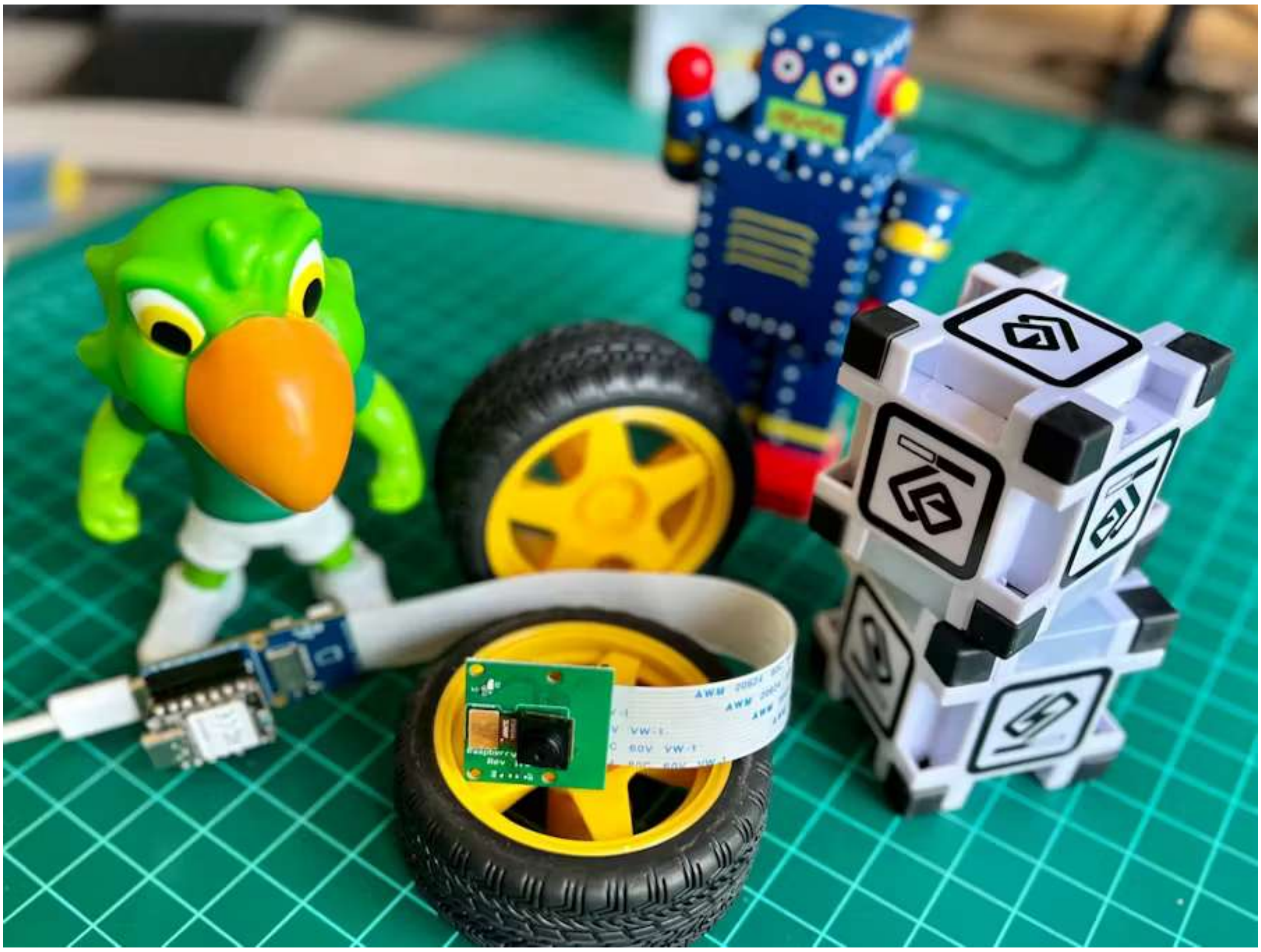
Computer Vision at the Edge with Grove Vision AI Module V2



In this tutorial, we will explore computer vision (CV) applications using the Seeed Studio *Grove Vision AI Module V2*, a powerful but still tiny device specially developed for applications in embedded machine learning. Based on the Himax WiseEye2 chip, this module is designed to enable AI capabilities on edge devices, making it a perfect tool for Edge Machine Learning (ML) applications.

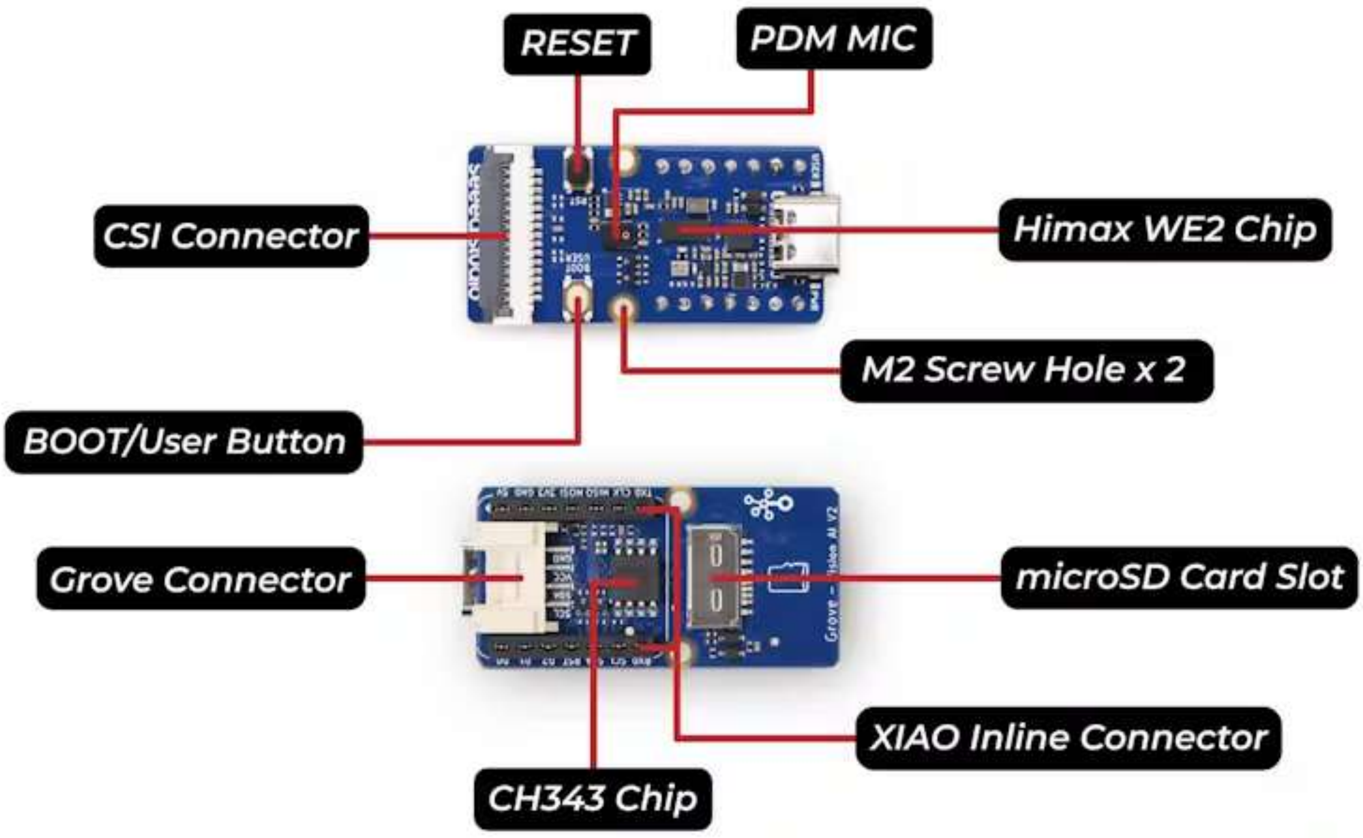
Marcelo Rovai @ February, 2024

1. Introduction

Grove Vision AI Module (V2) Overview

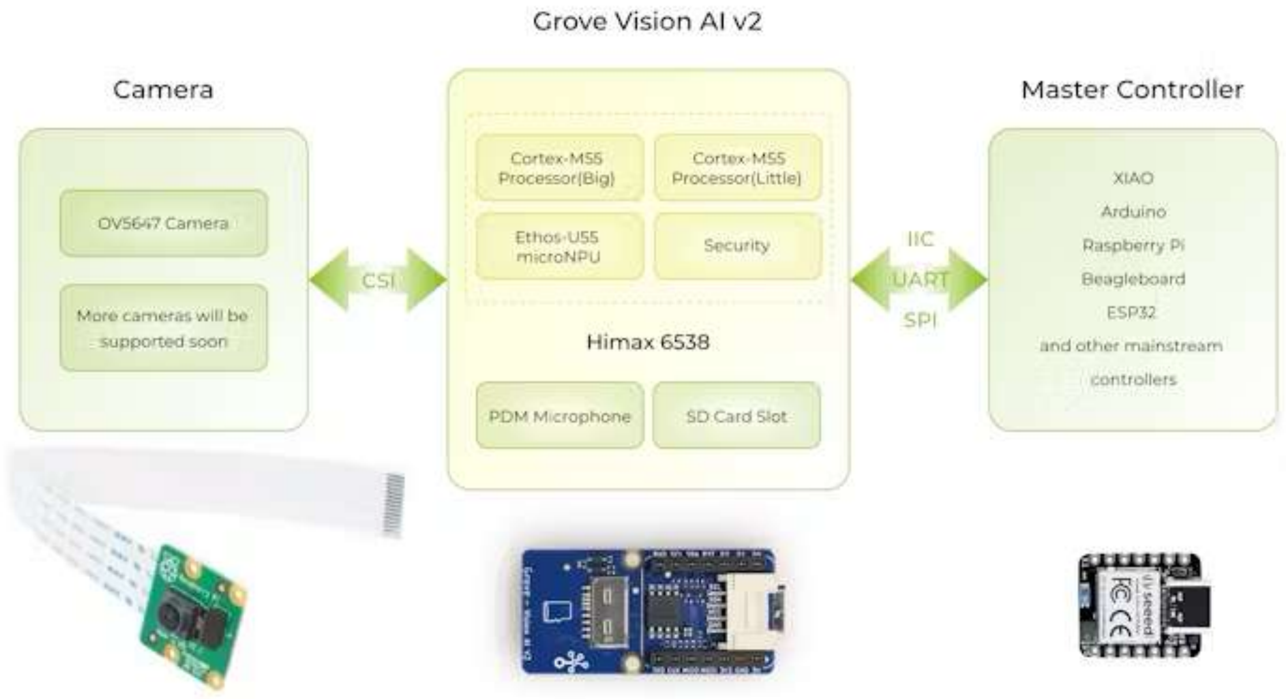
The Grove Vision AI (V2) is an MCU-based vision AI module that utilizes a Himax WiseEye2 HX6538 processor with a dual-core Arm Cortex-M55 and integrated ARM Ethos-U55 neural network unit. The Arm Ethos-U55 is a new machine learning (ML) processor class, called *microNPU*, specifically designed to accelerate 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 400Mhz, and its internal system memory (SRAM) is configurable, up to 2.4MB.

> Note: Besed on Seeed Studio documentation, besides the Himax internal memory of 2.5MB (2.4MB SRAM + 64KB ROM), the Grove Vision AI (V2) also is equiped with a 16MB/133MHz of external flash.



> In this article, [2024 MCU AI Vision Boards: Performance Comparison](), it is possible to confirm how powerful Grove Vision AI (V2) is. For example, when running the same application ([Face Detection - Swift-YOLO]()), the Grove Vision AI (V2) inference latency was 33ms versus 180ms on an XIAO ESP32S3 and on an Arduino Nicla Vision. So, very promising!

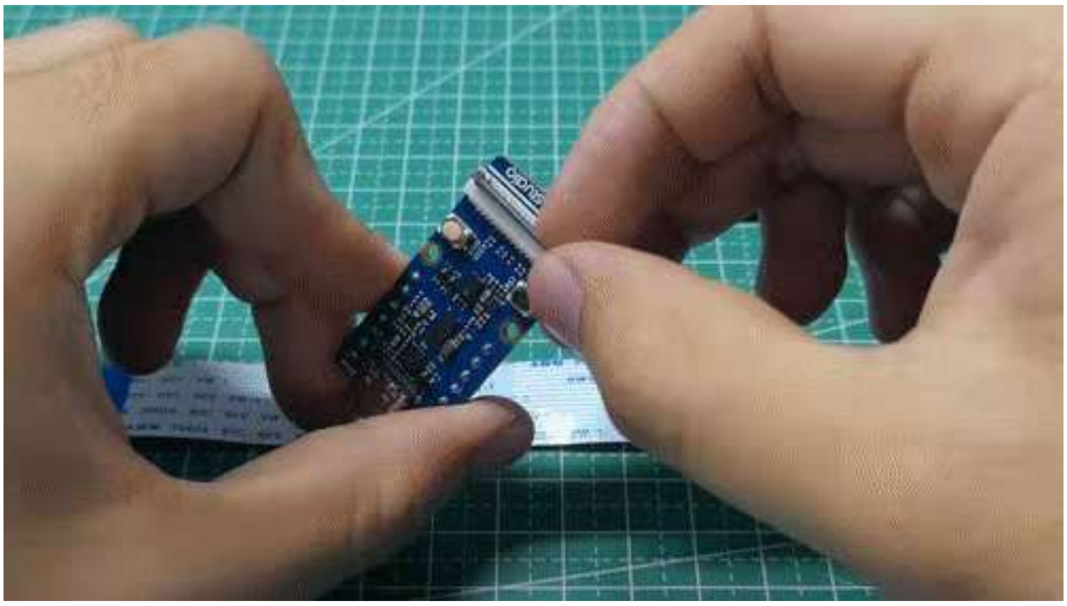
Below is a block Diagram of the Grove Vision AI (V2) system, including a camera and a master controller.



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 Arduino IDE or Micropython and conveniently connect to the cloud or dedicated servers like Home Assistance.

Camera Installation

Having the Grove Vision AI (V2) and camera ready, you can connect them via the CSI cable. When connecting, please pay attention to the direction of the row of pins, and don't plug them in backward.

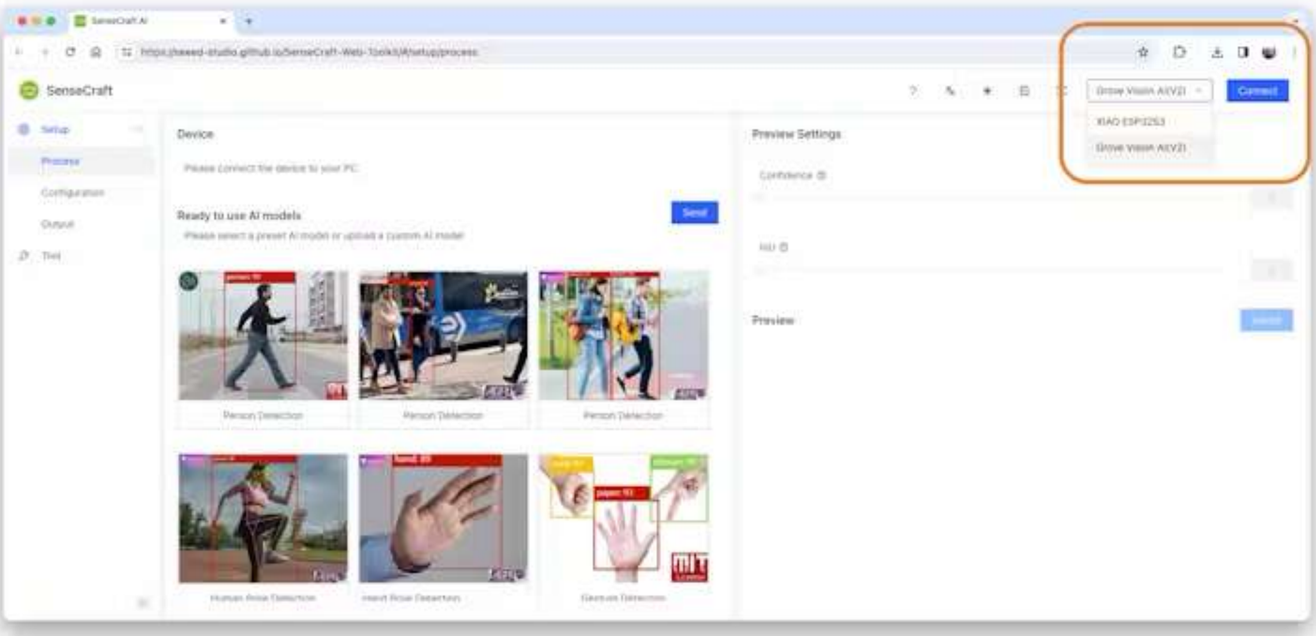


2. The SenseCraft Web-Toolkit

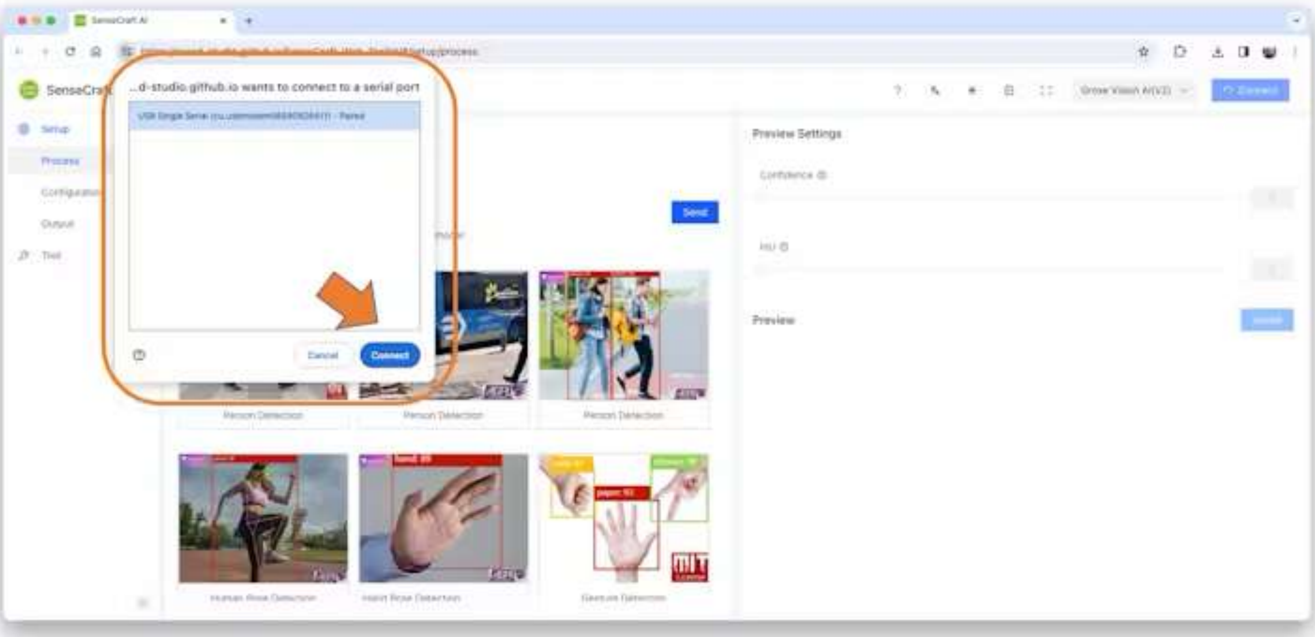
The SenseCraft Web Toolkit is a visual model deployment tool included in the SSCMA (Seeed SenseCraft Model Assistant). This tool allows you to deploy models to various platforms easily through simple operations. The tool offers a user-friendly interface and does not require any coding.

