bold font
  - `u8g2_font_ncenB14_tr`: Large bold font

Shapes

The code added a simple decoration, drawing a frame around the text

```
u8g2.drawFrame(2, 2, 68, 36);
```

But other shapes are available:

- **Rectangle outline**: `drawFrame(x, y, width, height)`
- **Filled rectangle**: `drawBox(x, y, width, height)`
- **Circle**: `drawCircle(x, y, radius)`
- **Line**: `drawLine(x1, y1, x2, y2)`
- **Individual pixels**: `drawPixel(x, y)`

Coordinates

The display uses a coordinate system where:

- **Origin (0,0)**: Top-left corner
- **X-axis**: Increases from left to right (0 to 71)
- **Y-axis**: Increases from top to bottom (0 to 39)
- **Text positioning**: `setCursor(x, y)` where y is the baseline of text

Display Rotation

- You can change the rotation parameter by using:
  - `U8G2_R0`: Normal orientation
  - `U8G2_R1`: 90° clockwise
  - `U8G2_R2`: 180° (upside down)
  - `U8G2_R3`: 270° clockwise

Custom Characters:

```
// Draw custom bitmap
static const unsigned char myBitmap[] = {0x00, 0x3c, 0x42, 0x42, 0x3c, 0x00};
u8g2.drawBitmap(x, y, 1, 6, myBitmap);
```

Text Measurements:

```
int width = u8g2.getStrWidth("Hello");  // Get text width
int height = u8g2.getAscent();          // Get font height
```

The OLED display is now ready to show your sensor data, system status, or any custom graphics you design for your ML projects!

Summary

The XIAOML Kit with ESP32S3 Sense represents a powerful, yet accessible entry point into the world of TinyML and embedded machine learning. Through this setup process, we have systematically tested every component of the XIAOML Kit, confirming that all sensors and peripherals are functioning correctly. The ESP32S3's dual-core processor and 8MB of PSRAM provide sufficient computational power for real-time ML inference, while the OV2640 camera, digital microphone, LSM6DS3TR-C IMU, and 0.42" OLED display create a complete multimodal sensing platform. WiFi connectivity opens possibilities for edge-to-cloud ML workflows, and our Arduino IDE development environment is now properly configured with all necessary libraries.

Beyond mere functionality tests, we've gained practical insights into coordinate systems, data formats, and operational characteristics of each sensor—knowledge that will prove invaluable when designing ML data collection and preprocessing pipelines for the upcoming projects.

This setup process demonstrates key principles that extend far beyond this specific kit. Working with the ESP32S3's memory limitations and processing capabilities provides an authentic experience with the resource constraints inherent in edge AI—the same considerations that apply when deploying models on smartphones, IoT devices, or autonomous systems. Having multiple modalities (vision, audio, motion) on a single platform enables exploration of multimodal ML approaches, which are increasingly important in real-world AI applications.

Most importantly, from raw sensor data to model inference to user feedback via the OLED display, the kit provides a complete ML deployment cycle in miniature, mirroring the challenges faced in production AI systems.

With this foundation in place, you're now equipped to tackle the core TinyML applications in the following chapters:

- **Vision Projects**: Leveraging the camera for image classification and object detection
- **Audio Projects**: Processing audio streams for keyword spotting and voice recognition
- **Motion Projects**: Using IMU data for activity recognition and anomaly detection

Each application will build upon the hardware understanding and software infrastructure we've established, demonstrating how artificial intelligence can be deployed not just in data centers, but in resource-constrained devices that directly interact with the physical world.

The principles encountered with this kit—real-time processing, sensor fusion, and edge inference—are the same ones driving the future of AI deployment in autonomous vehicles, smart cities, medical devices, and industrial automation. By completing this setup successfully, you're now prepared to explore this exciting frontier of embedded machine learning.

Resources

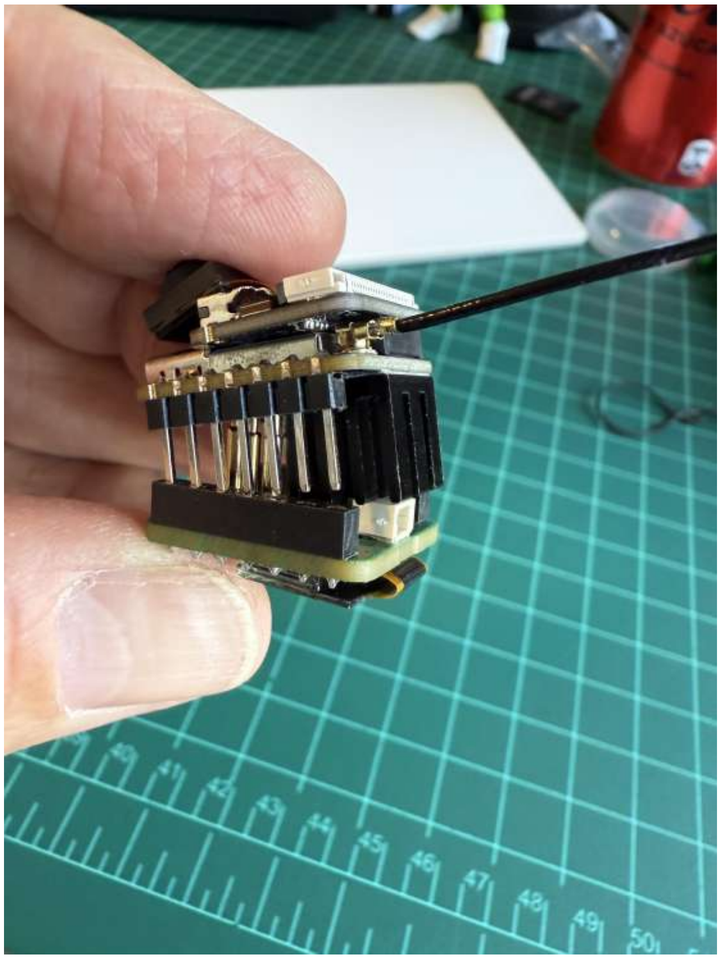
- XIAOML Kit Code
- XIAO ESP32S3 Sense manual & example code
- Usage of Seeed Studio XIAO ESP32S3 microphone
- File System and XIAO ESP32S3 Sense
- Camera Usage in Seeed Studio XIAO ESP32S3 Sense

Appendix

Heat Sink Considerations

If you need to use the XIAO ESP32S3 Sense for camera applications WITHOUT the Expansion Board, you may install the heat sink.

Note that having the heat sink installed, it is not possible to connect the XIAO ESP32S3 Sense with the Expansion Board.



Installing the Heat Sink

To ensure optimal cooling for your XIAO ESP32S3 Sense, you should install the provided heat sink during camera applications. Its design is specifically tailored to address cooling needs, particularly during intensive operations such as camera usage.

> Two heat sinks are included in the kit, but you can use only one to guarantee access to the Battery pins.

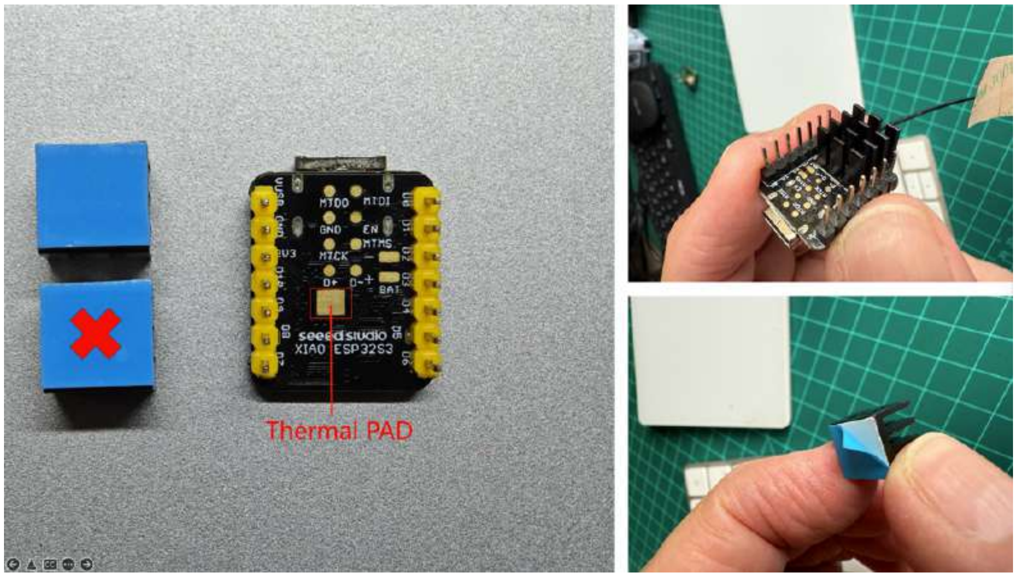
Installation:

- Ensure your device is powered off and unplugged from any power source before you start.
- Prioritize covering the Thermal PAD with the heat sink, as it is directly above the ESP32S3 chip, the primary source of heat. Proper alignment ensures optimal heat dissipation, and **it is essential to keep the BAT pins as unobstructed as possible**.

Now, let's begin the installation process:

Step 1. Prepare the Heat Sink: Start by removing the protective cover from the heat sink to expose the thermal adhesive. This will prepare the heat sink for a secure attachment to the ESP32S3 chip.

Step 2. Assemble the Heat Sink:



> After installation, ensure everything is properly secured with no risk of short circuits. Verify that the heat sink is properly aligned and securely attached.

If one heat synk is not enough, a second one can be installed, sharing both the thermal pad, but in this situation, be aware that all pins became unavailable.

Attention

Remove carefully the heat sinks before using the IMU expansion board again

Image Classification

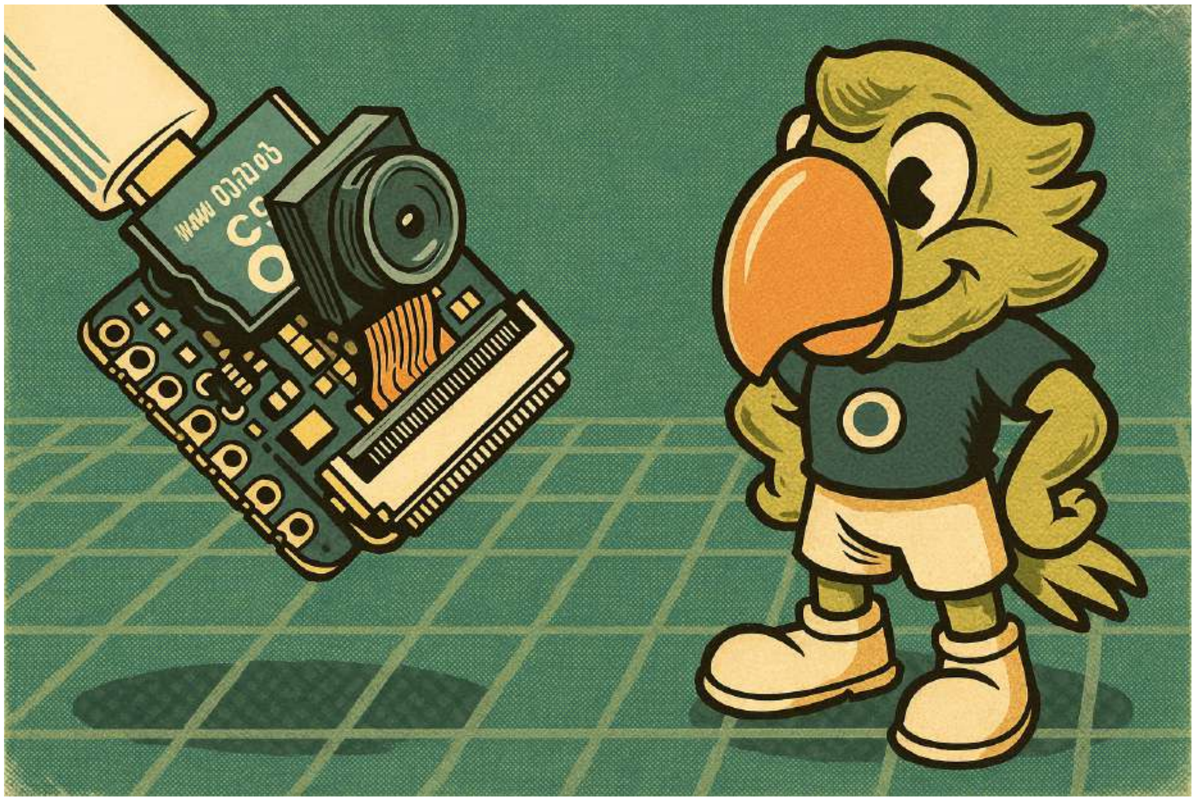


Figure 2: *DALL · E prompt - 1950s style cartoon illustration based on a real image by Marcelo Rovai*

Overview

We are increasingly facing an artificial intelligence (AI) revolution, where, as Gartner states, **Edge AI and Computer Vision** have a very high impact potential, and **it is for now**!

When we look into Machine Learning (ML) applied to vision, the first concept that greets us is **Image Classification**, a kind of ML's *Hello World* that is both simple and profound!

The Seeed Studio XIAOML Kit provides a comprehensive hardware solution centered around theXIAO ESP32-S3 Sense, featuring an integrated **OV3660** camera and SD card support. Those features make the XIAO ESP32S3 Sense an excellent starting point for exploring TinyML vision AI.

In this Lab, we will explore Image Classification using the non-code tool **SenseCraft AI** and explore a more detailed development with **Edge Impulse Studio** and **Arduino IDE**.

Learning Objectives

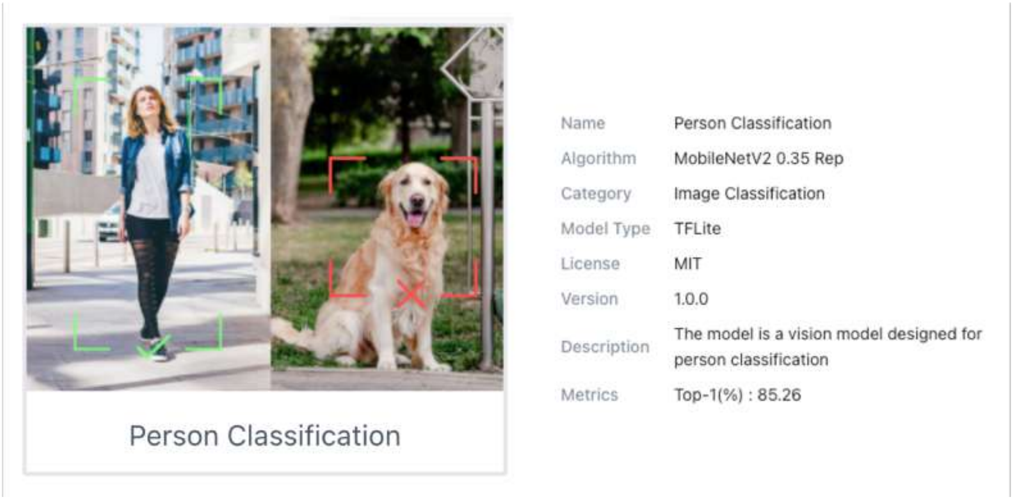
- **Deploy Pre-trained Models** using SenseCraft AI Studio for immediate computer vision applications
- **Collect and Manage Image Datasets** for custom classification tasks with proper data organization
- **Train Custom Image Classification Models** using transfer learning with MobileNet V2 architecture
- **Optimize Models for Edge Deployment** through quantization and memory-efficient preprocessing
- **Implement Post-processing Pipelines,** including GPIO control and real-time inference integration
- **Compare Development Approaches** between no-code and advanced ML platforms for embedded applications

Image Classification

Image classification is a fundamental task in computer vision that involves categorizing entire images into one of several predefined classes. This process entails analyzing the visual content of an image and assigning it a label from a fixed set of categories based on the dominant object or scene it depicts.

Image classification is crucial in various applications, ranging from organizing and searching through large databases of images in digital libraries and social media platforms to enabling autonomous systems to comprehend their surroundings. Common architectures that have significantly advanced the field of image classification include Convolutional Neural Networks (CNNs), such as AlexNet, VGGNet, and ResNet. These models have demonstrated remarkable accuracy on challenging datasets, such as ImageNet, by learning hierarchical representations of visual data.

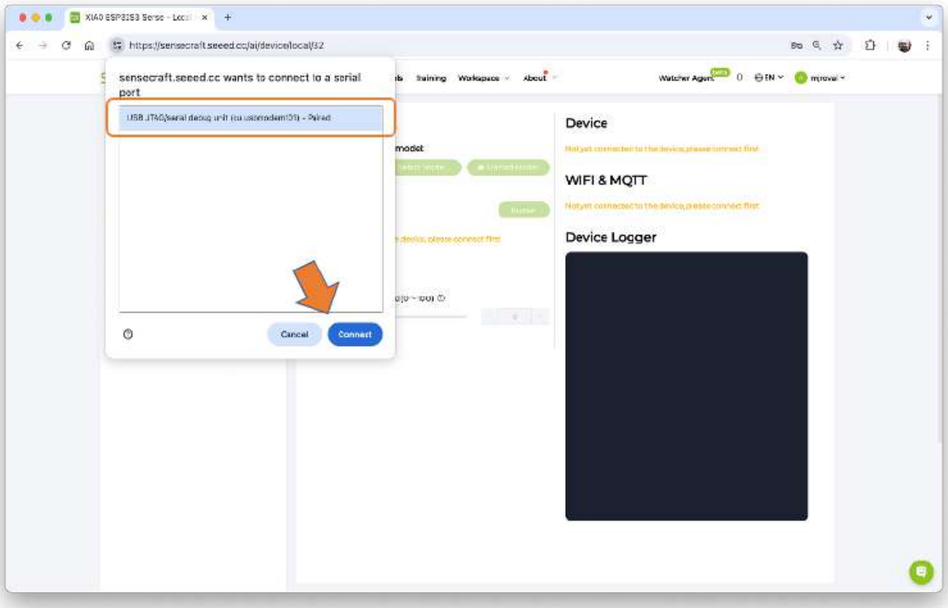
As the cornerstone of many computer vision systems, image classification drives innovation, laying the groundwork for more complex tasks like object detection and image segmentation, and facilitating a deeper understanding of visual data across various industries. So, let's start exploring the Person Classification model ("Person - No Person"), a ready-to-use computer vision application on the **SenseCraft AI**.



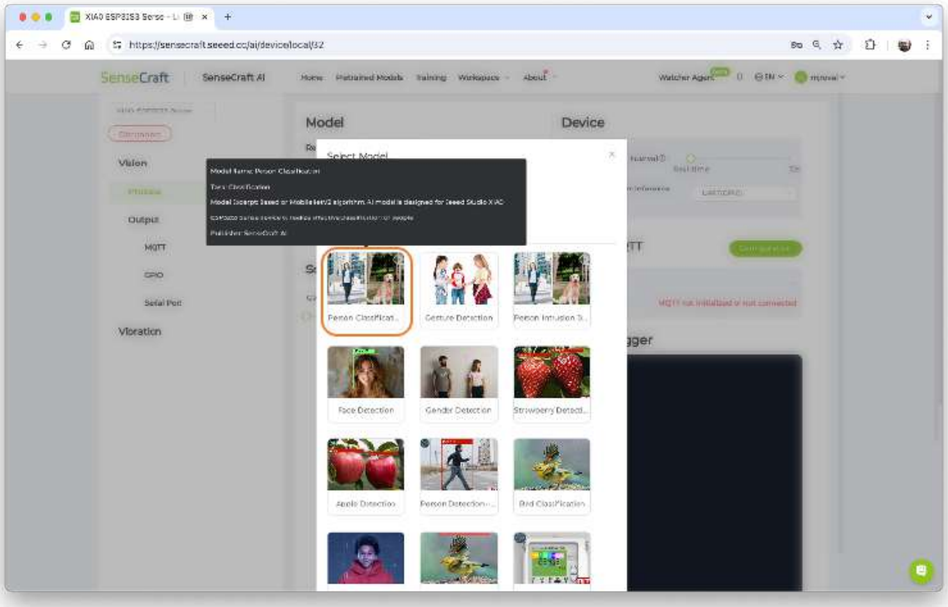
Name	Person Classification
Algorithm	MobileNetV2 0.35 Rep
Category	Image Classification
Model Type	TFLite
License	MIT
Version	1.0.0
Description	The model is a vision model designed for person classification
Metrics	Top-1(%) : 85.26

Image Classification on the SenseCraft AI Workspace

Start by connecting the XIAOML Kit (or just the XIAO ESP32S3 Sense, disconnected from the Expansion Board) to the computer via USB-C, and then open the SenseCraft AI Workspace to connect it.


sensecraft.seeed.cc wants to connect to a serial port
USB JTAG/serial debug unit (cu.usbmodem101) - Paired
Cancel
Connect
Device
WiFi & MQTT
Device Logger

Once connected, select the option `[Select Model...]` and enter in the search window: "*Person Classification*". From the options available, select the one trained over the MobileNet V2 (passing the mouse over the models will open a pop-up window with its main characteristics).

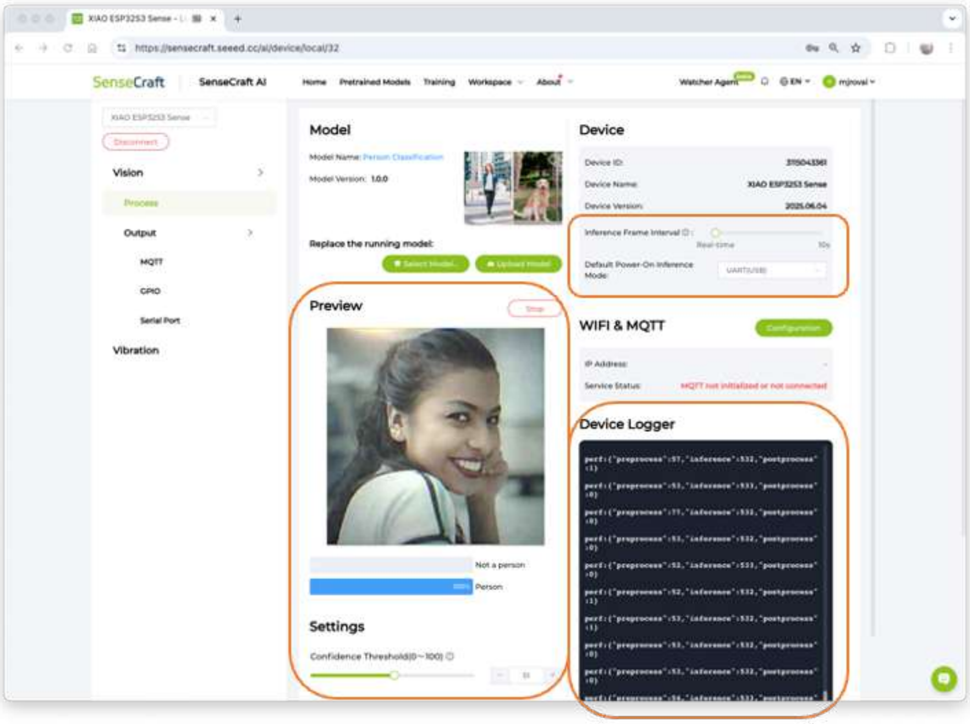


Click on the chosen model and confirm the deployment. A new firmware for the model should start uploading to our device.

> Note that the percentage of models downloaded and firmware uploaded will be displayed. If not, try disconnecting the device, then reconnect it and press the boot button.

After the model is uploaded successfully, we can view the live feed from the XIAO camera and the classification result (`Person` or `Not a Person`) in the **Preview** area, along with the inference details displayed in the **Device Logger**.

> Note that we can also select our **Inference Frame Interval**, from "Real-Time" (Default) to 10 seconds, and the **Mode** (UART, I2C, etc) as the data is shared by the device (the default is UART via USB).



Model Name: Person Classification
Model Version: 1.0.0

Device ID: 3115043361
Device Name: XIAO ESP32S3 Sense
Device Version: 2025.06.04
Inference Frame Interval: Real-time … 10s
Default Power-On Inference Mode: UART(USB)

IP Address: -
Service Status: MQTT not initialized or not connected

Not a person
Person 100%

Confidence Threshold(0~100)

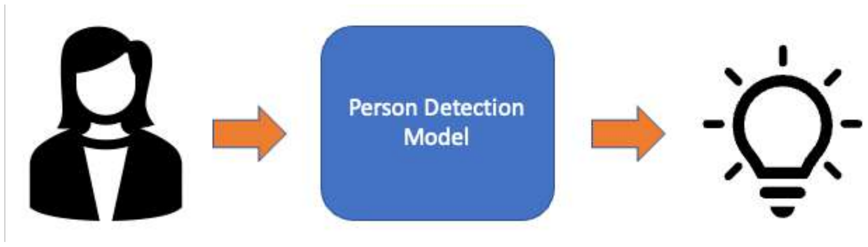
```
perf:{"preprocess":57,"inference":532,"postprocess":1}
perf:{"preprocess":53,"inference":533,"postprocess":0}
perf:{"preprocess":77,"inference":532,"postprocess":0}
perf:{"preprocess":53,"inference":532,"postprocess":0}
perf:{"preprocess":52,"inference":533,"postprocess":0}
perf:{"preprocess":52,"inference":532,"postprocess":1}
perf:{"preprocess":53,"inference":532,"postprocess":1}
perf:{"preprocess":53,"inference":532,"postprocess":0}
perf:{"preprocess":53,"inference":532,"postprocess":0}
```

At the Device Logger, we can see that the latency of the model is from 52 to 78 ms for pre-processing and around 532ms for inference, which will give us a total time of a little less than 600ms, or about **1.7 Frames per second (FPS)**.

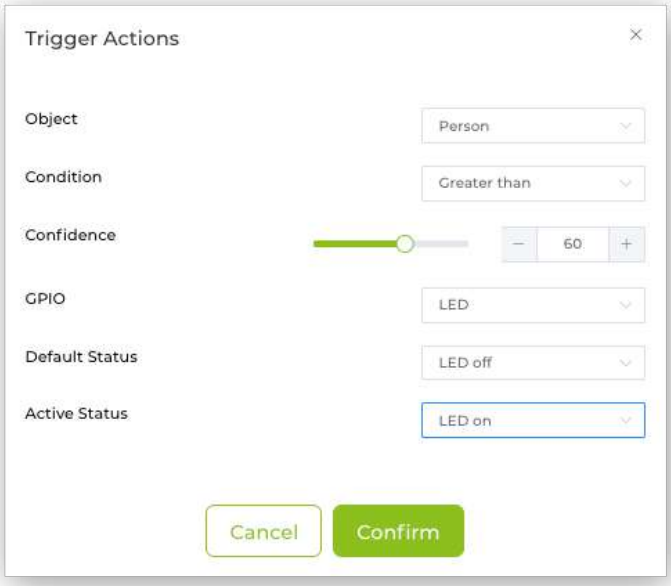
> To run the Mobilenet V2 0.35, the XIAO had a peak current of 160mA at 5.23V, resulting in a **power consumption of 830mW**.

Post-Processing

An essential step in an Image Classification project pipeline is to define what we want to do with the inference result. So, imagine that we will use the XIAO to automatically turn on the room lights if a person is detected.

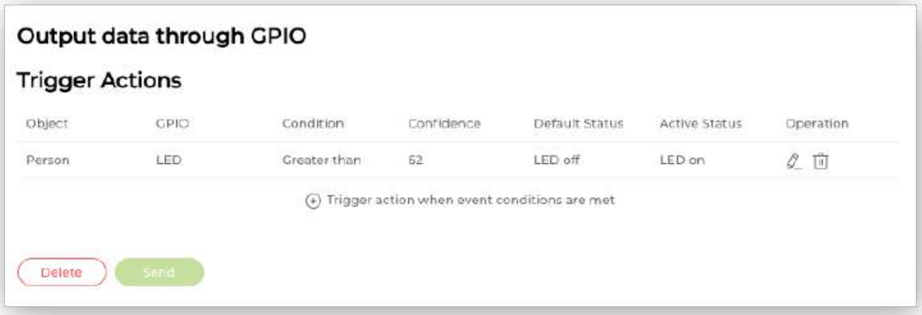


With the SebseCraft AI, we can do it on the `Output -> GPIO` section. Click on the Add icon to trigger the action when event conditions are met. A pop-up window will open, where you can define the action to be taken. For example, if a person is detected with a confidence of more than 60% the internal `LED` should be ON. In a real scenario, a GPIO, for example, `D0`, `D1`, `D2`, `D11`, or `D12`, would be used to trigger a relay to turn on a light.


Trigger Actions

Object	Person
Condition	Greater than
Confidence	60
GPIO	LED
Default Status	LED off
Active Status	LED on

Once confirmed, the created **Trigger Action** will be shown. Press `Send` to upload the command to the XIAO.

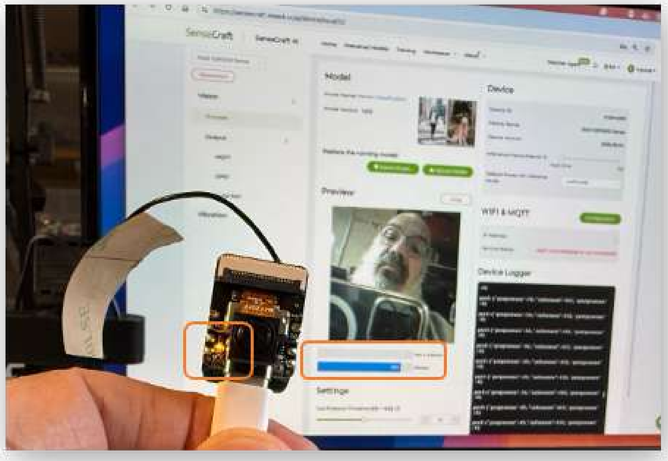


Output data through GPIO

Trigger Actions

Object	GPIO	Condition	Confidence	Default Status	Active Status	Operation
Person	LED	Greater than	62	LED off	LED on	

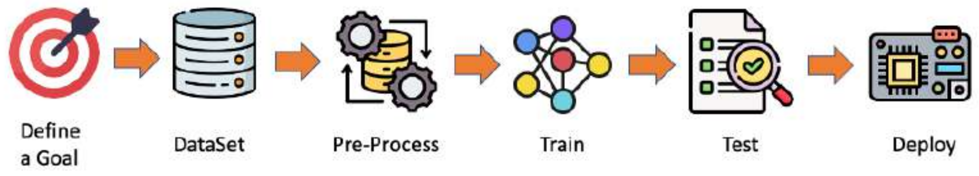
Now, pointing the XIAO at a person will make the internal LED go ON.



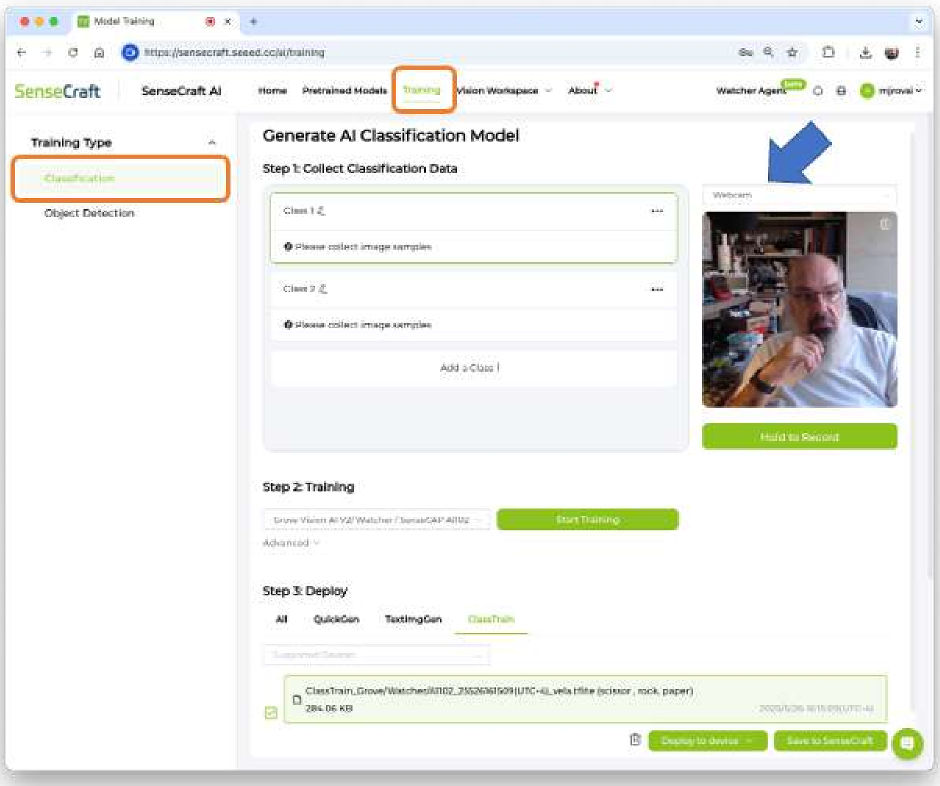
We will explore more trigger actions and post-processing techniques further in this lab.

An Image Classification Project

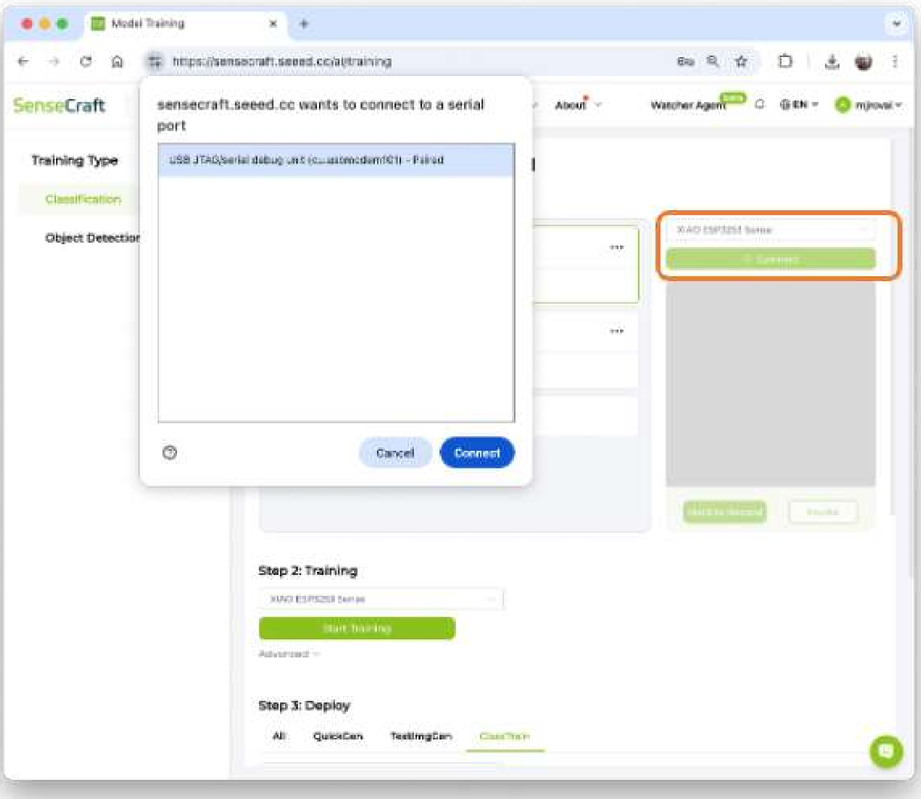
Let's create a simple Image Classification project using SenseCraft AI Studio. Below, we can see a typical machine learning pipeline that will be used in our project.



On SenseCraft AI Studio: Let's open the tab Training:



The default is to train a `Classification` model with a WebCam if it is available. Let's select the `XIAOESP32S3 Sense` instead. Pressing the green button `[Connect]` will cause a Pop-Up window to appear. Select the corresponding Port and press the blue button `[Connect]`.

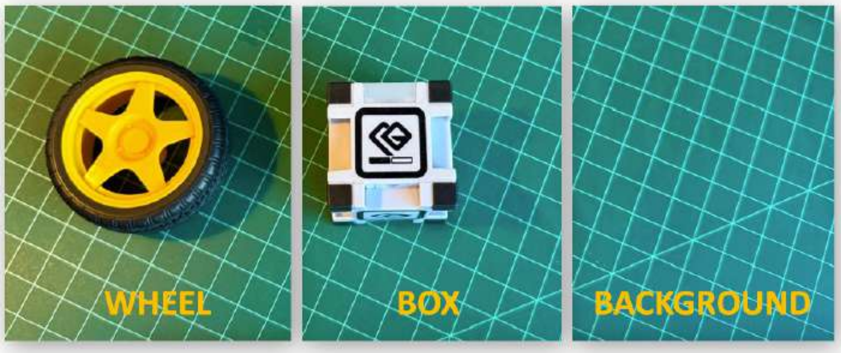


The image streamed from the Grove Vision AI V2 will be displayed.

The Goal

The first step, as we can see in the ML pipeline, is to define a goal. Let's imagine that we have an industrial installation that should automatically sort wheels and boxes.

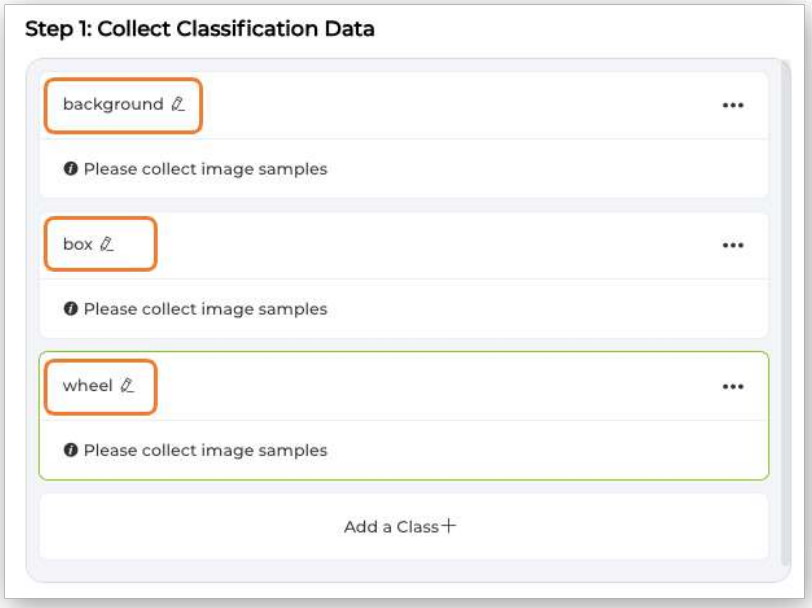
So, let's simulate it, classifying, for example, a toy `box` and a toy `wheel`. We should also include a 3rd class of images, `background`, where there are no objects in the scene.



Data Collection

Let's create the classes, following, for example, an alphabetical order:

- Class1: background
- Class 2: box
- Class 3: wheel



Step 1: Collect Classification Data

background

Please collect image samples

box

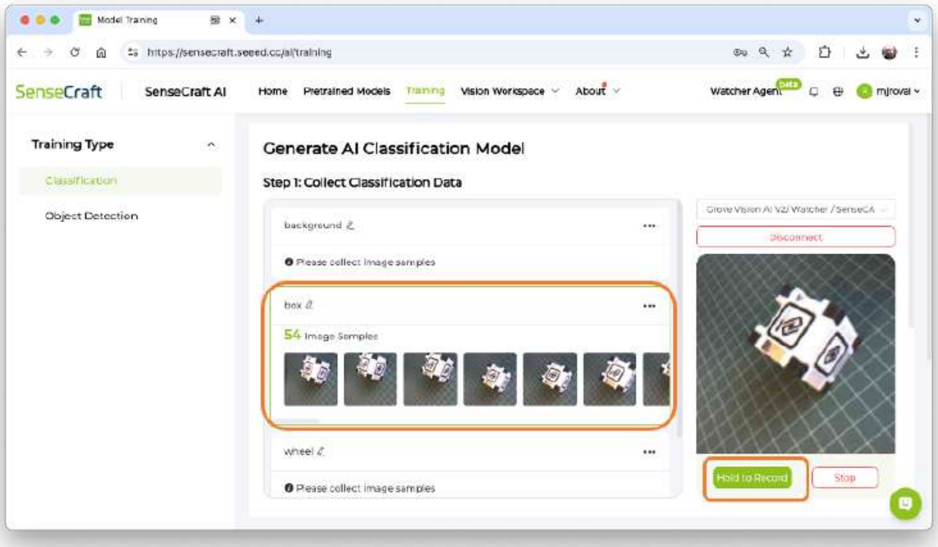
Please collect image samples

wheel

Please collect image samples

Add a Class +

Select one of the classes and keep pressing the green button (`Hold to Record`) under the preview area. The collected images (and their counting) will appear on the Image Samples Screen. Carefully and slowly, move the camera to capture different angles of the object. To modify the position or interfere with the image, release the green button, rearrange the object, and then hold it again to resume the capture.



Generate AI Classification Model

Step 1: Collect Classification Data

background

Please collect image samples

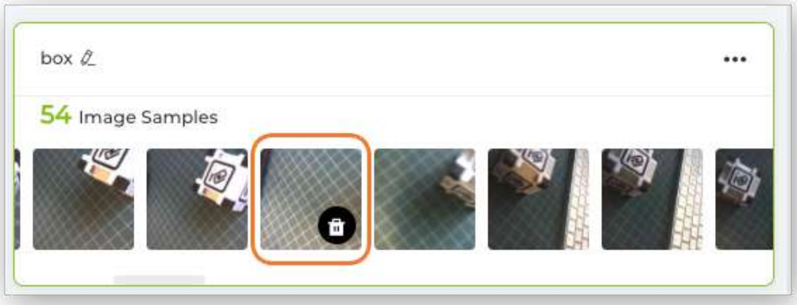
box

54 Image Samples

wheel

Please collect image samples

After collecting the images, review them and delete any incorrect ones.

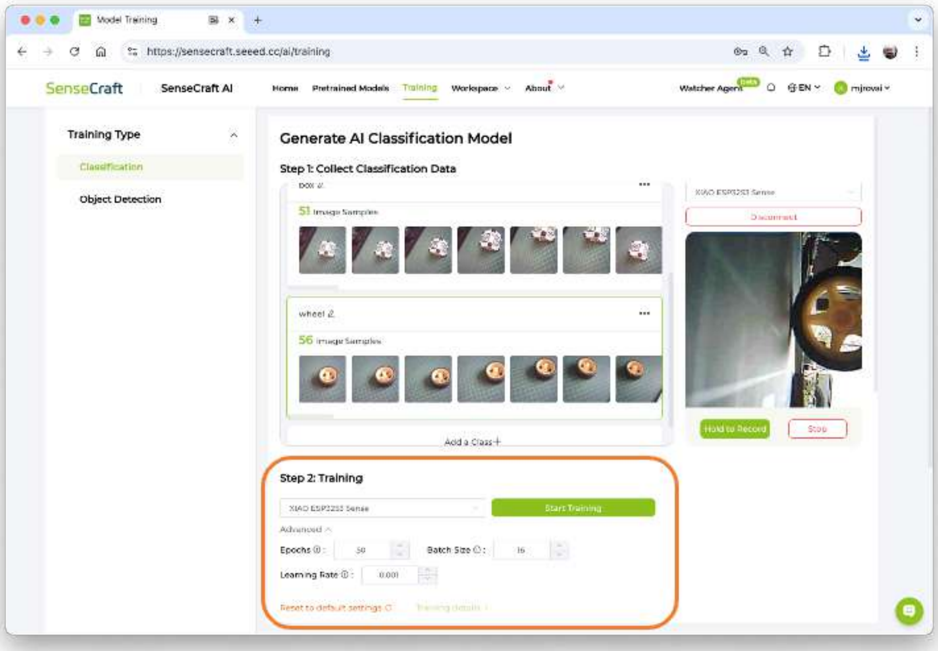


Collect around **50 images** from each class and go to Training Step.

> Note that it is possible to download the collected images to be used in another application, for example, with the Edge Impulse Studio.

Training

Confirm if the correct device is selected (`XIAO ESP32S3 Sense`) and press `[Start Training]`



Step 2: Training

XIAO ESP32S3 Sense

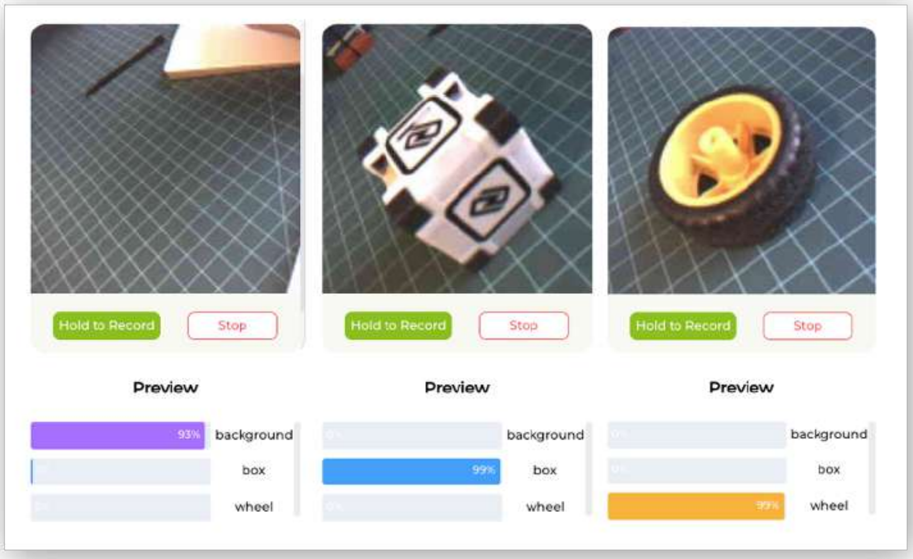
Epochs: 50   Batch Size: 16

Learning Rate: 0.001

Test

After training, the inference result can be previewed.

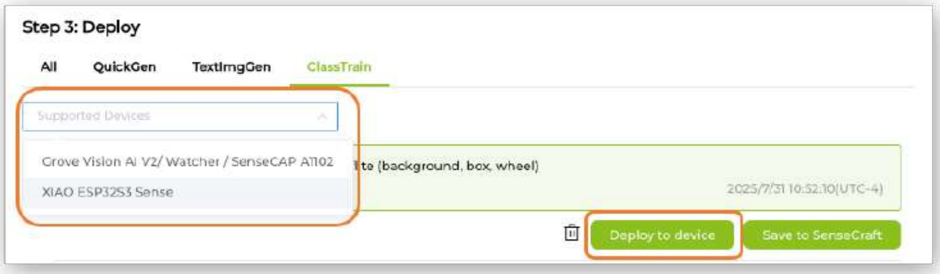
> Note that the model is not running on the device. We are, in fact, only capturing the images with the device and performing a **live preview** using the training model, which is running in the Studio.



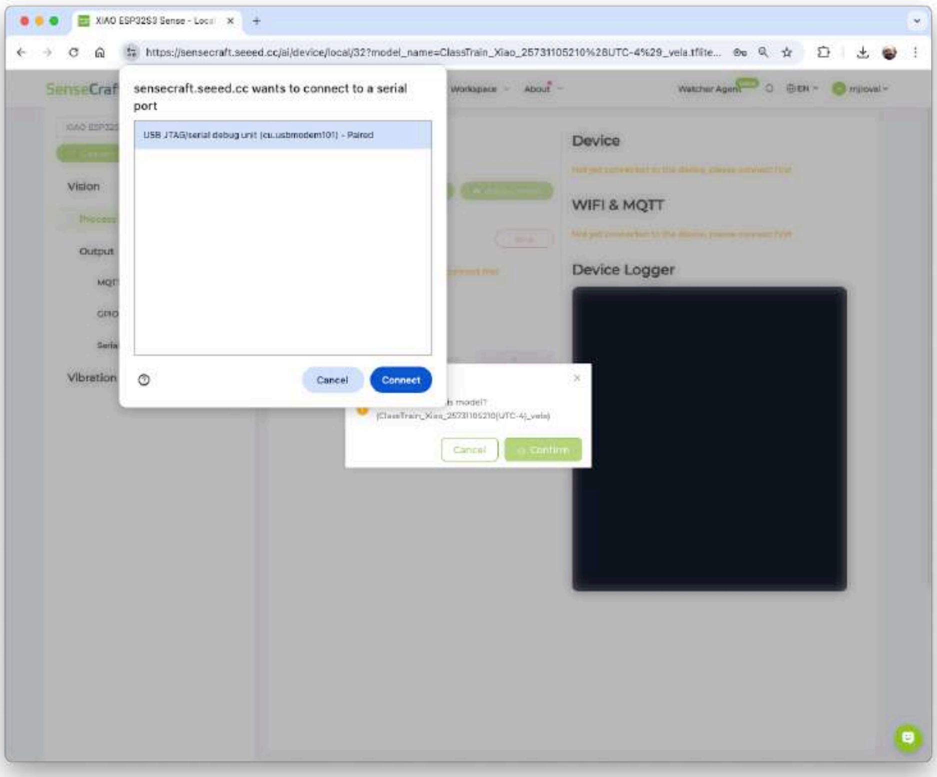
Now is the time to really deploy the model in the device.

Deployment

Select the trained model and `XIAO ESP32S3 Sense` at the `Supported Devices` window. And press `[Deploy to device]`.

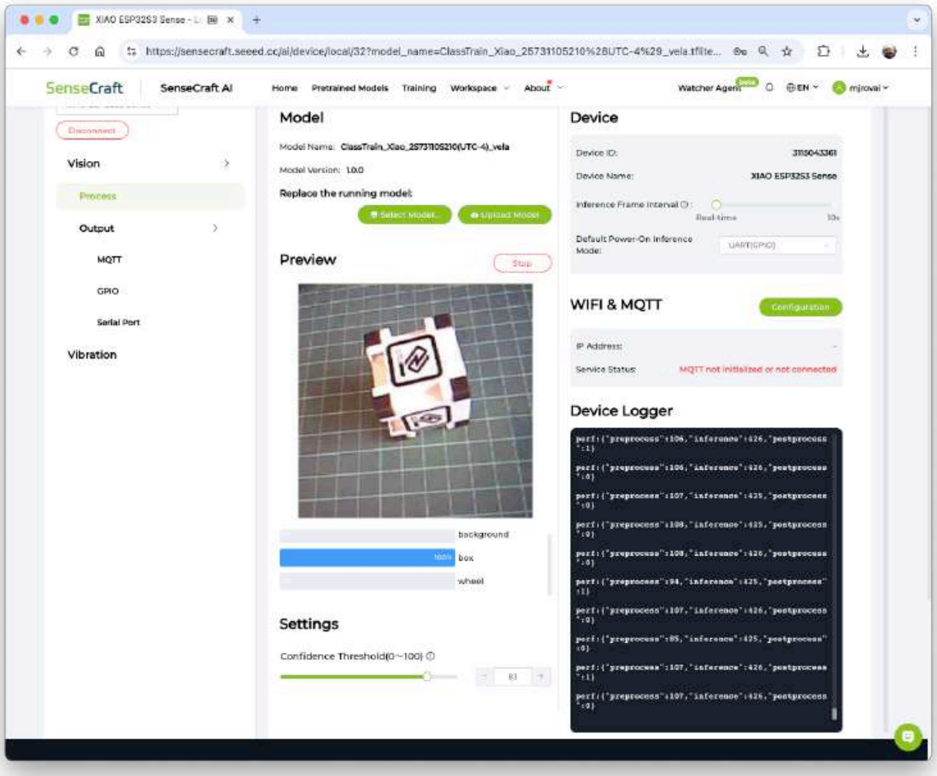


The SeneCrafit AI will redirect us to the **Vision Workplace** tab. `Confirm` the deployment, select the Port, and `Connect` it.


sensecraft.seeed.cc wants to connect to a serial port
USB JTAG/serial debug unit (cu.usbmodem101) - Paired
Cancel
Connect
Device
WIFI & MQTT
Device Logger
Vision
Output
Vibration
Confirm

The model will be flashed into the device. After an automatic reset, the model will start running on the device. On the Device Logger, we can see that the inference has a **latency of approximately 426 ms**, plus a **pre-processing of around 110ms**, corresponding to a **frame rate of 1.8 frames per second (FPS)**.

Also, note that in **Settings**, it is possible to adjust the model's confidence.



Model

Model Name: ClassTrain_Xiao_25731105210(UTC-4)_vela

Model Version: 1.0.0

Replace the running model:

Preview

Settings

Confidence Threshold(0~100): 83

Device

Device ID: 3115043361

Device Name: XIAO ESP32S3 Sense

Inference Frame Interval: Real-time … 10s

Default Power-On Inference Mode: UART(GPIO)

WIFI & MQTT

IP Address: -

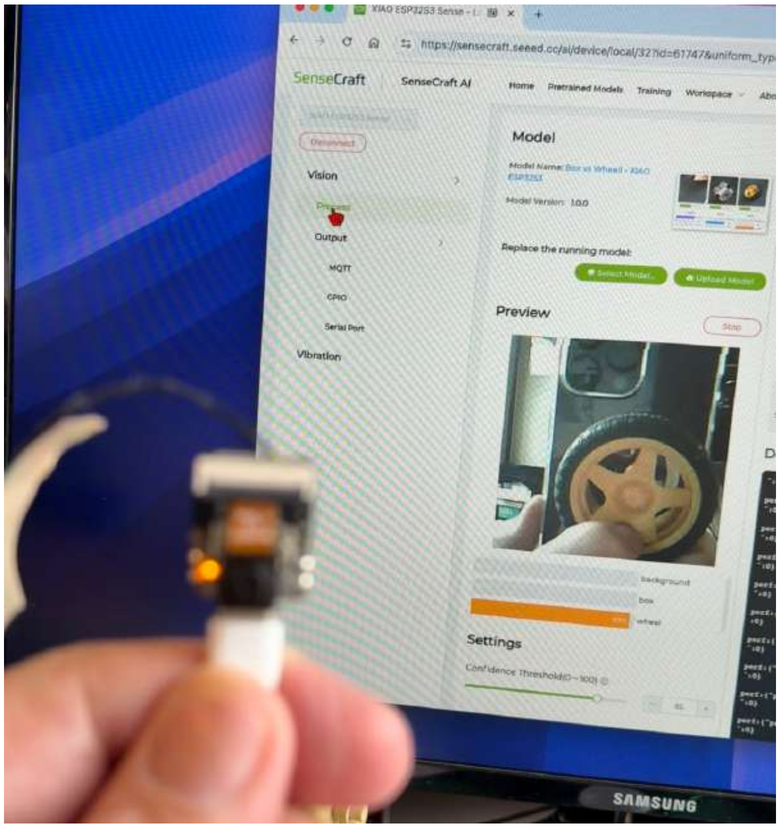
Service Status: MQTT not initialized or not connected

Device Logger

```
perf:{"preprocess":106,"inference":426,"postprocess
":1}
perf:{"preprocess":106,"inference":426,"postprocess
":0}
perf:{"preprocess":107,"inference":425,"postprocess
":0}
perf:{"preprocess":108,"inference":425,"postprocess
":0}
perf:{"preprocess":108,"inference":426,"postprocess
":0}
perf:{"preprocess":94,"inference":425,"postprocess"
:1}
perf:{"preprocess":107,"inference":426,"postprocess
":0}
perf:{"preprocess":85,"inference":425,"postprocess"
:0}
perf:{"preprocess":107,"inference":426,"postprocess
":1}
perf:{"preprocess":107,"inference":426,"postprocess
":0}
```

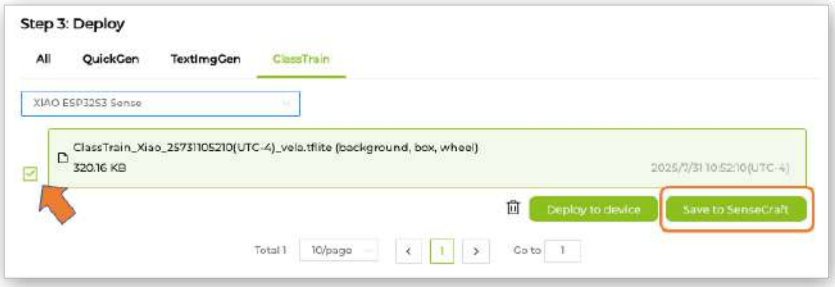
To run the Image Classification Model, the XIAO ESP32S3 had a peak current of 14mA at 5.23V, resulting in a **power consumption of 730mW**.

As before, in the **Output –> GPIO**, we can turn the GPIOs or the Internal LED ON based on the detected class. For example, the LED will be turned ON when the wheel is detected.

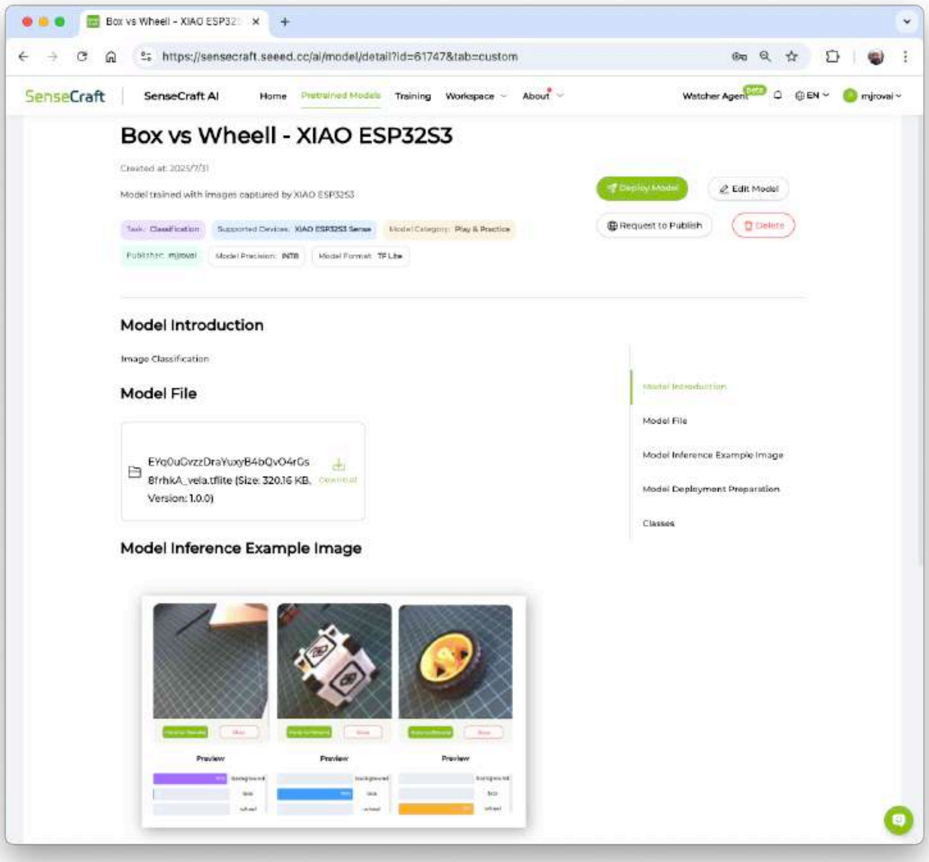


Saving the Model

It is possible to save the model in the SenseCraft AI Studio. The Studio will retain all our models for later deployment. For that, return to the `Training` tab and select the button [`Save to SenseCraft`]:



Follow the instructions to enter the model's name, description, image, and other details.


https://sensecraft.seeed.cc/ai/model/detail?id=61747&tab=custom

Box vs Wheell - XIAO ESP32S3

Created at: 2025/7/31

Model trained with images captured by XIAO ESP32S3

Task: Classification
Supported Devices: XIAO ESP32S3 Sense
Model Category: Play & Practice
Publisher: mjrovai
Model Precision: INT8
Model Format: TF Lite

Model Introduction

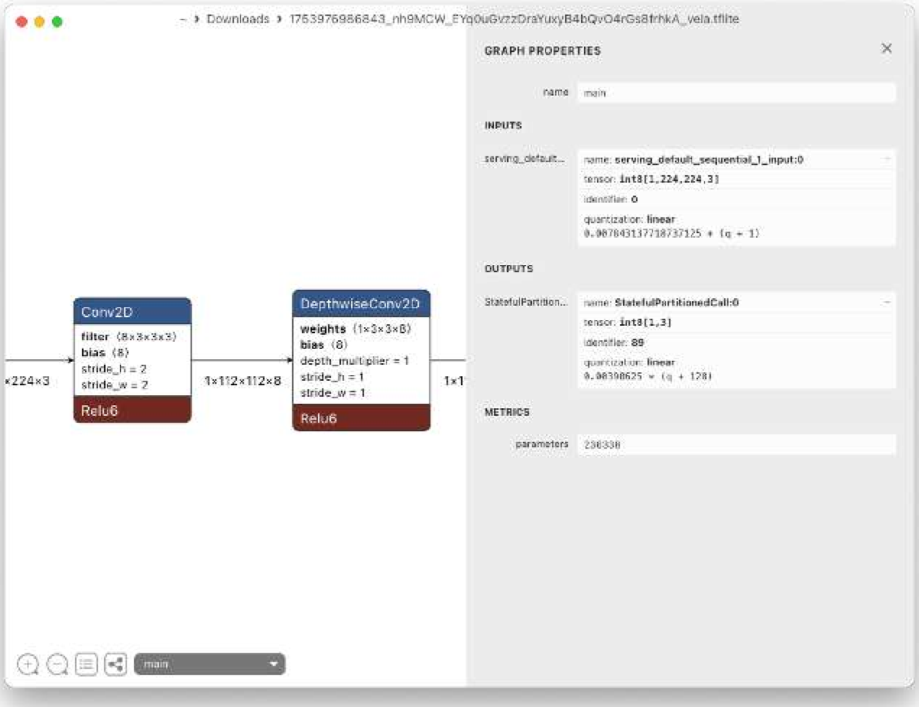
Image Classification

Model File

EYq0uOvzzDraYuxyB4bQvO4rGs8frhkA_vela.tflite (Size: 320.16 KB, Version: 1.0.0)

Model Inference Example Image

Note that the trained model (an Int8 MobileNet V2 with a size of 320KB) can be downloaded for further use or even analysis, for example, using Netron. Note that the model uses images of size 224x224x3 as its Input Tensor. In the next step, we will use different hyperparameters on the Edge Impulse Studio.



GRAPH PROPERTIES

name: main

INPUTS

serving_default...
name: serving_default_sequential_1_input:0
tensor: int8[1,224,224,3]
identifier: 0
quantization: linear
0.007843137718737125 * (q + 1)

OUTPUTS

StatefulPartition...
name: StatefulPartitionedCall:0
tensor: int8[1,3]
identifier: 89
quantization: linear
0.00390625 * (q + 128)

METRICS

parameters: 236338

Also, the model can be deployed again to the device at any time. Automatically, the **Workspace** will be open on the SenseCraft AI.

Image Classification Project from a Dataset

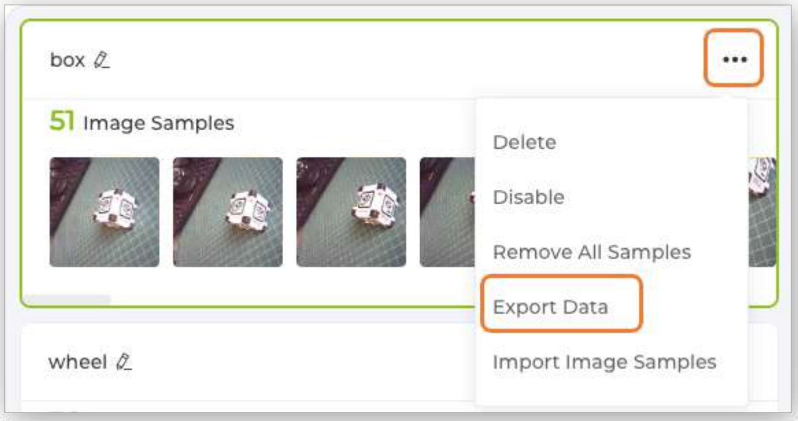
The primary objective of our project is to train a model and perform inference on the XIAO ESP32S3 Sense. For training, we should find some data **(in fact, tons of data!)**.

But as we alheady know, first of all, we need a goal! What do we want to classify?

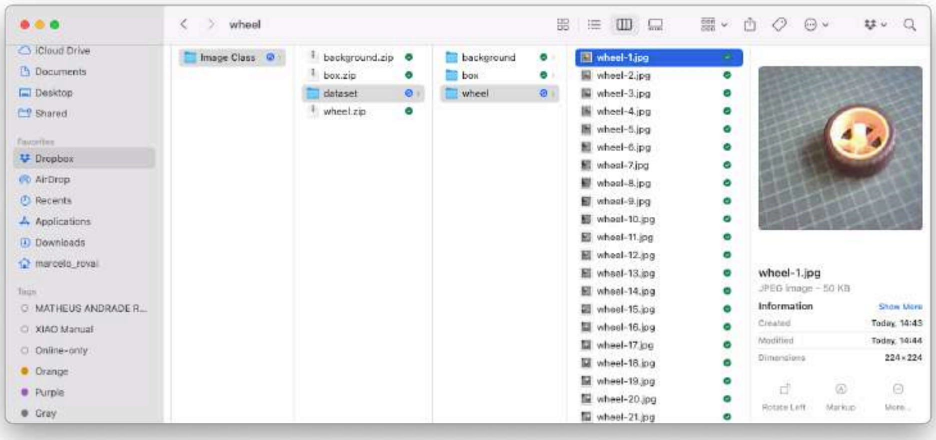
With TinyML, a set of techniques associated with machine learning inference on embedded devices, we should limit the classification to three or four categories due to limitations (mainly memory). We can, for example, train the images captured for the Box versus Wheel, which can be downloaded from the SenseCraft AI Studio.

> Alternatively, we can use a completely new dataset, such as one that differentiates apples from bananas and potatoes, or other categories. If possible, try finding a specific dataset that includes images from those categories. Kaggle fruit-and-vegetable-image-recognition is a good start.

Let's download the dataset captured in the previous section. Open the menu (3 dots) on each of the captured classes and select `Export Data`.



The dataset will be downloaded to the computer as a .ZIP file, with one file for each class. Save them in your working folder and unzip them. You should have three folders, one for each class.



Optionally, you can add some fresh images, using, for example, the code discussed in the setup lab.

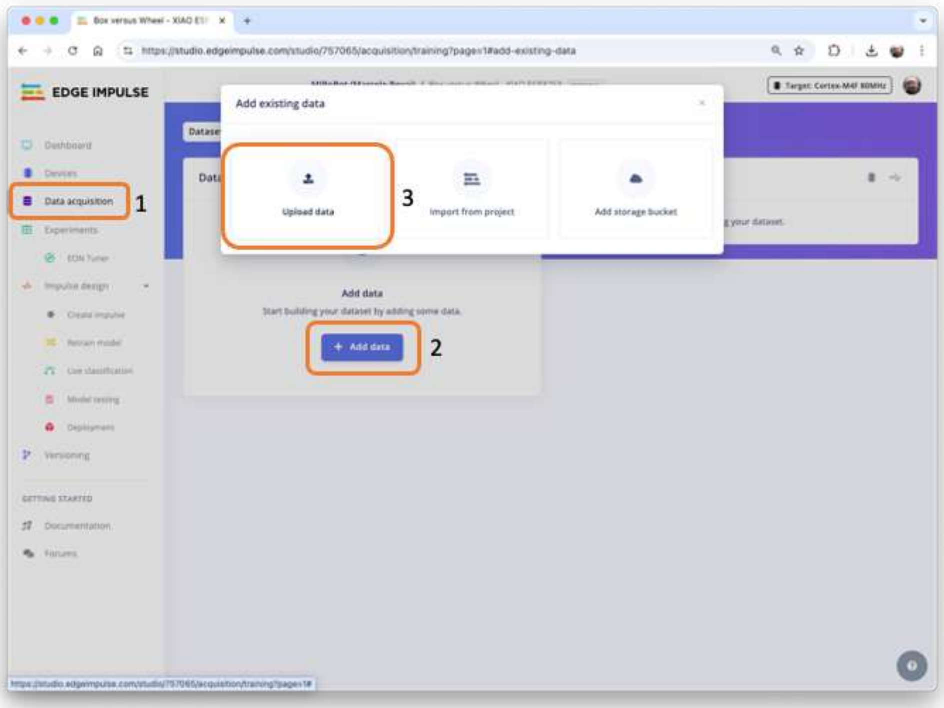
Training the model with Edge Impulse Studio

We will use the Edge Impulse Studio to train our model. Edge Impulse is a leading development platform for machine learning on edge devices.

Enter your account credentials (or create a free account) at Edge Impulse. Next, create a new project:

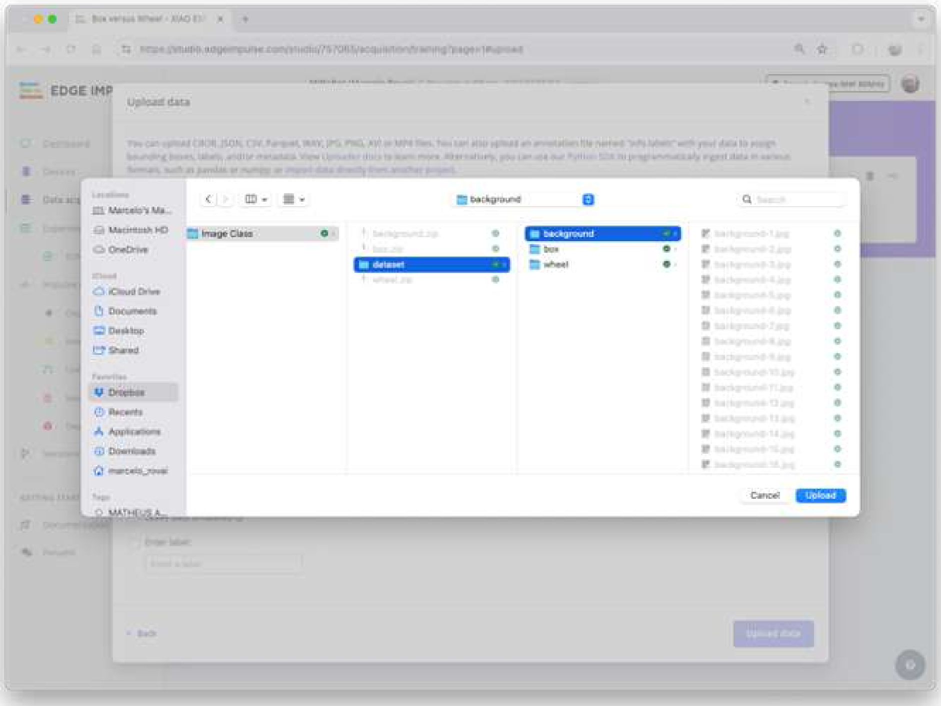
Data Acquisition

Next, go to the **Data acquisition** section and there, select `+ Add data`. A pop-up window will appear. Select `UPLOAD DATA`.

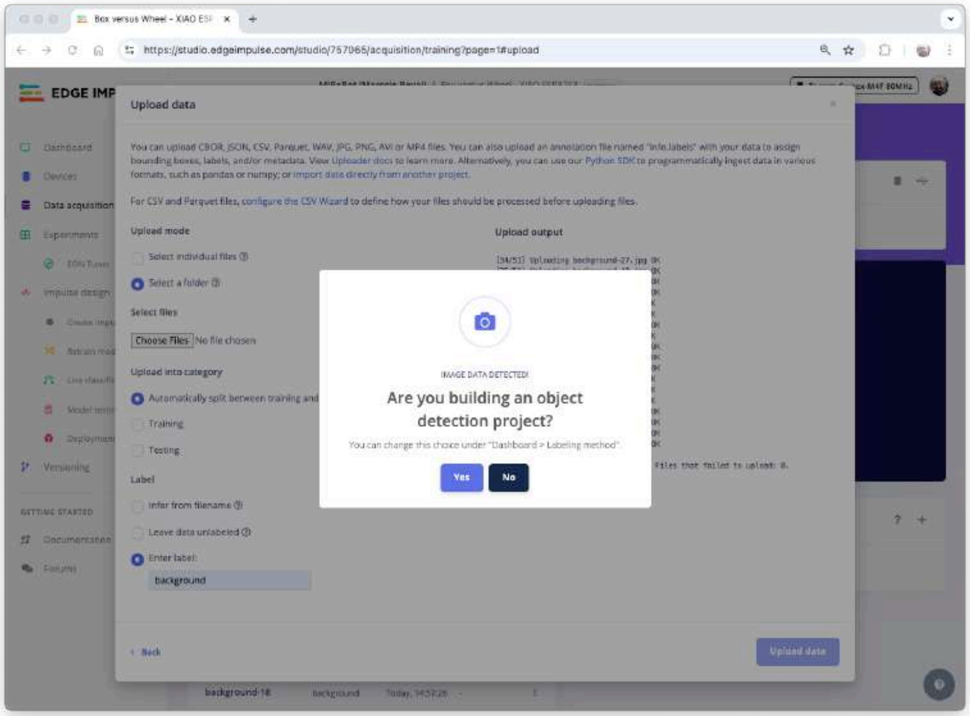


After selection, a new Pop-Up window will appear, asking to update the data.

- In Upload mode: `select a folder` and press `[Choose Files]`.
- Go to the folder that contains one of the classes and press `[Upload]`



- You will return automatically to the Upload data window.
- Select `Automatically split between training and testing`
- And enter the label of the images that are in the folder.
- Select `[Upload data]`
- At this point, the files will start to be uploaded, and after that, another Pop-Up window will appear asking if you are building an object detection project. Select `[no]`


Upload data

You can upload CBOR, JSON, CSV, Parquet, WAV, JPG, PNG, AVI or MP4 files. You can also upload an annotation file named "info.labels" with your data to assign bounding boxes, labels, and/or metadata. View Uploader docs to learn more. Alternatively, you can use our Python SDK to programmatically ingest data in various formats, such as pandas or numpy; or import data directly from another project.

For CSV and Parquet files, configure the CSV Wizard to define how your files should be processed before uploading files.

Upload mode: Select individual files [✓ Select a folder]

Select files: Choose Files No file chosen

Upload into category: [✓ Automatically split between training and] Training Testing

Label: Infer from filename Leave data unlabeled [✓ Enter label:] background

IMAGE DATA DETECTED!

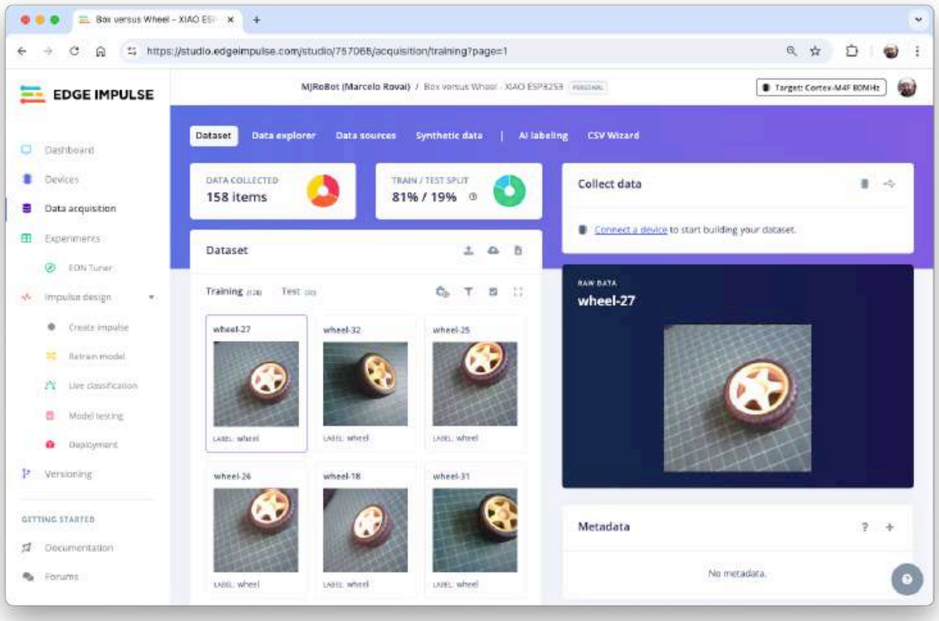
Are you building an object detection project?

You can change this choice under "Dashboard > Labeling method".

```
[34/51] Uploading background-27.jpg OK
Files that failed to upload: 0.
```

Repeat the procedure for all classes. **Do not forget to change the label's name**. If you forget and the images are uploaded, please note that they will be mixed in the Studio. Do not worry, you can manually move the data between classes further.

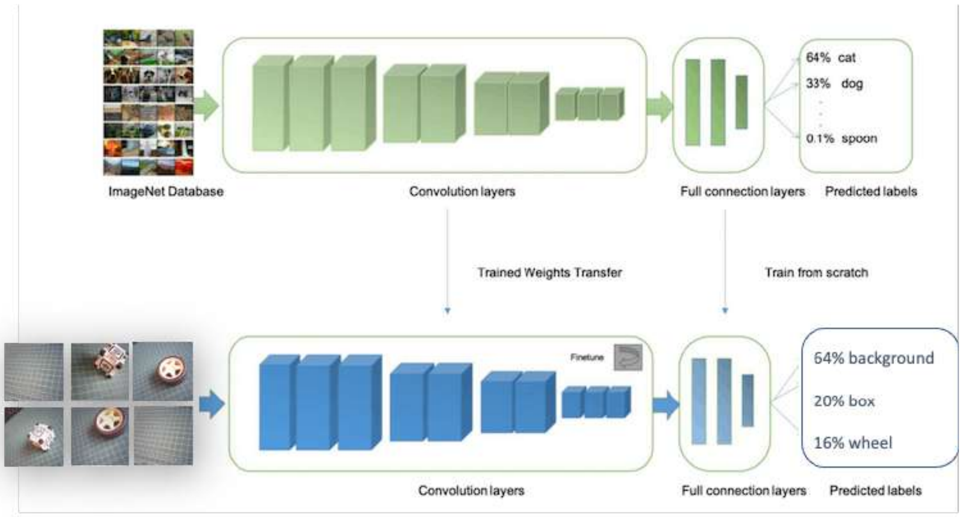
Close the Upload Data window and return to the **Data acquisition** page. We can see that all dataset was uploaded. Note that on the upper panel, we can see that we have 158 items, all of which are balanced. Also, 19% of the images were left for testing.



Impulse Design

> An impulse takes raw data (in this case, images), extracts features (resizes pictures), and then uses a learning block to classify new data.

Classifying images is the most common application of deep learning, but a substantial amount of data is required to accomplish this task. We have around 50 images for each category. Is this number enough? Not at all! We will need thousands of images to "teach" or "model" each class, allowing us to differentiate them. However, we can resolve this issue by retraining a previously trained model using thousands of images. We refer to this technique as **"Transfer Learning" (TL)**. With TL, we can fine-tune a pre-trained image classification model on our data, achieving good performance even with relatively small image datasets, as in our case.



With TL, we can fine-tune a pre-trained image classification model on our data, performing well even with relatively small image datasets (our case).

So, starting from the raw images, we will resize them (96 × 96) Pixels are fed to our Transfer Learning block. Let's create an Inpulse.

> At this point, we can also define our target device to monitor our "budget" (memory and latency). The XIAO ESP32S3 is not officially supported by Edge Impulse, so let's consider the Espressif ESP-EYE, which is similar but slower.

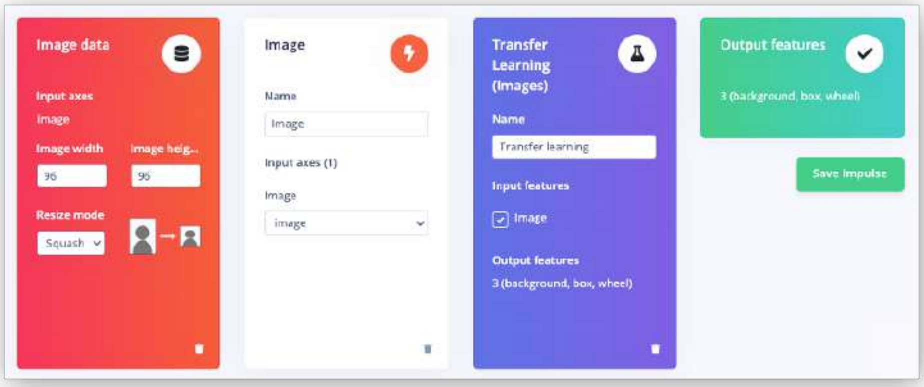


Image data
Input axes
image
Image width: 96
Image height: 96
Resize mode: Squash

Image
Name: Image
Input axes (1)
Image: image

Transfer Learning (Images)
Name: Transfer learning
Input features
✓ Image
Output features
3 (background, box, wheel)

Output features
3 (background, box, wheel)

Save Impulse

Save the Impulse, as shown above, and go to the **Image** section.

Pre-processing (Feature Generation)

Besides resizing the images, we can convert them to grayscale or retain their original RGB color depth. Let's select `[RGB]` in the `Image` section. Doing that, each data sample will have a dimension of 27,648 features (96x96x3). Pressing `[Save Parameters]` will open a new tab, `Generate Features`. Press the button `[Generate Features]`to generate the features.

Model Design, Training, and Test

In 2007, Google introduced MobileNetV1. In 2018, MobileNetV2: Inverted Residuals and Linear Bottlenecks, was launched, and, in 2019, the V3. The Mobilinet is a family of general-purpose computer vision neural networks explicitly designed for mobile devices to support classification, detection, and other applications. MobileNets are small, low-latency, low-power models parameterized to meet the resource constraints of various use cases.

Although the base MobileNet architecture is already compact and has low latency, a specific use case or application may often require the model to be even smaller and faster. MobileNets introduce a straightforward parameter, (alpha), called the width multiplier to construct these smaller, less computationally expensive models. The role of the width multiplier is to thin a network uniformly at each layer.

Edge Impulse Studio has available MobileNet V1 (96x96 images) and V2 (96x96 and 160x160 images), with several different values (from 0.05 to 1.0). For example, you will get the highest accuracy with V2, 160x160 images, and =1.0. Of course, there is a trade-off. The higher the accuracy, the more memory (around 1.3M RAM and 2.6M ROM) will be needed to run the model, implying more latency. The smaller footprint will be obtained at another extreme with MobileNet V1 and =0.10 (around 53.2K RAM and 101K ROM).

> We will use the **MobileNet V2 0.35** as our base model (but a model with a greater alpha can be used here). The final layer of our model, preceding the output layer, will have 16 neurons with a 10% dropout rate for preventing overfitting.

Another necessary technique to use with deep learning is **data augmentation**. Data augmentation is a method that can help improve the accuracy of machine learning models by creating additional artificial data. A data augmentation system makes small, random changes to your training data during the training process (such as flipping, cropping, or rotating the images).

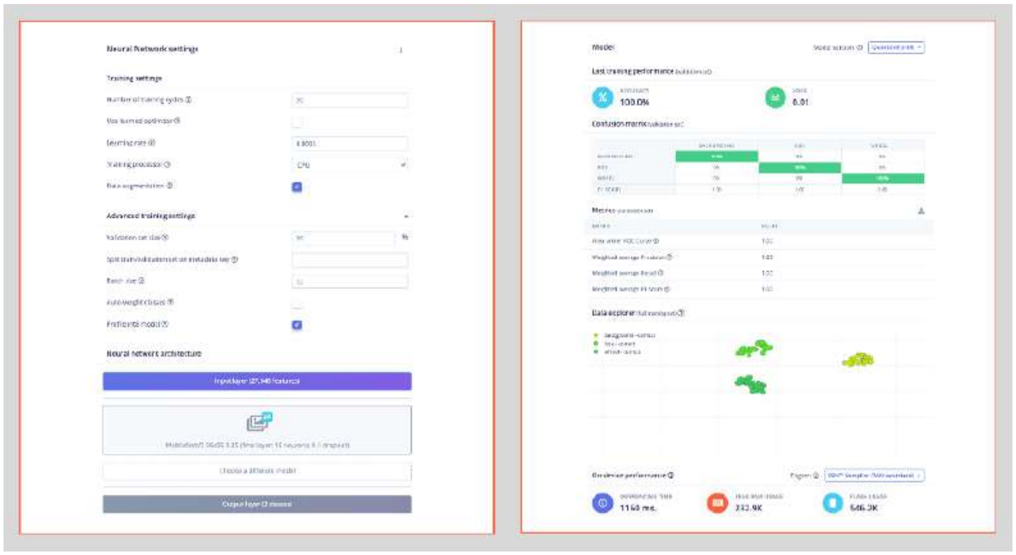
Under the hood, here you can see how Edge Impulse implements a data Augmentation policy on your data:

```
# Implements the data augmentation policy
def augment_image(image, label):
    # Flips the image randomly
    image = tf.image.random_flip_left_right(image)

    # Increase the image size, then randomly crop it down to
    # the original dimensions
    resize_factor = random.uniform(1, 1.2)
    new_height = math.floor(resize_factor * INPUT_SHAPE[0])
    new_width = math.floor(resize_factor * INPUT_SHAPE[1])
    image = tf.image.resize_with_crop_or_pad(image, new_height,
                                              new_width)
    image = tf.image.random_crop(image, size=INPUT_SHAPE)

    # Vary the brightness of the image
    image = tf.image.random_brightness(image, max_delta=0.2)

    return image, label
```

Now, let's us define the hyperparameters:

- Epochs: 20,
- Bach Size: 32
- Learning Rate: 0.0005
- Validation size: 20%

And, so, we have as a training result:


Neural Network settings

Training settings

Training processor: CPU

Advanced training settings

Neural network architecture

Model

Last training performance

100.0%

0.01

Confusion matrix

Metrics

Data explorer

On-device performance

1160 ms.

233.9K

546.3K

The model profile predicts **233 KB of RAM and 546 KB of Flash**, indicating no problem with the Xiao ESP32S3, which has 8 MB of PSRAM. Additionally, the Studio indicates a

latency of around 1160 ms, which is very high. However, this is to be expected, given that we are using the ESP-EYE, whose CPU is an Extensa LX6, and the ESP32S3 uses a newer and more powerful Xtensa LX7.

> With the test data, we also achieved 100% accuracy, even with a quantized INT8 model. This result is not typical in real projects, but our project here is relatively simple, with two objects that are very distinctive from each other.

Model Deployment

We can deploy the trained model:

- As `.TFLITE` to be used on the **SenseCraft AI**
- As an `Arduino Library` in the **Edge Impulse Studio**.

Let's start with the SenseCraft, which is more straightforward and more intuitive.

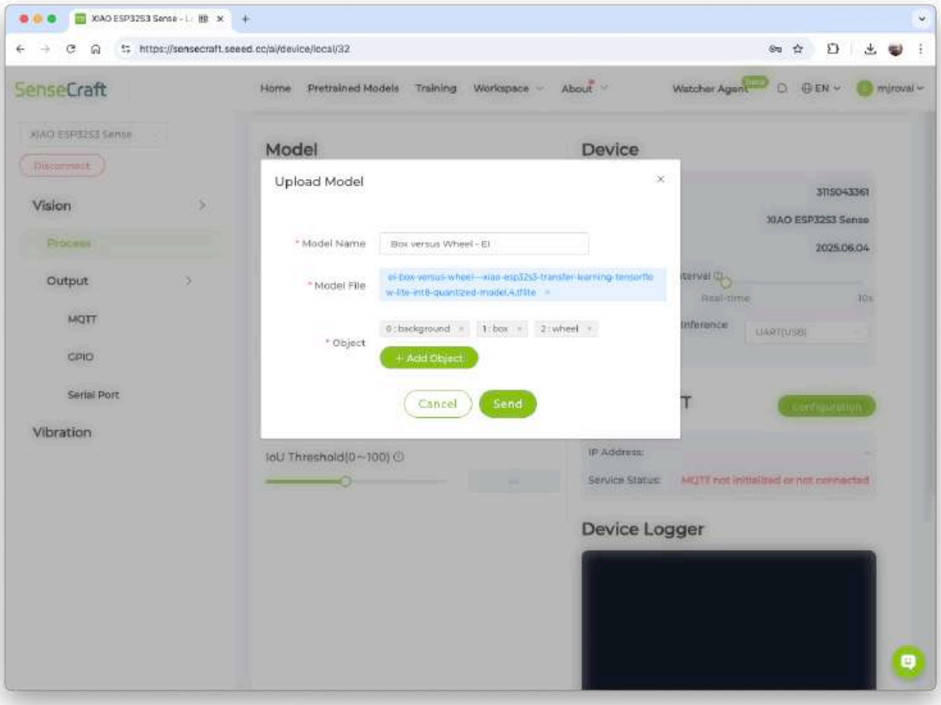
Model Deployment on the SenseCraft AI

On the **Dashboard**, it is possible to download the trained model in several different formats. Let's download `TensorFlow Lite (int8 quantized)`, which has a size of 623KB.

Transfer learning model   TensorFlow Lite (int8 quantized)   623 KB

On **SenseCraft AI Studio**, go to the `Workspace` tab, select `XIAO ESP32S3`, the corresponding Port, and connect the device.

You should see the last model that was uploaded to the device. Select the green button `[Upload Model]`. A pop-up window will prompt you to enter the model name, the model file, and the class names (**objects**). We should use labels in alphabetical order: `0: background`, `1: box`, and `2: wheel`, and then press `[Send]`.