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

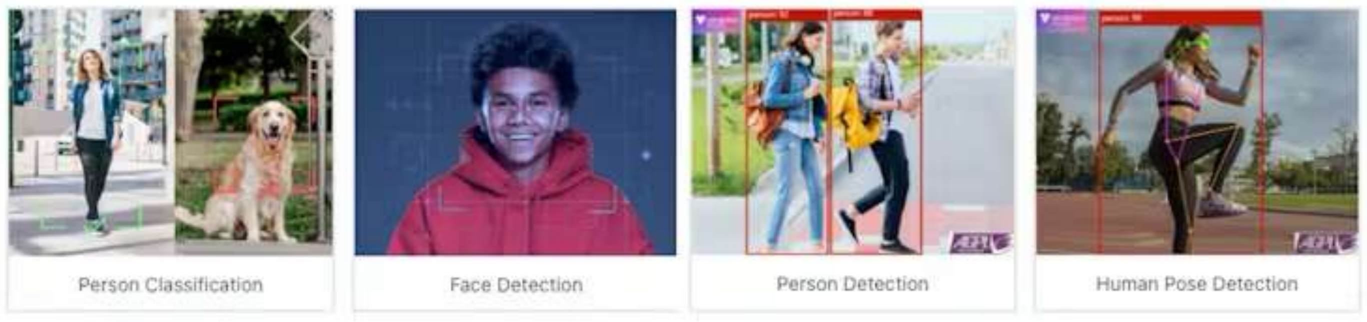
- Open the SenseCraft-Web-Toolkit website.
- Connect Grove Vision AI (V2) to your computer using a Type-C cable.
- Having the XIAO connected, select it as below:



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


...d-studio.github.io wants to connect to a serial port
USB Single Serial - Paired
Cancel
Connect

You can try several Computer Vision models previously uploaded by Seeed Studio. Passing the cursor over the AI models, you 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).



You can choose one of those ready-to-use AI models by clicking on it and pressing the [Send] button.

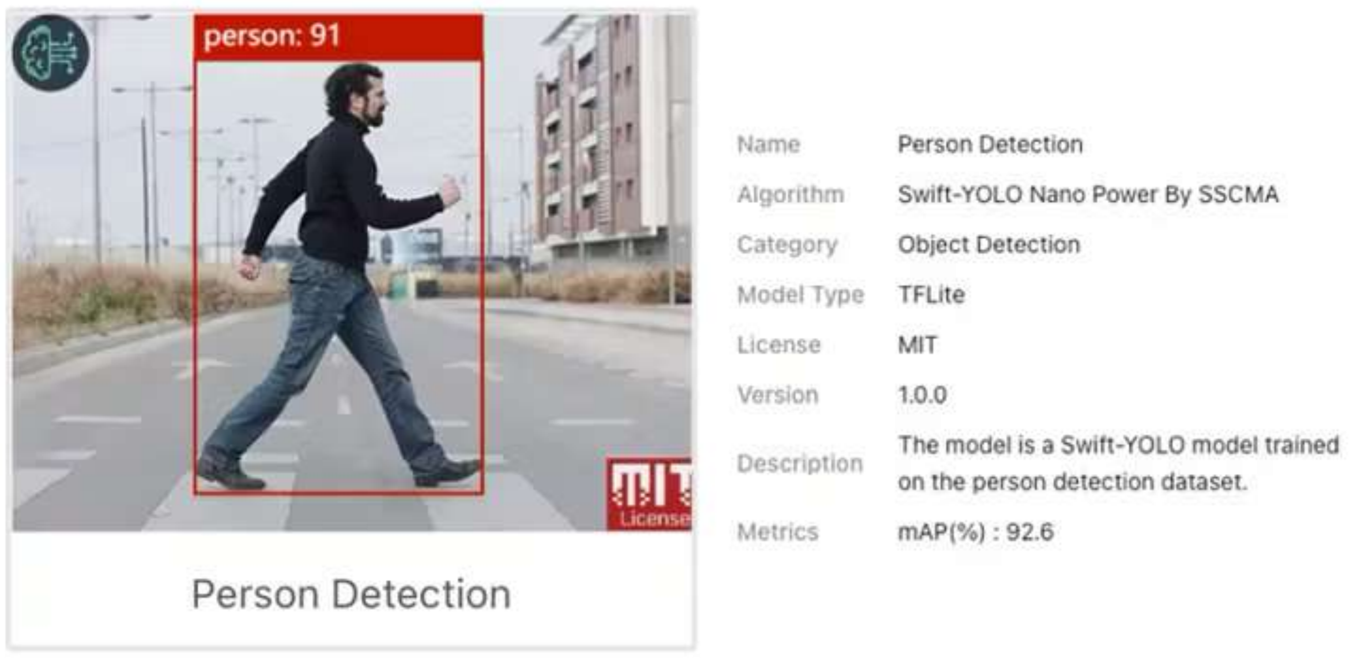
3. Exploring CV AI models

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.

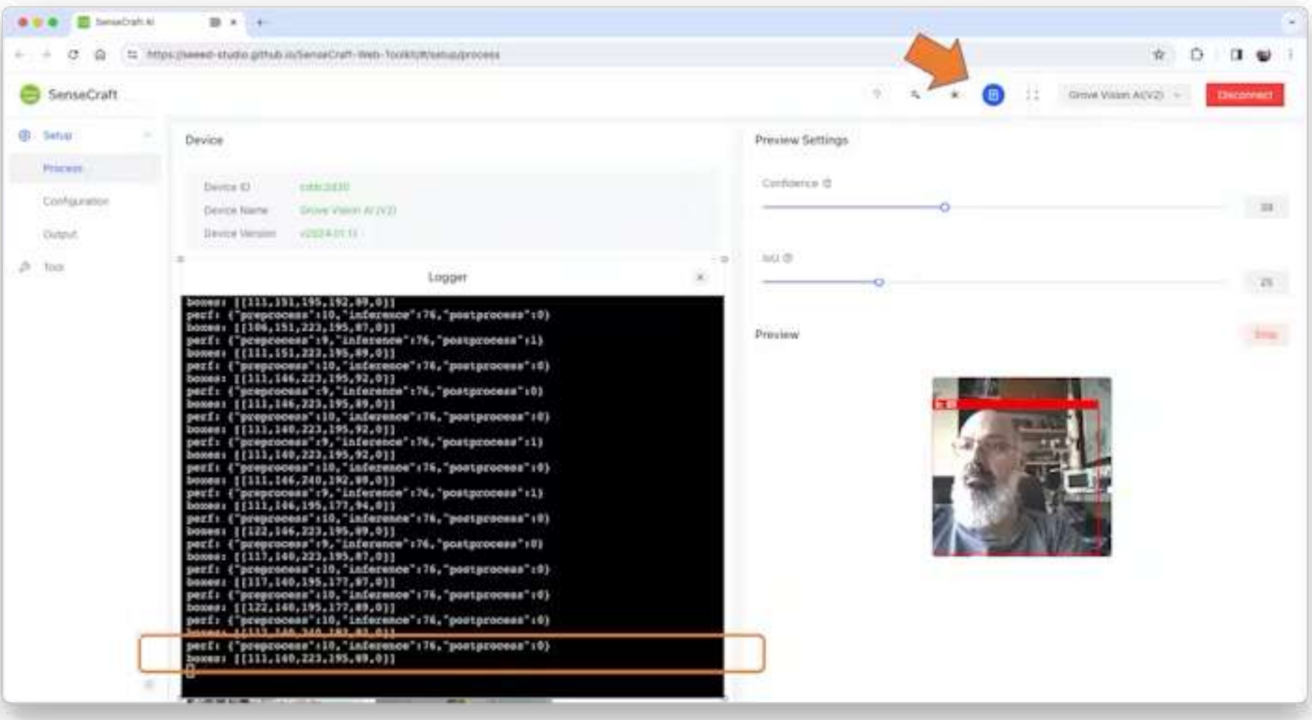
Common architectures that have set the benchmark in object detection include the YOLO (You Only Look Once), SSD (Single Shot MultiBox Detector), FOMO (Faster Objects, More Objects), and Faster R-CNN (Region-based Convolutional Neural Networks) models.

Let's choose one of the ready-to-use AI models, for example, Person Detection, which was trained with a 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

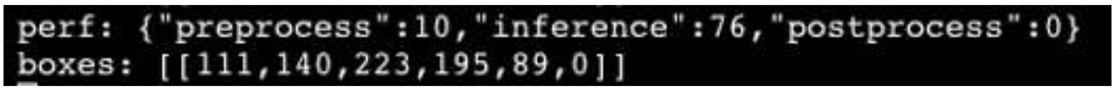
Once the model is uploaded successfully, you can see the live feed from the Grove Vision AI (V2) camera in the Preview area on the right. Also, the inference details can be shown on the Serial Monitor by clicking on the [Device Log] button at the top.



Device Name: Grove Vision AI (V2)

```
boxes: [[111,151,195,192,89,0]]
perf: {"preprocess":10,"inference":76,"postprocess":0}
boxes: [[106,151,223,195,87,0]]
perf: {"preprocess":9,"inference":76,"postprocess":1}
boxes: [[111,151,223,195,89,0]]
perf: {"preprocess":10,"inference":76,"postprocess":0}
boxes: [[111,146,223,195,92,0]]
perf: {"preprocess":9,"inference":76,"postprocess":0}
boxes: [[111,146,223,195,89,0]]
perf: {"preprocess":10,"inference":76,"postprocess":0}
boxes: [[111,140,223,195,92,0]]
perf: {"preprocess":9,"inference":76,"postprocess":1}
boxes: [[111,140,223,195,92,0]]
perf: {"preprocess":10,"inference":76,"postprocess":0}
boxes: [[111,146,240,192,89,0]]
perf: {"preprocess":9,"inference":76,"postprocess":1}
boxes: [[111,146,195,177,94,0]]
perf: {"preprocess":10,"inference":76,"postprocess":0}
boxes: [[122,146,223,195,89,0]]
perf: {"preprocess":9,"inference":76,"postprocess":0}
boxes: [[117,140,223,195,87,0]]
perf: {"preprocess":10,"inference":76,"postprocess":0}
boxes: [[117,140,195,177,87,0]]
perf: {"preprocess":10,"inference":76,"postprocess":0}
boxes: [[122,140,195,177,89,0]]
perf: {"preprocess":10,"inference":76,"postprocess":0}
boxes: [[117,140,240,192,92,0]]
perf: {"preprocess":10,"inference":76,"postprocess":0}
boxes: [[111,140,223,195,89,0]]
```

Point the camera on me; only one person was detected so that the model output will be only one "box." Looking in detail, the module sends continuously 2 lines of information:



```
perf: {"preprocess":10,"inference":76,"postprocess":0}
boxes: [[111,140,223,195,89,0]]
```

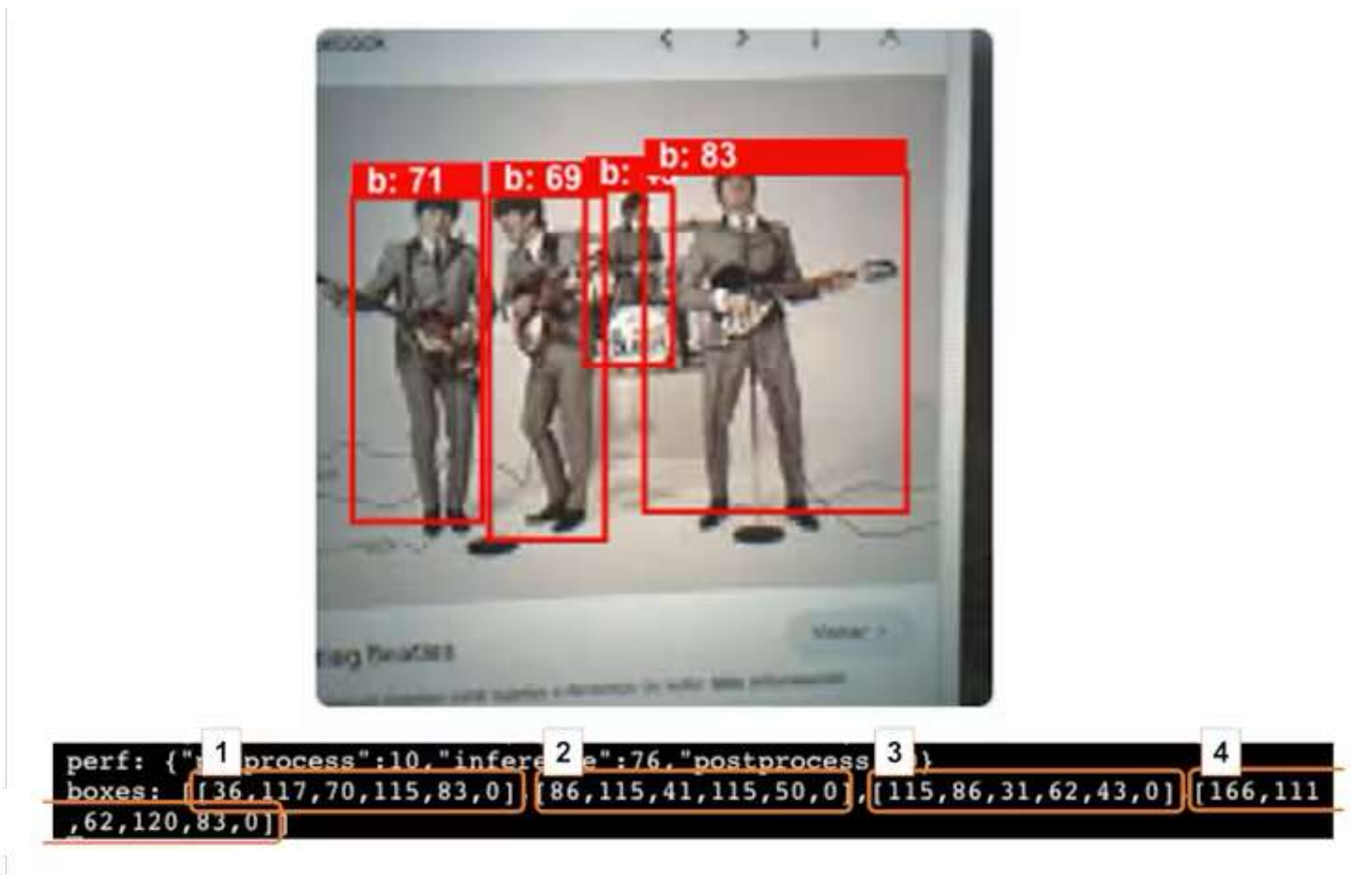
perf (Performance), which shows the latency in milliseconds.

- Preprocess time (image capture and Crop): 10ms;
- 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 on the image. In this case, only one.

- The box has the x, y, w, and h coordinates of (**111, 140**), (**223, 195**), and the object (person, label **0**) was captured with a value of.**89**.

If we point the camera for an image with several people, we will get one box for each person (object):

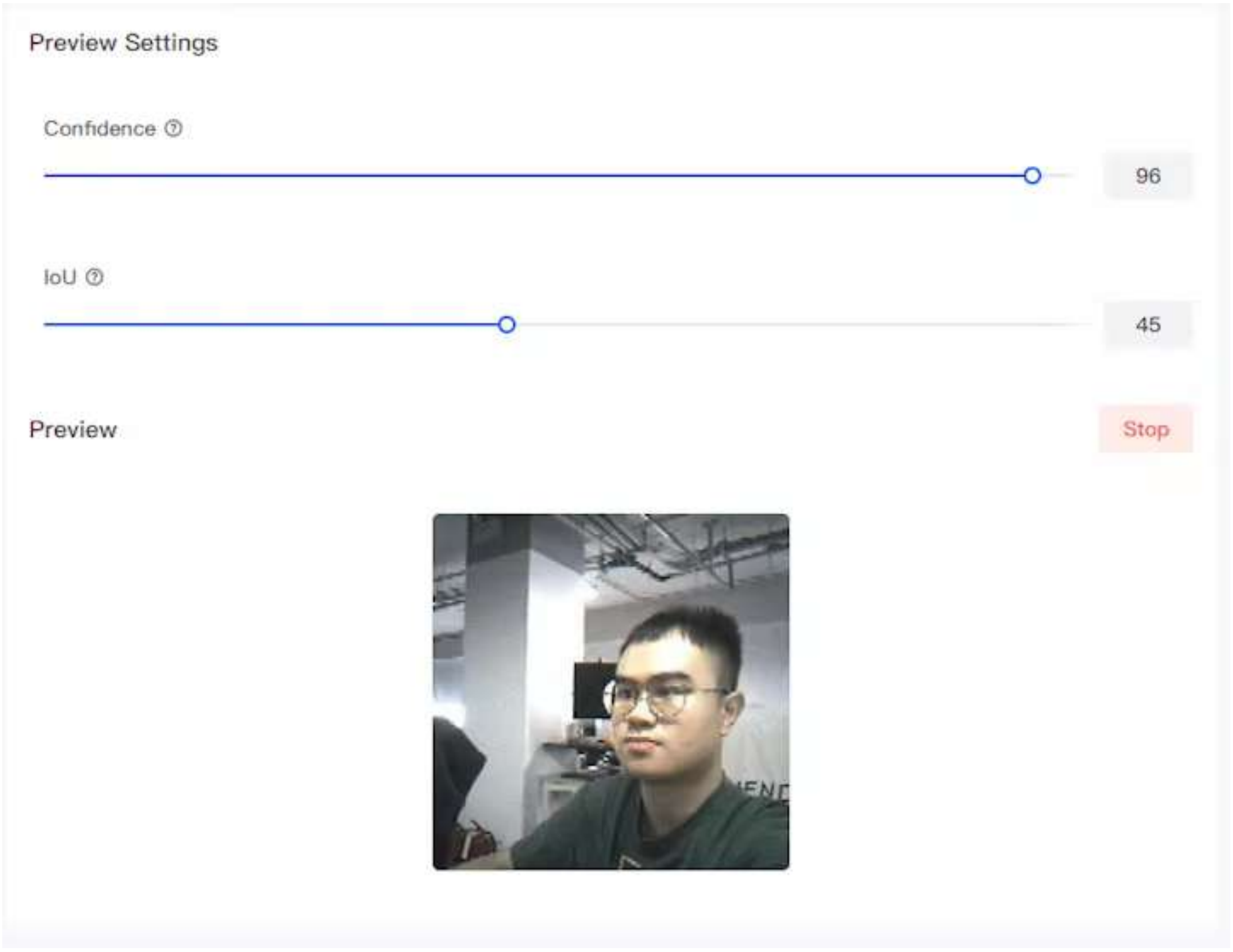


```
perf: {"[illegible]process":10,"infere[illegible]e":76,"postprocess[illegible]}
boxes: [[36,117,70,115,83,0],[86,115,41,115,50,0],[115,86,31,62,43,0],[166,111
,62,120,83,0]]
```

Preview Settings

We can see that in the Preview Settings on the right-hand side, two setting options can be changed to optimize the recognition accuracy of the model.

- **Confidence:** refers to the level of certainty or probability assigned to its predictions by a model.



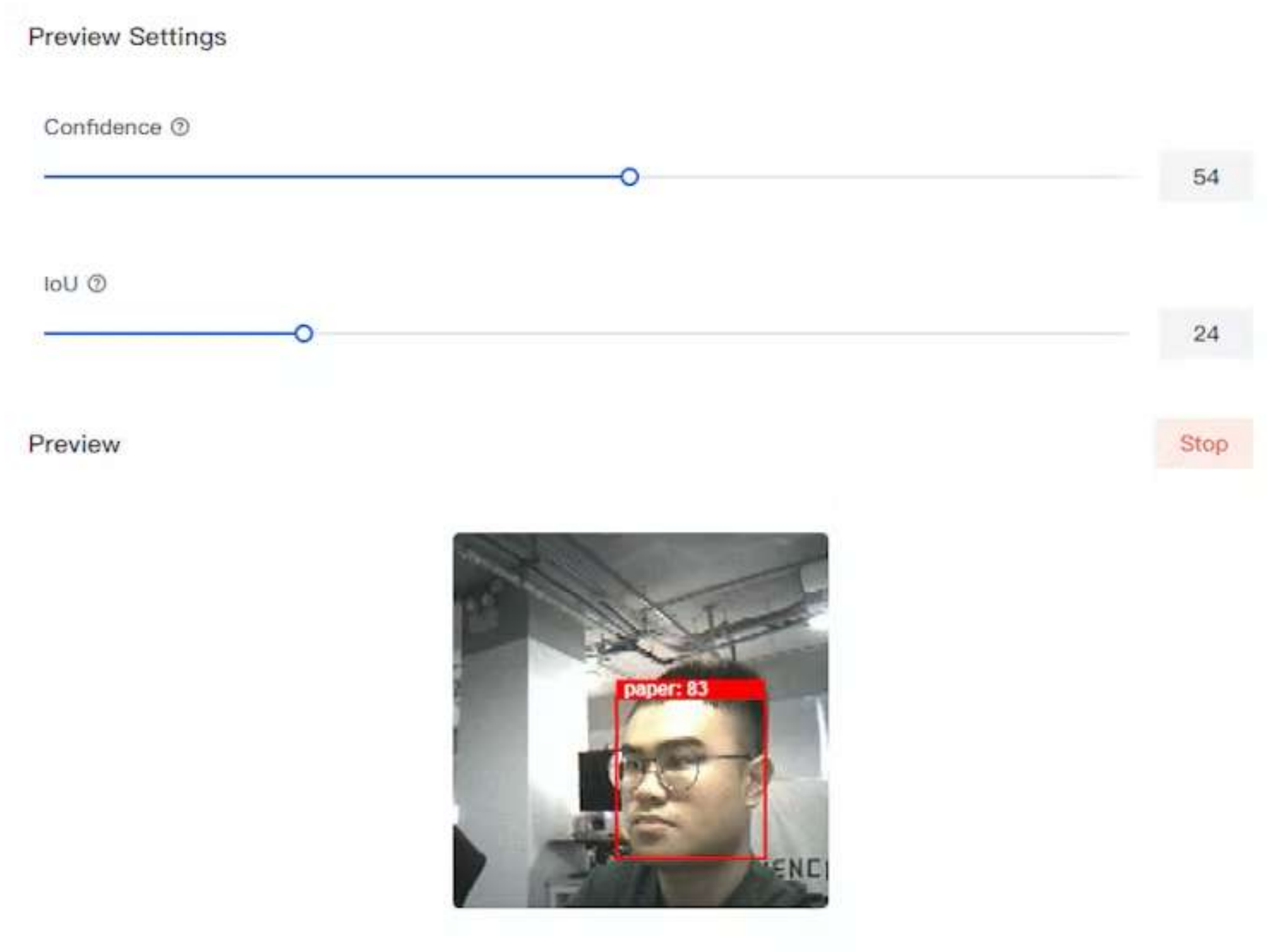
Preview Settings

Confidence: 96

IoU: 45

Preview

- **IoU:** used to assess the accuracy of predicted bounding boxes compared to truth bounding boxes.


Preview Settings

Confidence 54

IoU 24

Pose/Keypoint Detection

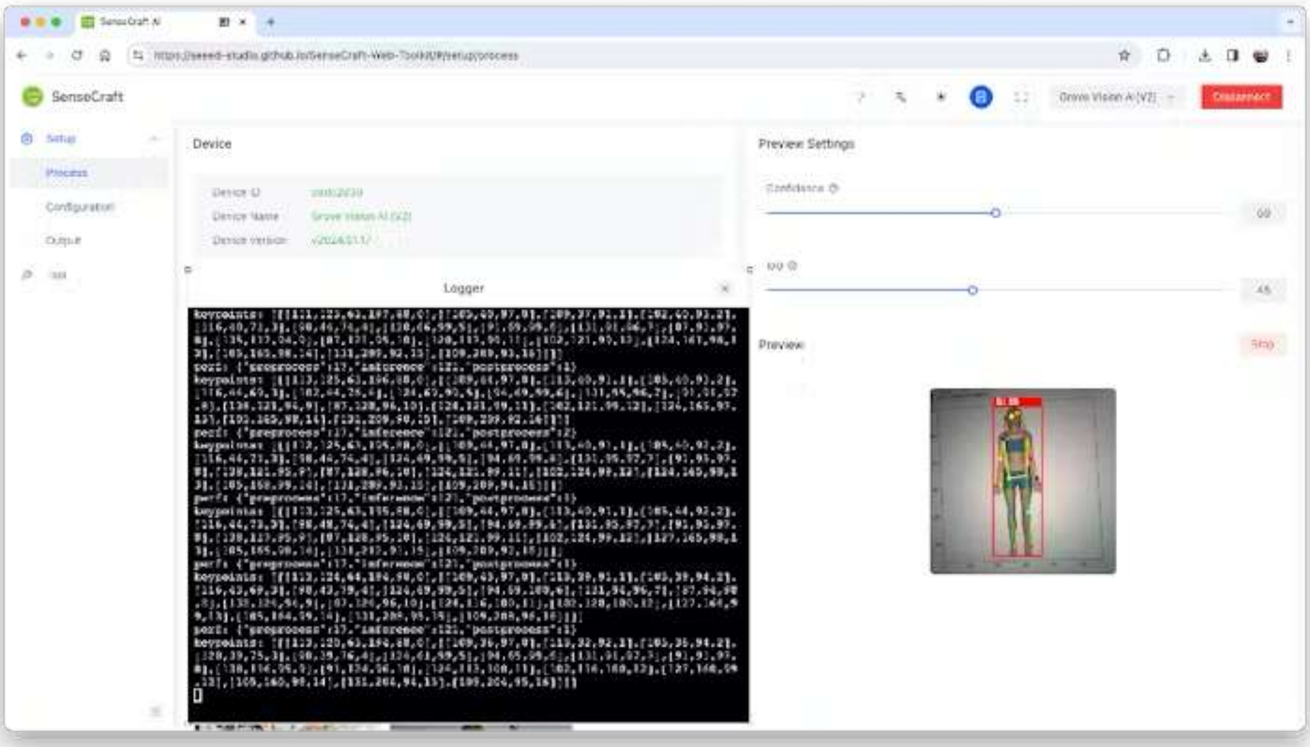
Pose or keypoint detection is a sophisticated area within computer vision that focuses on identifying specific points of interest within an image or video frame, often related to human bodies, faces, or other objects. This technology can detect and map out the various keypoints of a subject, such as 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 many applications, ranging from augmented reality and human-computer interaction to 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

When 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 and the inference details shown on the Serial Monitor (by clicking on the [Device Log] button at the top).



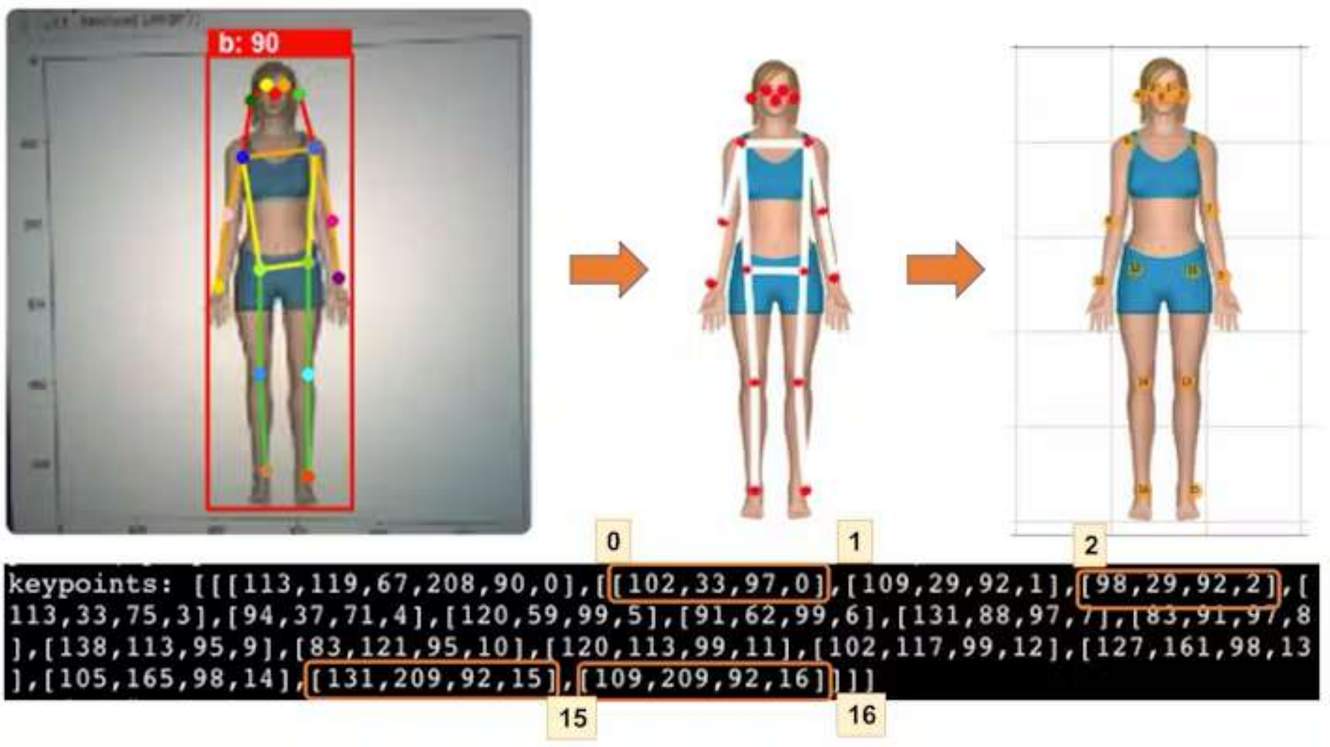
```
keypoints: [[[111,123,62,197,88,0],[[109,40,97,0],[109,37,91,1],[102,40,92,2],[116,40,71,3],[98,46,74,4],[120,46,99,5],[91,49,99,6],[131,91,96,7],[87,91,97,8],[135,117,94,9],[87,121,95,10],[120,117,90,11],[102,121,95,12],[124,161,98,13],[105,165,98,14],[131,209,92,15],[109,209,93,16]]]]
perf: {"preprocess":17,"inference":121,"postprocess":1}
keypoints: [illegible]
perf: {"preprocess":17,"inference":121,"postprocess":2}
keypoints: [illegible]
perf: {"preprocess":17,"inference":121,"postprocess":1}
keypoints: [illegible]
perf: {"preprocess":17,"inference":121,"postprocess":1}
keypoints: [illegible]
perf: {"preprocess":17,"inference":121,"postprocess":1}
keypoints: [illegible]
```

This YOLOV8 Pose model was trained using the COCO-Pose Dataset, which contains 200K images labeled with **17** keypoints for pose estimation tasks.

If we look at a single screenshot of the inference, we can note that we have 2 lines, one with the inference **performance** in milliseconds 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]]]]
```

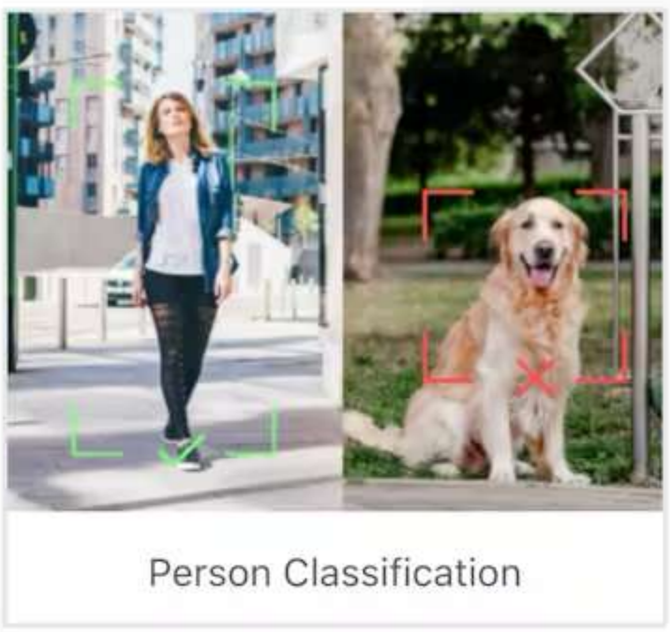
> To understand a pose estimation project more deeply, read this tutorial: Exploring AI at the Edge! - Pose Estimation.

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.

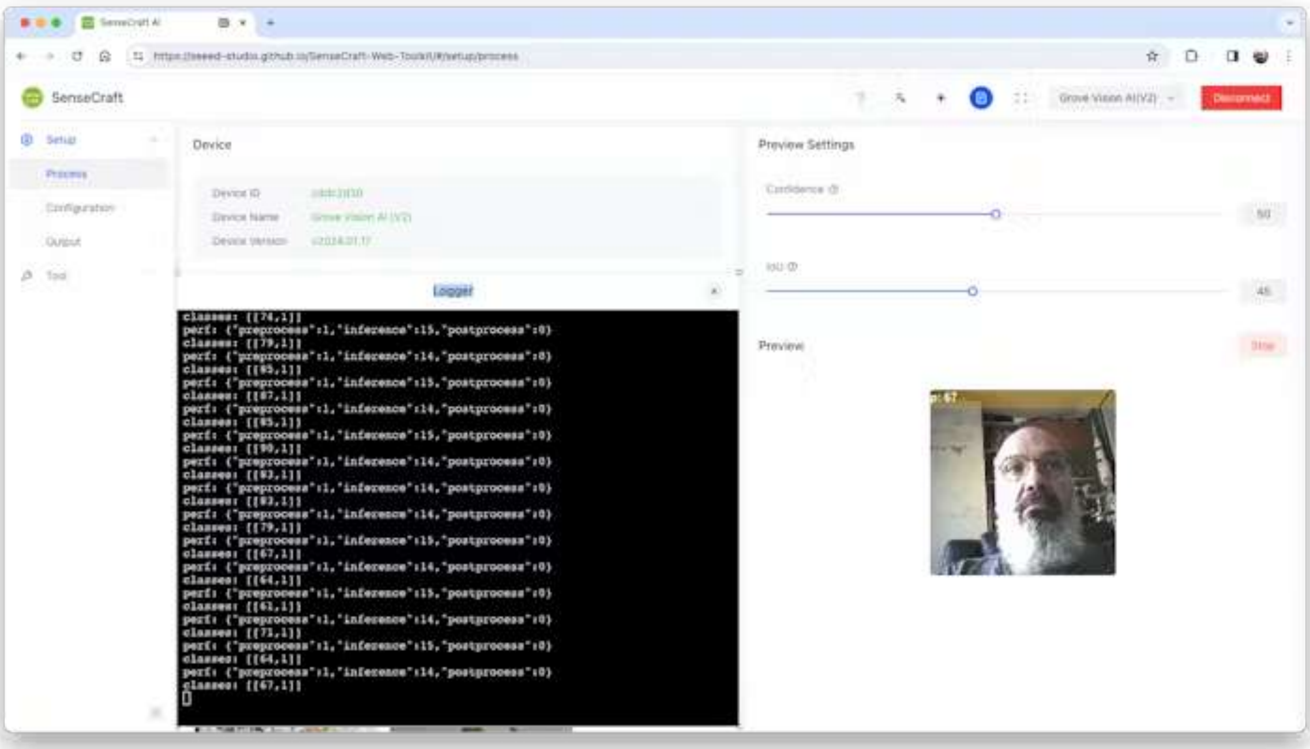
Image classification is critical in numerous applications, from organizing and searching through large databases of images in digital libraries and social media platforms to enabling autonomous systems to understand their surroundings. Common architectures that have significantly advanced the field of 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

After 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 and the inference details shown on the Serial Monitor (by clicking on the [Device Log] button at the top).



Device ID: [illegible]
Device Name: Grove Vision AI (V2)
Device Version: v2024.01.17

```
classes: [[74,1]]
perf: {"preprocess":1,"inference":15,"postprocess":0}
classes: [[79,1]]
perf: {"preprocess":1,"inference":14,"postprocess":0}
classes: [[85,1]]
perf: {"preprocess":1,"inference":15,"postprocess":0}
classes: [[87,1]]
perf: {"preprocess":1,"inference":14,"postprocess":0}
classes: [[85,1]]
perf: {"preprocess":1,"inference":15,"postprocess":0}
classes: [[90,1]]
perf: {"preprocess":1,"inference":14,"postprocess":0}
classes: [[83,1]]
perf: {"preprocess":1,"inference":14,"postprocess":0}
classes: [[83,1]]
perf: {"preprocess":1,"inference":14,"postprocess":0}
classes: [[79,1]]
perf: {"preprocess":1,"inference":15,"postprocess":0}
classes: [[67,1]]
perf: {"preprocess":1,"inference":14,"postprocess":0}
classes: [[64,1]]
perf: {"preprocess":1,"inference":15,"postprocess":0}
classes: [[61,1]]
perf: {"preprocess":1,"inference":14,"postprocess":0}
classes: [[71,1]]
perf: {"preprocess":1,"inference":15,"postprocess":0}
classes: [[64,1]]
perf: {"preprocess":1,"inference":14,"postprocess":0}
classes: [[67,1]]
```

The result is straight full, and we will receive a score and the class as output.