Upload Model

* Model Name: Box versus Wheel - EI

* Model File: ei-box-versus-wheel---xiao-esp32s3-transfer-learning-tensorflow-lite-int8-quantized-model.4.tflite

* Object: 0 : background, 1 : box, 2 : wheel

IoU Threshold(0~100)

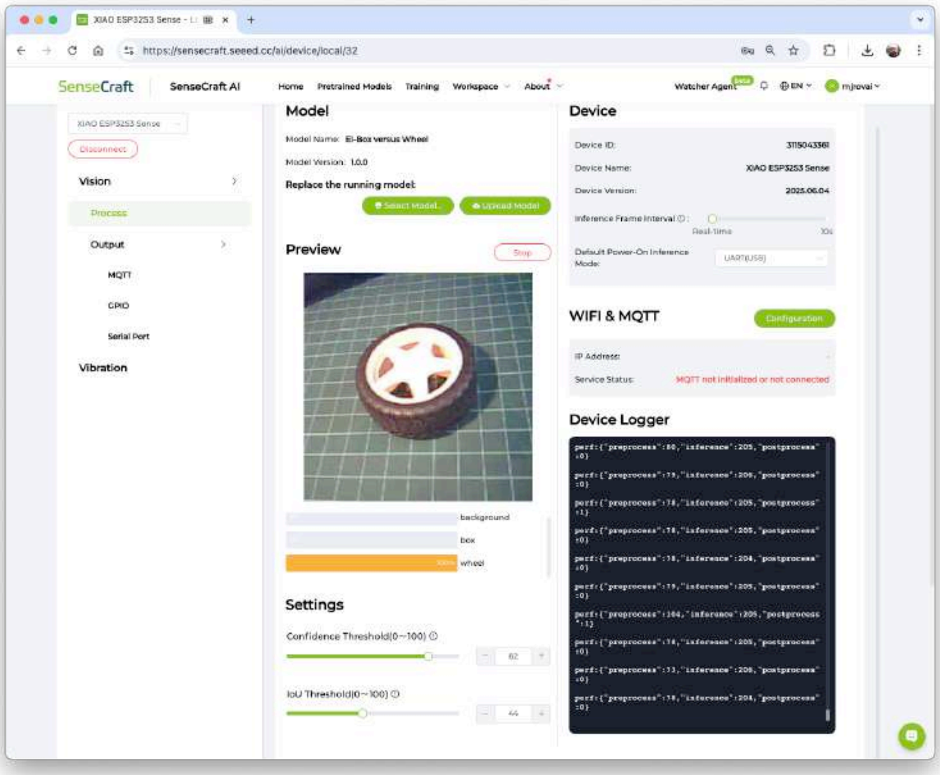
IP Address:

Service Status: MQTT not initialized or not connected

After a few seconds, the model will be uploaded ("flashed") to our device, and the camera image will appear in real-time on the **Preview** Sector. The Classification result will be displayed under the image preview. It is also possible to select the `Confidence Threshold` of your inference using the cursor on **Settings**.

On the **Device Logger**, we can view the Serial Monitor, where we can observe the latency, which is approximately 81 ms for pre-processing and 205 ms for inference, **corresponding to a frame rate of 3.4 frames per second (FPS)**, what is double of we got, training the model on SenseCraft, because we are working with smaller images (96x96 versus 224x224).

> The total latency is around **4 times faster** than the estimation made in Edge Impulse Studio on an Xtensa LX6 CPU; now we are performing the inference on an Xtensa LX7 CPU.


Model

Model Name: EI-Box versus Wheel

Model Version: 1.0.0

Device

Device ID: 3115043361

Device Name: XIAO ESP32S3 Sense

Device Version: 2025.06.04

WIFI & MQTT

Service Status: MQTT not initialized or not connected

Settings

Confidence Threshold(0~100): 62

IoU Threshold(0~100): 44

Device Logger

```
perf:{"preprocess":80,"inference":205,"postprocess":0}
perf:{"preprocess":79,"inference":206,"postprocess":0}
perf:{"preprocess":78,"inference":205,"postprocess":1}
perf:{"preprocess":78,"inference":205,"postprocess":0}
perf:{"preprocess":78,"inference":204,"postprocess":0}
perf:{"preprocess":79,"inference":205,"postprocess":0}
perf:{"preprocess":104,"inference":205,"postprocess":1}
perf:{"preprocess":78,"inference":205,"postprocess":0}
perf:{"preprocess":73,"inference":206,"postprocess":0}
perf:{"preprocess":78,"inference":204,"postprocess":0}
```

Post-Processing

It is possible to obtain the output of a model inference, including Latency, Class ID, and Confidence, as shown on the Device Logger in SenseCraft AI. This allows us to utilize the **XIAO ESP32S3 Sense as an AI sensor**. In other words, we can retrieve the model data using different communication protocols such as MQTT, UART, I2C, or SPI, depending on our project requirements.

> The idea is similar to what we have done on the Seeed Grove Vision AI V2 Image Classification Post-Processing Lab.

Below is an example of a connection using the I2C bus.

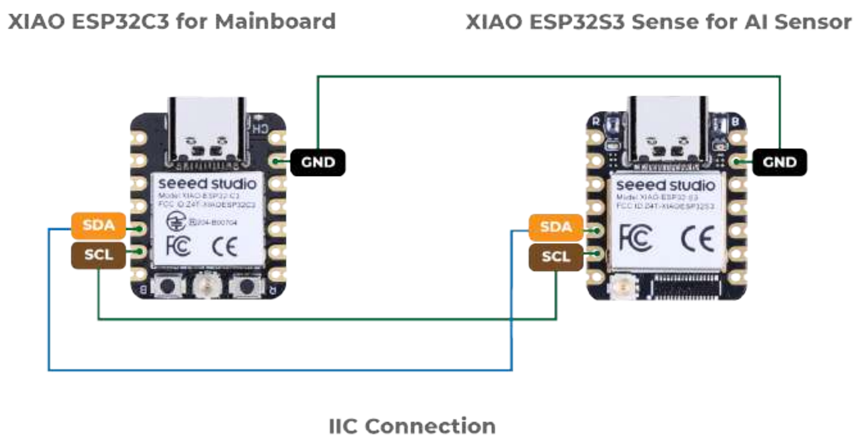
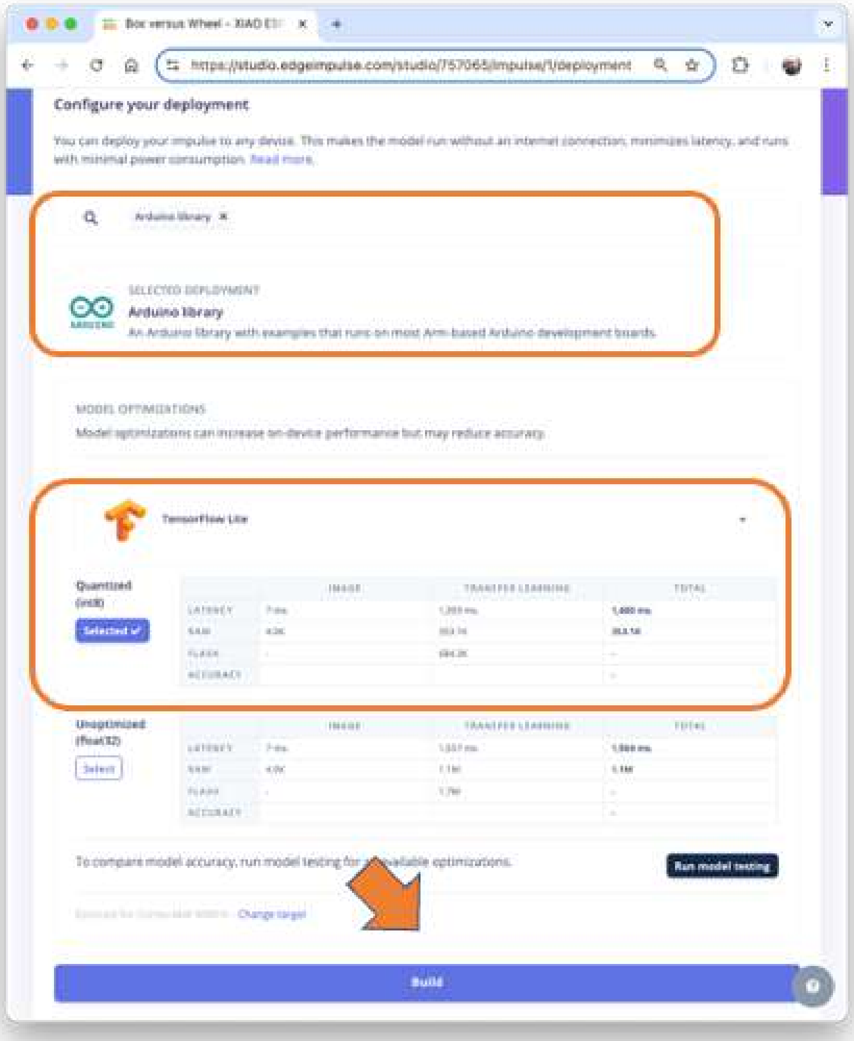


Figure 3: As a Sensor | Seeed Studio Wiki

Please refer to the Seeed Studio Wiki for more information.

Model Deployment as an Arduino Library at EI Studio

On the **Deploy** section at Edge Impulse Studio, Select `Arduino library`, `TensorFlow Lite`, `Quantized(int8)`, and press `[Build]`. The trained model will be downloaded as a .zip Arduino library:


Box versus Wheel - XIAO ES
https://studio.edgeimpulse.com/studio/757065/impulse/1/deployment

Configure your deployment

You can deploy your impulse to any device. This makes the model run without an internet connection, minimizes latency, and runs with minimal power consumption. Read more.

Arduino library

SELECTED DEPLOYMENT
Arduino library
An Arduino library with examples that runs on most Arm-based Arduino development boards.

MODEL OPTIMIZATIONS
Model optimizations can increase on-device performance but may reduce accuracy.

TensorFlow Lite

Quantized (int8) [✓ Selected]

	IMAGE	TRANSFER LEARNING	TOTAL
LATENCY	7 ms.	1,393 ms.	1,400 ms.
RAM	4.3K	353.7K	353.1K
FLASH	-	684.2K	-
ACCURACY			-

Unoptimized (float32) Select

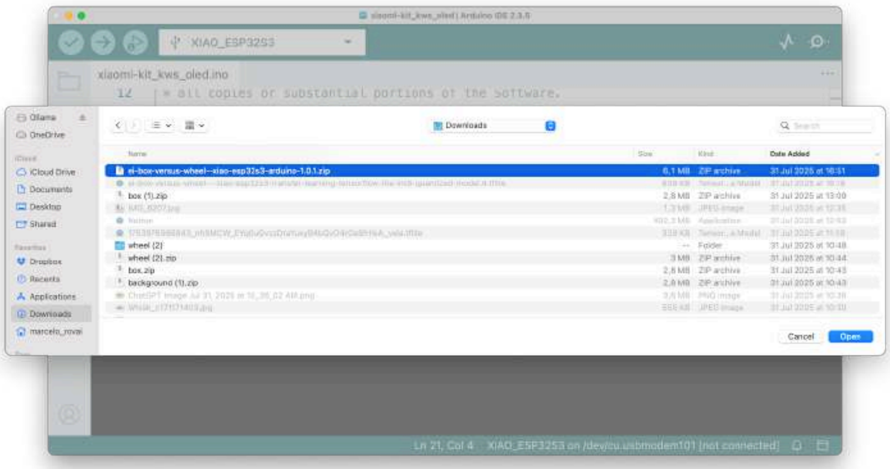
	IMAGE	TRANSFER LEARNING	TOTAL
LATENCY	7 ms.	1,557 ms.	1,564 ms.
RAM	4.0K	1.1M	1.1M
FLASH	-	1.7M	-
ACCURACY			-

To compare model accuracy, run model testing for all available optimizations. Run model testing

Estimate for Espressif ESP-EYE · Change target

Build

Open your Arduino IDE, and under **Sketch,** go to **Include Library** and **add .ZIP Library.** Next, select the file downloaded from Edge Impulse Studio and press `[Open]`.



Go to the Arduino IDE `Examples` and look for the project by its name (in this case: “Box_versus_Whell_...Interfering”. Open `esp32 -> esp32_camera`. The sketch `esp32_camera.ino` will be downloaded to the IDE.

This sketch was developed for the standard ESP32 and will not work with the XIAO ESP32S3 Sense. It should be modified. Let’s download the modified one from the project GitHub: Image_class_XIAOML-Kit.ino.

XIAO ESP32S3 Image Classification Code Explained

The code captures images from the onboard camera, processes them, and classifies them (in this case, “Box”, “Wheel”, or “Background”) using the trained model on EI Studio. It runs continuously, performing real-time inference on the edge device.

In short,:

Camera → JPEG Image → RGB888 Conversion → Resize to 96x96 → Neural Network → Classification Results → Serial Output

0.0.0.0.1 * Key Components

1. **Library Includes and Dependencies**

```
#include <Box_versus_Wheel_-_XIAO_ESP32S3_inferencing.h>
#include "edge-impulse-sdk/dsp/image/image.hpp"
#include "esp_camera.h"
```

- **Edge Impulse Inference Library**: Contains our trained model and inference engine

- **Image Processing**: Provides functions for image manipulation
- **ESP Camera**: Hardware interface for the camera module

2. **Camera Pin Configurations**

The XIAO ESP32S3 Sense can work with different camera sensors (OV2640 or OV3660), which may have different pin configurations. The code defines three possible configurations:

```
// Configuration 1: Most common OV2640 configuration
#define CONFIG_1_XCLK_GPIO_NUM    10
#define CONFIG_1_SIOD_GPIO_NUM    40
#define CONFIG_1_SIOC_GPIO_NUM    39
// ... more pins
```

This flexibility allows the code to automatically try different pin mappings if the first one doesn't work, making it more robust across different hardware revisions.

3. **Memory Management Settings**

```
#define EI_CAMERA_RAW_FRAME_BUFFER_COLS   320
#define EI_CAMERA_RAW_FRAME_BUFFER_ROWS   240
#define EI_CLASSIFIER_ALLOCATION_HEAP      1
```

- **Frame Buffer Size**: Defines the raw image size (320x240 pixels)
- **Heap Allocation**: Uses dynamic memory allocation for flexibility
- **PSRAM Support**: The ESP32S3 has 8MB of PSRAM for storing large data like images

0.0.0.0.2 * `setup()` - Initialization

```
void setup() {
    Serial.begin(115200);
    while (!Serial);

    if (ei_camera_init() == false) {
        ei_printf("Failed to initialize Camera!\r\n");
    } else {
        ei_printf("Camera initialized\r\n");
    }

    ei_sleep(2000);  // Wait 2 seconds before starting
}
```

This function:

3. Initializes serial communication for debugging output
4. Initializes the camera with automatic configuration detection
5. Waits 2 seconds before starting continuous inference

0.0.0.0.3 * `loop()` - Main Processing Loop

The loop performs these steps continuously:

Step 1: Memory Allocation

```
snapshot_buf = (uint8_t*)ps_malloc(EI_CAMERA_RAW_FRAME_BUFFER_COLS *
                                   EI_CAMERA_RAW_FRAME_BUFFER_ROWS *
                                   EI_CAMERA_FRAME_BYTE_SIZE);
```

Allocates memory for the image buffer, preferring PSRAM (faster external RAM) but falling back to regular heap if needed.

Step 2: Image Capture

```
if (ei_camera_capture((size_t)EI_CLASSIFIER_INPUT_WIDTH,
                     (size_t)EI_CLASSIFIER_INPUT_HEIGHT,
                     snapshot_buf) == false) {
    ei_printf("Failed to capture image\r\n");
    free(snapshot_buf);
    return;
}
```

Captures an image from the camera and stores it in the buffer.

Step 3: Run Inference

```
ei_impulse_result_t result = { 0 };
EI_IMPULSE_ERROR err = run_classifier(&signal, &result, false);
```

Runs the machine learning model on the captured image.

Step 4: Output Results

```
for (uint16_t i = 0; i < EI_CLASSIFIER_LABEL_COUNT; i++) {
    ei_printf("  %s: %.5f\r\n",
              ei_classifier_inferencing_categories[i],
              result.classification[i].value);
}
```

Prints the classification results showing confidence scores for each category.

0.0.0.0.4 * `ei_camera_init()` - Smart Camera Initialization

This function implements an intelligent initialization sequence:

```
bool ei_camera_init(void) {
    // Try Configuration 1 (OV2640 common)
    update_camera_config(1);
    esp_err_t err = esp_camera_init(&camera_config);
    if (err == ESP_OK) goto camera_init_success;

    // Try Configuration 2 (OV3660)
    esp_camera_deinit();
    update_camera_config(2);
    err = esp_camera_init(&camera_config);
    if (err == ESP_OK) goto camera_init_success;

    // Continue trying other configurations...
}
```

The function:

1. Tries multiple pin configurations
2. Tests different clock frequencies (10MHz or 16MHz)
3. Attempts PSRAM first, then falls back to DRAM
4. Applies sensor-specific settings based on detected hardware

0.0.0.0.5 * `ei_camera_capture()` - Image Processing Pipeline

```
bool ei_camera_capture(uint32_t img_width, uint32_t img_height, uint8_t *out_buf) {
    // 1. Get frame from camera
    camera_fb_t *fb = esp_camera_fb_get();

    // 2. Convert JPEG to RGB888 format
    bool converted = fmt2rgb888(fb->buf, fb->len, PIXFORMAT_JPEG, snapshot_buf);

    // 3. Return frame buffer to camera driver
    esp_camera_fb_return(fb);

    // 4. Resize if needed
    if (do_resize) {
        ei::image::processing::crop_and_interpolate_rgb888(...);
    }
}
```

This function:

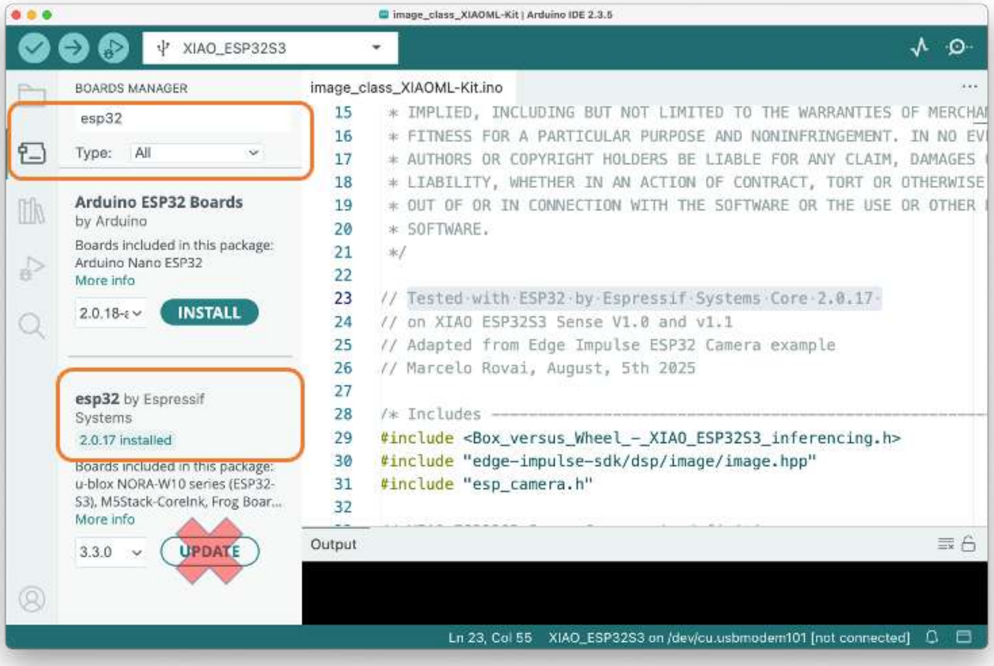
1. Captures a JPEG image from the camera
2. Converts it to RGB888 format (required by the ML model)
3. Resizes the image to match the model's input size (96x96 pixels)

Inference

- Upload the code to the XIAO ESP32S3 Sense.

  Attention

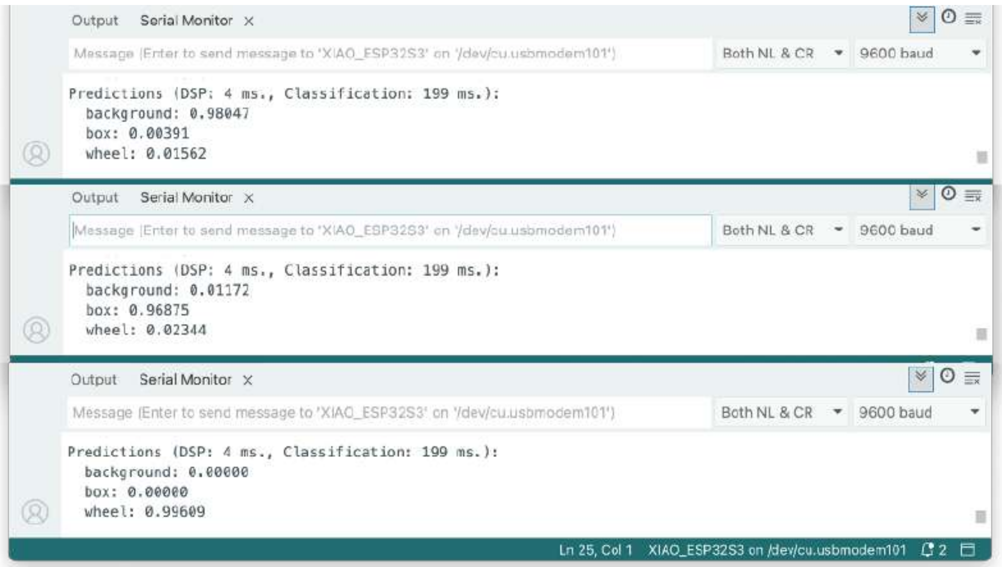
  - The Xiao ESP32S3 **MUST** have the PSRAM enabled. You can check it on the Arduino IDE upper menu: `Tools–> PSRAM:OPI PSRAM`
  - The Arduino Library (`esp32 by Espressif Systems` should be **version 2.017**. Do not update it)



image_class_XIAOML-Kit | Arduino IDE 2.3.5
XIAO_ESP32S3
BOARDS MANAGER
esp32
Type: All
Arduino ESP32 Boards
by Arduino
Boards included in this package:
Arduino Nano ESP32
More info
2.0.18-
INSTALL
esp32 by Espressif Systems
2.0.17 installed
Boards included in this package:
u-blox NORA-W10 series (ESP32-S3), M5Stack-CoreInk, Frog Boar...
More info
3.3.0
UPDATE
image_class_XIAOML-Kit.ino
Output
Ln 23, Col 55 XIAO_ESP32S3 on /dev/cu.usbmodem101 [not connected]

```
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHA
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EV
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
 * SOFTWARE.
 */

// Tested with ESP32 by Espressif Systems Core 2.0.17
// on XIAO ESP32S3 Sense V1.0 and v1.1
// Adapted from Edge Impulse ESP32 Camera example
// Marcelo Rovai, August, 5th 2025

/* Includes ---------------------------------------------------------
#include <Box_versus_Wheel_-_XIAO_ESP32S3_inferencing.h>
#include "edge-impulse-sdk/dsp/image/image.hpp"
#include "esp_camera.h"
```

- Open the Serial Monitor
- Point the camera at the objects, and check the result on the Serial Monitor.


Output
Serial Monitor
Message (Enter to send message to 'XIAO_ESP32S3' on '/dev/cu.usbmodem101')
Both NL & CR
9600 baud

```
Predictions (DSP: 4 ms., Classification: 199 ms.):
  background: 0.98047
  box: 0.00391
  wheel: 0.01562
```

```
Predictions (DSP: 4 ms., Classification: 199 ms.):
  background: 0.01172
  box: 0.96875
  wheel: 0.02344
```

```
Predictions (DSP: 4 ms., Classification: 199 ms.):
  background: 0.00000
  box: 0.00000
  wheel: 0.99609
```

Ln 25, Col 1   XIAO_ESP32S3 on /dev/cu.usbmodem101   2

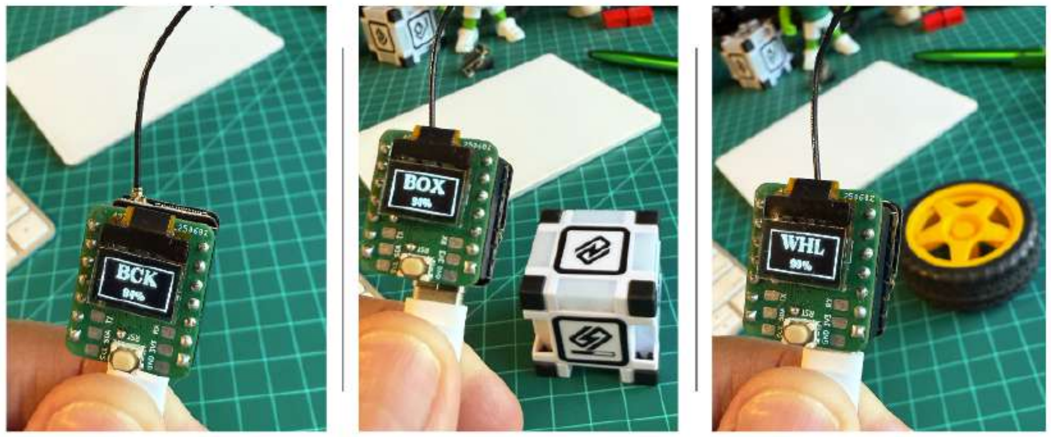
Post-Processing

In edge AI applications, the inference result is only as valuable as our ability to act upon it. While serial output provides detailed information for debugging and development, real-world deployments require immediate, human-readable feedback that doesn't depend on external monitors or connections.

The XIAOML Kit tiny 0.42" OLED display (72×40 pixels) serves as a crucial post-processing component that transforms raw ML inference results into immediate, human-readable feedback—displaying detected class names and confidence levels directly on the device, eliminating the need for external monitors and enabling truly standalone edge AI deployment in industrial, agricultural, or retail environments where instant visual confirmation of AI predictions is essential.

So, let's modify the sketch to automatically adapt to the model trained on Edge Impulse by reading the class names and count directly from the model. The display will show abbreviated class names (3 letters) with larger fonts for better visibility on the tiny 72x40 pixel display. Download the code from the GitHub: XIAOML-Kit-Img_Class_OLED_Gen.

Running the code, we can see the result:



Summary

The XIAO ESP32S3 Sense is a remarkably capable and flexible platform for image classification applications. Through this lab, we've explored two distinct development approaches that cater to different skill levels and project requirements.

- The **SenseCraft AI Studio** provides an accessible entry point with its **no-code interface**, enabling rapid prototyping and deployment of pre-trained models like person detection. With real-time inference and integrated post-processing capabilities, it demonstrates how AI can be deployed without extensive programming or ML knowledge.
- For more advanced applications, **Edge Impulse Studio** offers comprehensive machine learning pipeline tools, including custom dataset management, transfer learning with several pre-trained models, such as MobileNet, and model optimization.

Key insights from this lab include the importance of image resolution trade-offs, the effectiveness of transfer learning for small datasets, and the practical considerations of edge AI deployment, such as power consumption and memory constraints.

The Lab demonstrates fundamental TinyML principles that extend beyond this specific hardware: resource-constrained inference, real-time processing requirements, and the complete pipeline from data collection through model deployment to practical applications. With built-in post-processing capabilities including GPIO control and communication protocols, the XIAO serves as more than just an inference engine—it becomes a complete AI sensor platform.

This foundation in image classification prepares you for more complex computer vision tasks while showcasing how modern edge AI makes sophisticated computer vision accessible, cost-effective, and deployable in real-world embedded applications ranging from industrial automation to smart home systems.

Resources

- Getting Started with the XIAO ESP32S3
- SenseCraft AI Studio Home
- SenseCraft Vision Workspace
- Dataset example
- Edge Impulse Project
- XIAO as an AI Sensor
- Seeed Arduino SSCMA Library
- XIAOML Kit Code

Object Detection

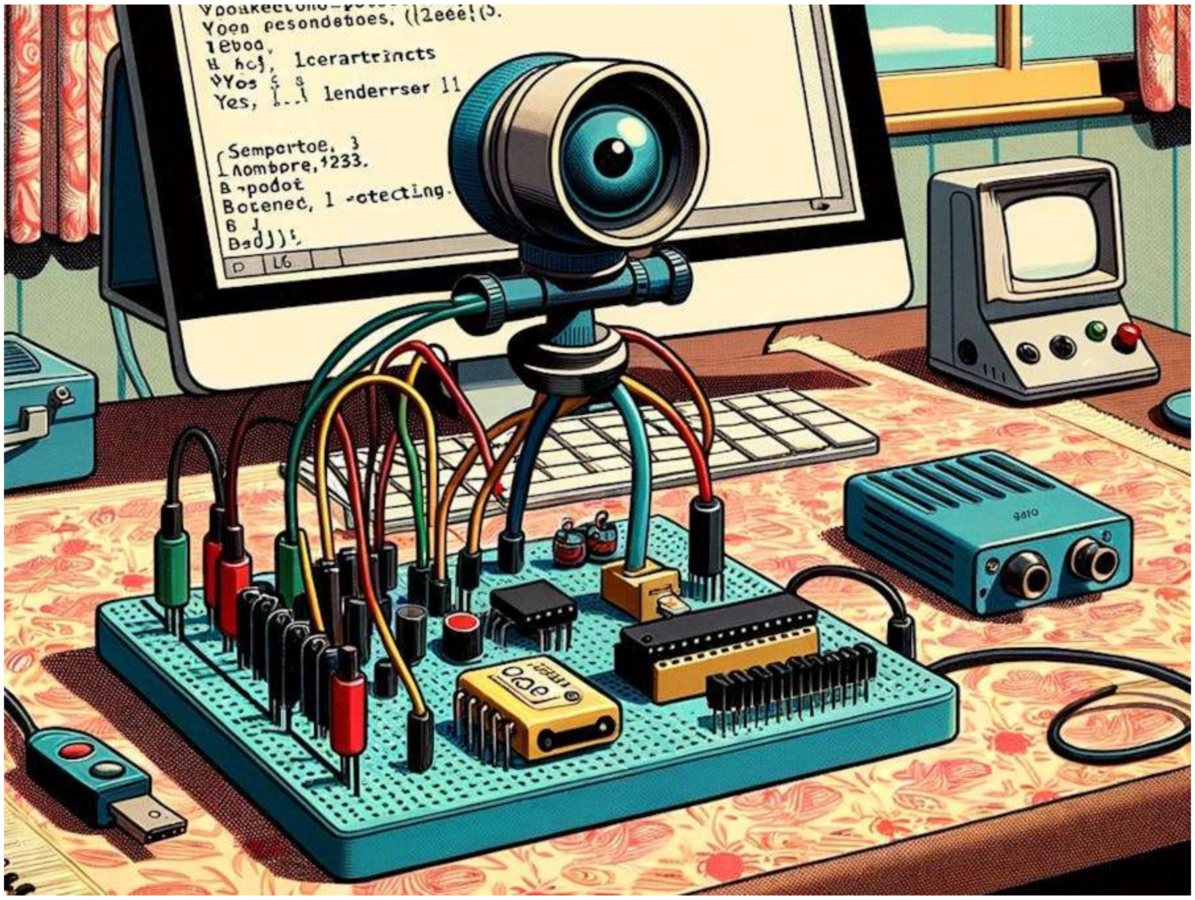


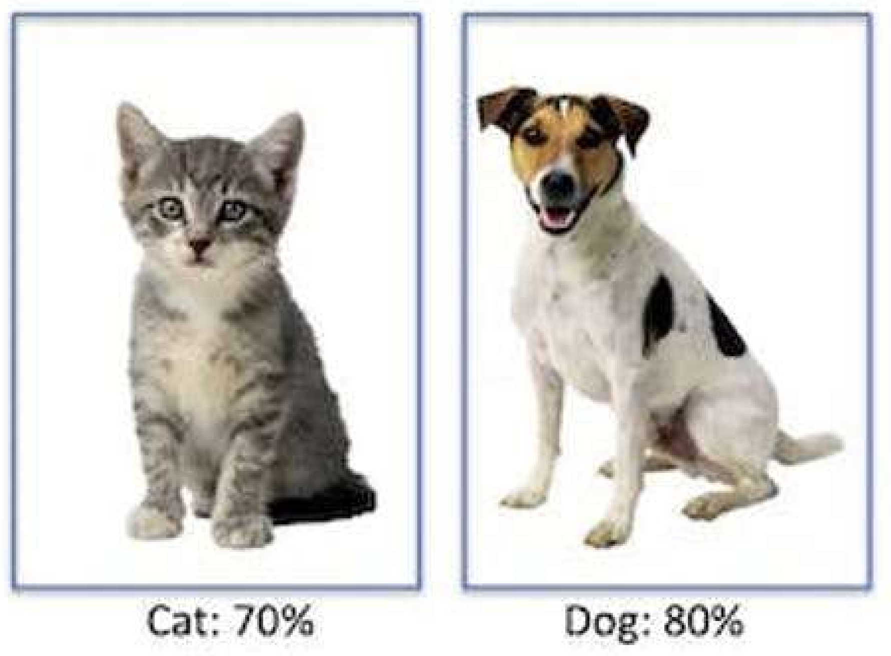
Figure 4: *DALL · E prompt - Cartoon styled after 1950s animations, showing a detailed board with sensors, particularly a camera, on a table with patterned cloth. Behind the board, a computer with a large back showcases the Arduino IDE. The IDE's content hints at LED pin assignments and machine learning inference for detecting spoken commands. The Serial Monitor, in a distinct window, reveals outputs for the commands 'yes' and 'no'.*

Overview

In the last section regarding Computer Vision (CV) and the XIAO ESP32S3, *Image Classification,* we learned how to set up and classify images with this remarkable development board. Continuing our CV journey, we will explore **Object Detection** on microcontrollers.

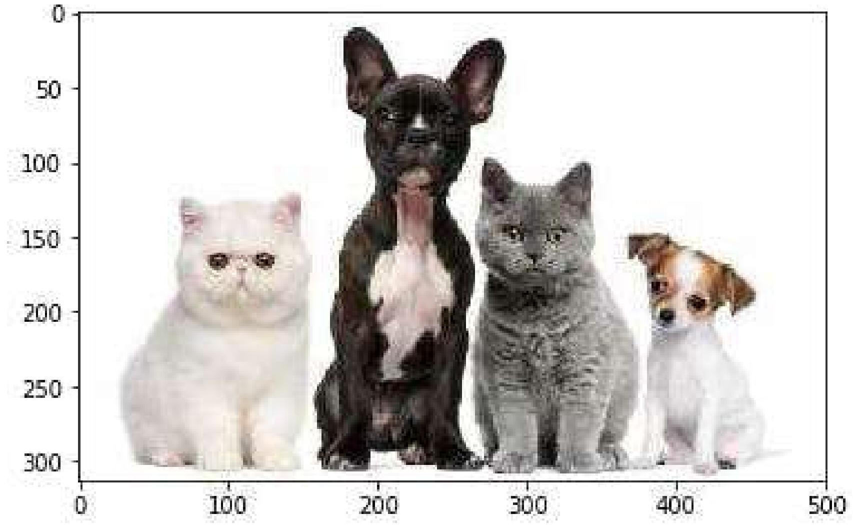
Object Detection versus Image Classification

The main task with Image Classification models is to identify the most probable object category present on an image, for example, to classify between a cat or a dog, dominant "objects" in an image:



But what happens if there is no dominant category in the image?

```
[PREDICTION]          [Prob]

ashcan                : 27%
Egyptian cat          : 19%
hamper                : 13%
```

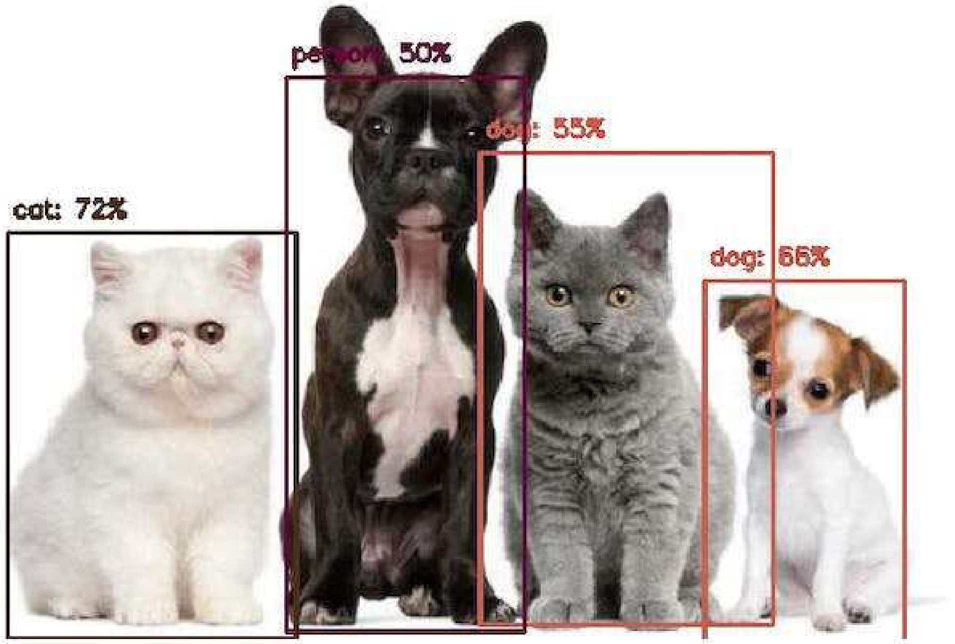


An image classification model identifies the above image utterly wrong as an “ashcan,” possibly due to the color tonalities.

> The model used in the previous images is MobileNet, which is trained with a large dataset, *ImageNet*, running on a Raspberry Pi.

To solve this issue, we need another type of model, where not only **multiple categories** (or labels) can be found but also **where** the objects are located on a given image.

As we can imagine, such models are much more complicated and bigger, for example, the **MobileNetV2 SSD FPN-Lite 320x320, trained with the COCO dataset.** This pre-trained object detection model is designed to locate up to 10 objects within an image, outputting a bounding box for each object detected. The below image is the result of such a model running on a Raspberry Pi:



Those models used for object detection (such as the MobileNet SSD or YOLO) usually have several MB in size, which is OK for use with Raspberry Pi but unsuitable for use with embedded devices, where the RAM usually has, at most, a few MB as in the case of the XIAO ESP32S3.

An Innovative Solution for Object Detection: FOMO

Edge Impulse launched in 2022, **FOMO** (Faster Objects, More Objects), a novel solution to perform object detection on embedded devices, such as the Nicla Vision and Portenta (Cortex M7), on Cortex M4F CPUs (Arduino Nano33 and OpenMV M4 series) as well the Espressif ESP32 devices (ESP-CAM, ESP-EYE and XIAO ESP32S3 Sense).

In this Hands-On project, we will explore Object Detection using FOMO.

> To understand more about FOMO, you can go into the official FOMO announcement by Edge Impulse, where Louis Moreau and Mat Kelcey explain in detail how it works.

The Object Detection Project Goal

All Machine Learning projects need to start with a detailed goal. Let's assume we are in an industrial or rural facility and must sort and count **oranges (fruits)** and particular **frogs**

(bugs).

In other words, we should perform a multi-label classification, where each image can have three classes:

- Background (No objects)
- Fruit
- Bug

Here are some not labeled image samples that we should use to detect the objects (fruits and bugs):

We are interested in which object is in the image, its location (centroid), and how many we can find on it. The object's size is not detected with FOMO, as with MobileNet SSD or YOLO, where the Bounding Box is one of the model outputs.

We will develop the project using the XIAO ESP32S3 for image capture and model inference. The ML project will be developed using the Edge Impulse Studio. But before starting the object detection project in the Studio, let's create a *raw dataset* (not labeled) with images that contain the objects to be detected.

Data Collection

You can capture images using the XIAO, your phone, or other devices. Here, we will use the XIAO with code from the Arduino IDE ESP32 library.

Collecting Dataset with the XIAO ESP32S3

Open the Arduino IDE and select the XIAO_ESP32S3 board (and the port where it is connected). On `File > Examples > ESP32 > Camera`, select `CameraWebServer`.

On the BOARDS MANAGER panel, confirm that you have installed the latest "stable" package.

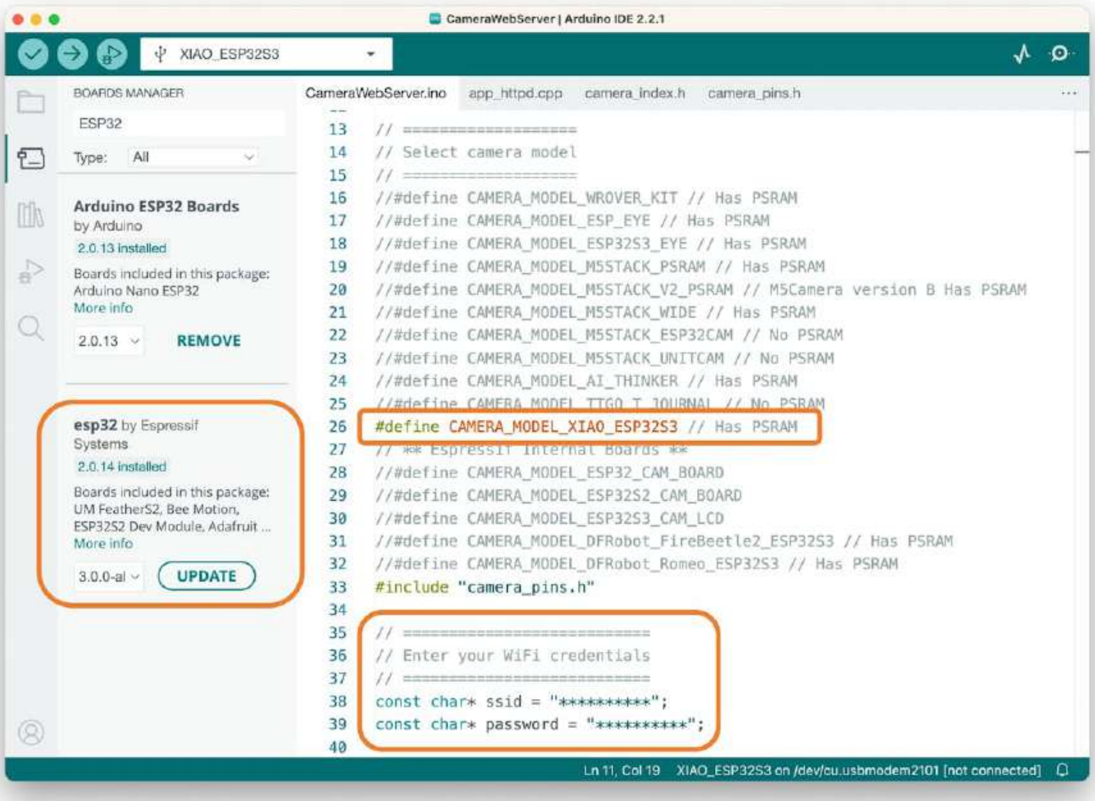
Attention

Alpha versions (for example, 3.x-alpha) do not work correctly with the XIAO and Edge Impulse. Use the last stable version (for example, 2.0.11) instead.

You also should comment on all cameras' models, except the XIAO model pins:

```
#define CAMERA_MODEL_XIAO_ESP32S3 // Has PSRAM
```

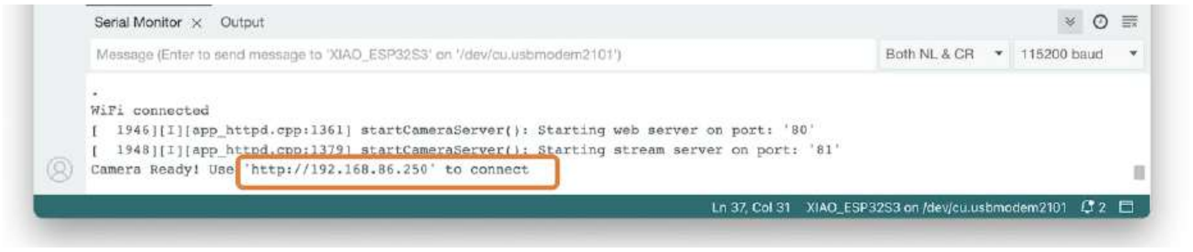
And on `Tools`, enable the PSRAM. Enter your wifi credentials and upload the code to the device:



CameraWebServer | Arduino IDE 2.2.1
XIAO_ESP32S3
BOARDS MANAGER
ESP32
Type: All
Arduino ESP32 Boards
by Arduino
2.0.13 installed
Boards included in this package:
Arduino Nano ESP32
More info
2.0.13
REMOVE
esp32 by Espressif Systems
2.0.14 installed
Boards included in this package:
UM FeatherS2, Bee Motion, ESP32S2 Dev Module, Adafruit ...
More info
3.0.0-al
UPDATE
CameraWebServer.ino
app_httpd.cpp
camera_index.h
camera_pins.h
Ln 11, Col 19   XIAO_ESP32S3 on /dev/cu.usbmodem2101 [not connected]

```
// ===================
// Select camera model
// ===================
//#define CAMERA_MODEL_WROVER_KIT // Has PSRAM
//#define CAMERA_MODEL_ESP_EYE // Has PSRAM
//#define CAMERA_MODEL_ESP32S3_EYE // Has PSRAM
//#define CAMERA_MODEL_M5STACK_PSRAM // Has PSRAM
//#define CAMERA_MODEL_M5STACK_V2_PSRAM // M5Camera version B Has PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE // Has PSRAM
//#define CAMERA_MODEL_M5STACK_ESP32CAM // No PSRAM
//#define CAMERA_MODEL_M5STACK_UNITCAM // No PSRAM
//#define CAMERA_MODEL_AI_THINKER // Has PSRAM
//#define CAMERA_MODEL_TTGO_T_JOURNAL // No PSRAM
#define CAMERA_MODEL_XIAO_ESP32S3 // Has PSRAM
// ** Espressif Internal Boards **
//#define CAMERA_MODEL_ESP32_CAM_BOARD
//#define CAMERA_MODEL_ESP32S2_CAM_BOARD
//#define CAMERA_MODEL_ESP32S3_CAM_LCD
//#define CAMERA_MODEL_DFRobot_FireBeetle2_ESP32S3 // Has PSRAM
//#define CAMERA_MODEL_DFRobot_Romeo_ESP32S3 // Has PSRAM
#include "camera_pins.h"

// ===========================
// Enter your WiFi credentials
// ===========================
const char* ssid = "**********";
const char* password = "**********";

```

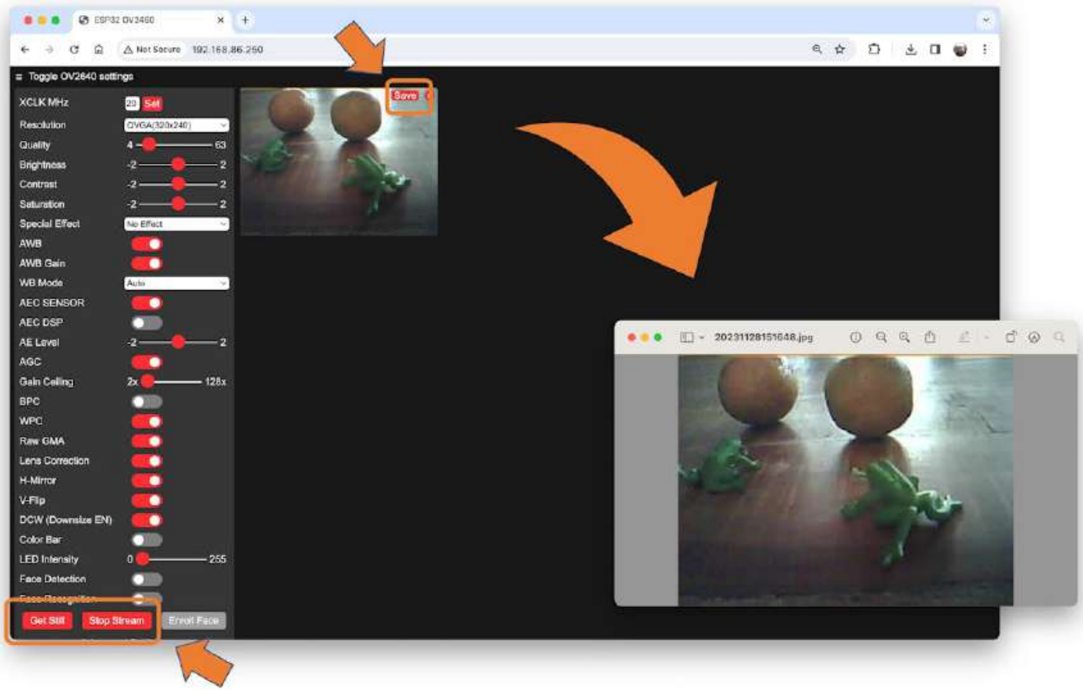
If the code is executed correctly, you should see the address on the Serial Monitor:



Serial Monitor
Output
Message (Enter to send message to 'XIAO_ESP32S3' on '/dev/cu.usbmodem2101')
Both NL & CR
115200 baud
Ln 37, Col 31   XIAO_ESP32S3 on /dev/cu.usbmodem2101

```
.
WiFi connected
[  1946][I][app_httpd.cpp:1361] startCameraServer(): Starting web server on port: '80'
[  1948][I][app_httpd.cpp:1379] startCameraServer(): Starting stream server on port: '81'
Camera Ready! Use 'http://192.168.86.250' to connect
```

Copy the address on your browser and wait for the page to be uploaded. Select the camera resolution (for example, QVGA) and select `[START STREAM]`. Wait for a few seconds/minutes, depending on your connection. You can save an image on your computer download area using the [Save] button.



XCLK MHz 20 Set
Resolution QVGA(320x240)
Quality 4 63
Brightness -2 2
Contrast -2 2
Saturation -2 2
Special Effect No Effect
AWB
AWB Gain
WB Mode Auto
AEC SENSOR
AEC DSP
AE Level -2 2
AGC
Gain Ceiling 2x 128x
BPC
WPC
Raw GMA
Lens Correction
H-Mirror
V-Flip
DCW (Downsize EN)
Color Bar
LED Intensity 0 255
Face Detection
Face Recognition

Edge impulse suggests that the objects should be similar in size and not overlapping for better performance. This is OK in an industrial facility, where the camera should be fixed, keeping the same distance from the objects to be detected. Despite that, we will also try using mixed sizes and positions to see the result.

> We do not need to create separate folders for our images because each contains multiple labels.

We suggest using around 50 images to mix the objects and vary the number of each appearing on the scene. Try to capture different angles, backgrounds, and light conditions.

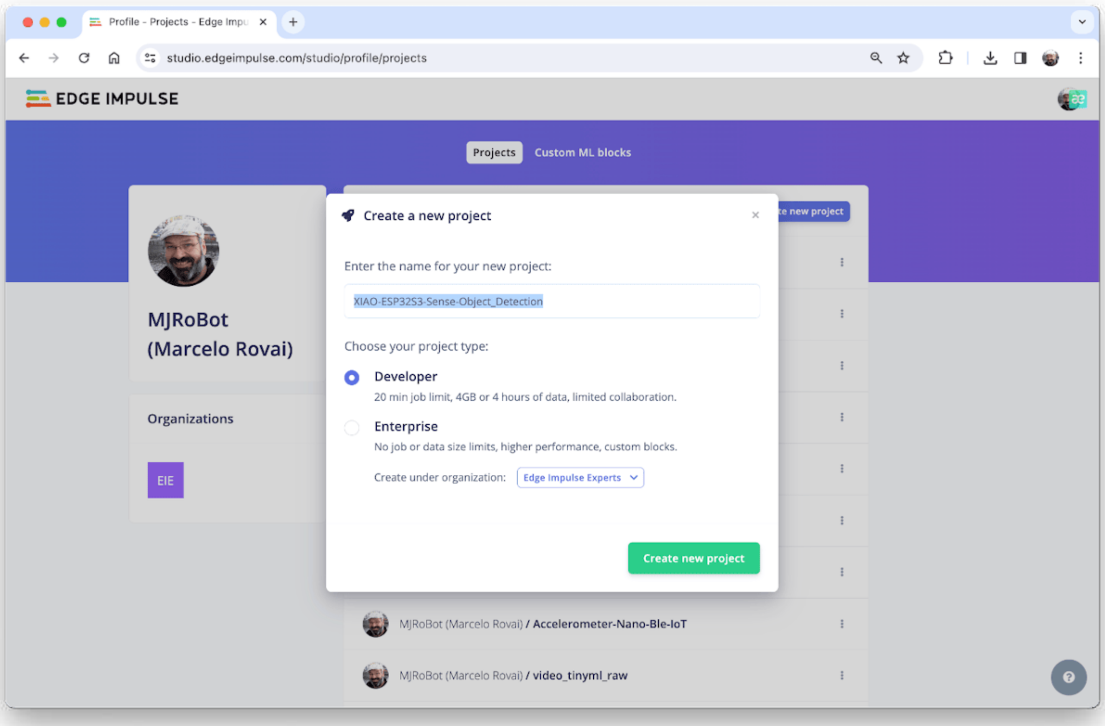
> The stored images use a QVGA frame size of 320×240 and RGB565 (color pixel format).

After capturing your dataset, `[Stop Stream]` and move your images to a folder.

Edge Impulse Studio

Setup the project

Go to Edge Impulse Studio, enter your credentials at **Login** (or create an account), and start a new project.


Profile - Projects - Edge Impulse
studio.edgeimpulse.com/studio/profile/projects
EDGE IMPULSE
Projects
Custom ML blocks
MJRoBot (Marcelo Rovai)
Organizations
EIE
MJRoBot (Marcelo Rovai) / Accelerometer-Nano-Ble-IoT
MJRoBot (Marcelo Rovai) / video_tinyml_raw

Create a new project

Enter the name for your new project: XIAO-ESP32S3-Sense-Object_Detection

Choose your project type:

[✓ Developer] 20 min job limit, 4GB or 4 hours of data, limited collaboration.

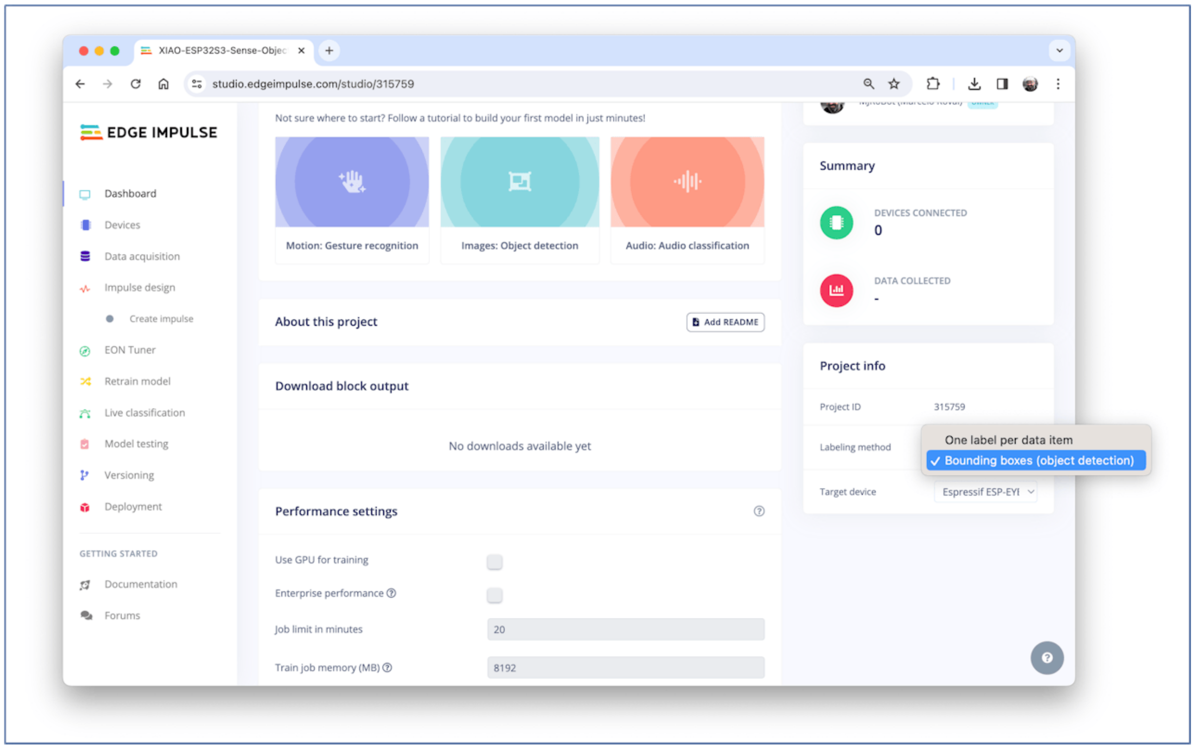
Enterprise — No job or data size limits, higher performance, custom blocks.

Create under organization: Edge Impulse Experts

Create new project

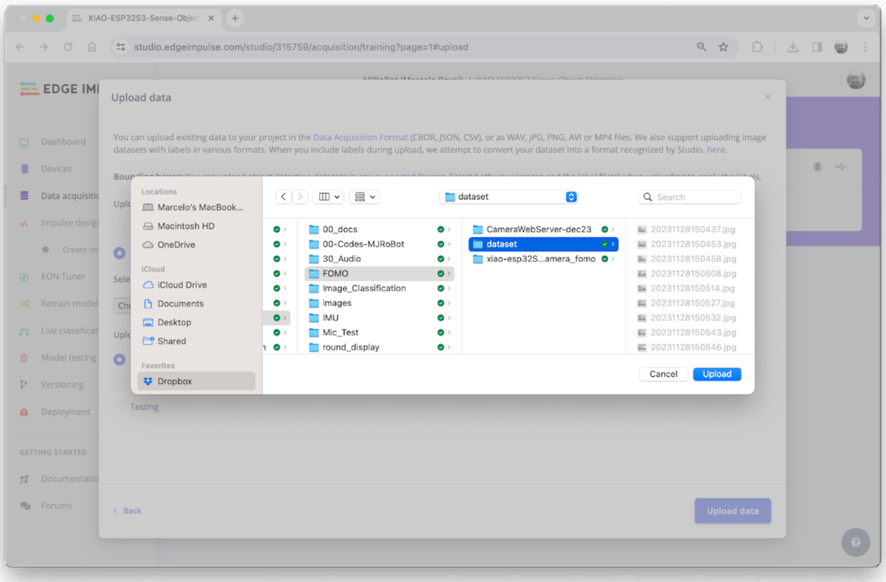
> Here, you can clone the project developed for this hands-on: XIAO-ESP32S3-Sense-Object_Detection

On your Project Dashboard, go down and on **Project info** and select **Bounding boxes (object detection)** and **Espressif ESP-EYE** (most similar to our board) as your Target Device:

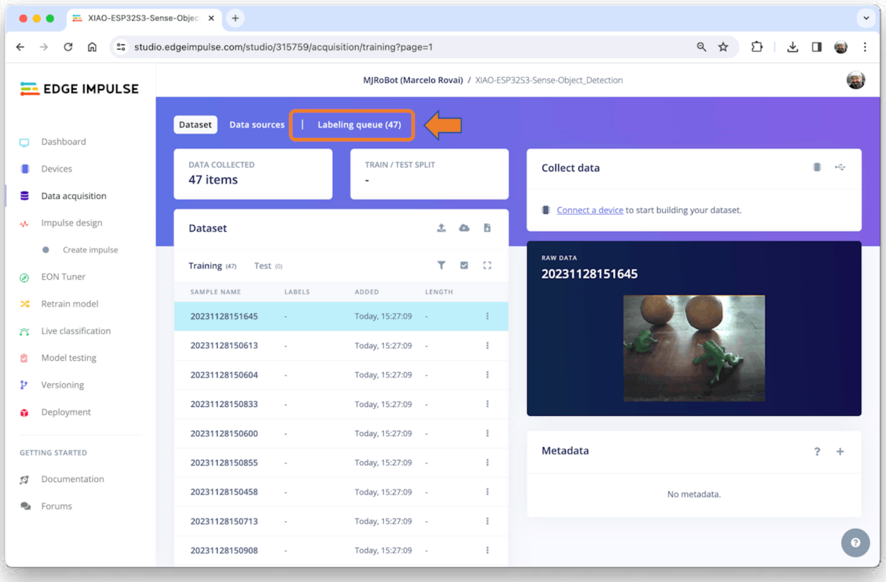


Uploading the unlabeled data

On Studio, go to the `Data acquisition` tab, and on the `UPLOAD DATA` section, upload files captured as a folder from your computer.



You can leave for the Studio to split your data automatically between Train and Test or do it manually. We will upload all of them as training.



All the not-labeled images (47) were uploaded but must be labeled appropriately before being used as a project dataset. The Studio has a tool for that purpose, which you can find in the link Labeling queue (47).

There are two ways you can use to perform AI-assisted labeling on the Edge Impulse Studio (free version):

- Using yolov5
- Tracking objects between frames

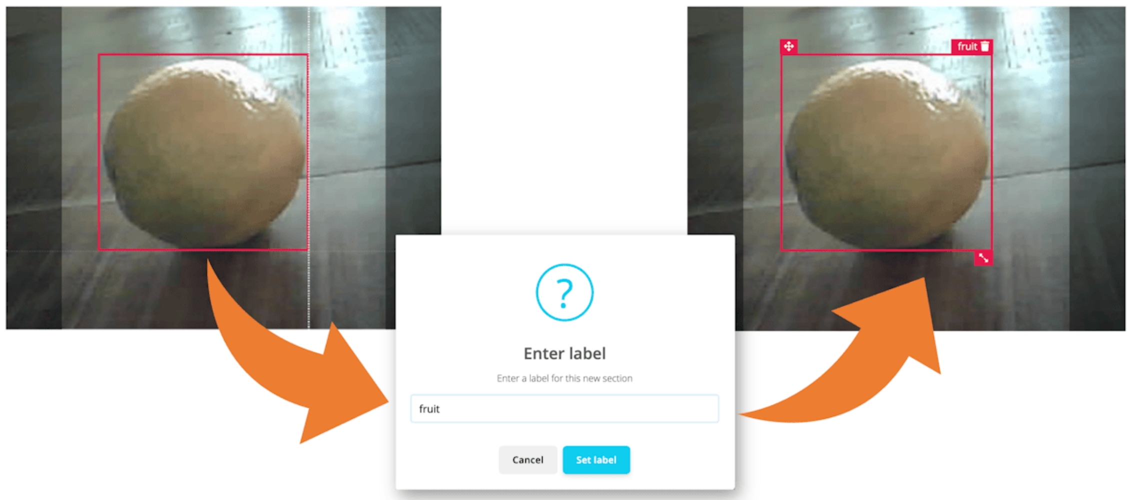
> Edge Impulse launched an auto-labeling feature for Enterprise customers, easing labeling tasks in object detection projects.

Ordinary objects can quickly be identified and labeled using an existing library of pre-trained object detection models from YOLOv5 (trained with the COCO dataset). But since, in our case, the objects are not part of COCO datasets, we should select the option of tracking objects. With this option, once you draw bounding boxes and label the images in one frame, the objects will be tracked automatically from frame to frame, *partially* labeling the new ones (not all are correctly labeled).

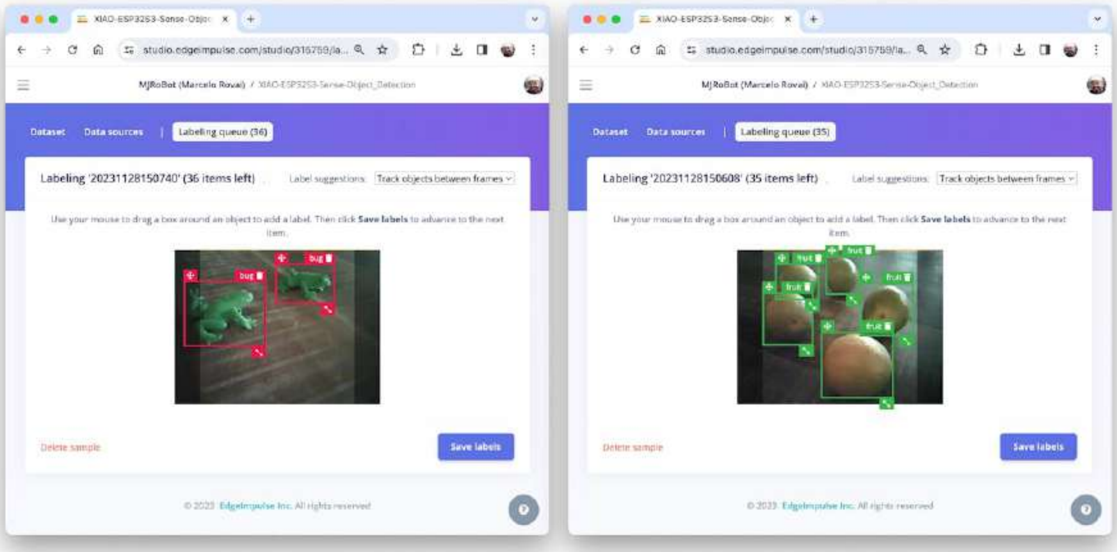
> You can use the EI uploader to import your data if you already have a labeled dataset containing bounding boxes.

Labeling the Dataset

Starting with the first image of your unlabeled data, use your mouse to drag a box around an object to add a label. Then click **Save labels** to advance to the next item.



Continue with this process until the queue is empty. At the end, all images should have the objects labeled as those samples below:



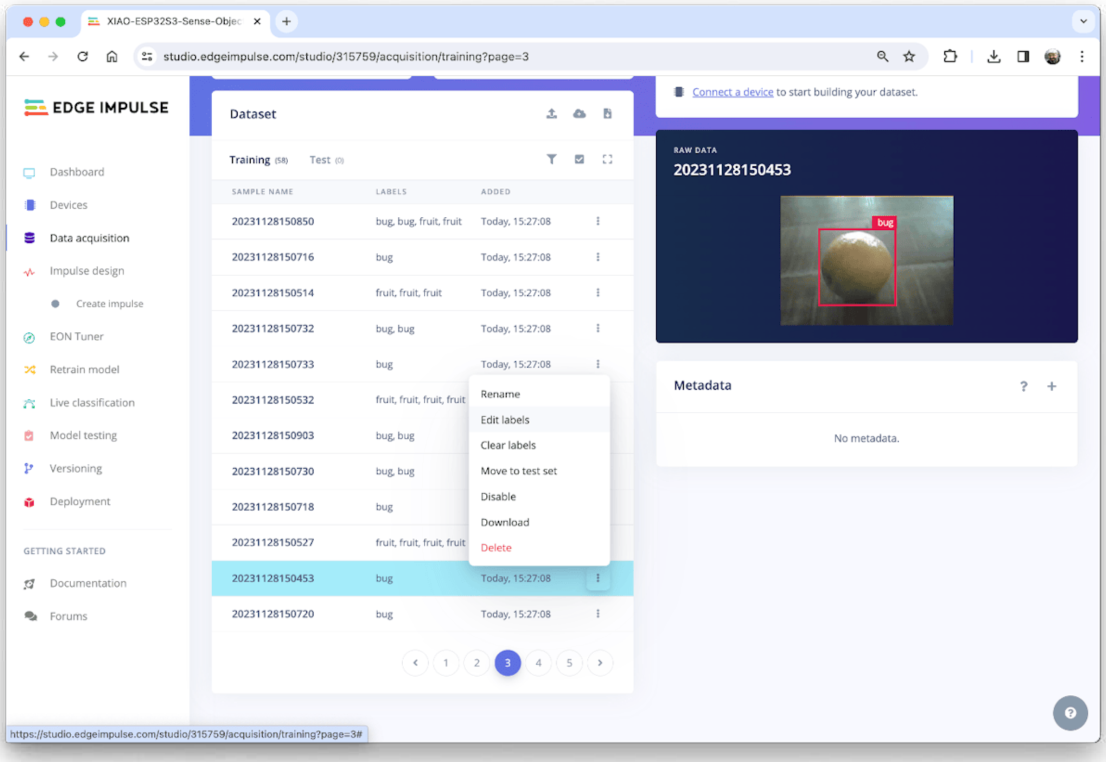
XIAO-ESP32S3-Sense-Objec
studio.edgeimpulse.com/studio/315759/la...
MJRoBot (Marcelo Rovai) / XIAO-ESP32S3-Sense-Object_Detection
Dataset
Data sources
Labeling queue (36)
Labeling queue (35)
bug
fruit
Delete sample
Save labels
© 2023 EdgeImpulse Inc. All rights reserved

Labeling '20231128150740' (36 items left) Label suggestions: Track objects between frames

Labeling '20231128150608' (35 items left) Label suggestions: Track objects between frames

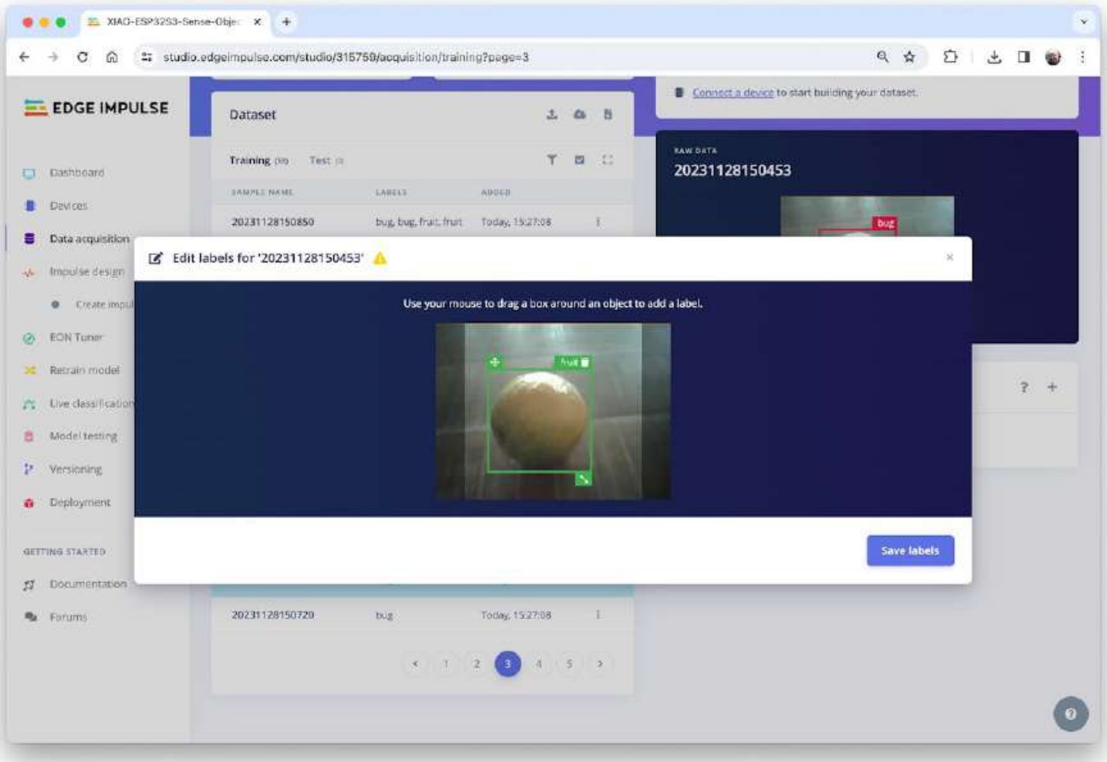
Use your mouse to drag a box around an object to add a label. Then click **Save labels** to advance to the next item.

Next, review the labeled samples on the `Data acquisition` tab. If one of the labels is wrong, you can edit it using the *three dots* menu after the sample name:



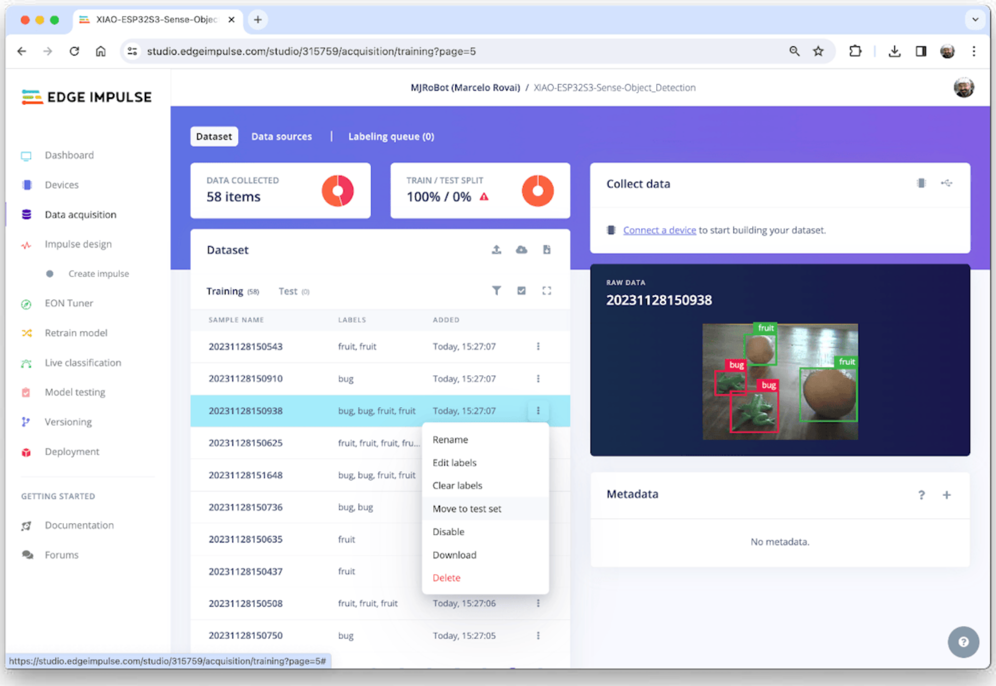
SAMPLE NAME	LABELS	ADDED
20231128150850	bug, bug, fruit, fruit	Today, 15:27:08
20231128150716	bug	Today, 15:27:08
20231128150514	fruit, fruit, fruit	Today, 15:27:08
20231128150732	bug, bug	Today, 15:27:08
20231128150733	bug	Today, 15:27:08
20231128150532	fruit, fruit, fruit, fruit	
20231128150903	bug, bug	
20231128150730	bug, bug	
20231128150718	bug	
20231128150527	fruit, fruit, fruit, fruit	
20231128150453	bug	Today, 15:27:08
20231128150720	bug	Today, 15:27:08

You will be guided to replace the wrong label and correct the dataset.


Edit labels for '20231128150453'
Use your mouse to drag a box around an object to add a label.
fruit
Save labels

Balancing the dataset and split Train/Test

After labeling all data, it was realized that the class fruit had many more samples than the bug. So, 11 new and additional bug images were collected (ending with 58 images). After labeling them, it is time to select some images and move them to the test dataset. You can do it using the three-dot menu after the image name. I selected six images, representing 13% of the total dataset.

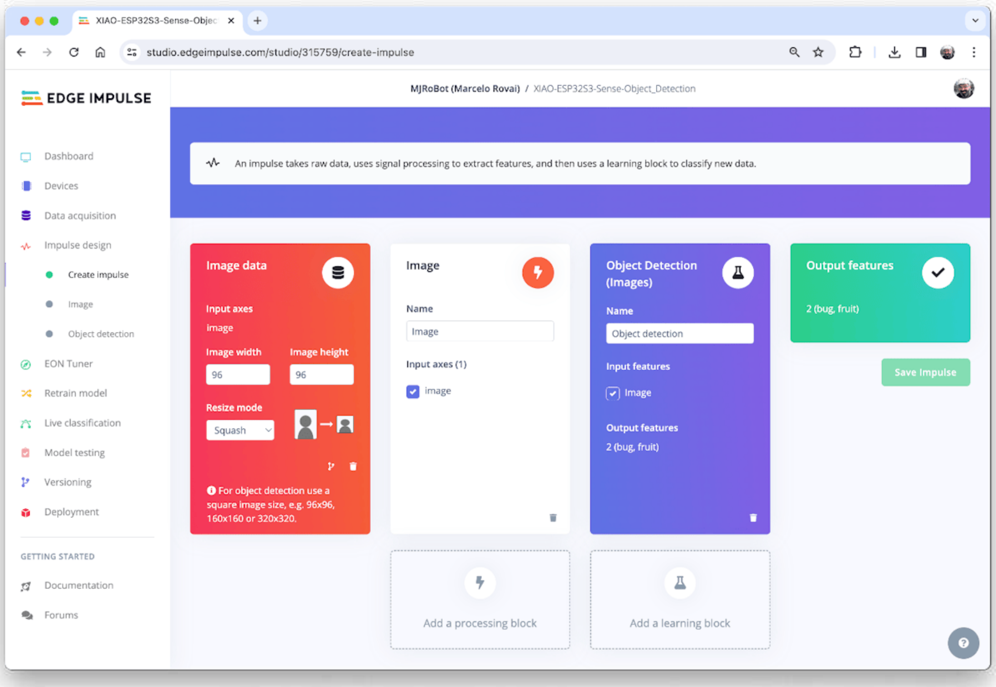


SAMPLE NAME	LABELS	ADDED
20231128150543	fruit, fruit	Today, 15:27:07
20231128150910	bug	Today, 15:27:07
20231128150938	bug, bug, fruit, fruit	Today, 15:27:07
20231128150625	fruit, fruit, fruit, fru...	
20231128151648	bug, bug, fruit, fruit	
20231128150736	bug, bug	
20231128150635	fruit	
20231128150437	fruit	
20231128150508	fruit, fruit, fruit	Today, 15:27:06
20231128150750	bug	Today, 15:27:05

The Impulse Design

In this phase, you should define how to:

- **Pre-processing** consists of resizing the individual images from 320×240 to 96×96 and squashing them (squared form, without cropping). Afterward, the images are converted from RGB to Grayscale.
- **Design a Model,** in this case, "Object Detection."



An impulse takes raw data, uses signal processing to extract features, and then uses a learning block to classify new data.

Image data

Input axes: image

Image width: 96

Image height: 96

Resize mode: Squash

For object detection use a square image size, e.g. 96x96, 160x160 or 320x320.

Image

Name: Image

Input axes (1): ✓ image

Object Detection (Images)

Name: Object detection

Input features: ✓ Image

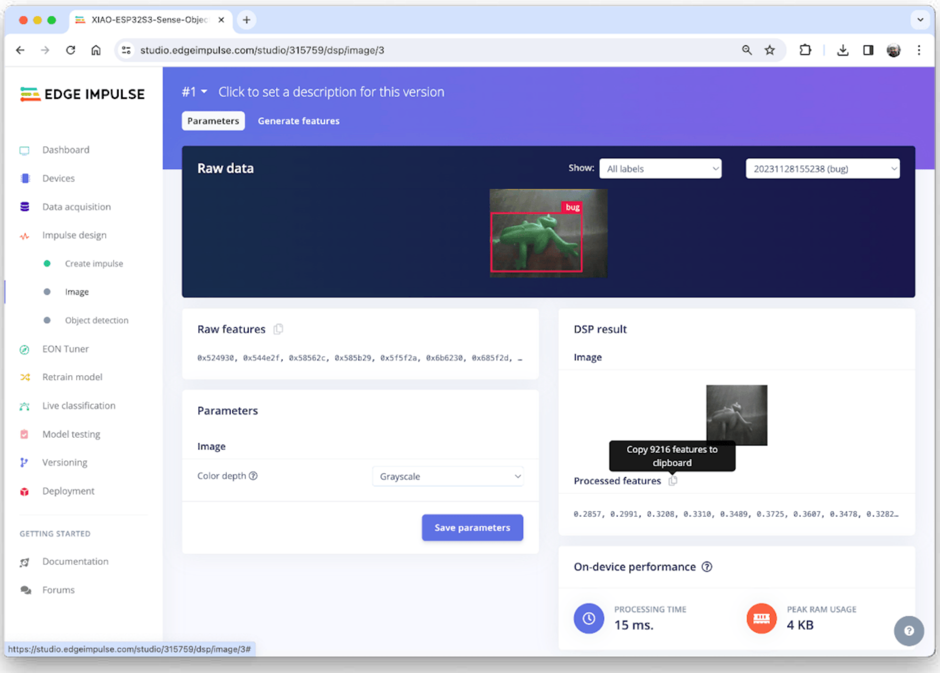
Output features: 2 (bug, fruit)

Output features

2 (bug, fruit)

Preprocessing all dataset

In this section, select **Color depth** as Grayscale, suitable for use with FOMO models and Save parameters.



Raw features

```
0x524930, 0x544e2f, 0x58562c, 0x585b29, 0x5f5f2a, 0x6b6230, 0x685f2d, …
```

Parameters

Image

Color depth: Grayscale

DSP result

Image

Processed features

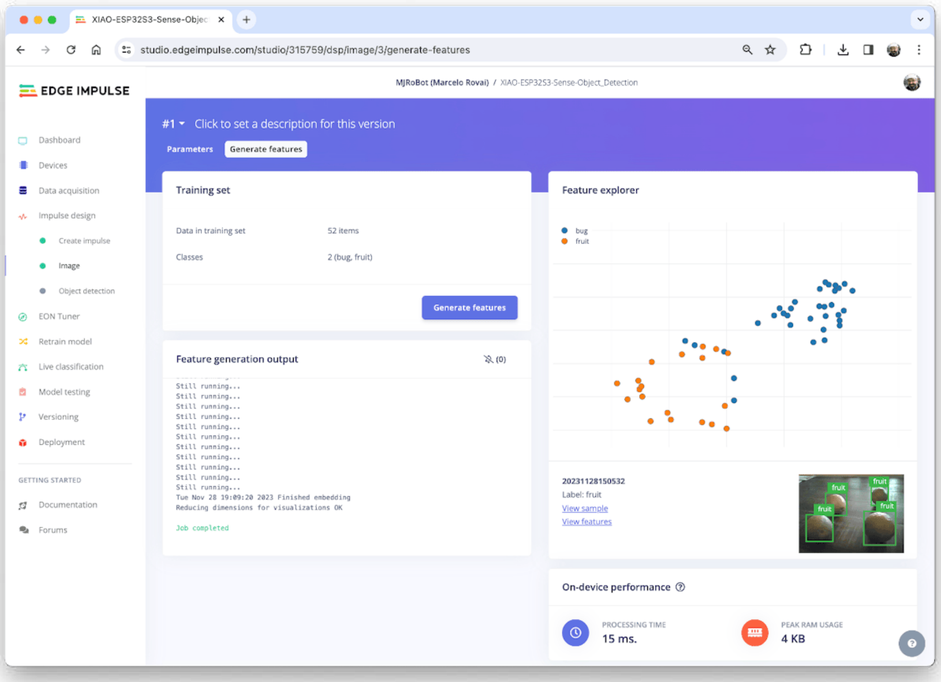
```
0.2857, 0.2991, 0.3208, 0.3310, 0.3489, 0.3725, 0.3607, 0.3478, 0.3282…
```

On-device performance

PROCESSING TIME: 15 ms.

PEAK RAM USAGE: 4 KB

The Studio moves automatically to the next section, Generate features, where all samples will be pre-processed, resulting in a dataset with individual $96 \times 96 \times 1$ images or 9,216 features.



Data in training set	52 items
Classes	2 (bug, fruit)

```
Still running...
Still running...
Still running...
Still running...
Still running...
Still running...
Still running...
Still running...
Still running...
Still running...
Still running...
Tue Nov 28 19:09:20 2023 Finished embedding
Reducing dimensions for visualizations OK

Job completed
```

The feature explorer shows that all samples evidence a good separation after the feature generation.

> Some samples seem to be in the wrong space, but clicking on them confirms the correct labeling.

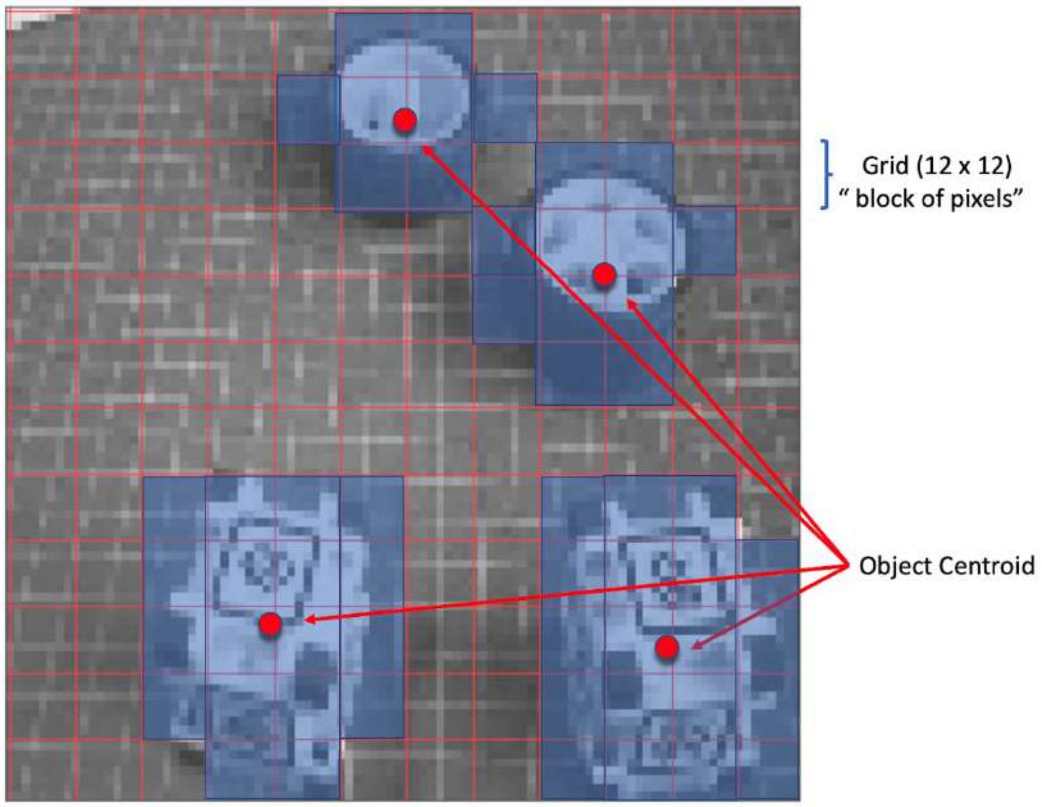
Model Design, Training, and Test

We will use FOMO, an object detection model based on MobileNetV2 (alpha 0.35) designed to coarsely segment an image into a grid of **background** vs **objects of interest** (here, *boxes* and *wheels*).

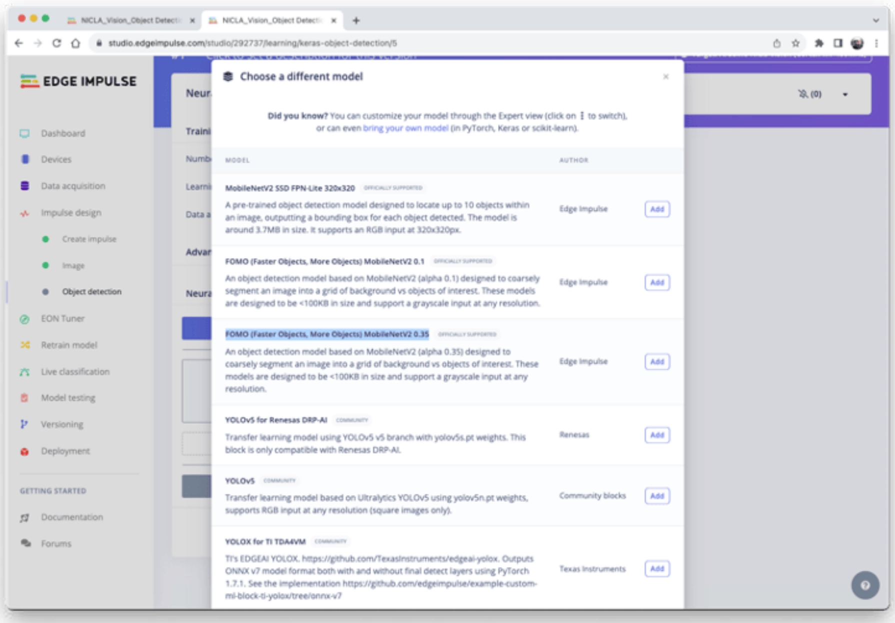
FOMO is an innovative machine learning model for object detection, which can use up to 30 times less energy and memory than traditional models like Mobilenet SSD and YOLOv5. FOMO can operate on microcontrollers with less than 200 KB of RAM. The main reason this is possible is that while other models calculate the object's size by drawing a square around it (bounding box), FOMO ignores the size of the image, providing only the information about where the object is located in the image through its centroid coordinates.

How FOMO works?

FOMO takes the image in grayscale and divides it into blocks of pixels using a factor of 8. For the input of 96×96, the grid would be 12×12 ($96/8 = 12$). Next, FOMO will run a classifier through each pixel block to calculate the probability that there is a box or a wheel in each of them and, subsequently, determine the regions that have the highest probability of containing the object (If a pixel block has no objects, it will be classified as *background*). From the overlap of the final region, the FOMO provides the coordinates (related to the image dimensions) of the centroid of this region.