```
perf: {"preprocess":1,"inference":14,"postprocess":0}
classes: [[67,1]]
```

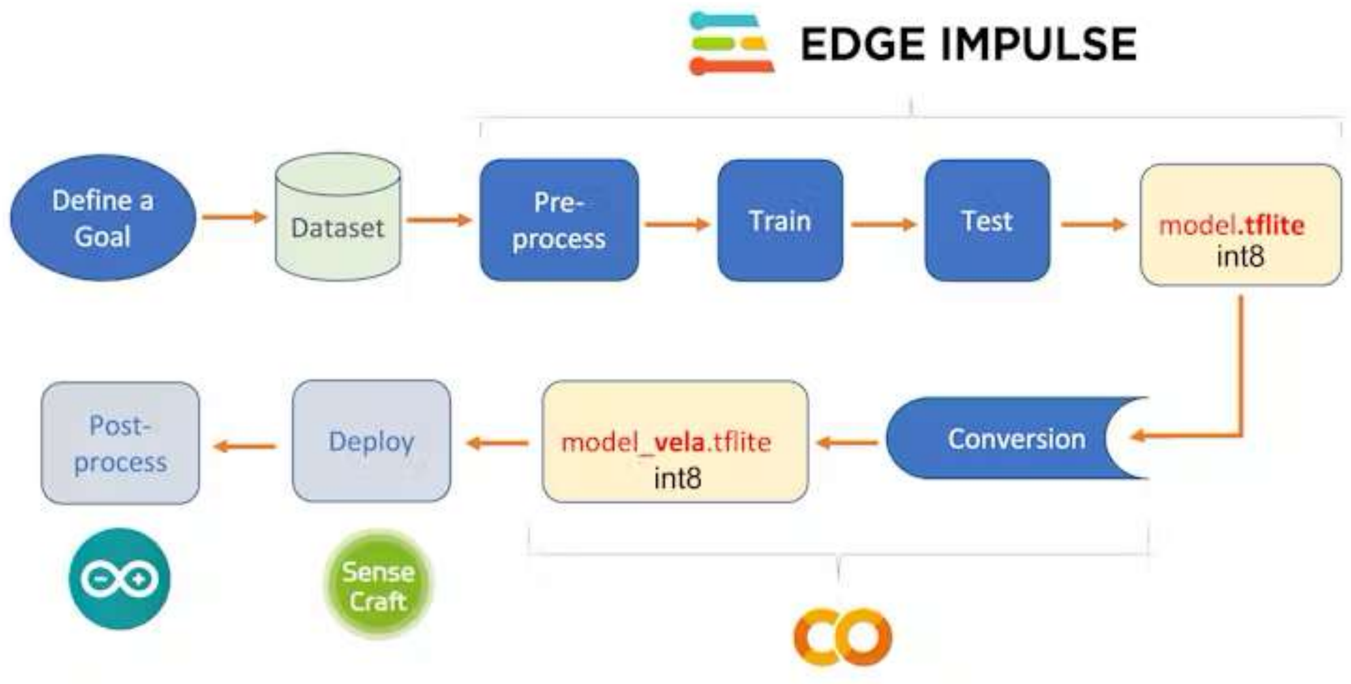
For example, [67, 1] means class: 1 (Person) with a score of 0.67. Once this model is a binary classification, class 0 will be "No Person" (or Background). The Inference latency is 14ms or 70fps.

4. An Image Classification Project from Scratch

Introduction

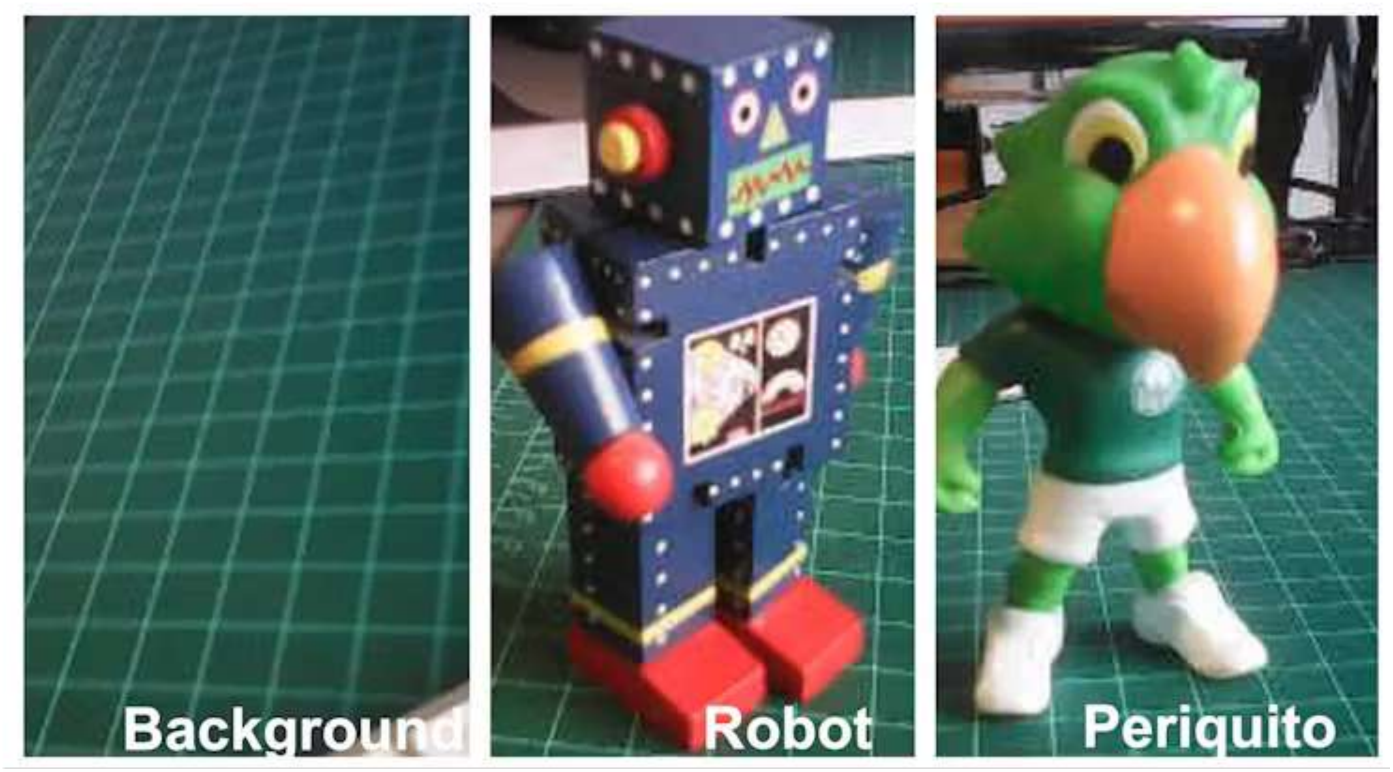
So far, we have explored several computer vision models previously uploaded by Seeed Studio. Let's develop our CV project from scratch, starting with image classification.

Below, we can see the project's main steps and where we will do it:



Project Goal

The first step in any ML project is the **goal definition**. In this case, it is to detect and classify two specific objects present in one 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. If your project uses images that can be found on public datasets, you can download them to be used in the project, as you can see in this tutorial: TinyML Made Easy: Image Classification w/ XIAO ESP32S3 Sense.

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 image. For example, in this tutorial, TinyML Made Easy: Object Detection with XIAO ESP32S3 Sense - Data Collection, an XIAO captures and saves the images in a computer.

Uploading the dataset to the 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 or clone this one: Periquito vs Robot - Img Class.

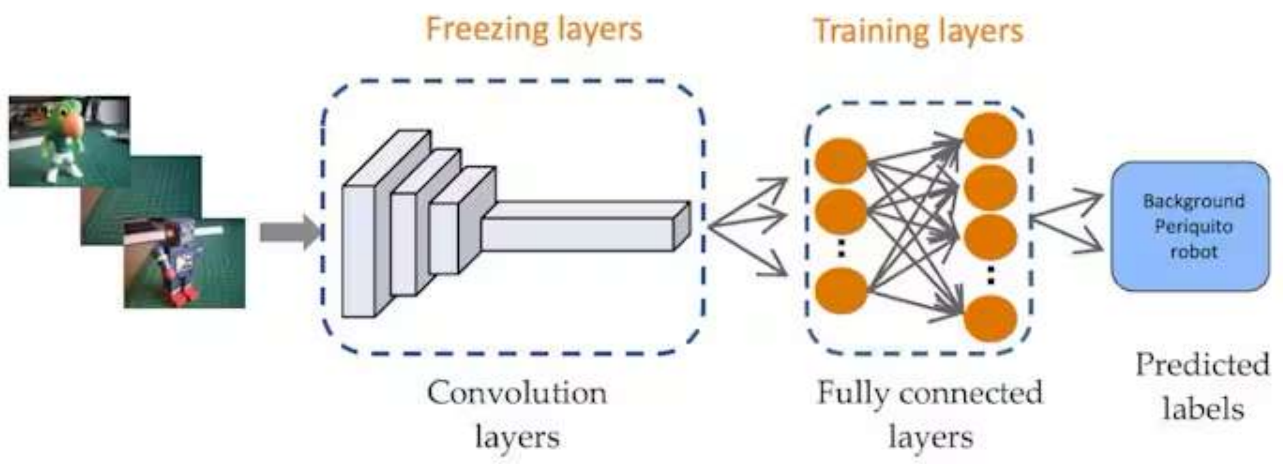
> The dataset has around 50 images per label (40 for training and 10 for testing).

Impulse Design and Pre-Processing

Impulse Design

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

Classifying images is the most common use of deep learning, but much data should be used 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" to differentiate each class. But, we can solve this issue by re-training a previously trained model with thousands of images. We call this technique "Transfer Learning" (TL). 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 (96x96) pixels and feed them to our Transfer Learning block:

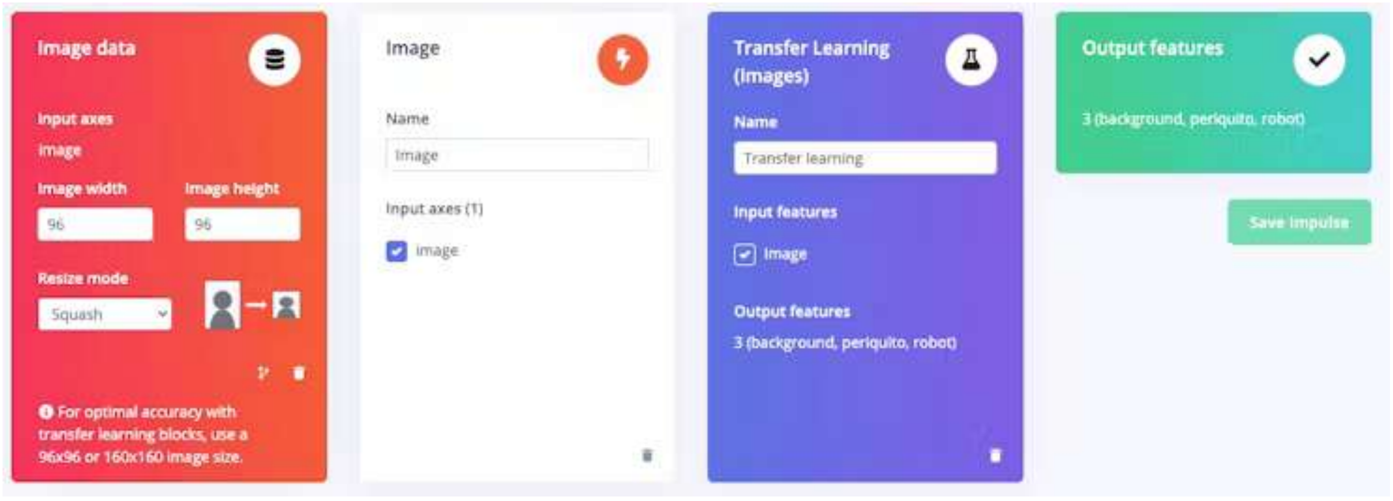


Image data
Input axes
image
Image width: 96
Image height: 96
Resize mode: Squash
For optimal accuracy with transfer learning blocks, use a 96x96 or 160x160 image size.

Image
Name: Image
Input axes (1)
✓ image

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

Output features
3 (background, periquito, robot)

Save Impulse

Pre-processing (Feature generation)

Besides resizing the images, we can change them to grayscale or keep the actual RGB color depth. Let's start selecting [RGB] in the Image section. Doing that, each data sample will have a dimension of 27, 648 features (96x96x3).

Model Design, Training, and Test

In 2007, Google introduced MobileNetV1 and, in 2018, launched MobileNetV2: Inverted Residuals and Linear Bottlenecks, a family of general-purpose computer vision neural networks designed with mobile devices in mind to support classification, detection, and more. 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 tiny and has low latency, many times, a specific use case or application may require the model to be smaller and faster. MobileNets introduces a straightforward parameter α (alpha) called width multiplier to construct these smaller, less computationally expensive models. The role of the width multiplier α 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 **α** values (from 0.05 to 1.0). For example, you will get the highest accuracy with V2, 160x160 images, and α=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 α=0.10 (around 53.2K RAM and 101K ROM).

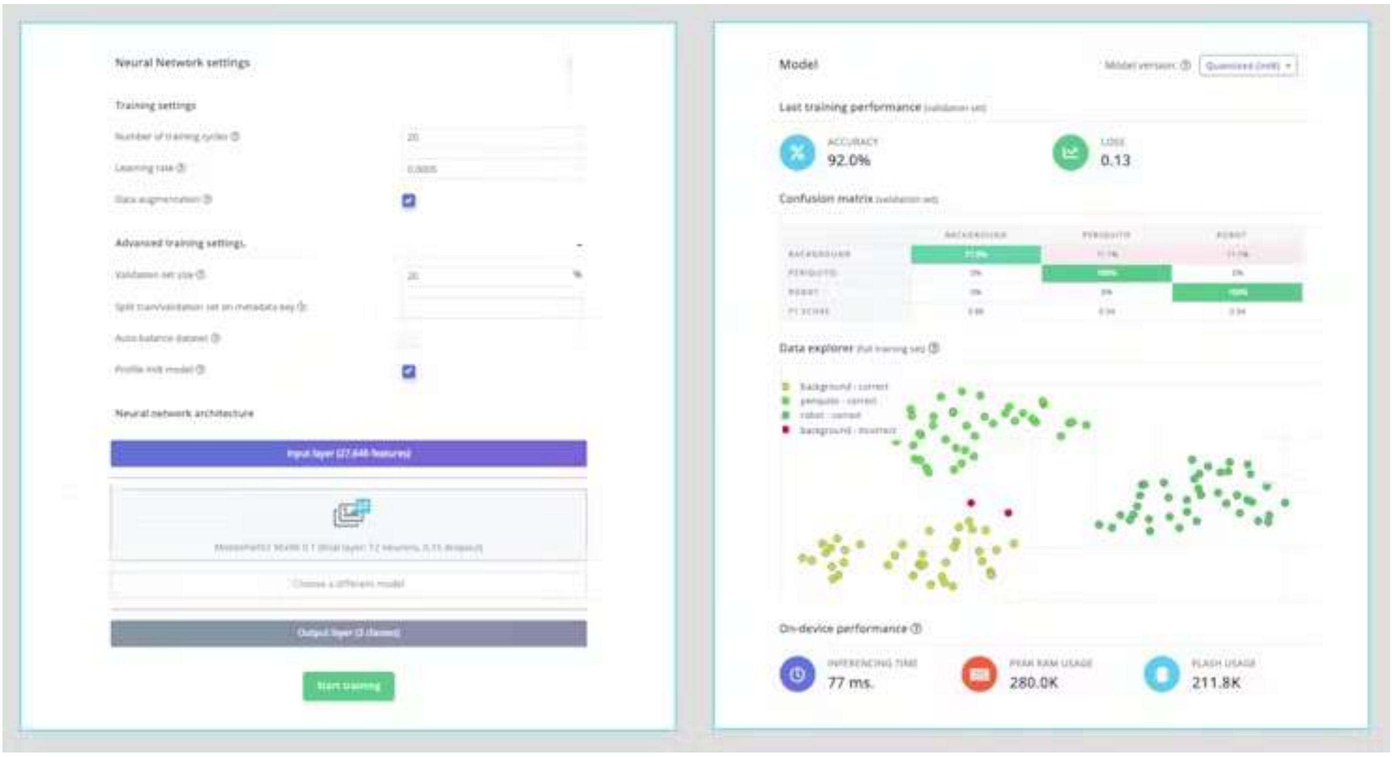
> For our project, we will use, as a base model, the **MobileNet V2 0.35**. The final layer of our model (before the output layer) will have 12 neurons with a 15% dropout for overfitting prevention.

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, 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:



Neural Network settings

Training settings

Number of training cycles	20
Learning rate	0.0005
Data augmentation	✓

Advanced training settings

Validation set size	20 %
Split train/validation set on metadata key	
Auto-balance dataset	
Profile int8 model	✓

Neural network architecture

Model

Model version: Quantized (int8)

Last training performance (validation set)

ACCURACY 92.0%

LOSS 0.13

Confusion matrix (validation set)

	BACKGROUND	PERIQUITO	ROBOT
BACKGROUND	[illegible]	[illegible]	[illegible]
PERIQUITO	0%	100%	0%
ROBOT	0%	0%	100%
F1 SCORE	[illegible]	[illegible]	[illegible]

Data explorer (full training set)

On-device performance

INFERENCING TIME 77 ms.

PEAK RAM USAGE 280.0K

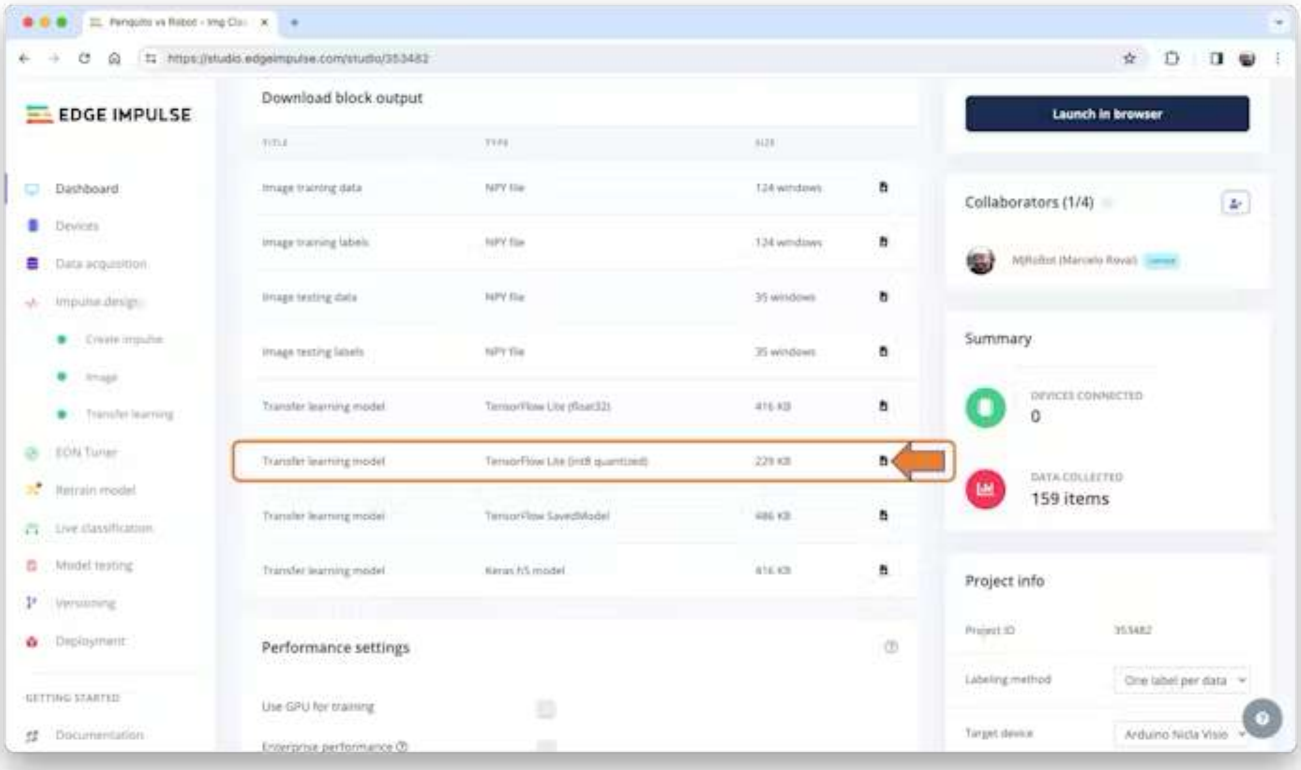
FLASH USAGE 211.8K

The model profile predicts 289KB of RAM and 212KB of Flash, indicating no problem with the Grove AI Vision (V2), with its almost 2.5MB of internal SRAM. Also, the Studio indicates 77ms of latency, based in this case on an Arduino Nicla Vision with a Cortex-M7 running at 480Mhz. As we saw in the last section, the latency with the Person Classification model (binary) was only 15ms, 5 times less than the prediction here.

> Using the spared data for testing, we confirmed an excellent result: Accuraccy of 94%.

Model Conversion

On the Dashboard section, download the model ("block output"): Transfer learning model - TensorFlow Lite (int8 quantized).


Download block output

TITLE	TYPE	SIZE
Image training data	NPY file	124 windows
Image training labels	NPY file	124 windows
Image testing data	NPY file	35 windows
Image testing labels	NPY file	35 windows
Transfer learning model	TensorFlow Lite (float32)	416 KB
Transfer learning model	TensorFlow Lite (int8 quantized)	229 KB
Transfer learning model	TensorFlow SavedModel	486 KB
Transfer learning model	Keras h5 model	416 KB

Performance settings

Use GPU for training

Summary

DEVICES CONNECTED 0

DATA COLLECTED 159 items

Project info

Project ID 353482

Labeling method One label per data

Target device Arduino Nicla Vision

With this, you will find on your computer the following file:

```
ei-periquito-vs-robot---img-class-transfer-learning-tensorflow-lite-int8-quantized-
model.lite
```

Let's rename it, simplify the name, and, more importantly, change the extension.lite to **.tflite**.

```
ei-periquito-vs-robot-int8.tflite
```

Now, we will use Vela, a tool to compile our quantized int8 TensorFlow Lite neural network model (ei-periquito-vs-robot-int8.tflite) into an optimized version that can run on an embedded system containing an Arm Ethos-U NPU (ei-periquito-vs-robot-int8_vela.tflite).

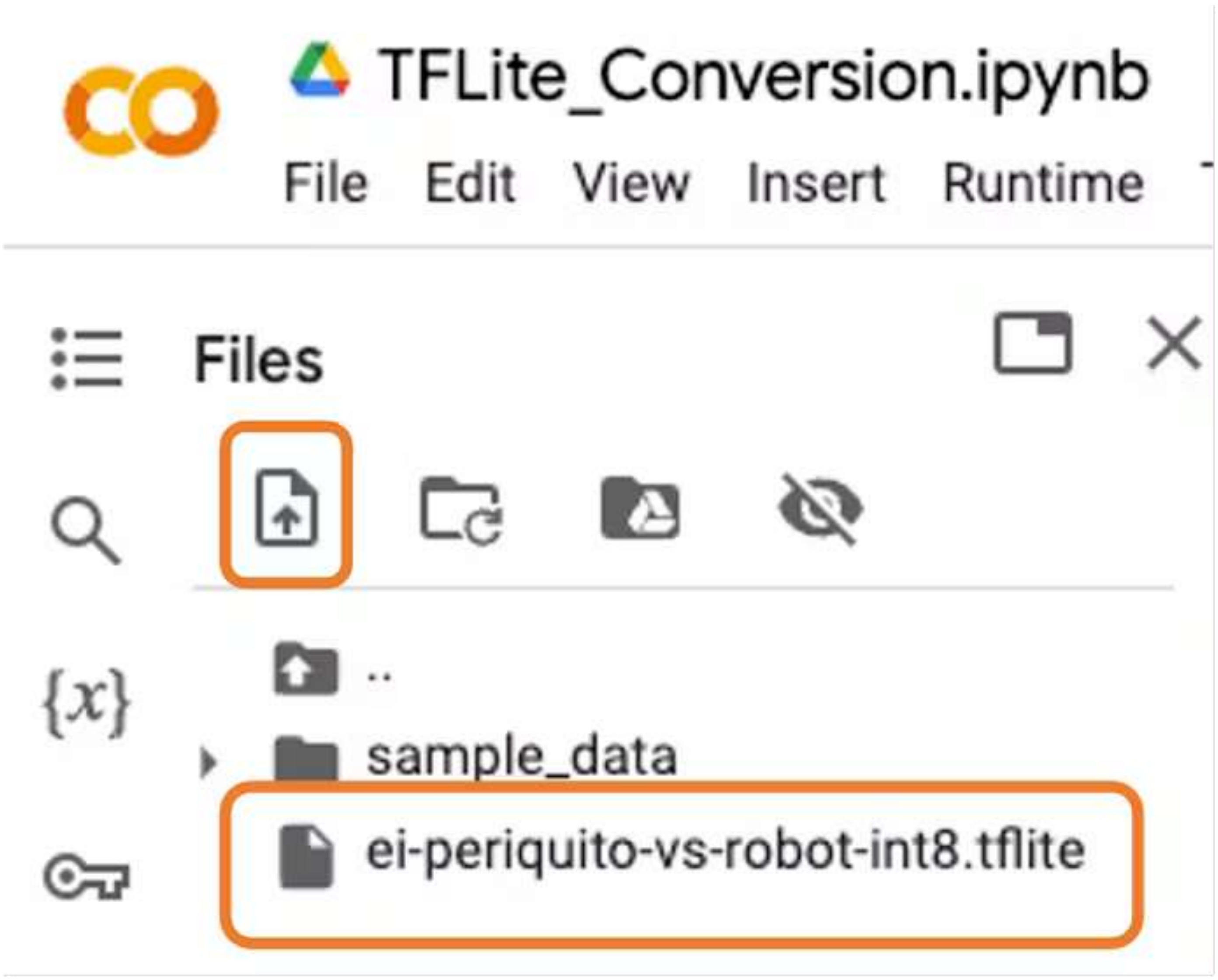
The optimized model will contain TensorFlow Lite Custom operators for those parts of the model that the Ethos-U NPU can accelerate. Parts of the model that cannot be accelerated are left unchanged and will run on the Cortex-M series CPU using an appropriate kernel (such as the Arm optimized CMSIS-NN kernels). After compilation, the optimized model can only be run on an Ethos-U NPU embedded system, such as the Grove AI Vision (V2).

On a Google CoLab, run the lines below:

```
!pip install ethos-u-vela
!!vela --version
```

We should get: 3.10.0

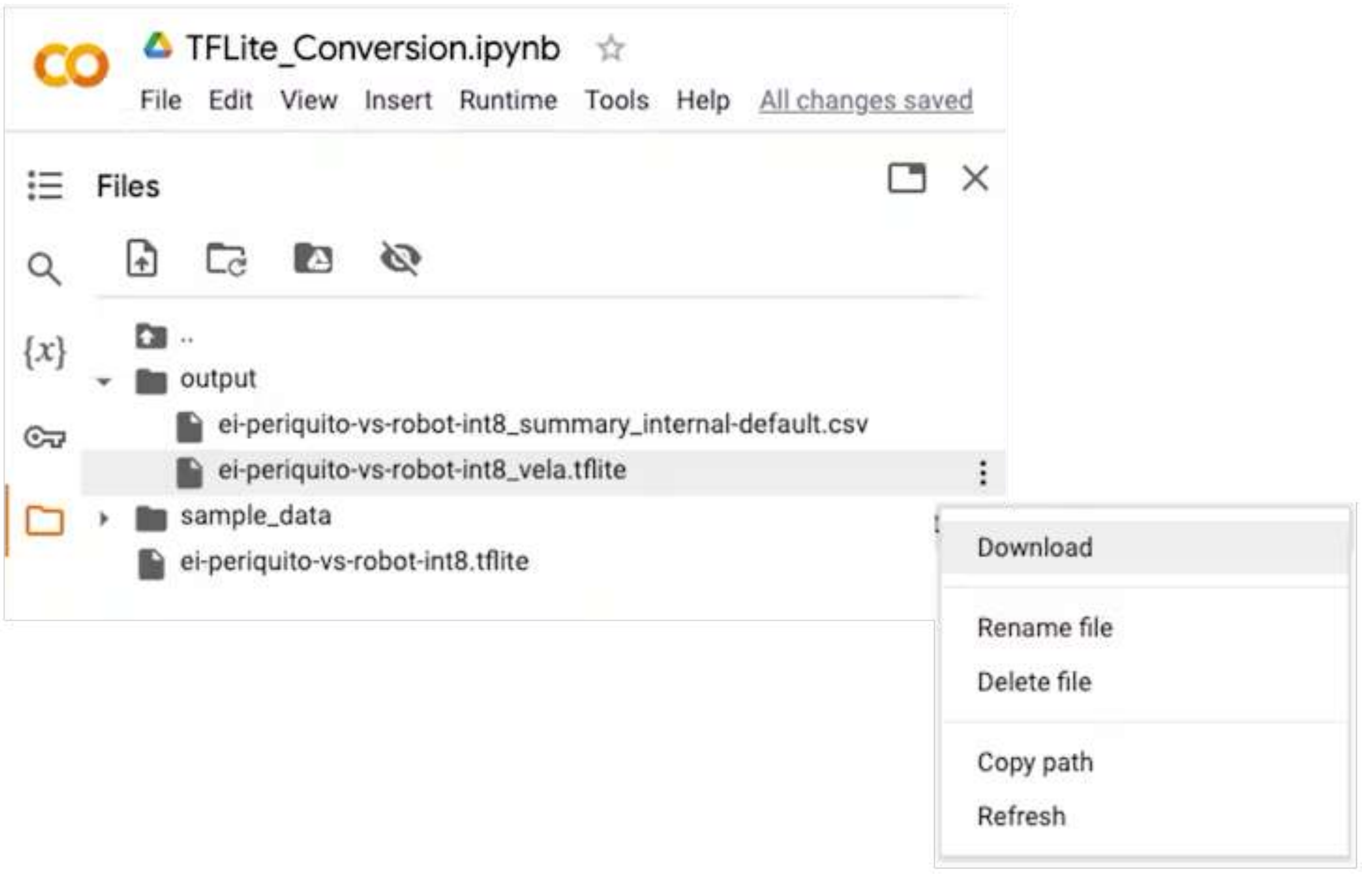
Next, upload the.tflite model to the Colab:



And run the line below:

```
!vela ei-periquito-vs-robot-int8.tflite --accelerator-config ethos-u55-64
```

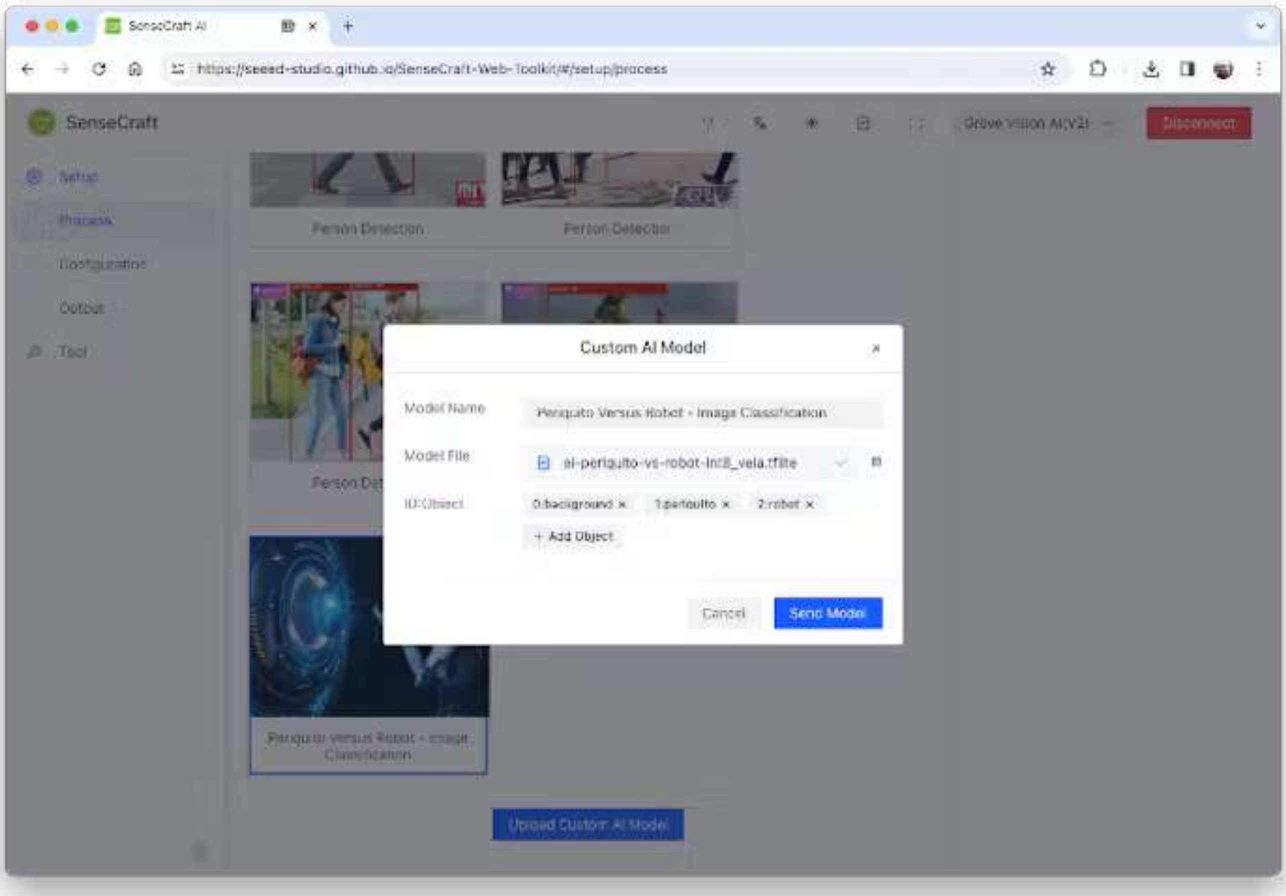
A new folder should be created (output). Open it and download the converted model (ei-periquito-vs-robot-int8_vela.tflite):


TFLite_Conversion.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Files
..
output
ei-periquito-vs-robot-int8_summary_internal-default.csv
ei-periquito-vs-robot-int8_vela.tflite
sample_data
ei-periquito-vs-robot-int8.tflite
Download
Rename file
Delete file
Copy path
Refresh

The CoLab notebook can be found at the Project repository

Deploy model on the SenseCraft Web-Toolkit

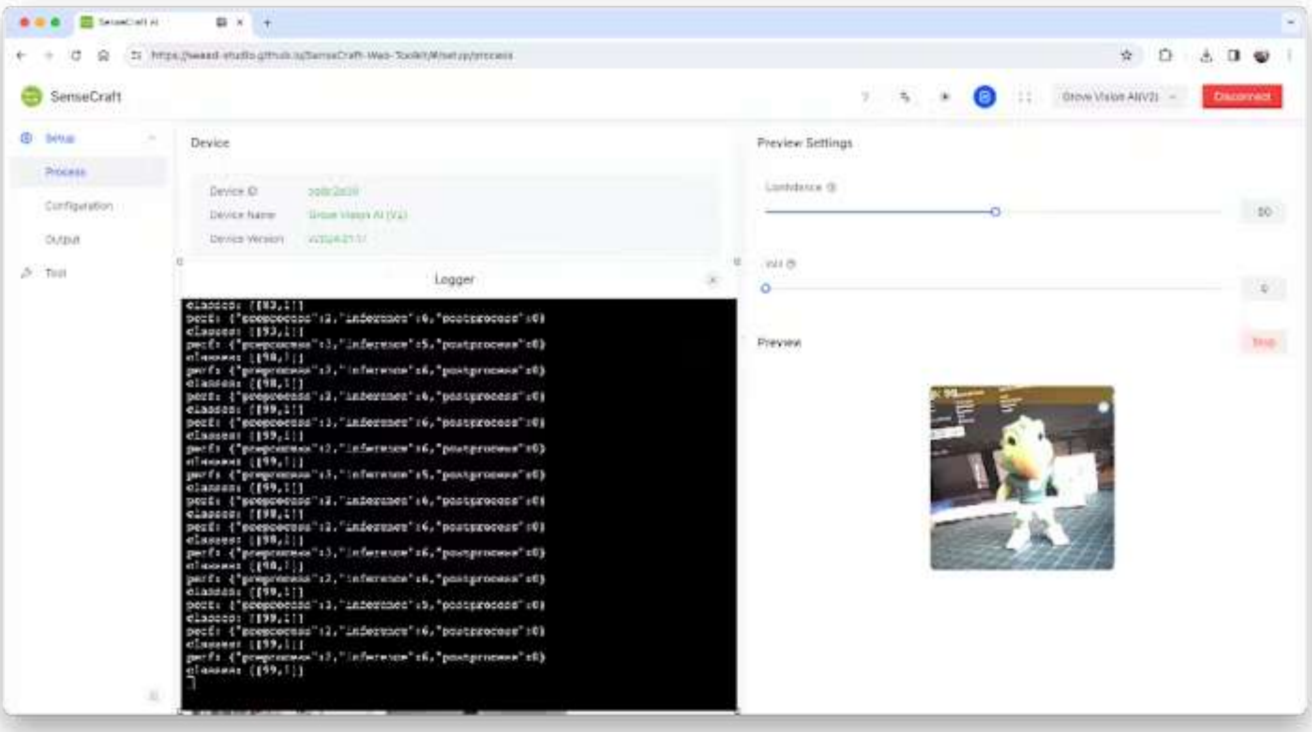
On SenseCraft-Web-Toolkit, use the blue button at the bottom of the page: [Upload Custom AI Model]. A window will pop up. Enter the Model file that you downloaded to your computer from Google Colab, choose a Model Name, and enter with labels (ID:Object):



> Note that you should use the labels trained on EI Studio, entering them at alphabetic order (in our case: 0: background, 1: periquito and 2: robot).

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. The Classification result will be at the top of the image (in my case, only the first letter of the class was shown). You can also select the Confidence of your inference cursor Confidence.

Click on the top button (Device Log) to open a Serial Monitor to follow the inference:



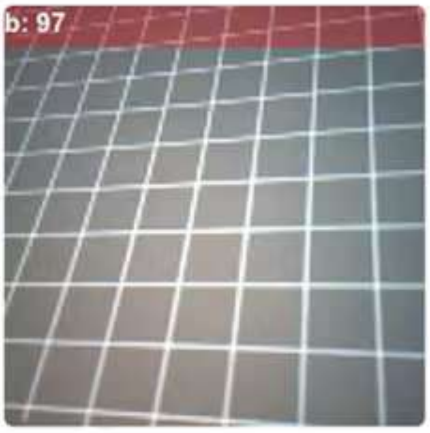
```
classes: [[93,1]]
perf: {"preprocess":2,"inference":6,"postprocess":0}
classes: [[93,1]]
perf: {"preprocess":3,"inference":5,"postprocess":0}
classes: [[98,1]]
perf: {"preprocess":3,"inference":6,"postprocess":0}
classes: [[98,1]]
perf: {"preprocess":2,"inference":6,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":3,"inference":6,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":2,"inference":6,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":3,"inference":5,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":2,"inference":6,"postprocess":0}
classes: [[98,1]]
perf: {"preprocess":2,"inference":6,"postprocess":0}
classes: [[98,1]]
perf: {"preprocess":3,"inference":6,"postprocess":0}
classes: [[98,1]]
perf: {"preprocess":2,"inference":6,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":3,"inference":5,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":2,"inference":6,"postprocess":0}
classes: [[99,1]]
perf: {"preprocess":3,"inference":6,"postprocess":0}
classes: [[99,1]]
```

On Device Log, you will get information as:

```
perf: {"preprocess":2,"inference":6,"postprocess":0}
classes: [[99,1]]
```

- Preprocess time (image capture and Crop): 2ms;
- Inference time (model latency): 6ms,
- Postprocess time (display of the image and inclusion of data): 0ms (<1ms).
- Output tensor (classes): [[99,1]]; where 1 is periquito (and 0 is the background, and 2 is a robot). 99 is the resulting score.

Here are other screenshots:



```
classes: [[97,0]]
```

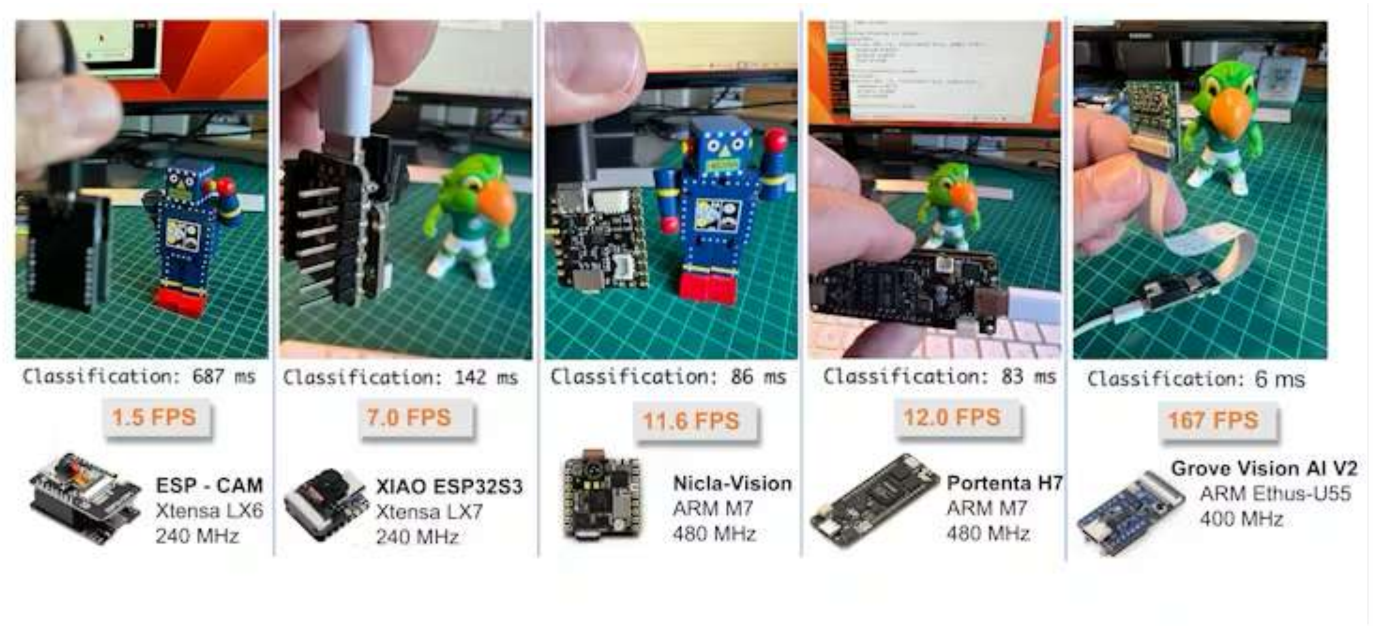
```
classes: [[97,1]]
```

```
classes: [[96,2]]
```

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, the Arduinos Nicla Vison, and Portenta.

Using the opportunity, a similarly trained model MobilenetV2 96x96, but with an alpha 0.1, was deployed on the ESP-CAM, the XIAO, and Portenta (in this one, the model was trained using grayscaled images to be compatible with its camera). Here is the result:



> The Grove Vision AI (V2) was 14 times faster than the devices with ARM-M7, even runing at smaller frequency and with a bigger model (alpha of.35).

Postprocessing

Now that we have the model uploaded to the board and working properly, classifying our images, let's connect a Master Device (in our case, an XIAO BLE Sense) to export the inference result to it and see the result completely offline (disconnected from the PC and, for example, powered by a battery).

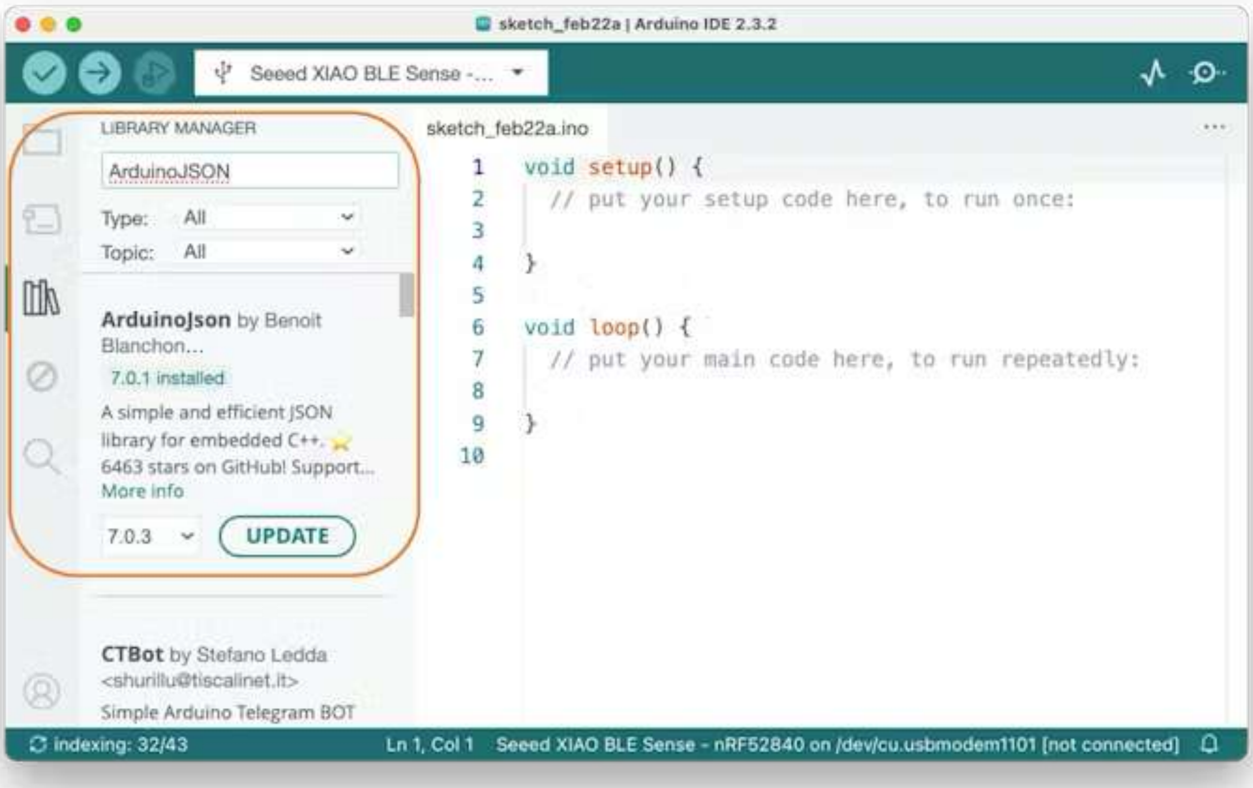
The idea is that whenever the image of a Periquito is detected, the LED Green will be ON; if it is a Robot, LED Blue will be ON; if it is a Background, the LED Red will be ON.

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. This library's main purpose is to process Grove Vision AI's data stream, which does not involve model inference or image processing.

> The Grove Vision AI (V2) communicates with the XIAO via the IIC; the device's IIC address is 0x62. Image information transfer is via the USB serial port (not uded in this project).

Step 1: Download the Arduino SSMA library as a zip file from its GitHub and install it to the Arduino IDE (sketch > Include Library > Add .Zip Library).

Step 2: Install the **ArduinoJSON** library.



sketch_feb22a | Arduino IDE 2.3.2

Seeed XIAO BLE Sense - ...

LIBRARY MANAGER

ArduinoJSON

Type: All

Topic: All

ArduinoJson by Benoit Blanchon...

7.0.1 installed

A simple and efficient JSON library for embedded C++. ⭐ 6463 stars on GitHub! Support... More info

7.0.3 UPDATE

CTBot by Stefano Ledda <shurillu@tiscalinet.it>

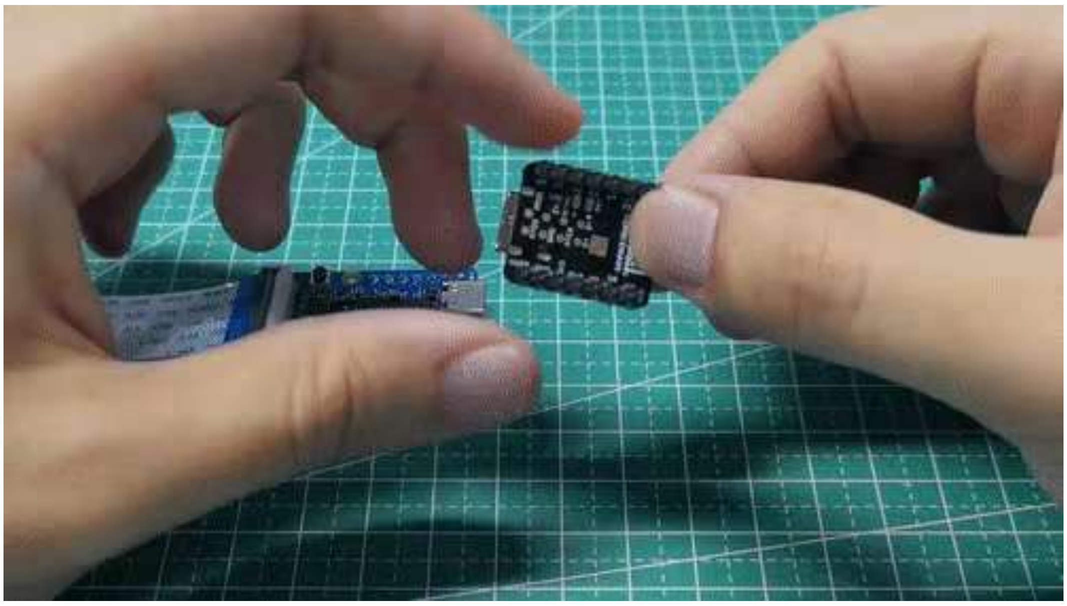
Simple Arduino Telegram BOT

sketch_feb22a.ino

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

}

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

}
```

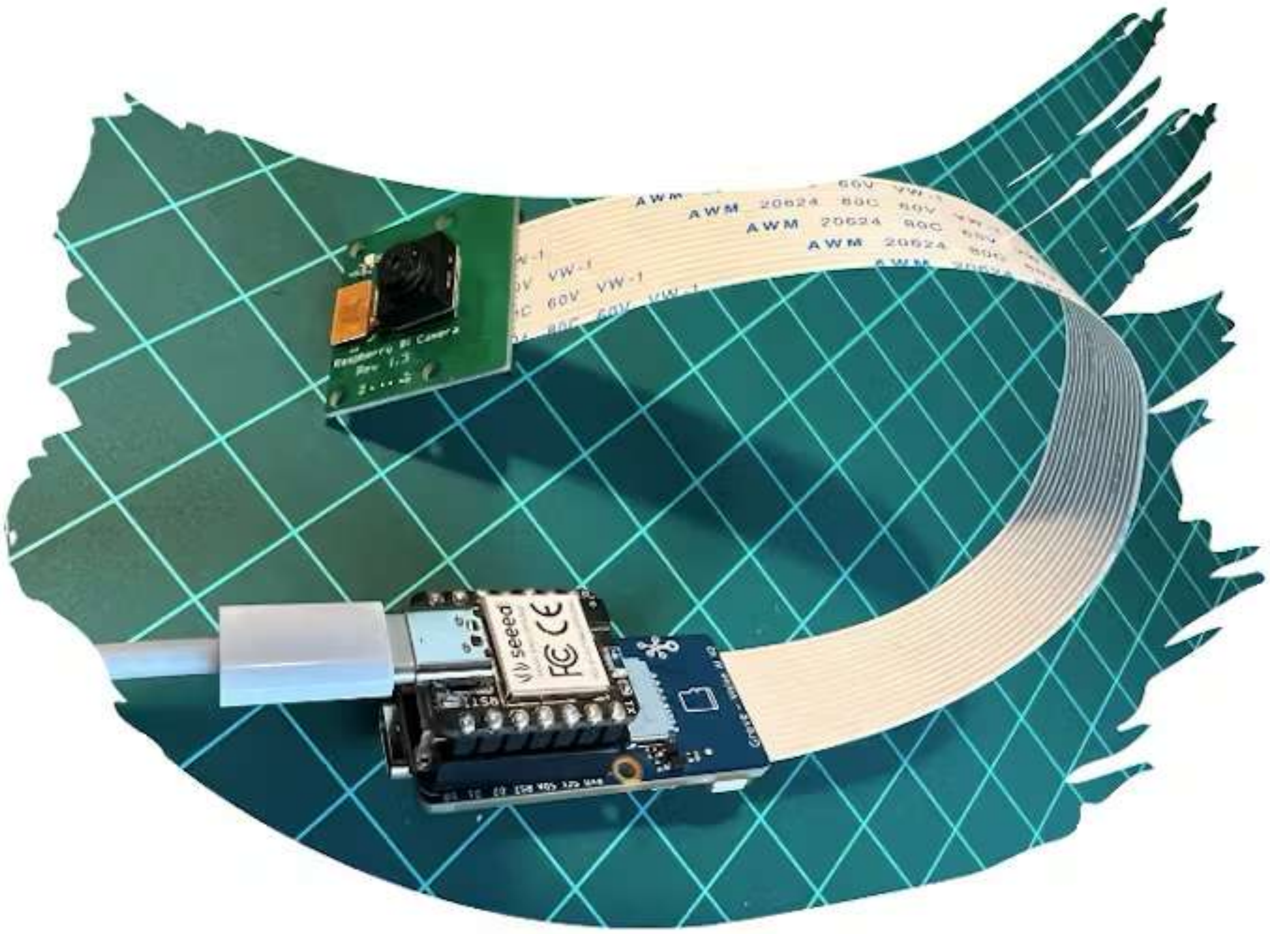
indexing: 32/43 Ln 1, Col 1 Seeed XIAO BLE Sense - nRF52840 on /dev/cu.usbmodem1101 [not connected]

Step 3: Now, we should connect the XIAO and Grove Vision AI (V2) via the socket (row of pins) available 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 4: Connect the **XIAO USB-C** port to your computer



Step 5: Select the Xiao board and USB port.

> For details about how to work with the XIAO BLE Sense, please see this tutorial: [TinyML Made Easy: Anomaly Detection & Motion Classification](#).

We are ready to create a code to receive and treat the inference result. Copy the below code and past it to your IDE:

```
/*
This Code for educational purposes, was developed by MJRovai
based on information from:
https://wiki.seeedstudio.com/grove_vision_ai_v2_software_support/
Using the XIAO BLE SENSE to show inference result in its RGB LED
*/

#include <Seeed_Arduino_SSCMA.h>
SSCMA AI;

void setup()
{
    AI.begin();

    Serial.begin(115200);
    while (!Serial);
    Serial.println("Inferencing - Grove AI V2 / XIAO Sense with RGB LEDs");

    // Pins for the built-in RGB LEDs
    pinMode(LEDR, OUTPUT);
    pinMode(LEDG, OUTPUT);
    pinMode(LEDB, OUTPUT);
    // Ensure the LEDs are OFF by default.
    // Note: The RGB LEDs are ON when the pin is LOW, OFF when HIGH.
    digitalWrite(LEDR, HIGH);
    digitalWrite(LEDG, HIGH);
    digitalWrite(LEDB, 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_leds(pred_index);
    }
}

/**
* @brief      turn_off_leds function - turn-off all RGB LEDs
*/
void turn_off_leds(){
    digitalWrite(LEDR, HIGH);
    digitalWrite(LEDG, HIGH);
    digitalWrite(LEDB, HIGH);
}

/**
* @brief      turn_on_leds function used to turn on the RGB LEDs
* @param[in]  pred_index
*             label 0: [0] ==> Red ON
*             label 1: [1] ==> Green ON
*             label 2: [2] ==> Blue ON
*             label 3: [3] ==> ALL OFF
*/
void turn_on_leds(int pred_index) {
    switch (pred_index)
    {
        case 0:
            turn_off_leds();
            digitalWrite(LEDR, LOW);
            break;
        case 1:
            turn_off_leds();
            digitalWrite(LEDG, LOW);
            break;
        case 2:
            turn_off_leds();
            digitalWrite(LEDB, LOW);
            break;
        case 3:
            turn_off_leds();
            break;
    }
}
```

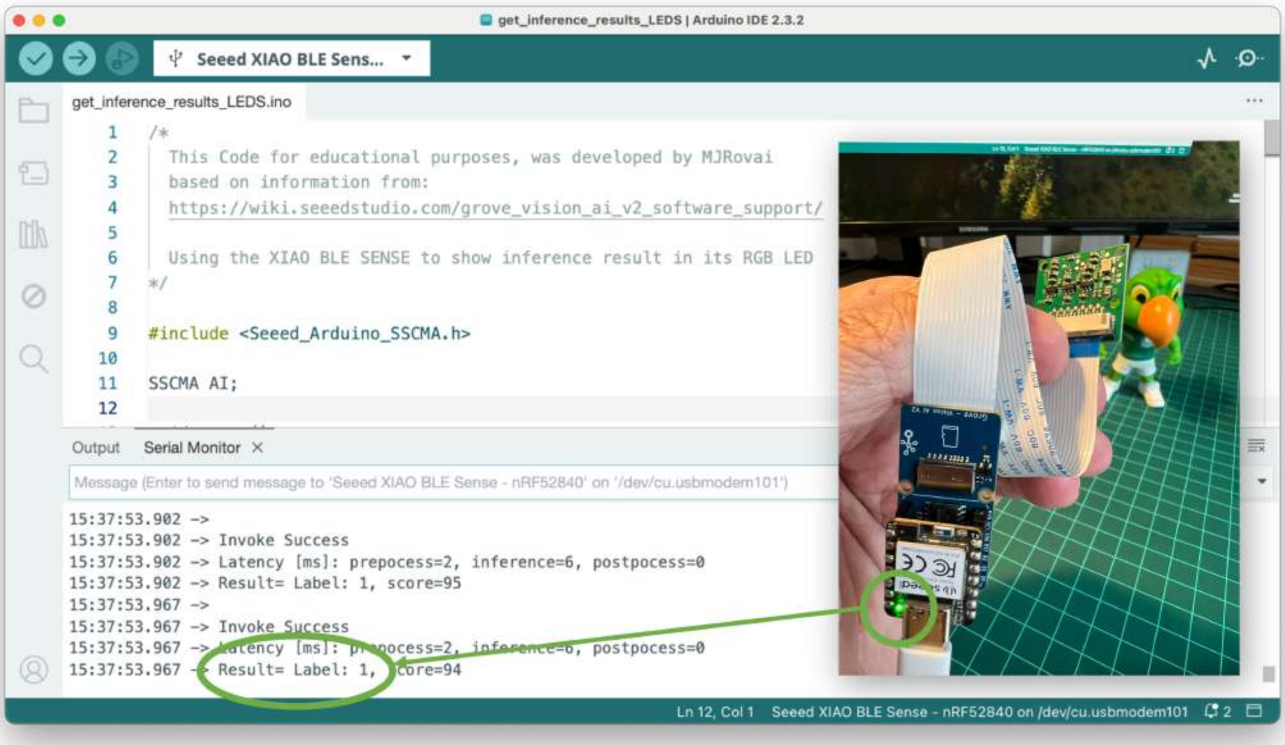
This sketch uses the Seeed_Arduino_SSCMA.h library to interface with the Grove Vision AI Module V2. AI module and LEDs are initialized in the setup() function, and serial communication is started.