For training, we should select a pre-trained model. Let's use the **FOMO (Faster Objects, More Objects) MobileNetV2 0.35.** This model uses around 250 KB of RAM and 80 KB of ROM (Flash), which suits well with our board.



Choose a different model

Did you know? You can customize your model through the Expert view (click on ⋮ to switch), or can even bring your own model (in PyTorch, Keras or scikit-learn).

MODEL	AUTHOR	
MobileNetV2 SSD FPN-Lite 320x320 OFFICIALLY SUPPORTED A pre-trained object detection model designed to locate up to 10 objects within an image, outputting a bounding box for each object detected. The model is around 3.7MB in size. It supports an RGB input at 320x320px.	Edge Impulse	Add
FOMO (Faster Objects, More Objects) MobileNetV2 0.1 OFFICIALLY SUPPORTED An object detection model based on MobileNetV2 (alpha 0.1) designed to coarsely segment an image into a grid of background vs objects of interest. These models are designed to be <100KB in size and support a grayscale input at any resolution.	Edge Impulse	Add
FOMO (Faster Objects, More Objects) MobileNetV2 0.35 OFFICIALLY SUPPORTED An object detection model based on MobileNetV2 (alpha 0.35) designed to coarsely segment an image into a grid of background vs objects of interest. These models are designed to be <100KB in size and support a grayscale input at any resolution.	Edge Impulse	Add
YOLOv5 for Renesas DRP-AI COMMUNITY Transfer learning model using YOLOv5 v5 branch with yolov5s.pt weights. This block is only compatible with Renesas DRP-AI.	Renesas	Add
YOLOv5 COMMUNITY Transfer learning model based on Ultralytics YOLOv5 using yolov5n.pt weights, supports RGB input at any resolution (square images only).	Community blocks	Add
YOLOX for TI TDA4VM COMMUNITY TI's EDGEAI YOLOX. https://github.com/TexasInstruments/edgeai-yolox. Outputs ONNX v7 model format both with and without final detect layers using PyTorch 1.7.1. See the implementation https://github.com/edgeimpulse/example-custom-ml-block-ti-yolox/tree/onnx-v7	Texas Instruments	Add

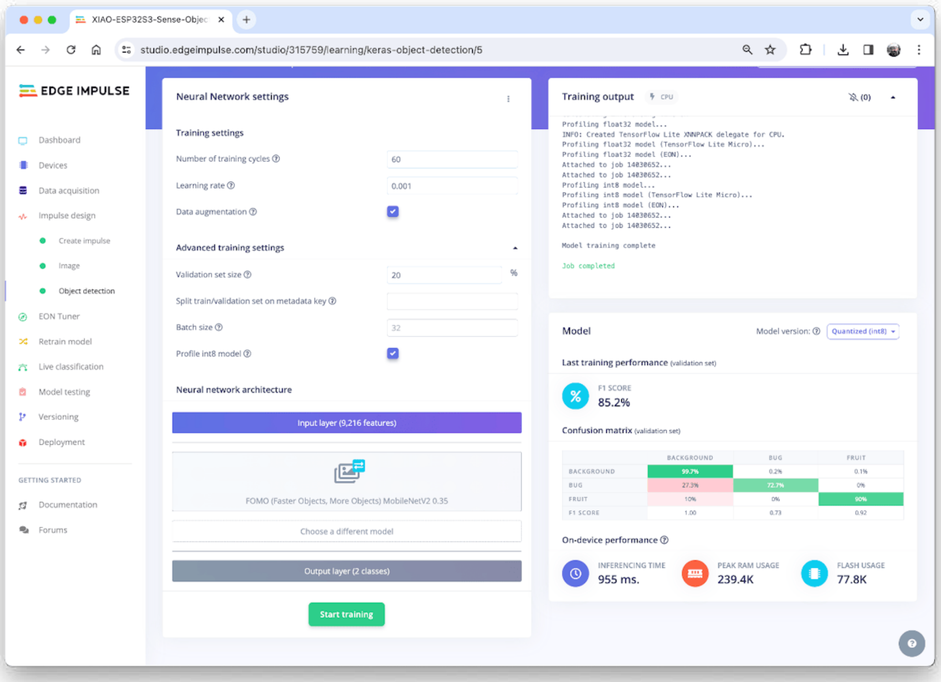
Regarding the training hyper-parameters, the model will be trained with:

- Epochs: 60
- Batch size: 32
- Learning Rate: 0.001.

For validation during training, 20% of the dataset (*validation_dataset*) will be spared. For the remaining 80% (*train_dataset*), we will apply Data Augmentation, which will randomly flip, change the size and brightness of the image, and crop them, artificially increasing the number of samples on the dataset for training.

As a result, the model ends with an overall F1 score of 85%, similar to the result when using the test data (83%).

> Note that FOMO automatically added a 3rd label background to the two previously defined (*box* and *wheel*).



Neural Network settings

Training settings

Number of training cycles	60
Learning rate	0.001
Data augmentation	✓

Advanced training settings

Validation set size	20 %
Split train/validation set on metadata key	
Batch size	32
Profile int8 model	✓

Neural network architecture

Input layer (9,216 features)

FOMO (Faster Objects, More Objects) MobileNetV2 0.35

Choose a different model

Output layer (2 classes)

Start training

Training output CPU

```
Profiling float32 model...
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
Profiling float32 model (TensorFlow Lite Micro)...
Profiling float32 model (EON)...
Attached to job 14030652...
Attached to job 14030652...
Profiling int8 model...
Profiling int8 model (TensorFlow Lite Micro)...
Profiling int8 model (EON)...
Attached to job 14030652...
Attached to job 14030652...

Model training complete

Job completed
```

Model

Model version: Quantized (int8)

Last training performance (validation set)

F1 SCORE 85.2%

Confusion matrix (validation set)

	BACKGROUND	BUG	FRUIT
BACKGROUND	99.7%	0.2%	0.1%
BUG	27.3%	72.7%	0%
FRUIT	10%	0%	90%
F1 SCORE	1.00	0.73	0.92

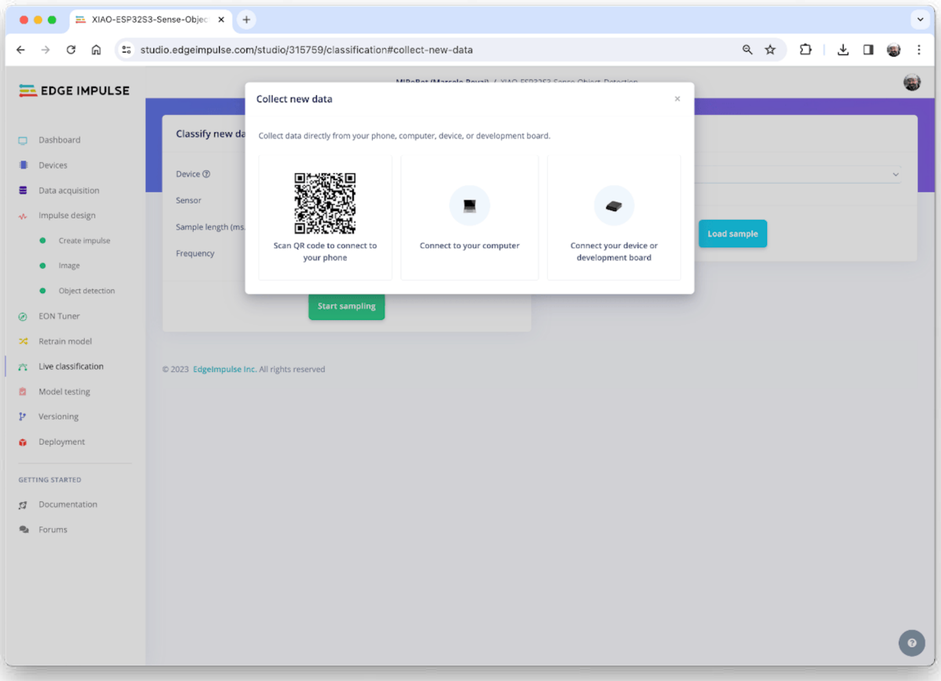
On-device performance

INFERENCING TIME	PEAK RAM USAGE	FLASH USAGE
955 ms.	239.4K	77.8K

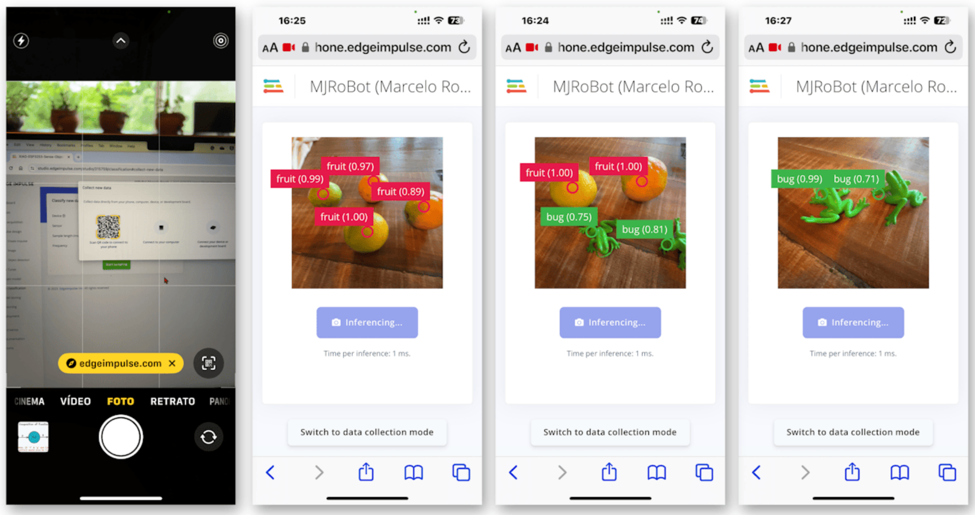
In object detection tasks, accuracy is generally not the primary evaluation metric. Object detection involves classifying objects and providing bounding boxes around them, making it a more complex problem than simple classification. The issue is that we do not have the bounding box, only the centroids. In short, using accuracy as a metric could be misleading and may not provide a complete understanding of how well the model is performing. Because of that, we will use the F1 score.

Test model with "Live Classification"

Once our model is trained, we can test it using the Live Classification tool. On the correspondent section, click on Connect a development board icon (a small MCU) and scan the QR code with your phone.



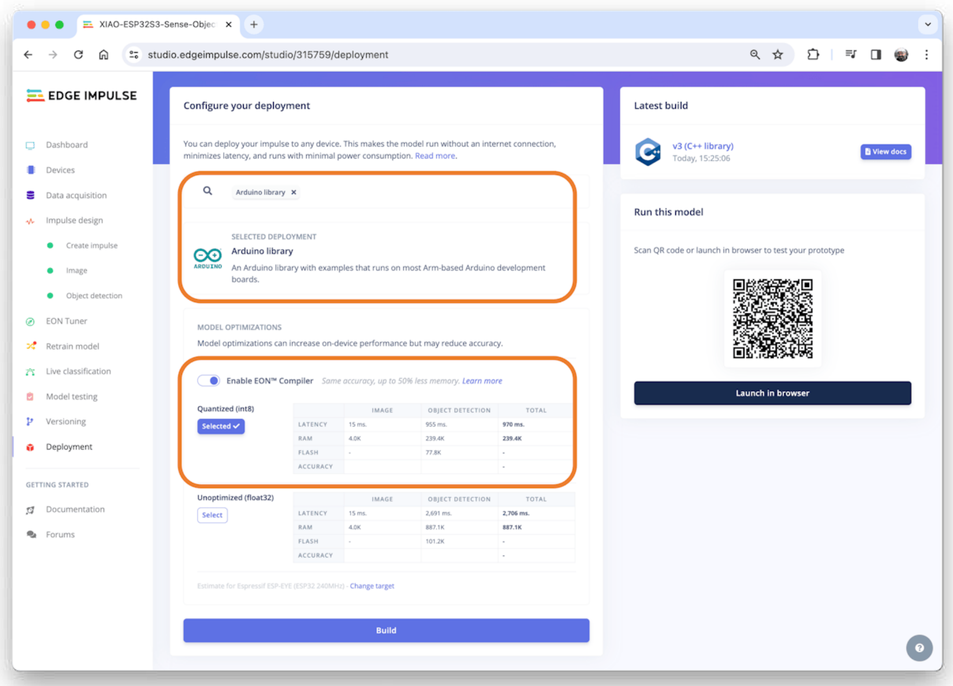
Once connected, you can use the smartphone to capture actual images to be tested by the trained model on Edge Impulse Studio.



One thing to be noted is that the model can produce false positives and negatives. This can be minimized by defining a proper Confidence Threshold (use the Three dots menu for the setup). Try with 0.8 or more.

Deploying the Model (Arduino IDE)

Select the Arduino Library and Quantized (int8) model, enable the EON Compiler on the Deploy Tab, and press [Build].



XIAO-ESP32S3-Sense-Objec

studio.edgeimpulse.com/studio/315759/deployment

EDGE IMPULSE

Dashboard
Devices
Data acquisition
Impulse design
Create impulse
Image
Object detection
EON Tuner
Retrain model
Live classification
Model testing
Versioning
Deployment

GETTING STARTED
Documentation
Forums

Configure your deployment

You can deploy your impulse to any device. This makes the model run without an internet connection, minimizes latency, and runs with minimal power consumption. Read more.

Arduino library ×

SELECTED DEPLOYMENT
Arduino library
An Arduino library with examples that runs on most Arm-based Arduino development boards.

MODEL OPTIMIZATIONS
Model optimizations can increase on-device performance but may reduce accuracy.

Enable EON™ Compiler *Same accuracy, up to 50% less memory. Learn more*

Quantized (int8) — Selected ✓

	IMAGE	OBJECT DETECTION	TOTAL
LATENCY	15 ms.	955 ms.	**970 ms.**
RAM	4.0K	239.4K	**239.4K**
FLASH	-	77.8K	-
ACCURACY			-

Unoptimized (float32) — Select

	IMAGE	OBJECT DETECTION	TOTAL
LATENCY	15 ms.	2,691 ms.	**2,706 ms.**
RAM	4.0K	887.1K	**887.1K**
FLASH	-	101.2K	-
ACCURACY			-

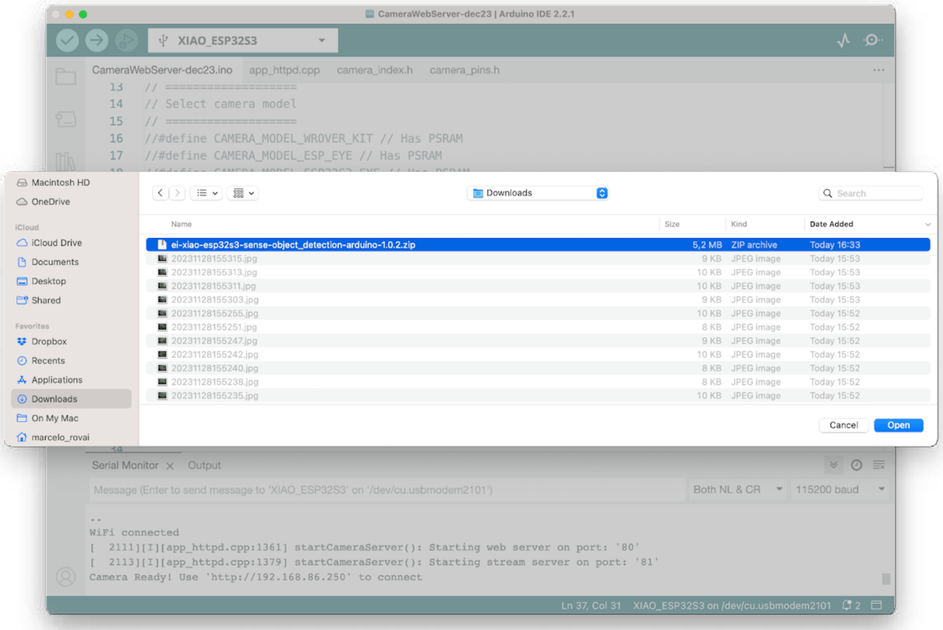
Estimate for Espressif ESP-EYE (ESP32 240MHz) - Change target

Build

Latest build
v3 (C++ library)
Today, 15:25:06
View docs

Run this model
Scan QR code or launch in browser to test your prototype
Launch in browser

Open your Arduino IDE, and under Sketch, go to Include Library and add.ZIP Library. Select the file you download from Edge Impulse Studio, and that's it!



```
// ===================
// Select camera model
// ===================
//#define CAMERA_MODEL_WROVER_KIT // Has PSRAM
//#define CAMERA_MODEL_ESP_EYE // Has PSRAM
```

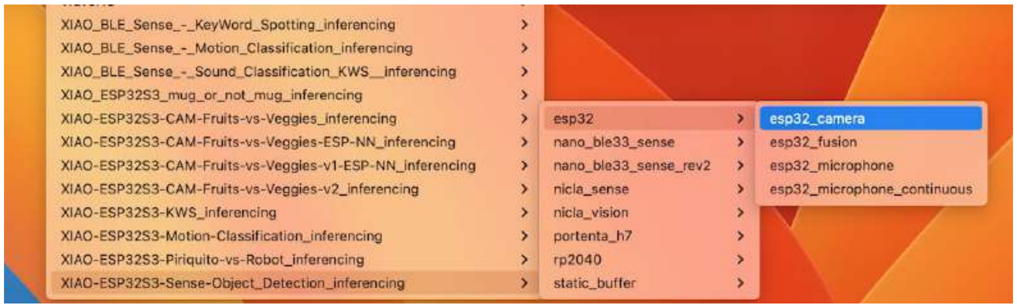
Downloads

Name	Size	Kind	Date Added
ei-xiao-esp32s3-sense-object_detection-arduino-1.0.2.zip	5,2 MB	ZIP archive	Today 16:33
20231128155315.jpg	9 KB	JPEG image	Today 15:53
20231128155313.jpg	10 KB	JPEG image	Today 15:53
20231128155311.jpg	10 KB	JPEG image	Today 15:53
20231128155303.jpg	9 KB	JPEG image	Today 15:53
20231128155255.jpg	10 KB	JPEG image	Today 15:52
20231128155251.jpg	8 KB	JPEG image	Today 15:52
20231128155247.jpg	9 KB	JPEG image	Today 15:52
20231128155242.jpg	10 KB	JPEG image	Today 15:52
20231128155240.jpg	8 KB	JPEG image	Today 15:52
20231128155238.jpg	8 KB	JPEG image	Today 15:52
20231128155235.jpg	10 KB	JPEG image	Today 15:52

Message (Enter to send message to 'XIAO_ESP32S3' on '/dev/cu.usbmodem2101')

```
..
WiFi connected
[  2111][I][app_httpd.cpp:1361] startCameraServer(): Starting web server on port: '80'
[  2113][I][app_httpd.cpp:1379] startCameraServer(): Starting stream server on port: '81'
Camera Ready! Use 'http://192.168.86.250' to connect
```

Under the Examples tab on Arduino IDE, you should find a sketch code (`esp32 > esp32_camera`) under your project name.



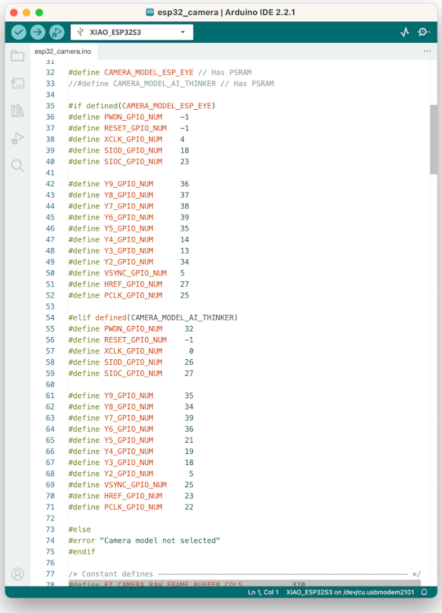
XIAO_BLE_Sense_-_KeyWord_Spotting_inferencing
XIAO_BLE_Sense_-_Motion_Classification_inferencing
XIAO_BLE_Sense_-_Sound_Classification_KWS__inferencing
XIAO_ESP32S3_mug_or_not_mug_inferencing
XIAO-ESP32S3-CAM-Fruits-vs-Veggies_inferencing
XIAO-ESP32S3-CAM-Fruits-vs-Veggies-ESP-NN_inferencing
XIAO-ESP32S3-CAM-Fruits-vs-Veggies-v1-ESP-NN_inferencing
XIAO-ESP32S3-CAM-Fruits-vs-Veggies-v2_inferencing
XIAO-ESP32S3-KWS_inferencing
XIAO-ESP32S3-Motion-Classification_inferencing
XIAO-ESP32S3-Piriquito-vs-Robot_inferencing
XIAO-ESP32S3-Sense-Object_Detection_inferencing
esp32
nano_ble33_sense
nano_ble33_sense_rev2
nicla_sense
nicla_vision
portenta_h7
rp2040
static_buffer
esp32_camera
esp32_fusion
esp32_microphone
esp32_microphone_continuous

You should change lines 32 to 75, which define the camera model and pins, using the data related to our model. Copy and paste the below lines, replacing the lines 32-75:

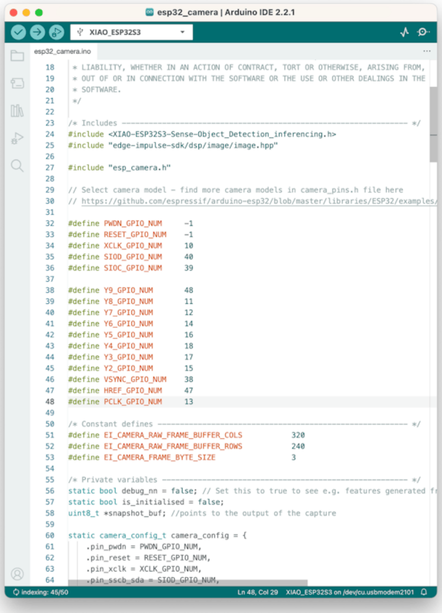
```
#define PWDN_GPIO_NUM     -1
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM     10
#define SIOD_GPIO_NUM     40
#define SIOC_GPIO_NUM     39
#define Y9_GPIO_NUM       48
#define Y8_GPIO_NUM       11
#define Y7_GPIO_NUM       12
```

```
#define Y6_GPIO_NUM       14
#define Y5_GPIO_NUM       16
#define Y4_GPIO_NUM       18
#define Y3_GPIO_NUM       17
#define Y2_GPIO_NUM       15
#define VSYNC_GPIO_NUM    38
#define HREF_GPIO_NUM     47
#define PCLK_GPIO_NUM     13
```

Here you can see the resulting code:



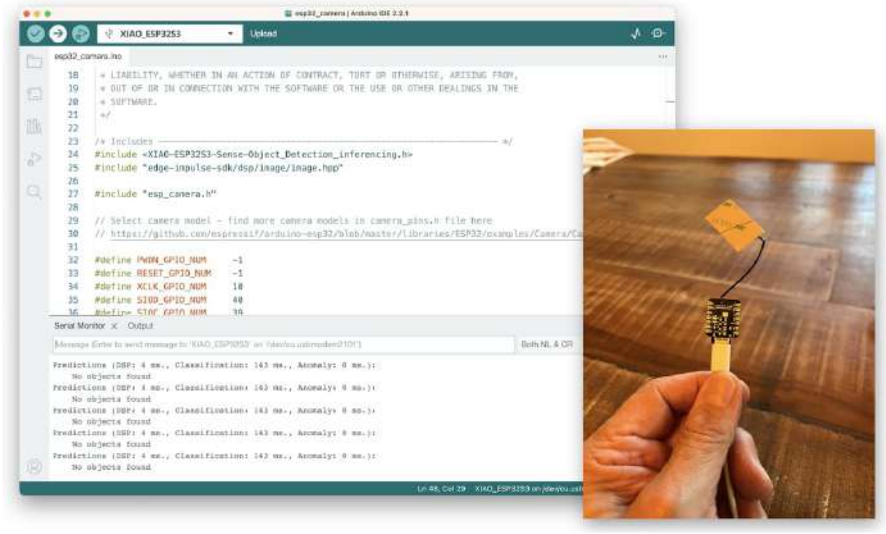
```
#define CAMERA_MODEL_ESP_EYE // Has PSRAM
//#define CAMERA_MODEL_AI_THINKER // Has PSRAM

#if defined(CAMERA_MODEL_ESP_EYE)
#define PWDN_GPIO_NUM    -1
#define RESET_GPIO_NUM   -1
#define XCLK_GPIO_NUM    4
#define SIOD_GPIO_NUM    18
#define SIOC_GPIO_NUM    23

#define Y9_GPIO_NUM      36
#define Y8_GPIO_NUM      37
#define Y7_GPIO_NUM      38
#define Y6_GPIO_NUM      39
#define Y5_GPIO_NUM      35
#define Y4_GPIO_NUM      14
#define Y3_GPIO_NUM      13
#define Y2_GPIO_NUM      34
#define VSYNC_GPIO_NUM   5
#define HREF_GPIO_NUM    27
#define PCLK_GPIO_NUM    25

#elif defined(CAMERA_MODEL_AI_THINKER)
#define PWDN_GPIO_NUM     32
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27

#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22

#else
#error "Camera model not selected"
#endif

/* Constant defines -------------------------------------------------------- */
#define EI_CAMERA_RAW_FRAME_BUFFER_COLS           320
```



```
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 */

/* Includes ---------------------------------------------------------------- */
#include <XIAO-ESP32S3-Sense-Object_Detection_inferencing.h>
#include "edge-impulse-sdk/dsp/image/image.hpp"

#include "esp_camera.h"

// Select camera model - find more camera models in camera_pins.h file here
// https://github.com/espressif/arduino-esp32/blob/master/libraries/ESP32/examples/

#define PWDN_GPIO_NUM     -1
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM     10
#define SIOD_GPIO_NUM     40
#define SIOC_GPIO_NUM     39

#define Y9_GPIO_NUM       48
#define Y8_GPIO_NUM       11
#define Y7_GPIO_NUM       12
#define Y6_GPIO_NUM       14
#define Y5_GPIO_NUM       16
#define Y4_GPIO_NUM       18
#define Y3_GPIO_NUM       17
#define Y2_GPIO_NUM       15
#define VSYNC_GPIO_NUM    38
#define HREF_GPIO_NUM     47
#define PCLK_GPIO_NUM     13

/* Constant defines -------------------------------------------------------- */
#define EI_CAMERA_RAW_FRAME_BUFFER_COLS           320
#define EI_CAMERA_RAW_FRAME_BUFFER_ROWS           240
#define EI_CAMERA_FRAME_BYTE_SIZE                 3

/* Private variables ------------------------------------------------------- */
static bool debug_nn = false; // Set this to true to see e.g. features generated fr
static bool is_initialised = false;
uint8_t *snapshot_buf; //points to the output of the capture

static camera_config_t camera_config = {
    .pin_pwdn = PWDN_GPIO_NUM,
    .pin_reset = RESET_GPIO_NUM,
    .pin_xclk = XCLK_GPIO_NUM,
    .pin_sscb_sda = SIOD_GPIO_NUM,
```

Upload the code to your XIAO ESP32S3 Sense, and you should be OK to start detecting fruits and bugs. You can check the result on Serial Monitor.

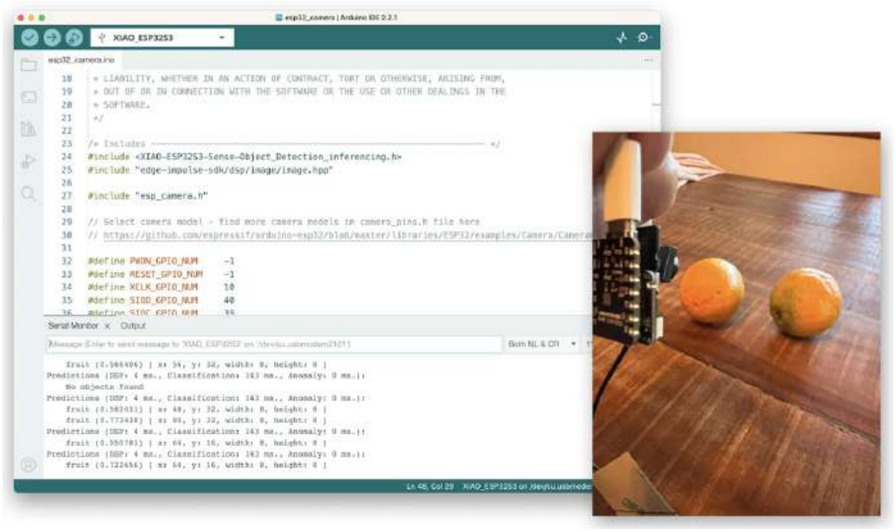
Background



```
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 */

/* Includes ---------------------------------------------------------------- */
#include <XIAO-ESP32S3-Sense-Object_Detection_inferencing.h>
#include "edge-impulse-sdk/dsp/image/image.hpp"

#include "esp_camera.h"

// Select camera model - find more camera models in camera_pins.h file here
// https://github.com/espressif/arduino-esp32/blob/master/libraries/ESP32/examples/Camera/Ca

#define PWDN_GPIO_NUM     -1
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM     10
#define SIOD_GPIO_NUM     40
#define SIOC_GPIO_NUM     39
```

```
Predictions (DSP: 4 ms., Classification: 143 ms., Anomaly: 0 ms.):
    No objects found
Predictions (DSP: 4 ms., Classification: 143 ms., Anomaly: 0 ms.):
    No objects found
Predictions (DSP: 4 ms., Classification: 143 ms., Anomaly: 0 ms.):
    No objects found
Predictions (DSP: 4 ms., Classification: 143 ms., Anomaly: 0 ms.):
    No objects found
Predictions (DSP: 4 ms., Classification: 143 ms., Anomaly: 0 ms.):
    No objects found
```

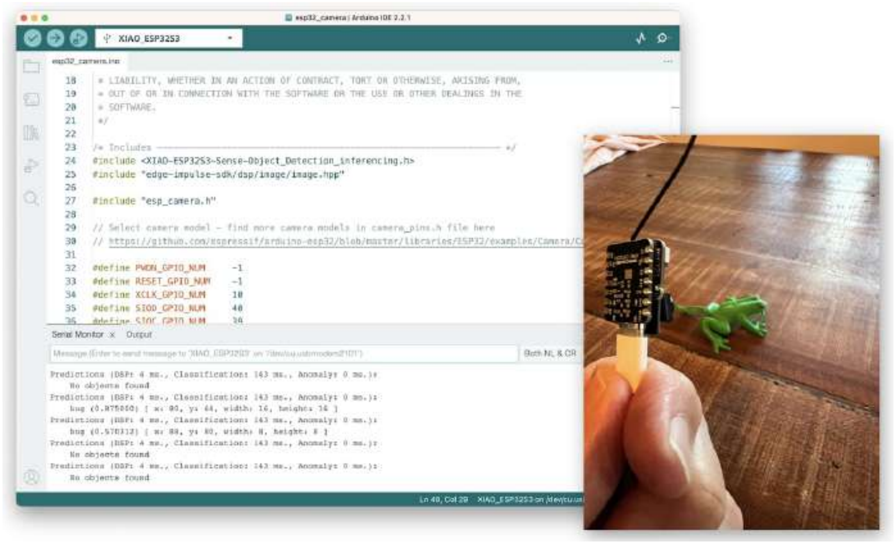
Fruits



```
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 */

/* Includes ---------------------------------------------------------------- */
#include <XIAO-ESP32S3-Sense-Object_Detection_inferencing.h>
#include "edge-impulse-sdk/dsp/image/image.hpp"

#include "esp_camera.h"

// Select camera model - find more camera models in camera_pins.h file here
// https://github.com/espressif/arduino-esp32/blob/master/libraries/ESP32/examples/Camera/Camera

#define PWDN_GPIO_NUM     -1
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM     10
#define SIOD_GPIO_NUM     40
#define SIOC_GPIO_NUM     39
```

```
    fruit (0.566406) [ x: 56, y: 32, width: 8, height: 8 ]
Predictions (DSP: 4 ms., Classification: 143 ms., Anomaly: 0 ms.):
    No objects found
Predictions (DSP: 4 ms., Classification: 143 ms., Anomaly: 0 ms.):
    fruit (0.582031) [ x: 48, y: 32, width: 8, height: 8 ]
    fruit (0.773438) [ x: 80, y: 32, width: 8, height: 8 ]
Predictions (DSP: 4 ms., Classification: 143 ms., Anomaly: 0 ms.):
    fruit (0.550781) [ x: 64, y: 16, width: 8, height: 8 ]
Predictions (DSP: 4 ms., Classification: 143 ms., Anomaly: 0 ms.):
    fruit (0.722656) [ x: 64, y: 16, width: 8, height: 8 ]
```

Bugs


esp32_camera | Arduino IDE 2.2.1
XIAO_ESP32S3
esp32_camera.ino
Serial Monitor
Output
Message (Enter to send message to 'XIAO_ESP32S3' on '/dev/cu.usbmodem2101')
Both NL & CR
Ln 40, Col 29
XIAO_ESP32S3 on /dev/cu.us

```
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 */

/* Includes ---------------------------------------------------------------- */
#include <XIAO-ESP32S3-Sense-Object_Detection_inferencing.h>
#include "edge-impulse-sdk/dsp/image/image.hpp"

#include "esp_camera.h"

// Select camera model - find more camera models in camera_pins.h file here
// https://github.com/espressif/arduino-esp32/blob/master/libraries/ESP32/examples/Camera/C

#define PWDN_GPIO_NUM     -1
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM     10
#define SIOD_GPIO_NUM     40
#define SIOC_GPIO_NUM     39
```

```
Predictions (DSP: 4 ms., Classification: 143 ms., Anomaly: 0 ms.):
    No objects found
Predictions (DSP: 4 ms., Classification: 143 ms., Anomaly: 0 ms.):
    bug (0.875000) [ x: 80, y: 44, width: 16, height: 16 ]
Predictions (DSP: 4 ms., Classification: 143 ms., Anomaly: 0 ms.):
    bug (0.570312) [ x: 88, y: 80, width: 8, height: 8 ]
Predictions (DSP: 4 ms., Classification: 143 ms., Anomaly: 0 ms.):
    No objects found
Predictions (DSP: 4 ms., Classification: 143 ms., Anomaly: 0 ms.):
    No objects found
```

Note that the model latency is 143 ms, and the frame rate per second is around 7 fps (similar to what we got with the Image Classification project). This happens because FOMO is cleverly built over a CNN model, not with an object detection model like the SSD MobileNet. For example, when running a MobileNetV2 SSD FPN-Lite 320 × 320 model on a Raspberry Pi 4, the latency is around five times higher (around 1.5 fps).

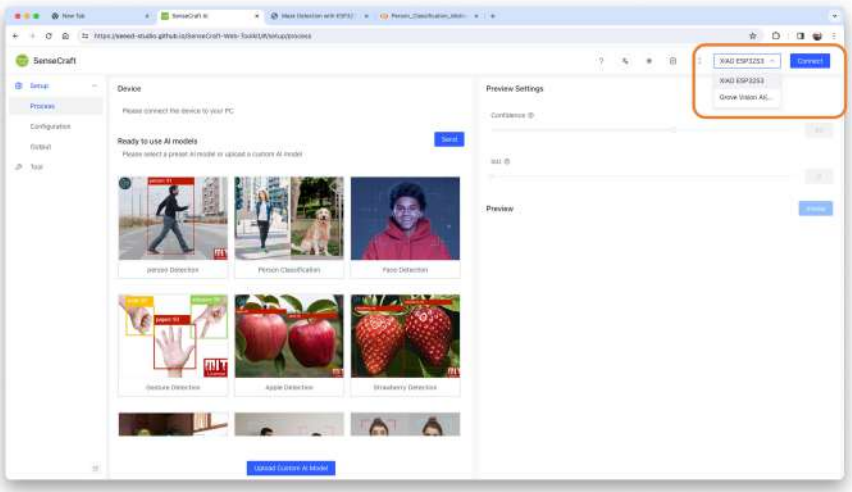
Deploying the Model (SenseCraft-Web-Toolkit)

As discussed in the Image Classification chapter, verifying inference with Image models on Arduino IDE is very challenging because we can not see what the camera focuses on. Again, let's use the **SenseCraft-Web Toolkit**.

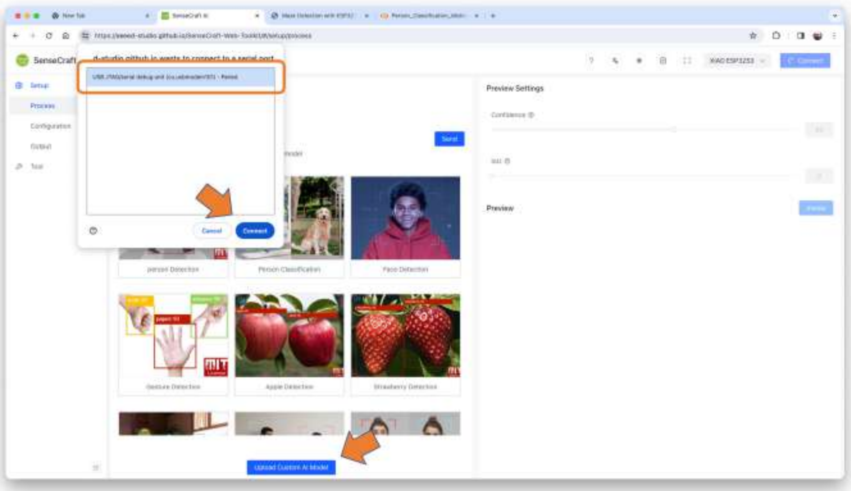
Follow the following steps to start the SenseCraft-Web-Toolkit:

1. Open the SenseCraft-Web-Toolkit website.
2. Connect the XIAO to your computer:

- Having the XIAO connected, select it as below:



- Select the device/Port and press `[Connect]`:



You can try several Computer Vision models previously uploaded by Seeed Studio. Try them and have fun!

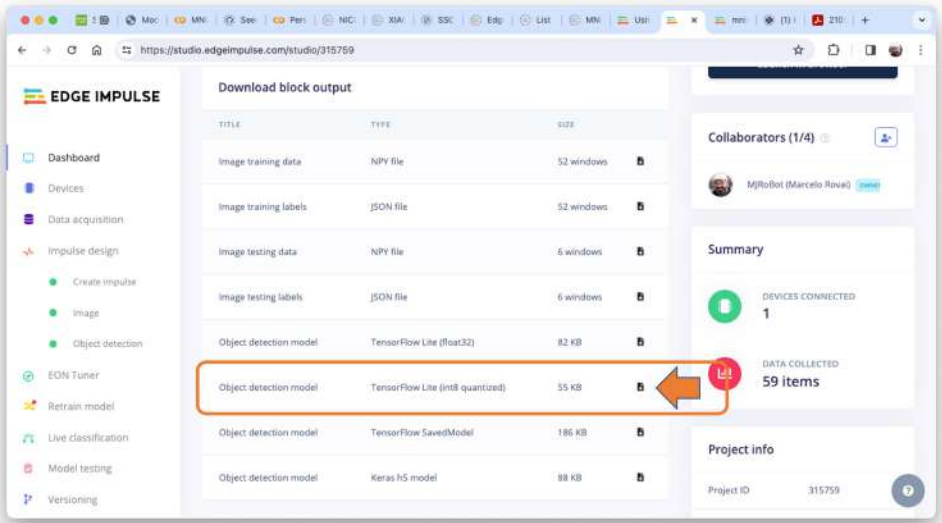
In our case, we will use the blue button at the bottom of the page: `[Upload Custom AI Model]`.

But first, we must download from Edge Impulse Studio our **quantized .tflite** model.

3. Go to your project at Edge Impulse Studio, or clone this one:

- XIAO-ESP32S3-CAM-Fruits-vs-Veggies-v1-ESP-NN

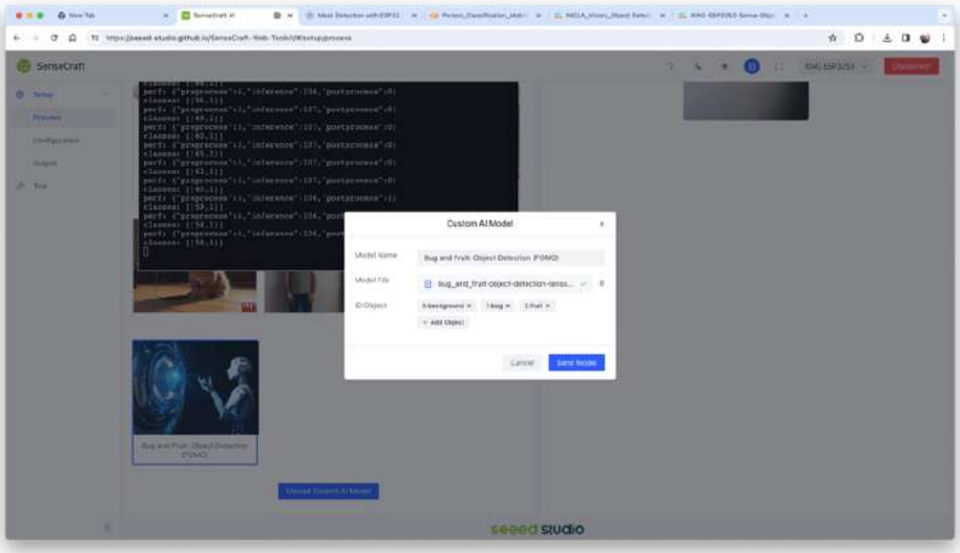
4. On `Dashboard`, download the model ("block output"): `Object Detection model - TensorFlow Lite (int8 quantized)`



Download block output

TITLE	TYPE	SIZE
Image training data	NPY file	52 windows
Image training labels	JSON file	52 windows
Image testing data	NPY file	6 windows
Image testing labels	JSON file	6 windows
Object detection model	TensorFlow Lite (float32)	82 KB
Object detection model	TensorFlow Lite (int8 quantized)	55 KB
Object detection model	TensorFlow SavedModel	186 KB
Object detection model	Keras h5 model	88 KB

5. On SenseCraft-Web-Toolkit, use the blue button at the bottom of the page: `[Upload Custom AI Model]`. A window will pop up. Enter the Model file that you downloaded to your computer from Edge Impulse Studio, choose a Model Name, and enter with labels (ID: Object):



Custom AI Model

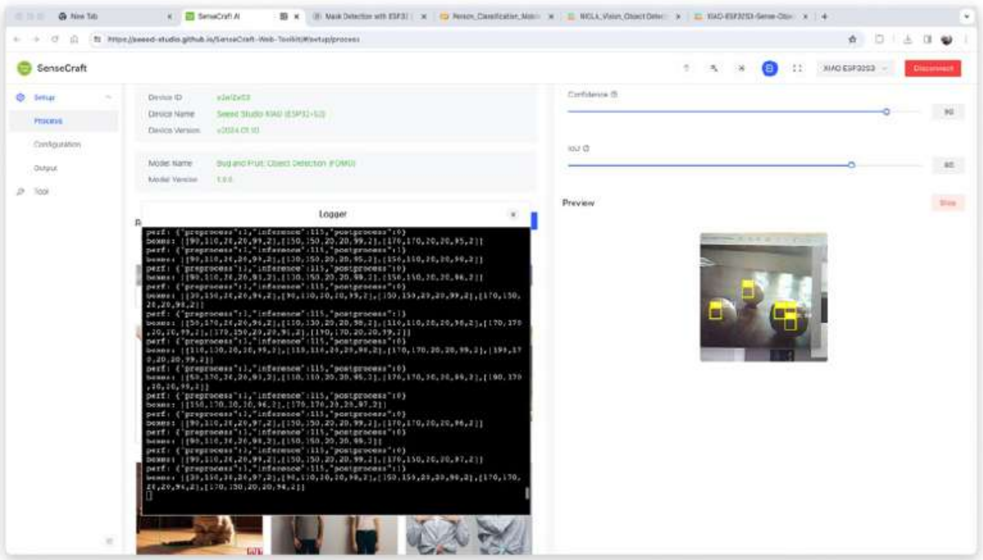
Model Name: Bug and Fruit Object Detection (FOMO)

Model File: bug_and_fruit-object-detection-tenso...

ID:Object: 0:background 1:bug 2:fruit

Note that you should use the labels trained on EI Studio and enter them in alphabetic order (in our case, background, bug, fruit).

After a few seconds (or minutes), the model will be uploaded to your device, and the camera image will appear in real-time on the Preview Sector:

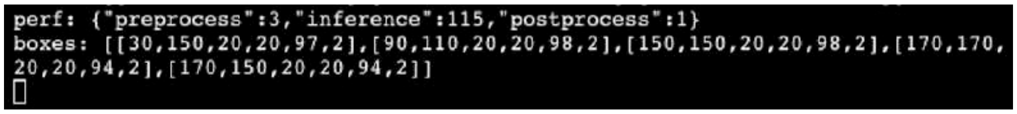


Device ID
Device Name Seeed Studio XIAO (ESP32-S3)
Device Version
Model Name Bug and Fruit: Object Detection (FOMO)
Model Version 1.0.0

```
perf: {"preprocess":1,"inference":115,"postprocess":0}
boxes: [[90,110,20,20,99,2],[150,150,20,20,99,2],[170,170,20,20,95,2]]
perf: {"preprocess":1,"inference":115,"postprocess":1}
boxes: [[90,110,20,20,99,2],[130,150,20,20,95,2],[150,110,20,20,98,2]]
perf: {"preprocess":1,"inference":115,"postprocess":0}
boxes: [[90,110,20,20,91,2],[130,150,20,20,99,2],[150,150,20,20,98,2]]
perf: {"preprocess":1,"inference":115,"postprocess":0}
boxes: [[30,150,20,20,94,2],[90,110,20,20,99,2],[150,150,20,20,99,2],[170,150,
20,20,98,2]]
perf: {"preprocess":1,"inference":115,"postprocess":1}
boxes: [[50,170,20,20,96,2],[110,130,20,20,98,2],[110,110,20,20,98,2],[170,170
,20,20,99,2],[170,150,20,20,91,2],[190,170,20,20,99,2]]
perf: {"preprocess":1,"inference":115,"postprocess":0}
boxes: [[110,130,20,20,99,2],[110,110,20,20,98,2],[170,170,20,20,99,2],[190,17
0,20,20,99,2]]
perf: {"preprocess":1,"inference":115,"postprocess":0}
boxes: [[50,170,20,20,91,2],[110,110,20,20,95,2],[170,170,20,20,99,2],[190,170
,20,20,99,2]]
perf: {"preprocess":1,"inference":115,"postprocess":0}
boxes: [[150,170,20,20,96,2],[170,170,20,20,97,2]]
perf: {"preprocess":1,"inference":115,"postprocess":0}
boxes: [[90,110,20,20,97,2],[150,150,20,20,99,2],[170,170,20,20,96,2]]
perf: {"preprocess":1,"inference":115,"postprocess":0}
boxes: [[90,110,20,20,98,2],[150,150,20,20,98,2]]
perf: {"preprocess":1,"inference":115,"postprocess":0}
boxes: [[90,110,20,20,99,2],[150,150,20,20,99,2],[170,150,20,20,97,2]]
perf: {"preprocess":1,"inference":115,"postprocess":1}
boxes: [[30,150,20,20,97,2],[90,110,20,20,98,2],[150,150,20,20,98,2],[170,170,
20,20,94,2],[170,150,20,20,94,2]]
```

The detected objects will be marked (the centroid). You can select the Confidence of your inference cursor `Confidence` and `IoU`, which is used to assess the accuracy of predicted bounding boxes compared to truth bounding boxes.

Clicking on the top button (Device Log), you can open a Serial Monitor to follow the inference, as we did with the Arduino IDE.



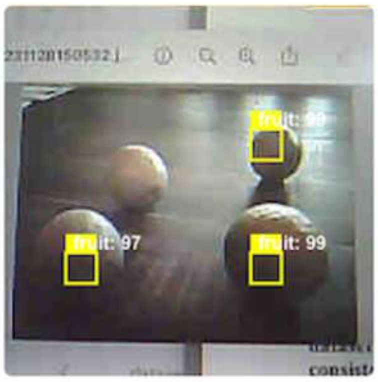
```
perf: {"preprocess":3,"inference":115,"postprocess":1}
boxes: [[30,150,20,20,97,2],[90,110,20,20,98,2],[150,150,20,20,98,2],[170,170,
20,20,94,2],[170,150,20,20,94,2]]
```

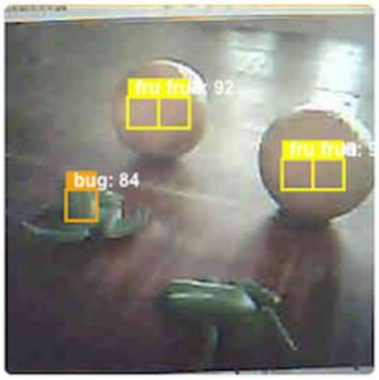
On Device Log, you will get information as:

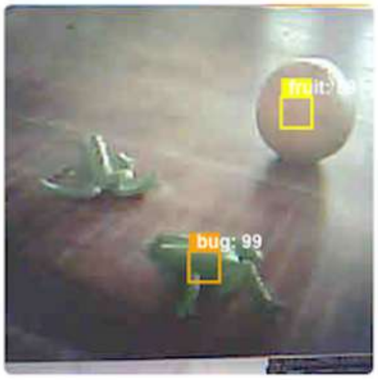
- Preprocess time (image capture and Crop): 3 ms,
- Inference time (model latency): 115 ms,
- Postprocess time (display of the image and marking objects): 1 ms.
- Output tensor (boxes), for example, one of the boxes: [[30,150, 20, 20,97, 2]]; where 30,150, 20, 20 are the coordinates of the box (around the centroid); 97 is the inference result, and 2 is the class (in this case 2: fruit).

  Note that in the above example, we got 5 boxes because none of the fruits got 3 centroids. One solution will be post-processing, where we can aggregate close centroids in one.

Here are other screenshots:







Summary

FOMO is a significant leap in the image processing space, as Louis Moreau and Mat Kelcey put it during its launch in 2022:

> FOMO is a ground-breaking algorithm that brings real-time object detection, tracking, and counting to microcontrollers for the first time.

Multiple possibilities exist for exploring object detection (and, more precisely, counting them) on embedded devices.

Resources

- Edge Impulse Project

Keyword Spotting (KWS)

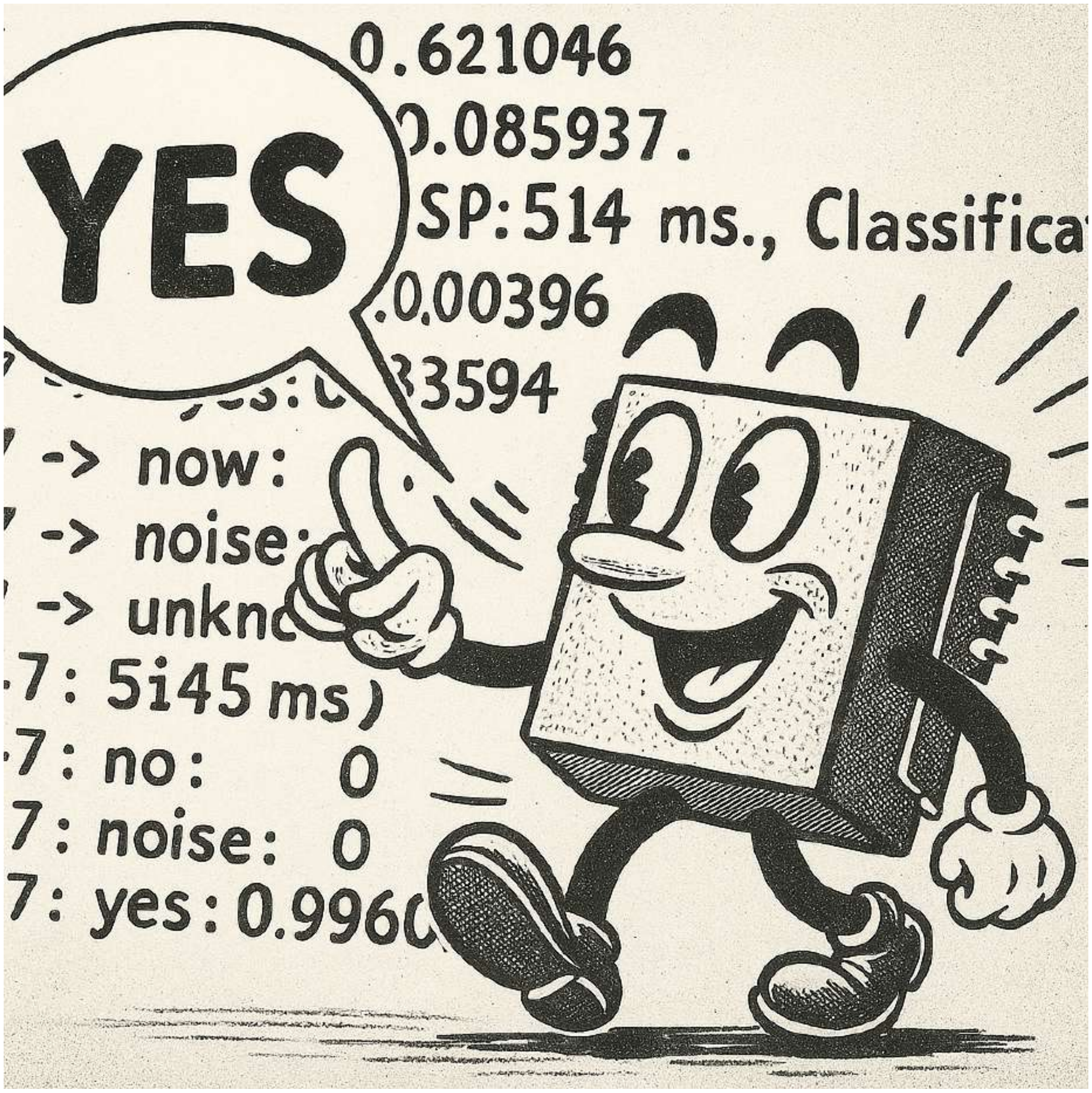


Figure 5: *DALL · E prompt - 1950s style cartoon illustration based on a real image by Marcelo Rovai*

Overview

Keyword Spotting (KWS) is integral to many voice recognition systems, enabling devices to respond to specific words or phrases. While this technology underpins popular devices like Google Assistant or Amazon Alexa, it's equally applicable and achievable on smaller, low-power devices. This lab will guide you through implementing a KWS system using TinyML on the XIAO ESP32S3 microcontroller board.

The XIAO ESP32S3, equipped with Espressif's ESP32-S3 chip, is a compact and potent microcontroller offering a dual-core Xtensa LX7 processor, integrated Wi-Fi, and Bluetooth. Its balance of computational power, energy efficiency, and versatile connectivity makes it a fantastic platform for TinyML applications. Also, with its expansion board, we will have access to the "sense" part of the device, which has a camera, an SD card slot, and a **digital microphone**. The integrated microphone and the SD card will be essential in this project.

We will use the Edge Impulse Studio, a powerful, user-friendly platform that simplifies creating and deploying machine learning models onto edge devices. We'll train a KWS model step-by-step, optimizing and deploying it onto the XIAO ESP32S3 Sense.

Our model will be designed to recognize keywords that can trigger device wake-up or specific actions (in the case of "YES"), bringing your projects to life with voice-activated commands.

Leveraging our experience with TensorFlow Lite for Microcontrollers (the engine "under the hood" on the EI Studio), we'll create a KWS system capable of real-time machine learning on the device.

As we progress through the lab, we'll break down each process stage – from data collection and preparation to model training and deployment – to provide a comprehensive understanding of implementing a KWS system on a microcontroller.

Learning Objectives

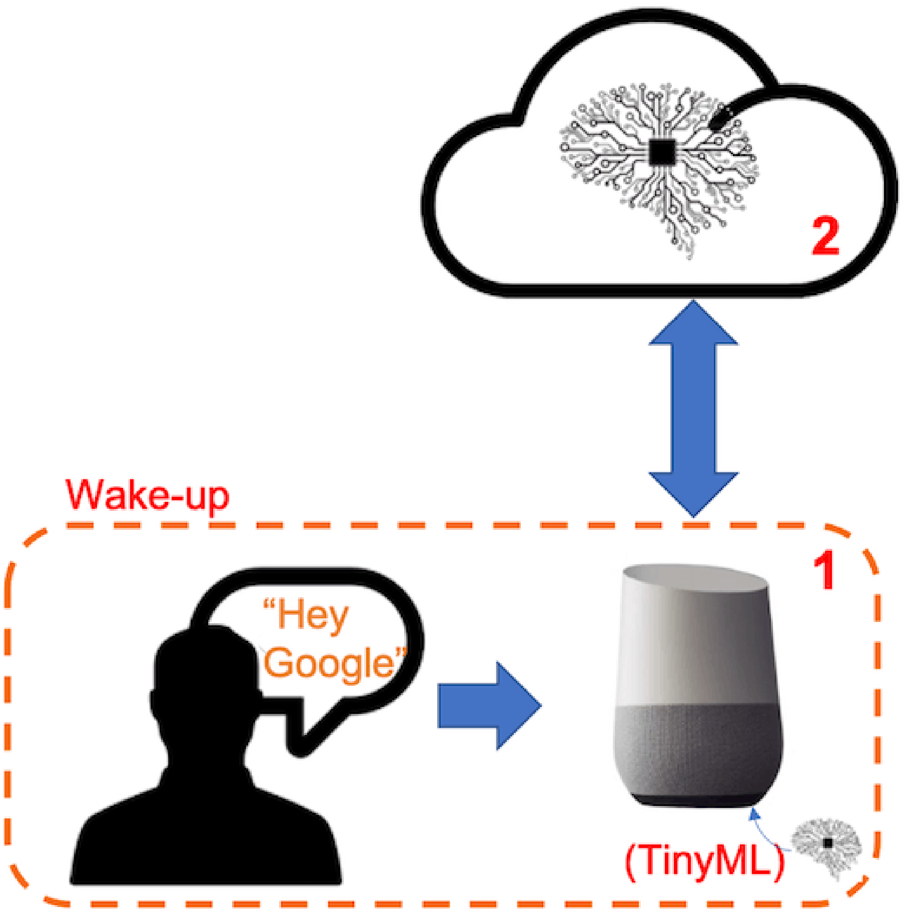
- **Understand Voice Assistant Architecture** including cascaded detection systems and the role of edge-based keyword spotting as the first stage of voice processing pipelines
- **Master Audio Data Collection Techniques** using both offline methods (XIAO ESP32S3 microphone with SD card storage) and online methods (smartphone integration with Edge Impulse Studio)
- **Implement Digital Signal Processing for Audio** including I2S protocol fundamentals, audio sampling at 16kHz/16-bit, and conversion between time-domain audio signals and frequency-domain features using MFCC
- **Train Convolutional Neural Networks for Audio Classification** using transfer learning techniques, data augmentation strategies, and model optimization for four-class classification (YES, NO, NOISE, UNKNOWN)

- **Deploy Optimized Models on Microcontrollers** through quantization (INT8), memory management with PSRAM, and real-time inference optimization for embedded systems
- **Develop Complete Post-Processing Pipelines** including confidence thresholding, GPIO control for external devices, OLED display integration, and creating standalone AI sensor systems
- **Compare Development Workflows** between no-code platforms (Edge Impulse Studio) and traditional embedded programming (Arduino IDE) for TinyML applications

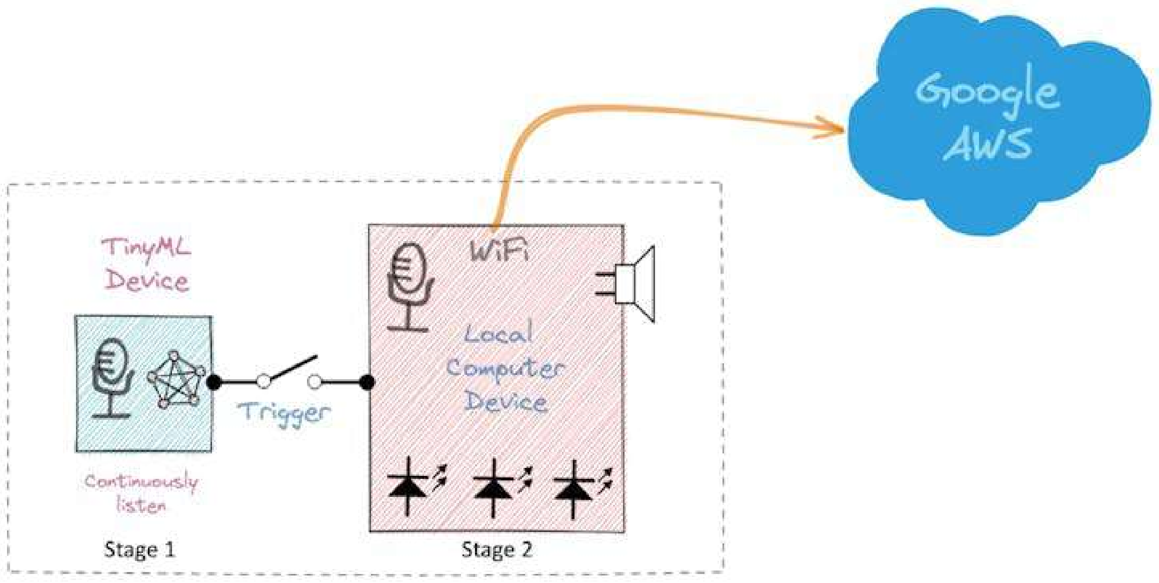
The KWS Project

How does a voice assistant work?

Keyword Spotting (KWS) is critical to many voice assistants, enabling devices to respond to specific words or phrases. To start, it is essential to realize that Voice Assistants on the market, like Google Home or Amazon Echo-Dot, only react to humans when they are "waked up" by particular keywords such as " Hey Google" on the first one and "Alexa" on the second.



In other words, recognizing voice commands is based on a multi-stage model or Cascade Detection.



Stage 1: A smaller microprocessor inside the Echo Dot or Google Home **continuously** listens to the sound, waiting for the keyword to be spotted. For such detection, a TinyML model at the edge is used (KWS application).

Stage 2: Only when triggered by the KWS application on Stage 1 is the data sent to the cloud and processed on a larger model.

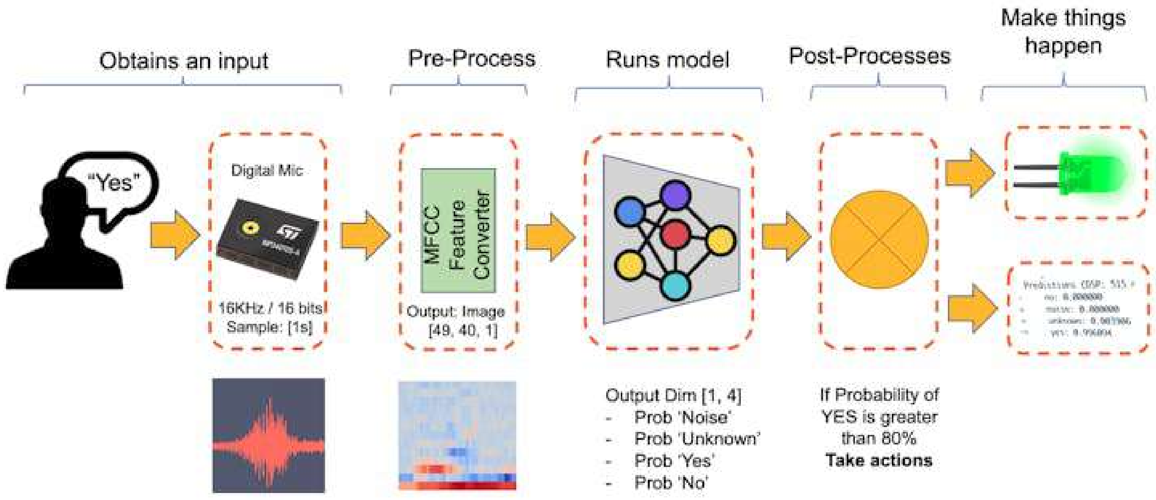
The video shows an example in which I emulate a Google Assistant on a Raspberry Pi (Stage 2), using an Arduino Nano 33 BLE as the tinyML device (Stage 1).

> If you want to go deeper into the full project, please see my tutorial: Building an Intelligent Voice Assistant From Scratch.

In this lab, we will focus on Stage 1 (KWS or Keyword Spotting), where we will use the XIAO ESP2S3 Sense, which has a digital microphone for spotting the keyword.

The Inference Pipeline

The diagram below will give an idea of how the final KWS application should work (during inference):



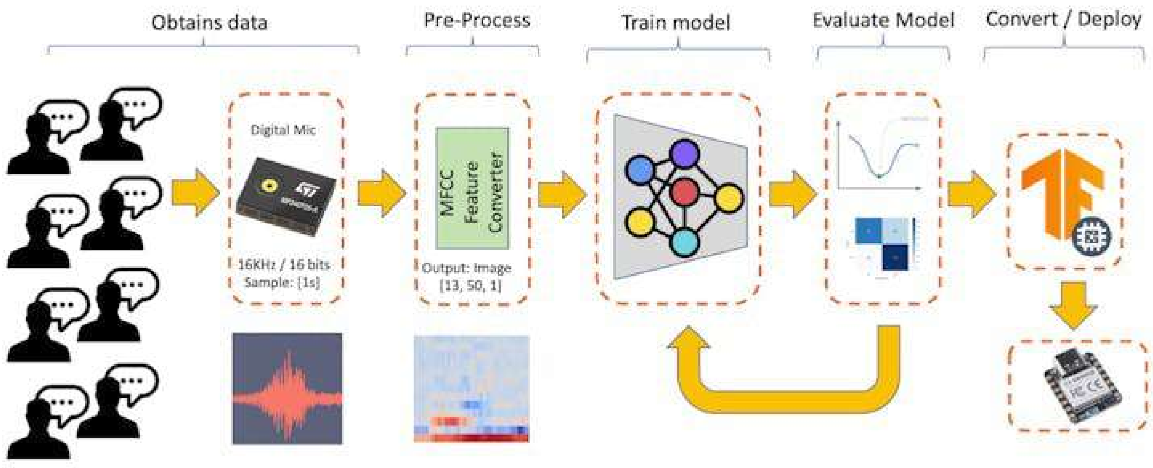
Our KWS application will recognize four classes of sound:

- **YES** (Keyword 1)
- **NO** (Keyword 2)
- **NOISE** (no keywords spoken, only background noise is present)
- **UNKNOWN** (a mix of different words than YES and NO)

  Optionally for real-world projects, it is always advised to include different words than keywords, such as "Noise" (or Background) and "Unknown."

The Machine Learning workflow

The main component of the KWS application is its model. So, we must train such a model with our specific keywords, noise, and other words (the "unknown"):



Dataset

The critical component of Machine Learning Workflow is the **dataset**. Once we have decided on specific keywords (*YES* and NO), we can take advantage of the dataset developed by Pete Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition." This dataset has 35 keywords (with +1,000 samples each), such as yes, no, stop, and go. In other words, we can get 1,500 samples of *yes* and *no*.

You can download a small portion of the dataset from Edge Studio (Keyword spotting pre-built dataset), which includes samples from the four classes we will use in this project: yes, no, noise, and background. For this, follow the steps below:

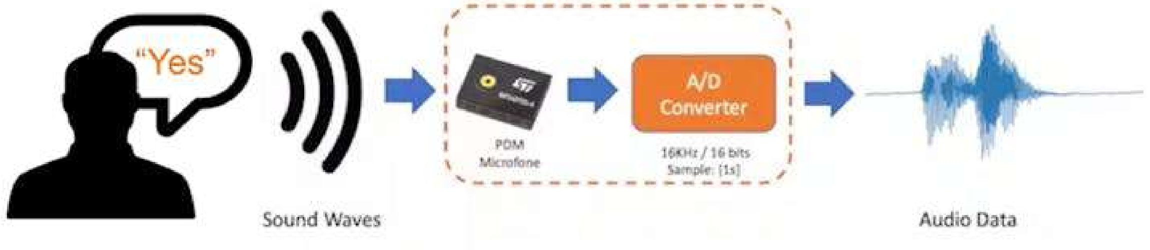
- Download the keywords dataset.
- Unzip the file in a location of your choice.

Although we have a lot of data from Pete's dataset, collecting some words spoken by us is advised. When working with accelerometers, creating a dataset with data captured by the same type of sensor was essential. In the case of *sound*, the classification differs because it involves, in reality, *audio* data.

> The key difference between sound and audio is their form of energy. Sound is mechanical wave energy (longitudinal sound waves) that propagate through a medium causing variations in pressure within the medium. Audio is made of electrical energy (analog or digital signals) that represent sound electrically.

The sound waves should be converted to audio data when we speak a keyword. The conversion should be done by sampling the signal generated by the microphone in 16 kHz with a 16-bit depth.

So, any device that can generate audio data with this basic specification (16 kHz/16 bits) will work fine. As a device, we can use the proper XIAO ESP32S3 Sense, a computer, or even your mobile phone.



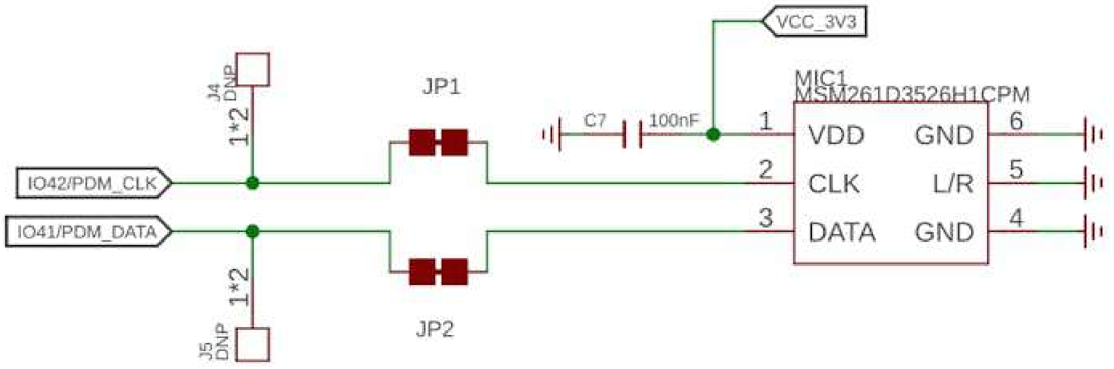
Capturing online Audio Data with Edge Impulse and a smartphone

In the lab Motion Classification and Anomaly Detection, we connect our device directly to Edge Impulse Studio for data capturing (having a sampling frequency of 50 Hz to 100 Hz). For such low frequency, we could use the EI CLI function *Data Forwarder,* but according to Jan Jongboom, Edge Impulse CTO, *audio (*16 kHz) *goes too fast for the data forwarder to be captured.* So, once we have the digital data captured by the microphone, we can turn *it into a WAV file* to be sent to the Studio via Data Uploader (same as we will do with Pete's dataset).

> If we want to collect audio data directly on the Studio, we can use any smartphone connected online with it. We will not explore this option here, but you can easily follow EI documentation.

Capturing (offline) Audio Data with the XIAO ESP32S3 Sense

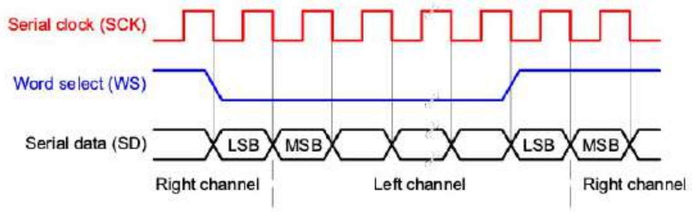
The built-in microphone is the MSM261D3526H1CPM, a PDM digital output MEMS microphone with Multi-modes. Internally, it is connected to the ESP32S3 via an I2S bus using pins IO41 (Clock) and IO41 (Data).



What is I2S?

I2S, or Inter-IC Sound, is a standard protocol for transmitting digital audio from one device to another. It was initially developed by Philips Semiconductor (now NXP Semiconductors). It is commonly used in audio devices such as digital signal processors, digital audio processors, and, more recently, microcontrollers with digital audio capabilities (our case here).

The I2S protocol consists of at least three lines:



1. Bit (or Serial) clock line (BCLK or CLK): This line toggles to indicate the start of a new bit of data (pin IO42).

2. Word select line (WS): This line toggles to indicate the start of a new word (left channel or right channel). The Word select clock (WS) frequency defines the sample rate. In our case, L/R on the microphone is set to ground, meaning that we will use only the left channel (mono).

3. Data line (SD): This line carries the audio data (pin IO41)

In an I2S data stream, the data is sent as a sequence of frames, each containing a left-channel word and a right-channel word. This makes I2S particularly suited for transmitting stereo audio data. However, it can also be used for mono or multichannel audio with additional data lines.

Let's start understanding how to capture raw data using the microphone. Go to the GitHub project and download the sketch: XIAOEsp2s3_Mic_Test:

Attention

- The Xiao ESP32S3 **MUST** have the PSRAM enabled. You can check it on the Arduino IDE upper menu: `Tools–> PSRAM:OPI PSRAM`
- The Arduino Library (`esp32 by Espressif Systems` should be **version 2.017**. Please do not update it.

```
/*
  XIAO ESP32S3 Simple Mic Test
*/

#include <I2S.h>

void setup() {
  Serial.begin(115200);
  while (!Serial) {
  }

  // start I2S at 16 kHz with 16-bits per sample
  I2S.setAllPins(-1, 42, 41, -1, -1);
  if (!I2S.begin(PDM_MONO_MODE, 16000, 16)) {
    Serial.println("Failed to initialize I2S!");
    while (1); // do nothing
  }
}

void loop() {
  // read a sample
  int sample = I2S.read();

  if (sample && sample != -1 && sample != 1) {
    Serial.println(sample);
  }
}
```

This code is a simple microphone test for the XIAO ESP32S3 using the I2S (Inter-IC Sound) interface. It sets up the I2S interface to capture audio data at a sample rate of 16 kHz with 16 bits per sample and then continuously reads samples from the microphone and prints them to the serial monitor.

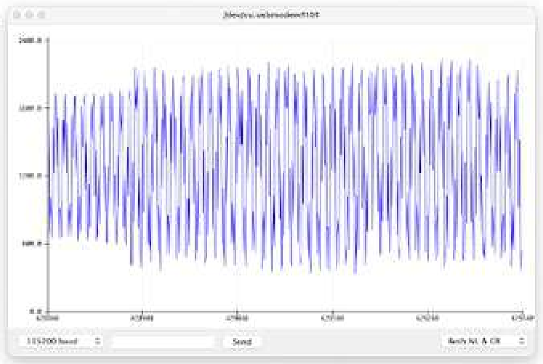
Let's dig into the code's main parts:

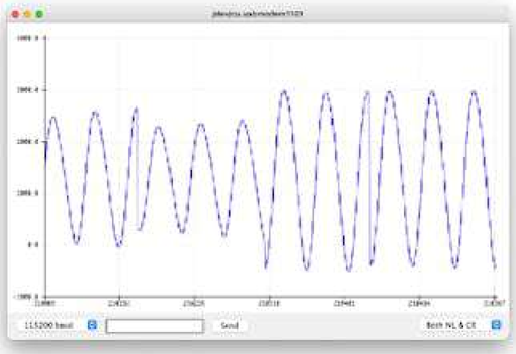
- Include the I2S library: This library provides functions to configure and use the I2S interface, which is a standard for connecting digital audio devices.

- I2S.setAllPins(–1, 42, 41, –1, –1): This sets up the I2S pins. The parameters are (–1, 42, 41, –1, –1), where the second parameter (42) is the PIN for the I2S clock (CLK), and the third parameter (41) is the PIN for the I2S data (DATA) line. The other parameters are set to –1, meaning those pins are not used.
- I2S.begin(PDM_MONO_MODE, 16000, 16): This initializes the I2S interface in Pulse Density Modulation (PDM) mono mode, with a sample rate of 16 kHz and 16 bits per sample. If the initialization fails, an error message is printed, and the program halts.
- int sample = I2S.read(): This reads an audio sample from the I2S interface.

If the sample is valid, it is printed on the Serial Monitor and Plotter.

Below is a test "whispering" in two different tones.



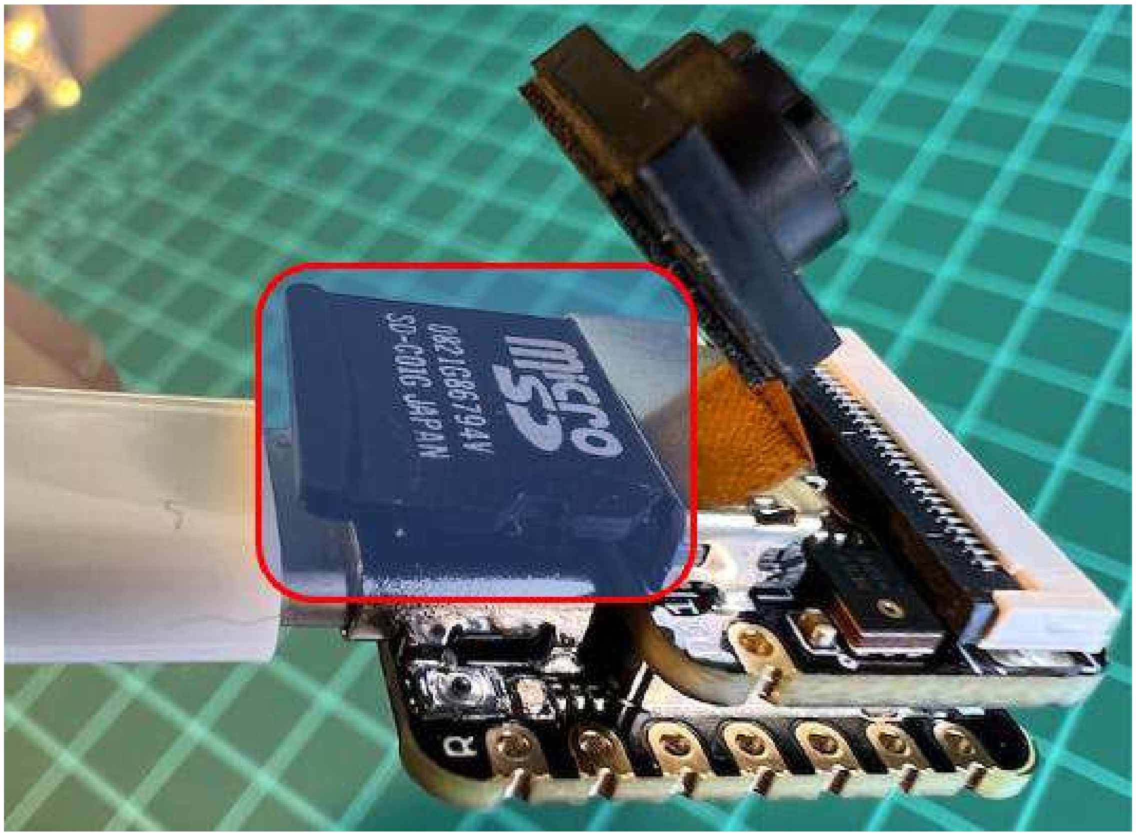


Save Recorded Sound Samples

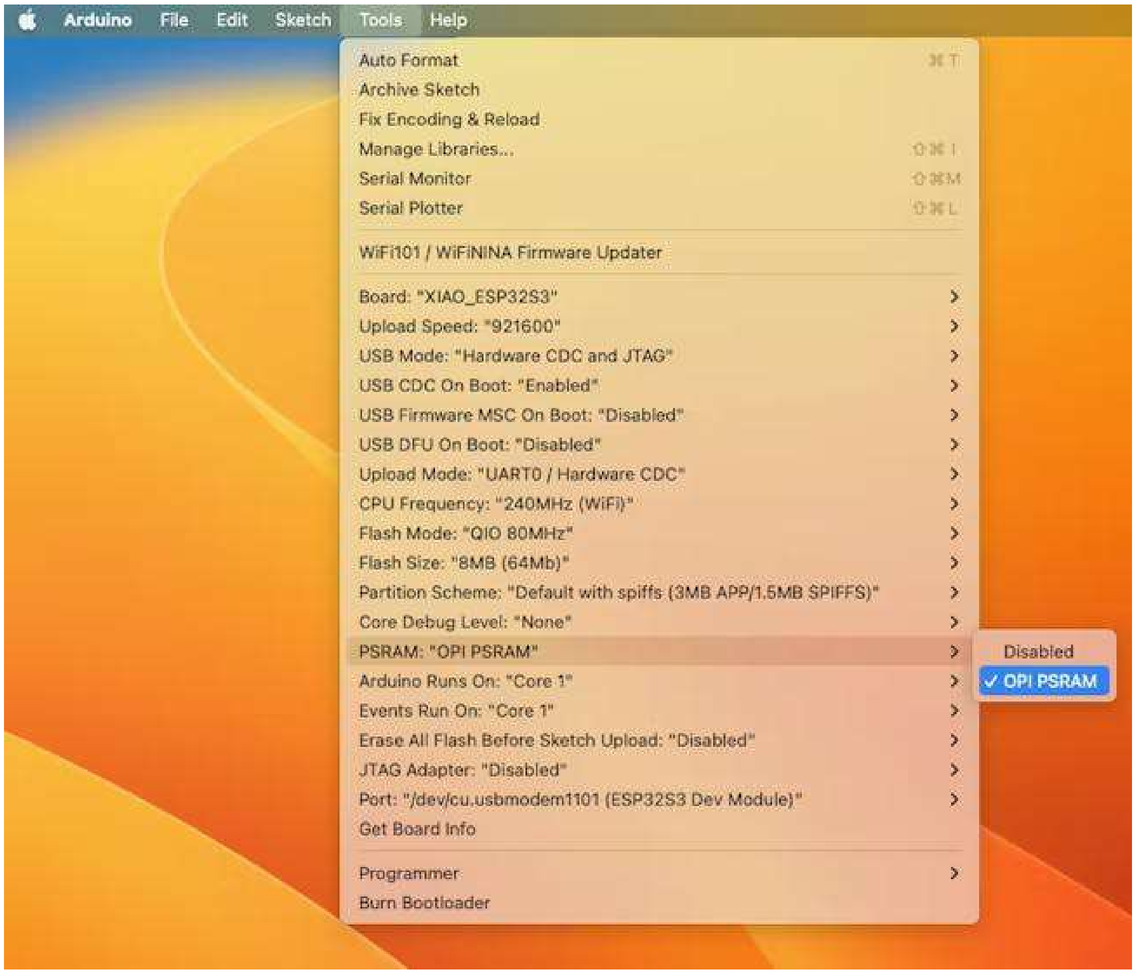
Let's use the onboard SD Card reader to save .wav audio files; we must habilitate the XIAO PSRAM first.

> ESP32-S3 has only a few hundred kilobytes of internal RAM on the MCU chip. It can be insufficient for some purposes so that ESP32-S3 can use up to 16 MB of external PSRAM (Psuedostatic RAM) connected in parallel with the SPI flash chip. The external memory is incorporated in the memory map and, with certain restrictions, is usable in the same way as internal data RAM.

For a start, Insert the SD Card on the XIAO as shown in the photo below (the SD Card should be formatted to FAT32).



Turn the PSRAM function of the ESP-32 chip on (Arduino IDE): Tools>PSRAM: “OPI PSRAM”>OPI PSRAM



Board: "XIAO_ESP32S3"
Upload Speed: "921600"
USB Mode: "Hardware CDC and JTAG"
USB CDC On Boot: "Enabled"
USB Firmware MSC On Boot: "Disabled"
USB DFU On Boot: "Disabled"
Upload Mode: "UART0 / Hardware CDC"
CPU Frequency: "240MHz (WiFi)"
Flash Mode: "QIO 80MHz"
Flash Size: "8MB (64Mb)"
Partition Scheme: "Default with spiffs (3MB APP/1.5MB SPIFFS)"
Core Debug Level: "None"
PSRAM: "OPI PSRAM" — Disabled [✓ OPI PSRAM]
Arduino Runs On: "Core 1"
Events Run On: "Core 1"
Erase All Flash Before Sketch Upload: "Disabled"
JTAG Adapter: "Disabled"
Port: "/dev/cu.usbmodem1101 (ESP32S3 Dev Module)"

- Download the sketch Wav_Record_dataset, which you can find on the project's GitHub.

This code records audio using the I2S interface of the Seeed XIAO ESP32S3 Sense board, saves the recording as a.wav file on an SD card, and allows for control of the recording process through commands sent from the serial monitor. The name of the audio file is customizable (it should be the class labels to be used with the training), and multiple recordings can be made, each saved in a new file. The code also includes functionality to increase the volume of the recordings.

Let's break down the most essential parts of it:

```
#include <I2S.h>
#include "FS.h"
#include "SD.h"
#include "SPI.h"
```

Those are the necessary libraries for the program. I2S.h allows for audio input, FS.h provides file system handling capabilities, SD.h enables the program to interact with an SD card, and SPI.h handles the SPI communication with the SD card.

```
#define RECORD_TIME   10
#define SAMPLE_RATE 16000U
#define SAMPLE_BITS 16
#define WAV_HEADER_SIZE 44
#define VOLUME_GAIN 2
```

Here, various constants are defined for the program.

- **RECORD_TIME** specifies the length of the audio recording in seconds.
- **SAMPLE_RATE** and **SAMPLE_BITS** define the audio quality of the recording.
- **WAV_HEADER_SIZE** specifies the size of the .wav file header.
- **VOLUME_GAIN** is used to increase the volume of the recording.

```
int fileNumber = 1;
String baseFileName;
bool isRecording = false;
```

These variables keep track of the current file number (to create unique file names), the base file name, and whether the system is currently recording.

```
void setup() {
  Serial.begin(115200);
  while (!Serial);

  I2S.setAllPins(-1, 42, 41, -1, -1);
  if (!I2S.begin(PDM_MONO_MODE, SAMPLE_RATE, SAMPLE_BITS)) {
    Serial.println("Failed to initialize I2S!");
    while (1);
  }

  if(!SD.begin(21)){
    Serial.println("Failed to mount SD Card!");
    while (1);
  }
  Serial.printf("Enter with the label name\n");
}
```

The setup function initializes the serial communication, I2S interface for audio input, and SD card interface. If the I2S did not initialize or the SD card fails to mount, it will print an error message and halt execution.

```
void loop() {
  if (Serial.available() > 0) {
    String command = Serial.readStringUntil('\n');
    command.trim();
    if (command == "rec") {
      isRecording = true;
    } else {
      baseFileName = command;
      fileNumber = 1; //reset file number each time a new
                        basefile name is set
      Serial.printf("Send rec for starting recording label \n");
    }
  }
  if (isRecording && baseFileName != "") {
    String fileName = "/" + baseFileName + "."
                      + String(fileNumber) + ".wav";
    fileNumber++;
    record_wav(fileName);
    delay(1000); // delay to avoid recording multiple files
                    at once
    isRecording = false;
  }
}
```

In the main loop, the program waits for a command from the serial monitor. If the command is rec, the program starts recording. Otherwise, the command is assumed to be the base name for the .wav files. If it's currently recording and a base file name is set, it records the audio and saves it as a.wav file. The file names are generated by appending the file number to the base file name.

```
void record_wav(String fileName)
{
  ...

  File file = SD.open(fileName.c_str(), FILE_WRITE);
  ...
  rec_buffer = (uint8_t *)ps_malloc(record_size);
  ...

  esp_i2s::i2s_read(esp_i2s::I2S_NUM_0,
                    rec_buffer,
                    record_size,
                    &sample_size,
                    portMAX_DELAY);
  ...
}
```

This function records audio and saves it as a.wav file with the given name. It starts by initializing the sample_size and record_size variables. record_size is calculated based on the sample rate, size, and desired recording time. Let's dig into the essential sections;

```
File file = SD.open(fileName.c_str(), FILE_WRITE);
// Write the header to the WAV file
uint8_t wav_header[WAV_HEADER_SIZE];
generate_wav_header(wav_header, record_size, SAMPLE_RATE);
file.write(wav_header, WAV_HEADER_SIZE);
```

This section of the code opens the file on the SD card for writing and then generates the .wav file header using the generate_wav_header function. It then writes the header to the file.

```
// PSRAM malloc for recording
rec_buffer = (uint8_t *)ps_malloc(record_size);
if (rec_buffer == NULL) {
  Serial.printf("malloc failed!\n");
  while(1) ;
}
Serial.printf("Buffer: %d bytes\n", ESP.getPsramSize()
               - ESP.getFreePsram());
```

The ps_malloc function allocates memory in the PSRAM for the recording. If the allocation fails (i.e., rec_buffer is NULL), it prints an error message and halts execution.

```
// Start recording
esp_i2s::i2s_read(esp_i2s::I2S_NUM_0,
         rec_buffer,
         record_size,
         &sample_size,
         portMAX_DELAY);
if (sample_size == 0) {
  Serial.printf("Record Failed!\n");
} else {
    Serial.printf("Record %d bytes\n", sample_size);
  }
```

The i2s_read function reads audio data from the microphone into rec_buffer. It prints an error message if no data is read (sample_size is 0).

```
// Increase volume
for (uint32_t i = 0; i < sample_size; i += SAMPLE_BITS/8) {
  (*(uint16_t *)(rec_buffer+i)) <<= VOLUME_GAIN;
}
```

This section of the code increases the recording volume by shifting the sample values by VOLUME_GAIN.

```
// Write data to the WAV file
Serial.printf("Writing to the file ...\n");
if (file.write(rec_buffer, record_size) != record_size)
  Serial.printf("Write file Failed!\n");

free(rec_buffer);
file.close();
Serial.printf("Recording complete: \n");
Serial.printf("Send rec for a new sample or enter
                a new label\n\n");
```

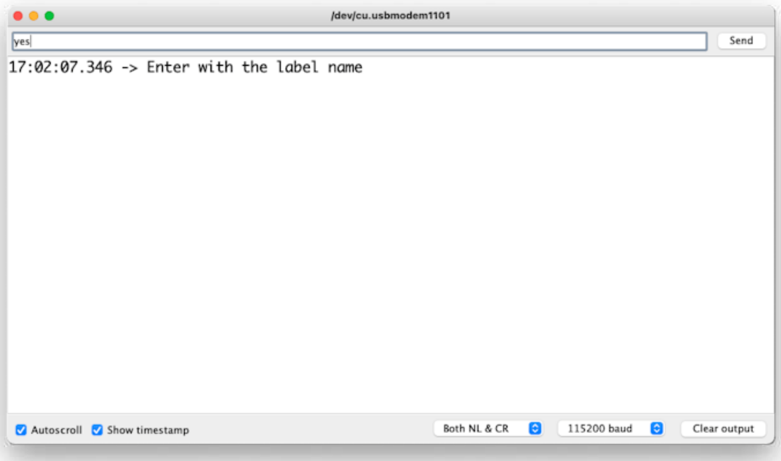
Finally, the audio data is written to the .wav file. If the write operation fails, it prints an error message. After writing, the memory allocated for rec_buffer is freed, and the file is closed. The function finishes by printing a completion message and prompting the user to send a new command.

```
void generate_wav_header(uint8_t *wav_header,
             uint32_t wav_size,
             uint32_t sample_rate)
{
  ...
  memcpy(wav_header, set_wav_header, sizeof(set_wav_header));
}
```

The generate_wav_header function creates a.wav file header based on the parameters (wav_size and sample_rate). It generates an array of bytes according to the .wav file format, which includes fields for the file size, audio format, number of channels, sample rate, byte rate, block alignment, bits per sample, and data size. The generated header is then copied into the wav_header array passed to the function.

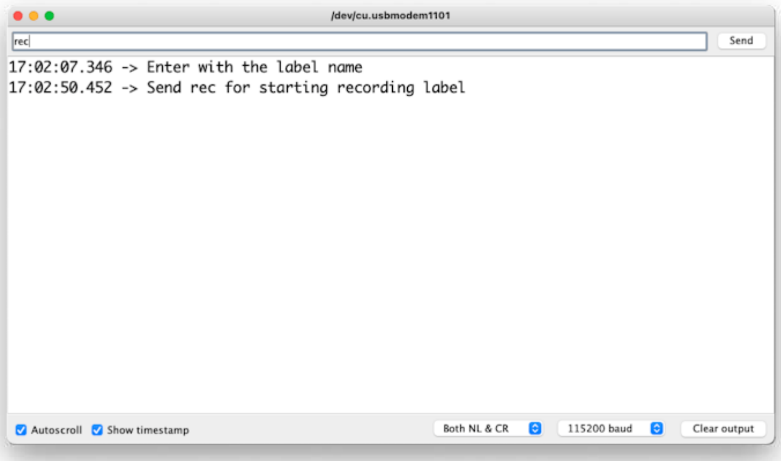
Now, upload the code to the XIAO and get samples from the keywords (yes and no). You can also capture noise and other words.

The Serial monitor will prompt you to receive the label to be recorded.



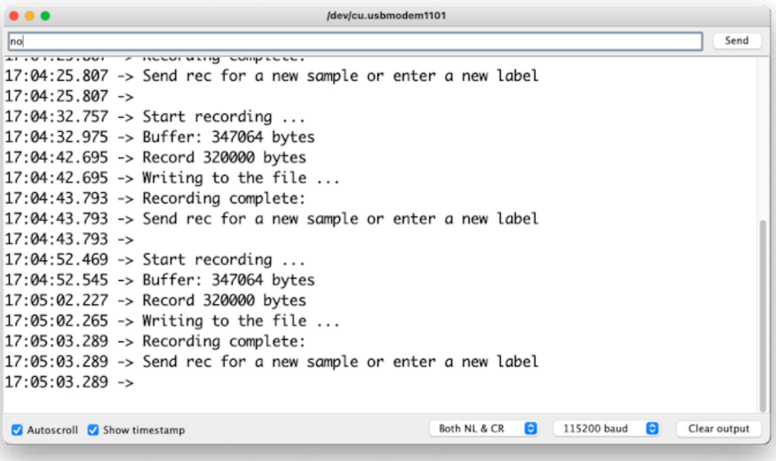
```
17:02:07.346 -> Enter with the label name
```

Send the label (for example, yes). The program will wait for another command: rec



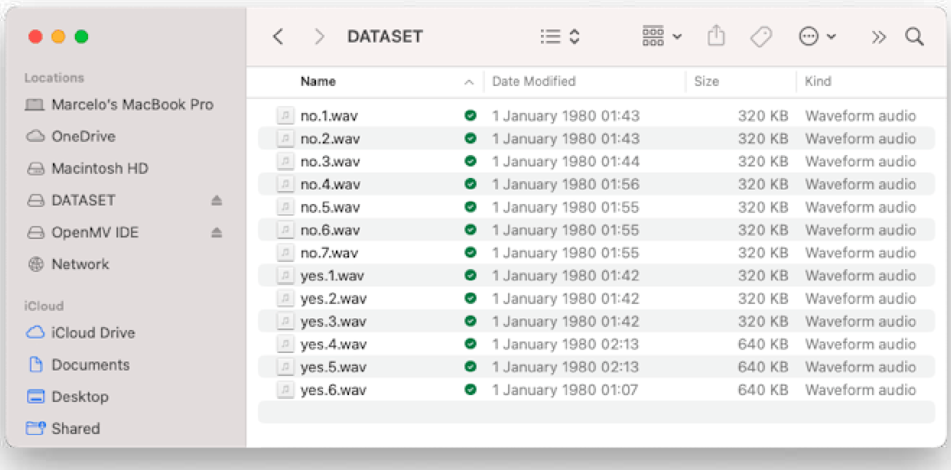
```
17:02:07.346 -> Enter with the label name
17:02:50.452 -> Send rec for starting recording label
```

And the program will start recording new samples every time a command rec is sent. The files will be saved as yes.1.wav, yes.2.wav, yes.3.wav, etc., until a new label (for example, no) is sent. In this case, you should send the command rec for each new sample, which will be saved as no.1.wav, no.2.wav, no.3.wav, etc.



```
17:04:25.807 -> Send rec for a new sample or enter a new label
17:04:25.807 ->
17:04:32.757 -> Start recording ...
17:04:32.975 -> Buffer: 347064 bytes
17:04:42.695 -> Record 320000 bytes
17:04:42.695 -> Writing to the file ...
17:04:43.793 -> Recording complete:
17:04:43.793 -> Send rec for a new sample or enter a new label
17:04:43.793 ->
17:04:52.469 -> Start recording ...
17:04:52.545 -> Buffer: 347064 bytes
17:05:02.227 -> Record 320000 bytes
17:05:02.265 -> Writing to the file ...
17:05:03.289 -> Recording complete:
17:05:03.289 -> Send rec for a new sample or enter a new label
17:05:03.289 ->
```

Ultimately, we will get the saved files on the SD card.



Name	Date Modified	Size	Kind
no.1.wav	1 January 1980 01:43	320 KB	Waveform audio
no.2.wav	1 January 1980 01:43	320 KB	Waveform audio
no.3.wav	1 January 1980 01:44	320 KB	Waveform audio
no.4.wav	1 January 1980 01:56	320 KB	Waveform audio
no.5.wav	1 January 1980 01:55	320 KB	Waveform audio
no.6.wav	1 January 1980 01:55	320 KB	Waveform audio
no.7.wav	1 January 1980 01:55	320 KB	Waveform audio
yes.1.wav	1 January 1980 01:42	320 KB	Waveform audio
yes.2.wav	1 January 1980 01:42	320 KB	Waveform audio
yes.3.wav	1 January 1980 01:42	320 KB	Waveform audio
yes.4.wav	1 January 1980 02:13	640 KB	Waveform audio
yes.5.wav	1 January 1980 02:13	640 KB	Waveform audio
yes.6.wav	1 January 1980 01:07	640 KB	Waveform audio

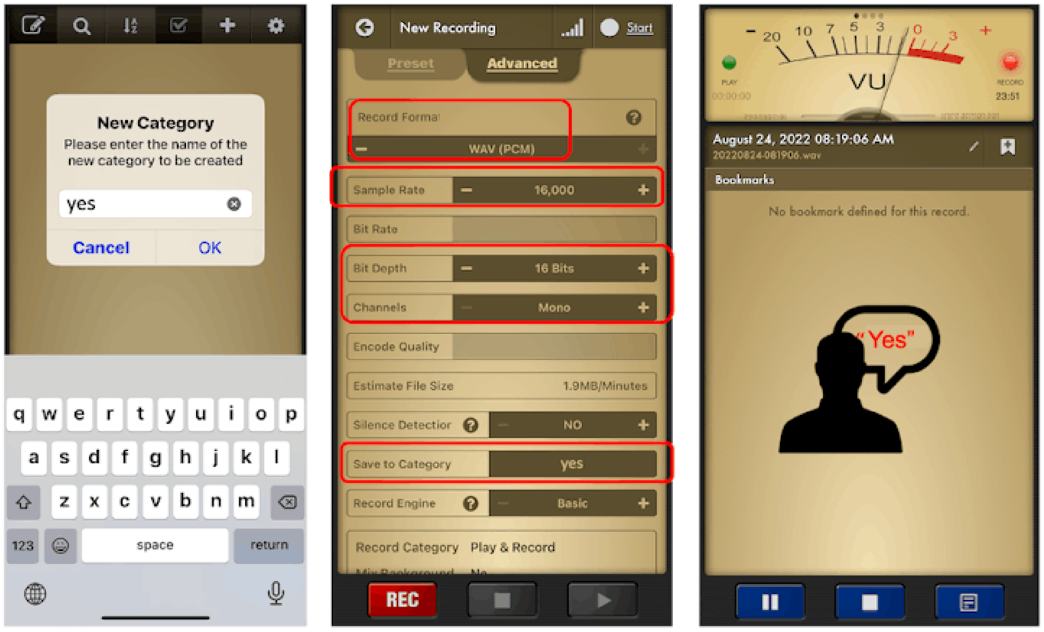
The files are ready to be uploaded to Edge Impulse Studio

Capturing (offline) Audio Data Apps

There are many ways to capture audio data; the simplest one is to use a mobile phone or a PC as a **connected device** on the Edge Impulse Studio.

> The PC or smartphone should capture audio data with a sampling frequency of 16 kHz and a bit depth of 16 Bits.

Another alternative is to use dedicated apps. A good app for that is *Voice Recorder Pro* (IOS). You should save your records as .wav files and send them to your computer.


New Category
Please enter the name of the new category to be created
yes
Cancel OK

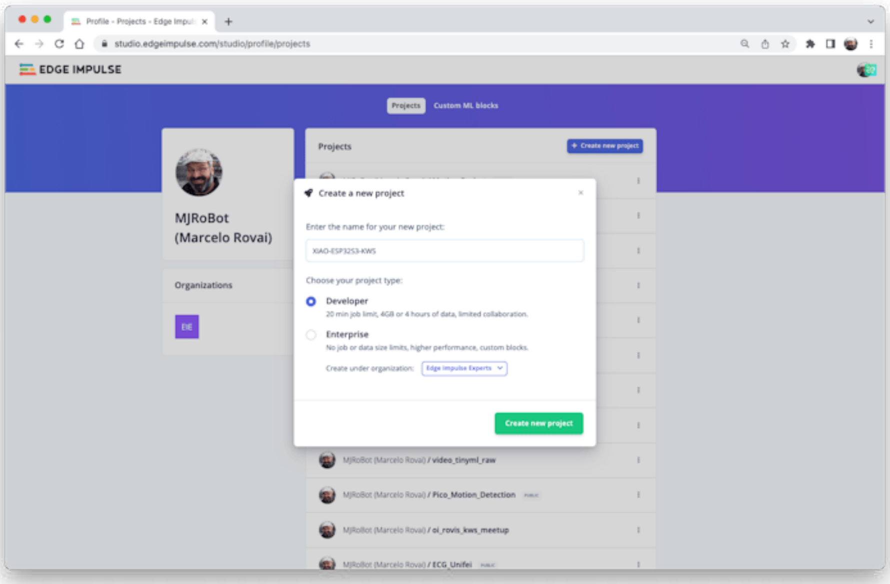
New Recording Start
Preset Advanced
Record Format WAV (PCM)
Sample Rate 16,000
Bit Rate
Bit Depth 16 Bits
Channels Mono
Encode Quality
Estimate File Size 1.9MB/Minutes
Silence Detection NO
Save to Category yes
Record Engine Basic
Record Category Play & Record
REC

August 24, 2022 08:19:06 AM
20220824-081906.wav
Bookmarks
No bookmark defined for this record.

Training model with Edge Impulse Studio

Uploading the Data

When the raw dataset is defined and collected (Pete's dataset + recorded keywords), we should initiate a new project at Edge Impulse Studio:


Create a new project

Enter the name for your new project: XIAO-ESP32S3-KWS

Choose your project type:

[✓ Developer] 20 min job limit, 4GB or 4 hours of data, limited collaboration.

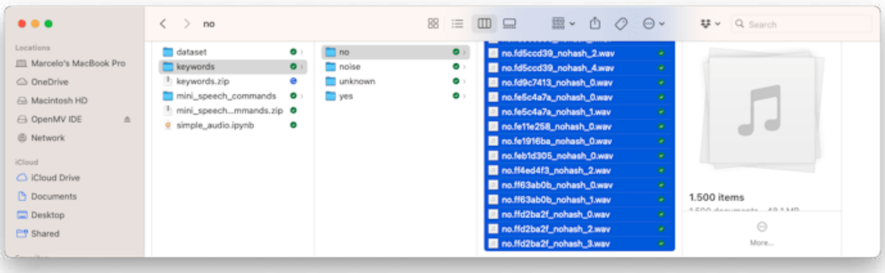
Enterprise No job or data size limits, higher performance, custom blocks.

Create under organization: Edge Impulse Experts

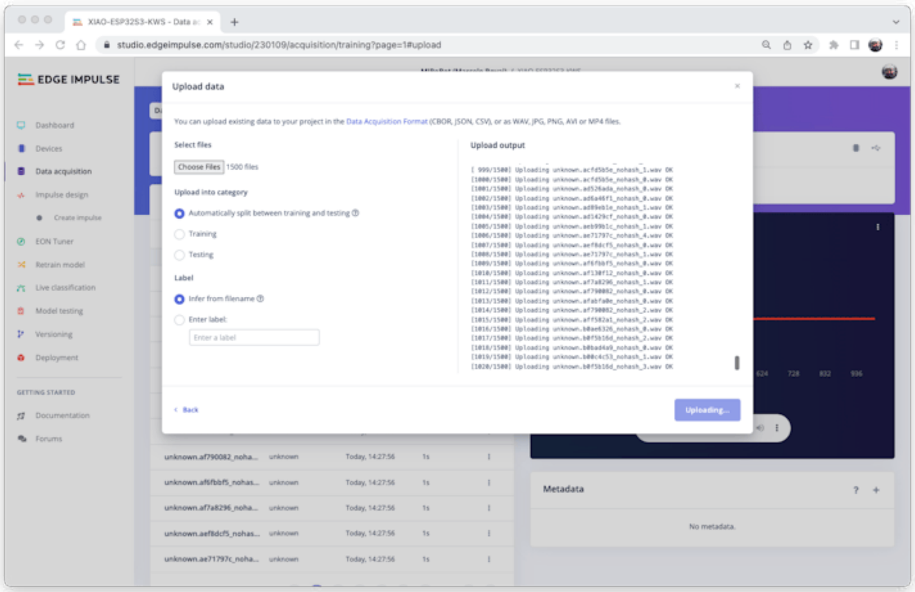
Create new project

studio.edgeimpulse.com/studio/profile/projects
EDGE IMPULSE
Projects
Custom ML blocks
+ Create new project
MJRoBot (Marcelo Rovai)
Organizations
MJRoBot (Marcelo Rovai) / video_tinyml_raw
MJRoBot (Marcelo Rovai) / Pico_Motion_Detection
MJRoBot (Marcelo Rovai) / oi_rovis_kws_meetup
MJRoBot (Marcelo Rovai) / ECG_Unifei

Once the project is created, select the Upload Existing Data tool in the Data Acquisition section. Choose the files to be uploaded:



And upload them to the Studio (You can automatically split data in train/test). Repeat to all classes and all raw data.



Upload data

You can upload existing data to your project in the Data Acquisition Format (CBOR, JSON, CSV), or as WAV, JPG, PNG, AVI or MP4 files.

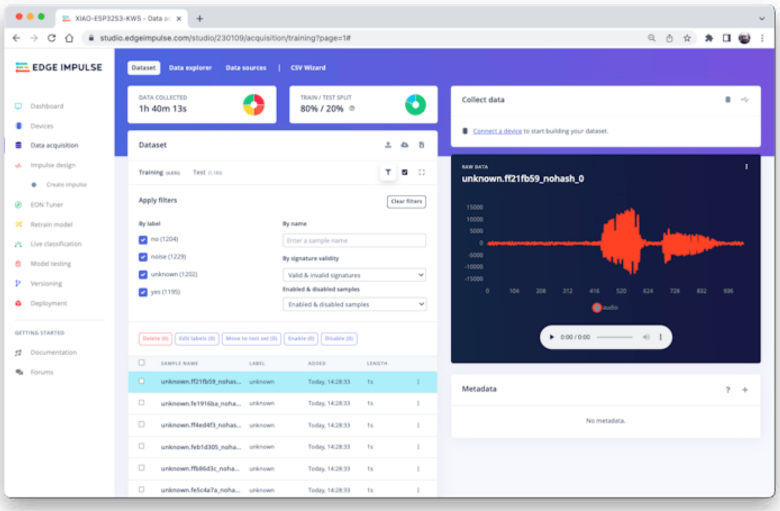
Select files: Choose Files 1500 files

Upload into category: [✓ Automatically split between training and testing] Training Testing

Label: [✓ Infer from filename] Enter label:

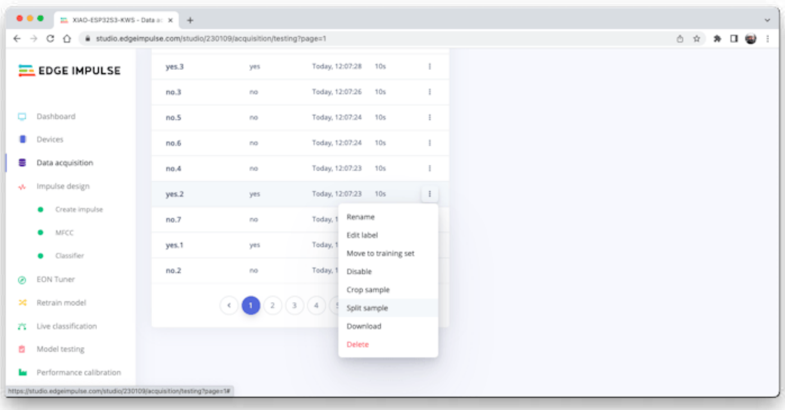
```
[ 999/1500] Uploading unknown.acfd5b5e_nohash_1.wav OK
[1000/1500] Uploading unknown.acfd5b5e_nohash_0.wav OK
[1001/1500] Uploading unknown.ad526ada_nohash_0.wav OK
[1002/1500] Uploading unknown.ad6a46f1_nohash_0.wav OK
[1003/1500] Uploading unknown.ad89eb1e_nohash_1.wav OK
[1004/1500] Uploading unknown.ad1429cf_nohash_0.wav OK
[1005/1500] Uploading unknown.aeb99b1c_nohash_1.wav OK
[1006/1500] Uploading unknown.ae71797c_nohash_4.wav OK
[1007/1500] Uploading unknown.aef8dcf5_nohash_0.wav OK
[1008/1500] Uploading unknown.ae71797c_nohash_1.wav OK
[1009/1500] Uploading unknown.af6fbbf5_nohash_0.wav OK
[1010/1500] Uploading unknown.af130f12_nohash_0.wav OK
[1011/1500] Uploading unknown.af7a8296_nohash_1.wav OK
[1012/1500] Uploading unknown.af790082_nohash_0.wav OK
[1013/1500] Uploading unknown.afabfa0e_nohash_0.wav OK
[1014/1500] Uploading unknown.af790082_nohash_2.wav OK
[1015/1500] Uploading unknown.aff582a1_nohash_2.wav OK
[1016/1500] Uploading unknown.b0ae6326_nohash_0.wav OK
[1017/1500] Uploading unknown.b0f5b16d_nohash_2.wav OK
[1018/1500] Uploading unknown.b0bad4a9_nohash_0.wav OK
[1019/1500] Uploading unknown.b00c4c53_nohash_1.wav OK
[1020/1500] Uploading unknown.b0f5b16d_nohash_3.wav OK
```

The samples will now appear in the Data acquisition section.

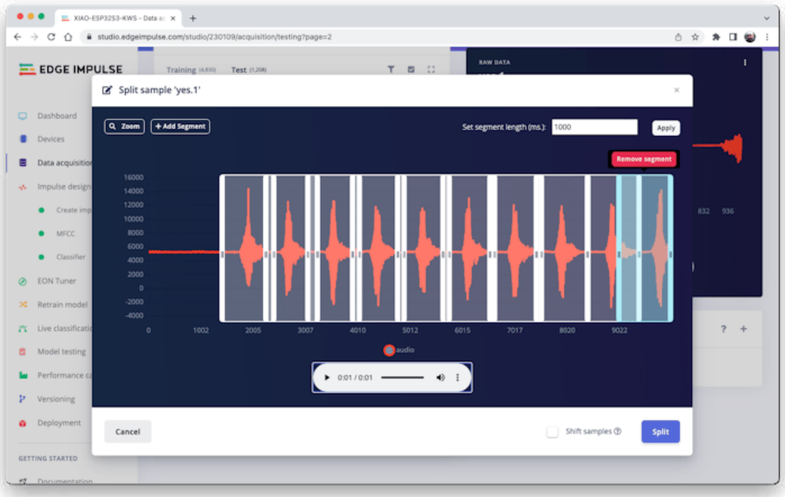


All data on Pete's dataset have a 1 s length, but the samples recorded in the previous section have 10 s and must be split into 1s samples to be compatible.

Click on three dots after the sample name and select Split sample.



Once inside the tool, split the data into 1-second records. If necessary, add or remove segments:



This procedure should be repeated for all samples.

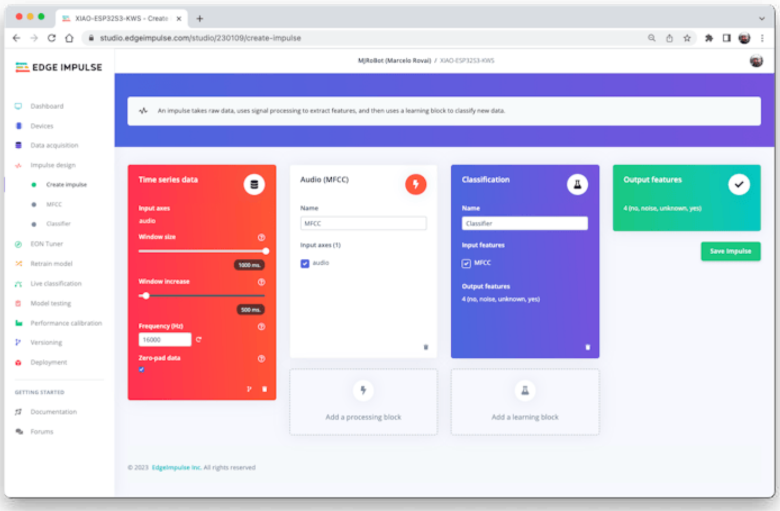
> Note: For longer audio files (minutes), first, split into 10-second segments and after that, use the tool again to get the final 1-second splits.

Suppose we do not split data automatically in train/test during upload. In that case, we can do it manually (using the three dots menu, moving samples individually) or using Perform Train / Test Split on Dashboard – Danger Zone.

We can optionally check all datasets using the tab Data Explorer.

Creating Impulse (Pre-Process / Model definition)

An **impulse** *takes raw data, uses signal processing to extract features, and then uses a learning block to classify new data.*



An impulse takes raw data, uses signal processing to extract features, and then uses a learning block to classify new data.

Time series data
Input axes
audio
Window size
1000 ms.
Window increase
500 ms.
Frequency (Hz)
16000
Zero-pad data
☑

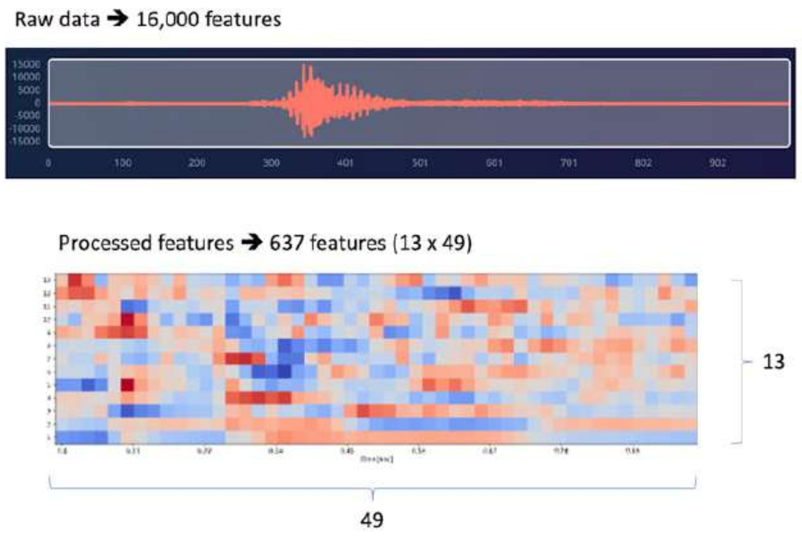
Audio (MFCC)
Name
MFCC
Input axes (1)
☑ audio

Classification
Name
Classifier
Input features
☑ MFCC
Output features
4 (no, noise, unknown, yes)

Output features
4 (no, noise, unknown, yes)

First, we will take the data points with a 1-second window, augmenting the data, sliding that window each 500 ms. Note that the option zero-pad data is set. It is essential to fill with zeros samples smaller than 1 second (in some cases, I reduced the 1000 ms window on the split tool to avoid noises and spikes).

Each 1-second audio sample should be pre-processed and converted to an image (for example, $13 \times 49 \times 1$). We will use MFCC, which extracts features from audio signals using Mel Frequency Cepstral Coefficients, which are great for the human voice.

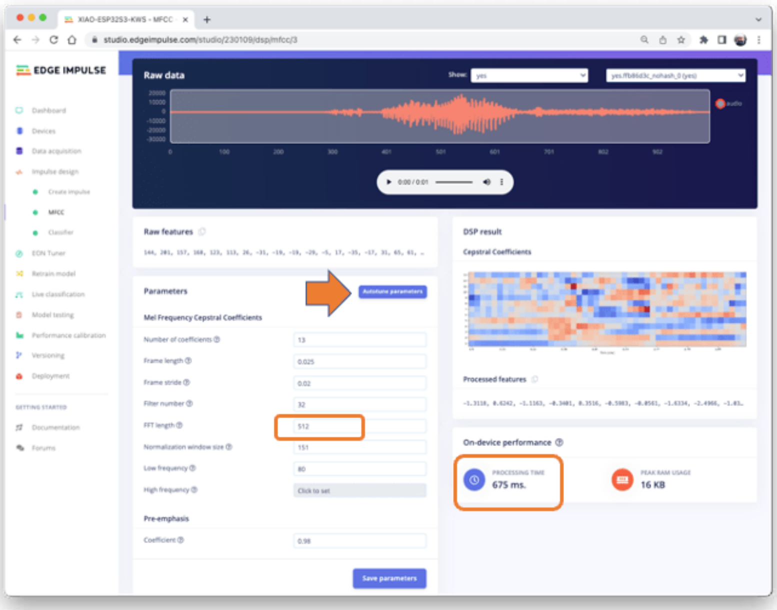


Next, we select KERAS for classification and build our model from scratch by doing Image Classification using Convolution Neural Network).

Pre-Processing (MFCC)

The next step is to create the images to be trained in the next phase:

We can keep the default parameter values or take advantage of the DSP Autotuneparameters option, which we will do.



Raw features
144, 201, 157, 168, 123, 113, 26, -31, -19, -19, -29, -5, 17, -35, -17, 31, 65, 61, ...

Parameters

Mel Frequency Cepstral Coefficients

Number of coefficients	13
Frame length	0.025
Frame stride	0.02
Filter number	32
FFT length	512
Normalization window size	151
Low frequency	80
High frequency	Click to set

Pre-emphasis

Coefficient	0.98

DSP result

Cepstral Coefficients

Processed features
-1.3118, 0.6242, -1.1163, -0.3401, 0.3516, -0.5983, -0.0561, -1.6334, -2.4966, -1.03...

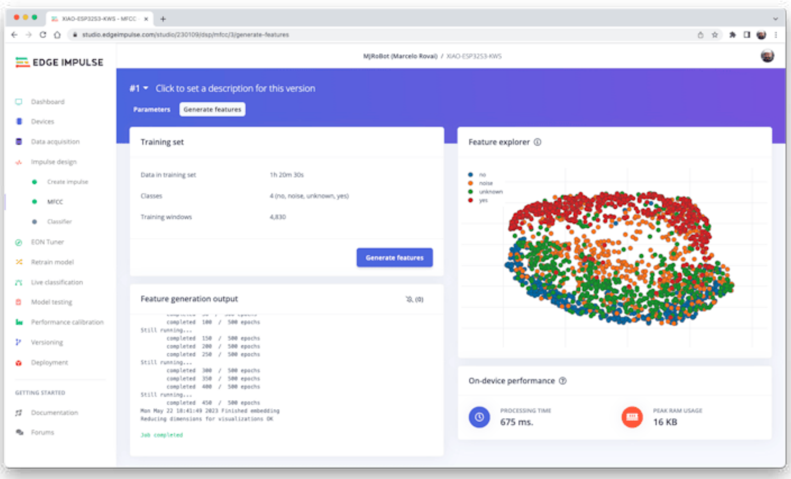
On-device performance

PROCESSING TIME 675 ms.
PEAK RAM USAGE 16 KB

The result will not spend much memory to pre-process data (only 16KB). Still, the estimated processing time is high, 675 ms for an Espressif ESP-EYE (the closest reference available), with a 240 kHz clock (same as our device), but with a smaller CPU (XTensa LX6, versus the LX7 on the ESP32S). The real inference time should be smaller.

Suppose we need to reduce the inference time later. In that case, we should return to the pre-processing stage and, for example, reduce the FFT length to 256, change the Number of coefficients, or another parameter.

For now, let's keep the parameters defined by the Autotuning tool. Save parameters and generate the features.



Training set

Data in training set	1h 20m 30s
Classes	4 (no, noise, unknown, yes)
Training windows	4,830

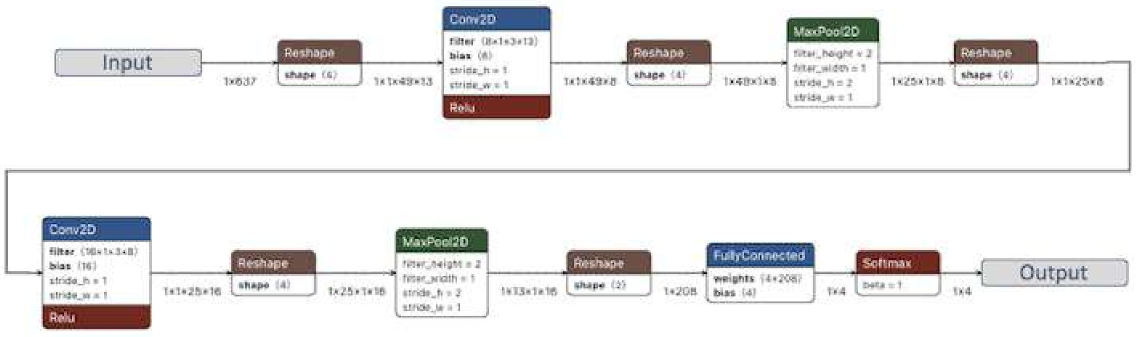
Feature generation output

```
        completed  100  /  500 epochs
Still running...
        completed  150  /  500 epochs
        completed  200  /  500 epochs
        completed  250  /  500 epochs
Still running...
        completed  300  /  500 epochs
        completed  350  /  500 epochs
        completed  400  /  500 epochs
Still running...
        completed  450  /  500 epochs
Mon May 22 18:41:49 2023 Finished embedding
Reducing dimensions for visualizations OK

Job completed
```

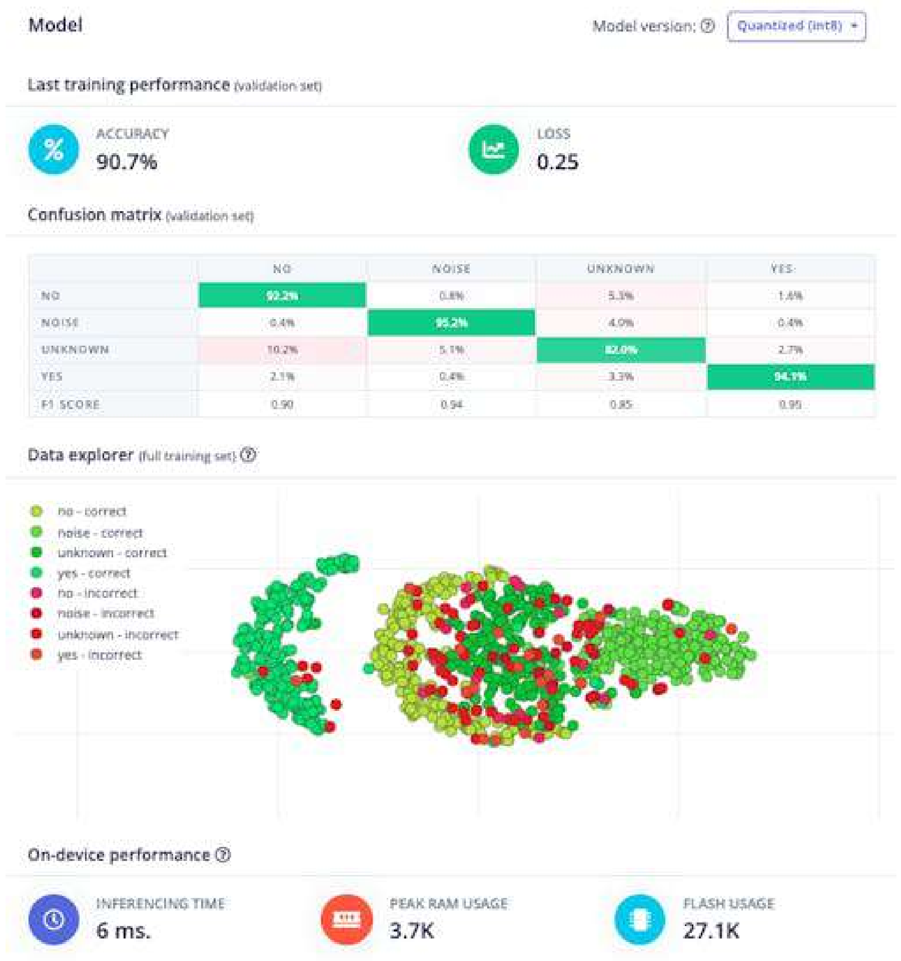
If you want to go further with converting temporal serial data into images using FFT, Spectrogram, etc., you can play with this CoLab: Audio Raw Data Analysis and digging into the lab KWS Feature Engineering

Model Design and Training

We will use a Convolution Neural Network (CNN) model. The basic architecture consists of two blocks of Conv1D + MaxPooling (with 8 and 16 neurons, respectively) and a 0.25 Dropout. And on the last layer, after flattening four neurons, one for each class:



As hyper-parameters, we will have a Learning Rate of 0.005 and a model that will be trained by 100 epochs. We will also include data augmentation, as some noise. The result seems OK:



Model

Model version: ⓘ Quantized (int8)

Last training performance (validation set)

ACCURACY
90.7%

LOSS
0.25

Confusion matrix (validation set)

	NO	NOISE	UNKNOWN	YES
NO	92.2%	0.8%	5.3%	1.6%
NOISE	0.4%	95.2%	4.0%	0.4%
UNKNOWN	10.2%	5.1%	82.0%	2.7%
YES	2.1%	0.4%	3.3%	94.1%
F1 SCORE	0.90	0.94	0.85	0.95

Data explorer (full training set) ⓘ

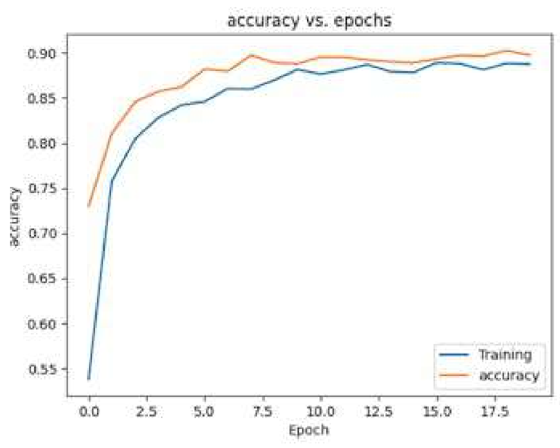
On-device performance ⓘ

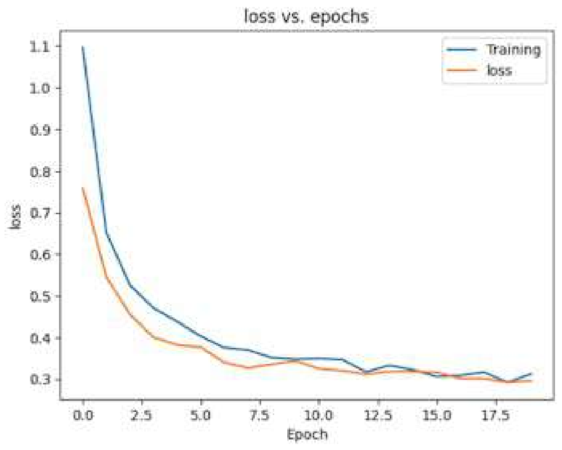
INFERENCING TIME
6 ms.

PEAK RAM USAGE
3.7K

FLASH USAGE
27.1K

If you want to understand what is happening “under the hood,” you can download the dataset and run a Jupyter Notebook playing with the code. For example, you can analyze the accuracy by each epoch:

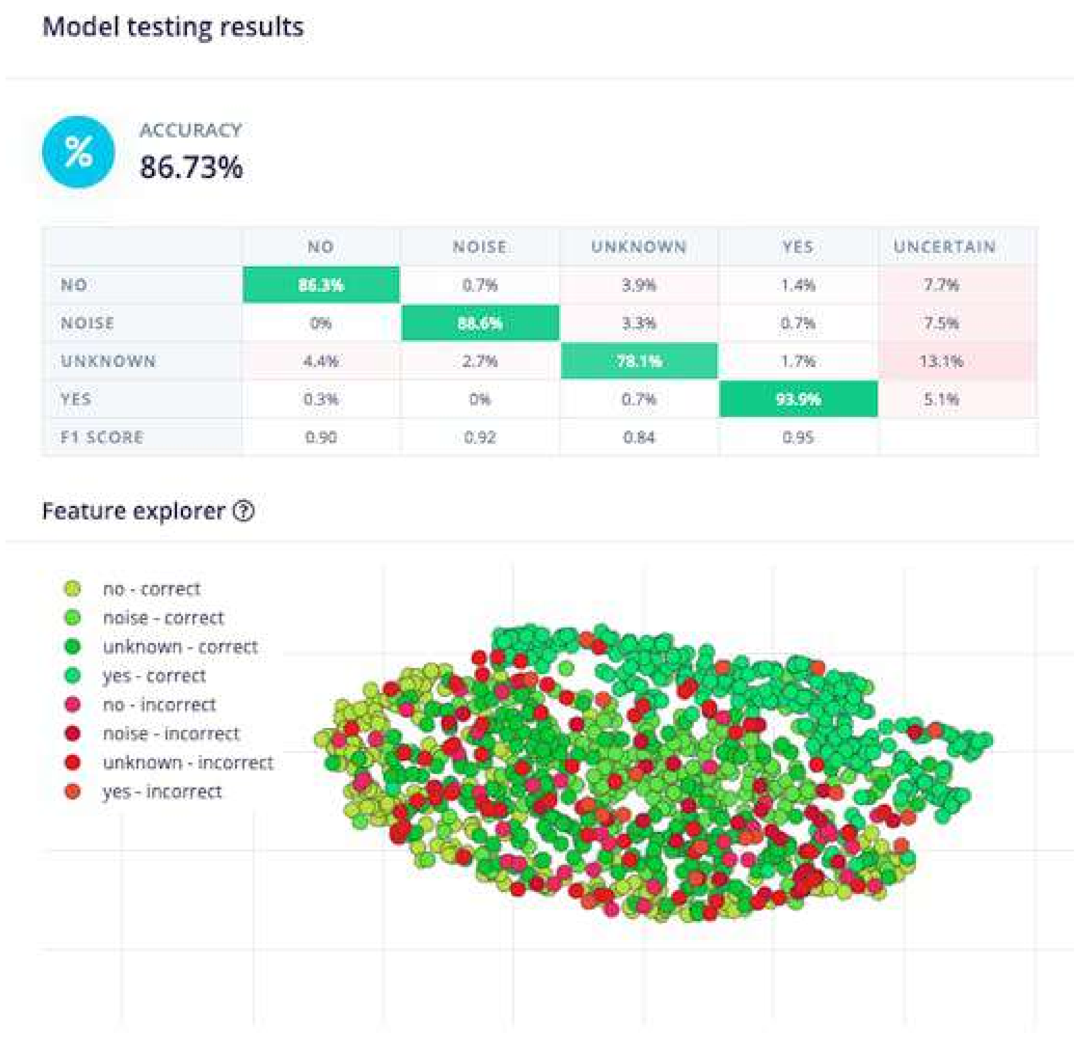




This CoLab Notebook can explain how you can go further: KWS Classifier Project - Looking "Under the hood."

Testing

Testing the model with the data put apart before training (Test Data), we got an accuracy of approximately 87%.

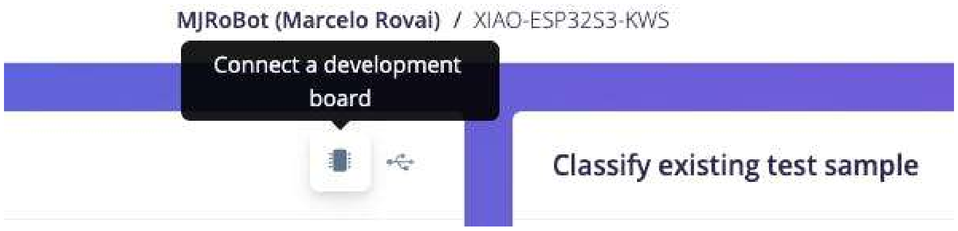

Model testing results

ACCURACY
86.73%

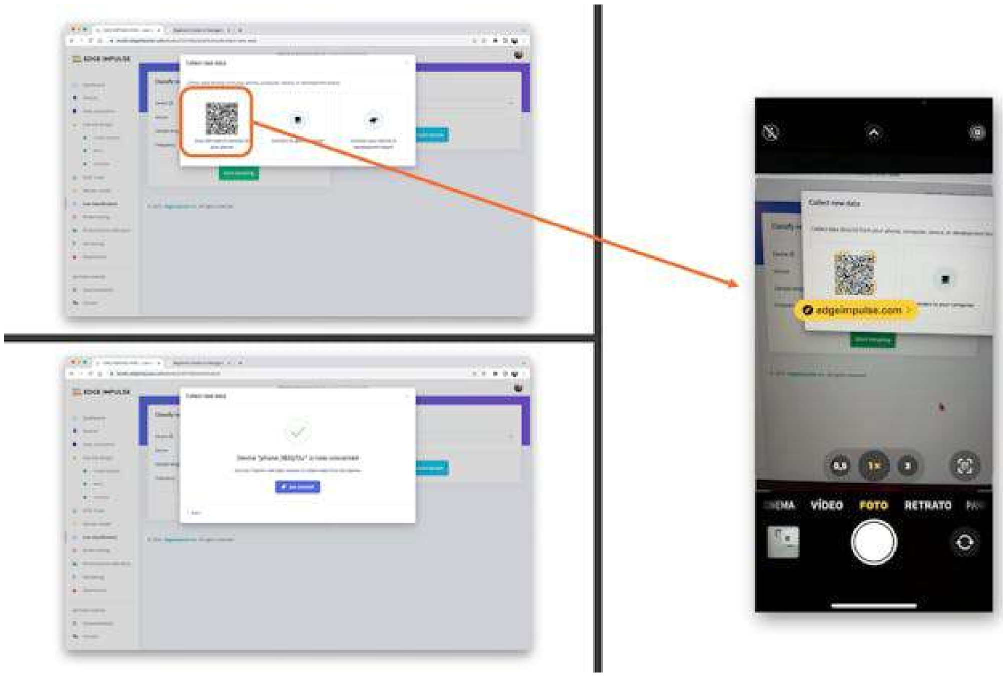
	NO	NOISE	UNKNOWN	YES	UNCERTAIN
NO	86.3%	0.7%	3.9%	1.4%	7.7%
NOISE	0%	88.6%	3.3%	0.7%	7.5%
UNKNOWN	4.4%	2.7%	78.1%	1.7%	13.1%
YES	0.3%	0%	0.7%	93.9%	5.1%
F1 SCORE	0.90	0.92	0.84	0.95	

Inspecting the F1 score, we can see that for YES, we got 0.95, an excellent result once we used this keyword to "trigger" our postprocessing stage (turn on the built-in LED). Even for NO, we got 0.90. The worst result is for unknown, what is OK.

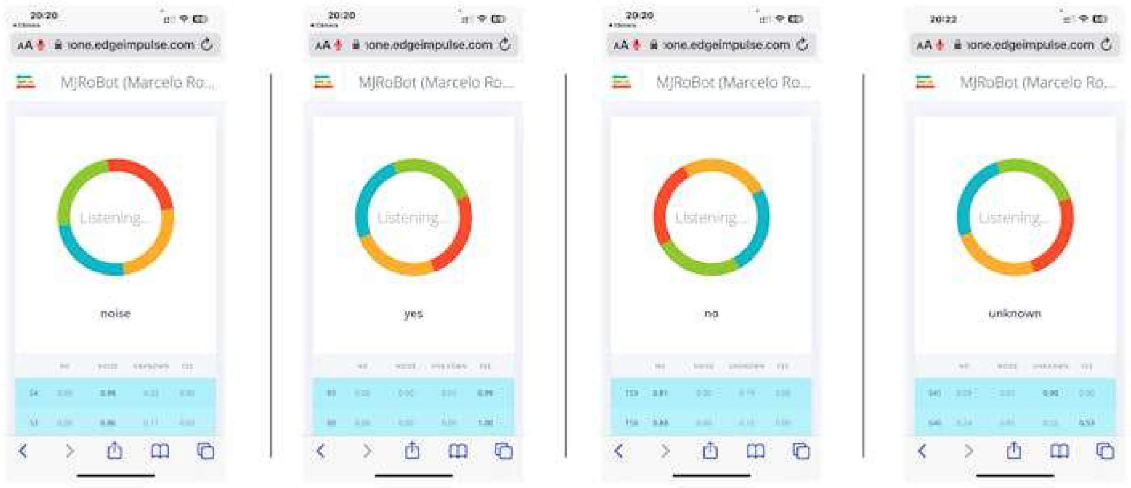
We can proceed with the project, but it is possible to perform Live Classification using a smartphone before deployment on our device. Go to the Live Classification section and click on Connect a Development board:


MJRoBot (Marcelo Rovai) / XIAO-ESP32S3-KWS
Connect a development board
Classify existing test sample

Point your phone to the barcode and select the link.

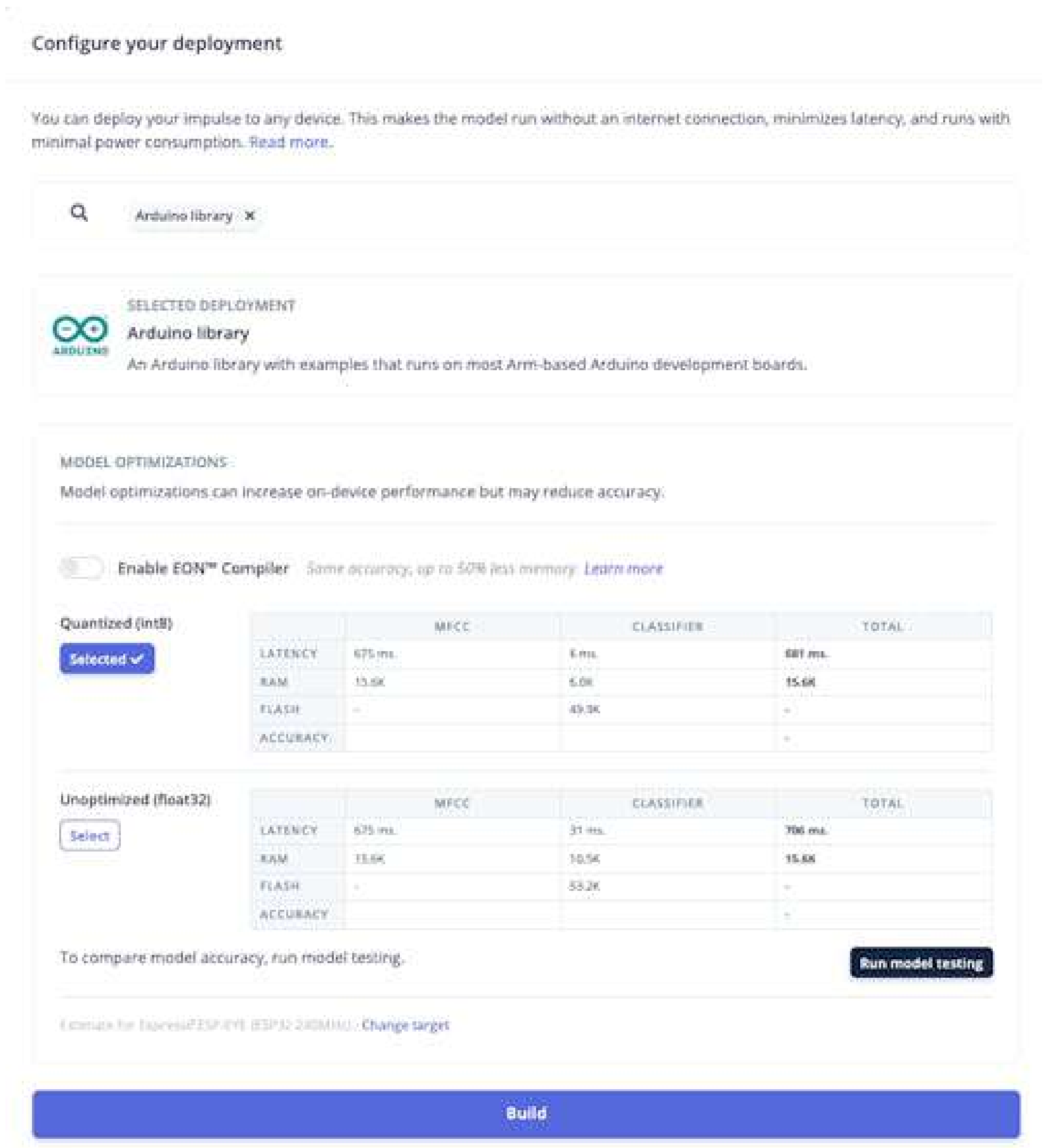


Your phone will be connected to the Studio. Select the option Classification on the app, and when it is running, start testing your keywords, confirming that the model is working with live and real data:



Deploy and Inference

The Studio will package all the needed libraries, preprocessing functions, and trained models, downloading them to your computer. Select the Arduino Library option, then choose Quantized (Int8) from the bottom menu and press Build.



Configure your deployment

You can deploy your impulse to any device. This makes the model run without an internet connection, minimizes latency, and runs with minimal power consumption. Read more.

Arduino library ✕

SELECTED DEPLOYMENT

Arduino library

An Arduino library with examples that runs on most Arm-based Arduino development boards.

MODEL OPTIMIZATIONS

Model optimizations can increase on-device performance but may reduce accuracy.

Enable EON™ Compiler Same accuracy, up to 50% less memory. Learn more

Quantized (int8) Selected ✓

	MFCC	CLASSIFIER	TOTAL
LATENCY	675 ms.	6 ms.	681 ms.
RAM	15.6K	6.0K	15.6K
FLASH	-	49.3K	-
ACCURACY			-

Unoptimized (float32) Select

	MFCC	CLASSIFIER	TOTAL
LATENCY	675 ms.	31 ms.	706 ms.
RAM	15.6K	10.5K	15.6K
FLASH	-	53.2K	-
ACCURACY			-

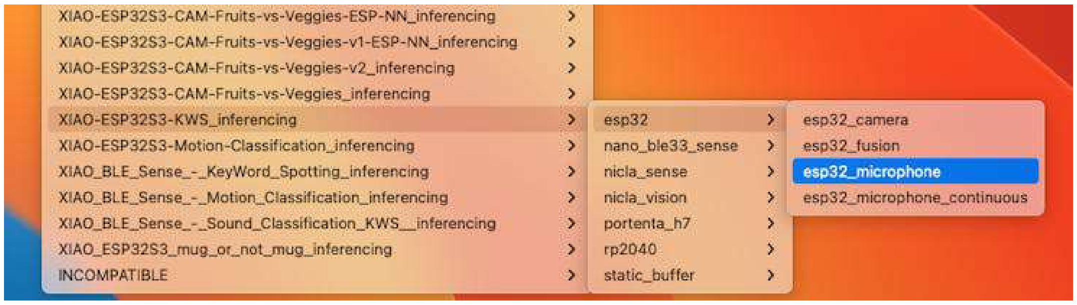
To compare model accuracy, run model testing. Run model testing

Estimate for Espressif ESP-EYE (ESP32 240MHz) · Change target

Build

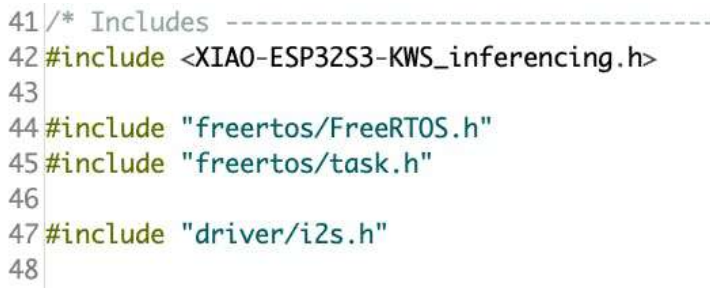
Now it is time for a real test. We will make inferences wholly disconnected from the Studio. Let's change one of the ESP32 code examples created when you deploy the Arduino Library.

In your Arduino IDE, go to the File/Examples tab look for your project, and select esp32/esp32_microphone:


XIAO-ESP32S3-CAM-Fruits-vs-Veggies-ESP-NN_inferencing
XIAO-ESP32S3-CAM-Fruits-vs-Veggies-v1-ESP-NN_inferencing
XIAO-ESP32S3-CAM-Fruits-vs-Veggies-v2_inferencing
XIAO-ESP32S3-CAM-Fruits-vs-Veggies_inferencing
XIAO-ESP32S3-KWS_inferencing
XIAO-ESP32S3-Motion-Classification_inferencing
XIAO_BLE_Sense_-_KeyWord_Spotting_inferencing
XIAO_BLE_Sense_-_Motion_Classification_inferencing
XIAO_BLE_Sense_-_Sound_Classification_KWS__inferencing
XIAO_ESP32S3_mug_or_not_mug_inferencing
INCOMPATIBLE
esp32
nano_ble33_sense
nicla_sense
nicla_vision
portenta_h7
rp2040
static_buffer
esp32_camera
esp32_fusion
esp32_microphone
esp32_microphone_continuous

This code was created for the ESP-EYE built-in microphone, which should be adapted for our device.

Start changing the libraries to handle the I2S bus:



```
/* Includes ---------------------------------
#include <XIAO-ESP32S3-KWS_inferencing.h>

#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

#include "driver/i2s.h"

```

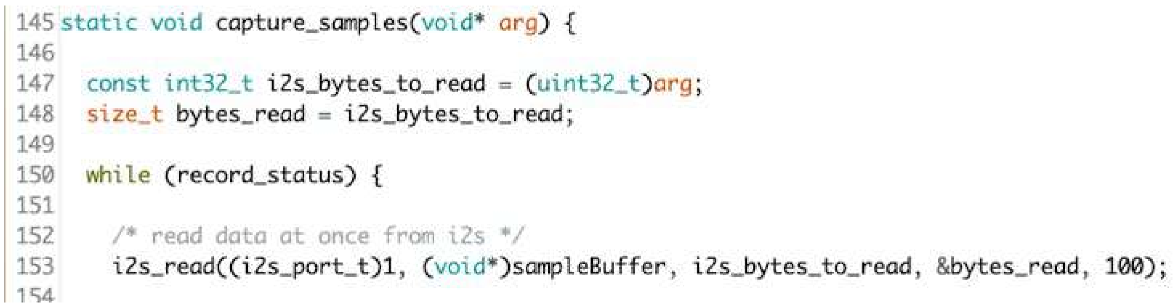
By:

```
#include <I2S.h>
#define SAMPLE_RATE 16000U
#define SAMPLE_BITS 16
```

Initialize the IS2 microphone at setup(), including the lines:

```
void setup()
{
...
    I2S.setAllPins(-1, 42, 41, -1, -1);
    if (!I2S.begin(PDM_MONO_MODE, SAMPLE_RATE, SAMPLE_BITS)) {
      Serial.println("Failed to initialize I2S!");
    while (1) ;
...
}
```

On the static void capture_samples(void* arg) function, replace the line 153 that reads data from I2S mic:

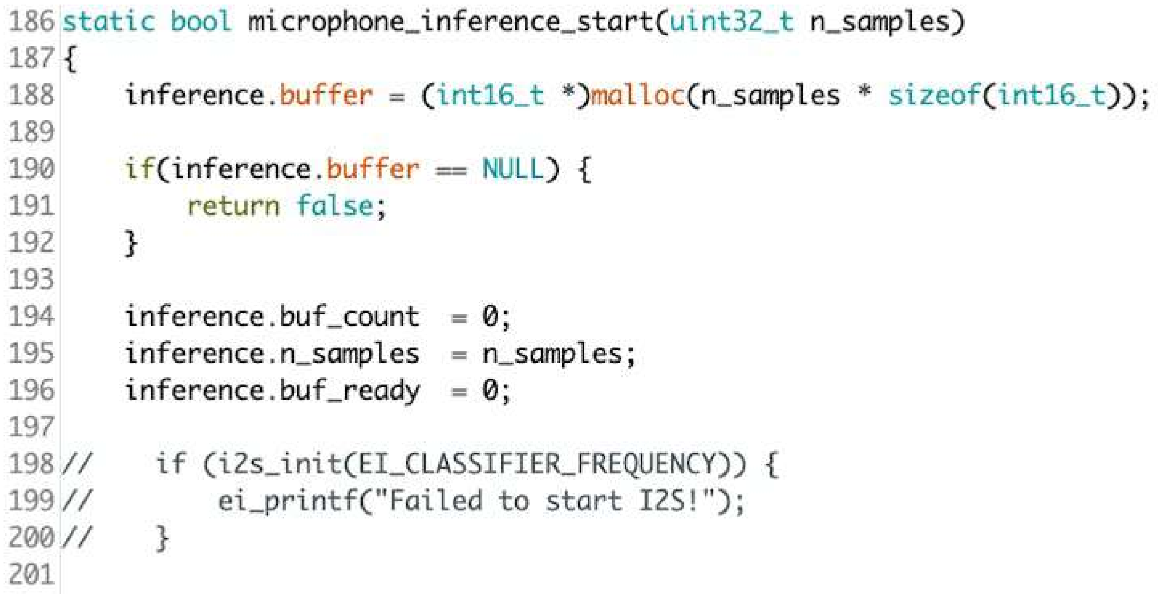


```
static void capture_samples(void* arg) {

  const int32_t i2s_bytes_to_read = (uint32_t)arg;
  size_t bytes_read = i2s_bytes_to_read;

  while (record_status) {

    /* read data at once from i2s */
    i2s_read((i2s_port_t)1, (void*)sampleBuffer, i2s_bytes_to_read, &bytes_read, 100);
```

By:

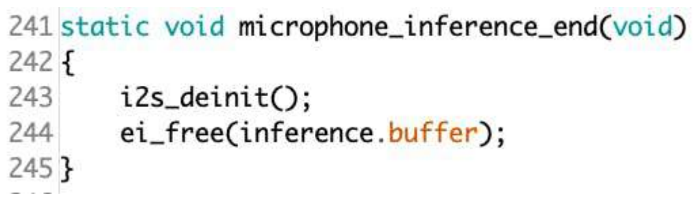
```
/* read data at once from i2s */
esp_i2s::i2s_read(esp_i2s::I2S_NUM_0,
                  (void*)sampleBuffer,
                  i2s_bytes_to_read,
                  &bytes_read, 100);
```

On function static bool microphone_inference_start(uint32_t n_samples), we should comment or delete lines 198 to 200, where the microphone initialization function is called. This is unnecessary because the I2S microphone was already initialized during the setup().



```
186 static bool microphone_inference_start(uint32_t n_samples)
187 {
188     inference.buffer = (int16_t *)malloc(n_samples * sizeof(int16_t));
189
190     if(inference.buffer == NULL) {
191         return false;
192     }
193
194     inference.buf_count  = 0;
195     inference.n_samples  = n_samples;
196     inference.buf_ready  = 0;
197
198 //    if (i2s_init(EI_CLASSIFIER_FREQUENCY)) {
199 //        ei_printf("Failed to start I2S!");
200 //    }
201
```

Finally, on static void microphone_inference_end(void) function, replace line 243:



```
static void microphone_inference_end(void)
{
    i2s_deinit();
    ei_free(inference.buffer);
}
```

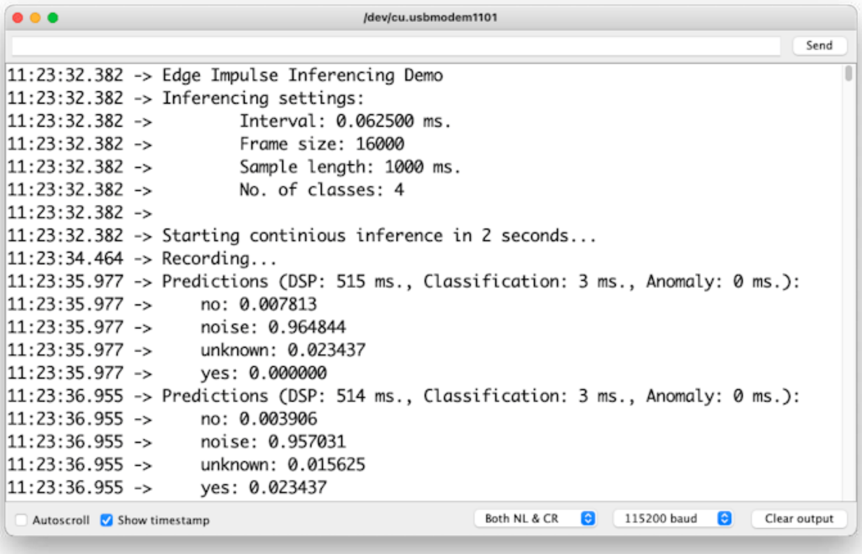
By:

```
static void microphone_inference_end(void)
{
    free(sampleBuffer);
    ei_free(inference.buffer);
}
```

You can find the complete code on the project's GitHub. Upload the sketch to your board and test some real inferences:

Attention

- The Xiao ESP32S3 **MUST** have the PSRAM enabled. You can check it on the Arduino IDE upper menu: `Tools–> PSRAM:OPI PSRAM`
- The Arduino Library (`esp32 by Espressif Systems` should be **version 2.017**. Please do not update it.



```
11:23:32.382 -> Edge Impulse Inferencing Demo
11:23:32.382 -> Inferencing settings:
11:23:32.382 ->         Interval: 0.062500 ms.
11:23:32.382 ->         Frame size: 16000
11:23:32.382 ->         Sample length: 1000 ms.
11:23:32.382 ->         No. of classes: 4
11:23:32.382 ->
11:23:32.382 -> Starting continious inference in 2 seconds...
11:23:34.464 -> Recording...
11:23:35.977 -> Predictions (DSP: 515 ms., Classification: 3 ms., Anomaly: 0 ms.):
11:23:35.977 ->     no: 0.007813
11:23:35.977 ->     noise: 0.964844
11:23:35.977 ->     unknown: 0.023437
11:23:35.977 ->     yes: 0.000000
11:23:36.955 -> Predictions (DSP: 514 ms., Classification: 3 ms., Anomaly: 0 ms.):
11:23:36.955 ->     no: 0.003906
11:23:36.955 ->     noise: 0.957031
11:23:36.955 ->     unknown: 0.015625
11:23:36.955 ->     yes: 0.023437
```

Postprocessing

In edge AI applications, the inference result is only as valuable as our ability to act upon it. While serial output provides detailed information for debugging and development, real-world deployments require immediate, human-readable feedback that doesn't depend on external monitors or connections.

Let's explore two post-processing approaches. Using the internal XIAO's LED and the OLED on the XIAOML Kit.

With LED

Now that we know the model is working by detecting our keywords, let's modify the code to see the internal LED go on every time a YES is detected.

You should initialize the LED:

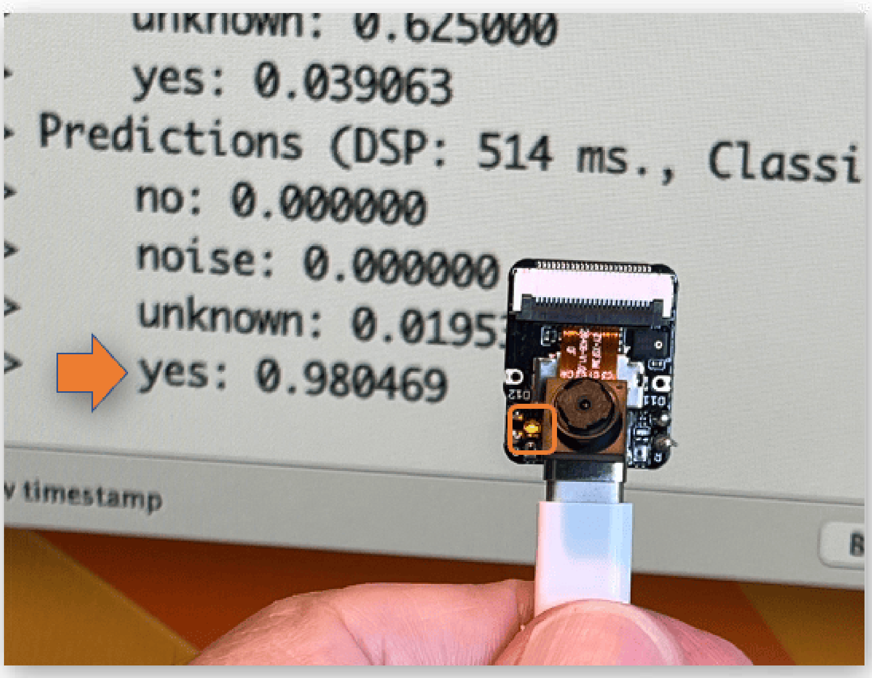
```
#define LED_BUILT_IN 21
...
void setup()
{
...
  pinMode(LED_BUILT_IN, OUTPUT); // Set the pin as output
  digitalWrite(LED_BUILT_IN, HIGH); //Turn off
...
}
```

And change the // print the predictions portion of the previous code (on loop():

```
int pred_index = 0;     // Initialize pred_index
float pred_value = 0;   // Initialize pred_value

// print the predictions
ei_printf("Predictions ");
ei_printf("(DSP: %d ms., Classification: %d ms., Anomaly: %d ms.)",
     result.timing.dsp, result.timing.classification,
     result.timing.anomaly);
ei_printf(": \n");
for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
      ei_printf("    %s: ", result.classification[ix].label);
      ei_printf_float(result.classification[ix].value);
      ei_printf("\n");
```

```
        if (result.classification[ix].value > pred_value){
           pred_index = ix;
           pred_value = result.classification[ix].value;
        }
  }

  // show the inference result on LED
  if (pred_index == 3){
      digitalWrite(LED_BUILT_IN, LOW); //Turn on
  }
  else{
     digitalWrite(LED_BUILT_IN, HIGH); //Turn off
  }
```

You can find the complete code on the project's GitHub. Upload the sketch to your board and test some real inferences:



```
    unknown: 0.625000
    yes: 0.039063
Predictions (DSP: 514 ms., Classi
    no: 0.000000
    noise: 0.000000
    unknown: 0.0195
    yes: 0.980469
```

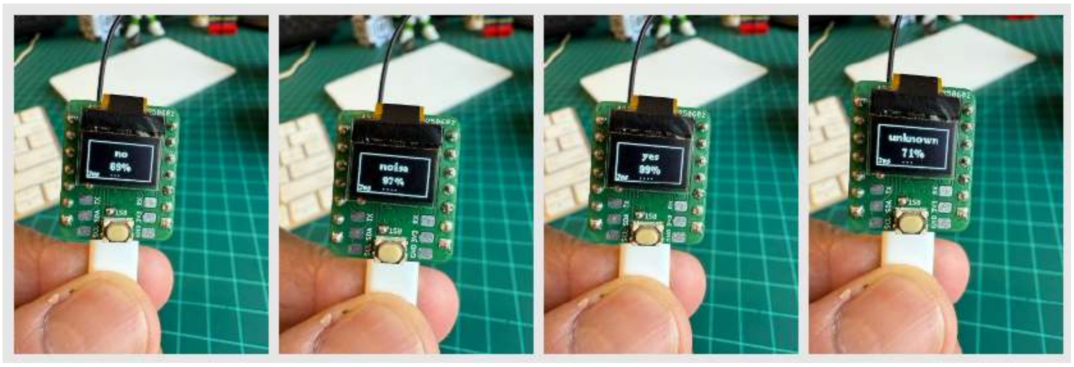
As shown in the video, the idea is that the LED will turn ON whenever the keyword YES is detected. Similarly, instead of turning on an LED, this could serve as a "trigger" for an external device, as we saw in the introduction.

With OLED Display

The XIAOML Kit tiny 0.42" OLED display (72×40 pixels) serves as a crucial post-processing component that transforms raw ML inference results into immediate, human-readable feedback—displaying detected class names and confidence levels directly on the device, eliminating the need for external monitors and enabling truly standalone edge AI deployment in industrial, agricultural, or retail environments where instant visual confirmation of AI predictions is essential.

So, let's modify the sketch to automatically adapt to the model trained on Edge Impulse by reading the class names and count directly from the model. Download the code from GitHub: xiaoml-kit_kws_oled.

Running the code, we can see the result:



Summary

This lab demonstrated the complete development cycle of a keyword spotting system using the XIAOML Kit, showcasing how modern TinyML platforms make sophisticated audio AI accessible on resource-constrained devices. Through hands-on implementation, we've bridged the gap between theoretical machine learning concepts and practical embedded AI deployment.

Technical Achievements:

The project successfully implemented a complete audio processing pipeline from raw sound capture through real-time inference. Using the XIAO ESP32S3's integrated digital microphone, we captured audio data at professional quality (16kHz/16-bit) and processed it using Mel Frequency Cepstral Coefficients (MFCC) for feature extraction. The deployed CNN model achieved excellent accuracy in distinguishing between our target keywords ("YES", "NO") and background conditions ("NOISE", "UNKNOWN"), with inference times suitable for real-time applications.

Platform Integration:

Edge Impulse Studio proved invaluable as a comprehensive MLOps platform for embedded systems, handling everything from data collection and labeling through model training, optimization, and deployment. The seamless integration between cloud-based training and edge deployment exemplifies modern TinyML workflows, while the Arduino IDE provided the flexibility needed for custom post-processing implementations.

Real-World Applications:

The techniques learned extend far beyond simple keyword detection. Voice-activated control systems, industrial safety monitoring through sound classification, medical applications for respiratory analysis, and environmental monitoring for wildlife or equipment sounds all leverage similar audio processing approaches. The cascaded detection architecture demonstrated here—using edge-based KWS to trigger more complex cloud processing—is fundamental to modern voice assistant systems.

Embedded AI Principles:

This project highlighted crucial TinyML considerations, including power management, memory optimization through PSRAM utilization, and the trade-offs between model complexity and inference speed. The successful deployment of a neural network performing real-time audio analysis on a microcontroller demonstrates how AI capabilities, once requiring powerful desktop computers, can now operate on battery-powered devices.

Development Methodology:

We explored multiple development pathways, from data collection strategies (offline SD card storage versus online streaming) to deployment options (Edge Impulse's automated library generation versus custom Arduino implementation). This flexibility is crucial for adapting to various project requirements and constraints.

Future Directions:

The foundation established here enables the exploration of more advanced audio AI applications. Multi-keyword recognition, speaker identification, emotion detection from voice, and environmental sound classification all build upon the same core techniques. The integration capabilities demonstrated with OLED displays and GPIO control illustrate how KWS can serve as the intelligent interface for broader IoT systems.

Consider that Sound Classification encompasses much more than just voice recognition. This project's techniques apply across numerous domains:

- **Security Applications**: Broken glass detection, intrusion monitoring, gunshot detection
- **Industrial IoT**: Machinery health monitoring, anomaly detection in manufacturing equipment
- **Healthcare**: Sleep disorder monitoring, respiratory condition assessment, elderly care systems

- **Environmental Monitoring**: Wildlife tracking, urban noise analysis, smart building acoustic management
- **Smart Home Integration**: Multi-room voice control, appliance status monitoring through sound signatures

Key Takeaways:

The XIAOML Kit proves that professional-grade AI development is achievable with accessible tools and modest budgets. The combination of capable hardware (ESP32S3 with PSRAM and integrated sensors), mature development platforms (Edge Impulse Studio), and comprehensive software libraries creates an environment where complex AI concepts become tangible, working systems.

This lab demonstrates that the future of AI isn't just in massive data centers but in intelligent edge devices that can process, understand, and respond to their environment in real time—opening possibilities for ubiquitous, privacy-preserving, and responsive artificial intelligence systems.

Resources

- XIAO ESP32S3 Codes
- XIAOML Kit Code
- Subset of Google Speech Commands Dataset
- KWS Feature Engineering
- KWS MFCC Analysis Colab Notebook
- KWS CNN training Colab Notebook
- XIAO ESP32S3 Post-processing Code
- Edge Impulse Project

Motion Classification and Anomaly Detection

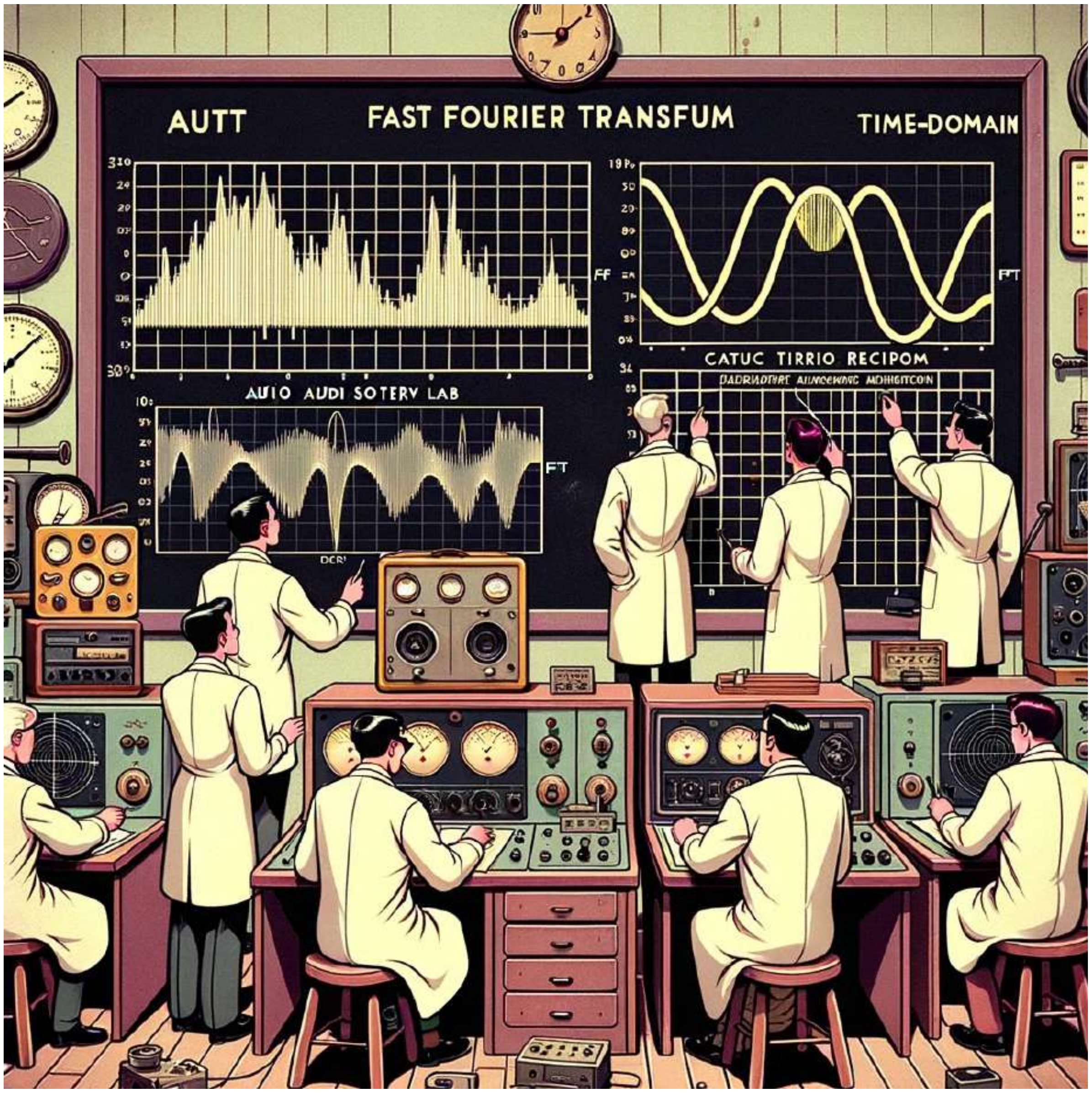


Figure 6: DALL · E prompt - 1950s style cartoon illustration set in a vintage audio lab. Scientists, dressed in classic attire with white lab coats, are intently analyzing audio data on large chalkboards. The boards display intricate FFT (Fast Fourier Transform) graphs and time-domain curves. Antique audio equipment is scattered around, but the data representations are clear and detailed, indicating their focus on audio analysis.

Overview

Transportation is the backbone of global commerce. Millions of containers are transported daily by ships, trucks, and trains to destinations worldwide. Ensuring the safe and efficient transit of these containers is a monumental task that requires leveraging modern technology, and TinyML is undoubtedly a key solution.

In this hands-on lab, we will work to solve real-world transportation problems. We will develop a Motion Classification and Anomaly Detection system using the XIAOML Kit, the Arduino IDE, and the Edge Impulse Studio. This project will help us understand how containers experience different forces and motions during various phases of transportation, including terrestrial and maritime transit, vertical movement via forklifts, and periods of stationary storage in warehouses.

Learning Objectives

- Setting up the XIAOML Kit
- Data Collection and Preprocessing
- Building the Motion Classification Model
- Implementing Anomaly Detection
- Real-world Testing and Analysis

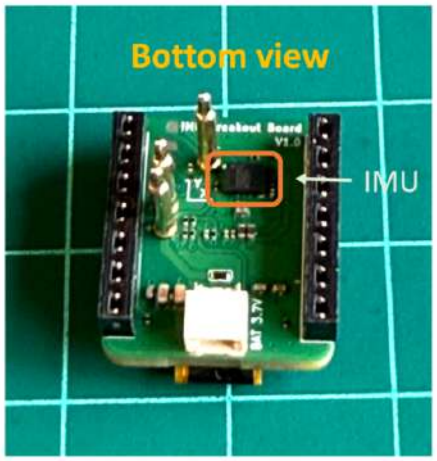
By the end of this lab, you'll have a working prototype that can classify different types of motion and detect anomalies during the transportation of containers. This knowledge can serve as a stepping stone to more advanced projects in the burgeoning field of TinyML, particularly those involving vibration.

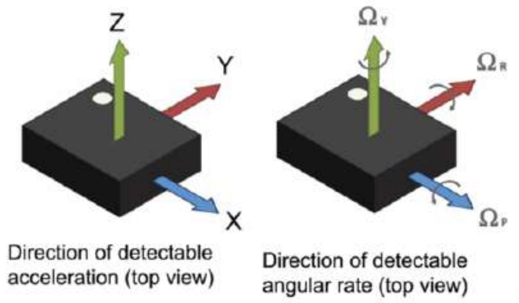
Installing the IMU

The XIAOML Kit comes with a built-in LSM6DS3TR-C 6-axis IMU sensor on the expansion board, eliminating the need for external sensor connections. This integrated approach offers a clean and reliable platform for motion-based machine learning applications.

The LSM6DS3TR-C combines a 3-axis accelerometer and 3-axis gyroscope in a single package, connected via I2C to the XIAO ESP32S3 at address 0x6A that provides:

- **Accelerometer ranges**: ±2/±4/±8/±16 g (we'll use ±2g by default)
- **Gyroscope ranges**: ±125/±250/±500/±1000/±2000 dps (we'll use ±250 dps by default)
- **Resolution**: 16-bit ADC
- **Communication**: I2C interface at address 0x6A
- **Power**: Ultra-low power design




Z
Y
X
Direction of detectable acceleration (top view)
Ω_Y
Ω_R
Ω_P
Direction of detectable angular rate (top view)

Coordinate System: The sensor operates within a right-handed coordinate system. When looking at the expansion board from the bottom (where you can see the IMU sensor with the point mark):

- **X-axis**: Points to the right
- **Y-axis**: Points forward (away from you)
- **Z-axis**: Points upward (out of the board)

Setting Up the Hardware

Since the XIAOML Kit comes pre-assembled with the expansion board, no additional hardware connections are required. The LSM6DS3TR-C IMU is already properly connected via I2C.

What's Already Connected:

- LSM6DS3TR-C IMU → I2C (SDA/SCL) → XIAO ESP32S3
- I2C Address: 0x6A
- Power: 3.3V from XIAO ESP32S3

Required Library: You should have the library installed during the Setup. If not, install the Seeed Arduino LSM6DS3 library following the steps:

1. Open Arduino IDE Library Manager
2. Search for "LSM6DS3"
3. Install **"Seeed Arduino LSM6DS3"** by Seeed Studio
4. **Important**: Do NOT install "Arduino_LSM6DS3 by Arduino" - that's for different boards!

Testing the IMU Sensor

Let's start with a simple test to verify the IMU is working correctly. Upload this code to test the sensor:

```
#include <LSM6DS3.h>
#include <Wire.h>

// Create IMU object using I2C interface
LSM6DS3 myIMU(I2C_MODE, 0x6A);

float accelX, accelY, accelZ;
float gyroX, gyroY, gyroZ;

void setup() {
  Serial.begin(115200);
  while (!Serial) delay(10);

  Serial.println("XIAOML Kit IMU Test");
  Serial.println("LSM6DS3TR-C 6-Axis IMU");
  Serial.println("====================");

  // Initialize the IMU
  if (myIMU.begin() != 0) {
      Serial.println("ERROR: IMU initialization failed!");
      while(1) delay(1000);
  } else {
      Serial.println("  IMU initialized successfully");
      Serial.println("Data Format: AccelX,AccelY,AccelZ,"
                    "GyroX,GyroY,GyroZ");
      Serial.println("Units: g-force, degrees/second");
      Serial.println();
  }
}

void loop() {
  // Read accelerometer data (in g-force)
  accelX = myIMU.readFloatAccelX();
  accelY = myIMU.readFloatAccelY();
  accelZ = myIMU.readFloatAccelZ();

  // Read gyroscope data (in degrees per second)
```

```
    gyroX = myIMU.readFloatGyroX();
    gyroY = myIMU.readFloatGyroY();
    gyroZ = myIMU.readFloatGyroZ();

    // Print readable format
    Serial.print("Accel (g): X="); Serial.print(accelX, 3);
    Serial.print(" Y="); Serial.print(accelY, 3);
    Serial.print(" Z="); Serial.print(accelZ, 3);
    Serial.print(" | Gyro (°/s): X="); Serial.print(gyroX, 2);
    Serial.print(" Y="); Serial.print(gyroY, 2);
    Serial.print(" Z="); Serial.println(gyroZ, 2);

    delay(100); // 10 Hz update rate
  }
```

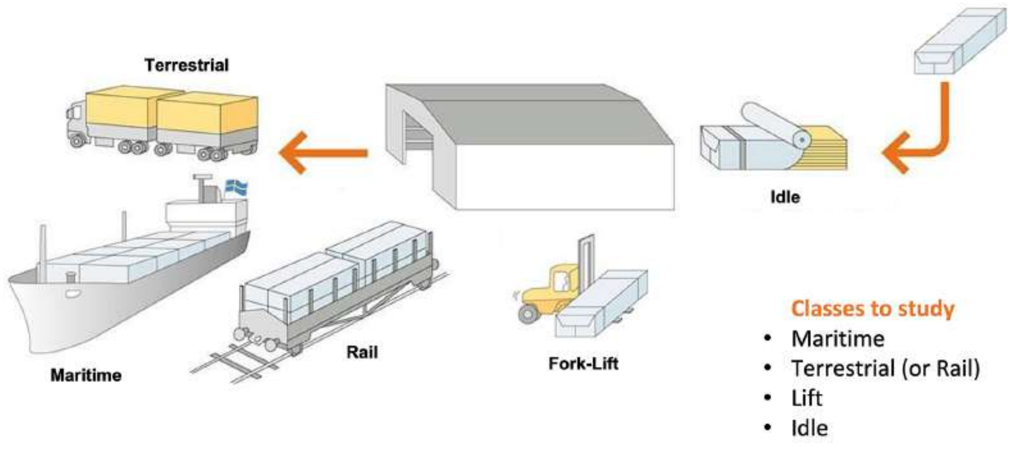
When the kit is resting flat on a table, you should see:

- Z-axis acceleration around +1.0g (gravity)
- X and Y acceleration near 0.0g
- All gyroscope values near 0.0°/s

Move the kit around to see the values change accordingly.

The TinyML Motion Classification Project

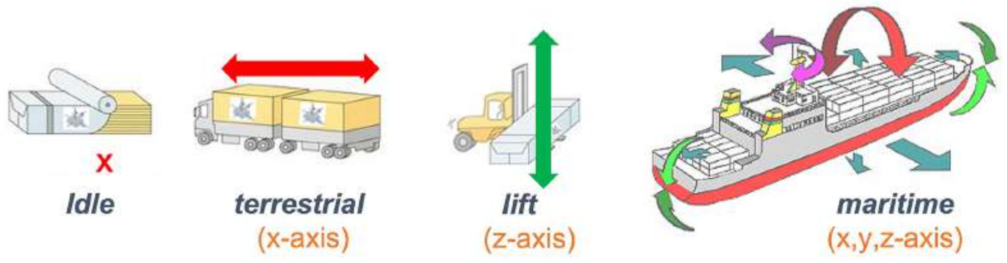
We will simulate container (or, more accurately, package) transportation through various scenarios to make this tutorial more relatable and practical.



Using the accelerometer of the XIAOML Kit, we'll capture motion data by manually simulating the conditions of:

- **Maritime** (pallets on boats) - Movement in all axes with wave-like patterns
- **Terrestrial** (pallets on trucks/trains) - Primarily horizontal movement
- **Lift** (pallets being moved by forklift) - Primarily vertical movement
- **Idle** (pallets in storage) - Minimal movement

From the above image, we can define for our simulation that primarily horizontal movements (x or y axis) should be associated with the "Terrestrial class." Vertical movements (z-axis) with the "Lift Class," no activity with the "Idle class," and movement on all three axes to Maritime class.