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:

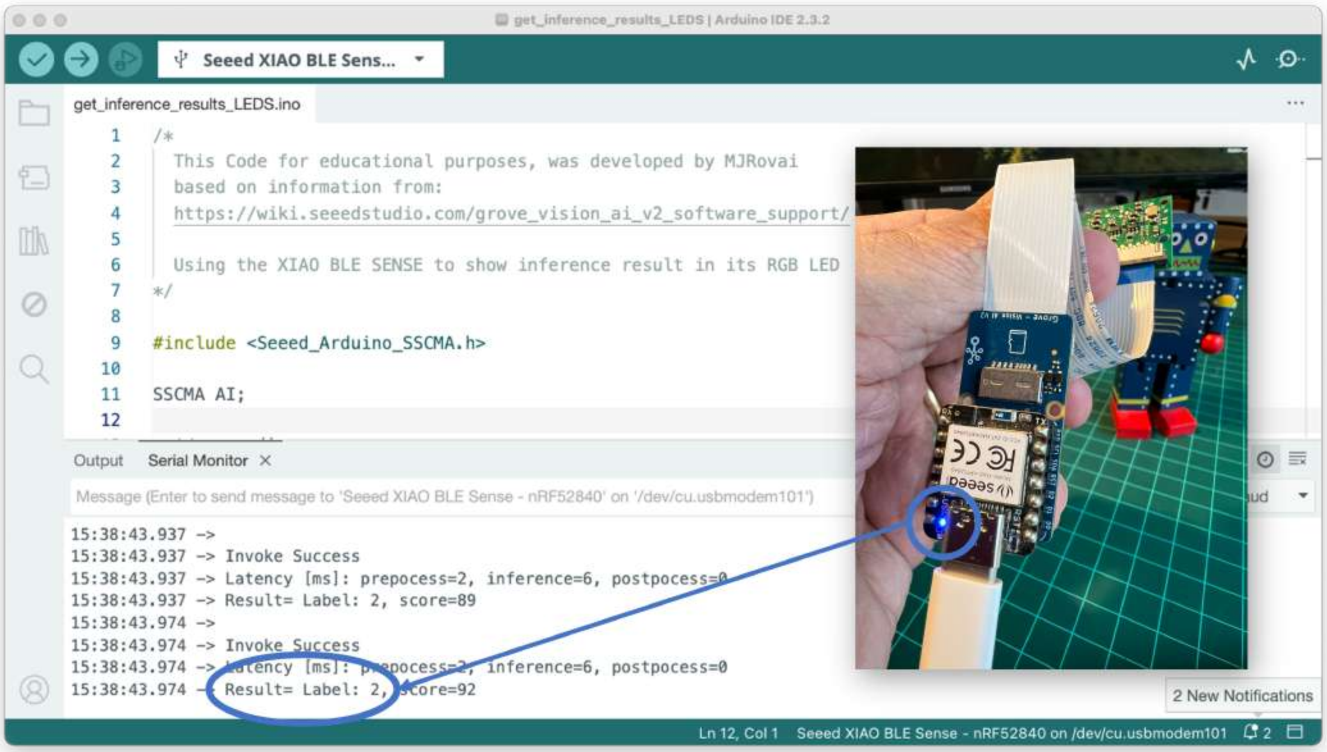
- (AI.classes()[0]) that identifies the class of image (.target) and its confidence score (.score).
- The inference result (class) is stored on the integer variable pred_index, which will be used as an input of the function turn_on_leds(). As a result, the proper LED will turn ON, depending on the classification result.

Here is the result:


get_inference_results_LEDS | Arduino IDE 2.3.2
Seeed XIAO BLE Sens...
get_inference_results_LEDS.ino
Output
Serial Monitor
Message (Enter to send message to 'Seeed XIAO BLE Sense - nRF52840' on '/dev/cu.usbmodem101')
Ln 12, Col 1
Seeed XIAO BLE Sense - nRF52840 on /dev/cu.usbmodem101

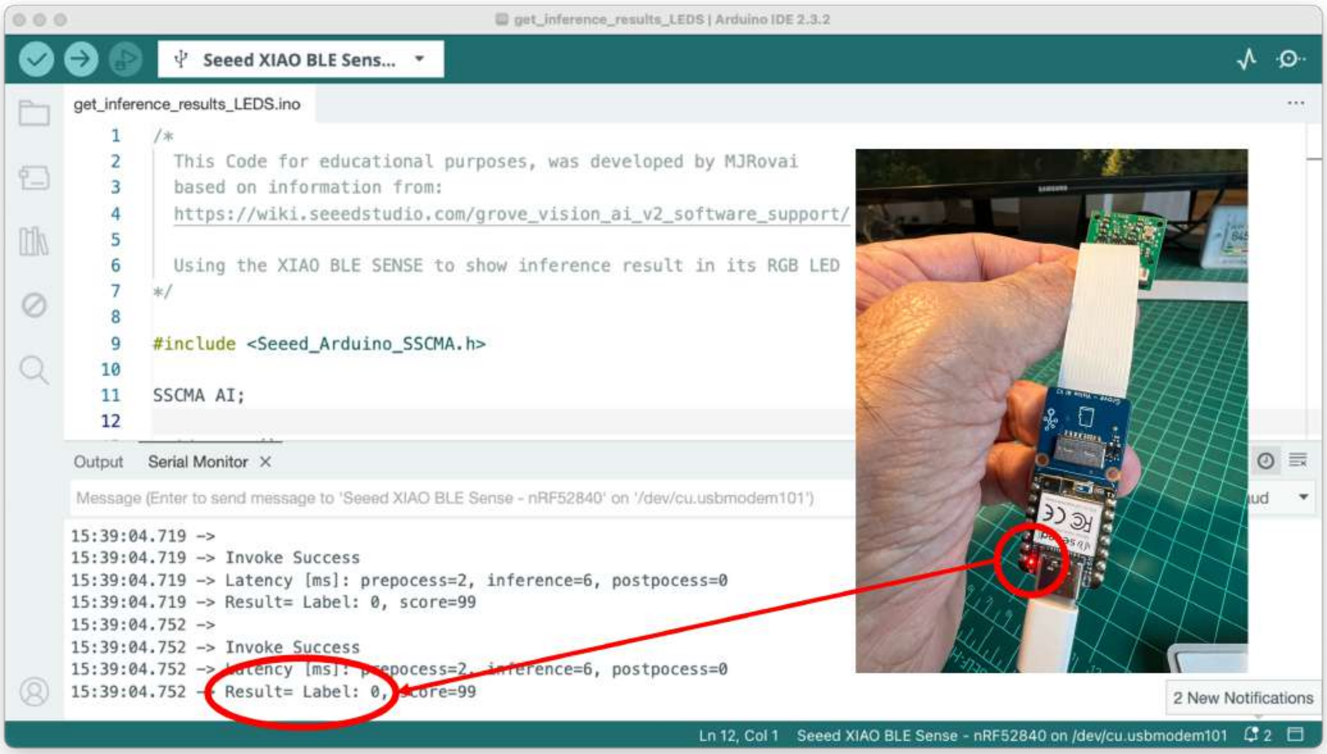
```
/*
  This Code for educational purposes, was developed by MJRovai
  based on information from:
  https://wiki.seeedstudio.com/grove_vision_ai_v2_software_support/

  Using the XIAO BLE SENSE to show inference result in its RGB LED
*/

#include <Seeed_Arduino_SSCMA.h>

SSCMA AI;

```

```
15:37:53.902 ->
15:37:53.902 -> Invoke Success
15:37:53.902 -> Latency [ms]: prepocess=2, inference=6, postpocess=0
15:37:53.902 -> Result= Label: 1, score=95
15:37:53.967 ->
15:37:53.967 -> Invoke Success
15:37:53.967 -> Latency [ms]: prepocess=2, inference=6, postpocess=0
15:37:53.967 -> Result= Label: 1, score=94
```


get_inference_results_LEDS | Arduino IDE 2.3.2
Seeed XIAO BLE Sens...
get_inference_results_LEDS.ino
Output
Serial Monitor
Message (Enter to send message to 'Seeed XIAO BLE Sense - nRF52840' on '/dev/cu.usbmodem101')
2 New Notifications
Ln 12, Col 1
Seeed XIAO BLE Sense - nRF52840 on /dev/cu.usbmodem101

```
/*
  This Code for educational purposes, was developed by MJRovai
  based on information from:
  https://wiki.seeedstudio.com/grove_vision_ai_v2_software_support/

  Using the XIAO BLE SENSE to show inference result in its RGB LED
*/

#include <Seeed_Arduino_SSCMA.h>

SSCMA AI;

```

```
15:38:43.937 ->
15:38:43.937 -> Invoke Success
15:38:43.937 -> Latency [ms]: prepocess=2, inference=6, postpocess=0
15:38:43.937 -> Result= Label: 2, score=89
15:38:43.974 ->
15:38:43.974 -> Invoke Success
15:38:43.974 -> Latency [ms]: prepocess=2, inference=6, postpocess=0
15:38:43.974 -> Result= Label: 2, score=92
```


get_inference_results_LEDS | Arduino IDE 2.3.2
Seeed XIAO BLE Sens...
get_inference_results_LEDS.ino
Output
Serial Monitor
2 New Notifications
Ln 12, Col 1
Seeed XIAO BLE Sense - nRF52840 on /dev/cu.usbmodem101

```
/*
  This Code for educational purposes, was developed by MJRovai
  based on information from:
  https://wiki.seeedstudio.com/grove_vision_ai_v2_software_support/

  Using the XIAO BLE SENSE to show inference result in its RGB LED
*/

#include <Seeed_Arduino_SSCMA.h>

SSCMA AI;
```

Message (Enter to send message to 'Seeed XIAO BLE Sense - nRF52840' on '/dev/cu.usbmodem101')

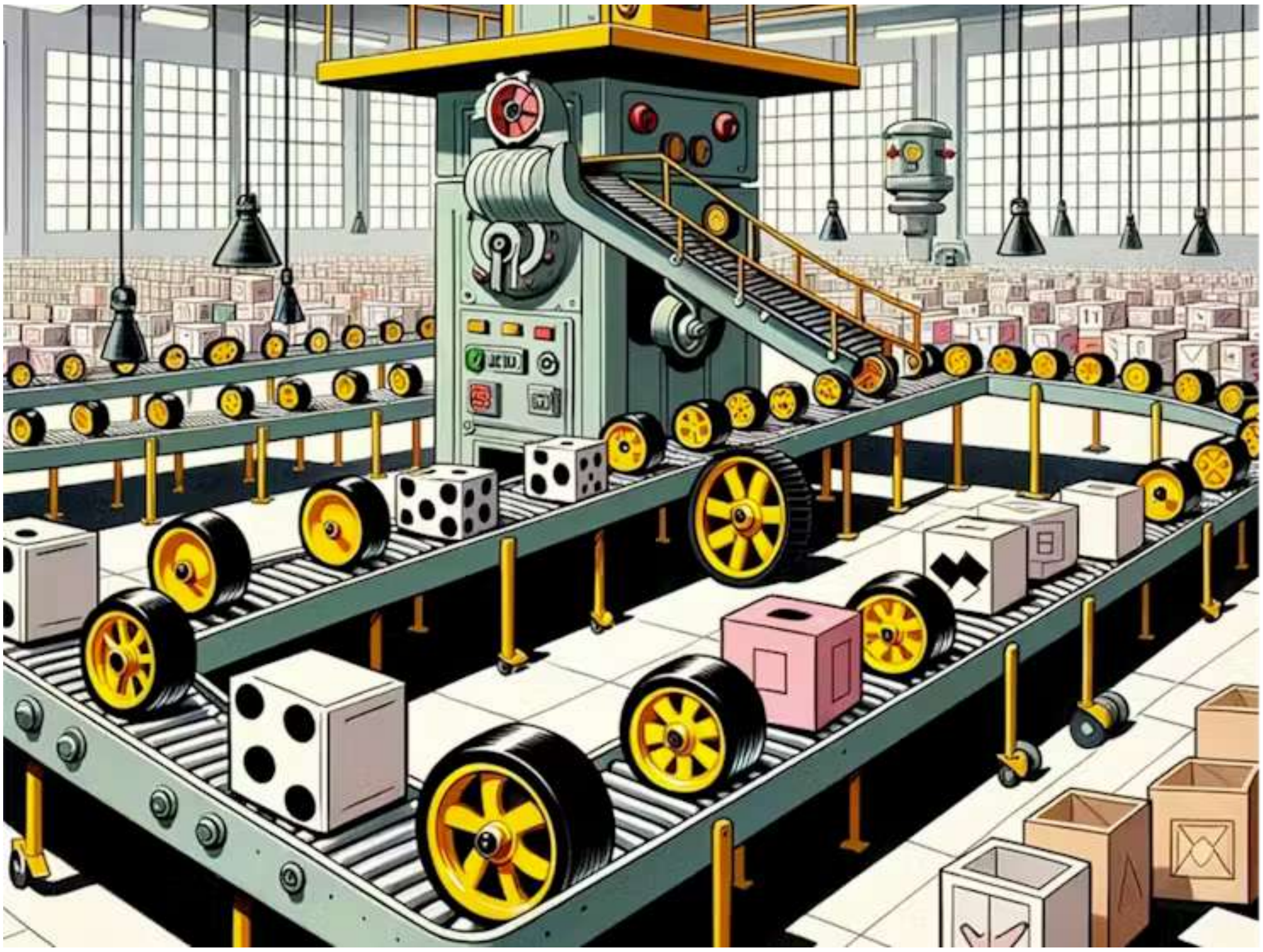
```
15:39:04.719 ->
15:39:04.719 -> Invoke Success
15:39:04.719 -> Latency [ms]: prepocess=2, inference=6, postpocess=0
15:39:04.719 -> Result= Label: 0, score=99
15:39:04.752 ->
15:39:04.752 -> Invoke Success
15:39:04.752 -> Latency [ms]: prepocess=2, inference=6, postpocess=0
15:39:04.752 -> Result= Label: 0, score=99
```

So, you can now feed the Xiao with an external battery, and the result of the inference will be shown by the LEDs completely offline. Of course, it is also possible to send the result using BLE or other communication protocols available on the Master Device used.

5. Testing a FOMO (Object Detection) Project

The steps to develop Object Detection using the FOMO algorithm are similar to those we used with the Image Classification project. To save time, let's use a TFLite model already trained in another project: TinyML Made Easy: Object Detection with Nicla Vision.

All Machine Learning projects need to start with a detailed goal. Let's assume we are in an industrial facility and must sort and count **wheels** and special **boxes**.



The full Edge Impulse public project can be found here: Box_and_Wheel_Object Detection.

Follow the steps:

- On the Edge Impulse Dashboard section, download the model ("block output"): Transfer learning model - TensorFlow Lite (int8 quantized).
- Rename the model downloaded, simplify the name, and, more importantly, change the extension.lite to **.tflite,** for example:

ei-box_and_wheel_fomo-int8.tflite

- On a Google CoLab, run the lines below:

```
!pip install ethos-u-vela
!!vela --version
```

We should get: 3.10.0

- Next, upload the.tflite model to the Colab and run the line below:

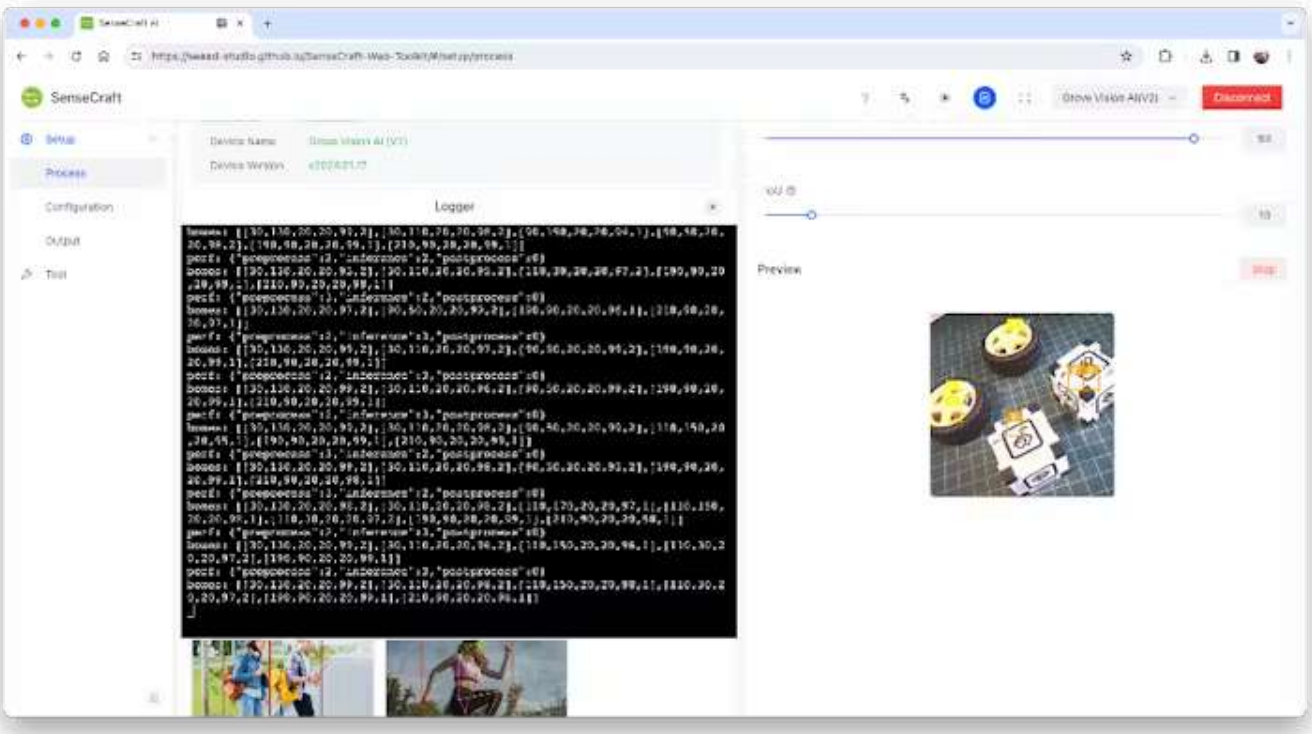
```
!vela ei-box_and_wheel_fomo-int8.tflite --accelerator-config ethos-u55-64
```

A new folder should be created (output). Open it and download the converted model (ei-box_and_wheel_fomo-int8_vela.tflite):

> **Caution**: It is not possible for Grove Vision AI (V2) to output a live screen, when having the XIAO attached.