Data Collection

For data collection, we have several options available. In a real-world scenario, we can connect our device directly to one container and store the collected data in a file (e.g., CSV) on an SD card. Data can also be sent remotely to a nearby repository, such as a mobile phone, using Wi-Fi or Bluetooth (as demonstrated in this project: Sensor DataLogger). Once your dataset is collected and stored as a .CSV file, we can upload it to the Studio using the CSV Wizard tool.

> This video shows alternative ways to send data to the Edge Impulse Studio.

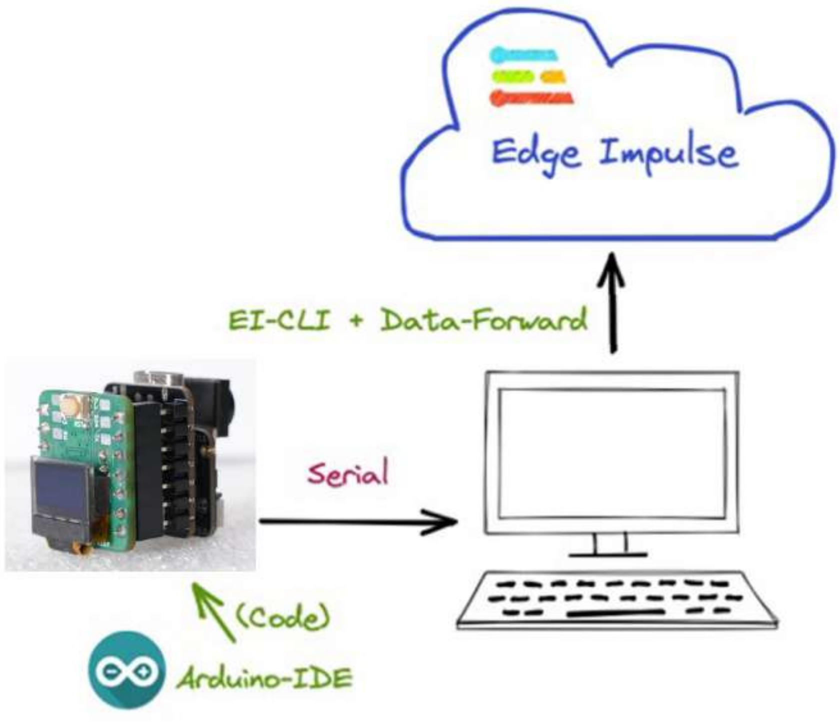
Preparing the Data Collection Code

In this lab, we will connect the Kit directly to the Edge Impulse Studio, which will also be used for data pre-processing, model training, testing, and deployment.

For data collection, we should first connect the Kit to Edge Impulse Studio, which will also be used for data pre-processing, model training, testing, and deployment.

> Follow the instructions here to install Node.js and Edge Impulse CLI on your computer.

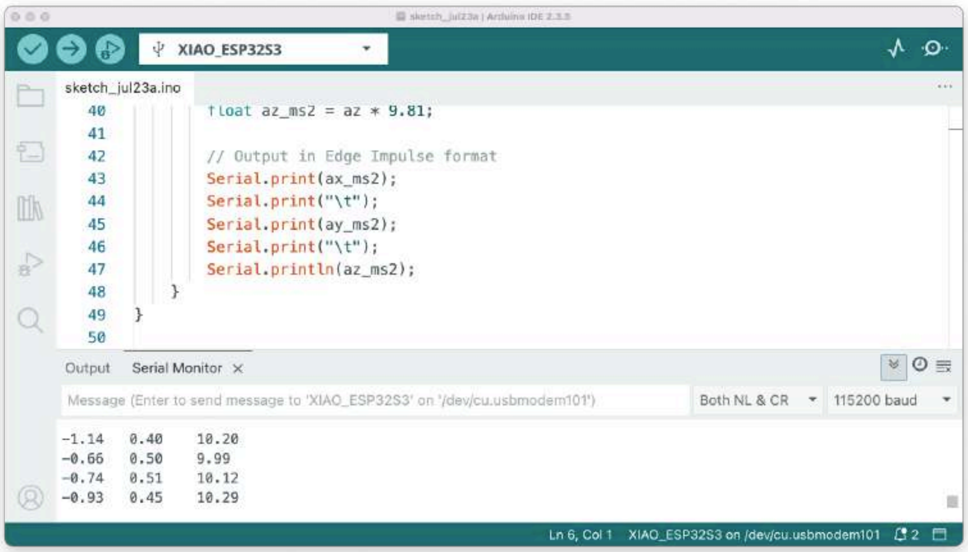
Once the XIAOML Kit is not a fully supported development board by Edge Impulse, we should, for example, use the CLI Data Forwarder to capture data from our sensor and send it to the Studio, as shown in this diagram:



We'll modify our test code to output data in a format suitable for Edge Impulse:

```
#include <LSM6DS3.h>
#include <Wire.h>

#define FREQUENCY_HZ        50
#define INTERVAL_MS         (1000 / (FREQUENCY_HZ + 1))

LSM6DS3 myIMU(I2C_MODE, 0x6A);
static unsigned long last_interval_ms = 0;

void setup() {
  Serial.begin(115200);
  while (!Serial) delay(10);

  Serial.println("XIAOML Kit - Motion Data Collection");
  Serial.println("LSM6DS3TR-C IMU Sensor");
  // Initialize IMU
  if (myIMU.begin() != 0) {
      Serial.println("ERROR: IMU initialization failed!");
      while(1) delay(1000);
  }

  delay(2000);
  Serial.println("Starting data collection in 3 seconds...");
  delay(3000);
}

void loop() {
  if (millis() > last_interval_ms + INTERVAL_MS) {
      last_interval_ms = millis();

      // Read accelerometer data
      float ax = myIMU.readFloatAccelX();
      float ay = myIMU.readFloatAccelY();
      float az = myIMU.readFloatAccelZ();

      // Convert to m/s² (multiply by 9.81)
      float ax_ms2 = ax * 9.81;
      float ay_ms2 = ay * 9.81;
      float az_ms2 = az * 9.81;

      // Output in Edge Impulse format
      Serial.print(ax_ms2);
      Serial.print("\t");
      Serial.print(ay_ms2);
      Serial.print("\t");
      Serial.println(az_ms2);
  }
}
```

Upload the code to the Arduino IDE. We should see the accelerometer values (converted to m/s²) at the Serial Monitor:

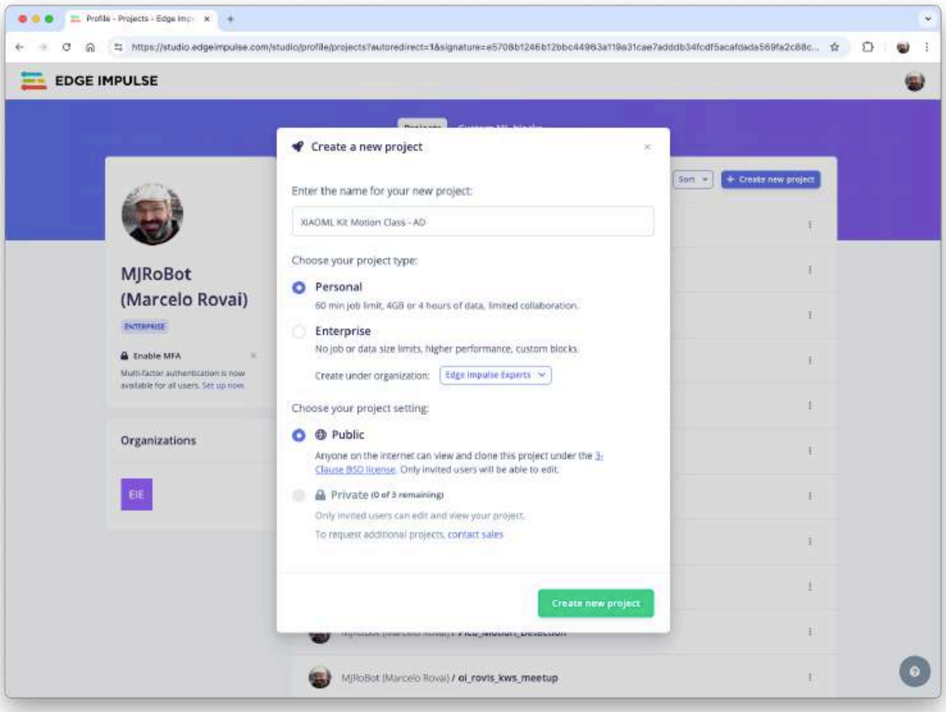

```
            float az_ms2 = az * 9.81;

            // Output in Edge Impulse format
            Serial.print(ax_ms2);
            Serial.print("\t");
            Serial.print(ay_ms2);
            Serial.print("\t");
            Serial.println(az_ms2);
        }
    }
```

```
-1.14   0.40    10.20
-0.66   0.50    9.99
-0.74   0.51    10.12
-0.93   0.45    10.29
```

Keep the code running, but **turn off the Serial Monitor**. The data generated by the Kit will be sent to the Edge Impulse Studio via Serial Connection.

Connecting to Edge Impulse for Data Collection

Create an Edge Impulse Project - Go to Edge Impulse Studio and create a new project - Choose a descriptive name (keep under 63 characters for Arduino library compatibility)



Create a new project

Enter the name for your new project:

XIAOML Kit Motion Class - AD

Choose your project type:

[✓ Personal] 60 min job limit, 4GB or 4 hours of data, limited collaboration.

Enterprise No job or data size limits, higher performance, custom blocks.

Create under organization: Edge Impulse Experts

Choose your project setting:

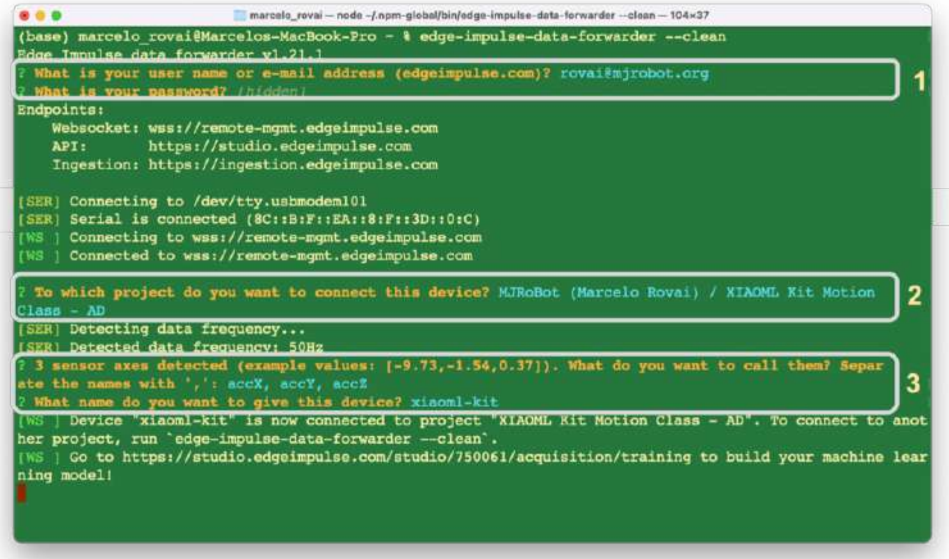
[✓ Public] Anyone on the internet can view and clone this project under the 3-Clause BSD license. Only invited users will be able to edit.

Private (0 of 3 remaining) Only invited users can edit and view your project.

To request additional projects, contact sales

Create new project

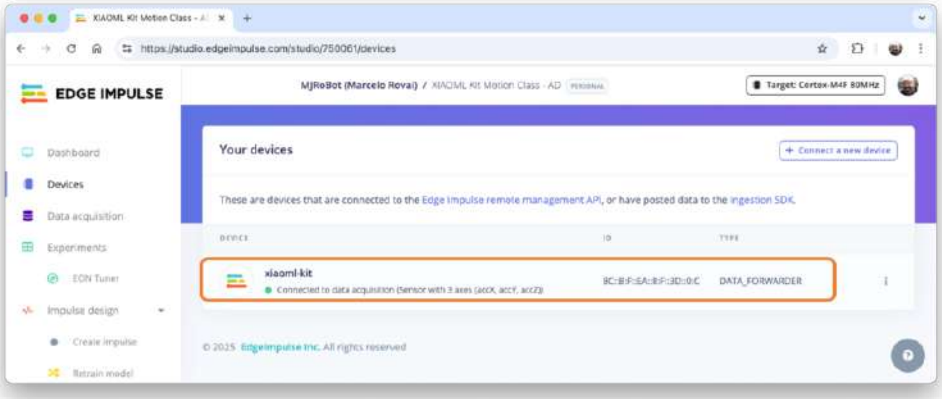
Set up CLI Data Forwarder - Install Edge Impulse CLI on your computer - Confirm that the XIAOML Kit is connected to the computer, **the code is running and the Serial Monitor is OFF**, otherwise we can get an error. - On the Computer Terminal, run: `edge-impulse-data-forwarder --clean` - Enter your Edge Impulse credentials - Select your project and configure device settings



marcelo_rovai — node ~/.npm-global/bin/edge-impulse-data-forwarder --clean — 104×37
1
2
3

```
(base) marcelo_rovai@Marcelos-MacBook-Pro ~ % edge-impulse-data-forwarder --clean
Edge Impulse data forwarder v1.21.1
? What is your user name or e-mail address (edgeimpulse.com)? rovai@mjrobot.org
? What is your password? [hidden]
Endpoints:
    Websocket: wss://remote-mgmt.edgeimpulse.com
    API:       https://studio.edgeimpulse.com
    Ingestion: https://ingestion.edgeimpulse.com

[SER] Connecting to /dev/tty.usbmodem101
[SER] Serial is connected (8C::B:F::EA::8:F::3D::0:C)
[WS ] Connecting to wss://remote-mgmt.edgeimpulse.com
[WS ] Connected to wss://remote-mgmt.edgeimpulse.com

? To which project do you want to connect this device? MJRoBot (Marcelo Rovai) / XIAOML Kit Motion
Class - AD
[SER] Detecting data frequency...
[SER] Detected data frequency: 50Hz
? 3 sensor axes detected (example values: [-9.73,-1.54,0.37]). What do you want to call them? Separ
ate the names with ',': accX, accY, accZ
? What name do you want to give this device? xiaoml-kit
[WS ] Device "xiaoml-kit" is now connected to project "XIAOML Kit Motion Class - AD". To connect to anot
her project, run `edge-impulse-data-forwarder --clean`.
[WS ] Go to https://studio.edgeimpulse.com/studio/750061/acquisition/training to build your machine lear
ning model!
```

- Go to the Edge Impulse Studio Project. On the `Device` section is possible to verify if the kit is correctly connected (the dot should be green).



Your devices

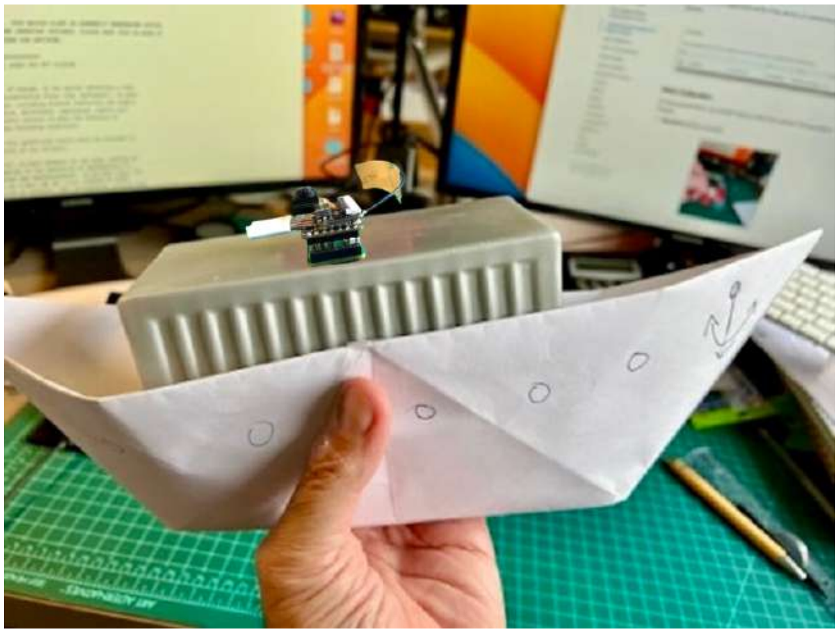
These are devices that are connected to the Edge Impulse remote management API, or have posted data to the ingestion SDK.

DEVICE	ID	TYPE
xiaoml-kit ● Connected to data acquisition (Sensor with 3 axes (accX, accY, accZ))	8C::B:F::EA::8:F::3D::0:C	DATA_FORWARDER

© 2025 EdgeImpulse Inc. All rights reserved

Data Collection at the Studio

As discussed before, we should capture data from all four **Transportation Classes.** Imagine that you have a container with a built-in accelerometer (In this case, our XIAOML Kit). Now imagine your container is on a boat, facing an angry ocean:



Or in a Truck, travelling on a road, or being moved with a forklift, etc.

Movement Simulation

Maritime Class:

- Hold the kit and simulate boat movement
- Move in all three axes with wave-like, undulating motions
- Include gentle rolling and pitching movements

Terrestrial Class:

- Move the kit horizontally in straight lines (left to right and vice versa)
- Simulate truck/train vibrations with small horizontal shakes
- Occasional gentle bumps and turns

Lift Class:

- Move the kit primarily in vertical directions (up and down)
- Simulate forklift operations: up, pause, down
- Include some short horizontal positioning movements

Idle Class:

- Place the kit on a stable surface
- Minimal to no movement

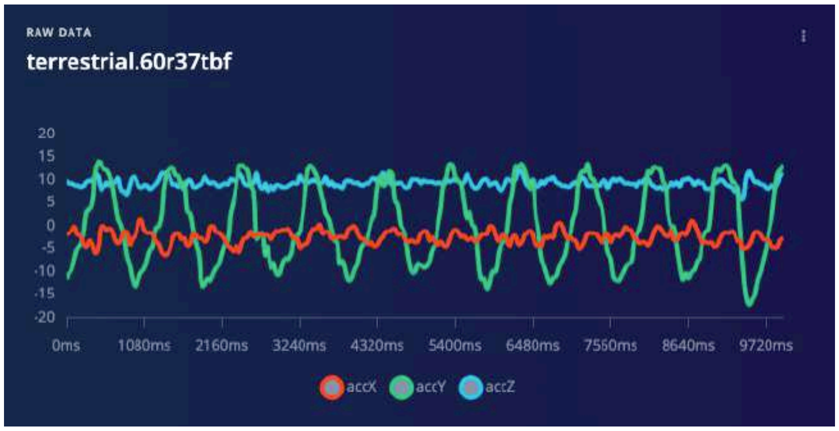
- Capture environmental vibrations and sensor noise

Data Acquisition

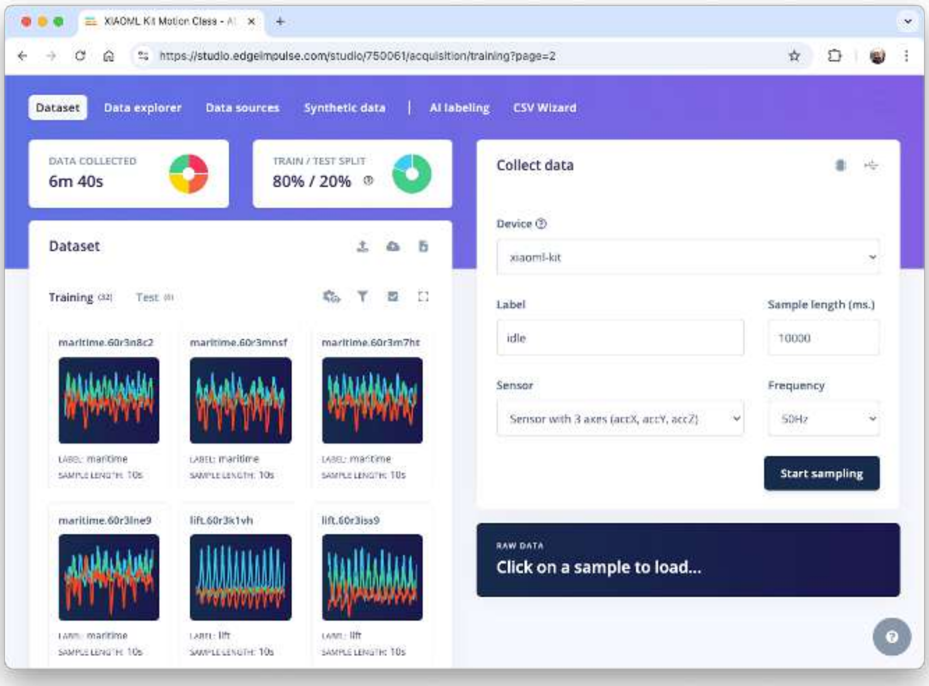
On the `Data Acquisition` section, you should see that your board `[xiaoml-kit]` is connected. The sensor is available: `[sensor with 3 axes (accX, accY, accZ)]` with a sampling frequency of `[50 Hz]`. The Studio suggests a sample length of `[10000]` ms (10 s). The last thing left is defining the sample label. Let's start, for example, with`[terrestrial]`.

Press `[Start Sample]`and move your kit horizontally (left to right), keeping it in one direction. After 10 seconds, our data will be uploaded to the Studio.

Below is one sample (raw data) of 10 seconds of collected data. It is notable that the ondulatory movement predominantly occurs along the Y-axis (left-right). The other axes are almost stationary (the X-axis is centered around zero, and the Z-axis is centered around 9.8 ms^2 due to gravity).



You should capture, for example, around 2 minutes (ten to twelve samples of 10 seconds each) for each of the four classes. Using the `3 dots` after each sample, select two and move them to the **Test set**. Alternatively, you can use the Automatic `Train/Test Split` tool on the **Danger Zone** of the `Dashboard` tab. Below, it is possible to see the result datasets:



DATA COLLECTED: 6m 40s

TRAIN / TEST SPLIT: 80% / 20%

Device: xiaoml-kit

Label: idle

Sample length (ms.): 10000

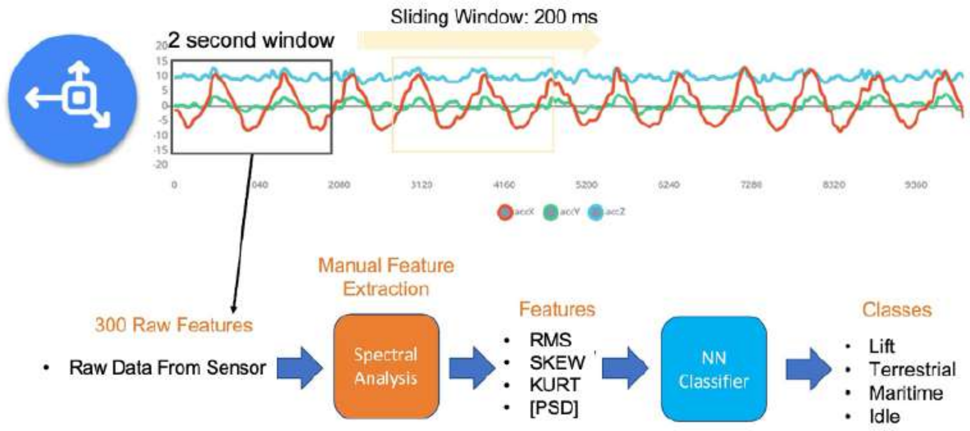
Sensor: Sensor with 3 axes (accX, accY, accZ)

Frequency: 50Hz

maritime.60r3n8c2	maritime.60r3mnsf	maritime.60r3m7ht
LABEL: maritime, SAMPLE LENGTH: 10s	LABEL: maritime, SAMPLE LENGTH: 10s	LABEL: maritime, SAMPLE LENGTH: 10s
maritime.60r3lne9	lift.60r3k1vh	lift.60r3iss9
LABEL: maritime, SAMPLE LENGTH: 10s	LABEL: lift, SAMPLE LENGTH: 10s	LABEL: lift, SAMPLE LENGTH: 10s

Data Pre-Processing

The raw data type captured by the accelerometer is a "time series" and should be converted to "tabular data". We can do this conversion using a sliding window over the sample data. For example, in the below figure,



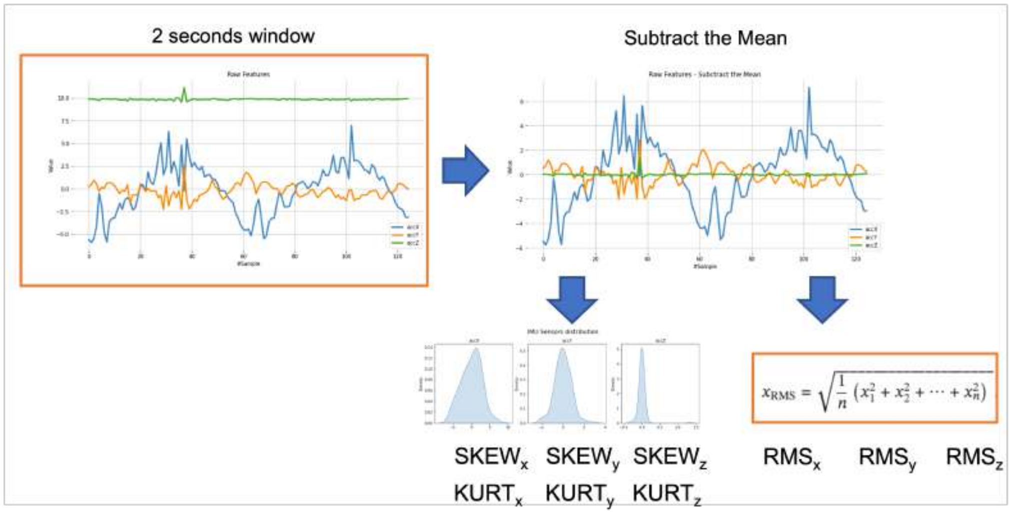
We can see 10 seconds of accelerometer data captured with a sample rate (SR) of 50 Hz. A 2-second window will capture 300 data points (3 axes × 2 seconds × 50 samples). We will slide this window every 200ms, creating a larger dataset where each instance has 300 raw features.

> You should use the best SR for your case, given Nyquist's theorem, which states that a periodic signal must be sampled at least twice the highest frequency component.

Data preprocessing is a challenging area for embedded machine learning. Still, Edge Impulse helps overcome this with its digital signal processing (DSP) preprocessing step and, more specifically, the Spectral Features.

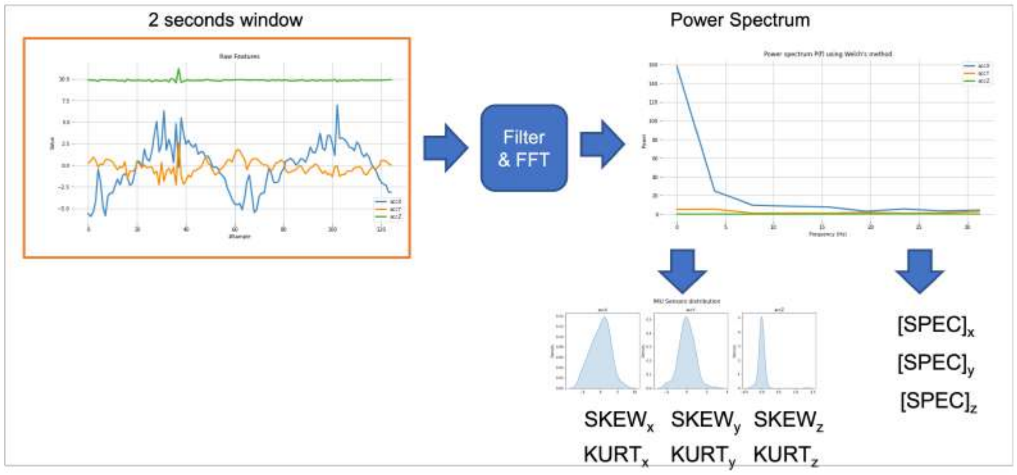
In the Studio, this dataset will be the input to a Spectral Analysis block, which is well-suited to analyzing repetitive motion, such as accelerometer data. This block will perform a DSP (Digital Signal Processing), extracting features such as "FFT" or "Wavelets". In the most common case, FFT, the **Time Domain Statistical features** per axis/channel are:

- RMS
- Skewness
- Kurtosis


2 seconds window
Subtract the Mean
$x_{\text{RMS}} = \sqrt{\frac{1}{n}\left(x_1^2 + x_2^2 + \cdots + x_n^2\right)}$
SKEW$_x$ SKEW$_y$ SKEW$_z$
KURT$_x$ KURT$_y$ KURT$_z$
RMS$_x$ RMS$_y$ RMS$_z$

And the **Frequency Domain Spectral features** per axis/channel are:

- Spectral Power
- Skewness
- Kurtosis



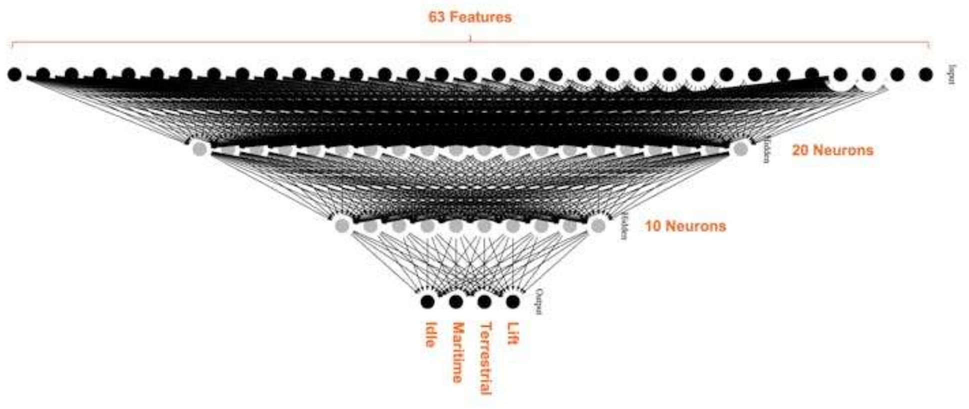
For example, for an FFT length of 32 points, the Spectral Analysis Block's resulting output will be 21 features per axis (a total of 63 features).

Those 63 features will serve as the input tensor for a Neural Network Classifier and the Anomaly Detection model (K-Means).

You can learn more by digging into the lab DSP Spectral Features

Model Design

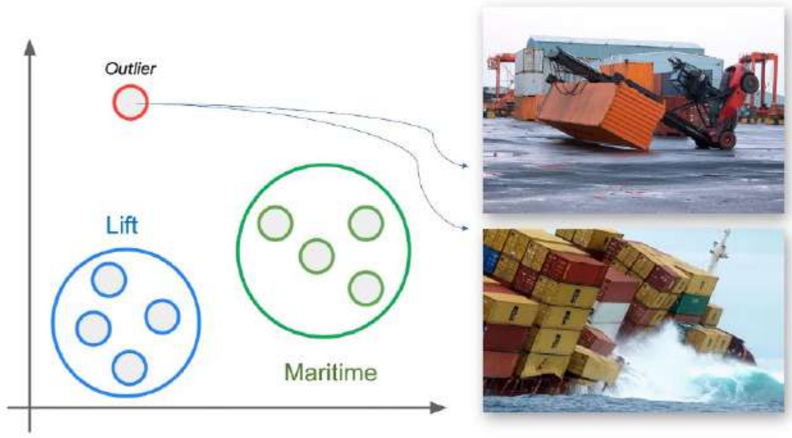
Our classifier will be a Dense Neural Network (DNN) that will have 63 neurons on its input layer, two hidden layers with 20 and 10 neurons, and an output layer with four neurons (one per each class), as shown here:



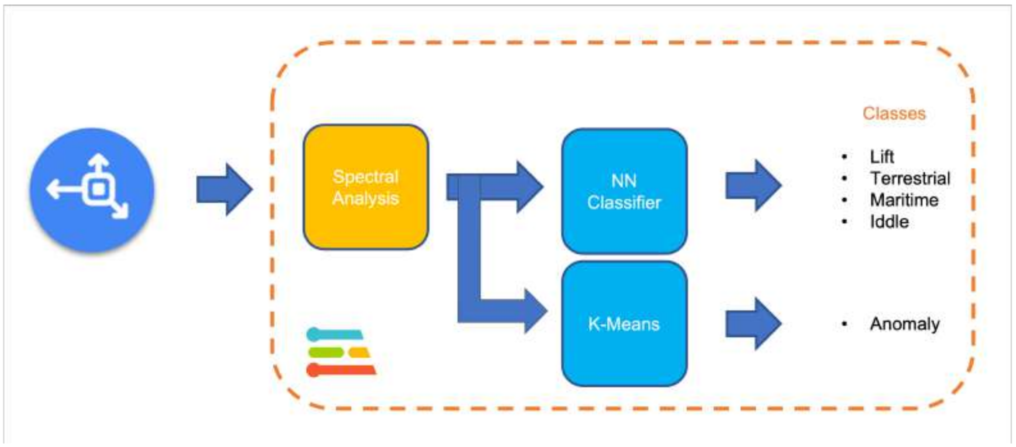
Impulse Design

An impulse takes raw data, uses signal processing to extract features, and then uses a learning block (**Dense model**) to classify new data.

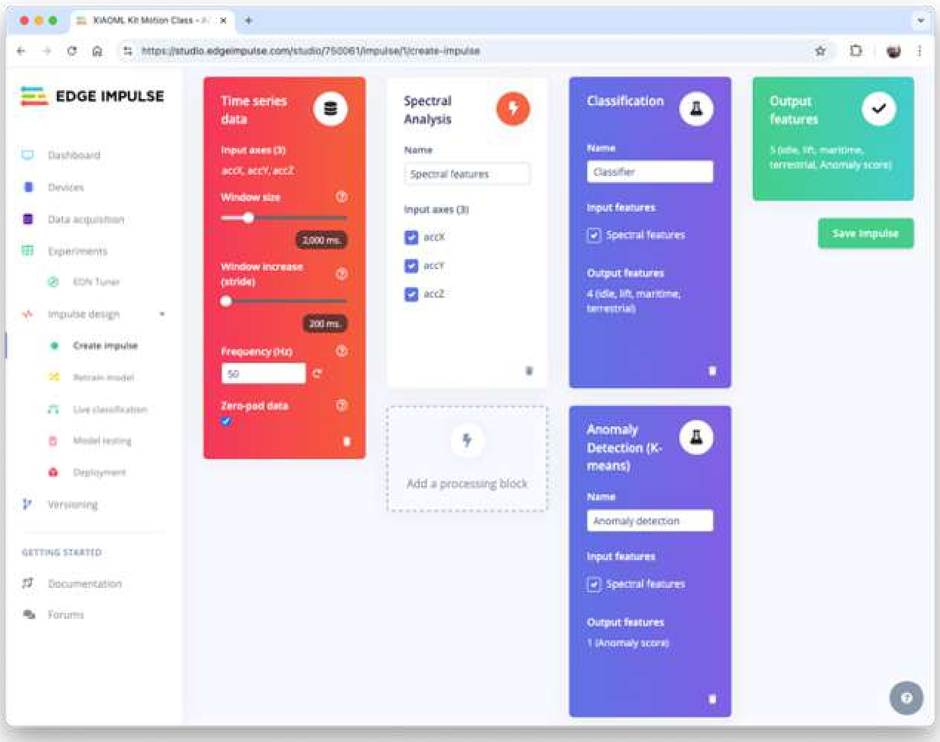
We also utilize a second model, the **K-means**, which can be used for Anomaly Detection. If we imagine that we could have our known classes as clusters, any sample that cannot fit into one of these clusters could be an outlier, an anomaly (for example, a container rolling out of a ship on the ocean or being upside down on the floor).



Imagine our XIAOML Kit rolling or moving upside-down, on a movement complement different from the one trained on.



Below the final Impulse design:



https://studio.edgeimpulse.com/studio/750061/impulse/1/create-impulse

Time series data
Input axes (3)
accX, accY, accZ
Window size: 2,000 ms.
Window increase (stride): 200 ms.
Frequency (Hz): 50
Zero-pad data ✓

Spectral Analysis
Name: Spectral features
Input axes (3)
✓ accX
✓ accY
✓ accZ

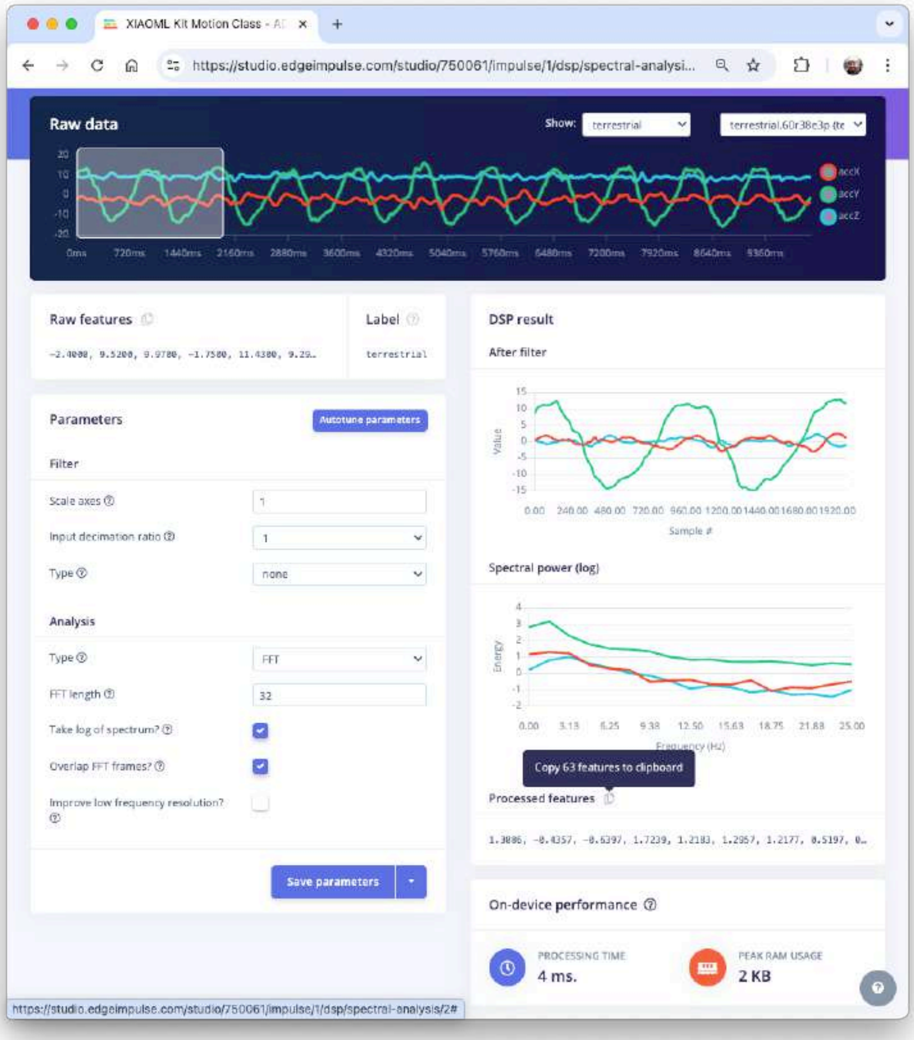
Classification
Name: Classifier
Input features
✓ Spectral features
Output features
4 (idle, lift, maritime, terrestrial)

Anomaly Detection (K-means)
Name: Anomaly detection
Input features
✓ Spectral features
Output features
1 (Anomaly score)

Output features
5 (idle, lift, maritime, terrestrial, Anomaly score)

Generating features

At this point in our project, we have defined the preprocessing method and designed the model. Now, it is time to have the job done. First, let's convert the raw data (time-series type) into tabular data. Go to the `Spectral Features` tab and select `[Save Parameters]`. Alternatively, instead of using the default values, we can select the `[Autotune parameters]` button. In this case, the Studio will define new hyperparameters, such as the filter design and FFT length, based on the raw data.


XIAOML Kit Motion Class - AI
https://studio.edgeimpulse.com/studio/750061/impulse/1/dsp/spectral-analysi...
Raw data
Show: terrestrial
terrestrial.60r38e3p
accX
accY
accZ
DSP result
After filter
Value
Sample #
Spectral power (log)
Energy
Frequency (Hz)
Copy 63 features to clipboard
Autotune parameters
Save parameters
https://studio.edgeimpulse.com/studio/750061/impulse/1/dsp/spectral-analysis/2#

Raw features
`-2.4000, 9.5200, 9.9780, -1.7580, 11.4300, 9.29…`

Label
terrestrial

Parameters

Filter

Scale axes	1
Input decimation ratio	1
Type	none

Analysis

Type	FFT
FFT length	32
Take log of spectrum?	✓
Overlap FFT frames?	✓
Improve low frequency resolution?	

Processed features
`1.3886, -0.4357, -0.6397, 1.7239, 1.2183, 1.2957, 1.2177, 0.5197, 0…`

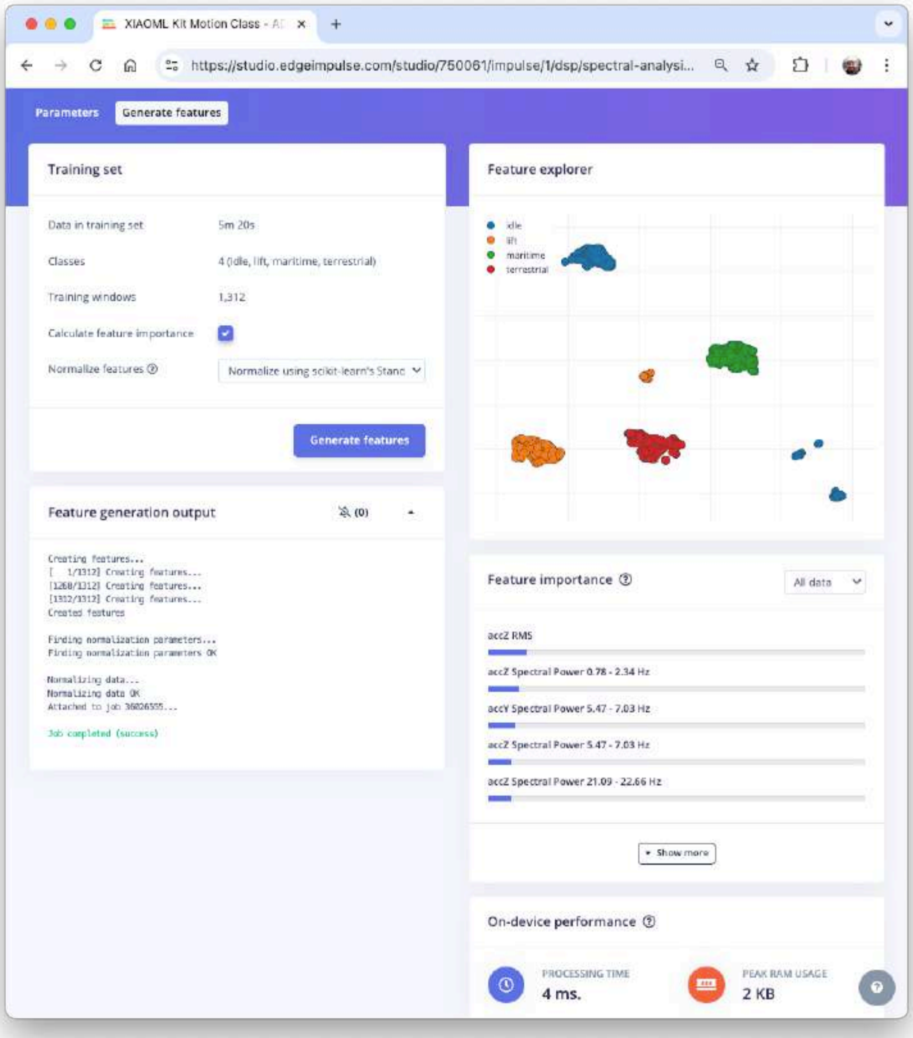
On-device performance

PROCESSING TIME	4 ms.
PEAK RAM USAGE	2 KB

At the top menu, select the `Generate features` tab, and there, select the options, `Calculate feature importance`, `Normalize features,` and press the `[Generate features]` button. Each 2-second window of data (300 data points) will be converted into a single tabular data point with 63 features.

> The Feature Explorer will display this data in 2D using UMAP. Uniform Manifold Approximation and Projection (UMAP) is a dimensionality-reduction technique used for visualization, similar to t-SNE, but also for general nonlinear dimensionality reduction.

The visualization shows that the classes are well separated, indicating that the classifier should perform well.


XIAOML Kit Motion Class - AI
https://studio.edgeimpulse.com/studio/750061/impulse/1/dsp/spectral-analysi...
Parameters
Generate features
Feature explorer
idle
lift
maritime
terrestrial
Generate features
Show more

Training set

Data in training set	5m 20s
Classes	4 (idle, lift, maritime, terrestrial)
Training windows	1,312
Calculate feature importance	✓
Normalize features	Normalize using scikit-learn's Stanc

Feature generation output

```
Creating features...
[   1/1312] Creating features...
[1268/1312] Creating features...
[1312/1312] Creating features...
Created features

Finding normalization parameters...
Finding normalization paramters OK

Normalizing data...
Normalizing data OK
Attached to job 36026555...

Job completed (success)
```

Feature importance — All data

- accZ RMS
- accZ Spectral Power 0.78 - 2.34 Hz
- accY Spectral Power 5.47 - 7.03 Hz
- accZ Spectral Power 5.47 - 7.03 Hz
- accZ Spectral Power 21.09 - 22.66 Hz

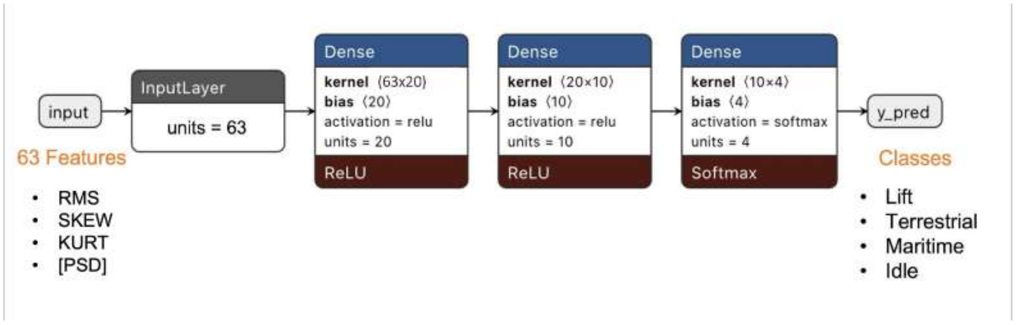
On-device performance

PROCESSING TIME	PEAK RAM USAGE
4 ms.	2 KB

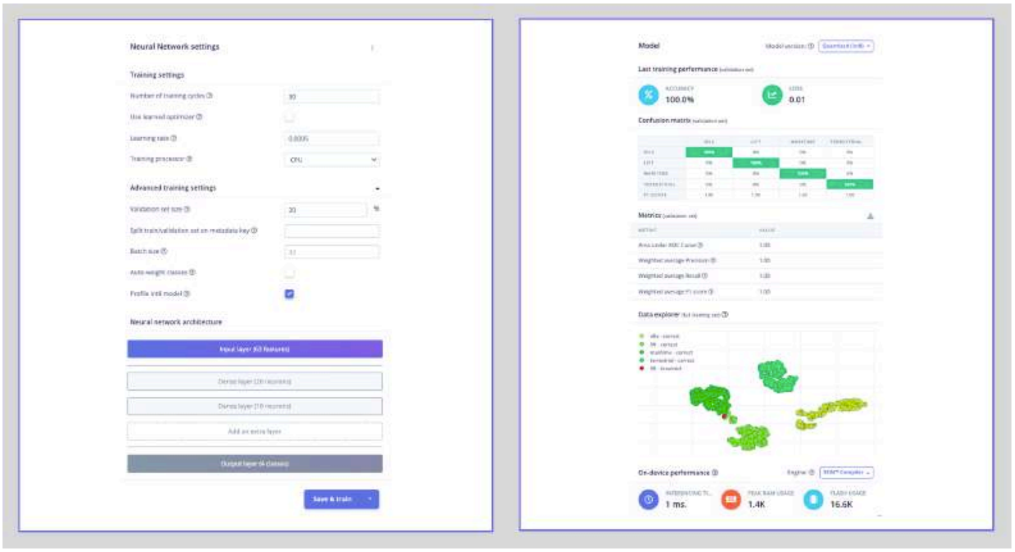
Optionally, you can analyze the relative importance of each feature for one class compared with other classes.

Training

Our classifier will be a Dense Neural Network (DNN) that will have 63 neurons on its input layer, two hidden layers with 20 and 10 neurons, and an output layer with four neurons (one per each class), as shown here:



As hyperparameters, we will use a Learning Rate of 0.005 and 20% of the data for validation for 30 epochs. After training, we can see that the accuracy is 100%.



Neural Network settings

Training settings

Number of training cycles	30
Use learned optimizer	
Learning rate	0.0005
Training processor	CPU

Advanced training settings

Validation set size	20 %
Split train/validation set on metadata key	
Batch size	32
Auto-weight classes	
Profile int8 model	✓

Neural network architecture

Input layer (63 features)

Dense layer (20 neurons)

Dense layer (10 neurons)

Add an extra layer

Output layer (4 classes)

Save & train

Model

Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY 100.0%

LOSS 0.01

Confusion matrix (validation set)

	IDLE	LIFT	MARITIME	TERRESTRIAL
IDLE	100%	0%	0%	0%
LIFT	0%	100%	0%	0%
MARITIME	0%	0%	100%	0%
TERRESTRIAL	0%	0%	0%	100%
F1 SCORE	1.00	1.00	1.00	1.00

Metrics (validation set)

METRIC	VALUE
Area under ROC Curve	1.00
Weighted average Precision	1.00
Weighted average Recall	1.00
Weighted average F1 score	1.00

Data explorer (full training set)

On-device performance

Engine: EON™ Compiler

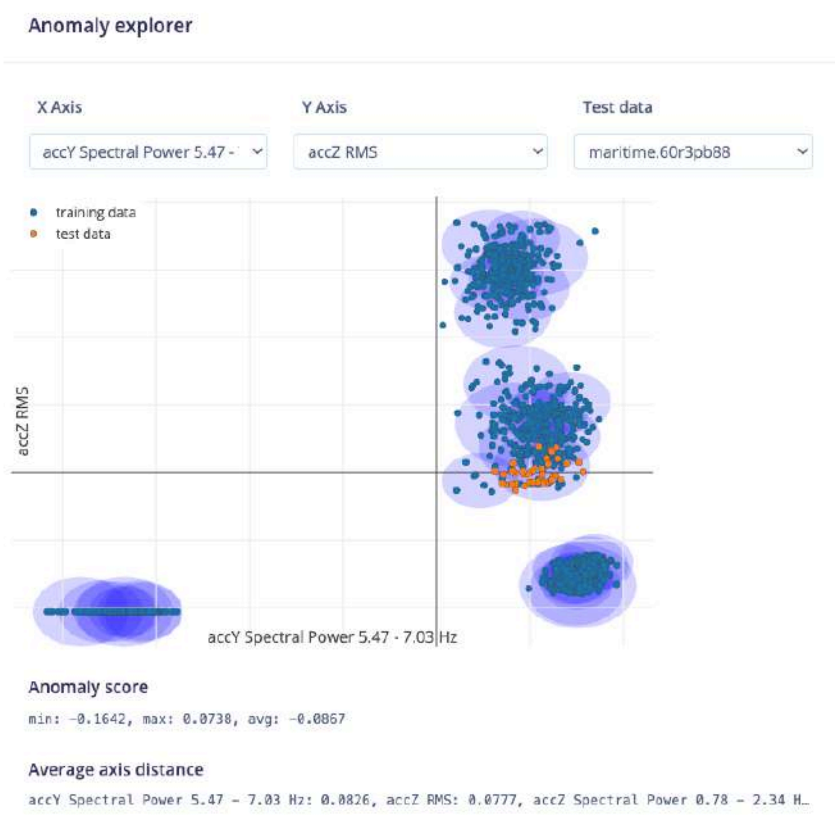
INFERENCING TIME 1 ms.

PEAK RAM USAGE 1.4K

FLASH USAGE 16.6K

For anomaly detection, we should select the suggested features that are most important for feature extraction. The number of clusters will be 32, as suggested by the Studio. After training, we can select some data for testing, such as maritime data. The resulting Anomaly score was `min: -0.1642, max: 0.0738, avg: -0.0867`.

When changing the data, it is possible to realize that small or negative Anomaly Scores indicate that the data are normal.


Anomaly explorer

X Axis: accY Spectral Power 5.47 - 7.03 Hz

Y Axis: accZ RMS

Test data: maritime.60r3pb88

Anomaly score

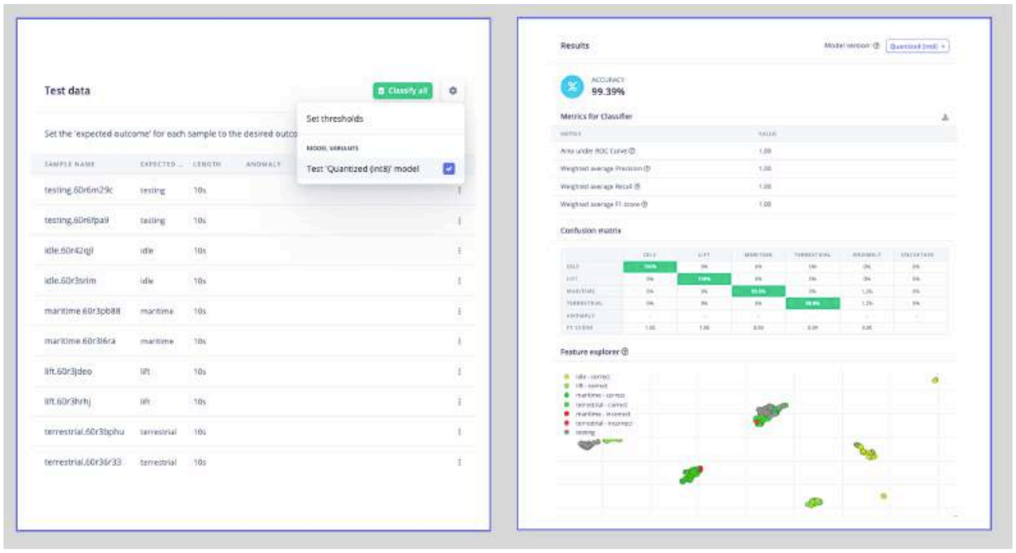
```
min: -0.1642, max: 0.0738, avg: -0.0867
```

Average axis distance

```
accY Spectral Power 5.47 - 7.03 Hz: 0.0826, accZ RMS: 0.0777, accZ Spectral Power 0.78 - 2.34 H…
```

Testing

Using 20% of the data left behind during the data capture phase, we can verify how our model will behave with unknown data; if not 100% (what is expected), the result was very good (8%).


Test data

Set the 'expected outcome' for each sample to the desired outco

SAMPLE NAME	EXPECTED ...	LENGTH	ANOMALY
testing.60r6m29c	testing	10s	
testing.60r6fpa9	testing	10s	
idle.60r42qjl	idle	10s	
idle.60r3srim	idle	10s	
maritime.60r3pb88	maritime	10s	
maritime.60r3l6ra	maritime	10s	
lift.60r3jdeo	lift	10s	
lift.60r3hrhj	lift	10s	
terrestrial.60r3bphu	terrestrial	10s	
terrestrial.60r36r33	terrestrial	10s	

Results

Model version: Quantized (int8)

ACCURACY
99.39%

Metrics for Classifier

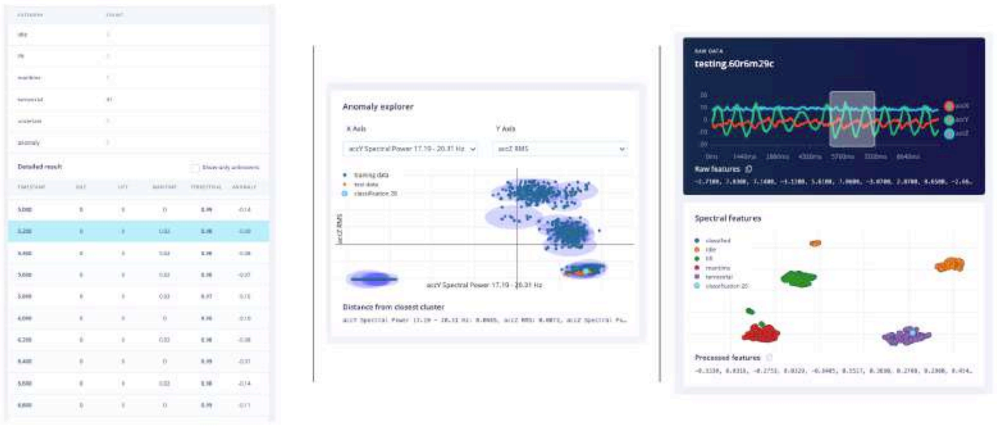
METRIC	VALUE
Area under ROC Curve	1.00
Weighted average Precision	1.00
Weighted average Recall	1.00
Weighted average F1 score	1.00

Confusion matrix

	IDLE	LIFT	MARITIME	TERRESTRIAL	ANOMALY	UNCERTAIN
IDLE	100%	0%	0%	0%	0%	0%
LIFT	0%	100%	0%	0%	0%	0%
MARITIME	0%	0%	[illegible]	0%	1.2%	0%
TERRESTRIAL	0%	0%	0%	[illegible]	1.2%	0%
ANOMALY	-	-	-	-	-	-
F1 SCORE	1.00	1.00	[illegible]	[illegible]	[illegible]	

Feature explorer

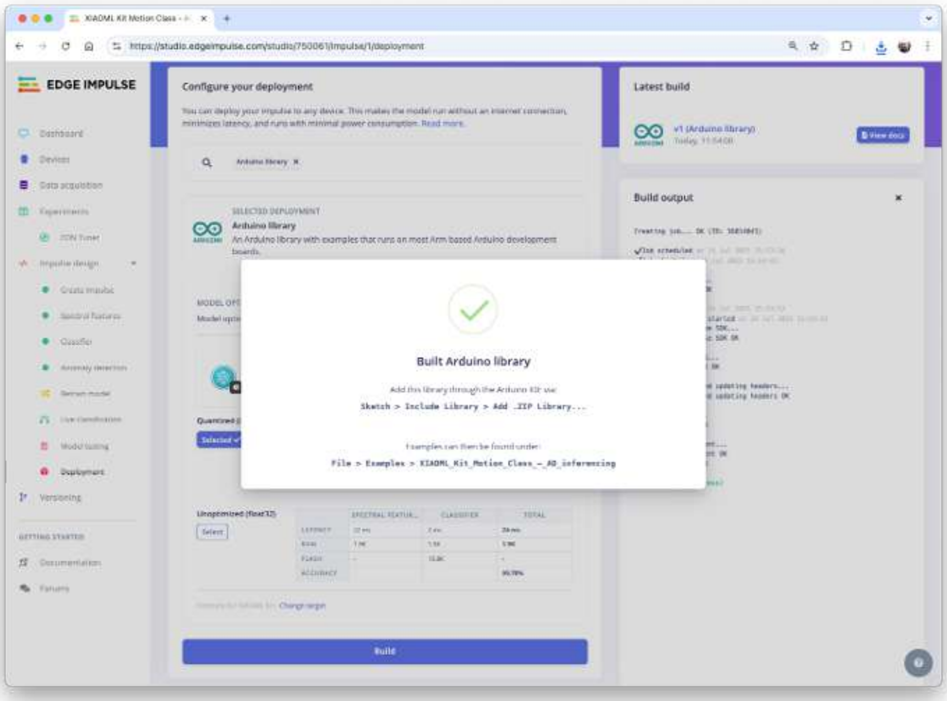
You should also use your kit (which is still connected to the Studio) and perform some Live Classification. For example, let's test some "terrestrial" movement:



Be aware that here, you will capture real data with your device and upload it to the Studio, where an inference will be made using the trained model (note that the model is not on your device).

Deploy

Now it is time for magic! The Studio will package all the needed libraries, preprocessing functions, and trained models, downloading them to your computer. You should select the Arduino Library option, and then, at the bottom, choose Quantized (Int8) and click `[Build]`. A ZIP file will be created and downloaded to your computer.



Built Arduino library

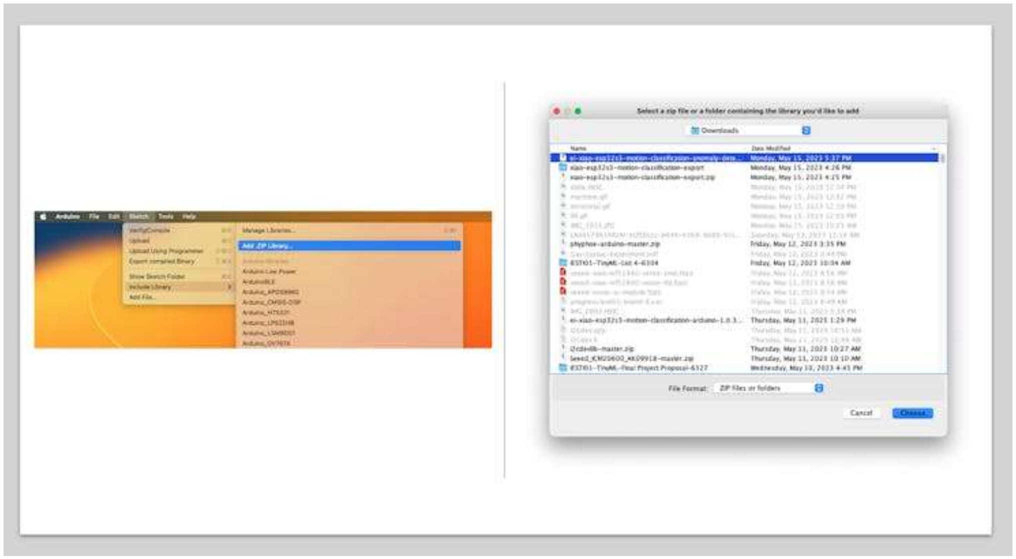
Add this library through the Arduino IDE via:

```
Sketch > Include Library > Add .ZIP Library...
```

Examples can then be found under:

```
File > Examples > XIAOML_Kit_Motion_Class_-_AD_inferencing
```

On your Arduino IDE, go to the Sketch tab, select the option Add.ZIP Library, and Choose the.zip file downloaded by the Studio:


Sketch
Include Library
Manage Libraries...
Add .ZIP Library...
Select a zip file or a folder containing the library you'd like to add
Downloads
File Format: ZIP Files or folders
Cancel
Choose

Inference

Now, it is time for a real test. We will make inferences that are wholly disconnected from the Studio. Let's change one of the code examples created when you deploy the Arduino Library.

In your Arduino IDE, go to the `File/Examples` tab and look for your project, and in examples, select `nano_ble_sense_accelerometer`:

Of course, this is not your board, but we can have the code working with only a few changes.

For example, at the beginning of the code, you have the library related to Arduino Sense IMU:

```
/* Includes ---------------------------------------------- */
#include <XIAOML_Kit_Motion_Class_-_AD_inferencing.h>
#include <Arduino_LSM9DS1.h>
```

Change the "includes" portion with the code related to the IMU:

```
#include <XIAOML_Kit_Motion_Class_-_AD_inferencing.h>
#include <LSM6DS3.h>
#include <Wire.h>
```

Change the Constant Defines

```
// IMU setup
LSM6DS3 myIMU(I2C_MODE, 0x6A);

// Inference settings
#define CONVERT_G_TO_MS2    9.81f
#define MAX_ACCEPTED_RANGE  2.0f * CONVERT_G_TO_MS2
```

On the setup function, initiate the IMU:

```
   // Initialize IMU
    if (myIMU.begin() != 0) {
        Serial.println("ERROR: IMU initialization failed!");
        return;
    }
```

At the loop function, the buffers buffer[ix], buffer[ix + 1], and buffer[ix + 2] will receive the 3-axis data captured by the accelerometer. In the original code, you have the line:

```
IMU.readAcceleration(buffer[ix], buffer[ix + 1], buffer[ix + 2]);
```

Change it with this block of code:

```
// Read IMU data
float x = myIMU.readFloatAccelX();
float y = myIMU.readFloatAccelY();
float z = myIMU.readFloatAccelZ();
```

You should reorder the following two blocks of code. First, you make the conversion to raw data to "Meters per squared second (m/s^2)", followed by the test regarding the maximum acceptance range (that here is in m/s^2, but on Arduino, was in Gs):

```
  // Convert to m/s²
  buffer[i + 0] = x * CONVERT_G_TO_MS2;
  buffer[i + 1] = y * CONVERT_G_TO_MS2;
  buffer[i + 2] = z * CONVERT_G_TO_MS2;

  // Apply range limiting
  for (int j = 0; j < 3; j++) {
      if (fabs(buffer[i + j]) > MAX_ACCEPTED_RANGE) {
          buffer[i + j] = copysign(MAX_ACCEPTED_RANGE, buffer[i + j]);
      }
  }
```

And this is enough. We can also adjust how the inference is displayed in the Serial Monitor. You can now upload the complete code below to your device and proceed with the inferences.

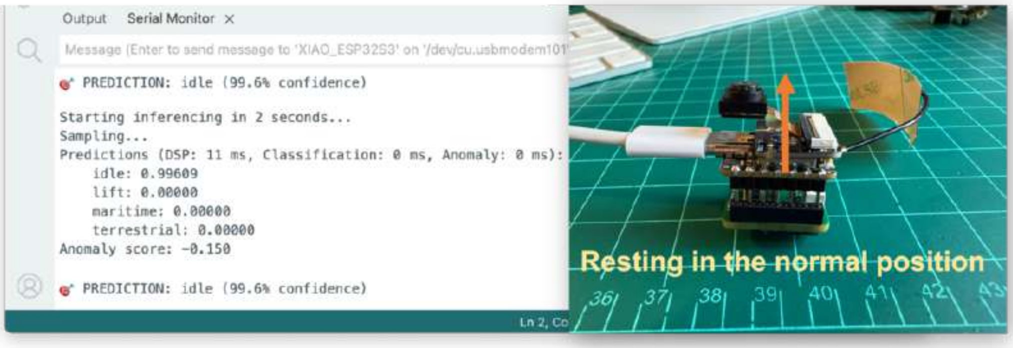
```
// Motion Classification with LSM6DS3TR-C IMU
#include <XIAOML_Kit_Motion_Class_-_AD_inferencing.h>
#include <LSM6DS3.h>
#include <Wire.h>

// IMU setup
LSM6DS3 myIMU(I2C_MODE, 0x6A);

// Inference settings
#define CONVERT_G_TO_MS2    9.81f
#define MAX_ACCEPTED_RANGE  2.0f * CONVERT_G_TO_MS2

static bool debug_nn = false;
static float buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE] = { 0 };
static float inference_buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE];

void setup() {
    Serial.begin(115200);
    while (!Serial) delay(10);

    Serial.println("XIAOML Kit - Motion Classification");
    Serial.println("LSM6DS3TR-C IMU Inference");

    // Initialize IMU
    if (myIMU.begin() != 0) {
        Serial.println("ERROR: IMU initialization failed!");
        return;
    }

    Serial.println("  IMU initialized");

    if (EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME != 3) {
        Serial.println("ERROR: EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME"
                       "should be 3");
        return;
    }

    Serial.println("  Model loaded");
    Serial.println("Starting motion classification...");
}

void loop() {
    ei_printf("\nStarting inferencing in 2 seconds...\n");
    delay(2000);

    ei_printf("Sampling...\n");

    // Clear buffer
    for (size_t i = 0; i < EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE; i++) {
        buffer[i] = 0.0f;
    }

    // Collect accelerometer data
    for (int i = 0; i < EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE; i += 3) {
        uint64_t next_tick = micros() +
          (EI_CLASSIFIER_INTERVAL_MS * 1000);

        // Read IMU data
        float x = myIMU.readFloatAccelX();
        float y = myIMU.readFloatAccelY();
        float z = myIMU.readFloatAccelZ();

        // Convert to m/s²
        buffer[i + 0] = x * CONVERT_G_TO_MS2;
        buffer[i + 1] = y * CONVERT_G_TO_MS2;
        buffer[i + 2] = z * CONVERT_G_TO_MS2;

        // Apply range limiting
        for (int j = 0; j < 3; j++) {
            if (fabs(buffer[i + j]) > MAX_ACCEPTED_RANGE) {
                buffer[i + j] = copysign(MAX_ACCEPTED_RANGE,
                                         buffer[i + j]);
            }
        }

        delayMicroseconds(next_tick - micros());
    }

    // Copy to inference buffer
    for (int i = 0; i < EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE; i++) {
        inference_buffer[i] = buffer[i];
    }

    // Create signal from buffer
    signal_t signal;
    int err = numpy::signal_from_buffer(inference_buffer,
              EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, &signal);
    if (err != 0) {
        ei_printf("ERROR: Failed to create signal from buffer (%d)\n",
                  err);
        return;
    }

    // Run the classifier
    ei_impulse_result_t result = { 0 };
    err = run_classifier(&signal, &result, debug_nn);
```

```
    if (err != EI_IMPULSE_OK) {
        ei_printf("ERROR: Failed to run classifier (%d)\n", err);
        return;
    }

    // Print predictions
    ei_printf("Predictions (DSP: %d ms, Classification: %d ms, "
              "Anomaly: %d ms):\n",
        result.timing.dsp, result.timing.classification, result.timing.anomaly);

    for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
        ei_printf("    %s: %.5f\n", result.classification[ix].label,
                  result.classification[ix].value);
    }

    // Print anomaly score
#if EI_CLASSIFIER_HAS_ANOMALY == 1
    ei_printf("Anomaly score: %.3f\n", result.anomaly);
#endif

    // Determine prediction
    float max_confidence = 0.0;
    String predicted_class = "unknown";

    for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
        if (result.classification[ix].value > max_confidence) {
            max_confidence = result.classification[ix].value;
            predicted_class = String(result.classification[ix].label);
        }
    }

    // Display result with confidence threshold
    if (max_confidence > 0.6) {
        ei_printf("\n  PREDICTION: %s (%.1f%% confidence)\n",
                 predicted_class.c_str(), max_confidence * 100);
    } else {
        ei_printf("\n  UNCERTAIN: Highest confidence is %s (%.1f%%)\n",
                 predicted_class.c_str(), max_confidence * 100);
    }

    // Check for anomaly
```

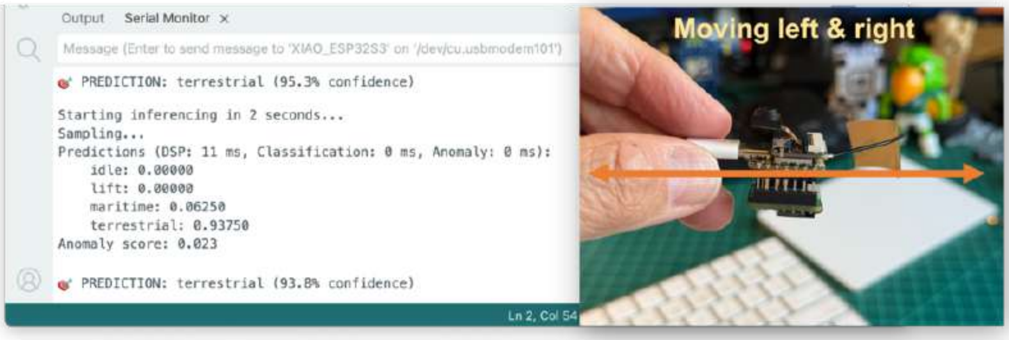
```
#if EI_CLASSIFIER_HAS_ANOMALY == 1
    if (result.anomaly > 0.5) {
        ei_printf("  ANOMALY DETECTED! Score: %.3f\n", result.anomaly);
    }
#endif

    delay(1000);
}

void ei_printf(const char *format, ...) {
    static char print_buf[1024] = { 0 };
    va_list args;
    va_start(args, format);
    int r = vsnprintf(print_buf, sizeof(print_buf), format, args);
    va_end(args);
    if (r > 0) {
        Serial.write(print_buf);
    }
}
```

The complete code is available on the Lab's GitHub.

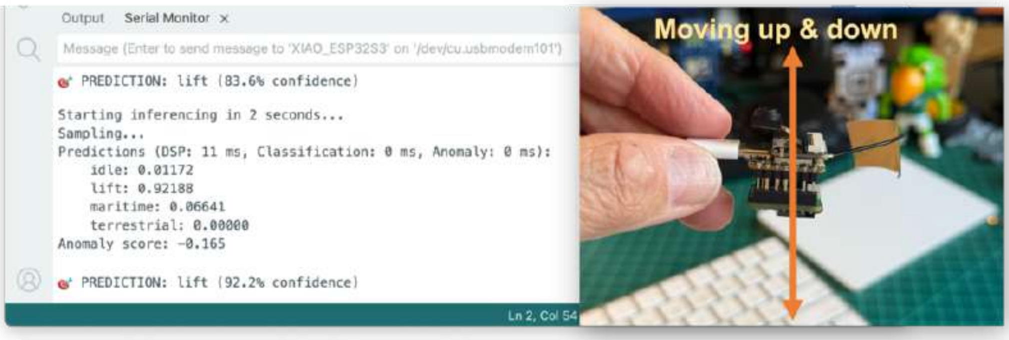
Now you should try your movements, seeing the result of the inference of each class on the images:


Output
Serial Monitor
Message (Enter to send message to 'XIAO_ESP32S3' on '/dev/cu.usbmodem101'
Ln 2, Co
Resting in the normal position

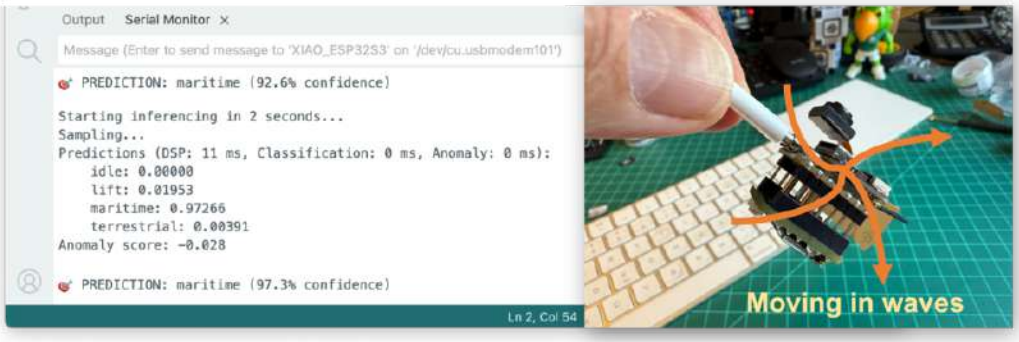
```
PREDICTION: idle (99.6% confidence)

Starting inferencing in 2 seconds...
Sampling...
Predictions (DSP: 11 ms, Classification: 0 ms, Anomaly: 0 ms):
    idle: 0.99609
    lift: 0.00000
    maritime: 0.00000
    terrestrial: 0.00000
Anomaly score: -0.150

PREDICTION: idle (99.6% confidence)
```



```
Output   Serial Monitor ×
Message (Enter to send message to 'XIAO_ESP32S3' on '/dev/cu.usbmodem101')
PREDICTION: terrestrial (95.3% confidence)

Starting inferencing in 2 seconds...
Sampling...
Predictions (DSP: 11 ms, Classification: 0 ms, Anomaly: 0 ms):
    idle: 0.00000
    lift: 0.00000
    maritime: 0.06250
    terrestrial: 0.93750
Anomaly score: 0.023

PREDICTION: terrestrial (93.8% confidence)
Ln 2, Col 54
```

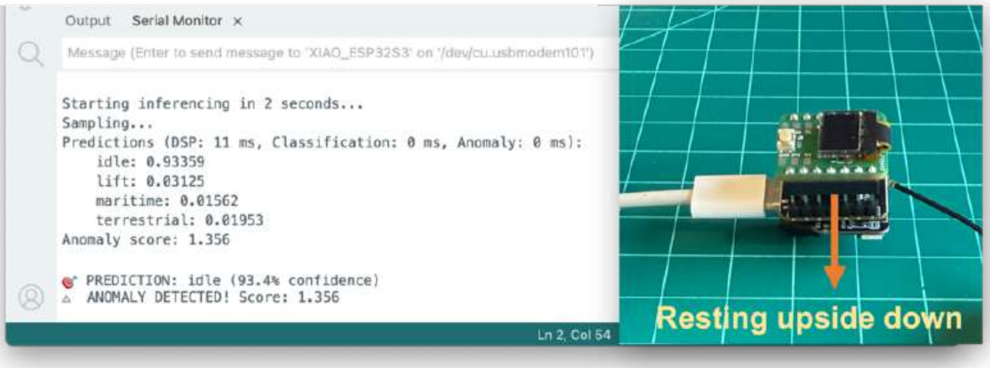


```
Output   Serial Monitor ×
Message (Enter to send message to 'XIAO_ESP32S3' on '/dev/cu.usbmodem101')
PREDICTION: lift (83.6% confidence)

Starting inferencing in 2 seconds...
Sampling...
Predictions (DSP: 11 ms, Classification: 0 ms, Anomaly: 0 ms):
    idle: 0.01172
    lift: 0.92188
    maritime: 0.06641
    terrestrial: 0.00000
Anomaly score: -0.165

PREDICTION: lift (92.2% confidence)
Ln 2, Col 54
```



```
Output   Serial Monitor ×
Message (Enter to send message to 'XIAO_ESP32S3' on '/dev/cu.usbmodem101')
PREDICTION: maritime (92.6% confidence)

Starting inferencing in 2 seconds...
Sampling...
Predictions (DSP: 11 ms, Classification: 0 ms, Anomaly: 0 ms):
    idle: 0.00000
    lift: 0.01953
    maritime: 0.97266
    terrestrial: 0.00391
Anomaly score: -0.028

PREDICTION: maritime (97.3% confidence)
Ln 2, Col 54
```

And, of course, some "anomaly", for example, putting the XIAO upside-down. The anomaly score will be over 0.5:

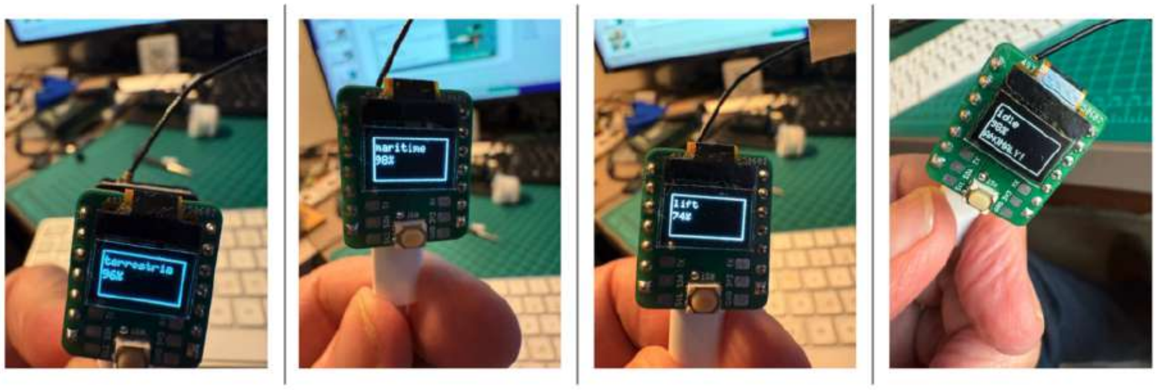

Output
Serial Monitor
Message (Enter to send message to 'XIAO_ESP32S3' on '/dev/cu.usbmodem101')
Ln 2, Col 54
Resting upside down

```
Starting inferencing in 2 seconds...
Sampling...
Predictions (DSP: 11 ms, Classification: 0 ms, Anomaly: 0 ms):
    idle: 0.93359
    lift: 0.03125
    maritime: 0.01562
    terrestrial: 0.01953
Anomaly score: 1.356

PREDICTION: idle (93.4% confidence)
ANOMALY DETECTED! Score: 1.356
```

Post-Prossessing

Now that we know the model is working, we suggest modifying the code to see the result with the Kit completely offline (disconnected from the PC and powered by a battery, a power bank, or an independent 5V power supply).

The idea is that if a specific movement is detected, a corresponding message will appear on the OLED display.



The modified inference code to have the OLED display is available on the Lab's GitHub.

Summary

This lab demonstrated how to build a complete motion classification system using the XIAOML Kit's built-in LSM6DS3TR-C IMU sensor. Key achievements include:

Technical Implementation:

- Utilized the integrated 6-axis IMU for motion sensing
- Collected labeled training data for four transportation scenarios
- Implemented spectral feature extraction for time-series analysis
- Deployed a neural network classifier optimized for microcontroller inference
- Added anomaly detection for identifying unusual movements

Machine Learning Pipeline:

- Data collection directly from embedded sensors
- Feature engineering using frequency domain analysis
- Model training and optimization in Edge Impulse
- Real-time inference on resource-constrained hardware
- Performance monitoring and validation

Practical Applications: The techniques learned apply directly to real-world scenarios, including:

- Asset tracking and logistics monitoring
- Predictive maintenance for machinery
- Human activity recognition
- Vehicle and equipment monitoring
- IoT sensor networks for smart cities

Key Learnings:

- Working with IMU coordinate systems and sensor fusion
- Balancing model accuracy with inference speed on edge devices
- Implementing robust data collection and preprocessing pipelines
- Deploying machine learning models to embedded systems
- Integrating multiple sensors (IMU + display) for complete solutions

The integration of motion classification with the XIAOML Kit demonstrates how modern embedded systems can perform sophisticated AI tasks locally, enabling real-time decision-making without reliance on the cloud. This approach is fundamental to the future of edge AI in industrial IoT, autonomous systems, and smart device applications.

Resources

- XIAOML KIT Code
- DSP Spectral Features
- Edge Impulse Project
- Edge Impulse Spectral Features Block Colab Notebook

- Edge Impulse Documentation
- Edge Impulse Spectral Features
- Seeed Studio LSM6DS3 Library

#

Preprocessing Deepdiving

DSP Spectral Features

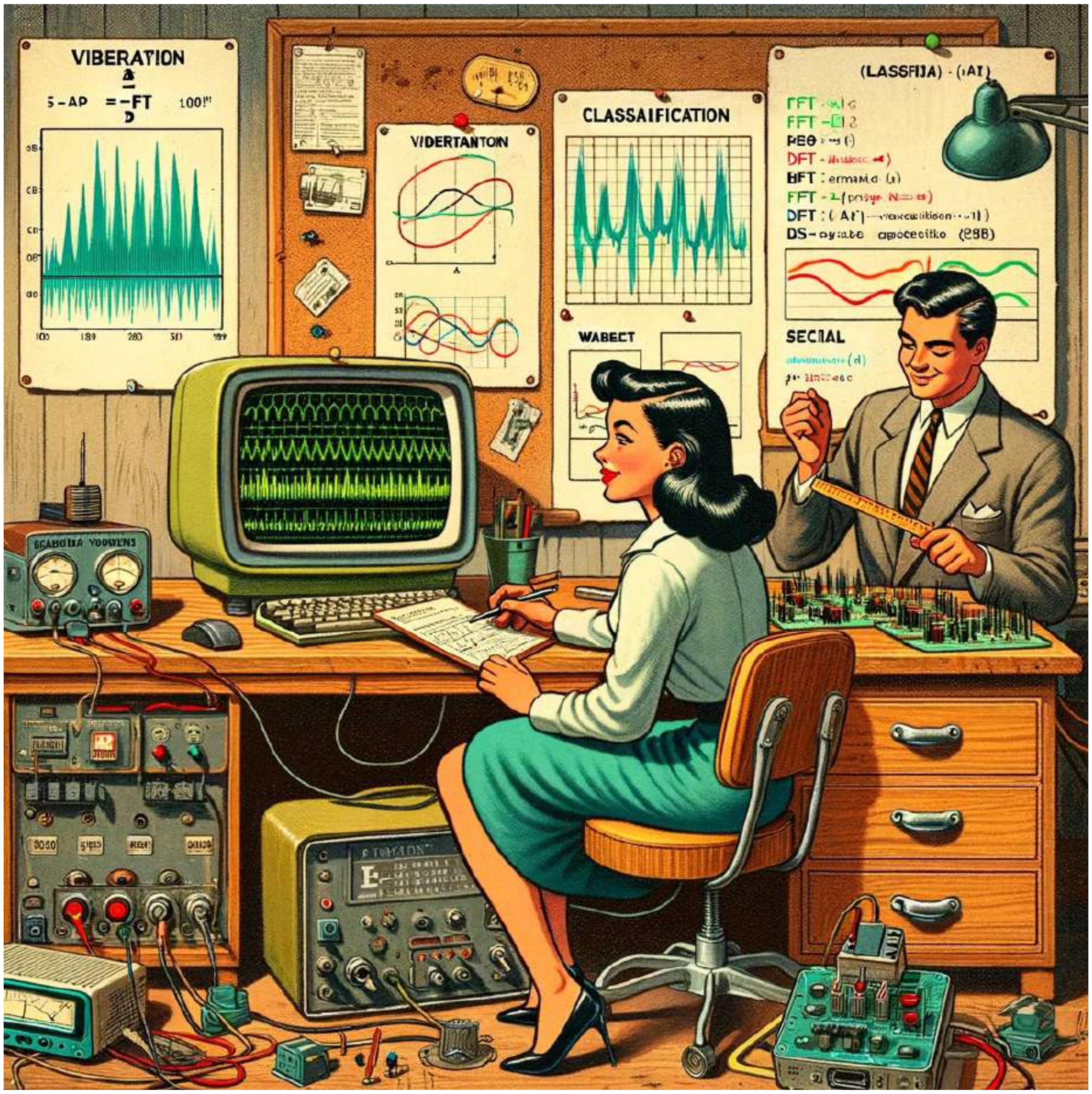


Figure 7: *DALL · E 3 Prompt: 1950s style cartoon illustration of a Latin male and female scientist in a vibration research room. The man is using a calculus ruler to examine ancient circuitry. The woman is at a computer with complex vibration graphs. The wooden table has boards with sensors, prominently an accelerometer. A classic, rounded-back computer shows the Arduino IDE with code for LED pin assignments and machine learning algorithms for movement detection. The Serial Monitor displays FFT, classification, wavelets, and DSPs. Vintage lamps, tools, and charts with FFT and Wavelets graphs complete the scene.*

Overview

TinyML projects related to motion (or vibration) involve data from IMUs (usually **accelerometers** and **Gyroscopes**). These time-series type datasets should be preprocessed before inputting them into a Machine Learning model training, which is a challenging area for embedded machine learning. Still, Edge Impulse helps overcome this complexity with its digital signal processing (DSP) preprocessing step and, more specifically, the Spectral Features Block for Inertial sensors.

But how does it work under the hood? Let's dig into it.

Extracting Features Review

Extracting features from a dataset captured with inertial sensors, such as accelerometers, involves processing and analyzing the raw data. Accelerometers measure the acceleration of an object along one or more axes (typically three, denoted as X, Y, and Z). These measurements can be used to understand various aspects of the object's motion, such as movement patterns and vibrations. Here's a high-level overview of the process:

Data collection: First, we need to gather data from the accelerometers. Depending on the application, data may be collected at different sampling rates. It's essential to ensure that the sampling rate is high enough to capture the relevant dynamics of the studied motion (the sampling rate should be at least double the maximum relevant frequency present in the signal).

Data preprocessing: Raw accelerometer data can be noisy and contain errors or irrelevant information. Preprocessing steps, such as filtering and normalization, can help clean and standardize the data, making it more suitable for feature extraction.

> The Studio does not perform normalization or standardization, so sometimes, when working with Sensor Fusion, it could be necessary to perform this step before uploading data to the Studio. This is particularly crucial in sensor fusion projects, as seen in this tutorial, Sensor Data Fusion with Spresense and CommonSense.

Segmentation: Depending on the nature of the data and the application, dividing the data into smaller segments or **windows** may be necessary. This can help focus on specific events or activities within the dataset, making feature extraction more manageable and meaningful. The **window size** and overlap (**window span**) choice depend on the application and the frequency of the events of interest. As a rule of thumb, we should try to capture a couple of "data cycles."

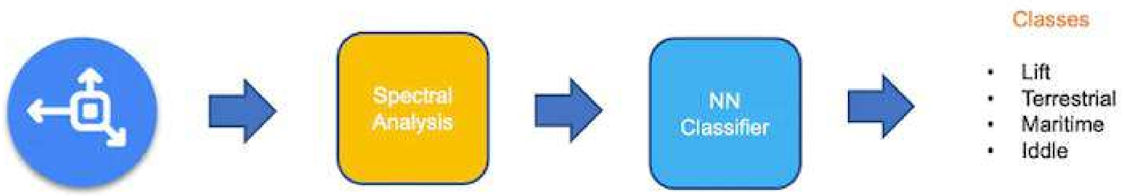
Feature extraction: Once the data is preprocessed and segmented, you can extract features that describe the motion's characteristics. Some typical features extracted from accelerometer data include:

- **Time-domain** features describe the data's statistical properties within each segment, such as mean, median, standard deviation, skewness, kurtosis, and zero-crossing rate.
- **Frequency-domain** features are obtained by transforming the data into the frequency domain using techniques like the Fast Fourier Transform (FFT). Some typical frequency-domain features include the power spectrum, spectral energy, dominant frequencies (amplitude and frequency), and spectral entropy.
- **Time-frequency** domain features combine the time and frequency domain information, such as the Short-Time Fourier Transform (STFT) or the Discrete Wavelet Transform (DWT). They can provide a more detailed understanding of how the signal's frequency content changes over time.

In many cases, the number of extracted features can be large, which may lead to overfitting or increased computational complexity. Feature selection techniques, such as mutual information, correlation-based methods, or principal component analysis (PCA), can help identify the most relevant features for a given application and reduce the dimensionality of the dataset. The Studio can help with such feature-relevant calculations.

Let's explore in more detail a typical TinyML Motion Classification project covered in this series of Hands-Ons.

A TinyML Motion Classification project

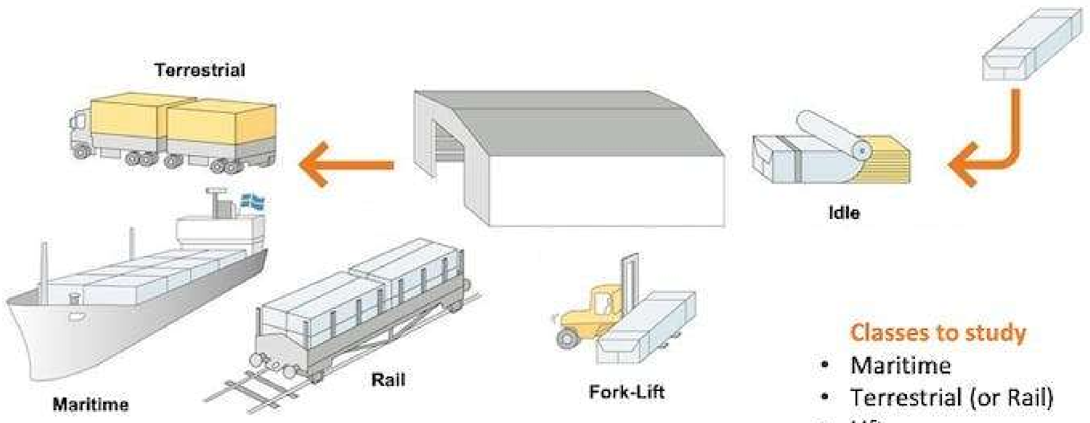


In the hands-on project, *Motion Classification and Anomaly Detection*, we simulated mechanical stresses in transport, where our problem was to classify four classes of movement:

- **Maritime** (pallets in boats)
- **Terrestrial** (pallets in a Truck or Train)
- **Lift** (pallets being handled by Fork-Lift)
- **Idle** (pallets in Storage houses)

The accelerometers provided the data on the pallet (or container).

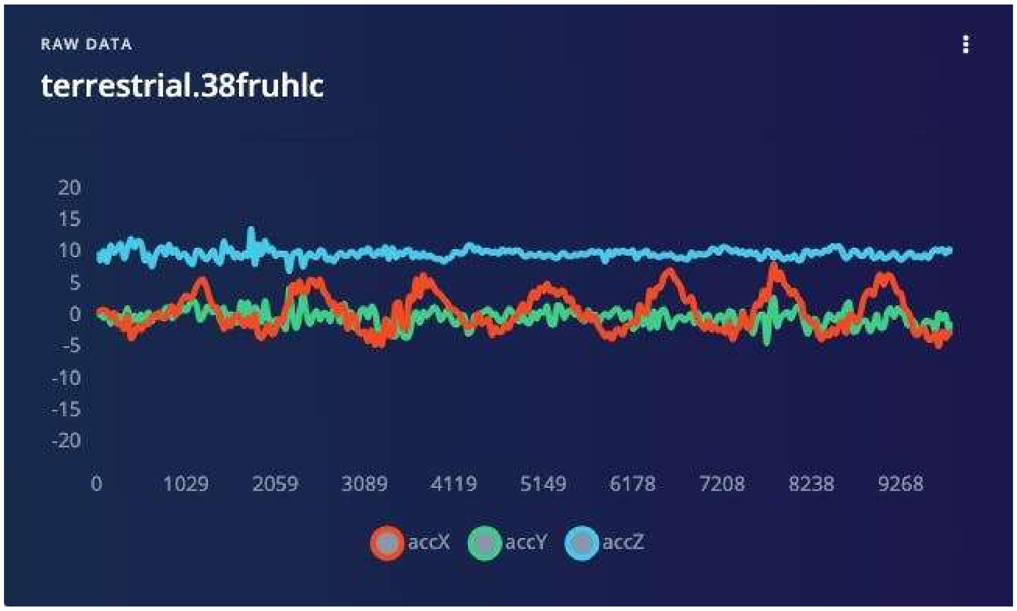
Case Study: Mechanical Stresses in Transport



Classes to study

- Maritime
- Terrestrial (or Rail)
- Lift
- Idle

Below is one sample (raw data) of 10 seconds, captured with a sampling frequency of 50 Hz:

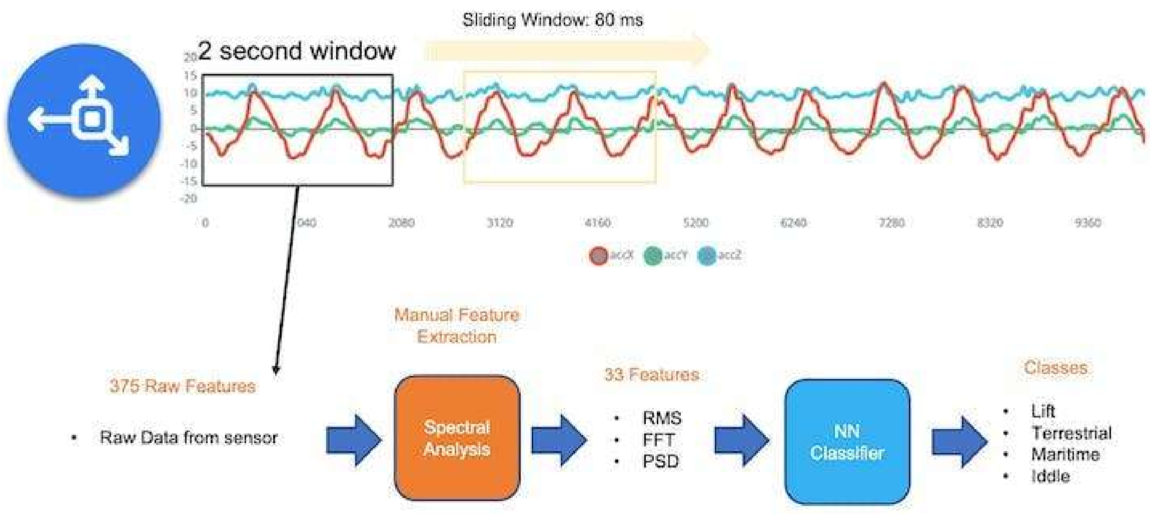


The result is similar when this analysis is done over another dataset with the same principle, using a different sampling frequency, 62.5 Hz instead of 50 Hz.

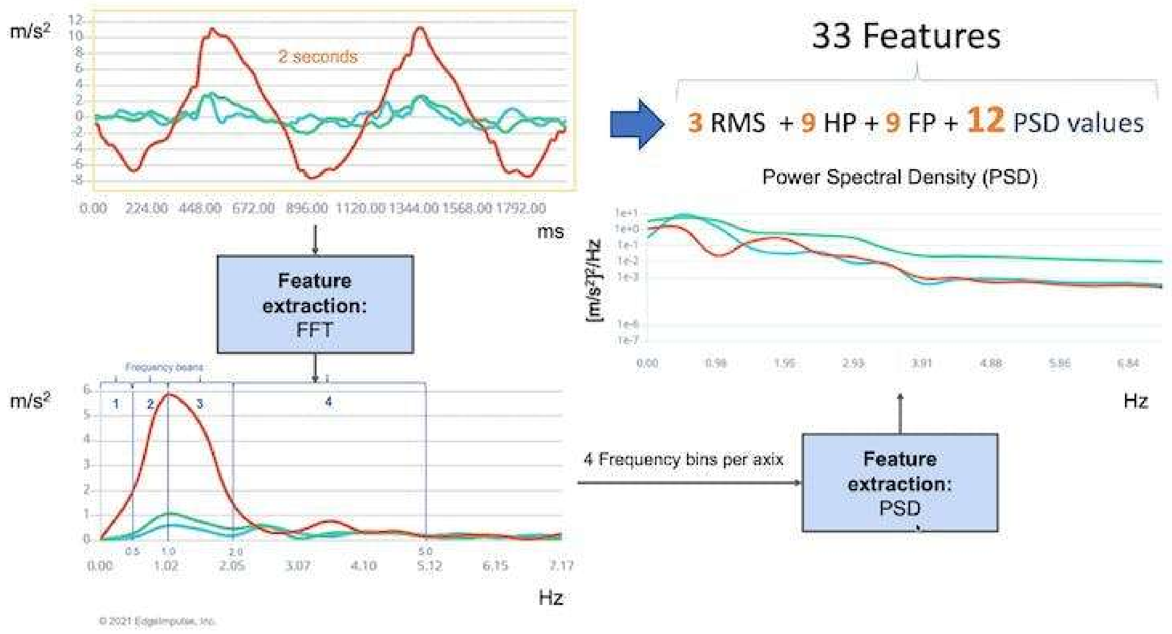
Data Pre-Processing

The raw data captured by the accelerometer (a “time series” data) should be converted to “tabular data” using one of the typical Feature Extraction methods described in the last section.

We should segment the data using a sliding window over the sample data for feature extraction. The project captured accelerometer data every 10 seconds with a sample rate of 62.5 Hz. A 2-second window captures 375 data points (3 axis × 2 seconds × 62.5 samples). The window is slid every 80 ms, creating a larger dataset where each instance has 375 “raw features.”



On the Studio, the previous version (V1) of the **Spectral Analysis Block** extracted as time-domain features only the RMS, and for the frequency-domain, the peaks and frequency (using FFT) and the power characteristics (PSD) of the signal over time resulting in a fixed tabular dataset of 33 features (11 per each axis),



Those 33 features were the Input tensor of a Neural Network Classifier.

In 2022, Edge Impulse released version 2 of the Spectral Analysis block, which we will explore here.

Edge Impulse - Spectral Analysis Block V.2 under the hood

In Version 2, Time Domain Statistical features per axis/channel are:

- RMS
- Skewness
- Kurtosis

And the Frequency Domain Spectral features per axis/channel are:

- Spectral Power
- Skewness (in the next version)
- Kurtosis (in the next version)

In this link, we can have more details about the feature extraction.

> Clone the public project. You can also follow the explanation, playing with the code using my Google CoLab Notebook: Edge Impulse Spectral Analysis Block Notebook.

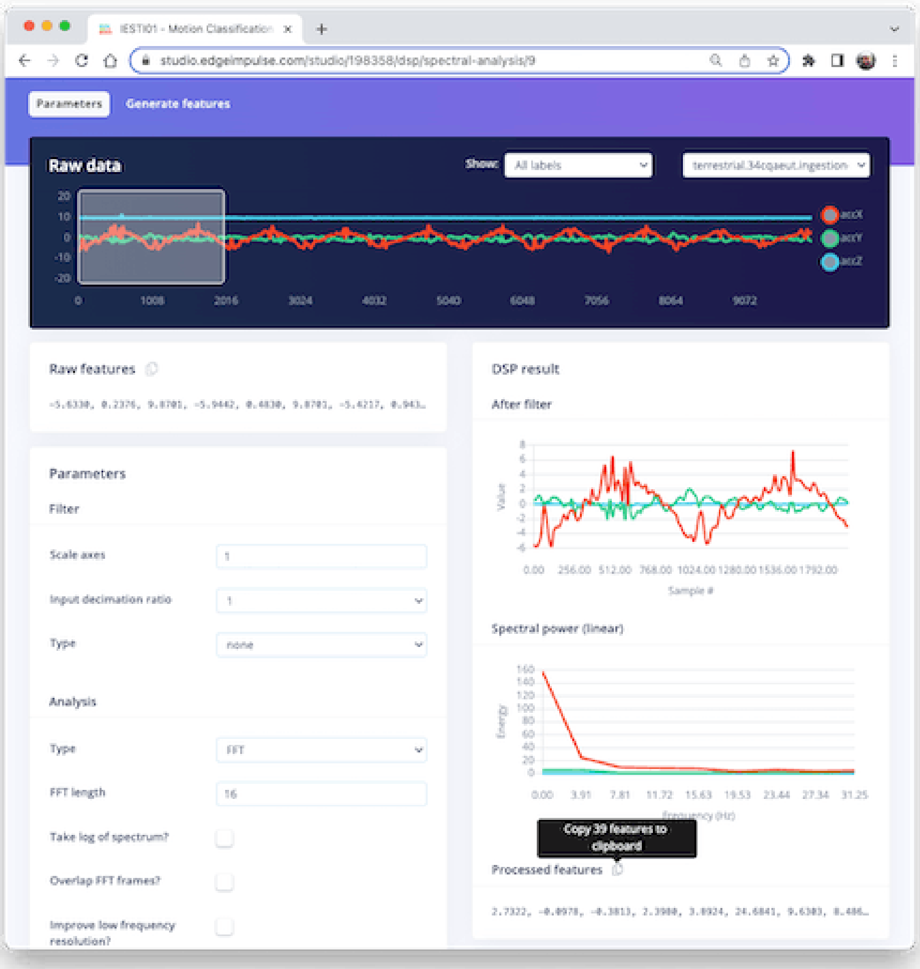
Start importing the libraries:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
from scipy.stats import skew, kurtosis
from scipy import signal
from scipy.signal import welch
from scipy.stats import entropy
from sklearn import preprocessing
import pywt

plt.rcParams['figure.figsize'] = (12, 6)
plt.rcParams['lines.linewidth'] = 3
```

From the studied project, let's choose a data sample from accelerometers as below:

- Window size of 2 seconds: `[2,000]` ms
- Sample frequency: `[62.5]` Hz
- We will choose the `[None]` filter (for simplicity) and a
- FFT length: `[16]`.

```
f =  62.5 # Hertz
wind_sec = 2 # seconds
FFT_Lenght = 16
axis = ['accX', 'accY', 'accZ']
n_sensors = len(axis)
```


IESTI01 - Motion Classification
studio.edgeimpulse.com/studio/198358/dsp/spectral-analysis/9
Parameters
Generate features
Raw data
Show:
All labels
terrestrial.34cqaeut.ingestion
accX
accY
accZ
DSP result
After filter
Value
Sample #
Spectral power (linear)
Energy
Frequency (Hz)
Copy 39 features to clipboard

Raw features

```
-5.6330, 0.2376, 9.8701, -5.9442, 0.4830, 9.8701, -5.4217, 0.943…
```

Parameters

Filter

Scale axes	1
Input decimation ratio	1
Type	none

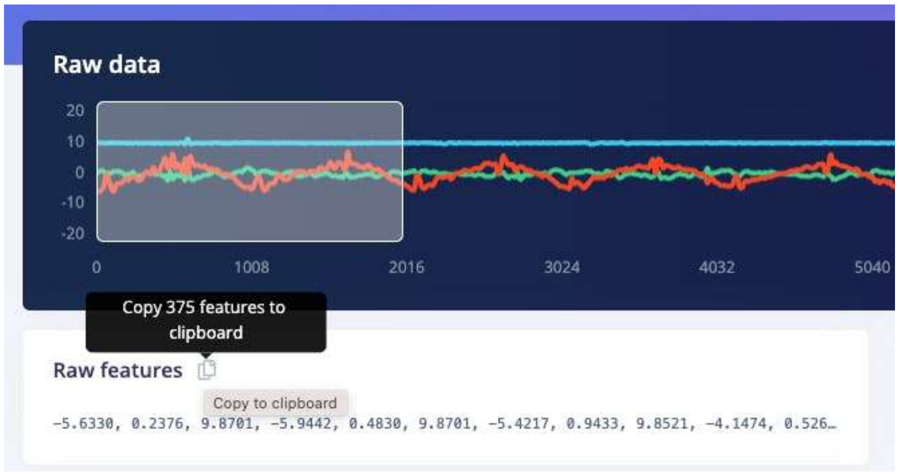
Analysis

Type	FFT
FFT length	16
Take log of spectrum?	☐
Overlap FFT frames?	☐
Improve low frequency resolution?	☐

Processed features

```
2.7322, -0.0978, -0.3813, 2.3988, 3.0924, 24.6841, 9.6383, 0.486…
```

Selecting the *Raw Features* on the Studio Spectral Analysis tab, we can copy all 375 data points of a particular 2-second window to the clipboard.



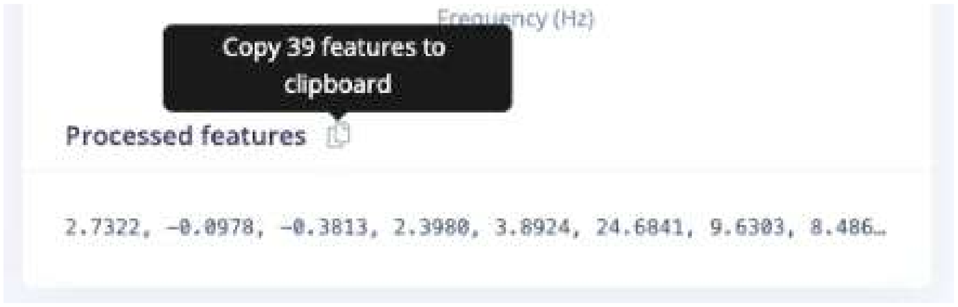
-5.6330, 0.2376, 9.8701, -5.9442, 0.4830, 9.8701, -5.4217, 0.9433, 9.8521, -4.1474, 0.526…

Paste the data points to a new variable *data*:

```
data = [
    -5.6330,  0.2376,  9.8701,
    -5.9442,  0.4830,  9.8701,
    -5.4217, ...
]
No_raw_features = len(data)
N = int(No_raw_features/n_sensors)
```

The total raw features are 375, but we will work with each axis individually, where $N = 125$ (number of samples per axis).

We aim to understand how Edge Impulse gets the processed features.



2.7322, -0.0978, -0.3813, 2.3980, 3.8924, 24.6841, 9.6303, 8.486…

So, you should also past the processed features on a variable (to compare the calculated features in Python with the ones provided by the Studio) :

```
features = [
    2.7322, -0.0978, -0.3813,
    2.3980, 3.8924, 24.6841,
    9.6303, ...
]
N_feat = len(features)
N_feat_axis = int(N_feat/n_sensors)
```

The total number of processed features is 39, which means 13 features/axis.

Looking at those 13 features closely, we will find 3 for the time domain (RMS, Skewness, and Kurtosis):

- `[rms] [skew] [kurtosis]`

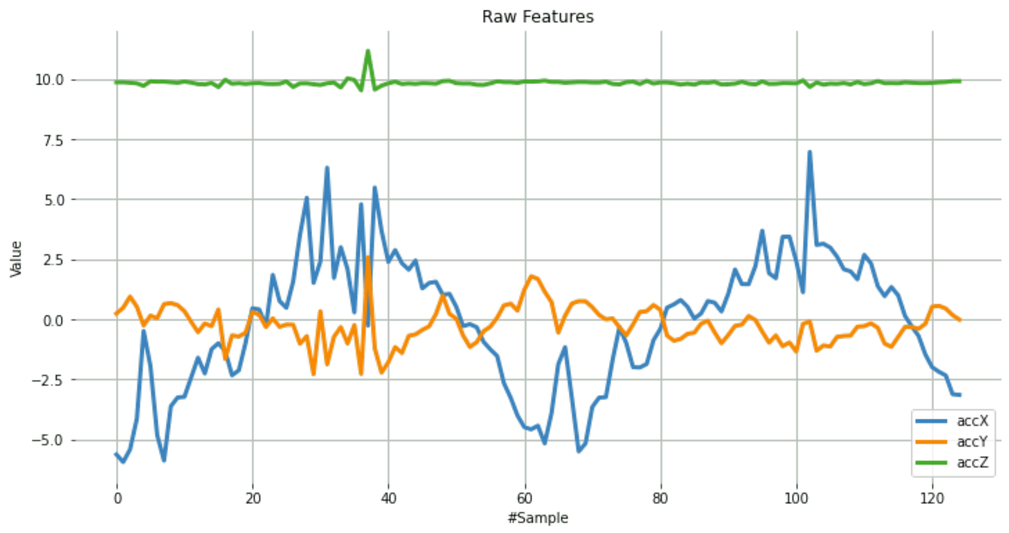
and 10 for the frequency domain (we will return to this later).

- `[spectral skew][spectral kurtosis][Spectral Power 1] ... [Spectral Power 8]`

Splitting raw data per sensor

The data has samples from all axes; let's split and plot them separately:

```
def plot_data(sensors, axis, title):
    [plt.plot(x, label=y) for x,y in zip(sensors, axis)]
    plt.legend(loc='lower right')
    plt.title(title)
    plt.xlabel('#Sample')
    plt.ylabel('Value')
    plt.box(False)
    plt.grid()
    plt.show()

accX = data[0::3]
accY = data[1::3]
accZ = data[2::3]
sensors = [accX, accY, accZ]
plot_data(sensors, axis, 'Raw Features')
```



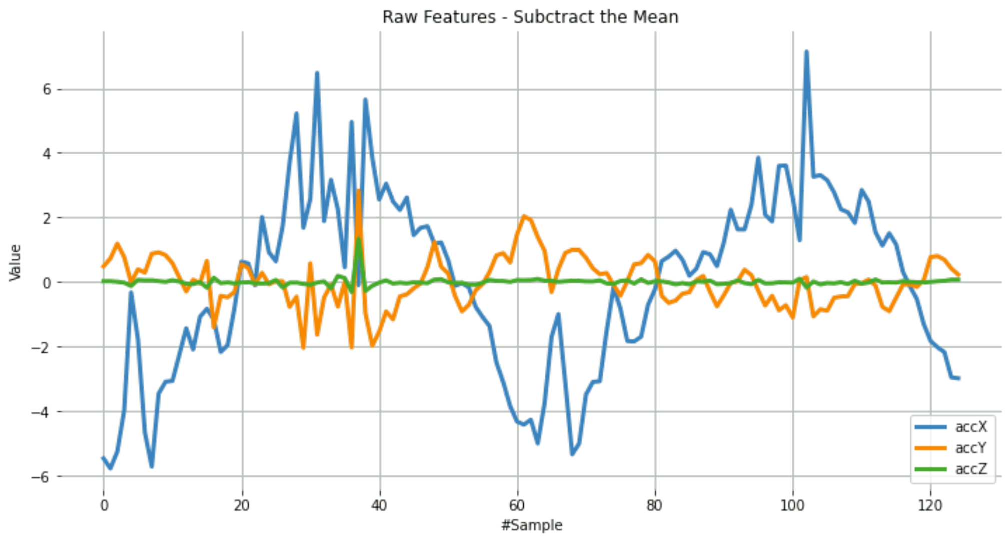
Subtracting the mean

Next, we should subtract the mean from the *data*. Subtracting the mean from a data set is a common data pre-processing step in statistics and machine learning. The purpose of subtracting the mean from the data is to center the data around zero. This is important because it can reveal patterns and relationships that might be hidden if the data is not centered.

Here are some specific reasons why subtracting the mean can be helpful:

- It simplifies analysis: By centering the data, the mean becomes zero, making some calculations simpler and easier to interpret.
- It removes bias: If the data is biased, subtracting the mean can remove it and allow for a more accurate analysis.
- It can reveal patterns: Centering the data can help uncover patterns that might be hidden if the data is not centered. For example, centering the data can help you identify trends over time if you analyze a time series dataset.
- It can improve performance: In some machine learning algorithms, centering the data can improve performance by reducing the influence of outliers and making the data more easily comparable. Overall, subtracting the mean is a simple but powerful technique that can be used to improve the analysis and interpretation of data.

```
dtmean = [
    (sum(x) / len(x))
    for x in sensors
]

[
    print('mean_' + x + ' =', round(y, 4))
    for x, y in zip(axis, dtmean)
][0]

accX = [(x - dtmean[0]) for x in accX]
accY = [(x - dtmean[1]) for x in accY]
accZ = [(x - dtmean[2]) for x in accZ]
sensors = [accX, accY, accZ]

plot_data(sensors, axis, 'Raw Features - Subctract the Mean')
```



Time Domain Statistical features

RMS Calculation

The RMS value of a set of values (or a continuous-time waveform) is the square root of the arithmetic mean of the squares of the values or the square of the function that defines the continuous waveform. In physics, the RMS value of an electrical current is defined as the "value of the direct current that dissipates the same power in a resistor."

In the case of a set of n values $x_1, x_2, \dots, x_n$, the RMS is:

$$x_{\text{RMS}} = \sqrt{\frac{1}{n}\left(x_1^2 + x_2^2 + \cdots + x_n^2\right)}$$

> NOTE that the RMS value is different for the original raw data, and after subtracting the mean

```
# Using numpy and standardized data (subtracting mean)
rms = [np.sqrt(np.mean(np.square(x))) for x in sensors]
```

We can compare the calculated RMS values here with the ones presented by Edge Impulse:

```
[print('rms_'+x+'= ', round(y, 4)) for x,y in zip(axis, rms)][0]
print("\nCompare with Edge Impulse result features")
print(features[0:N_feat:N_feat_axis])
```

```
rms_accX=  2.7322

rms_accY=  0.7833

rms_accZ=  0.1383
```

Compared with Edge Impulse result features:

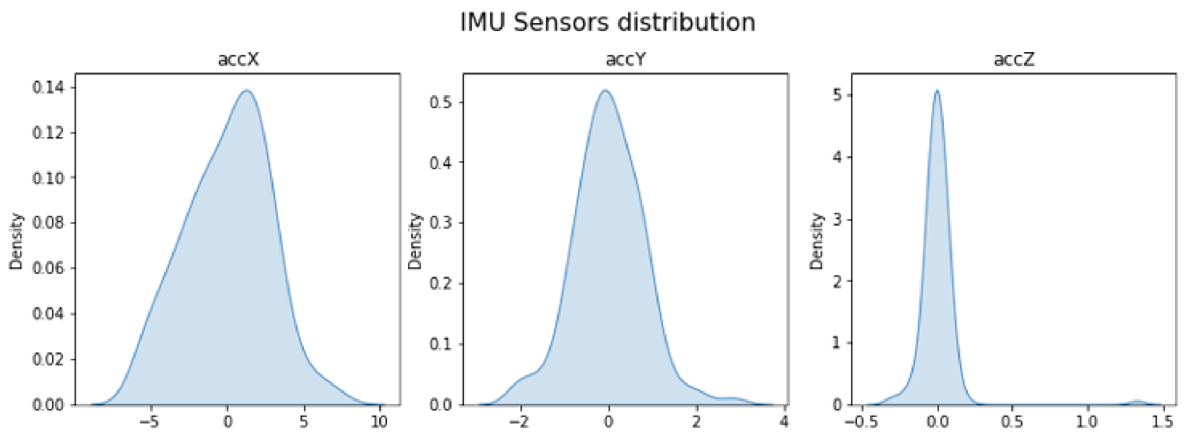
```
[2.7322, 0.7833, 0.1383]
```

Skewness and kurtosis calculation

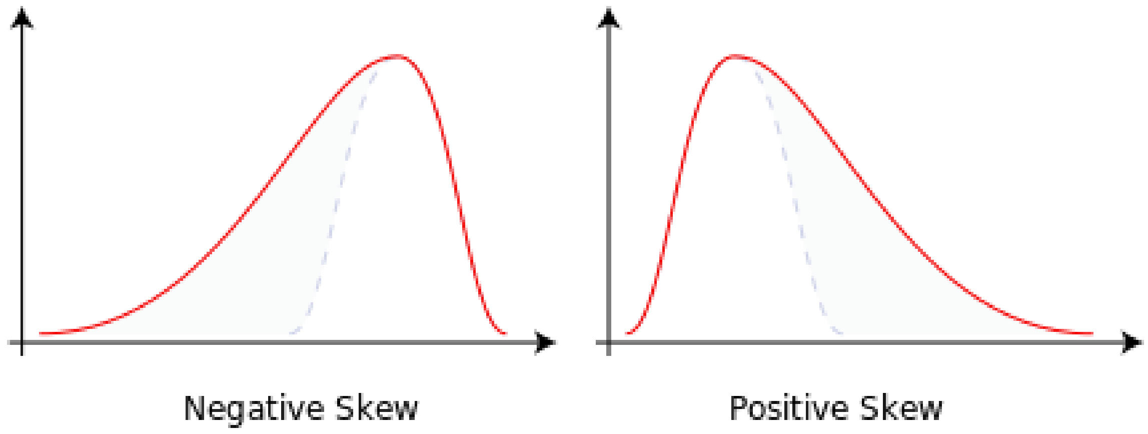
In statistics, skewness and kurtosis are two ways to measure the **shape of a distribution**.

Here, we can see the sensor values distribution:

```
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(13, 4))
sns.kdeplot(accX, fill=True, ax=axes[0])
sns.kdeplot(accY, fill=True, ax=axes[1])
sns.kdeplot(accZ, fill=True, ax=axes[2])
axes[0].set_title('accX')
axes[1].set_title('accY')
axes[2].set_title('accZ')
plt.suptitle('IMU Sensors distribution', fontsize=16, y=1.02)
plt.show()
```



Skewness is a measure of the asymmetry of a distribution. This value can be positive or negative.



- A negative skew indicates that the tail is on the left side of the distribution, which extends towards more negative values.
- A positive skew indicates that the tail is on the right side of the distribution, which extends towards more positive values.
- A zero value indicates no skewness in the distribution at all, meaning the distribution is perfectly symmetrical.

```
skew = [skew(x, bias=False) for x in sensors]
[print('skew_'+x+'= ', round(y, 4))
  for x,y in zip(axis, skew)][0]
print("\nCompare with Edge Impulse result features")
features[1:N_feat:N_feat_axis]
```

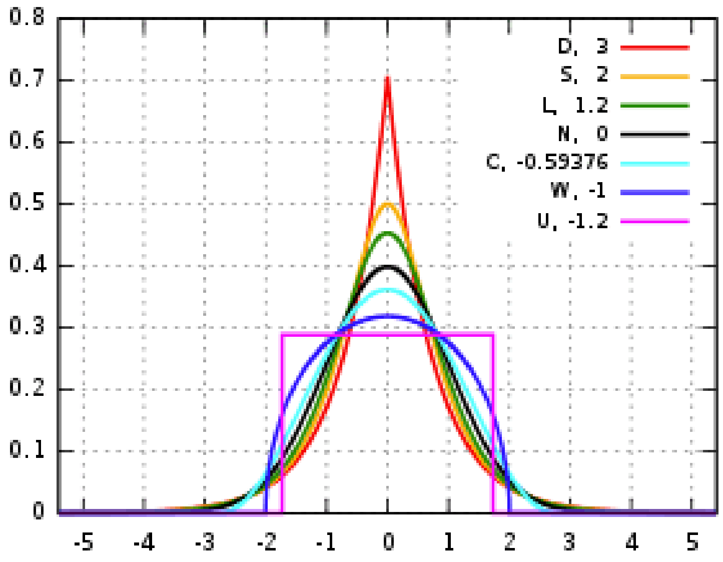
```
skew_accX=  -0.099
```

```
skew_accY=  0.1756

skew_accZ=  6.9463
```

Compared with Edge Impulse result features:

```
[-0.0978, 0.1735, 6.8629]
```

Kurtosis is a measure of whether or not a distribution is heavy-tailed or light-tailed relative to a normal distribution.



- The kurtosis of a normal distribution is zero.
- If a given distribution has a negative kurtosis, it is said to be playkurtic, which means it tends to produce fewer and less extreme outliers than the normal distribution.
- If a given distribution has a positive kurtosis , it is said to be leptokurtic, which means it tends to produce more outliers than the normal distribution.

```
kurt = [kurtosis(x, bias=False) for x in sensors]
[print('kurt_'+x+'= ', round(y, 4))
  for x,y in zip(axis, kurt)][0]
print("\nCompare with Edge Impulse result features")
features[2:N_feat:N_feat_axis]
```

```
kurt_accX=  -0.3475

kurt_accY=  1.2673

kurt_accZ=  68.1123
```

Compared with Edge Impulse result features:

```
[-0.3813, 1.1696, 65.3726]
```

Spectral features

The filtered signal is passed to the Spectral power section, which computes the **FFT** to generate the spectral features.

Since the sampled window is usually larger than the FFT size, the window will be broken into frames (or "sub-windows"), and the FFT is calculated over each frame.

FFT length - The FFT size. This determines the number of FFT bins and the resolution of frequency peaks that can be separated. A low number means more signals will average together in the same FFT bin, but it also reduces the number of features and model size. A high number will separate more signals into separate bins, generating a larger model.

- The total number of Spectral Power features will vary depending on how you set the filter and FFT parameters. With No filtering, the number of features is 1/2 of the FFT Length.

Spectral Power - Welch's method

We should use Welch's method to split the signal on the frequency domain in bins and calculate the power spectrum for each bin. This method divides the signal into overlapping segments, applies a window function to each segment, computes the periodogram of each segment using DFT, and averages them to obtain a smoother estimate of the power spectrum.

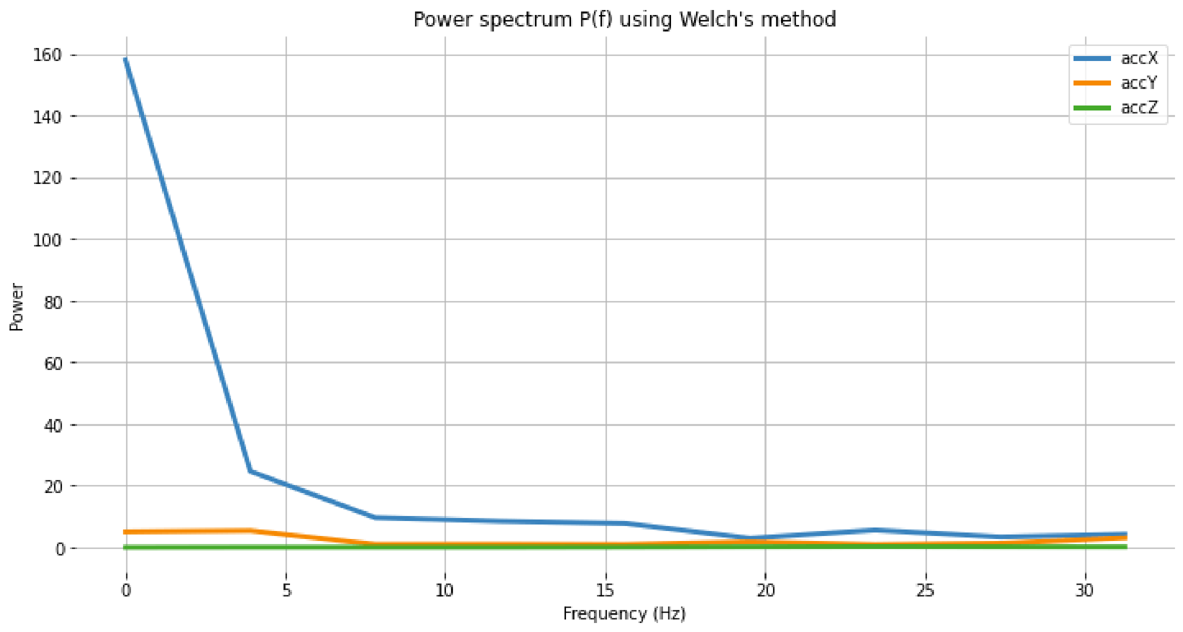
```python
# Function used by Edge Impulse instead of scipy.signal.welch().
def welch_max_hold(fx, sampling_freq, nfft, n_overlap):
    n_overlap = int(n_overlap)
    spec_powers = [0 for _ in range(nfft//2+1)]
    ix = 0
    while ix <= len(fx):
        # Slicing truncates if end_idx > len,
        # and rfft will auto-zero pad
        fft_out = np.abs(np.fft.rfft(fx[ix:ix+nfft], nfft))
        spec_powers = np.maximum(spec_powers, fft_out**2/nfft)
        ix = ix + (nfft-n_overlap)
    return np.fft.rfftfreq(nfft, 1/sampling_freq), spec_powers
```

Applying the above function to 3 signals:

```
fax,Pax = welch_max_hold(accX, fs, FFT_Lenght, 0)
fay,Pay = welch_max_hold(accY, fs, FFT_Lenght, 0)
faz,Paz = welch_max_hold(accZ, fs, FFT_Lenght, 0)
specs = [Pax, Pay, Paz ]
```

We can plot the Power Spectrum P(f):

```
plt.plot(fax,Pax, label='accX')
plt.plot(fay,Pay, label='accY')
plt.plot(faz,Paz, label='accZ')
plt.legend(loc='upper right')
plt.xlabel('Frequency (Hz)')
#plt.ylabel('PSD [V**2/Hz]')
plt.ylabel('Power')
plt.title('Power spectrum P(f) using Welch's method')
plt.grid()
plt.box(False)
plt.show()
```



Besides the Power Spectrum, we can also include the skewness and kurtosis of the features in the frequency domain (should be available on a new version):

```
spec_skew = [skew(x, bias=False) for x in specs]
spec_kurtosis = [kurtosis(x, bias=False) for x in specs]
```

Let's now list all Spectral features per axis and compare them with EI:

```
print("EI Processed Spectral features (accX): ")
print(features[3:N_feat_axis][0:])
print("\nCalculated features:")
print (round(spec_skew[0],4))
print (round(spec_kurtosis[0],4))
[print(round(x, 4)) for x in Pax[1:]][0]
```

EI Processed Spectral features (accX):

2.398, 3.8924, 24.6841, 9.6303, 8.4867, 7.7793, 2.9963, 5.6242, 3.4198, 4.2735

Calculated features:

2.9069 8.5569 24.6844 9.6304 8.4865 7.7794 2.9964 5.6242 3.4198 4.2736

```
print("EI Processed Spectral features (accY): ")
print(features[16:26][0:]) # 13: 3+N_feat_axis;
                           # 26 = 2x N_feat_axis
print("\nCalculated features:")
print (round(spec_skew[1],4))
print (round(spec_kurtosis[1],4))
[print(round(x, 4)) for x in Pay[1:]][0]
```

EI Processed Spectral features (accY):

0.9426, -0.8039, 5.429, 0.999, 1.0315, 0.9459, 1.8117, 0.9088, 1.3302, 3.112

Calculated features:

1.1426 -0.3886 5.4289 0.999 1.0315 0.9458 1.8116 0.9088 1.3301 3.1121

```
print("EI Processed Spectral features (accZ): ")
print(features[29:][0:]) #29: 3+(2*N_feat_axis);
print("\nCalculated features:")
print (round(spec_skew[2],4))
print (round(spec_kurtosis[2],4))
[print(round(x, 4)) for x in Paz[1:]][0]
```

EI Processed Spectral features (accZ):

0.3117, -1.3812, 0.0606, 0.057, 0.0567, 0.0976, 0.194, 0.2574, 0.2083, 0.166

Calculated features:

0.3781 -1.4874 0.0606 0.057 0.0567 0.0976 0.194 0.2574 0.2083 0.166

Time-frequency domain

Wavelets

Wavelet is a powerful technique for analyzing signals with transient features or abrupt changes, such as spikes or edges, which are difficult to interpret with traditional Fourier-based methods.

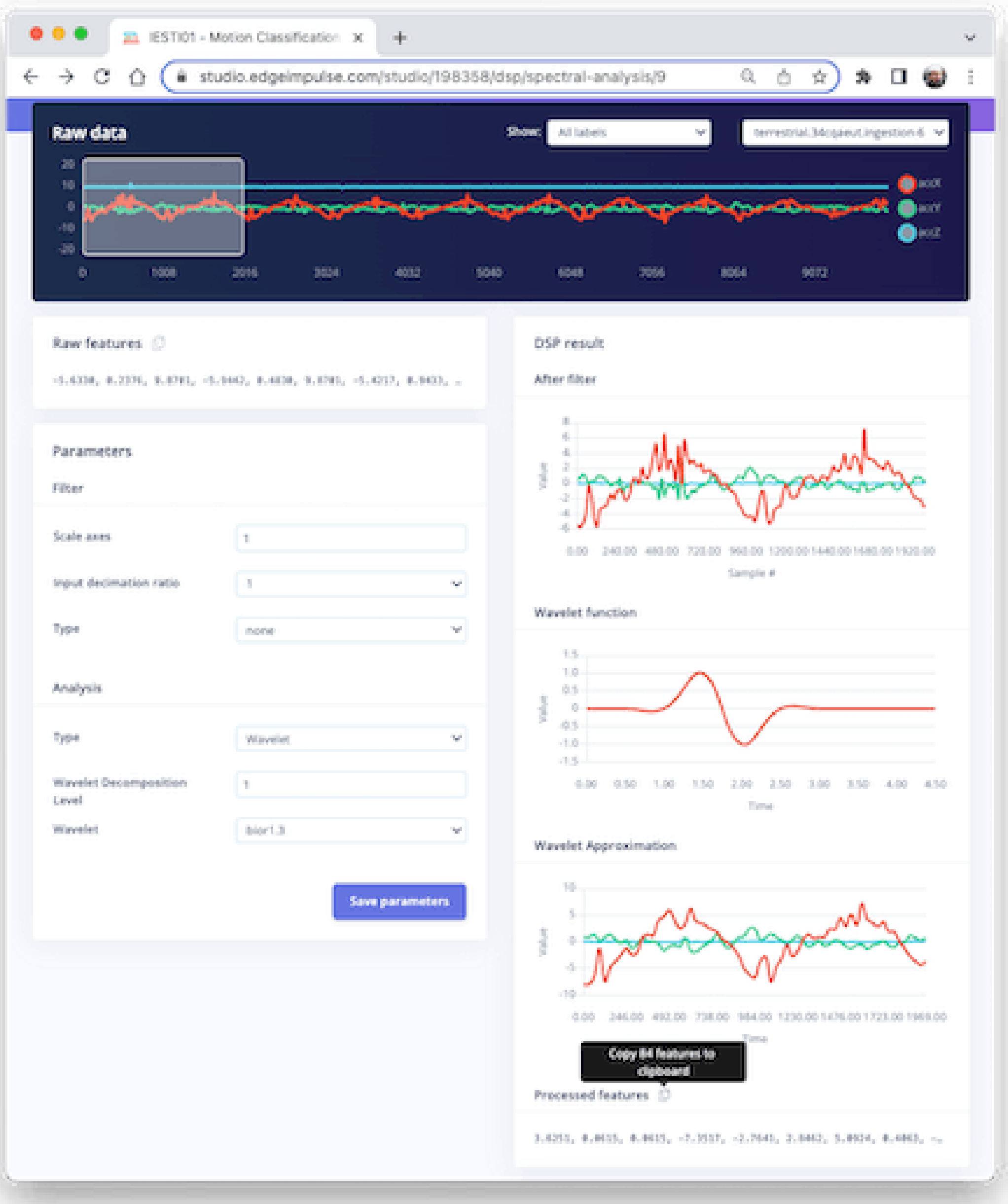
Wavelet transforms work by breaking down a signal into different frequency components and analyzing them individually. The transformation is achieved by convolving the signal with a **wavelet function**, a small waveform centered at a specific time and frequency. This process effectively decomposes the signal into different frequency bands, each of which can be analyzed separately.

One of the critical benefits of wavelet transforms is that they allow for time-frequency analysis, which means that they can reveal the frequency content of a signal as it changes over time. This makes them particularly useful for analyzing non-stationary signals, which vary over time.

Wavelets have many practical applications, including signal and image compression, denoising, feature extraction, and image processing.

Let's select Wavelet on the Spectral Features block in the same project:

- Type: Wavelet
- Wavelet Decomposition Level: 1
- Wavelet: bior1.3


IESTI01 - Motion Classification
studio.edgeimpulse.com/studio/198358/dsp/spectral-analysis/9
Raw data
Show: All labels
terrestrial.34cqaeut.ingestion-6
accX
accY
accZ
DSP result
After filter
Sample #
Value
Wavelet function
Time
Wavelet Approximation
Copy 84 features to clipboard
Save parameters

Raw features

-5.6330, 0.2376, 9.8781, -5.5442, 0.4830, 9.8781, -5.4217, 0.9433, …

Parameters

Filter

Scale axes: 1

Input decimation ratio: 1

Type: none

Analysis

Type: Wavelet

Wavelet Decomposition Level: 1

Wavelet: bior1.3

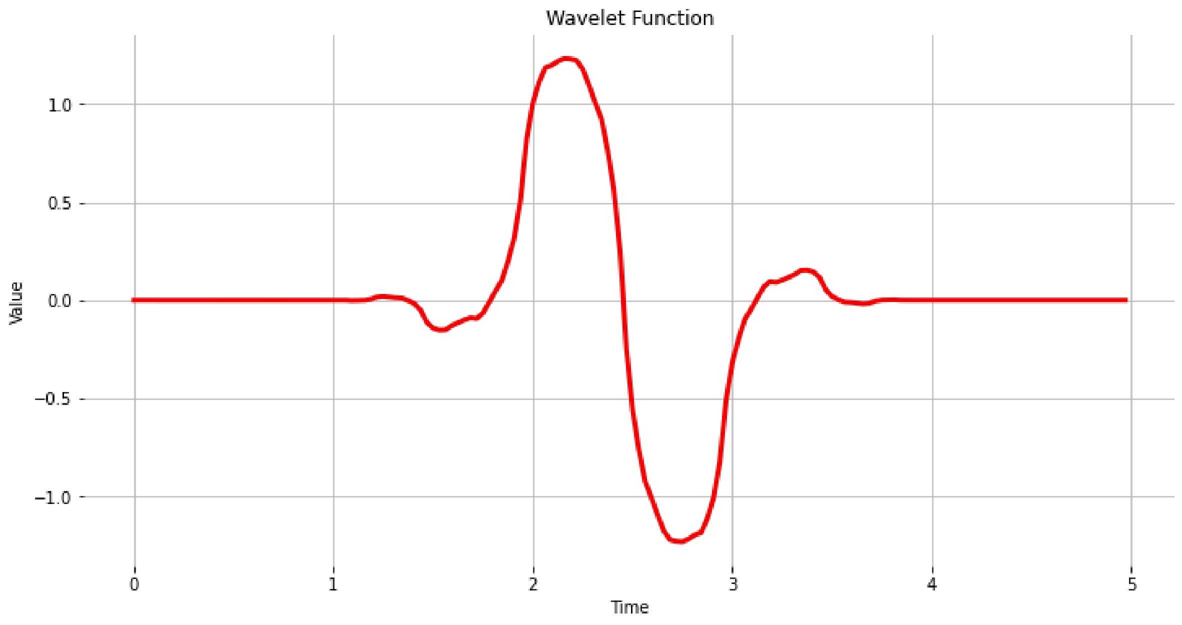
Processed features

3.4253, 0.0655, 0.4615, -7.3537, -2.7641, 2.8462, 5.8924, 0.4863, …

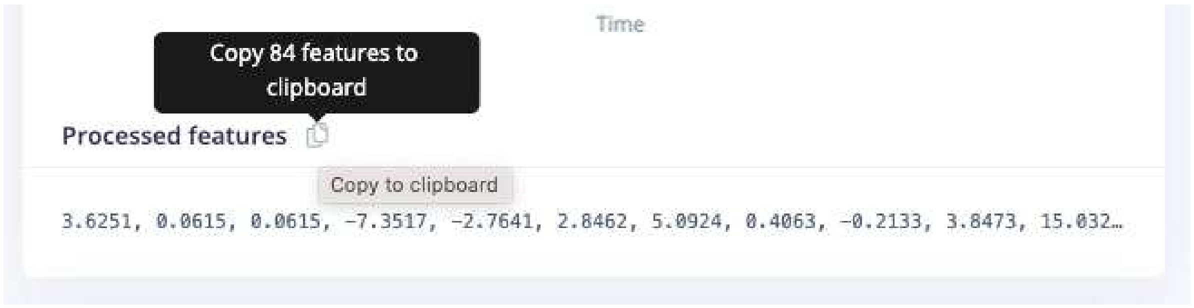
The Wavelet Function

```
wavelet_name='bior1.3'
num_layer = 1
```

```
wavelet = pywt.Wavelet(wavelet_name)
[phi_d,psi_d,phi_r,psi_r,x] = wavelet.wavefun(level=5)
plt.plot(x, psi_d, color='red')
plt.title('Wavelet Function')
plt.ylabel('Value')
plt.xlabel('Time')
plt.grid()
plt.box(False)
plt.show()
```



As we did before, let's copy and past the Processed Features:



```
3.6251, 0.0615, 0.0615, -7.3517, -2.7641, 2.8462, 5.0924, 0.4063, -0.2133, 3.8473, 15.032…
```

```
features = [
    3.6251, 0.0615, 0.0615,
    -7.3517, -2.7641, 2.8462,
    5.0924, ...
]
N_feat = len(features)
N_feat_axis = int(N_feat/n_sensors)
```

Edge Impulse computes the Discrete Wavelet Transform (DWT) for each one of the Wavelet Decomposition levels selected. After that, the features will be extracted.

In the case of **Wavelets**, the extracted features are *basic statistical values*, *crossing values*, and *entropy.* There are, in total, 14 features per layer as below:

- [11] Statiscal Features: **n5, n25, n75, n95, mean, median,** standard deviation **(std)**, variance **(var)** root mean square **(rms), kurtosis**, and skewness **(skew)**.
- [2] Crossing Features: Zero crossing rate **(zcross)** and mean crossing rate **(mcross)** are the times that the signal passes through the baseline ($y = 0$) and the average level (y = u) per unit of time, respectively
- [1] Complexity Feature: **Entropy** is a characteristic measure of the complexity of the signal

All the above 14 values are calculated for each Layer (including L0, the original signal)

- The total number of features varies depending on how you set the filter and the number of layers. For example, with [None] filtering and Level[1], the number of features per axis will be 14×2 (L0 and L1) $= 28$. For the three axes, we will have a total of 84 features.

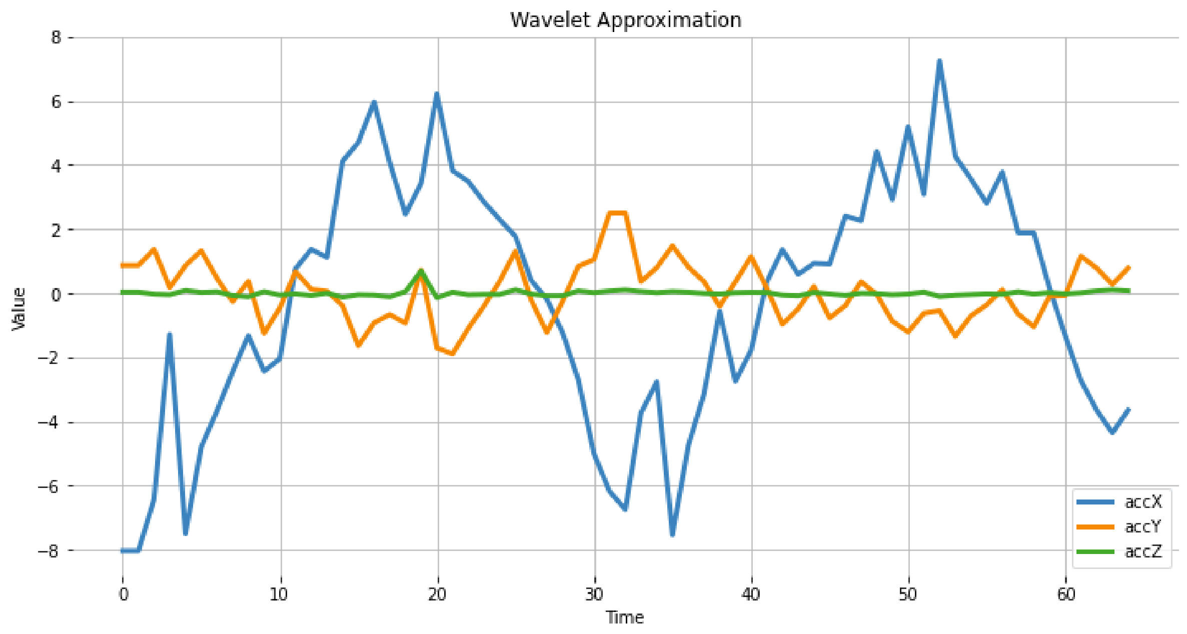
Wavelet Analysis

Wavelet analysis decomposes the signal (**accX, accY**, **and accZ**) into different frequency components using a set of filters, which separate these components into low-frequency (slowly varying parts of the signal containing long-term patterns), such as **accX_l1, accY_l1, accZ_l1** and, high-frequency (rapidly varying parts of the signal containing short-term patterns) components, such as **accX_d1, accY_d1, accZ_d1**, permitting the extraction of features for further analysis or classification.

Only the low-frequency components (approximation coefficients, or cA) will be used. In this example, we assume only one level (Single-level Discrete Wavelet Transform), where the function will return a tuple. With a multilevel decomposition, the "Multilevel 1D Discrete Wavelet Transform", the result will be a list (for detail, please see: Discrete Wavelet Transform (DWT))

```
(accX_l1, accX_d1) = pywt.dwt(accX, wavelet_name)
(accY_l1, accY_d1) = pywt.dwt(accY, wavelet_name)
```

```
(accZ_l1, accZ_d1) = pywt.dwt(accZ, wavelet_name)
sensors_l1 = [accX_l1, accY_l1, accZ_l1]

# Plot power spectrum versus frequency
plt.plot(accX_l1, label='accX')
plt.plot(accY_l1, label='accY')
plt.plot(accZ_l1, label='accZ')
plt.legend(loc='lower right')
plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Wavelet Approximation')
plt.grid()
plt.box(False)
plt.show()
```



Feature Extraction

Let's start with the basic statistical features. Note that we apply the function for both the original signals and the resultant cAs from the DWT:

```
def calculate_statistics(signal):
    n5 = np.percentile(signal, 5)
    n25 = np.percentile(signal, 25)
    n75 = np.percentile(signal, 75)
    n95 = np.percentile(signal, 95)
    median = np.percentile(signal, 50)
    mean = np.mean(signal)
    std = np.std(signal)
    var = np.var(signal)
    rms = np.sqrt(np.mean(np.square(signal)))
    return [n5, n25, n75, n95, median, mean, std, var, rms]

stat_feat_l0 = [calculate_statistics(x) for x in sensors]
stat_feat_l1 = [calculate_statistics(x) for x in sensors_l1]
```

The Skelness and Kurtosis:

```
skew_l0 = [skew(x, bias=False) for x in sensors]
skew_l1 = [skew(x, bias=False) for x in sensors_l1]
kurtosis_l0 = [kurtosis(x, bias=False) for x in sensors]
kurtosis_l1 = [kurtosis(x, bias=False) for x in sensors_l1]
```

Zero crossing (zcross) is the number of times the wavelet coefficient crosses the zero axis. It can be used to measure the signal's frequency content since high-frequency signals tend to have more zero crossings than low-frequency signals.

Mean crossing (mcross), on the other hand, is the number of times the wavelet coefficient crosses the mean of the signal. It can be used to measure the amplitude since high-amplitude signals tend to have more mean crossings than low-amplitude signals.

```
def getZeroCrossingRate(arr):
    my_array = np.array(arr)
    zcross = float(
        "{:.2f}".format(
          (((my_array[:-1] * my_array[1:]) < 0).sum()) / len(arr)
        )
    )
    return zcross

def getMeanCrossingRate(arr):
    mcross = getZeroCrossingRate(np.array(arr) - np.mean(arr))
    return mcross
```

```
def calculate_crossings(list):
    zcross=[]
    mcross=[]
    for i in range(len(list)):
        zcross_i = getZeroCrossingRate(list[i])
        zcross.append(zcross_i)
        mcross_i = getMeanCrossingRate(list[i])
        mcross.append(mcross_i)
    return zcross, mcross

cross_l0 = calculate_crossings(sensors)
cross_l1 = calculate_crossings(sensors_l1)
```

In wavelet analysis, **entropy** refers to the degree of disorder or randomness in the distribution of wavelet coefficients. Here, we used Shannon entropy, which measures a signal's uncertainty or randomness. It is calculated as the negative sum of the probabilities of the different possible outcomes of the signal multiplied by their base 2 logarithm. In the context of wavelet analysis, Shannon entropy can be used to measure the complexity of the signal, with higher values indicating greater complexity.

```
def calculate_entropy(signal, base=None):
    value, counts = np.unique(signal, return_counts=True)
    return entropy(counts, base=base)

entropy_l0 = [calculate_entropy(x) for x in sensors]
entropy_l1 = [calculate_entropy(x) for x in sensors_l1]
```

Let's now list all the wavelet features and create a list by layers.

```
L1_features_names = [
    "L1-n5", "L1-n25", "L1-n75", "L1-n95", "L1-median",
    "L1-mean", "L1-std", "L1-var", "L1-rms", "L1-skew",
    "L1-Kurtosis", "L1-zcross", "L1-mcross", "L1-entropy"
]

L0_features_names = [
    "L0-n5", "L0-n25", "L0-n75", "L0-n95", "L0-median",
    "L0-mean", "L0-std", "L0-var", "L0-rms", "L0-skew",
    "L0-Kurtosis", "L0-zcross", "L0-mcross", "L0-entropy"
]
```

```
all_feat_l0 = []
for i in range(len(axis)):
    feat_l0 = (
        stat_feat_l0[i]
        + [skew_l0[i]]
        + [kurtosis_l0[i]]
        + [cross_l0[0][i]]
        + [cross_l0[1][i]]
        + [entropy_l0[i]]
    )
    [print(axis[i] + ' +x+= ', round(y, 4))
       for x, y in zip(L0_features_names, feat_l0)][0]
    all_feat_l0.append(feat_l0)

all_feat_l0 = [
    item
    for sublist in all_feat_l0
    for item in sublist
]
print(f"\nAll L0 Features = {len(all_feat_l0)}")

all_feat_l1 = []
for i in range(len(axis)):
    feat_l1 = (
        stat_feat_l1[i]
        + [skew_l1[i]]
        + [kurtosis_l1[i]]
        + [cross_l1[0][i]]
        + [cross_l1[1][i]]
        + [entropy_l1[i]]
    )
    [print(axis[i]+' '+x+'= ', round(y, 4))
       for x,y in zip(L1_features_names, feat_l1)][0]
    all_feat_l1.append(feat_l1)

all_feat_l1 = [
    item
    for sublist in all_feat_l1
    for item in sublist
]
print(f"\nAll L1 Features = {len(all_feat_l1)}")
```

```
accX L0-n5=  -4.9364          accX L1-n5=  -7.3516
accX L0-n25=  -1.8429         accX L1-n25=  -2.7641
accX L0-n75=  1.8842          accX L1-n75=  2.8462
accX L0-n95=  3.8096          accX L1-n95=  5.0924
accX L0-median=  0.4058       accX L1-median=  0.4064
accX L0-mean=  -0.0           accX L1-mean=  -0.2133
accX L0-std=  2.7322          accX L1-std=  3.8473
accX L0-var=  7.4651          accX L1-var=  14.8015
accX L0-rms=  2.7322          accX L1-rms=  3.8532
accX L0-skew=  -0.099         accX L1-skew=  -0.2975
accX L0-Kurtosis=  -0.3475    accX L1-Kurtosis=  -0.7631
accX L0-zcross=  0.06         accX L1-zcross=  0.06
accX L0-mcross=  0.06         accX L1-mcross=  0.06
accX L0-entropy=  4.8283      accX L1-entropy=  4.1744
accY L0-n5=  -1.149           accY L1-n5=  -1.3234
accY L0-n25=  -0.4475         accY L1-n25=  -0.6492
accY L0-n75=  0.4814          accY L1-n75=  0.7844
accY L0-n95=  1.1491          accY L1-n95=  1.361
accY L0-median=  -0.0315      accY L1-median=  0.0659
accY L0-mean=  0.0            accY L1-mean=  0.0276
accY L0-std=  0.7833          accY L1-std=  0.9345
accY L0-var=  0.6136          accY L1-var=  0.8732
accY L0-rms=  0.7833          accY L1-rms=  0.9349
accY L0-skew=  0.1756         accY L1-skew=  0.2874
accY L0-Kurtosis=  1.2673     accY L1-Kurtosis=  0.0347
accY L0-zcross=  0.29         accY L1-zcross=  0.31
accY L0-mcross=  0.29         accY L1-mcross=  0.31
accY L0-entropy=  4.8283      accY L1-entropy=  4.1317
accZ L0-n5=  -0.1242          accZ L1-n5=  -0.1126
accZ L0-n25=  -0.0429         accZ L1-n25=  -0.0493
accZ L0-n75=  0.0349          accZ L1-n75=  0.0348
accZ L0-n95=  0.0839          accZ L1-n95=  0.1022
accZ L0-median=  -0.0112      accZ L1-median=  -0.0137
accZ L0-mean=  0.0            accZ L1-mean=  0.0025
accZ L0-std=  0.1383          accZ L1-std=  0.1053
accZ L0-var=  0.0191          accZ L1-var=  0.0111
accZ L0-rms=  0.1383          accZ L1-rms=  0.1053
accZ L0-skew=  6.9463         accZ L1-skew=  4.4095
accZ L0-Kurtosis=  68.1123    accZ L1-Kurtosis=  28.6586
accZ L0-zcross=  0.35         accZ L1-zcross=  0.4
accZ L0-mcross=  0.35         accZ L1-mcross=  0.37
accZ L0-entropy=  4.5649      accZ L1-entropy=  4.1531

All L0 Features = 42          All L1 Features = 42
```

Summary

Edge Impulse Studio is a powerful online platform that can handle the pre-processing task for us. Still, given our engineering perspective, we want to understand what is happening under the hood. This knowledge will help us find the best options and hyper-parameters for tuning our projects.

Daniel Situnayake wrote in his blog: "Raw sensor data is highly dimensional and noisy. Digital signal processing algorithms help us sift the signal from the noise. DSP is an essential part of embedded engineering, and many edge processors have on-board acceleration for DSP. As an ML engineer, learning basic DSP gives you superpowers for handling high-frequency time series data in your models." I recommend you read Dan's excellent post in its totality: nn to cpp: What you need to know about porting deep learning models to the edge.

Part I

key:backmatter

KWS Feature Engineering

Figure 8: *DALL · E 3 Prompt: 1950s style cartoon scene set in an audio research room. Two scientists, one holding a magnifying glass and the other taking notes, examine large charts pinned to the wall. These charts depict FFT graphs and time curves related to audio data analysis. The room has a retro ambiance, with wooden tables, vintage lamps, and classic audio analysis tools.*

Overview

In this hands-on tutorial, the emphasis is on the critical role that feature engineering plays in optimizing the performance of machine learning models applied to audio classification tasks, such as speech recognition. It is essential to be aware that the performance of any machine learning model relies heavily on the quality of features used, and we will deal with "under-the-hood" mechanics of feature extraction, mainly focusing on Mel-frequency Cepstral Coefficients (MFCCs), a cornerstone in the field of audio signal processing.

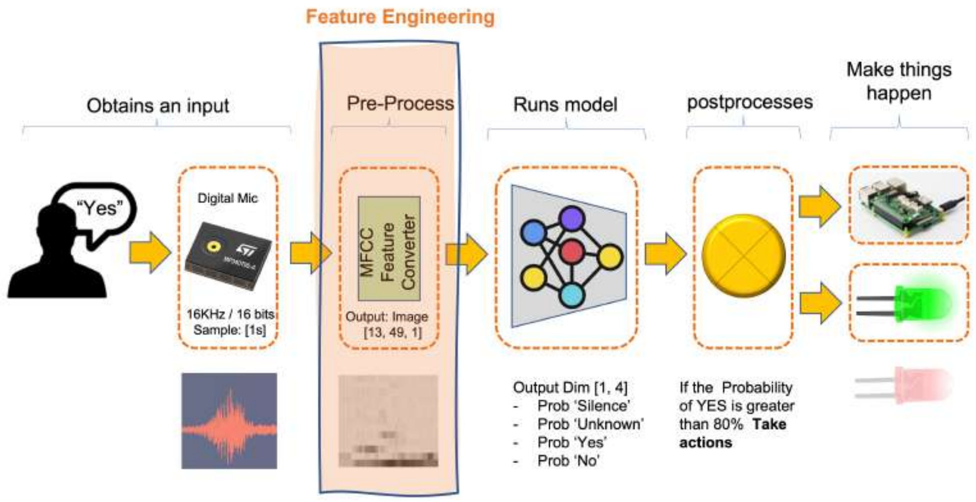
Machine learning models, especially traditional algorithms, don't understand audio waves. They understand numbers arranged in some meaningful way, i.e., features. These features encapsulate the characteristics of the audio signal, making it easier for models to distinguish between different sounds.

> This tutorial will deal with generating features specifically for audio classification. This can be particularly interesting for applying machine learning to a variety of audio data, whether for speech recognition, music categorization, insect classification based on wingbeat sounds, or other sound analysis tasks

The KWS

The most common TinyML application is Keyword Spotting (KWS), a subset of the broader field of speech recognition. While general speech recognition transcribes all spoken words into text, Keyword Spotting focuses on detecting specific "keywords" or "wake words" in a continuous audio stream. The system is trained to recognize these keywords as predefined phrases or words, such as *yes* or *no*. In short, KWS is a specialized form of speech recognition with its own set of challenges and requirements.

Here a typical KWS Process using MFCC Feature Converter:



Applications of KWS

- **Voice Assistants**: In devices like Amazon's Alexa or Google Home, KWS is used to detect the wake word ("Alexa" or "Hey Google") to activate the device.
- **Voice-Activated Controls**: In automotive or industrial settings, KWS can be used to initiate specific commands like "Start engine" or "Turn off lights."
- **Security Systems**: Voice-activated security systems may use KWS to authenticate users based on a spoken passphrase.
- **Telecommunication Services**: Customer service lines may use KWS to route calls based on spoken keywords.

Differences from General Speech Recognition

- **Computational Efficiency**: KWS is usually designed to be less computationally intensive than full speech recognition, as it only needs to recognize a small set of phrases.
- **Real-time Processing**: KWS often operates in real-time and is optimized for low-latency detection of keywords.
- **Resource Constraints**: KWS models are often designed to be lightweight, so they can run on devices with limited computational resources, like microcontrollers or mobile phones.
- **Focused Task**: While general speech recognition models are trained to handle a broad range of vocabulary and accents, KWS models are fine-tuned to recognize specific keywords, often in noisy environments accurately.

Overview to Audio Signals

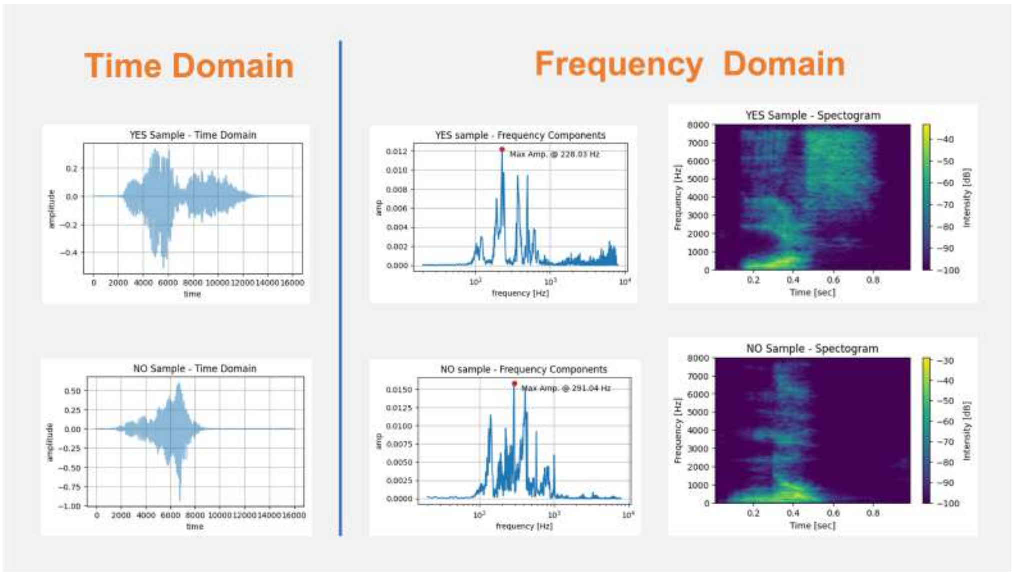
Understanding the basic properties of audio signals is crucial for effective feature extraction and, ultimately, for successfully applying machine learning algorithms in audio classification tasks. Audio signals are complex waveforms that capture fluctuations in air pressure over time. These signals can be characterized by several fundamental attributes: sampling rate, frequency, and amplitude.

- **Frequency and Amplitude**: Frequency refers to the number of oscillations a waveform undergoes per unit time and is also measured in Hz. In the context of audio signals, different frequencies correspond to different pitches. Amplitude, on the other hand, measures the magnitude of the oscillations and correlates with the loudness of the sound. Both frequency and amplitude are essential features that capture audio signals' tonal and rhythmic qualities.
- **Sampling Rate**: The sampling rate, often denoted in Hertz (Hz), defines the number of samples taken per second when digitizing an analog signal. A higher sampling rate allows for a more accurate digital representation of the signal but also demands more computational resources for processing. Typical sampling rates include 44.1 kHz for CD-quality audio and 16 kHz or 8 kHz for speech recognition tasks. Understanding the trade-offs in selecting an appropriate sampling rate is essential for balancing accuracy and computational efficiency. In general, with TinyML projects, we work with 16 kHz. Although music tones can be heard at frequencies up to 20 kHz, voice maxes out at 8 kHz. Traditional telephone systems use an 8 kHz sampling frequency.

  For an accurate representation of the signal, the sampling rate must be at least twice the highest frequency present in the signal.

- **Time Domain vs. Frequency Domain**: Audio signals can be analyzed in the time and frequency domains. In the time domain, a signal is represented as a waveform where the amplitude is plotted against time. This representation helps to observe temporal features like onset and duration but the signal's tonal characteristics are not well evidenced. Conversely, a frequency domain representation provides a view of the signal's constituent frequencies and their respective amplitudes, typically obtained via a Fourier Transform. This is invaluable for tasks that require understanding the signal's spectral content, such as identifying musical notes or speech phonemes (our case).

The image below shows the words `YES` and `NO` with typical representations in the Time (Raw Audio) and Frequency domains:



Why Not Raw Audio?

While using raw audio data directly for machine learning tasks may seem tempting, this approach presents several challenges that make it less suitable for building robust and efficient models.

Using raw audio data for Keyword Spotting (KWS), for example, on TinyML devices poses challenges due to its high dimensionality (using a 16 kHz sampling rate), computational complexity for capturing temporal features, susceptibility to noise, and lack of semantically meaningful features, making feature extraction techniques like MFCCs a more practical choice for resource-constrained applications.

Here are some additional details of the critical issues associated with using raw audio:

- **High Dimensionality**: Audio signals, especially those sampled at high rates, result in large amounts of data. For example, a 1-second audio clip sampled at 16 kHz will have 16,000 individual data points. High-dimensional data increases computational complexity, leading to longer training times and higher computational costs, making it impractical for resource-constrained environments. Furthermore, the wide dynamic range of audio signals requires a significant amount of bits per sample, while conveying little useful information.
- **Temporal Dependencies**: Raw audio signals have temporal structures that simple machine learning models may find hard to capture. While recurrent neural networks like LSTMs can model such dependencies, they are computationally intensive and tricky to train on tiny devices.

- **Noise and Variability**: Raw audio signals often contain background noise and other non-essential elements affecting model performance. Additionally, the same sound can have different characteristics based on various factors such as distance from the microphone, the orientation of the sound source, and acoustic properties of the environment, adding to the complexity of the data.
- **Lack of Semantic Meaning**: Raw audio doesn't inherently contain semantically meaningful features for classification tasks. Features like pitch, tempo, and spectral characteristics, which can be crucial for speech recognition, are not directly accessible from raw waveform data.
- **Signal Redundancy**: Audio signals often contain redundant information, with certain portions of the signal contributing little to no value to the task at hand. This redundancy can make learning inefficient and potentially lead to overfitting.

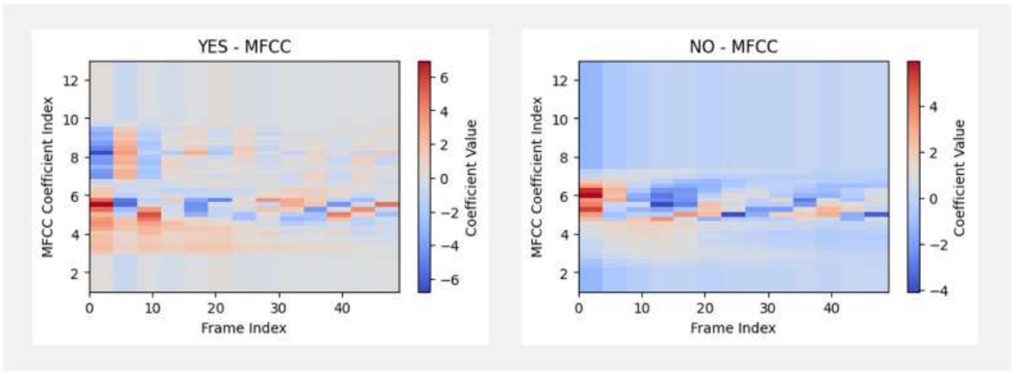
For these reasons, feature extraction techniques such as Mel-frequency Cepstral Coefficients (MFCCs), Mel-Frequency Energies (MFEs), and simple Spectograms are commonly used to transform raw audio data into a more manageable and informative format. These features capture the essential characteristics of the audio signal while reducing dimensionality and noise, facilitating more effective machine learning.

Overview to MFCCs

What are MFCCs?

Mel-frequency Cepstral Coefficients (MFCCs) are a set of features derived from the spectral content of an audio signal. They are based on human auditory perceptions and are commonly used to capture the phonetic characteristics of an audio signal. The MFCCs are computed through a multi-step process that includes pre-emphasis, framing, windowing, applying the Fast Fourier Transform (FFT) to convert the signal to the frequency domain, and finally, applying the Discrete Cosine Transform (DCT). The result is a compact representation of the original audio signal's spectral characteristics.

The image below shows the words `YES` and `NO` in their MFCC representation:



This video explains the Mel Frequency Cepstral Coefficients (MFCC) and how to compute them.

Why are MFCCs important?

MFCCs are crucial for several reasons, particularly in the context of Keyword Spotting (KWS) and TinyML:

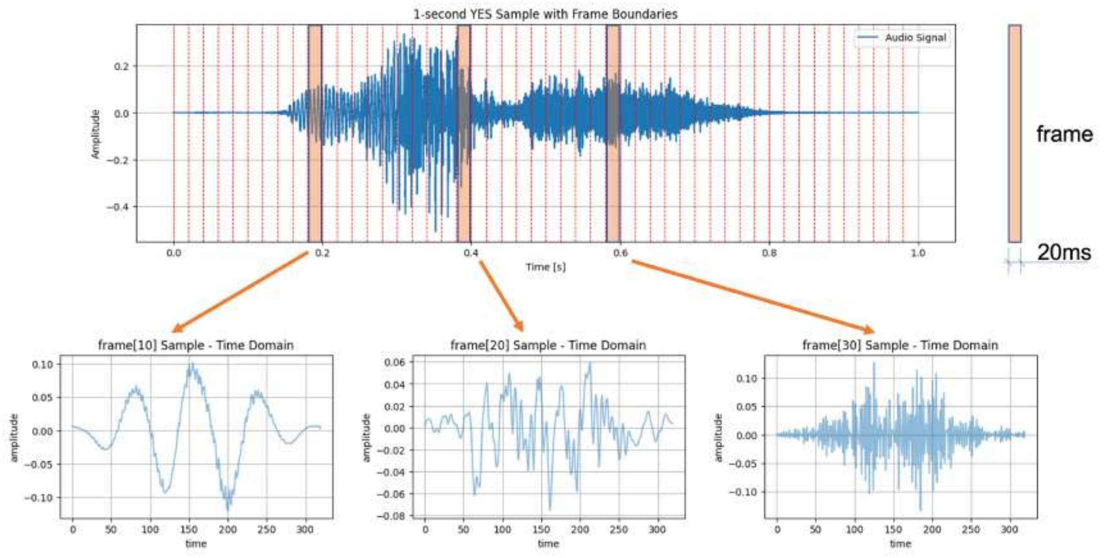
- **Dimensionality Reduction**: MFCCs capture essential spectral characteristics of the audio signal while significantly reducing the dimensionality of the data, making it ideal for resource-constrained TinyML applications.
- **Robustness**: MFCCs are less susceptible to noise and variations in pitch and amplitude, providing a more stable and robust feature set for audio classification tasks.
- **Human Auditory System Modeling**: The Mel scale in MFCCs approximates the human ear's response to different frequencies, making them practical for speech recognition where human-like perception is desired.
- **Computational Efficiency**: The process of calculating MFCCs is computationally efficient, making it well-suited for real-time applications on hardware with limited computational resources.

In summary, MFCCs offer a balance of information richness and computational efficiency, making them popular for audio classification tasks, particularly in constrained environments like TinyML.

Computing MFCCs

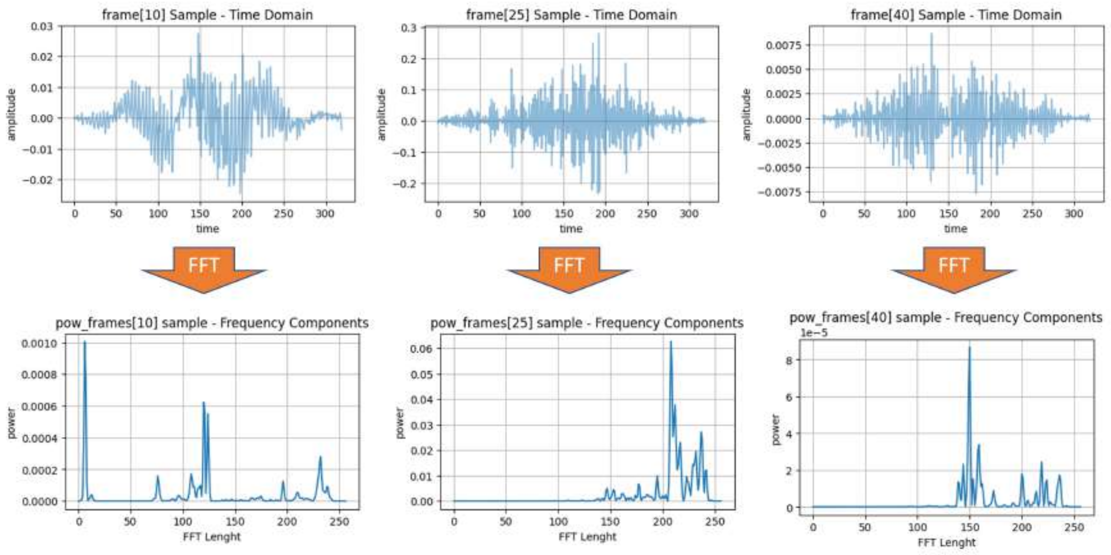
The computation of Mel-frequency Cepstral Coefficients (MFCCs) involves several key steps. Let's walk through these, which are particularly important for Keyword Spotting (KWS) tasks on TinyML devices.

- **Pre-emphasis**: The first step is pre-emphasis, which is applied to accentuate the high-frequency components of the audio signal and balance the frequency spectrum. This is achieved by applying a filter that amplifies the difference between consecutive samples. The formula for pre-emphasis is: $y(t) = x(t) - \alpha x(t-1)$, where α is the pre-emphasis factor, typically around 0.97.
- **Framing**: Audio signals are divided into short frames (the *frame length*), usually 20 to 40 milliseconds. This is based on the assumption that frequencies in a signal are stationary over a short period. Framing helps in analyzing the signal in such small time slots. The *frame stride* (or step) will displace one frame and the adjacent. Those steps could be sequential or overlapped.
- **Windowing**: Each frame is then windowed to minimize the discontinuities at the frame boundaries. A commonly used window function is the Hamming window. Windowing prepares the signal for a Fourier transform by minimizing the edge effects. The image below shows three frames (10, 20, and 30) and the time samples after windowing (note that the frame length and frame stride are 20 ms):

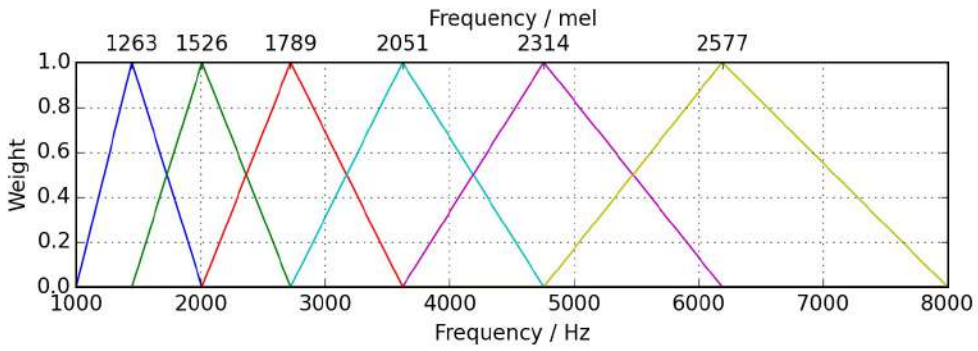


- **Fast Fourier Transform (FFT)** The Fast Fourier Transform (FFT) is applied to each windowed frame to convert it from the time domain to the frequency domain. The FFT gives us a complex-valued representation that includes both magnitude and phase information. However, for MFCCs, only the magnitude is used to calculate the Power Spectrum. The power spectrum is the square of the magnitude spectrum and measures the energy present at each frequency component.

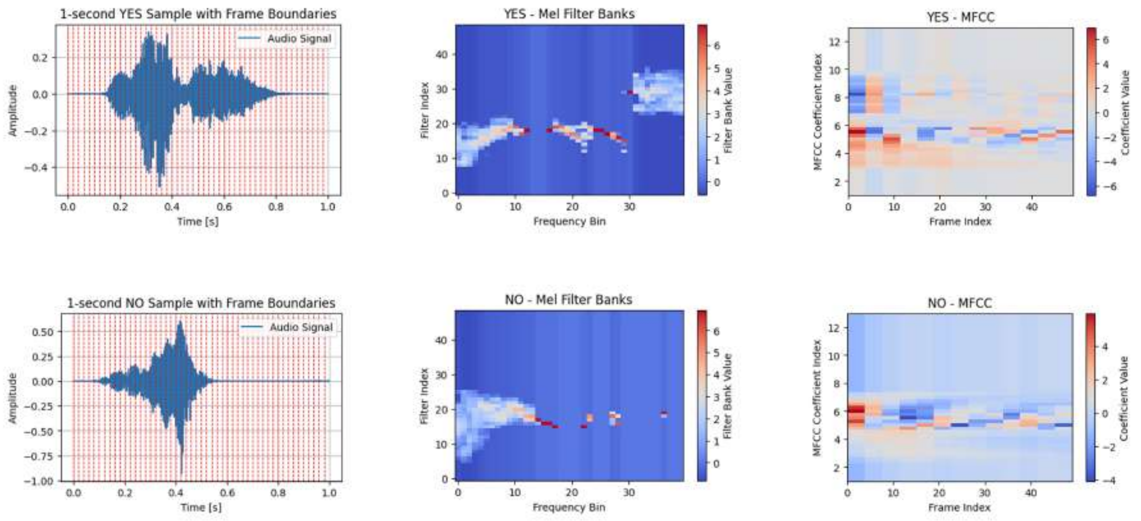
The power spectrum $P(f)$ of a signal $x(t)$ is defined as $P(f) = |X(f)|^2$, where $X(f)$ is the Fourier Transform of $x(t)$. By squaring the magnitude of the Fourier Transform, we emphasize *stronger* frequencies over *weaker* ones, thereby capturing more relevant spectral characteristics of the audio signal. This is important in applications like audio classification, speech recognition, and Keyword Spotting (KWS), where the focus is on identifying distinct frequency patterns that characterize different classes of audio or phonemes in speech.



- **Mel Filter Banks**: The frequency domain is then mapped to the Mel scale, which approximates the human ear's response to different frequencies. The idea is to extract more features (more filter banks) in the lower frequencies and less in the high frequencies. Thus, it performs well on sounds distinguished by the human ear. Typically, 20 to 40 triangular filters extract the Mel-frequency energies. These energies are then log-transformed to convert multiplicative factors into additive ones, making them more suitable for further processing.



- **Discrete Cosine Transform (DCT)**: The last step is to apply the Discrete Cosine Transform (DCT) to the log Mel energies. The DCT helps to decorrelate the energies, effectively compressing the data and retaining only the most discriminative features. Usually, the first 12-13 DCT coefficients are retained, forming the final MFCC feature vector.



Hands-On using Python

Let's apply what we discussed while working on an actual audio sample. Open the notebook on Google CoLab and extract the MLCC features on your audio samples: [Open In Colab]

Summary

What Feature Extraction technique should we use?

Mel-frequency Cepstral Coefficients (MFCCs), Mel-Frequency Energies (MFEs), or Spectrogram are techniques for representing audio data, which are often helpful in different contexts.

In general, MFCCs are more focused on capturing the envelope of the power spectrum, which makes them less sensitive to fine-grained spectral details but more robust to noise. This is often desirable for speech-related tasks. On the other hand, spectrograms or MFEs preserve more detailed frequency information, which can be advantageous in tasks that require discrimination based on fine-grained spectral content.

MFCCs are particularly strong for

1. **Speech Recognition**: MFCCs are excellent for identifying phonetic content in speech signals.
2. **Speaker Identification**: They can be used to distinguish between different speakers based on voice characteristics.
3. **Emotion Recognition**: MFCCs can capture the nuanced variations in speech indicative of emotional states.
4. **Keyword Spotting**: Especially in TinyML, where low computational complexity and small feature size are crucial.

Spectrograms or MFEs are often more suitable for

1. **Music Analysis**: Spectrograms can capture harmonic and timbral structures in music, which is essential for tasks like genre classification, instrument recognition, or music transcription.
2. **Environmental Sound Classification**: In recognizing non-speech, environmental sounds (e.g., rain, wind, traffic), the full spectrogram can provide more discriminative features.
3. **Birdsong Identification**: The intricate details of bird calls are often better captured using spectrograms.
4. **Bioacoustic Signal Processing**: In applications like dolphin or bat call analysis, the fine-grained frequency information in a spectrogram can be essential.
5. **Audio Quality Assurance**: Spectrograms are often used in professional audio analysis to identify unwanted noises, clicks, or other artifacts.

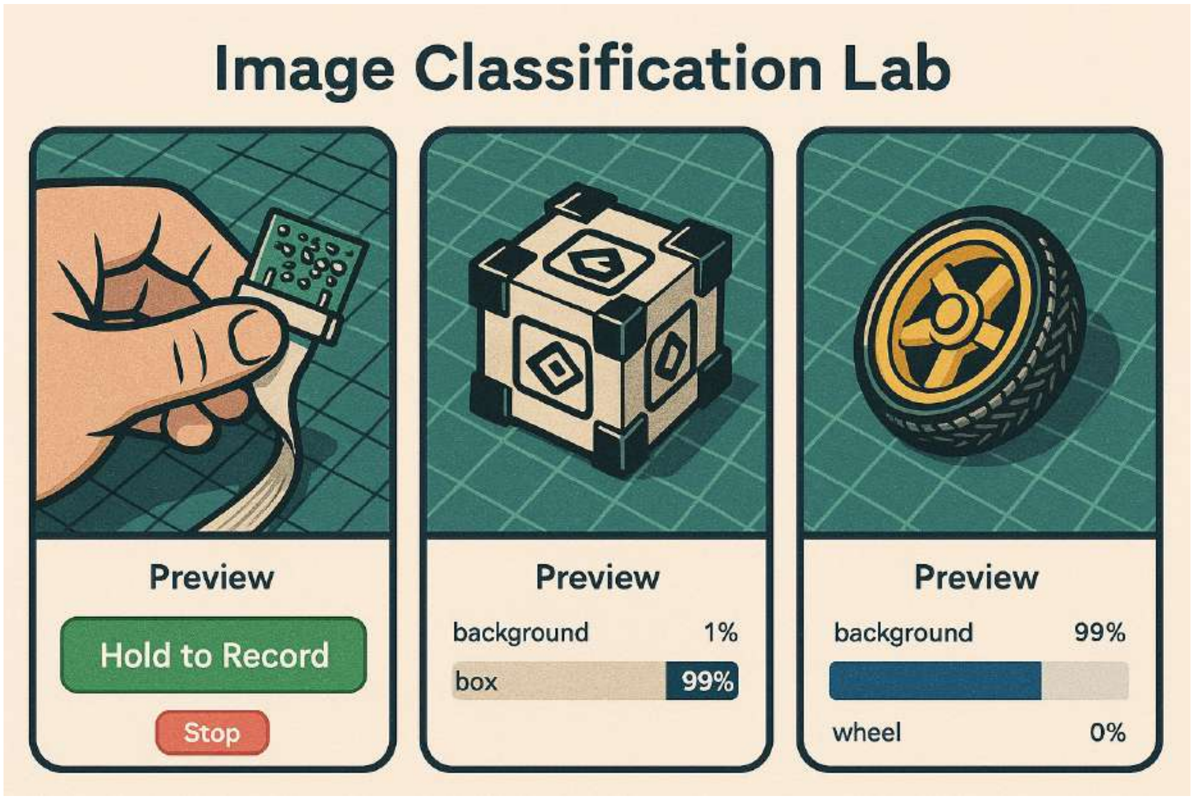
Resources

- Audio_Data_Analysis Colab Notebook

#

Grove Vision AI V2

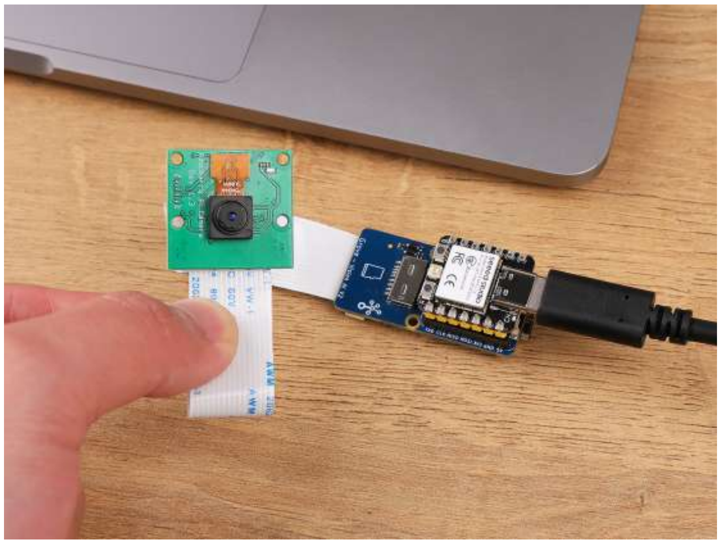
Setup and No-Code Applications



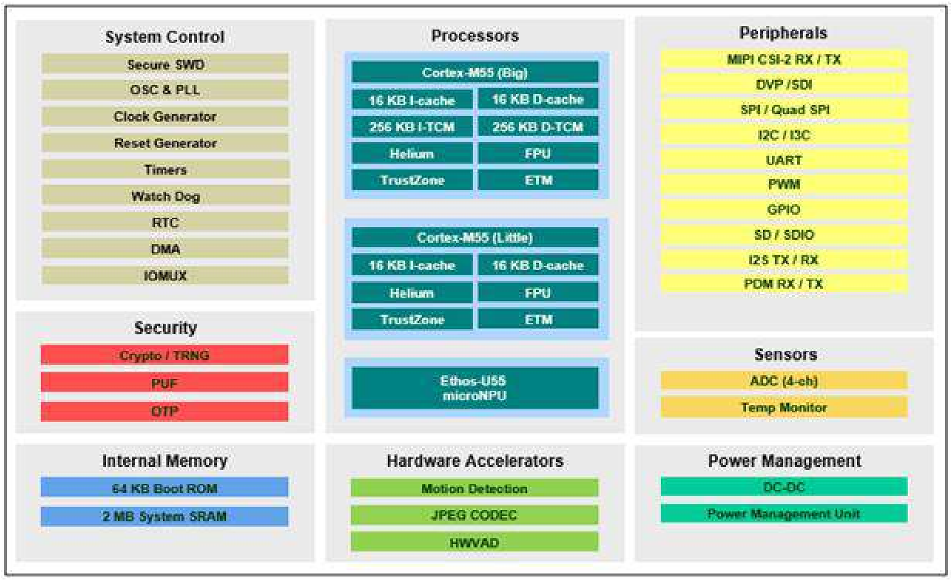
In this Lab, we will explore computer vision (CV) applications using the Seeed Studio *Grove Vision AI Module V2*, a powerful yet compact device specifically designed for embedded machine learning applications. Based on the **Himax WiseEye2** chip, this module is designed to enable AI capabilities on edge devices, making it an ideal tool for Edge Machine Learning (ML) applications.

Introduction

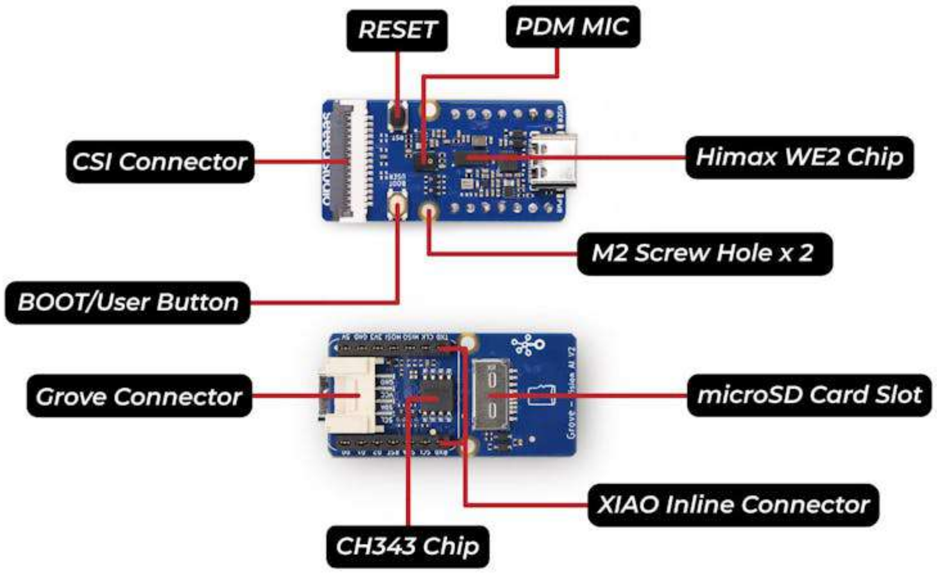
Grove Vision AI Module (V2) Overview



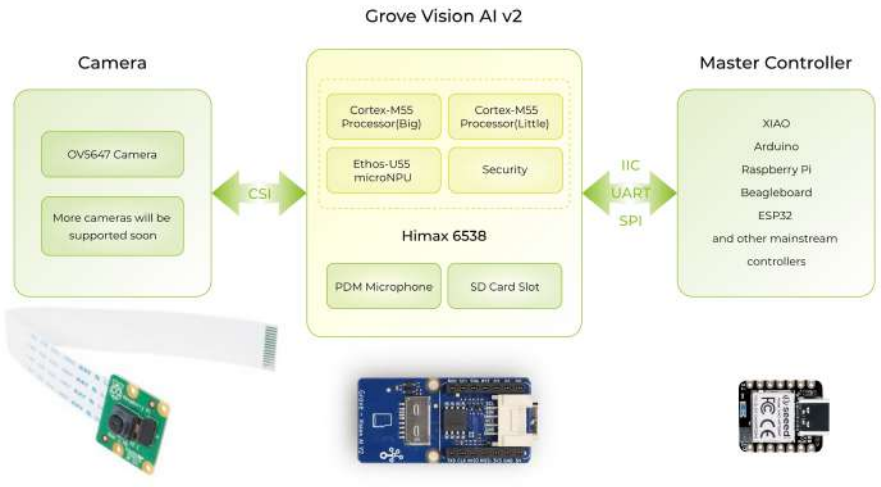
The Grove Vision AI (V2) is an MCU-based vision AI module that utilizes a Himax WiseEye2 HX6538 processor featuring a **dual-core Arm Cortex-M55 and an integrated ARM Ethos-U55 neural network unit**. The Arm Ethos-U55 is a machine learning (ML) processor class, specifically designed as a microNPU, to accelerate ML inference in area-constrained embedded and IoT devices. The Ethos-U55, combined with the AI-capable Cortex-M55 processor, provides a 480x uplift in ML performance over existing Cortex-M-based systems. Its clock frequency is 400 MHz, and its internal system memory (SRAM) is configurable, with a maximum capacity of 2.4 MB.



Note: Based on Seeed Studio documentation, besides the Himax internal memory of 2.5MB (2.4MB SRAM + 64KB ROM), the Grove Vision AI (V2) is also equipped with a 16MB/133 MHz external flash.

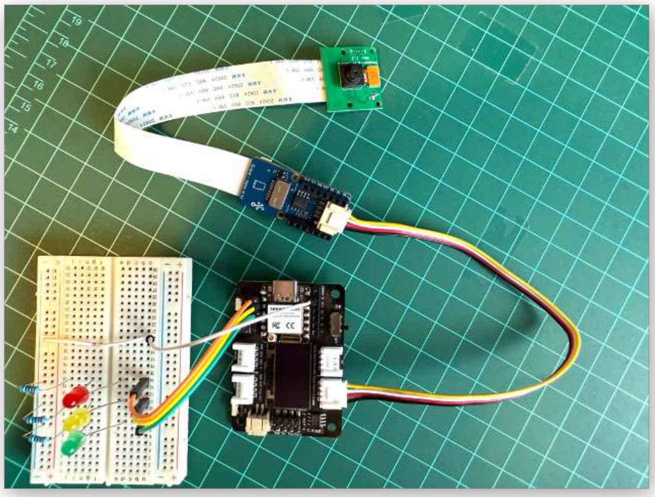


Below is a block Diagram of the Grove Vision AI (V2) system, including a camera and a master controller.

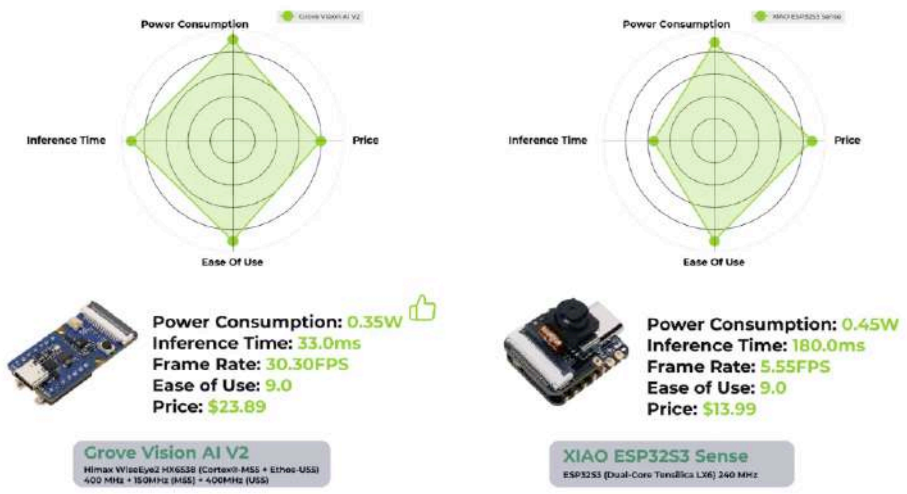


With interfaces like **IIC, UART, SPI, and Type-C,** the Grove Vision AI (V2) can be easily connected to devices such as **XIAO, Raspberry Pi, BeagleBoard**, **and ESP-based products** for further development. For instance, integrating Grove Vision AI V2 with one of the devices from the XIAO family makes it easy to access the data resulting from inference on the device through the Arduino IDE or MicroPython, and conveniently connect to the cloud or dedicated servers, such as Home Assistance.

> Using the **I2C Grove connector**, the Grove Vision AI V2 can be easily connected with any Master Device.



Besides performance, another area to comment on is **Power Consumption**. For example, in a comparative test against the XIAO ESP32S3 Sense, running Swift-YOLO Tiny 96x96, despite achieving higher performance (30 FPS vs. 5.5 FPS), the Grove Vision AI V2 exhibited lower power consumption (0.35 W vs. 0.45 W) when compared with the XIAO ESP32S3 Sense.

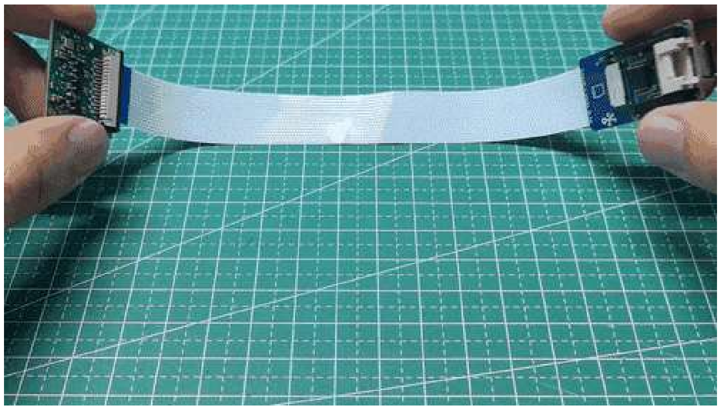


The above comparison (and with other devices) can be found in the article 2024 MCU AI Vision Boards: Performance Comparison, which confirms the power of Grove Vision AI (V2).

Camera Installation

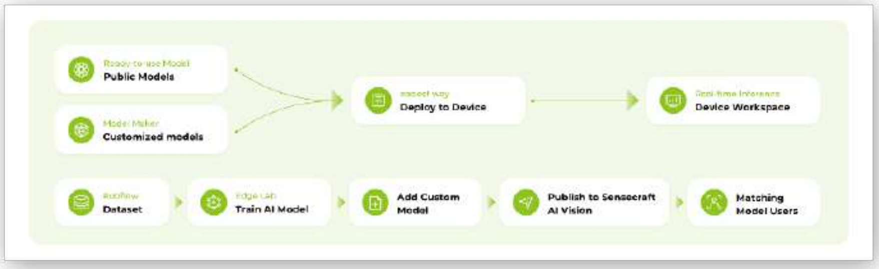
Having the Grove Vision AI (V2) and camera ready, you can connect, for example, a **Raspberry Pi OV5647 Camera Module** via the CSI cable.

When connecting, please pay attention to the direction of the row of pins and ensure they are plugged in correctly, not in the opposite direction.



The SenseCraft AI Studio

The SenseCraft AI Studio is a robust platform that offers a wide range of AI models compatible with various devices, including the XIAO ESP32S3 Sense and the **Grove Vision AI V2**. In this lab, we will walk through the process of using an AI model with the Grove Vision AI V2 and preview the model's output. We will also explore some key concepts, settings, and how to optimize the model's performance.



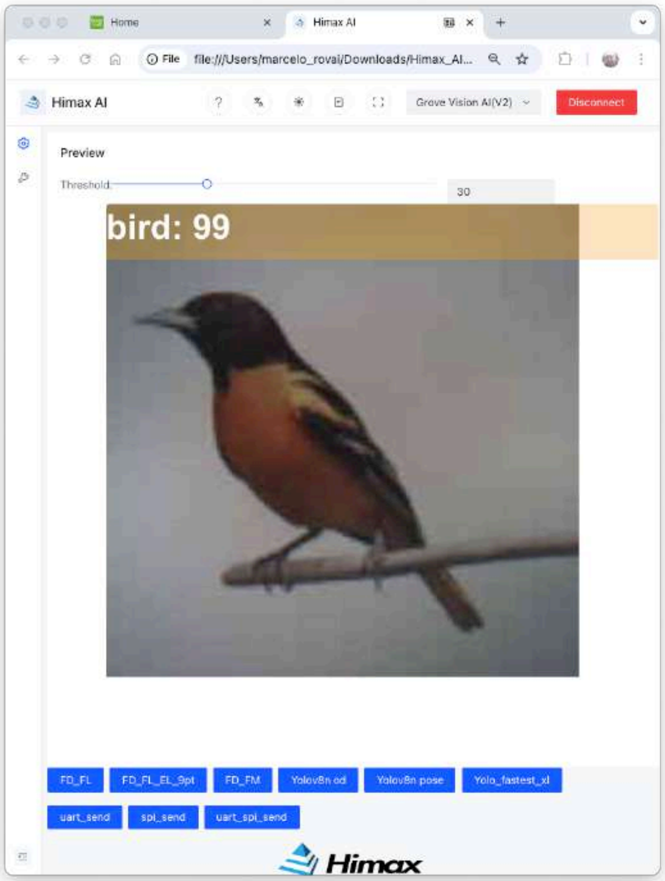
Models can also be deployed using the **SenseCraft Web Toolkit**, a simplified version of the SenseCraft AI Studio.

> We can start using the SenseCraft Web Toolkit for simplicity, or go directly to the SenseCraft AI Studio, which has more resources.

The SenseCraft Web-Toolkit

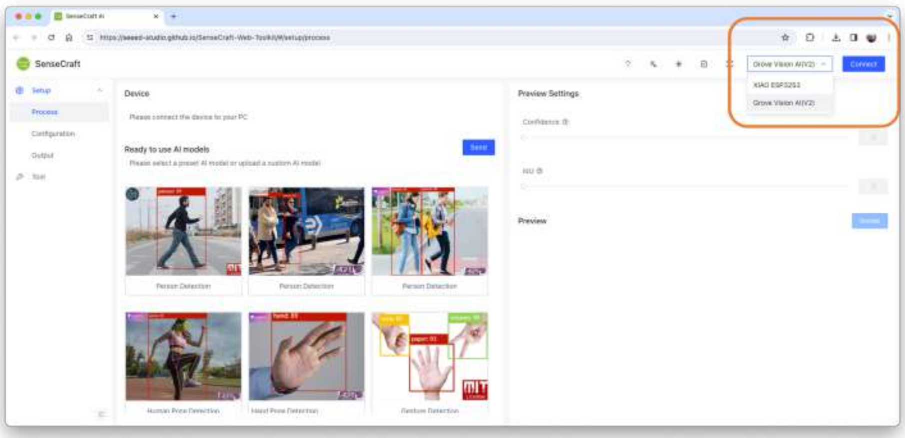
The SenseCraft Web Toolkit is a visual model deployment tool included in the SSCMA(Seeed SenseCraft Model Assistant). This tool enables us to deploy models to various platforms with ease through simple operations. The tool offers a user-friendly interface and does not require any coding.

The SenseCraft Web Toolkit is based on the Himax AI Web Toolkit, which can (**optionally**) be downloaded from here. Once downloaded and unzipped to the local PC, double-click `index.html` to run it locally.

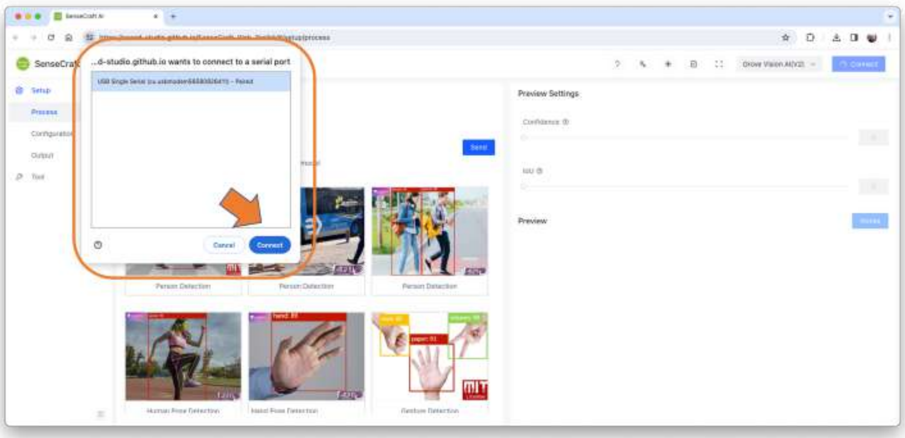


But in our case, let's follow the steps below to start the **SenseCraft-Web-Toolkit**:

- Open the SenseCraft-Web-Toolkit website on a web browser as **Chrome**.
- Connect Grove Vision AI (V2) to your computer using a Type-C cable.
- Having the XIAO connected, select it as below:



- Select the device/Port and press `[Connect]`:



Note: The **WebUSB tool** may not function correctly in certain browsers, such as Safari. Use Chrome instead.

We can try several Basic Computer Vision models previously uploaded by Seeed Studio. Passing the cursor over the AI models, we can have some information about them, such as name, description, **category** (Image Classification, Object Detection, or Pose/Keypoint Detection), the **algorithm** (like YOLO V5 or V8, FOMO, MobileNet V2, etc.) and **metrics** (Accuracy or mAP).

We can choose one of those ready-to-use AI models by clicking on it and pressing the `[Send]` button, or upload our model.

For the **SenseCraft AI** platform, follow the instructions here.

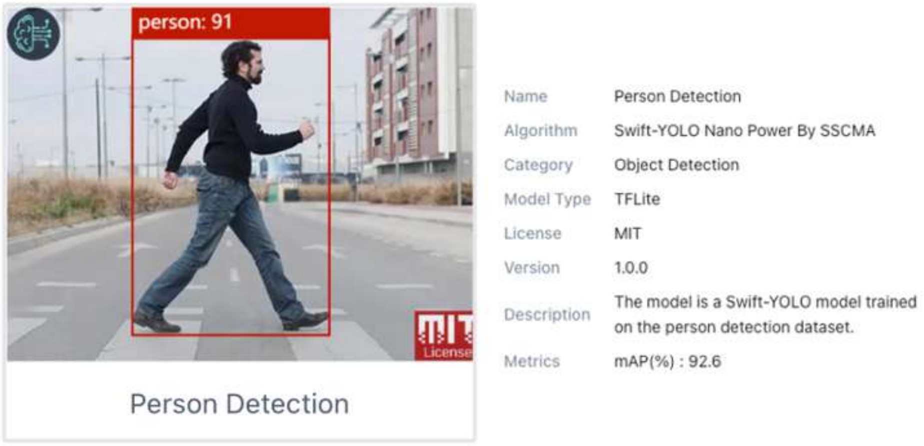
Exploring CV AI models

Object Detection

Object detection is a pivotal technology in computer vision that focuses on identifying and locating objects within digital images or video frames. Unlike image classification, which categorizes an entire image into a single label, object detection recognizes multiple objects within the image and determines their precise locations, typically represented by bounding boxes. This capability is crucial for a wide range of applications, including autonomous vehicles, security, surveillance systems, and augmented reality, where understanding the context and content of the visual environment is essential.

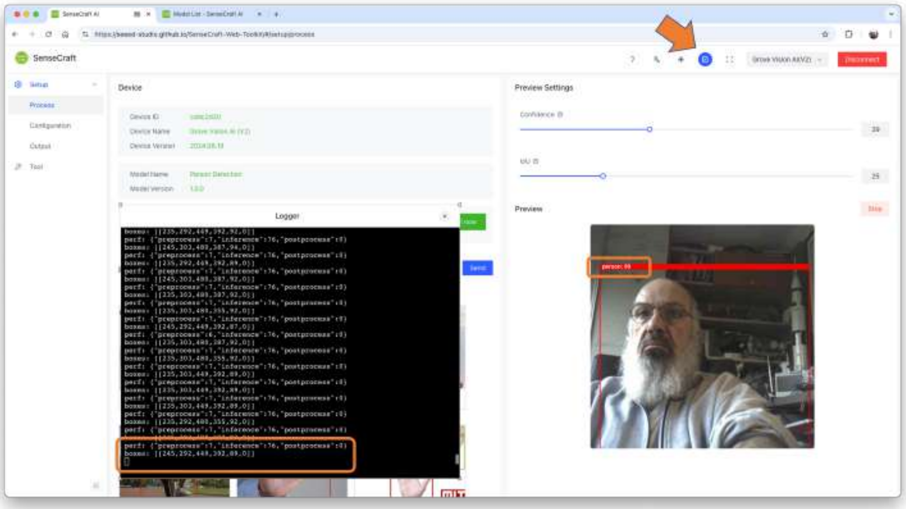
Common architectures that have set the benchmark in object detection include the YOLO (You Only Look Once), SSD (Single Shot MultiBox Detector), FOMO (Faster Objects, More Objects), and Faster R-CNN (Region-based Convolutional Neural Networks) models.

Let's choose one of the ready-to-use AI models, such as **Person Detection**, which was trained using the Swift-YOLO algorithm.



Name	Person Detection
Algorithm	Swift-YOLO Nano Power By SSCMA
Category	Object Detection
Model Type	TFLite
License	MIT
Version	1.0.0
Description	The model is a Swift-YOLO model trained on the person detection dataset.
Metrics	mAP(%) : 92.6

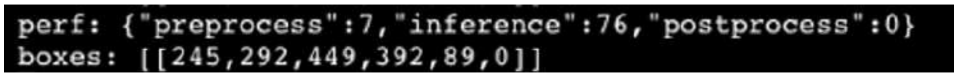
Once the model is uploaded successfully, you can see the live feed from the Grove Vision AI (V2) camera in the Preview area on the right. Also, the inference details can be shown on the Serial Monitor by clicking on the `[Device Log]` button at the top.



```
boxes: [[235,292,449,392,92,0]]
perf: {"preprocess":7,"inference":76,"postprocess":0}
boxes: [[245,303,488,387,94,0]]
perf: {"preprocess":7,"inference":76,"postprocess":0}
boxes: [[235,292,449,392,89,0]]
perf: {"preprocess":7,"inference":76,"postprocess":0}
boxes: [[245,303,488,387,92,0]]
perf: {"preprocess":7,"inference":76,"postprocess":0}
boxes: [[235,303,488,387,92,0]]
perf: {"preprocess":7,"inference":76,"postprocess":0}
boxes: [[235,303,488,355,92,0]]
perf: {"preprocess":7,"inference":76,"postprocess":0}
boxes: [[245,292,449,392,87,0]]
perf: {"preprocess":6,"inference":76,"postprocess":0}
boxes: [[235,303,488,387,92,0]]
perf: {"preprocess":7,"inference":76,"postprocess":0}
boxes: [[235,303,488,355,92,0]]
perf: {"preprocess":7,"inference":76,"postprocess":0}
boxes: [[235,303,449,392,89,0]]
perf: {"preprocess":7,"inference":76,"postprocess":0}
boxes: [[235,303,449,392,89,0]]
perf: {"preprocess":7,"inference":76,"postprocess":0}
boxes: [[235,303,449,392,89,0]]
perf: {"preprocess":7,"inference":76,"postprocess":0}
boxes: [[235,292,488,355,92,0]]
perf: {"preprocess":7,"inference":76,"postprocess":0}
[illegible]
perf: {"preprocess":7,"inference":76,"postprocess":0}
boxes: [[245,292,449,392,89,0]]
```

In the SenseCraft AI Studio, the Device Logger is always on the screen.

Pointing the camera at me, only one person was detected, so that the model output will be a single "box". Looking in detail, the module sends continuously two lines of information:



```
perf: {"preprocess":7,"inference":76,"postprocess":0}
boxes: [[245,292,449,392,89,0]]
```

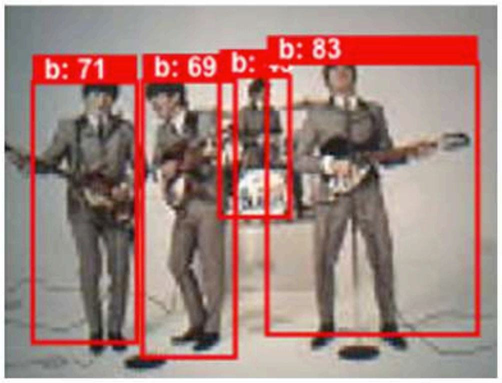
perf (Performance), displays latency in milliseconds.

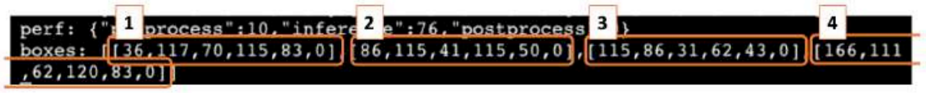
- Preprocess time (image capture and Crop): **7ms**;
- Inference time (model latency): **76ms (13 fps)**
- Postprocess time (display of the image and inclusion of data): less than 0ms.

boxes: Show the objects detected in the image. In this case, only one.

- The box has the x, y, w, and h coordinates of (**245**, **292**,**449**,**392**), and the object (person, label **0**) was captured with a value of .**89**.

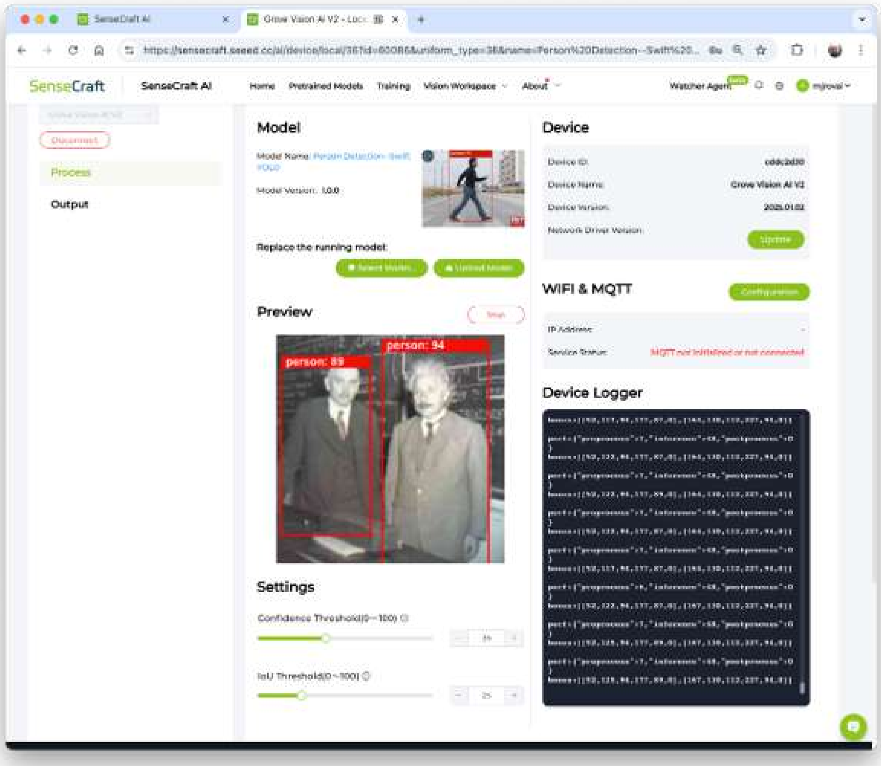
If we point the camera at an image with several people, we will get one box for each person (object):





```
perf: {" process":10,"infere e":76,"postprocess }
boxes: [[36,117,70,115,83,0],[86,115,41,115,50,0],[115,86,31,62,43,0],[166,111
,62,120,83,0]]
```

On the SenseCraft AI Studio, the inference latency (48ms) is lower than on the SenseCraft ToolKit (76ms), due to a distinct deployment implementation.



Model

Model Name: Person Detection--Swift YOLO

Model Version: 1.0.0

Replace the running model:

Preview

Settings

Confidence Threshold(0~100) 35

IoU Threshold(0~100) 25

Device

Device ID: cddc2d30

Device Name: Grove Vision AI V2

Device Version: 2025.01.02

Network Driver Version:

WIFI & MQTT

IP Address: -

Service Status: MQTT not initialized or not connected

Device Logger

```
boxes:[[52,117,94,177,87,0],[165,130,112,227,94,0]]

perf:{"preprocess":7,"inference":48,"postprocess":0
}
boxes:[[52,122,94,177,87,0],[165,130,112,227,94,0]]

perf:{"preprocess":7,"inference":48,"postprocess":0
}
boxes:[[52,122,94,177,89,0],[165,130,112,227,94,0]]

perf:{"preprocess":7,"inference":48,"postprocess":0
}
boxes:[[52,122,94,177,87,0],[165,130,112,227,94,0]]

perf:{"preprocess":7,"inference":48,"postprocess":0
}
boxes:[[52,117,94,177,87,0],[165,130,112,227,94,0]]

perf:{"preprocess":8,"inference":48,"postprocess":0
}
boxes:[[52,122,94,177,87,0],[167,130,112,227,94,0]]

perf:{"preprocess":7,"inference":58,"postprocess":0
}
boxes:[[52,125,94,177,89,0],[167,130,112,227,94,0]]

perf:{"preprocess":7,"inference":48,"postprocess":0
}
boxes:[[52,125,94,177,89,0],[167,130,112,227,94,0]]
```

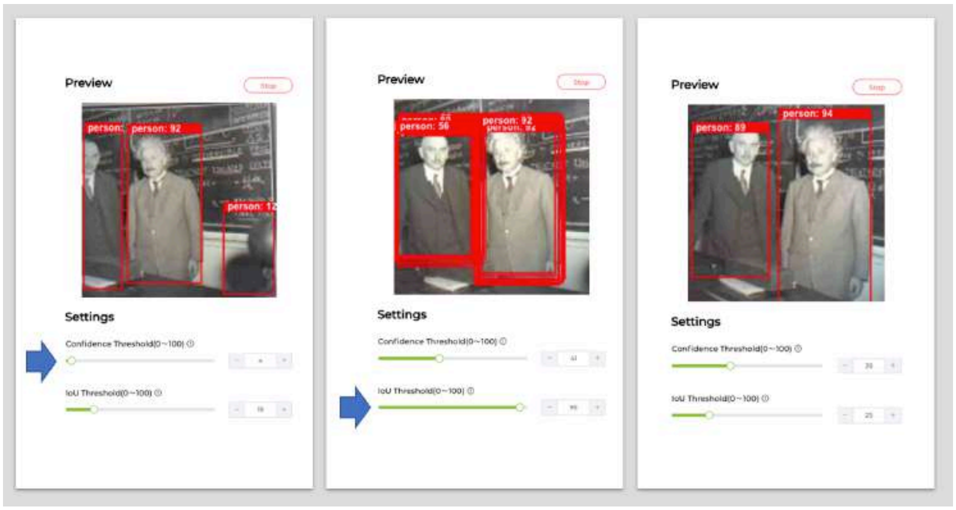
Power Consumption

The peak power consumption running this Swift-YOLO model was 410 milliwatts.

Preview Settings

We can see that in the Settings, two settings options can be adjusted to optimize the model's recognition accuracy.

- **Confidence:** Refers to the level of certainty or probability assigned to its predictions by a model. This value determines the minimum confidence level required for the model to consider a detection as valid. A higher confidence threshold will result in fewer detections but with higher certainty, while a lower threshold will allow more detections but may include some false positives.
- **IoU:** Used to assess the accuracy of predicted bounding boxes compared to truth bounding boxes. IoU is a metric that measures the overlap between the predicted bounding box and the ground truth bounding box. It is used to determine the accuracy of the object detection. The IoU threshold sets the minimum IoU value required for a detection to be considered a true positive. Adjusting this threshold can help in fine-tuning the model's precision and recall.



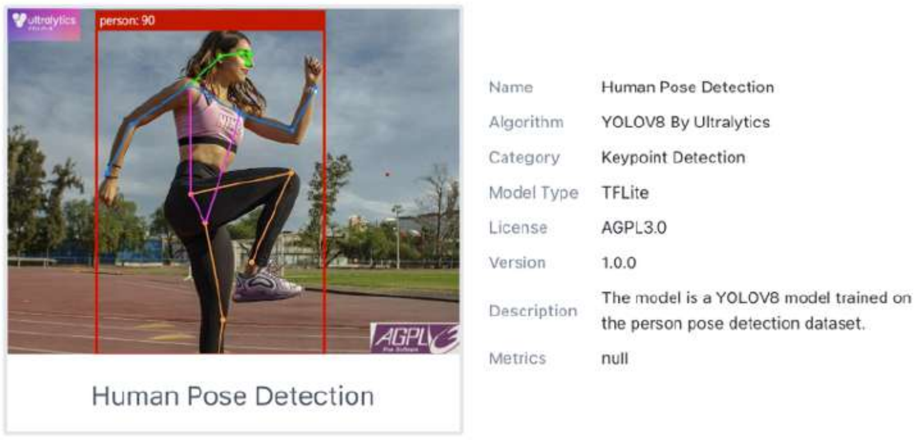
Experiment with different values for the Confidence Threshold and IoU Threshold to find the optimal balance between detecting persons accurately and minimizing false positives. The best settings may vary depending on our specific application and the characteristics of the images or video feed.

Pose/Keypoint Detection

Pose or keypoint detection is a sophisticated area within computer vision that focuses on identifying specific points of interest within an image or video frame, often related to human bodies, faces, or other objects of interest. This technology can detect and map out the various keypoints of a subject, such as the **joints on a human body** or the features of a face, enabling the analysis of postures, movements, and gestures. This has profound implications for various applications, including augmented reality, human-computer interaction, sports analytics, and healthcare monitoring, where understanding human motion and activity is crucial.

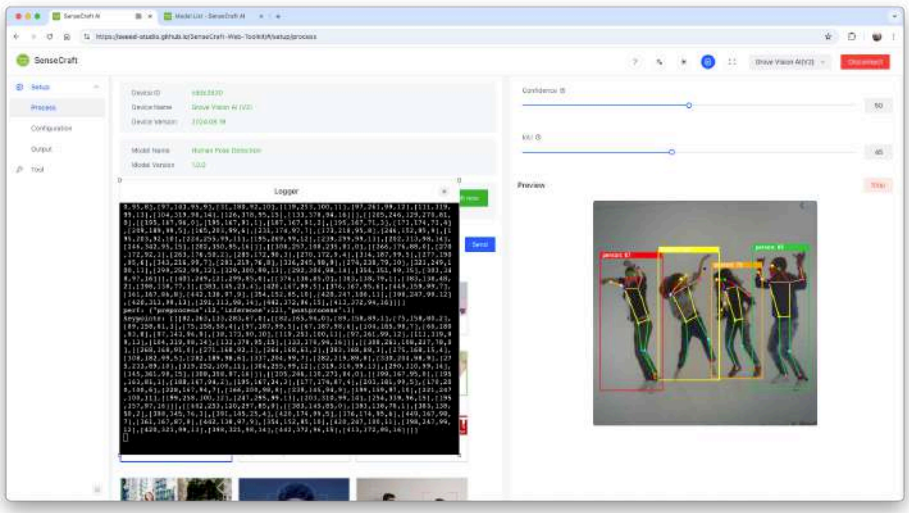
Unlike general object detection, which identifies and locates objects, pose detection drills down to a finer level of detail, capturing the nuanced positions and orientations of specific parts. Leading architectures in this field include OpenPose, AlphaPose, and PoseNet, each designed to tackle the challenges of pose estimation with varying degrees of complexity and precision. Through advancements in deep learning and neural networks, pose detection has become increasingly accurate and efficient, offering real-time insights into the intricate dynamics of subjects captured in visual data.

So, let's explore this popular CV application, *Pose/Keypoint Detection*.



Name	Human Pose Detection
Algorithm	YOLOV8 By Ultralytics
Category	Keypoint Detection
Model Type	TFLite
License	AGPL3.0
Version	1.0.0
Description	The model is a YOLOV8 model trained on the person pose detection dataset.
Metrics	null

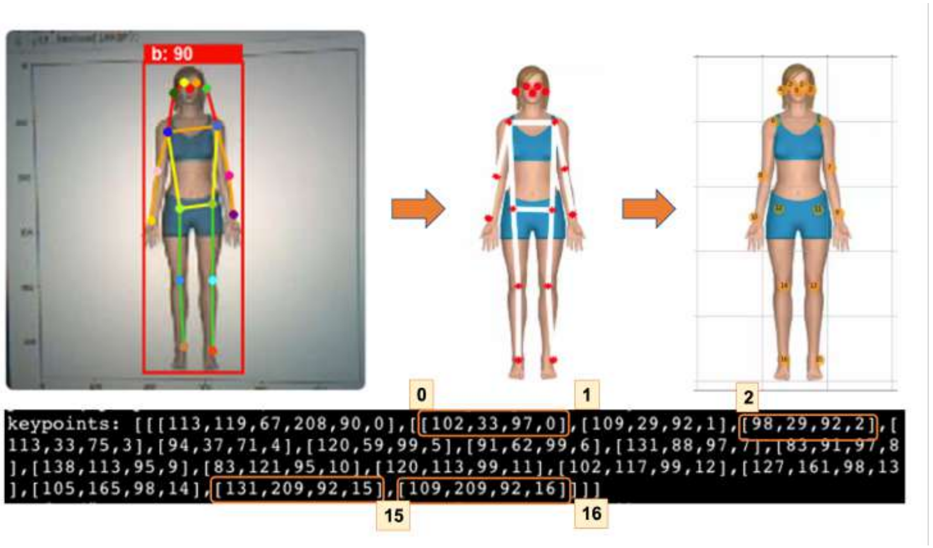
Stop the current model inference by pressing `[Stop]` in the Preview area. Select the model and press `[Send]`. Once the model is uploaded successfully, you can view the live feed from the Grove Vision AI (V2) camera in the Preview area on the right, along with the inference details displayed in the Serial Monitor (accessible by clicking the `[Device Log]` button at the top).



The YOLOV8 Pose model was trained using the COCO-Pose Dataset, which contains 200K images labeled with **17** keypoints for pose estimation tasks.

Let's look at a single screenshot of the inference (to simplify, let's analyse an image with a single person in it). We can note that we have two lines, one with the inference **performance** in milliseconds (121 ms) and a second line with the **keypoints** as below:

- 1 box of info, the same as we got with the object detection example (box coordinates (113, 119, 67, 208), inference result (90), label (0).
- 17 groups of 4 numbers represent the 17 "joints" of the body, where '0' is the nose, '1' and '2' are the eyes, '15' and' 16' are the feet, and so on.



```
keypoints: [[[113,119,67,208,90,0],[[102,33,97,0],[109,29,92,1],[98,29,92,2],[
113,33,75,3],[94,37,71,4],[120,59,99,5],[91,62,99,6],[131,88,97,7],[83,91,97,8
],[138,113,95,9],[83,121,95,10],[120,113,99,11],[102,117,99,12],[127,161,98,13
],[105,165,98,14],[131,209,92,15],[109,209,92,16]]]]
```

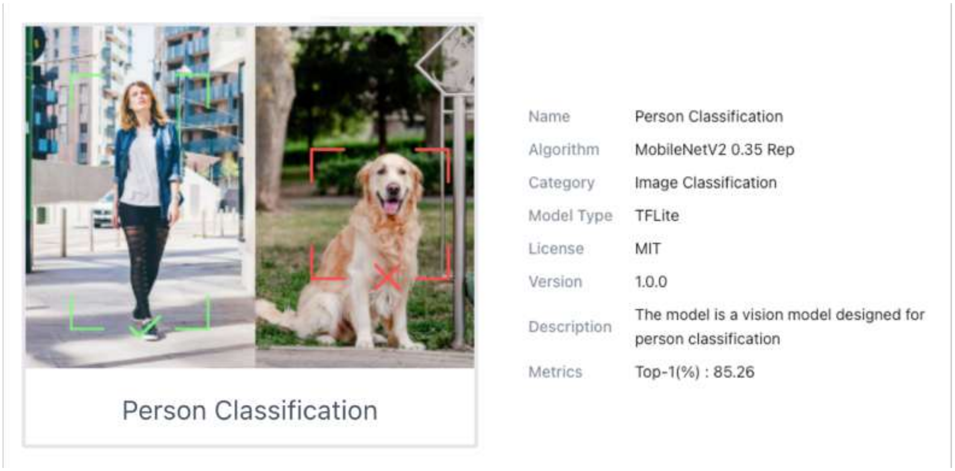
To understand a pose estimation project more deeply, please refer to the tutorial: Exploring AI at the Edge! - Pose Estimation.

Image Classification

Image classification is a foundational task within computer vision aimed at categorizing **entire images** into one of several predefined classes. This process involves analyzing the visual content of an image and assigning it a label from a fixed set of categories based on the predominant object or scene it contains.

Image classification is crucial in various applications, ranging from organizing and searching through large databases of images in digital libraries and social media platforms to enabling autonomous systems to comprehend their surroundings. Common architectures that have significantly advanced the field of image classification include Convolutional Neural Networks (CNNs), such as AlexNet, VGGNet, and ResNet. These models have demonstrated remarkable accuracy on challenging datasets, such as **ImageNet,** by learning hierarchical representations of visual data.

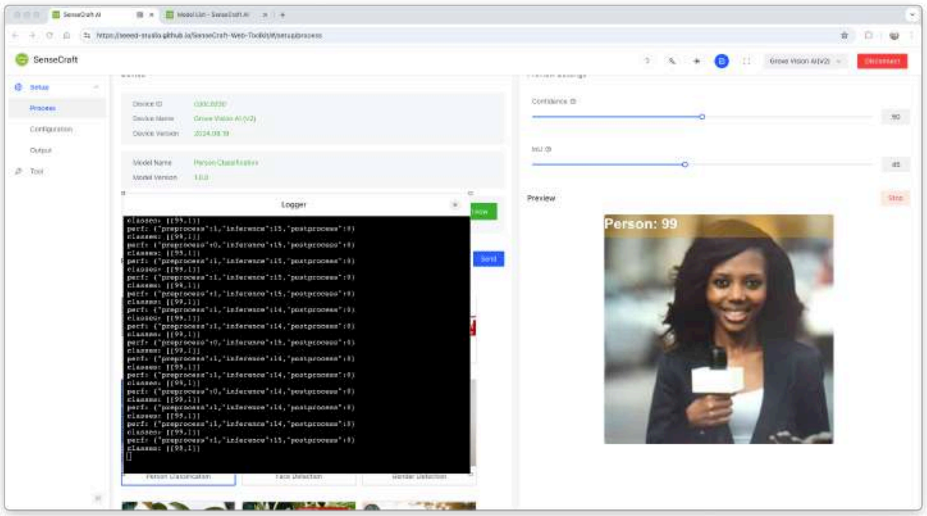
As the cornerstone of many computer vision systems, image classification drives innovation, laying the groundwork for more complex tasks like object detection and image segmentation, and facilitating a deeper understanding of visual data across various industries. So, let's also explore this computer vision application.



Name	Person Classification
Algorithm	MobileNetV2 0.35 Rep
Category	Image Classification
Model Type	TFLite
License	MIT
Version	1.0.0
Description	The model is a vision model designed for person classification
Metrics	Top-1(%) : 85.26

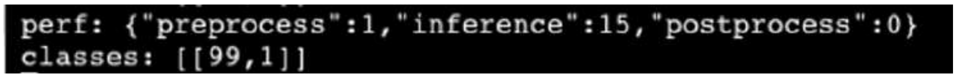
This example is available on the SenseCraft ToolKit, but not in the SenseCraft AI Studio. In the last one, it is possible to find other examples of Image Classification.

After the model is uploaded successfully, we can view the live feed from the Grove Vision AI (V2) camera in the Preview area on the right, along with the inference details displayed in the Serial Monitor (by clicking the `[Device Log]` button at the top).



```
classes: [[99,1]]
perf: {"preprocess":1,"inference":15,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":0,"inference":15,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":1,"inference":15,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":1,"inference":15,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":1,"inference":15,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":1,"inference":14,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":1,"inference":14,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":0,"inference":15,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":1,"inference":14,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":1,"inference":14,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":0,"inference":14,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":1,"inference":14,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":1,"inference":14,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":1,"inference":15,"postprocess":0}
classes: [[98,1]]
```

As a result, we will receive a score and the class as output.



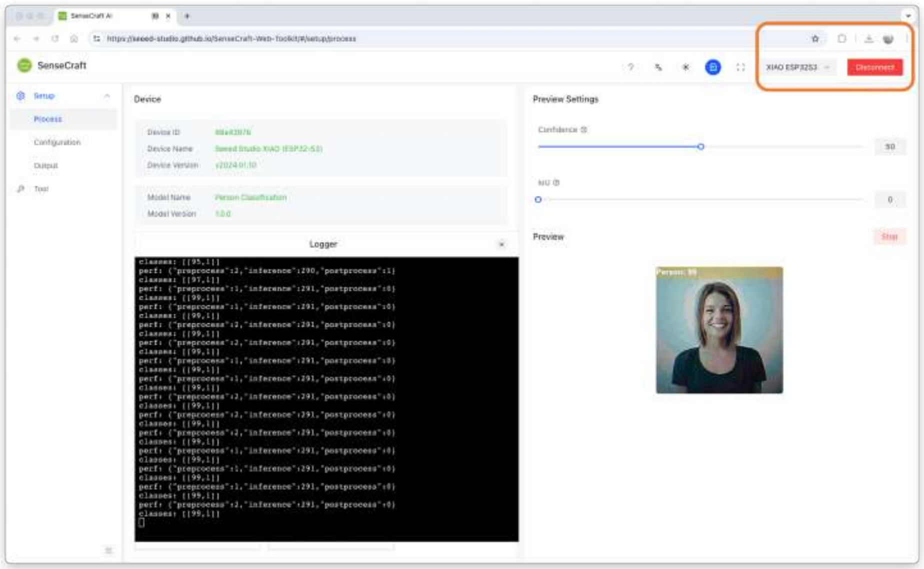
```
perf: {"preprocess":1,"inference":15,"postprocess":0}
classes: [[99,1]]
```

For example, **[99, 1]** means class: 1 (Person) with a score of 0.99. Once this model is a binary classification, class 0 will be "No Person" (or Background). The Inference latency is **15ms** or around 70fps.

Power Consumption

To run the Mobilenet V2 0.35, the Grove Vision AI V2 had a peak current of 80mA at 5.24V, resulting in a **power consumption of 420mW**.

Runing the same model on XIAO ESP32S3 Sense, the **power consumption was 523mW** with a latency of 291ms.



Device

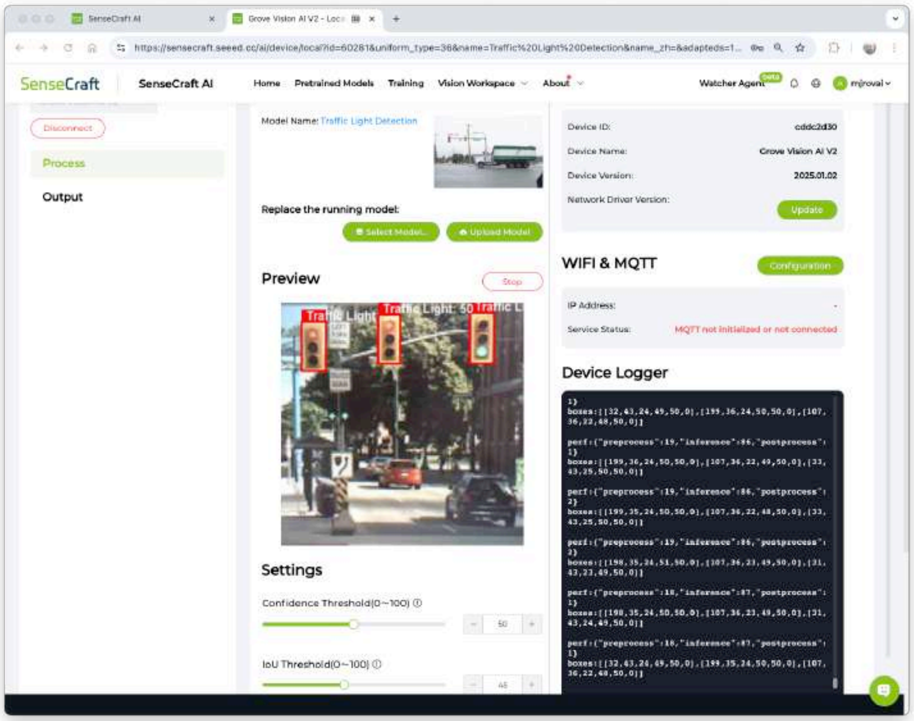
Device ID	88e43976
Device Name	Seeed Studio XIAO (ESP32-S3)
Device Version	v2024.01.10
Model Name	Person Classification
Model Version	1.0.0

Logger

```
classes: [[95,1]]
perf: {"preprocess":2,"inference":290,"postprocess":1}
classes: [[97,1]]
perf: {"preprocess":1,"inference":291,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":1,"inference":291,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":2,"inference":291,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":2,"inference":291,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":1,"inference":291,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":1,"inference":291,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":2,"inference":291,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":2,"inference":291,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":2,"inference":291,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":1,"inference":291,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":1,"inference":291,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":1,"inference":291,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":2,"inference":291,"postprocess":0}
classes: [[99,1]]
```

Exploring Other Models on SenseCraft AI Studio

Several public AI models can also be downloaded from the SenseCraft AI WebPage. For example, you can run a Swift-YOLO model, detecting traffic lights as shown here:



Model Name: Traffic Light Detection

Replace the running model:

Device ID: cddc2d30
Device Name: Grove Vision AI V2
Device Version: 2025.01.02
Network Driver Version:

WIFI & MQTT

IP Address: -
Service Status: MQTT not initialized or not connected

Confidence Threshold(0~100) 50

IoU Threshold(0~100) 45

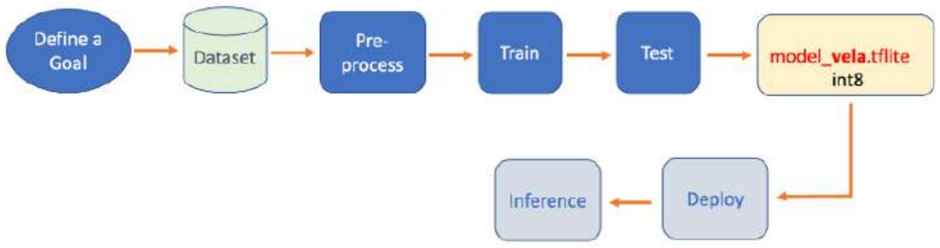
Device Logger

```
1}
boxes:[[32,43,24,49,50,0],[199,36,24,50,50,0],[107,
36,22,48,50,0]]

perf:{"preprocess":19,"inference":86,"postprocess":
1}
boxes:[[199,36,24,50,50,0],[107,36,22,49,50,0],[33,
43,25,50,50,0]]

perf:{"preprocess":19,"inference":86,"postprocess":
2}
boxes:[[199,35,24,50,50,0],[107,36,22,48,50,0],[33,
43,25,50,50,0]]

perf:{"preprocess":19,"inference":86,"postprocess":
2}
boxes:[[198,35,24,51,50,0],[107,36,23,49,50,0],[31,
43,23,49,50,0]]

perf:{"preprocess":18,"inference":87,"postprocess":
1}
boxes:[[198,35,24,50,50,0],[107,36,23,49,50,0],[31,
43,24,49,50,0]]

perf:{"preprocess":18,"inference":87,"postprocess":
1}
boxes:[[32,43,24,49,50,0],[199,35,24,50,50,0],[107,
36,22,48,50,0]]
```

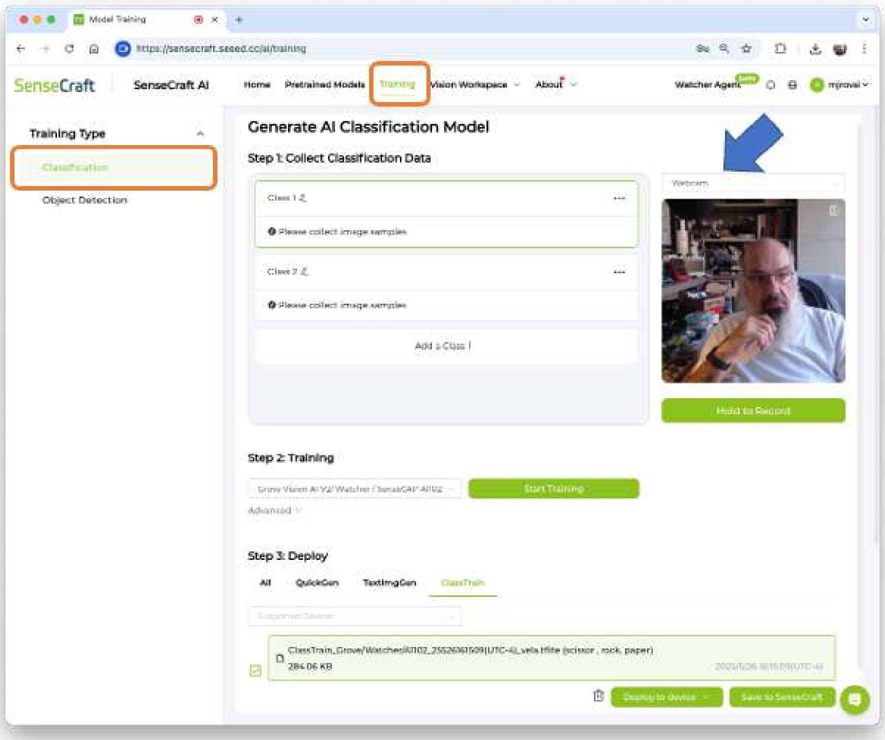
The latency of this model is approximately 86 ms, with an average power consumption of 420 mW.

An Image Classification Project

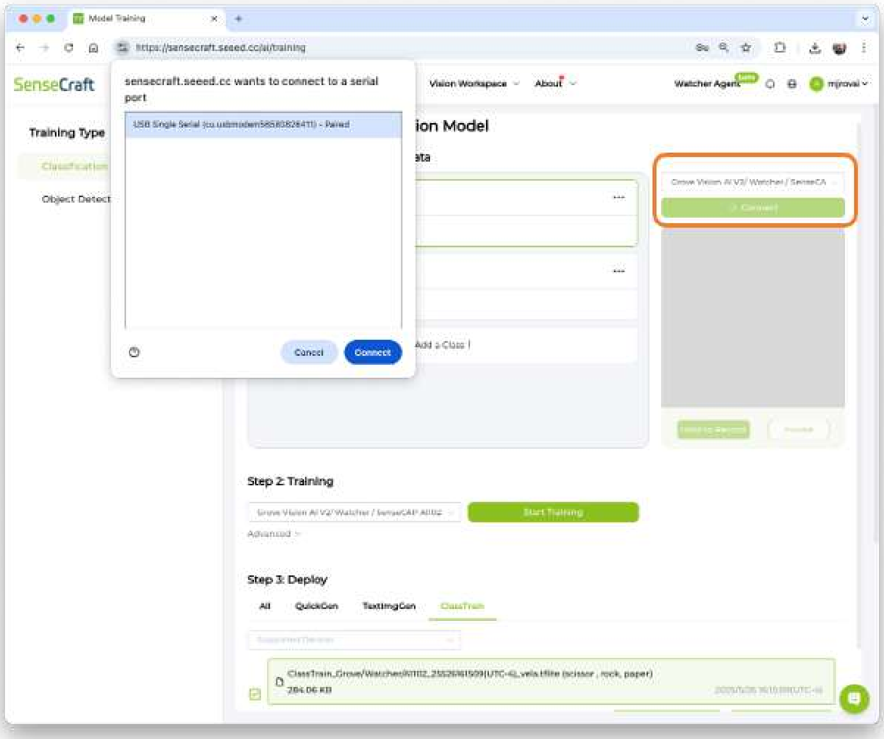
Let's create a complete Image Classification project, using the SenseCraft AI Studio.



On SenseCraft AI Studio: Let's open the tab Training:



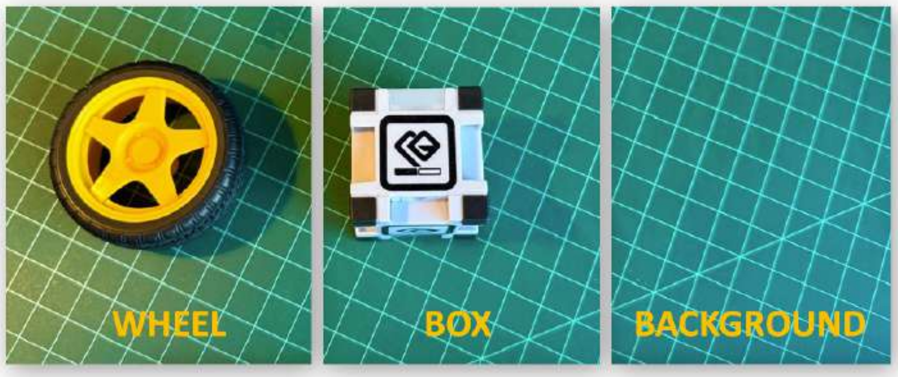
The default is to train a `Classification` model with a WebCam if it is available. Let's select the Grove Vision AI V2 instead. Pressing the green button`[Connect]`, a Pop-Up window will appear. Select the corresponding Port and press the blue button `[Connect]`.



The image streamed from the Grove Vision AI V2 will be displayed.

The Goal

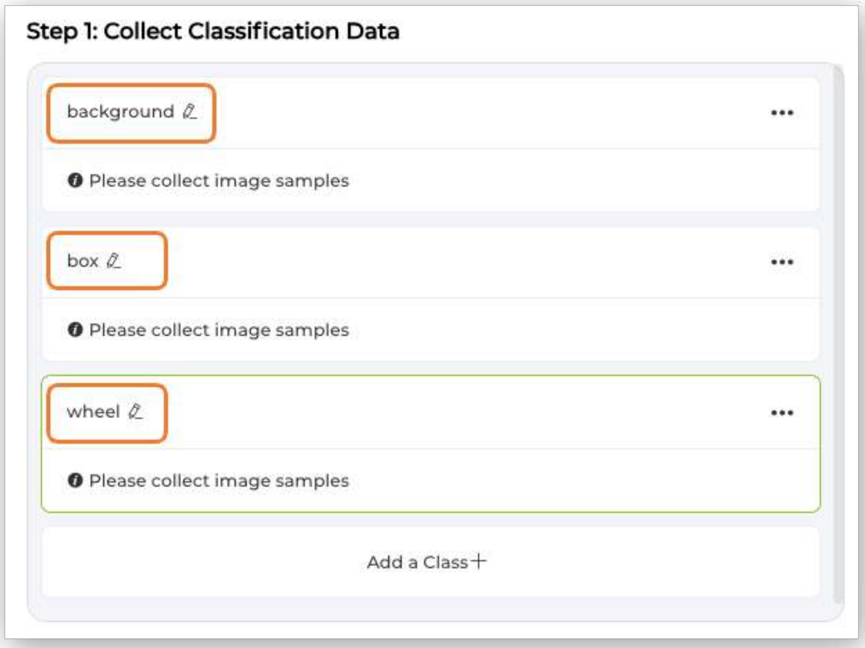
The first step is always to define a goal. Let's classify, for example, two simple objects—for instance, a toy `box` and a toy `wheel`. We should also include a 3rd class of images, `background`, where no object is in the scene.



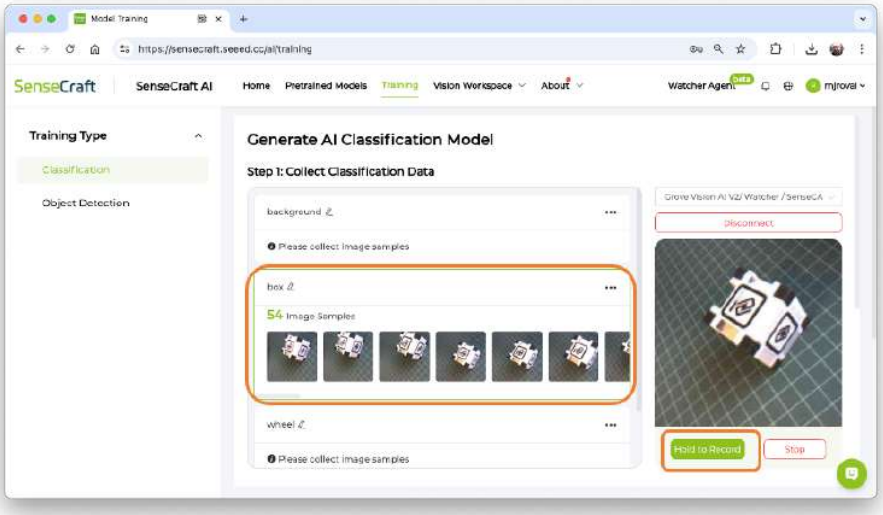
Data Collection

Let's create the classes, following, for example, an alphabetical order:

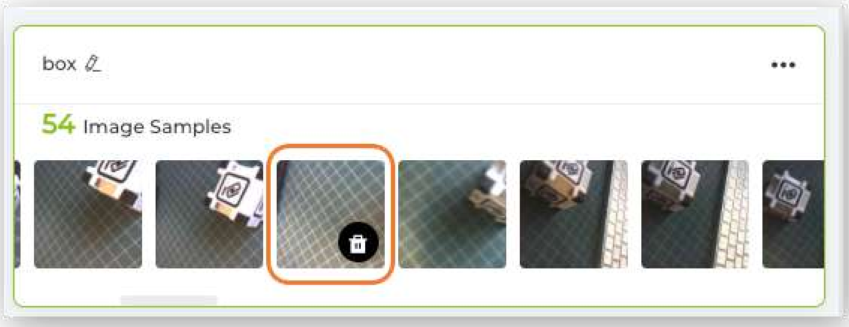
- Class1: background
- Class 2: box
- Class 3: wheel


Step 1: Collect Classification Data
background
Please collect image samples
box
Please collect image samples
wheel
Please collect image samples
Add a Class+

Select one of the classes and keep pressing the green button under the preview area. The collected images will appear on the Image Samples Screen.



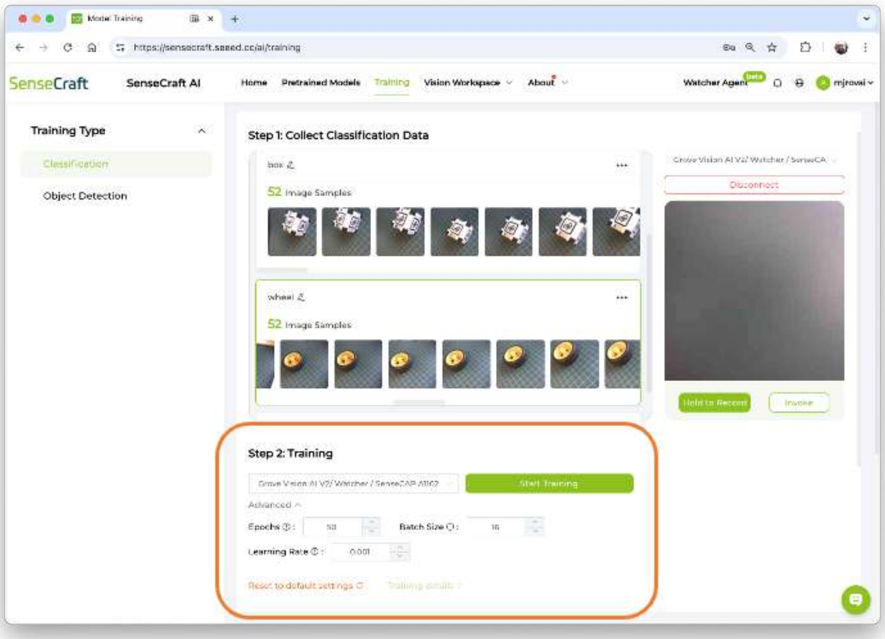
After collecting the images, review them and delete any incorrect ones.



Collect around 50 images from each class and go to Training Step:

Training

Confirm if the correct device is selected (`Grove Vision AI V2`) and press `[Start Training]`

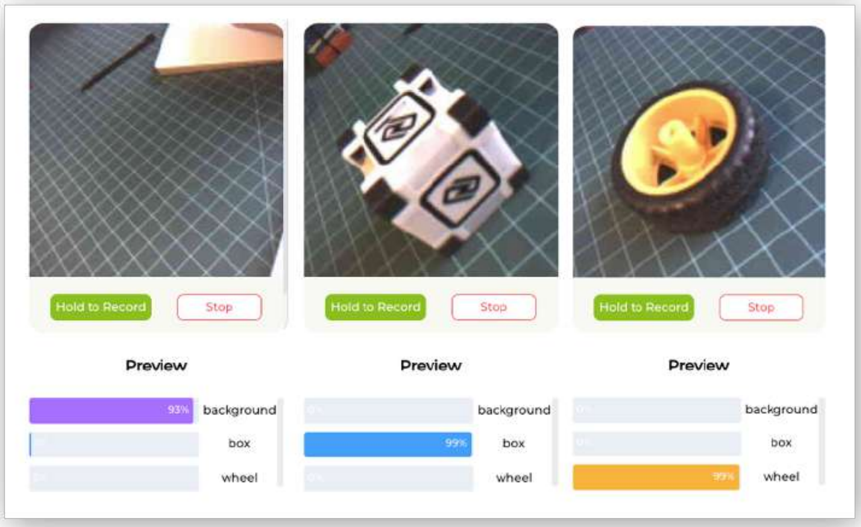


Epochs: 50
Batch Size: 16
Learning Rate: 0.001

Test

After training, the inference result can be previewed.

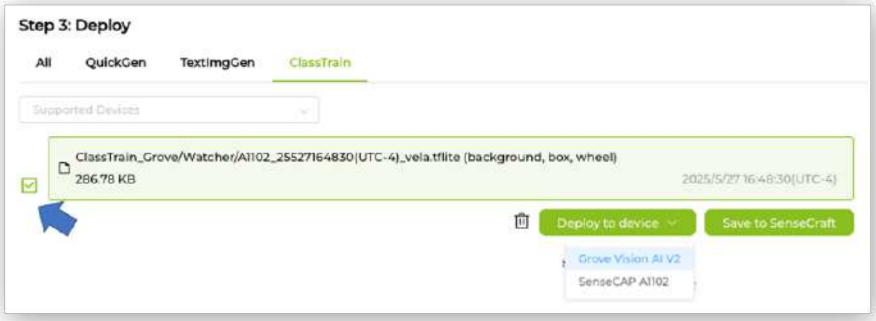
> Note that the model is not running on the device. We are, in fact, only capturing the images with the device and performing a live preview using the training model, which is running in the Studio.



Now is time to really deploy the model in the device:

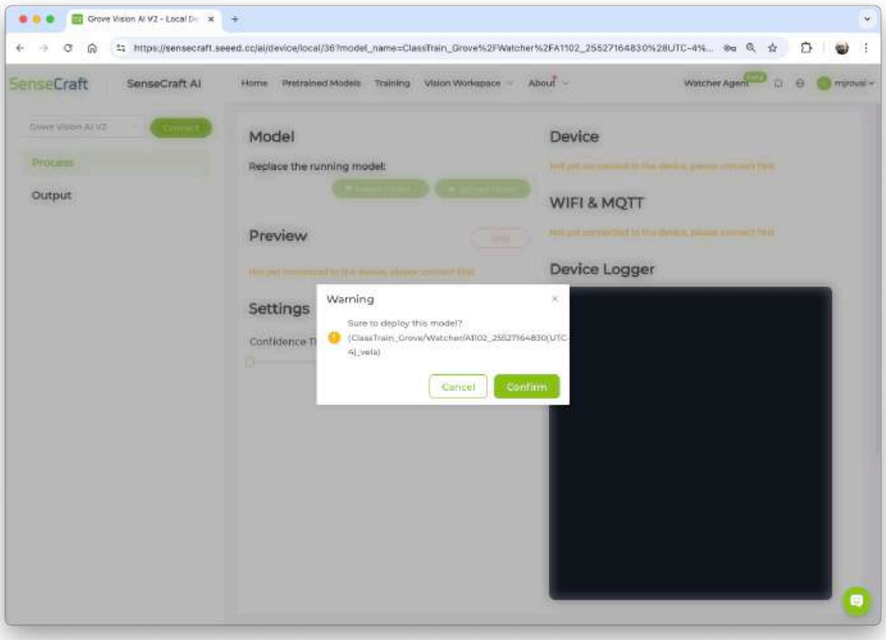
Deployment

Select the trained model on `[Deploy to device]`, select the Grove Vision AI V2:



ClassTrain_Grove/Watcher/A1102_25527164830(UTC-4)_vela.tflite (background, box, wheel)
286.78 KB
2025/5/27 16:48:30(UTC-4)

The Studio will redirect us to the `Vision Workplace` tab. Confirm the deployment, select the appropriate Port, and connect it:

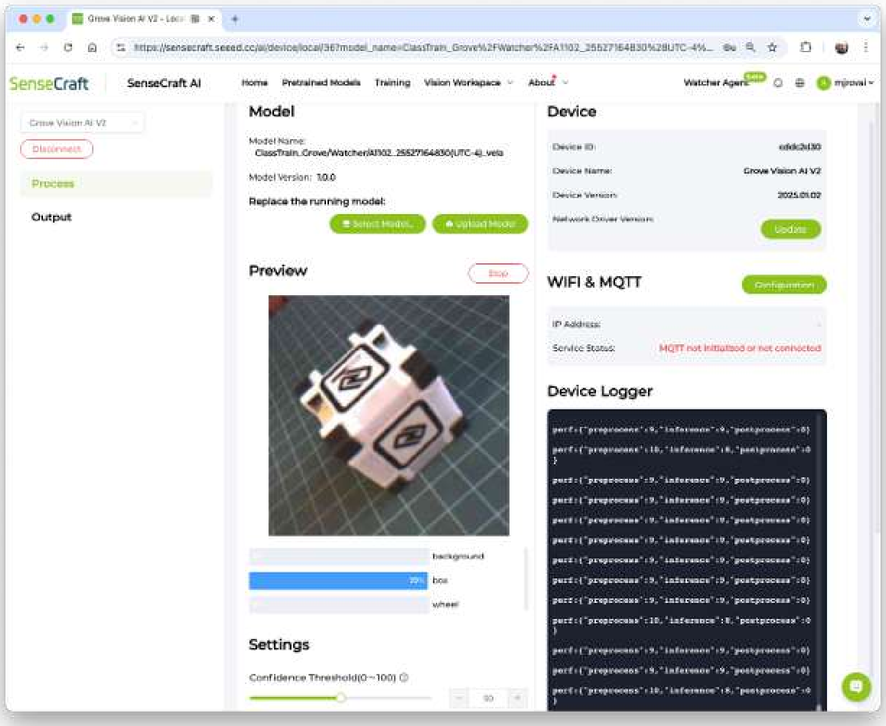


Warning

Sure to deploy this model?
(ClassTrain_Grove/Watcher/AI102_25527164830(UTC-4)_vela)

The model will be flashed into the device. After an automatic reset, the model will start runing on the device. On the Device Logger, we can see that the inference has a **latency of approximately 8 ms**, corresponding to a **frame rate of 125 frames per second (FPS)**.

Also, note that it is possible to adjust the model's confidence.



Model

Model Name: ClassTrain_Grove/Watcher/AI102_25527164830(UTC-4)_vela

Model Version: 1.0.0

Replace the running model:

Preview

Settings

Confidence Threshold(0~100)

Device

Device ID: cddc2d30

Device Name: Grove Vision AI V2

Device Version: 2025.01.02

Network Driver Version:

WIFI & MQTT

IP Address:

Service Status: MQTT not initialized or not connected

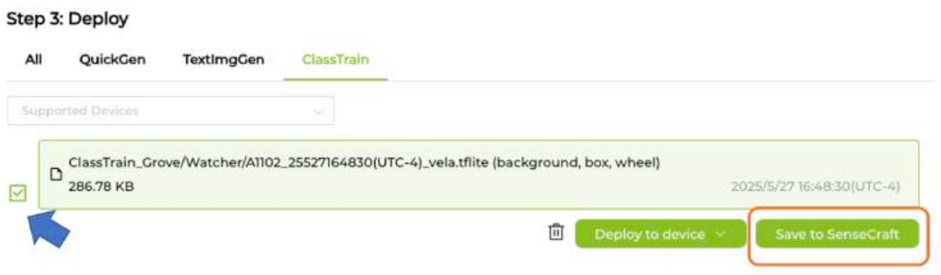
Device Logger

```
perf:{"preprocess":9,"inference":9,"postprocess":0}
perf:{"preprocess":10,"inference":8,"postprocess":0
}
perf:{"preprocess":9,"inference":9,"postprocess":0}
perf:{"preprocess":9,"inference":9,"postprocess":0}
perf:{"preprocess":9,"inference":9,"postprocess":0}
perf:{"preprocess":9,"inference":9,"postprocess":0}
perf:{"preprocess":9,"inference":9,"postprocess":0}
perf:{"preprocess":9,"inference":9,"postprocess":0}
perf:{"preprocess":9,"inference":9,"postprocess":0}
perf:{"preprocess":10,"inference":8,"postprocess":0
}
perf:{"preprocess":9,"inference":9,"postprocess":0}
perf:{"preprocess":9,"inference":9,"postprocess":0}
perf:{"preprocess":10,"inference":8,"postprocess":0
}
```

To run the Image Classification Model, the Grove Vision AI V2 had a peak current of 80mA at 5.24V, resulting in a **power consumption of 420mW**.

Saving the Model

It is possible to save the model in the SenseCraft AI Studio. The Studio will keep all our models, which can be deployed later. For that, return to the `Training` tab and select the button `[Save to SenseCraft]`:



ClassTrain_Grove/Watcher/AI102_25527164830(UTC-4)_vela.tflite (background, box, wheel)
286.78 KB
2025/5/27 16:48:30(UTC-4)

Conclusion

In this lab, we explored several computer vision (CV) applications using the Seeed Studio Grove Vision AI Module V2, demonstrating its exceptional capabilities as a powerful yet compact device specifically designed for embedded machine learning applications.

Performance Excellence: The Grove Vision AI V2 demonstrated remarkable performance across multiple computer vision tasks. With its **Himax WiseEye2 chip** featuring a **dual-core Arm Cortex-M55 and integrated ARM Ethos-U55 neural network unit**, the device delivered:

- **Image Classification**: **15 ms** inference time (67 FPS)
- **Object Detection (Person)**: **48 ms to 76 ms** inference time (21 FPS to 13 FPS)
- **Pose Detection**: **121 ms** real-time keypoint detection with 17-joint tracking (8 FPS)

Power Efficiency Leadership: One of the most compelling advantages of the Grove Vision AI V2 is its superior power efficiency. Comparative testing revealed significant improvements over traditional embedded platforms:

- **Grove Vision AI V2**: 80 mA (**410 mW**) peak consumption (60+ FPS)
- **XIAO ESP32S3**: Performing similar CV tasks (Image Classification) **523 mW** (3+ FPS)

Practical Implementation: The device's versatility was demonstrated through a comprehensive end-to-end project, encompassing dataset creation, model training, deployment, and offline inference.

Developer-Friendly Ecosystem: The SenseCraft AI Studio, with its no-code deployment and integration capabilities for custom applications, makes the Grove Vision AI V2 accessible to both beginners and advanced developers. The extensive library of pre-trained models and support for custom model deployment provide flexibility for diverse applications.

The Grove Vision AI V2 represents a significant advancement in edge AI hardware, offering professional-grade computer vision capabilities in a compact, energy-efficient package that democratizes AI deployment for embedded applications across industrial, IoT, and educational domains.

Key Takeaways

This Lab demonstrates that sophisticated computer vision applications are not limited to cloud-based solutions or power-hungry hardware, as the Raspberry Pi or Jetson Nanos – they can now be deployed effectively at the edge with remarkable efficiency and performance.

Optionally, we can have the XIAO Vision AI Camera. This innovative vision solution seamlessly combines the Grove Vision AI V2 module, XIAO ESP32-C3 controller, and an OV5647 camera, all housed in a custom 3D-printed enclosure:

Resourses

SenseCraft AI Studio Instructions.

SenseCraft-Web-Toolkit website.

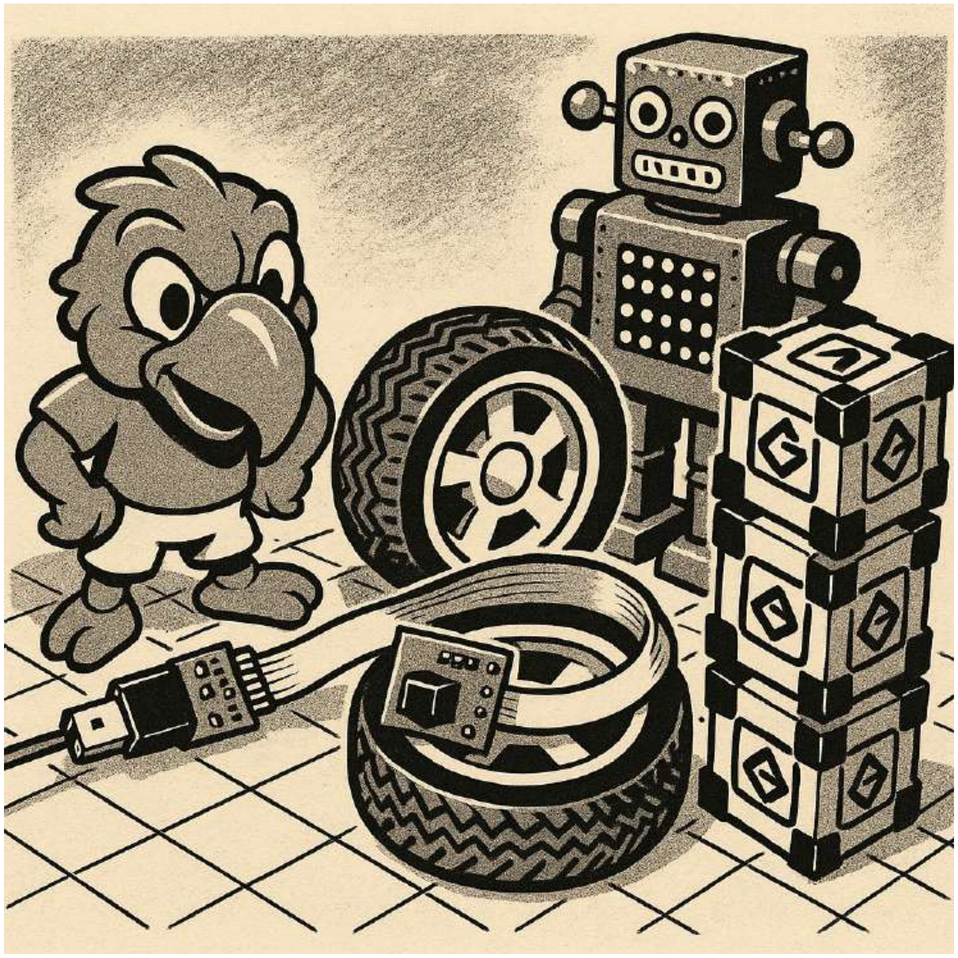
SenseCraft AI Studio

Himax AI Web Toolkit

Himax examples

Image Classification

Using Seeed Studio Grove Vision AI Module V2 (Himax WiseEye2)

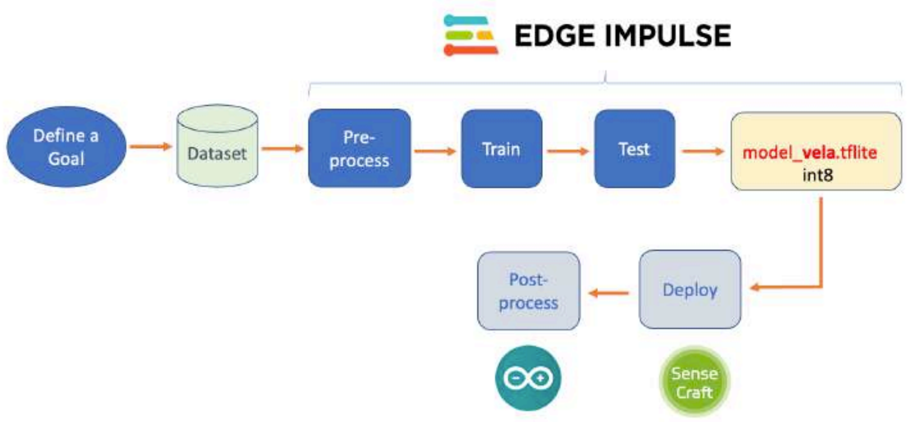


In this Lab, we will explore Image Classification using the Seeed Studio *Grove Vision AI Module V2*, a powerful yet compact device specifically designed for embedded machine learning applications. Based on the **Himax WiseEye2** chip, this module is designed to enable AI capabilities on edge devices, making it an ideal tool for Edge Machine Learning (ML) applications.

Introduction

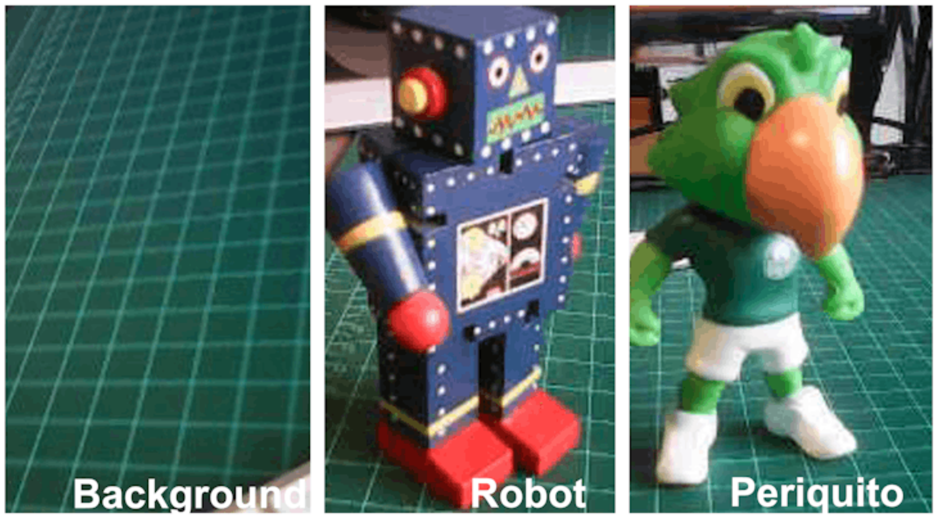
So far, we have explored several computer vision models previously uploaded by Seeed Studio or used the SenseCraft AI Studio for Image Classification, without choosing a specific model. Let's now develop our Image Classification project from scratch, where we will select our data and model.

Below, we can see the project's main steps and where we will work with them:



Project Goal

The first step in any machine learning (ML) project is defining the goal. In this case, the goal is to detect and classify two specific objects present in a single image. For this project, we will use two small toys: a robot and a small Brazilian parrot (named *Periquito*). Also, we will collect images of a background where those two objects are absent.

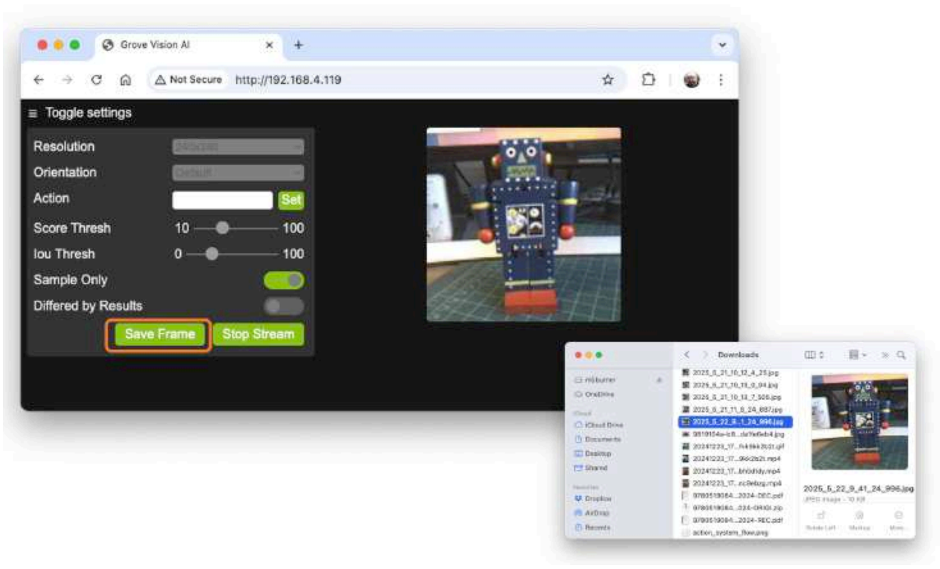


Data Collection

With the Machine Learning project goal defined, dataset collection is the next and most crucial step. Suppose your project utilizes images that are publicly available on datasets, for example, to be used on a **Person Detection** project. In that case, you can download the Wake Vision dataset for use in the project.

But, in our case, we define a project where the images do not exist publicly, so we need to generate them. We can use a phone, computer camera, or other devices to capture the photos, offline or connected to the Edge Impulse Studio.

If you want to use the Grove Vision AI V2 to capture your dataset, you can use the SenseCraft AI Studio as we did in the previous Lab, or the `camera_web_server` sketch as we will describe later in the **Postprocessing / Getting the Video Stream** section of this Lab.

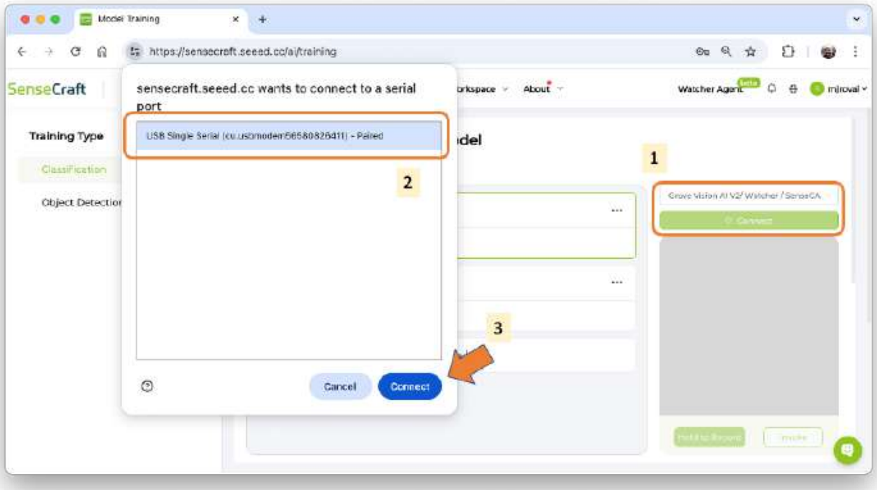


In this Lab, we will use the SenseCraft AI Studio to collect the dataset.

Collecting Data with the SenseCraft AI Studio

On SenseCraft AI Studio: Let's open the tab Training.

The default is to train a `Classification` model with a WebCam if it is available. Let's select the `Grove Vision AI V2 instead`. Pressing the green button `[Connect]` **(1),** a Pop-Up window will appear. Select the corresponding Port **(2)** and press the blue button `[Connect]` **(3)**.

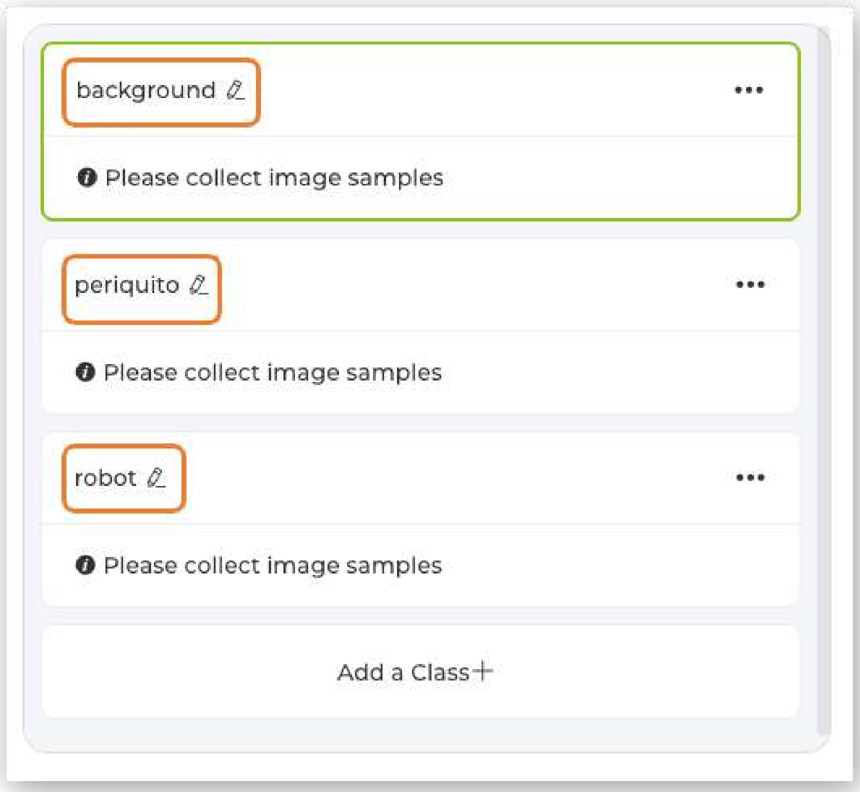

sensecraft.seeed.cc wants to connect to a serial port
USB Single Serial (cu.usbmodem56580826411) - Paired
Cancel
Connect
1
2
3
Grove Vision AI V2/ Watcher / SenseCA
Connect
Training Type
Classification
Object Detection
SenseCraft

The image streamed from the Grove Vision AI V2 will be displayed.

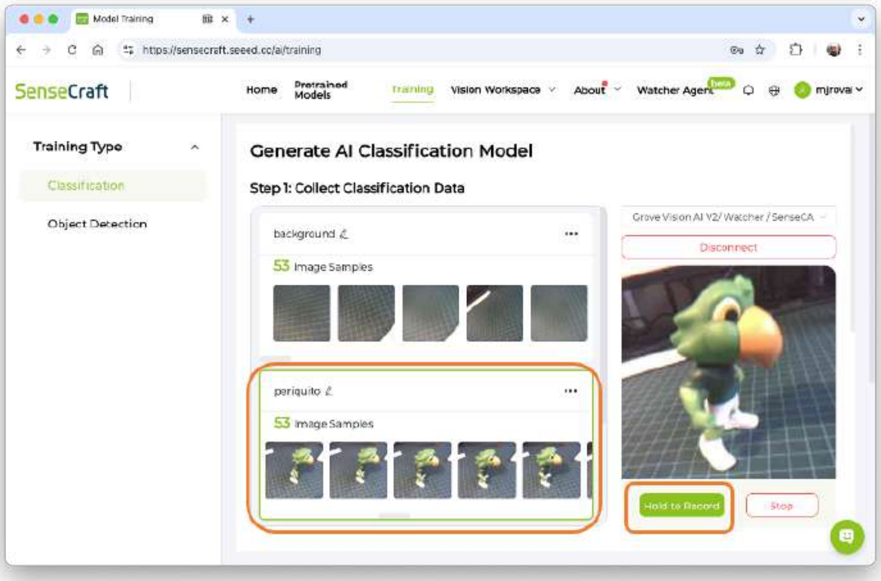
Image Collection

Let's create the classes, following, for example, an alphabetical order:

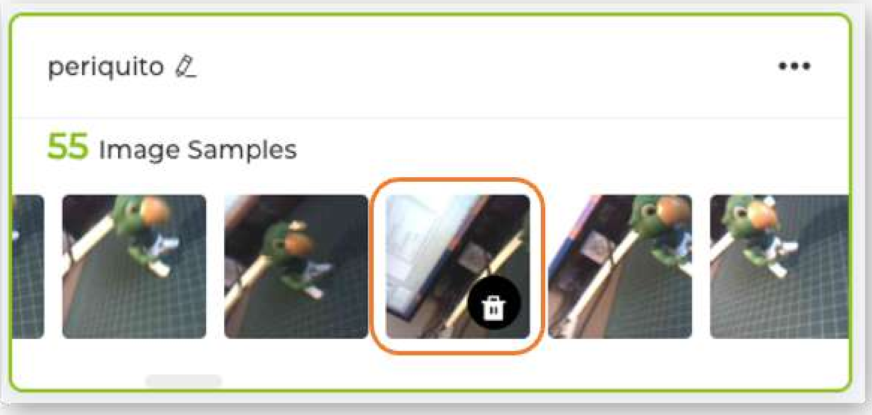
- Class1: background
- Class 2: periquito
- Class 3: robot


background
•••
Please collect image samples
periquito
•••
Please collect image samples
robot
•••
Please collect image samples
Add a Class+

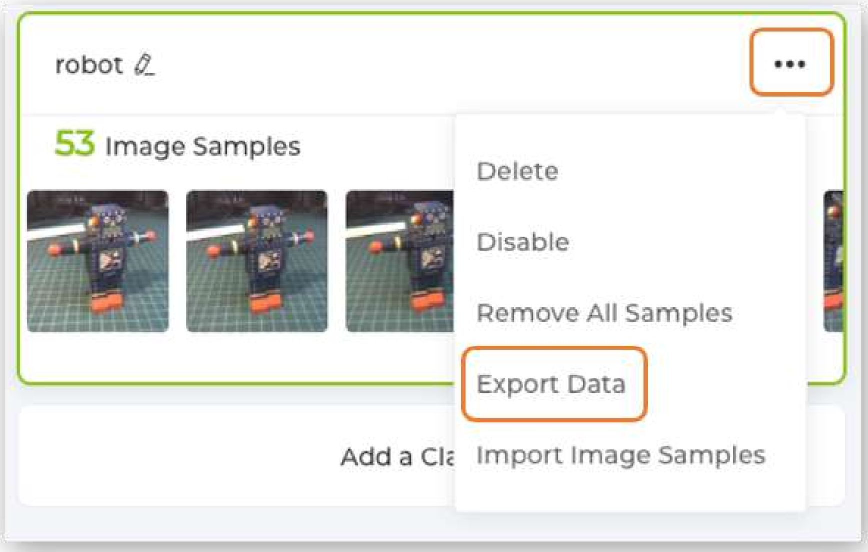
Select one of the classes (note that a green line will be around the window) and keep pressing the green button under the preview area. The collected images will appear on the Image Samples Screen.



After collecting the images, review them and, if necessary, delete any incorrect ones.



Collect around 50 images from each class. After you collect the three classes, open the menu on each of them and select `Export Data`.

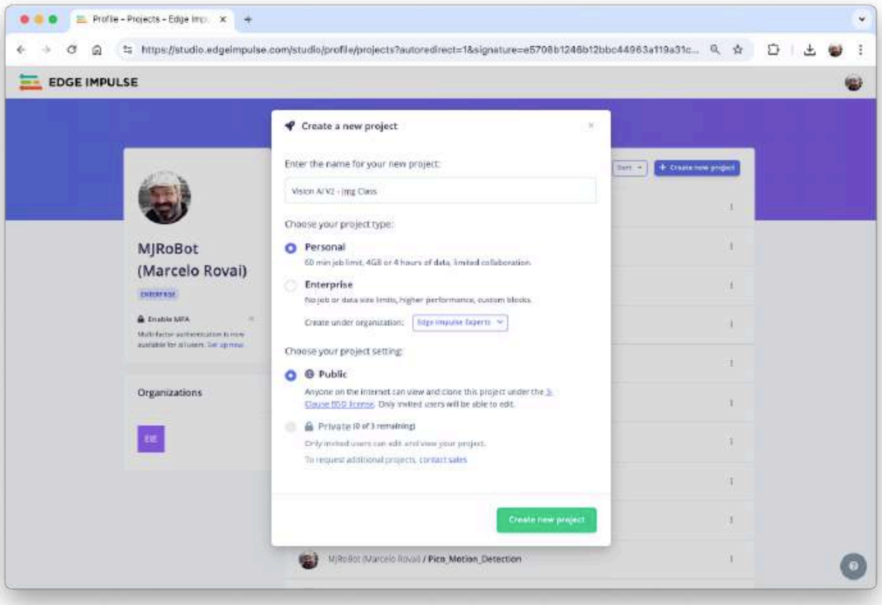


In the Download area of the Computer, we will get three zip files, each one with its corresponding class name. Each Zip file contains a folder with the images.

Uploading the dataset to the Edge Impulse Studio

We will use the Edge Impulse Studio to train our model. Edge Impulseis a leading development platform for machine learning on edge devices.

- Enter your account credentials (or create a free account) at Edge Impulse.
- Next, create a new project:



Enter the name for your new project:
Vision AI V2 - Img Class

Choose your project type:
[✓ Personal] 60 min job limit, 4GB or 4 hours of data, limited collaboration.
Enterprise — No job or data size limits, higher performance, custom blocks. Create under organization: Edge Impulse Experts

Choose your project setting:
[✓ Public] Anyone on the internet can view and clone this project under the 3-Clause BSD license. Only invited users will be able to edit.
Private (0 of 3 remaining) — Only invited users can edit and view your project. To request additional projects, contact sales.

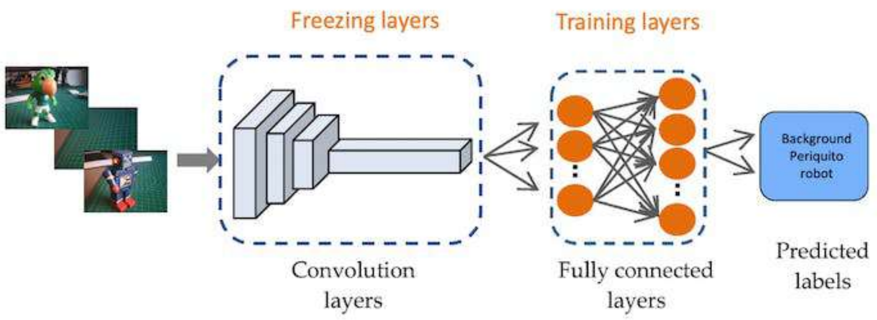
The dataset comprises approximately 50 images per label, with 40 for training and 10 for testing.

Impulse Design and Pre-Processing

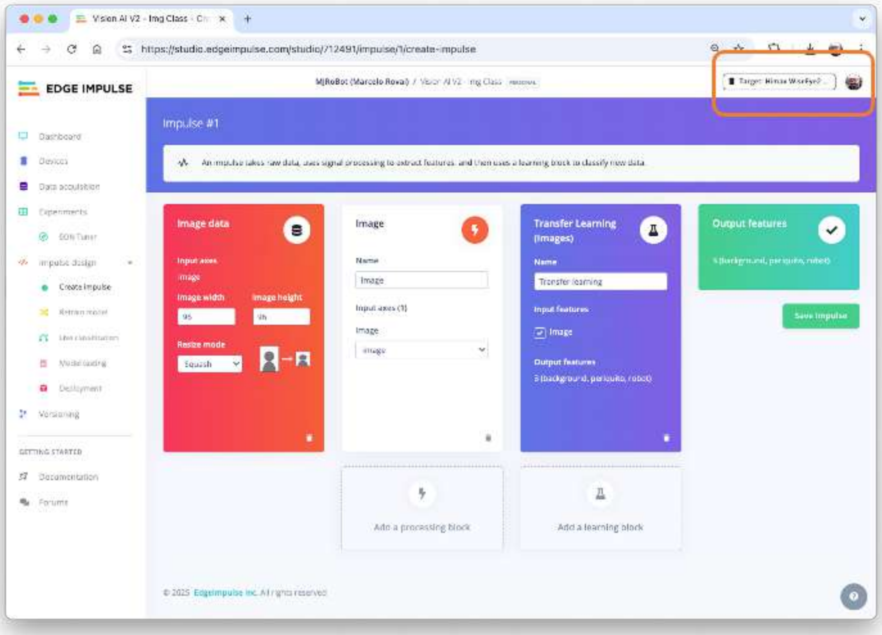
Impulse Design

An impulse takes raw data (in this case, images), extracts features (resizes pictures), and then uses a learning block to classify new data.

Classifying images is the most common application of deep learning, but a substantial amount of data is required to accomplish this task. We have around 50 images for each category. Is this number enough? Not at all! We will need thousands of images to "teach" or "model" each class, allowing us to differentiate them. However, we can resolve this issue by retraining a previously trained model using thousands of images. We refer to this technique as "Transfer Learning" (TL). With TL, we can fine-tune a pre-trained image classification model on our data, achieving good performance even with relatively small image datasets, as in our case.



So, starting from the raw images, we will resize them (96x96) pixels and feed them to our Transfer Learning block:



Impulse #1

An impulse takes raw data, uses signal processing to extract features, and then uses a learning block to classify new data.

Image data
Input axes
image
Image width 96
Image height 96
Resize mode Squash

Image
Name Image
Input axes (1)
Image image

Transfer Learning (Images)
Name Transfer learning
Input features
✓ Image
Output features
3 (background, periquito, robot)

Output features
3 (background, periquito, robot)

> For comparison, we will keep the image size as 96 x 96. However, keep in mind that with the Grove Vision AI Module V2 and its internal SRAM of 2.4 MB, larger images can be utilized (for example, 160 x 160).

Also select the `Target` device (`Himax WiseEye2 (M55 400 MHz + U55)`) on the up-right corner.

Pre-processing (Feature generation)

Besides resizing the images, we can convert them to grayscale or retain their original RGB color depth. Let's select `[RGB]` in the `Image` section. Doing that, each data sample will have a dimension of 27,648 features (96x96x3). Pressing `[Save Parameters]` will open a new tab, `Generate Features`. Press the button `[Generate Features]`to generate the features.

Model Design, Training, and Test

In 2007, Google introduced MobileNetV1. In 2018, MobileNetV2: Inverted Residuals and Linear Bottlenecks, was launched, and, in 2019, the V3. The Mobilinet is a family of general-purpose computer vision neural networks explicitly designed for mobile devices to support classification, detection, and other applications. MobileNets are small, low-latency, low-power models parameterized to meet the resource constraints of various use cases.

Although the base MobileNet architecture is already compact and has low latency, a specific use case or application may often require the model to be even smaller and faster. MobileNets introduce a straightforward parameter, (alpha), called the width multiplier to construct these smaller, less computationally expensive models. The role of the width multiplier is to thin a network uniformly at each layer.

Edge Impulse Studio has available MobileNet V1 (96x96 images) and V2 (96x96 and 160x160 images), with several different values (from 0.05 to 1.0). For example, you will get the highest accuracy with V2, 160x160 images, and =1.0. Of course, there is a trade-off. The higher the accuracy, the more memory (around 1.3M RAM and 2.6M ROM) will be needed to run the model, implying more latency. The smaller footprint will be obtained at another extreme with MobileNet V1 and =0.10 (around 53.2K RAM and 101K ROM).

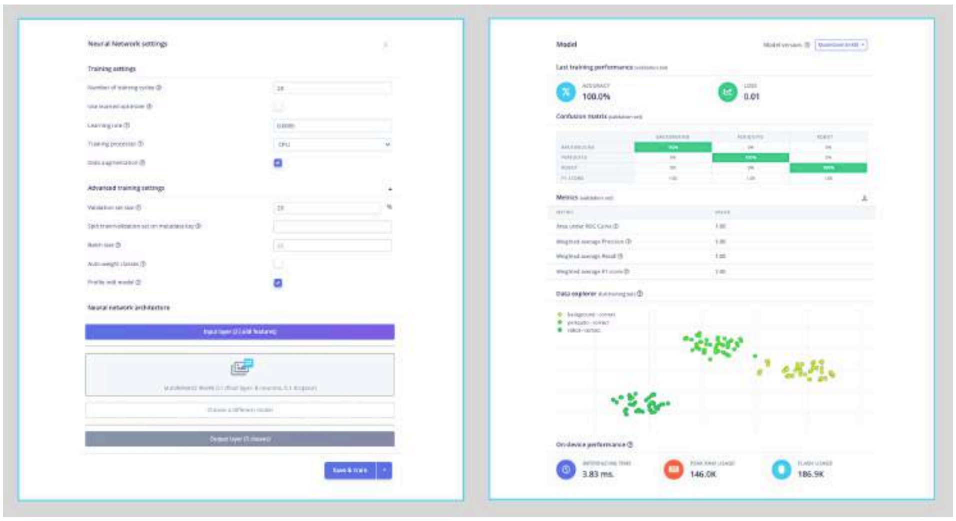
> For comparison, we will use the **MobileNet V2 0.1** as our base model (but a model with a greater alpha can be used here). The final layer of our model, preceding the output layer, will have 8 neurons with a 10% dropout rate for preventing overfitting.

Another necessary technique to use with deep learning is **data augmentation**. Data augmentation is a method that can help improve the accuracy of machine learning models by creating additional artificial data. A data augmentation system makes small, random changes to your training data during the training process (such as flipping, cropping, or rotating the images).

Set the Hyperparameters:

- Epochs: 20,
- Bach Size: 32
- Learning Rate: 0.0005
- Validation size: 20%

Training result:



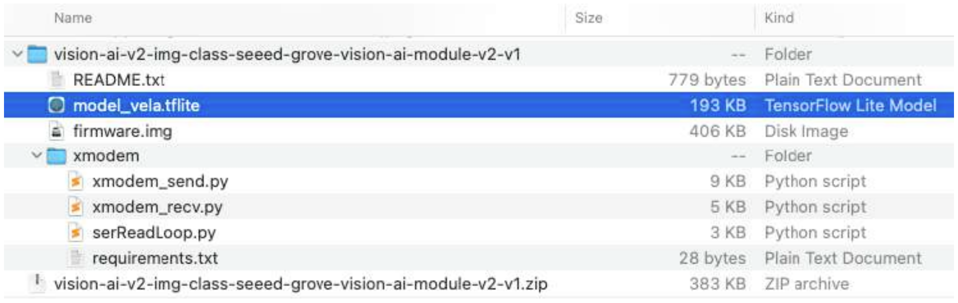
The model profile predicts **146 KB of RAM and 187 KB of Flash**, indicating no problem with the Grove AI Vision (V2), which has almost 2.5 MB of internal SRAM. Additionally, the Studio indicates a **latency of around 4 ms.**

> Despite this, with a 100% accuracy on the Validation set when using the spare data for testing, we confirmed an Accuracy of 81%, using the Quantized (Int8) trained model. However, it is sufficient for our purposes in this lab.

Model Deployment

On the Deployment tab, we should select: `Seeed Grove Vision AI Module V2 (Himax WiseEye2)` and press `[Build]`. A ZIP file will be downloaded to our computer.

The Zip file contains the `model_vela.tflite`, which is a TensorFlow Lite (TFLite) model optimized for neural processing units (NPUs) using the Vela compiler, a tool developed by Arm to adapt TFLite models for Ethos-U NPUs.

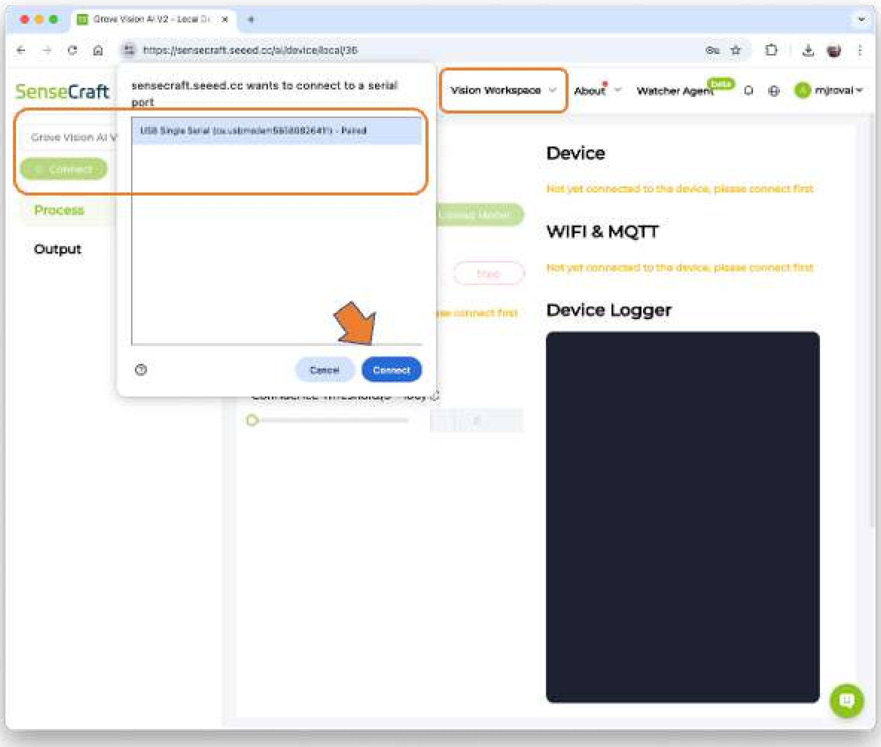


Name	Size	Kind
vision-ai-v2-img-class-seeed-grove-vision-ai-module-v2-v1	--	Folder
README.txt	779 bytes	Plain Text Document
model_vela.tflite	193 KB	TensorFlow Lite Model
firmware.img	406 KB	Disk Image
xmodem	--	Folder
xmodem_send.py	9 KB	Python script
xmodem_recv.py	5 KB	Python script
serReadLoop.py	3 KB	Python script
requirements.txt	28 bytes	Plain Text Document
vision-ai-v2-img-class-seeed-grove-vision-ai-module-v2-v1.zip	383 KB	ZIP archive

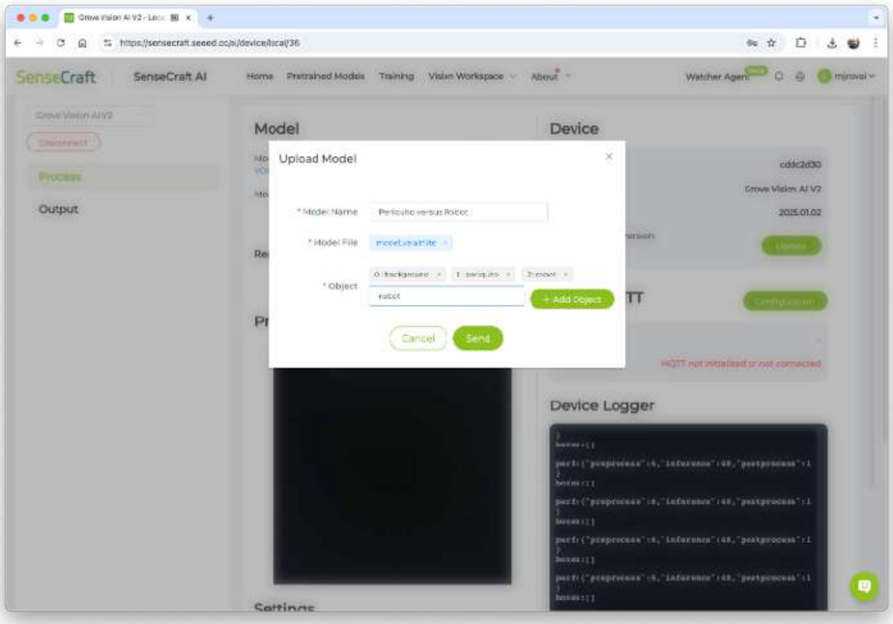
We can flash the model following the instructions in the `README.txt` or use the SenseCraft AI Studio. We will use the latter.

Deploy the model on the SenseCraft AI Studio

On SenseCraft AI Studio, go to the `Vision Workspace` tab, and connect the device:


sensecraft.seeed.cc wants to connect to a serial port
USB Single Serial (cu.usbmodem58180826411) - Paired
Cancel
Connect
Vision Workspace
About
Watcher Agent
Device
Not yet connected to the device, please connect first
WIFI & MQTT
Not yet connected to the device, please connect first
Device Logger
Process
Output

You should see the last model that was uploaded to the device. Select the green button `[Upload Model]`. A pop-up window will ask for the **model name**, the **model file,** and to enter the class names (**objects**). We should use labels following alphabetical order: `0: background`, `1: periquito`, and `2: robot`, and then press `[Send]`.



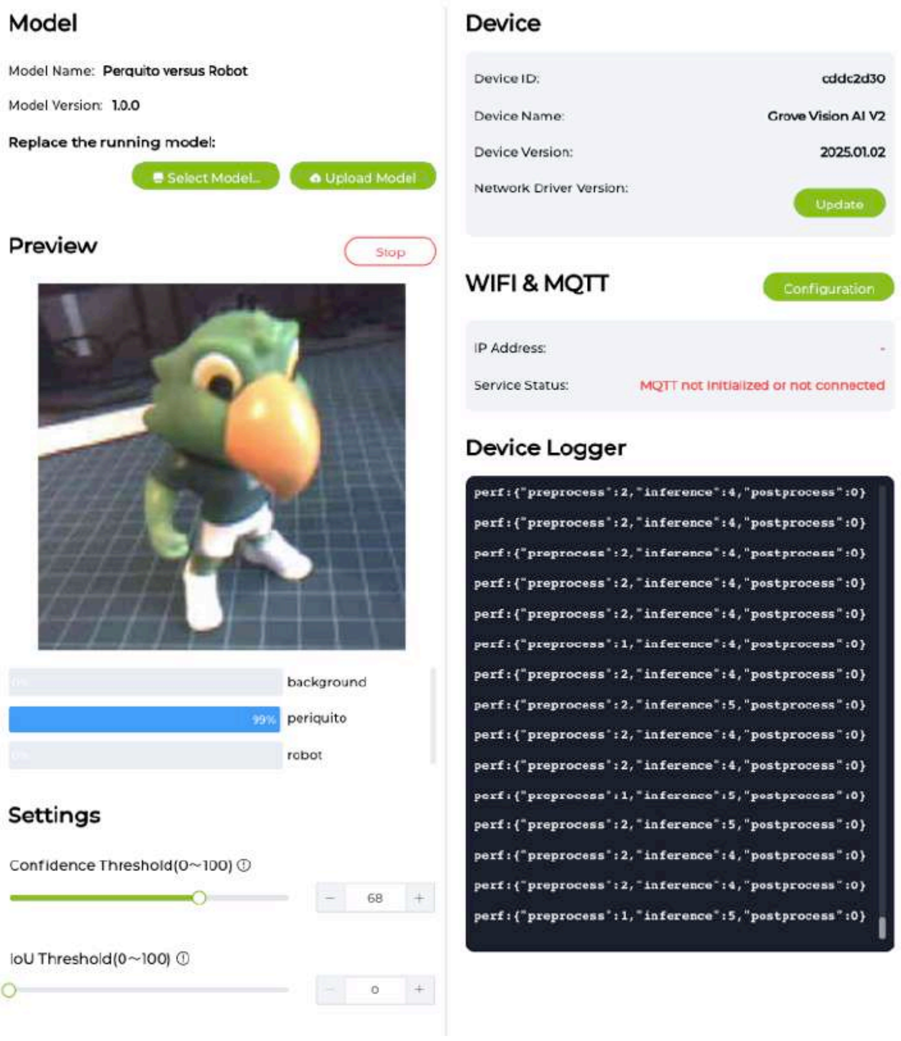
* Model Name: Periquito versus Robot

* Model File: model.eii.tflite

* Object: 0: background, 1: periquito, 2: robot

After a few seconds, the model will be uploaded ("flashed") to our device, and the camera image will appear in real-time on the **Preview** Sector. The Classification result will be displayed under the image preview. It is also possible to select the `Confidence Threshold` of your inference using the cursor on **Settings**.

On the **Device Logger**, we can view the Serial Monitor, where we can observe the latency, which is approximately 1 to 2 ms for pre-processing and 4 to 5 ms for inference, aligning with the estimates made in Edge Impulse Studio.



Model Name: Perquito versus Robot

Model Version: 1.0.0

Replace the running model:

0% background

99% periquito

0% robot

Confidence Threshold(0~100): 68

IoU Threshold(0~100): 0

Device ID: cddc2d30

Device Name: Grove Vision AI V2

Device Version: 2025.01.02

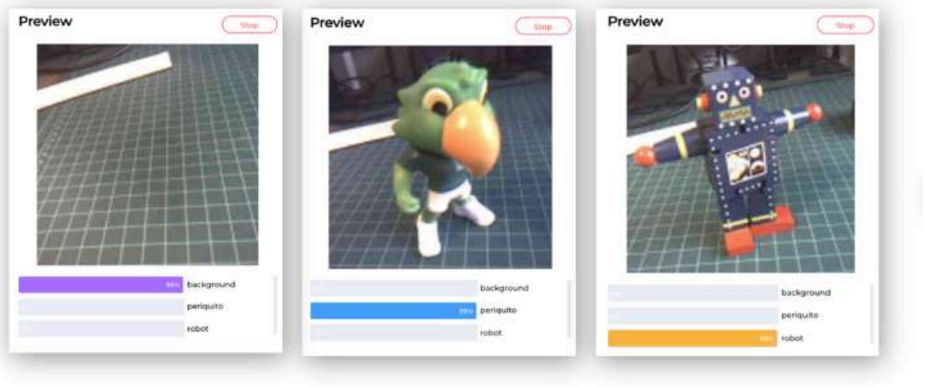
Network Driver Version:

IP Address: -

Service Status: MQTT not initialized or not connected

```
perf:{"preprocess":2,"inference":4,"postprocess":0}
perf:{"preprocess":2,"inference":4,"postprocess":0}
perf:{"preprocess":2,"inference":4,"postprocess":0}
perf:{"preprocess":2,"inference":4,"postprocess":0}
perf:{"preprocess":2,"inference":4,"postprocess":0}
perf:{"preprocess":1,"inference":4,"postprocess":0}
perf:{"preprocess":2,"inference":4,"postprocess":0}
perf:{"preprocess":2,"inference":5,"postprocess":0}
perf:{"preprocess":2,"inference":4,"postprocess":0}
perf:{"preprocess":2,"inference":4,"postprocess":0}
perf:{"preprocess":1,"inference":5,"postprocess":0}
perf:{"preprocess":2,"inference":5,"postprocess":0}
perf:{"preprocess":2,"inference":4,"postprocess":0}
perf:{"preprocess":2,"inference":4,"postprocess":0}
perf:{"preprocess":1,"inference":5,"postprocess":0}
```

Here are other screenshots:

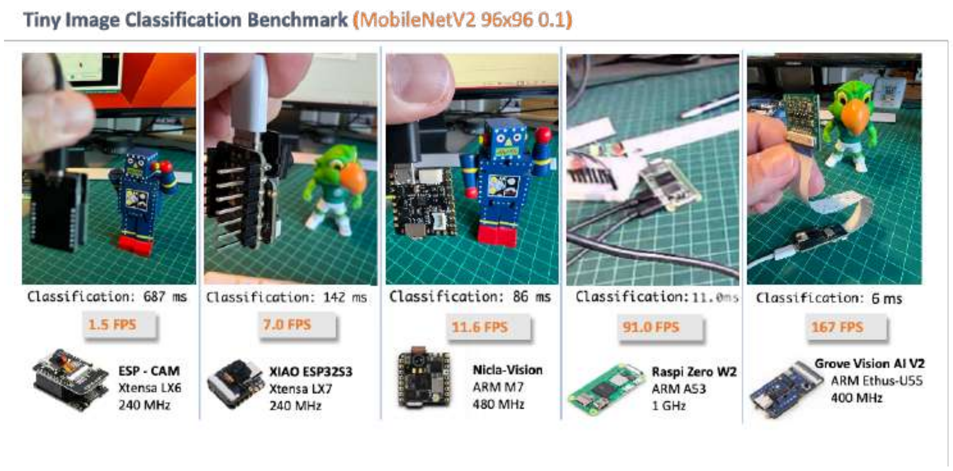


The power consumption of this model is approximately 70 mA, equivalent to 0.4 W.

Image Classification (non-official) Benchmark

Several development boards can be used for embedded machine learning (tinyML), and the most common ones (so far) for Computer Vision applications (with low energy) are the **ESP32 CAM,** the **Seeed XIAO ESP32S3 Sense**, and the Arduino **Nicla Vision**.

Taking advantage of this opportunity, a similarly trained model, MobilenetV2 96x96, with an alpha of 0.1, was also deployed on the ESP-CAM, the XIAO, and a Raspberry Pi Zero W2. Here is the result:



The Grove Vision AI V2 with an **ARM Ethus-U55** was approximately 14 times faster than devices with an ARM-M7, and more than 100 times faster than an Xtensa LX6 (ESP-CAM). Even when compared to a Raspberry Pi, with a much more powerful CPU, the U55 reduces latency by almost half. Additionally, the power consumption is lower than that of other devices (see the full article here for power consumption comparison).

Postprocessing

Now that we have the model uploaded to the board and working correctly, classifying our images, let's connect a Master Device to export the inference result to it and see the result completely offline (disconnected from the PC and, for example, powered by a battery).

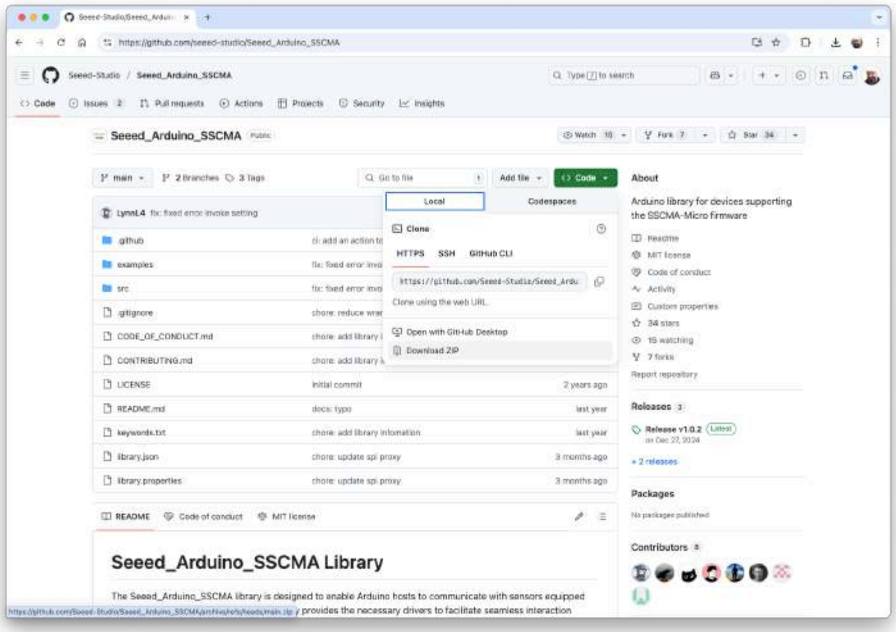
> Note that we can use any microcontroller as a Master Controller, such as the XIAO, Arduino, or Raspberry Pi.

Getting the Video Stream

The image processing and model inference are processed locally in Grove Vision AI (V2), and we want the result to be output to the XIAO (Master Controller) via IIC. For that, we will use the **Arduino SSMA library**. This library's primary purpose is to process Grove Vision AI's data stream, which does not involve model inference.

> The Grove Vision AI (V2) communicates (Inference result) with the XIAO via the IIC; the device's IIC address is 0x62. Image information transfer is via the USB serial port.

Step 1: Download the Arduino SSMA library as a zip file from its GitHub:



https://github.com/seeed-studio/Seeed_Arduino_SSCMA

LynnL4 fix: fixed error invoke setting

.github	ci: add an action to	
examples	fix: fixed error invo	
src	fix: fixed error invo	
.gitignore	chore: reduce wrar	
CODE_OF_CONDUCT.md	chore: add library i	
CONTRIBUTING.md	chore: add library i	
LICENSE	initial commit	2 years ago
README.md	docs: typo	last year
keywords.txt	chore: add library information	last year
library.json	chore: update spi proxy	3 months ago
library.properties	chore: update spi proxy	3 months ago

```
https://github.com/Seeed-Studio/Seeed_Ardu
```
Clone using the web URL.

About

Arduino library for devices supporting the SSCMA-Micro firmware

Readme · MIT license · Code of conduct · Activity · Custom properties · 34 stars · 15 watching · 7 forks · Report repository

Releases 3 · Release v1.0.2 Latest on Dec 27, 2024 · + 2 releases

Packages · No packages published

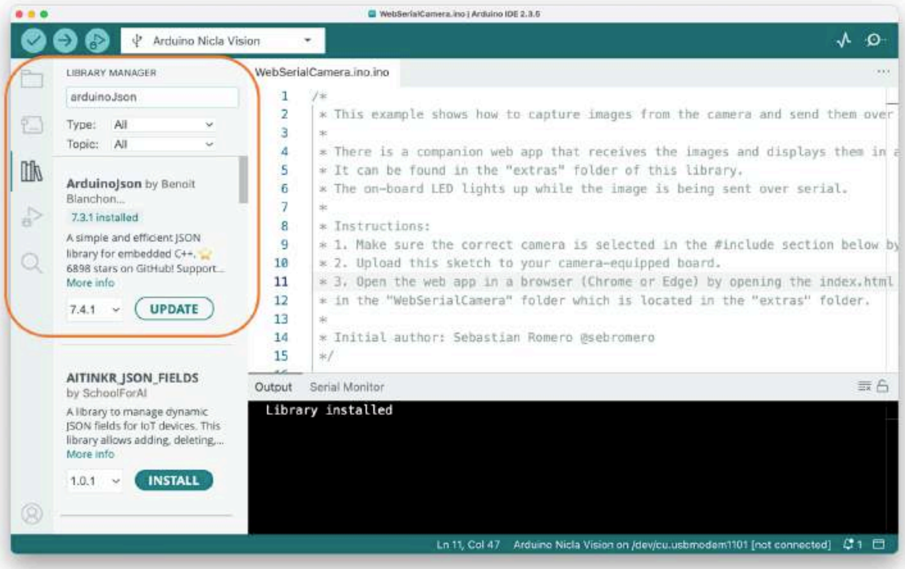
Contributors 8

Seeed_Arduino_SSCMA Library

The Seeed_Arduino_SSCMA library is designed to enable Arduino hosts to communicate with sensors equipped … provides the necessary drivers to facilitate seamless interaction

Step 2: Install it in the Arduino IDE (`sketch > Include Library > Add .Zip Library`).

Step 3: Install the **ArduinoJSON** library.



ArduinoJson by Benoit Blanchon…
7.3.1 installed
A simple and efficient JSON library for embedded C++. ⭐ 6898 stars on GitHub! Support…
More info
7.4.1

AITINKR_JSON_FIELDS by SchoolForAI
A library to manage dynamic JSON fields for IoT devices. This library allows adding, deleting,…
More info
1.0.1

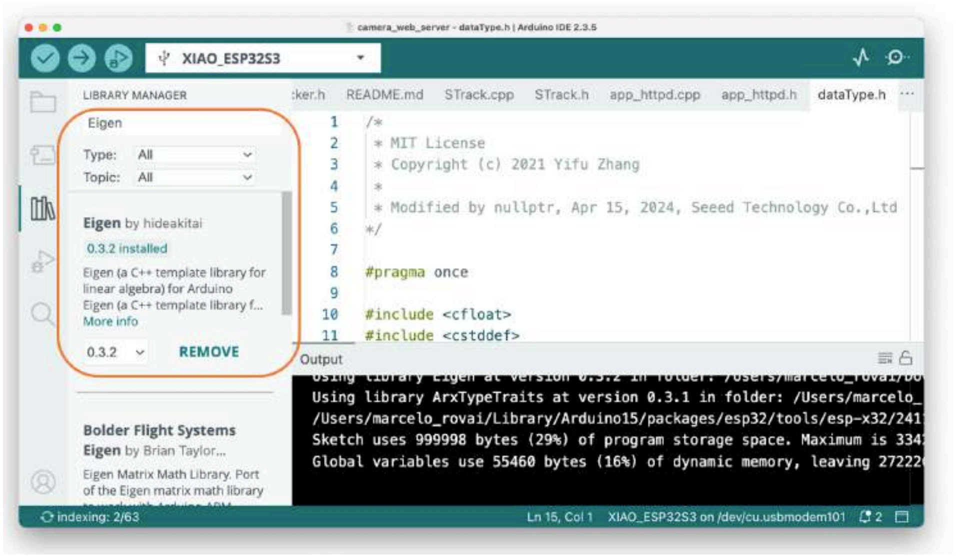
WebSerialCamera.ino.ino

```
/*
 * This example shows how to capture images from the camera and send them over
 *
 * There is a companion web app that receives the images and displays them in a
 * It can be found in the "extras" folder of this library.
 * The on-board LED lights up while the image is being sent over serial.
 *
 * Instructions:
 * 1. Make sure the correct camera is selected in the #include section below by
 * 2. Upload this sketch to your camera-equipped board.
 * 3. Open the web app in a browser (Chrome or Edge) by opening the index.html
 * in the "WebSerialCamera" folder which is located in the "extras" folder.
 *
 * Initial author: Sebastian Romero @sebromero
 */
```

```
Library installed
```

Ln 11, Col 47   Arduino Nicla Vision on /dev/cu.usbmodem1101 [not connected]

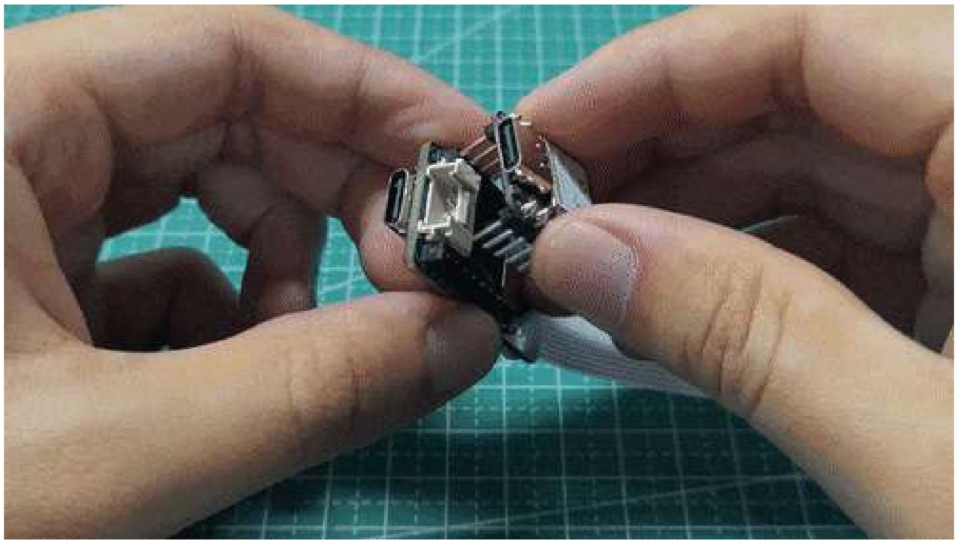
Step 4: Install the **Eigen** Library



```
/*
 * MIT License
 * Copyright (c) 2021 Yifu Zhang
 *
 * Modified by nullptr, Apr 15, 2024, Seeed Technology Co.,Ltd
*/

#pragma once

#include <cfloat>
#include <cstddef>
```

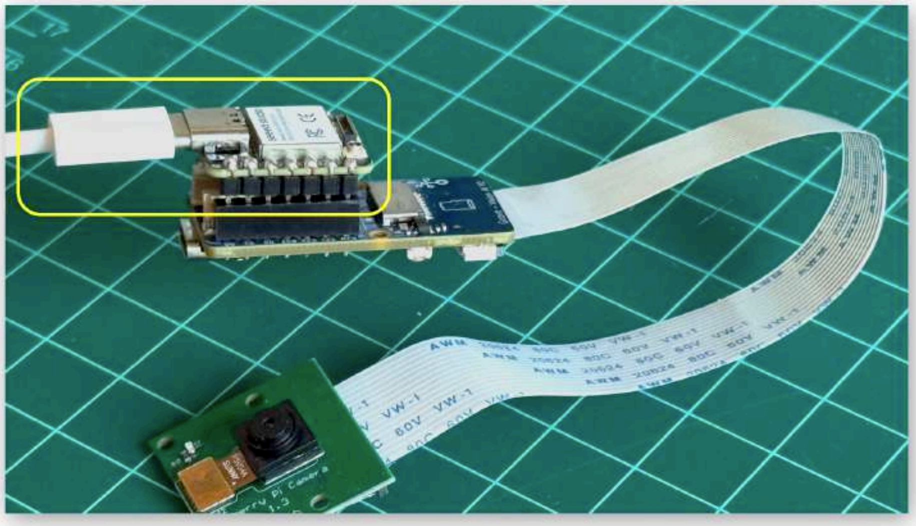
```
Using library ArxTypeTraits at version 0.3.1 in folder: /Users/marcelo_
/Users/marcelo_rovai/Library/Arduino15/packages/esp32/tools/esp-x32/241
Sketch uses 999998 bytes (29%) of program storage space. Maximum is 334
Global variables use 55460 bytes (16%) of dynamic memory, leaving 27222
```

Step 3: Now, connect the XIAO and Grove Vision AI (V2) via the socket (a row of pins) located at the back of the device.



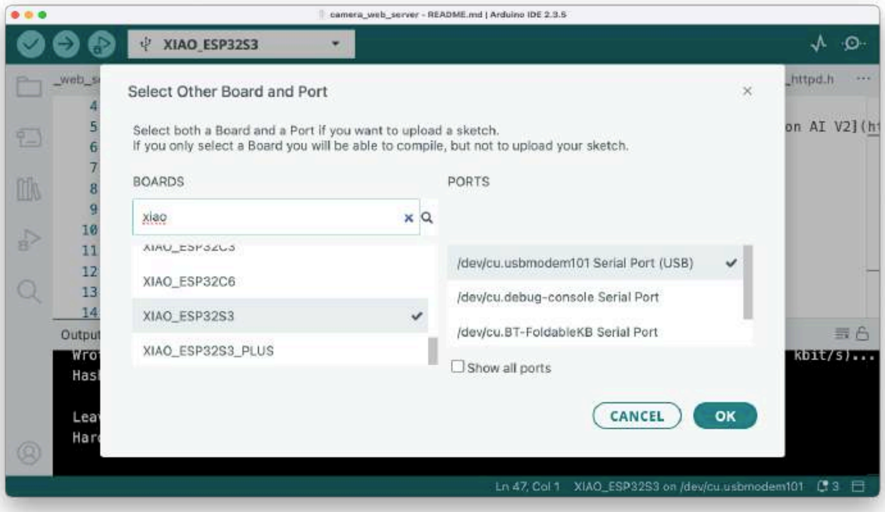
> **CAUTION**: Please note the direction of the connection, Grove Vision AI's Type-C connector should be in the same direction as XIAO's Type-C connector.

Step 5: Connect the **XIAO USB-C** port to your computer



Step 6: In the Arduino IDE, select the Xiao board and the corresponding USB port.

Once we want to stream the video to a webpage, we will use the **XIAO ESP32S3**, which has wifi and enough memory to handle images. Select `XIAO_ESP32S3` and the appropriate USB Port:



camera_web_server - README.md | Arduino IDE 2.3.5

XIAO_ESP32S3

Select Other Board and Port

Select both a Board and a Port if you want to upload a sketch.
If you only select a Board you will be able to compile, but not to upload your sketch.

BOARDS: xiao
- XIAO_ESP32C3
- XIAO_ESP32C6
- XIAO_ESP32S3 ✓
- XIAO_ESP32S3_PLUS

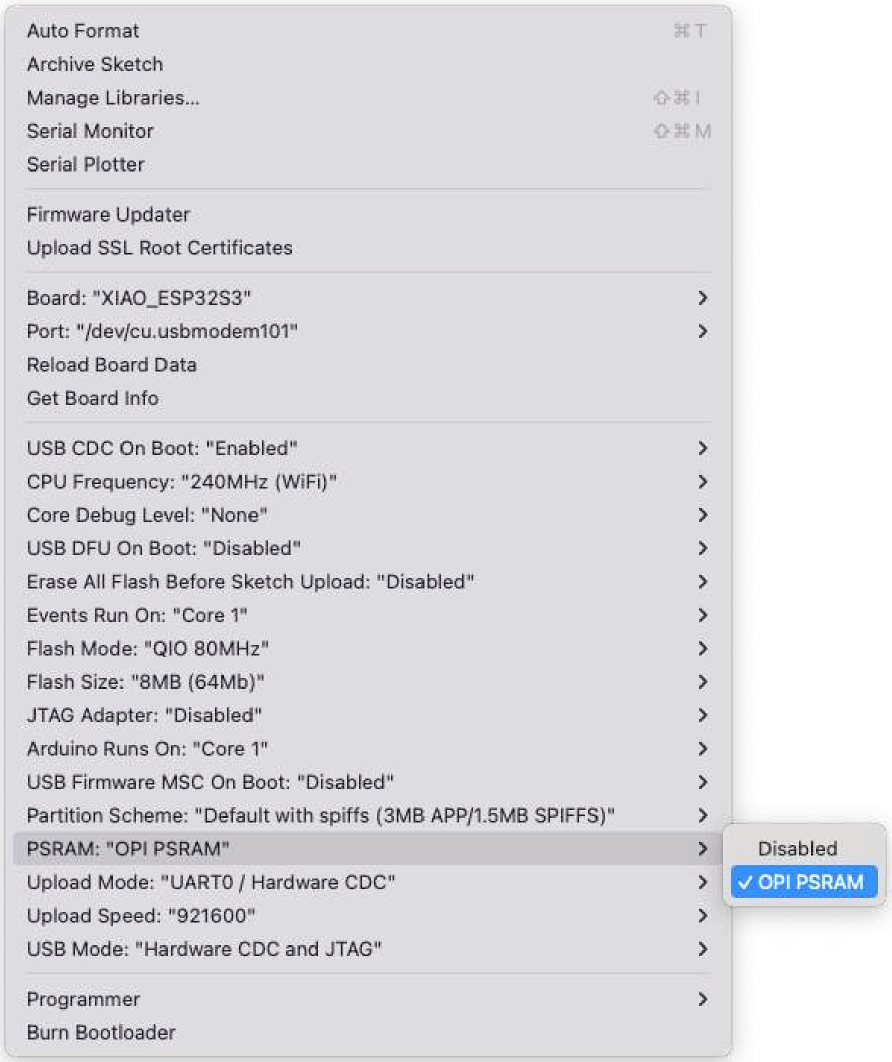
PORTS:
- /dev/cu.usbmodem101 Serial Port (USB) ✓
- /dev/cu.debug-console Serial Port
- /dev/cu.BT-FoldableKB Serial Port

☐ Show all ports

CANCEL OK

Ln 47, Col 1 XIAO_ESP32S3 on /dev/cu.usbmodem101

By default, the PSRAM is disabled. Open the `Tools` menu and on PSRAM: `"OPI PSRAM"`select `OPI PSRAM`.


Auto Format ⌘T
Archive Sketch
Manage Libraries... ⇧⌘I
Serial Monitor ⇧⌘M
Serial Plotter

Firmware Updater
Upload SSL Root Certificates

Board: "XIAO_ESP32S3"
Port: "/dev/cu.usbmodem101"
Reload Board Data
Get Board Info

USB CDC On Boot: "Enabled"
CPU Frequency: "240MHz (WiFi)"
Core Debug Level: "None"
USB DFU On Boot: "Disabled"
Erase All Flash Before Sketch Upload: "Disabled"
Events Run On: "Core 1"
Flash Mode: "QIO 80MHz"
Flash Size: "8MB (64Mb)"
JTAG Adapter: "Disabled"
Arduino Runs On: "Core 1"
USB Firmware MSC On Boot: "Disabled"
Partition Scheme: "Default with spiffs (3MB APP/1.5MB SPIFFS)"
PSRAM: "OPI PSRAM" — Disabled [✓ OPI PSRAM]
Upload Mode: "UART0 / Hardware CDC"
Upload Speed: "921600"
USB Mode: "Hardware CDC and JTAG"

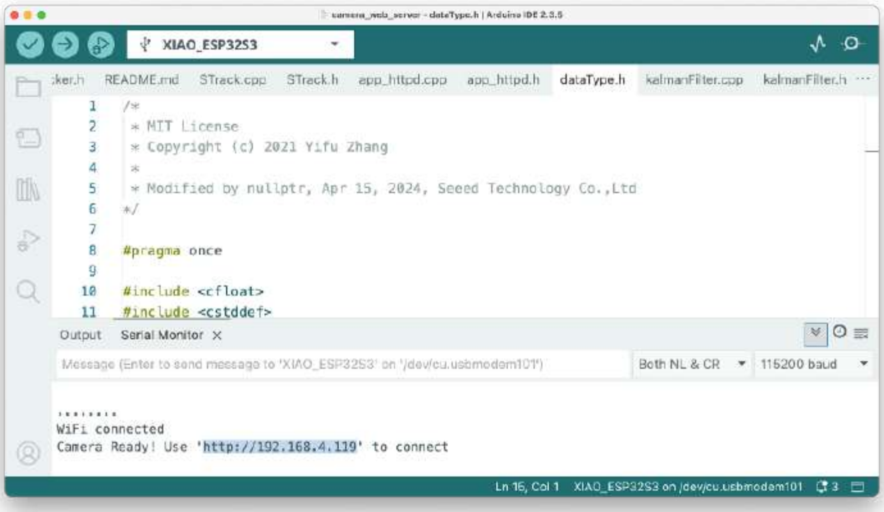
Programmer
Burn Bootloader

Step 7: Open the example in Arduino IDE:

`File` -> `Examples` -> `Seeed_Arduino_SSCMA` -> `camera_web_server`.

And edit the `ssid` and `password` in the `camera_web_server.ino` sketch to match the Wi-Fi network.

Step 8: Upload the sketch to the board and open the Serial Monitor. When connected to the Wi-Fi network, the board's IP address will be displayed.


camera_web_server - dataType.h | Arduino IDE 2.3.5
XIAO_ESP32S3
ker.h README.md STrack.cpp STrack.h app_httpd.cpp app_httpd.h dataType.h kalmanFilter.cpp kalmanFilter.h

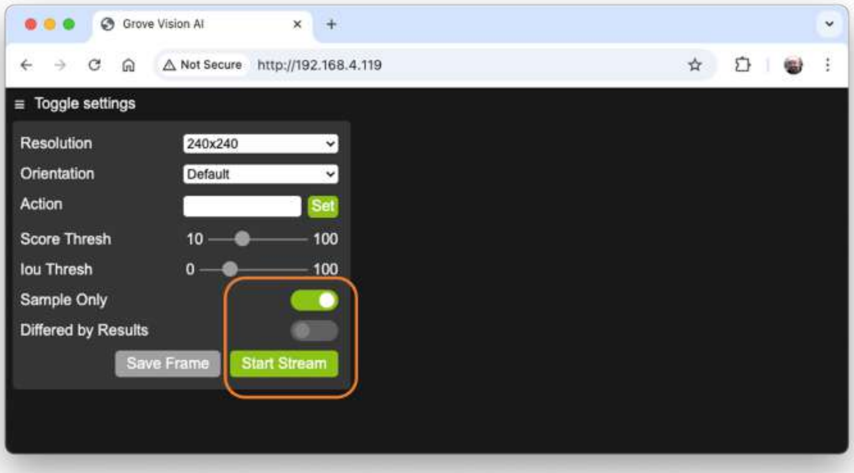
```
/*
 * MIT License
 * Copyright (c) 2021 Yifu Zhang
 *
 * Modified by nullptr, Apr 15, 2024, Seeed Technology Co.,Ltd
*/

#pragma once

#include <cfloat>
#include <cstddef>
```

Output Serial Monitor
Message (Enter to send message to 'XIAO_ESP32S3' on '/dev/cu.usbmodem101')
Both NL & CR
115200 baud

```
........
WiFi connected
Camera Ready! Use 'http://192.168.4.119' to connect
```

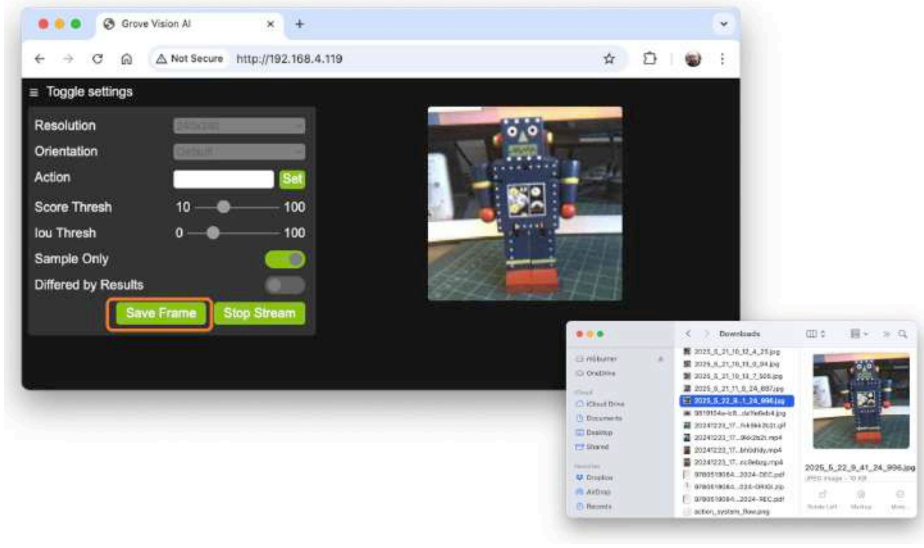
Ln 16, Col 1 XIAO_ESP32S3 on /dev/cu.usbmodem101 3

Open the address using a web browser. A Video App will be available. To see **only** the video stream from the Grove Vision AI V2, press `[Sample Only]` and `[Start Stream]`.



Resolution 240x240
Orientation Default
Action
Score Thresh 10 100
Iou Thresh 0 100
Sample Only
Differed by Results

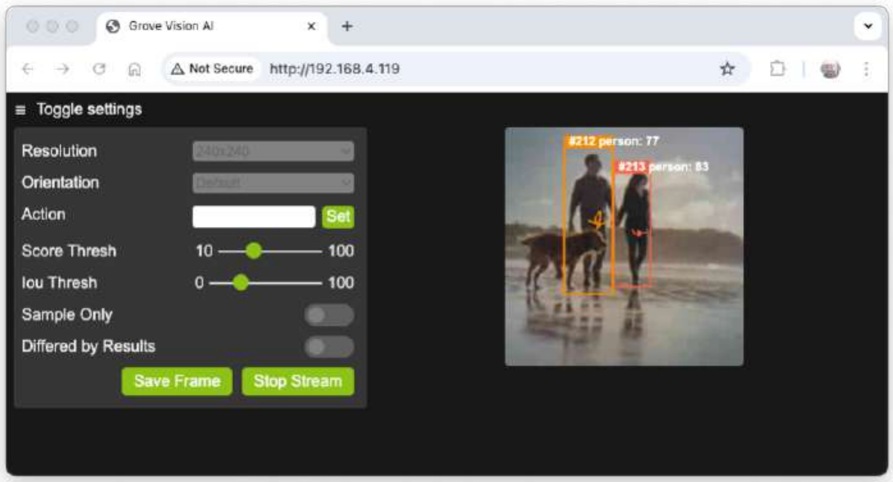
If you want to create an image dataset, you can use this app, saving frames of the video generated by the device. Pressing `[Save Frame]`, the image will be saved in the download area of our desktop.



Resolution 240x240
Orientation Default
Action
Score Thresh 10 — 100
Iou Thresh 0 — 100
Sample Only
Differed by Results

Opening the App **without** selecting `[Sample Only]`, the inference result should appear on the video screen, but this does not happen for Image Classification. For Object Detection or Pose Estimation, the result is embedded with the video stream.

For example, if the model is a Person Detection using YoloV8:

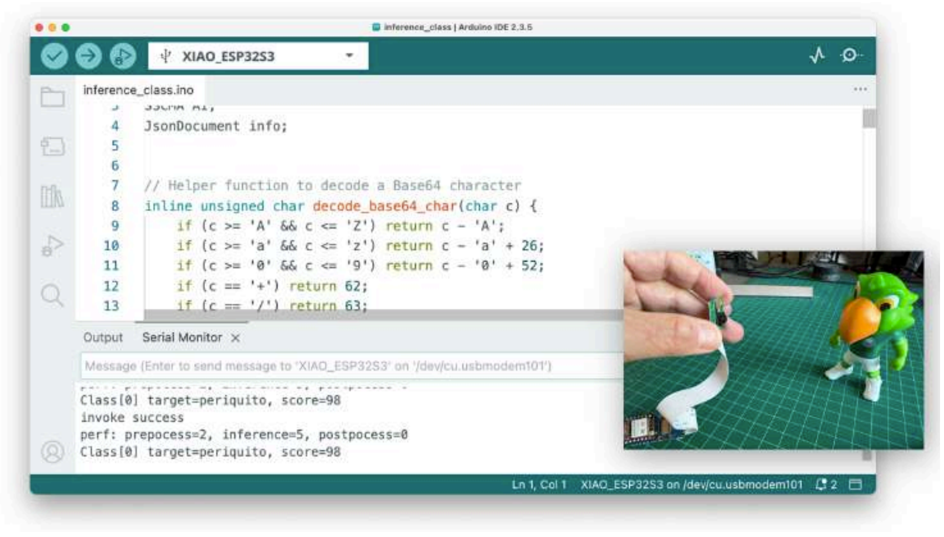


Resolution 240x240
Orientation Default
Action
Score Thresh 10 — 100
Iou Thresh 0 — 100
Sample Only
Differed by Results

Getting the Inference Result

- Go to `File` -> `Examples` -> `Seeed_Arduino_SSCMA` -> `inference_class`.

- Upload the sketch to the board, and open the Serial Monitor.
- Pointing the camera at one of our objects, we can see the inference result on the Serial Terminal.


inference_class | Arduino IDE 2.3.5
XIAO_ESP32S3
inference_class.ino
Output
Serial Monitor
Message (Enter to send message to 'XIAO_ESP32S3' on '/dev/cu.usbmodem101')
Ln 1, Col 1 XIAO_ESP32S3 on /dev/cu.usbmodem101

```
JsonDocument info;


// Helper function to decode a Base64 character
inline unsigned char decode_base64_char(char c) {
    if (c >= 'A' && c <= 'Z') return c - 'A';
    if (c >= 'a' && c <= 'z') return c - 'a' + 26;
    if (c >= '0' && c <= '9') return c - '0' + 52;
    if (c == '+') return 62;
    if (c == '/') return 63;
```

```
Class[0] target=periquito, score=98
invoke success
perf: prepocess=2, inference=5, postpocess=0
Class[0] target=periquito, score=98
```

The inference running on the Arduino IDE had an average consumption of 160 mA or 800 mW and a peak of 330 mA 1.65 W when transmitting the image to the App.

Postprocessing with LED

The idea behind our postprocessing is that whenever a specific image is detected (for example, the Periquito - Label:1), the User LED is turned on. If the Robot or a background is detected, the LED will be off.

Copy the below code and past it to your IDE:

```
#include <Seeed_Arduino_SSCMA.h>
SSCMA AI;

void setup()
{
    AI.begin();

    Serial.begin(115200);
    while (!Serial);
    Serial.println("Inferencing - Grove AI V2 / XIAO ESP32S3");
    // Pins for the built-in LED
    pinMode(LED_BUILTIN, OUTPUT);
    // Ensure the LED is OFF by default.
    // Note: The LED is ON when the pin is LOW, OFF when HIGH.
    digitalWrite(LED_BUILTIN, HIGH);
}

void loop()
{
    if (!AI.invoke()){
        Serial.println("\nInvoke Success");
        Serial.print("Latency [ms]: prepocess=");
        Serial.print(AI.perf().prepocess);
        Serial.print(", inference=");
        Serial.print(AI.perf().inference);
        Serial.print(", postpocess=");
        Serial.println(AI.perf().postprocess);
        int pred_index = AI.classes()[0].target;
        Serial.print("Result= Label: ");
        Serial.print(pred_index);
        Serial.print(", score=");
        Serial.println(AI.classes()[0].score);
        turn_on_led(pred_index);
    }
}

/**
* @brief      turn_off_led function - turn-off the User LED
*/
void turn_off_led(){
    digitalWrite(LED_BUILTIN, HIGH);
}

/**
* @brief      turn_on_led function used to turn on the User LED
* @param[in]  pred_index
*             label 0: [0] ==> ALL OFF
*             label 1: [1] ==> LED ON
*             label 2: [2] ==> ALL OFF
*             label 3: [3] ==> ALL OFF
*/
```

```
void turn_on_led(int pred_index) {
    switch (pred_index)
    {
        case 0:
            turn_off_led();
            break;
        case 1:
            turn_off_led();
            digitalWrite(LED_BUILTIN, LOW);
            break;
        case 2:
            turn_off_led();
            break;
        case 3:
            turn_off_led();
            break;
    }
}
```

This sketch uses the Seeed_Arduino_SSCMA.h library to interface with the Grove Vision AI Module V2. The AI module and the LED are initialized in the `setup()` function, and serial communication is started.

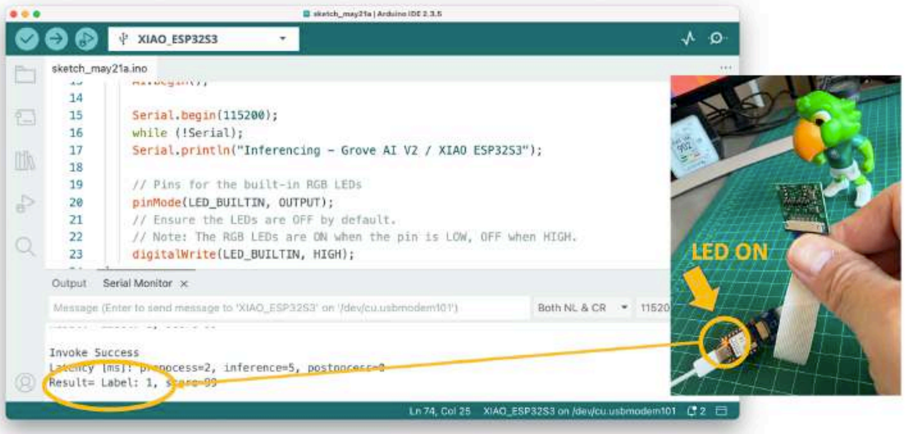
The `loop()` function repeatedly calls the `invoke()` method to perform inference using the built-in algorithms of the Grove Vision AI Module V2. Upon a successful inference, the sketch prints out performance metrics to the serial monitor, including preprocessing, inference, and postprocessing times.

The sketch processes and prints out detailed information about the results of the inference:

- (`AI.classes()[0]`) that identifies the class of image (`.target`) and its confidence score (`.score`).
- The inference result (class) is stored in the integer variable `pred_index`, which will be used as an input to the function `turn_on_led()`. As a result, the LED will turn ON, depending on the classification result.

Here is the result:

If the Periquito is detected (Label:1), the LED is ON:

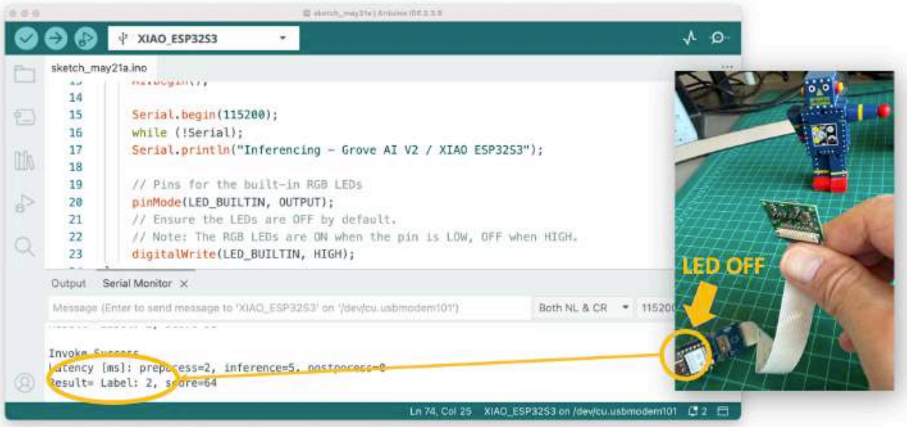


```
Serial.begin(115200);
while (!Serial);
Serial.println("Inferencing - Grove AI V2 / XIAO ESP32S3");

// Pins for the built-in RGB LEDs
pinMode(LED_BUILTIN, OUTPUT);
// Ensure the LEDs are OFF by default.
// Note: The RGB LEDs are ON when the pin is LOW, OFF when HIGH.
digitalWrite(LED_BUILTIN, HIGH);
```

```
Invoke Success
Latency [ms]: preprocess=2, inference=5, postprocess=0
Result= Label: 1, score=99
```

sketch_may21a | Arduino IDE 2.3.5
XIAO_ESP32S3
sketch_may21a.ino
Output
Serial Monitor
Message (Enter to send message to 'XIAO_ESP32S3' on '/dev/cu.usbmodem101')
Both NL & CR
11520
Ln 74, Col 25
XIAO_ESP32S3 on /dev/cu.usbmodem101
LED ON

If the Robot is detected (Label:2) the LED is OFF (Same for Background (Label:0):



```
Serial.begin(115200);
while (!Serial);
Serial.println("Inferencing - Grove AI V2 / XIAO ESP32S3");

// Pins for the built-in RGB LEDs
pinMode(LED_BUILTIN, OUTPUT);
// Ensure the LEDs are OFF by default.
// Note: The RGB LEDs are ON when the pin is LOW, OFF when HIGH.
digitalWrite(LED_BUILTIN, HIGH);
```

```
Invoke Success
Latency [ms]: preprocess=2, inference=5, postprocess=0
Result= Label: 2, score=64
```

sketch_may21a | Arduino IDE 2.3.5
XIAO_ESP32S3
sketch_may21a.ino
Output
Serial Monitor
Message (Enter to send message to 'XIAO_ESP32S3' on '/dev/cu.usbmodem101')
Both NL & CR
11520
Ln 74, Col 25
XIAO_ESP32S3 on /dev/cu.usbmodem101
LED OFF

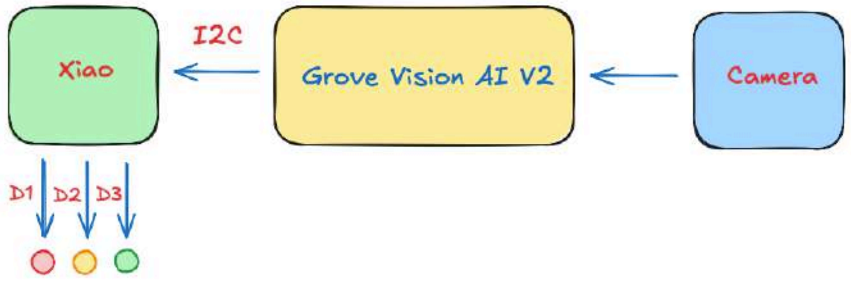
Therefore, we can now power the Grove Viaon AI V2 + Xiao ESP32S3 with an external battery, and the inference result will be displayed by the LED completely offline. The consumption is approximately 165 mA or 825 mW.

> It is also possible to send the result using Wifi, BLE, or other communication protocols available on the used Master Device.

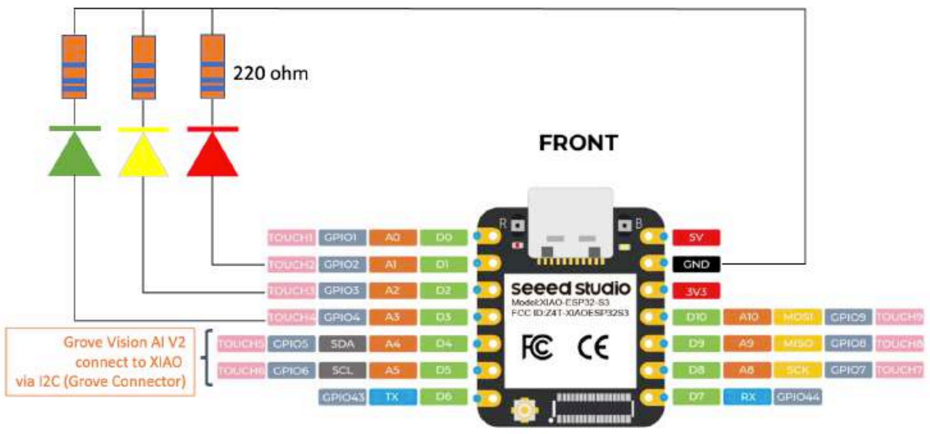
Optional: Post-processing on external devices

Of course, one of the significant advantages of working with EdgeAI is that devices can run entirely disconnected from the cloud, allowing for seamless **interactions with the real world**.

We did it in the last section, but using the internal Xiao LED. Now, we will connect external LEDs (which could be any actuator).



The LEDS should be connected to the XIAO ground via a 220-ohm resistor.



The idea is to modify the previous sketch to handle the three external LEDs.

GOAL: Whenever the image of a **Periquito** is detected, the LED **Green** will be ON; if it is a **Robot**, the LED **Yellow** will be ON; if it is a **Background**, the **LED Red** will be ON.

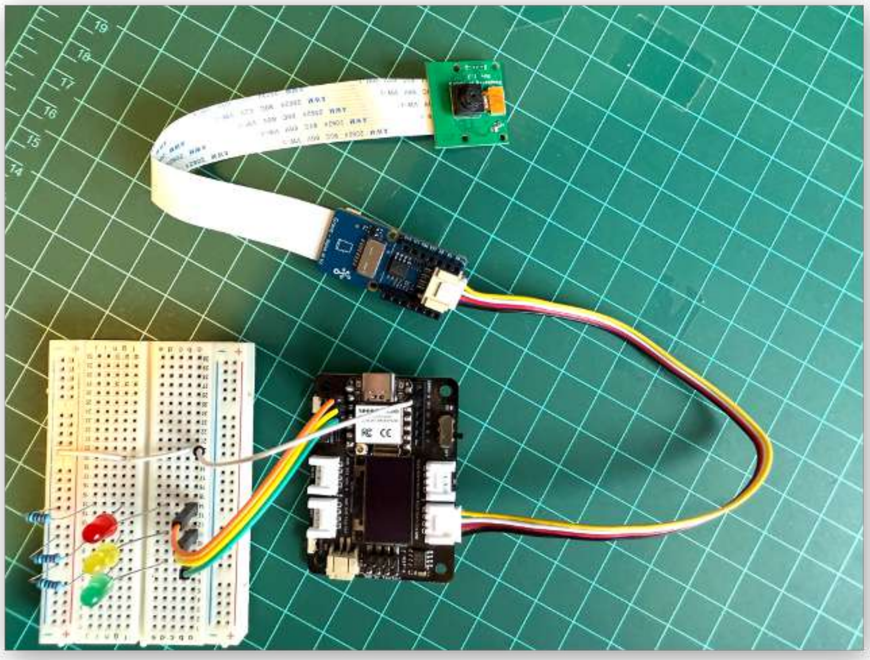
The image processing and model inference are processed locally in Grove Vision AI (V2), and we want the result to be output to the XIAO via IIC. For that, we will use the Arduino SSMA library again.

Here the scketch to be used:

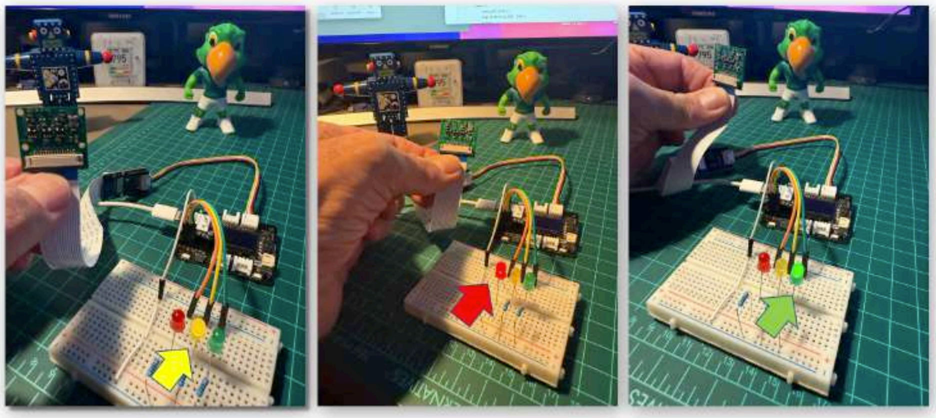
```
#include <Seeed_Arduino_SSCMA.h>
SSCMA AI;
```

```
// Define the LED pin according to the pin diagram
// The LEDS negative lead should be connected to the XIAO ground
// via a 220-ohm resistor.
int LEDR = D1; # XIAO ESP32S3 Pin 1
int LEDY = D2; # XIAO ESP32S3 Pin 2
int LEDG = D3; # XIAO ESP32S3 Pin 3

  void setup()
{
    AI.begin();

    Serial.begin(115200);
    while (!Serial);
    Serial.println("Inferencing - Grove AI V2 / XIAO ESP32S3");

// Initialize the external LEDs
    pinMode(LEDR, OUTPUT);
    pinMode(LEDY, OUTPUT);
    pinMode(LEDG, OUTPUT);
    // Ensure the LEDs are OFF by default.
    // Note: The LEDs are ON when the pin is HIGH, OFF when LOW.
    digitalWrite(LEDR, LOW);
    digitalWrite(LEDY, LOW);
    digitalWrite(LEDG, LOW);
}

void loop()
{
    if (!AI.invoke()){
        Serial.println("\nInvoke Success");
        Serial.print("Latency [ms]: prepocess=");
        Serial.print(AI.perf().prepocess);
        Serial.print(", inference=");
        Serial.print(AI.perf().inference);
        Serial.print(", postpocess=");
        Serial.println(AI.perf().postprocess);
        int pred_index = AI.classes()[0].target;
        Serial.print("Result= Label: ");
        Serial.print(pred_index);
        Serial.print(", score=");
        Serial.println(AI.classes()[0].score);
        turn_on_leds(pred_index);
    }
}


/**
* @brief turn_off_leds function - turn-off all LEDs
*/
void turn_off_leds(){
    digitalWrite(LEDR, LOW);
    digitalWrite(LEDY, LOW);
    digitalWrite(LEDG, LOW);
}

/**
* @brief turn_on_leds function used to turn on an specif LED
* @param[in]  pred_index
*             label 0: [0] ==> Red ON
*             label 1: [1] ==> Green ON
*             label 2: [2] ==> Yellow ON
*/
void turn_on_leds(int pred_index) {
    switch (pred_index)
    {
        case 0:
            turn_off_leds();
            digitalWrite(LEDR, HIGH);
            break;
        case 1:
            turn_off_leds();
            digitalWrite(LEDG, HIGH);
            break;
        case 2:
            turn_off_leds();
            digitalWrite(LEDY, HIGH);
            break;
        case 3:
            turn_off_leds();
            break;
    }
}
```

We should connect the Grove Vision AI V2 with the XIAO using its I2C Grove connector. For the XIAO, we will use an Expansion Board for the facility (although it is possible to connect the I2C directly to the XIAO's pins). We will power the boards using the USB-C connector, but a battery can also be used.



Here is the result:



The power consumption reached a peak of 240 mA (Green LED), equivalent to

1.2 W. Driving the Yellow and Red LEDs consumes 14 mA, equivalent to 0.7 W. Sending information to the terminal via serial has no impact on power consumption.

Conclusion

In this lab, we've explored the complete process of developing an image classification system using the Seeed Studio Grove Vision AI Module V2 powered by the Himax WiseEye2 chip. We've walked through every stage of the machine learning workflow, from defining our project goals to deploying a working model with real-world interactions.

The Grove Vision AI V2 has demonstrated impressive performance, with inference times of just 4-5ms, dramatically outperforming other common tinyML platforms. Our benchmark comparison showed it to be approximately 14 times faster than ARM-M7 devices and over 100 times faster than an Xtensa LX6 (ESP-CAM). Even when compared to a Raspberry Pi Zero W2, the Edge TPU architecture delivered nearly twice the speed while consuming less power.

Through this project, we've seen how transfer learning enables us to achieve good classification results with a relatively small dataset of custom images. The MobileNetV2 model with an alpha of 0.1 provided an excellent balance of accuracy and efficiency for our three-class problem, requiring only 146 KB of RAM and 187 KB of Flash memory, well within the capabilities of the Grove Vision AI Module V2's 2.4 MB internal SRAM.

We also explored several deployment options, from viewing inference results through the SenseCraft AI Studio to creating a standalone system with visual feedback using LEDs. The ability to stream video to a web browser and process inference results locally demonstrates the versatility of edge AI systems for real-world applications.

The power consumption of our final system remained impressively low, ranging from approximately 70mA (0.4W) for basic inference to 240mA (1.2W) when driving external components. This efficiency makes the Grove Vision AI Module V2 an excellent choice for battery-powered applications where power consumption is critical.

This lab has demonstrated that sophisticated computer vision tasks can now be performed entirely at the edge, without reliance on cloud services or powerful computers. With tools like Edge Impulse Studio and SenseCraft AI Studio, the development process has become accessible even to those without extensive machine learning expertise.

As edge AI technology continues to evolve, we can expect even more powerful capabilities from compact, energy-efficient devices like the Grove Vision AI Module V2, opening up new possibilities for smart sensors, IoT applications, and embedded intelligence in everyday objects.

Resources

Collecting Images with SenseCraft AI Studio.

Edge Impulse Studio Project

SenseCraft AI Studio - Vision Workplace (Deploy Models)

Other Himax examples

Arduino Sketches

Object Detection

This Lab is under Development

#

References & Author

References

To learn more:

Online Courses

- Harvard School of Engineering and Applied Sciences - CS249r: Tiny Machine Learning
- Professional Certificate in Tiny Machine Learning (TinyML) – edX/Harvard
- Introduction to Embedded Machine Learning - Coursera/Edge Impulse
- Computer Vision with Embedded Machine Learning - Coursera/Edge Impulse
- UNIFEI-IESTI01 TinyML: "Machine Learning for Embedding Devices"

Books

- "Python for Data Analysis" by Wes McKinney
- "Deep Learning with Python" by François Chollet - GitHub Notebooks
- "TinyML" by Pete Warden and Daniel Situnayake
- "TinyML Cookbook 2nd Edition" by Gian Marco Iodice
- "Technical Strategy for AI Engineers, In the Era of Deep Learning" by Andrew Ng
- "AI at the Edge" book by Daniel Situnayake and Jenny Plunkett
- "XIAO: Big Power, Small Board" by Lei Feng and Marcelo Rovai
- "MACHINE LEARNING SYSTEMS for TinyML" by a collaborative effort

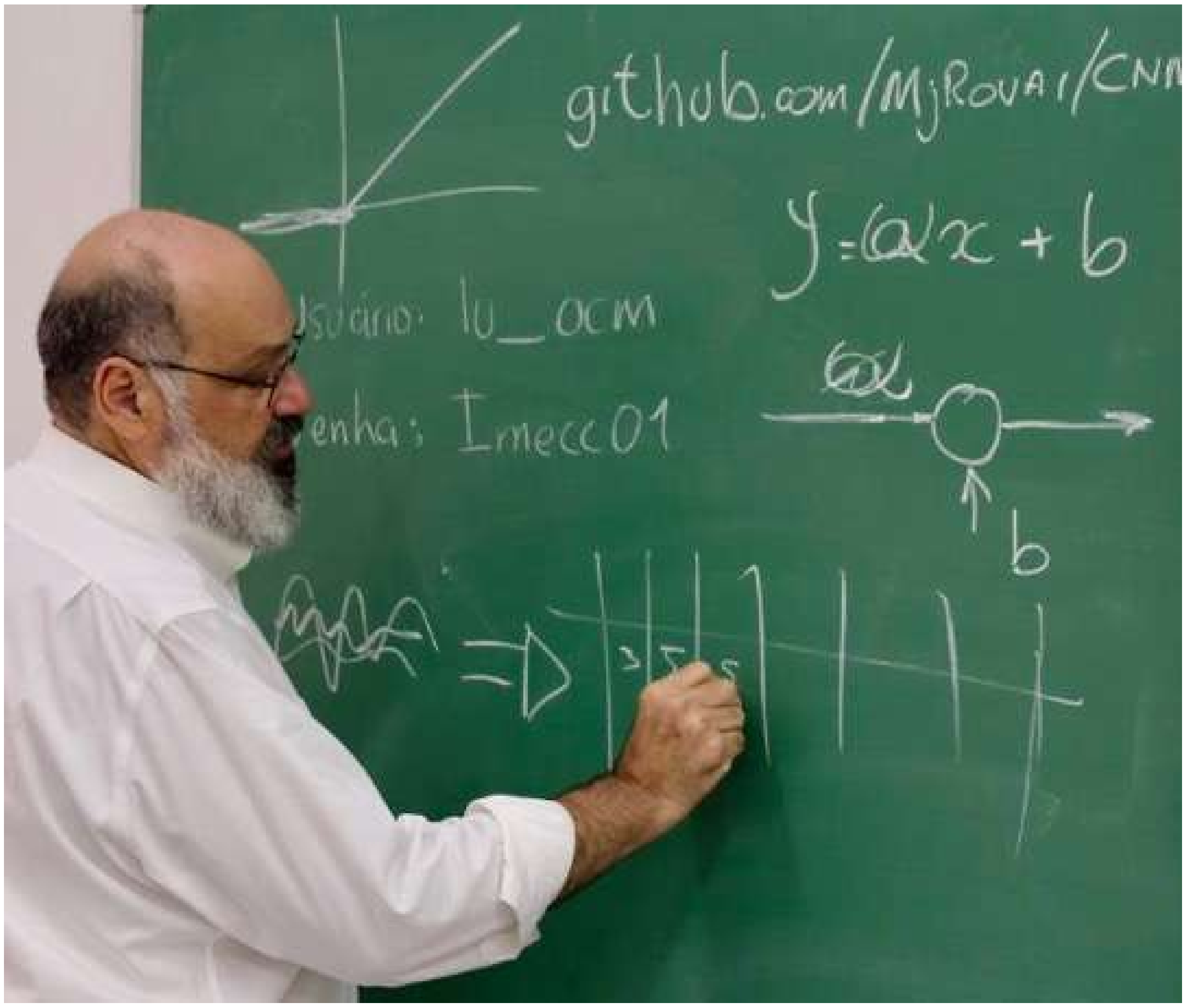
Projects Repository

- Edge Impulse Expert Network

TinyML4D

TinyML Made Easy, an eBook collection of a series of Hands-On tutorials, is part of the TinyML4D, an initiative to make Embedded Machine Learning (TinyML) education available to everyone, explicitly enabling innovative solutions for the unique challenges Developing Countries face.

About the author



Marcelo Rovai, a Brazilian living in Chile, is a recognized figure in engineering and technology education. He holds the title of Professor Honoris Causa from the Federal University of Itajubá (UNIFEI), Brazil. His educational background includes an Engineering degree from UNIFEI and a specialization from the Polytechnic School of São Paulo University (USP). Further enhancing his expertise, he earned an MBA from IBMEC (INSPER) and a Master's in Data Science from the Universidad del Desarrollo (UDD) in Chile.

With a career spanning several high-profile technology companies, including AVIBRAS Airspace, ATT, NCR, and IGT, where he served as Vice President for Latin America, he brings a wealth of industry experience to his academic endeavors. He is a prolific writer on electronics-related topics and shares his knowledge through open platforms like Hackster.io.

In addition to his professional pursuits, he is dedicated to educational outreach, serving as a volunteer professor at UNIFEI and as a Co-Chair of the TinyML4D group, which promotes TinyML education in developing countries. His work underscores a commitment to leveraging technology for societal advancement.

LinkedIn profile: https://www.linkedin.com/in/marcelo-jose-rovai-brazil-chile/

Lectures, books, papers, and tutorials: https://github.com/Mjrovai/TinyML4D