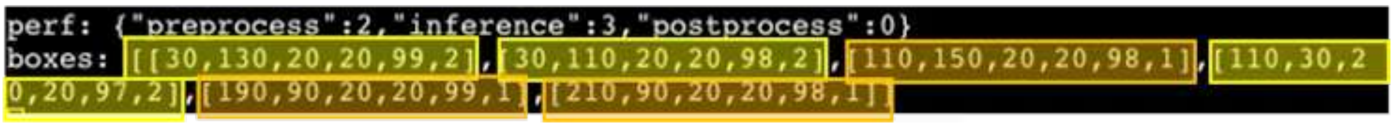
- Disconnect the Xiao and connect the Grove Vision AI (V2) to the computer via its USB-C connection
- On SenseCraft-Web-Toolkit, use the blue button at the bottom of the page: [Upload Custom AI Model]. A window will pop up. Enter the Model file that you downloaded to your computer from Google Colab, choose a Model Name, and enter with labels (ID:Object):

> Note that you should use the labels trained on EI Studio, entering them at alphabetic order (in our case: 0: background, 1: box and 2: wheel).



```
boxes: [[30,130,20,20,99,2],[30,110,20,20,98,2],[90,150,20,20,94,1],[90,90,20,
20,98,2],[190,90,20,20,99,1],[210,90,20,20,99,1]]
perf: {"preprocess":2,"inference":2,"postprocess":0}
boxes: [[30,130,20,20,92,2],[30,110,20,20,95,2],[110,30,20,20,97,2],[190,90,20
,20,99,1],[210,90,20,20,98,1]]
perf: {"preprocess":3,"inference":2,"postprocess":0}
boxes: [[30,130,20,20,97,2],[90,50,20,20,92,2],[190,90,20,20,96,1],[210,90,20,
20,97,1]]
perf: {"preprocess":2,"inference":3,"postprocess":0}
boxes: [[30,130,20,20,95,2],[30,110,20,20,97,2],[90,50,20,20,99,2],[190,90,20,
20,99,1],[210,90,20,20,99,1]]
perf: {"preprocess":2,"inference":2,"postprocess":0}
boxes: [[30,130,20,20,99,2],[30,110,20,20,96,2],[90,50,20,20,99,2],[190,90,20,
20,99,1],[210,90,20,20,99,1]]
perf: {"preprocess":2,"inference":3,"postprocess":0}
boxes: [[30,130,20,20,92,2],[30,110,20,20,98,2],[90,50,20,20,90,2],[110,150,20
,20,95,1],[190,90,20,20,99,1],[210,90,20,20,99,1]]
perf: {"preprocess":3,"inference":2,"postprocess":0}
boxes: [[30,130,20,20,99,2],[30,110,20,20,96,2],[90,50,20,20,91,2],[190,90,20,
20,99,1],[210,90,20,20,98,1]]
perf: {"preprocess":3,"inference":2,"postprocess":0}
boxes: [[30,130,20,20,98,2],[30,110,20,20,98,2],[110,170,20,20,97,1],[110,150,
20,20,98,1],[110,30,20,20,97,2],[190,90,20,20,99,1],[210,90,20,20,98,1]]
perf: {"preprocess":2,"inference":3,"postprocess":0}
boxes: [[30,130,20,20,99,2],[30,110,20,20,98,2],[110,150,20,20,96,1],[110,30,2
0,20,97,2],[190,90,20,20,99,1]]
perf: {"preprocess":2,"inference":3,"postprocess":0}
boxes: [[30,130,20,20,99,2],[30,110,20,20,98,2],[110,150,20,20,98,1],[110,30,2
0,20,97,2],[190,90,20,20,99,1],[210,90,20,20,98,1]]
```

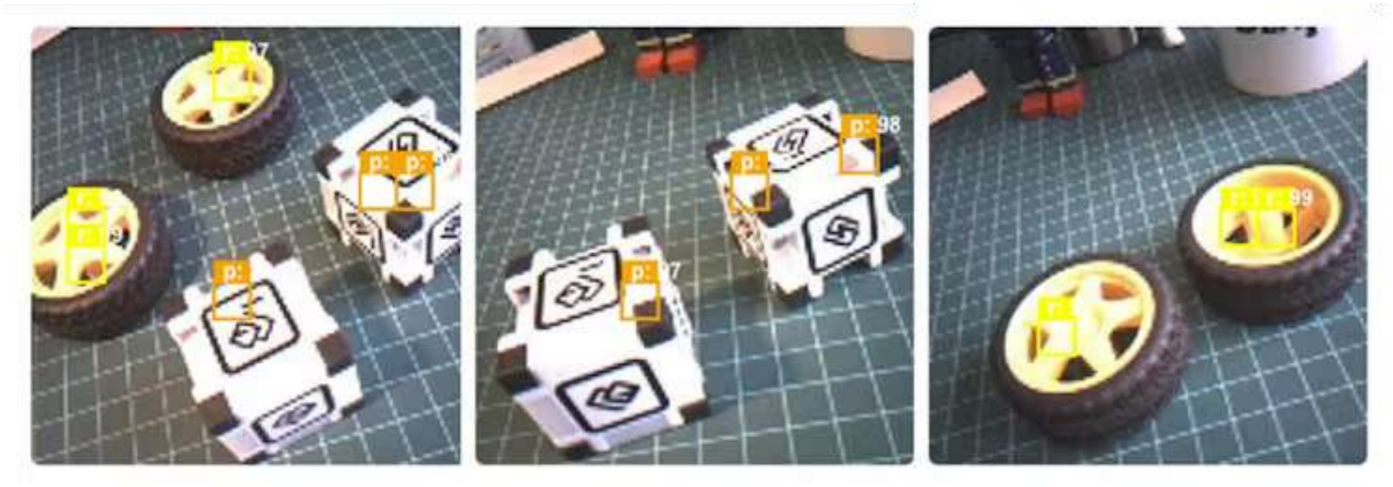
Looking to Log monitor:



```
perf: {"preprocess":2,"inference":3,"postprocess":0}
boxes: [[30,130,20,20,99,2],[30,110,20,20,98,2],[110,150,20,20,98,1],[110,30,2
0,20,97,2],[190,90,20,20,99,1],[210,90,20,20,98,1]]
```

We can observe that the inference time is only 3 ms, which means more than 330 fps. Although the model can identify the objects, more than one centroid is usually detected per object. This issue should be solved during post-processing.

Below the inference result, the boxes' centroids are marked with a small orange square, and the wheels are yellow.



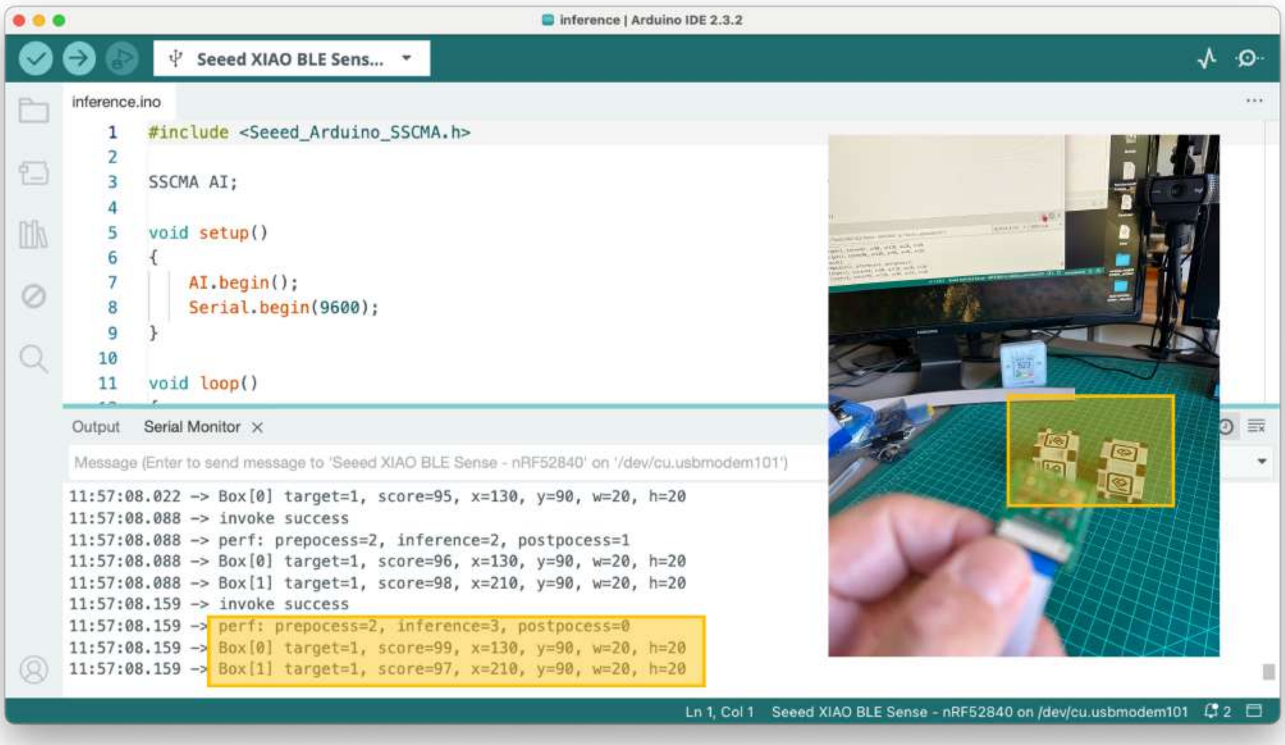
Inference at Arduino IDE

As we saw with the Image Classification project, we can see the inference result at the Arduino IDE Serial Monitor.

Follow the steps:

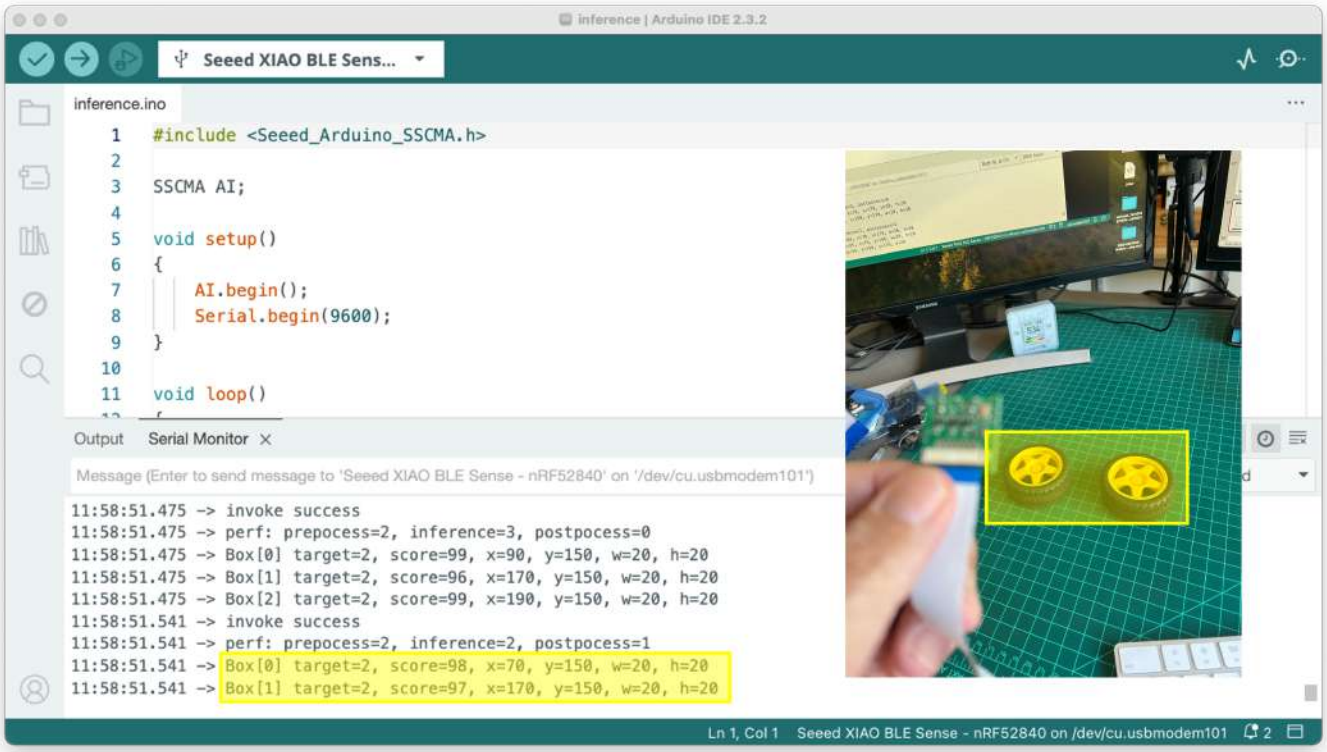
- Disconnect the Grove Vision AI (V2) from your computer.
- Install the XIAO as a Master.
- Connect the XIAO to the computer via a USB-C cable.
- Select the proper board/port
- Go to Files > Examples > Seeed_Arduino_SSCMA > inference
- Run the inference.ino

The inference result will be shown in the Serial Monitor:



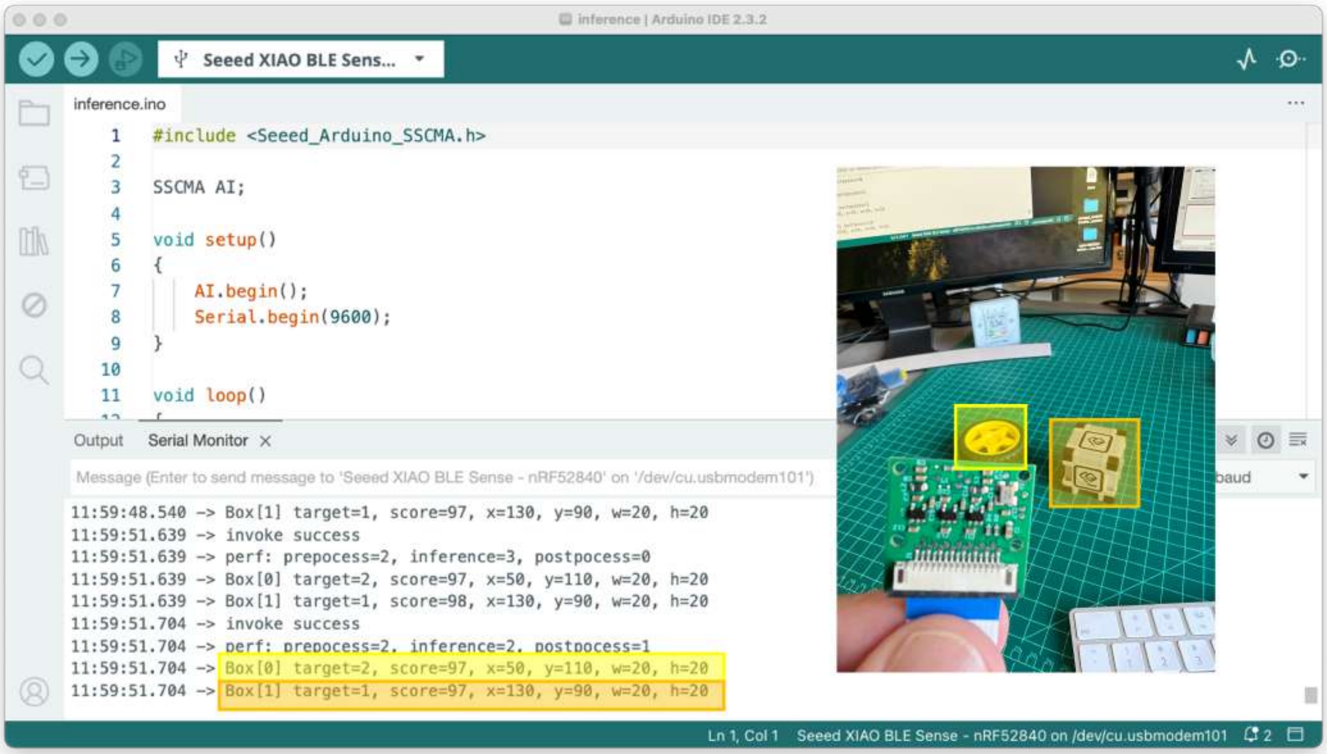
```
#include <Seeed_Arduino_SSCMA.h>

SSCMA AI;

void setup()
{
    AI.begin();
    Serial.begin(9600);
}

void loop()
```

```
11:57:08.022 -> Box[0] target=1, score=95, x=130, y=90, w=20, h=20
11:57:08.088 -> invoke success
11:57:08.088 -> perf: prepocess=2, inference=2, postpocess=1
11:57:08.088 -> Box[0] target=1, score=96, x=130, y=90, w=20, h=20
11:57:08.088 -> Box[1] target=1, score=98, x=210, y=90, w=20, h=20
11:57:08.159 -> invoke success
11:57:08.159 -> perf: prepocess=2, inference=3, postpocess=0
11:57:08.159 -> Box[0] target=1, score=99, x=130, y=90, w=20, h=20
11:57:08.159 -> Box[1] target=1, score=97, x=210, y=90, w=20, h=20
```



```
#include <Seeed_Arduino_SSCMA.h>

SSCMA AI;

void setup()
{
    AI.begin();
    Serial.begin(9600);
}

void loop()
```

```
11:58:51.475 -> invoke success
11:58:51.475 -> perf: prepocess=2, inference=3, postpocess=0
11:58:51.475 -> Box[0] target=2, score=99, x=90, y=150, w=20, h=20
11:58:51.475 -> Box[1] target=2, score=96, x=170, y=150, w=20, h=20
11:58:51.475 -> Box[2] target=2, score=99, x=190, y=150, w=20, h=20
11:58:51.541 -> invoke success
11:58:51.541 -> perf: prepocess=2, inference=2, postpocess=1
11:58:51.541 -> Box[0] target=2, score=98, x=70, y=150, w=20, h=20
11:58:51.541 -> Box[1] target=2, score=97, x=170, y=150, w=20, h=20
```


inference | Arduino IDE 2.3.2
Seeed XIAO BLE Sens...
inference.ino
Output
Serial Monitor
Message (Enter to send message to 'Seeed XIAO BLE Sense - nRF52840' on '/dev/cu.usbmodem101')
baud
Ln 1, Col 1
Seeed XIAO BLE Sense - nRF52840 on /dev/cu.usbmodem101

```
#include <Seeed_Arduino_SSCMA.h>

SSCMA AI;

void setup()
{
    AI.begin();
    Serial.begin(9600);
}

void loop()
```

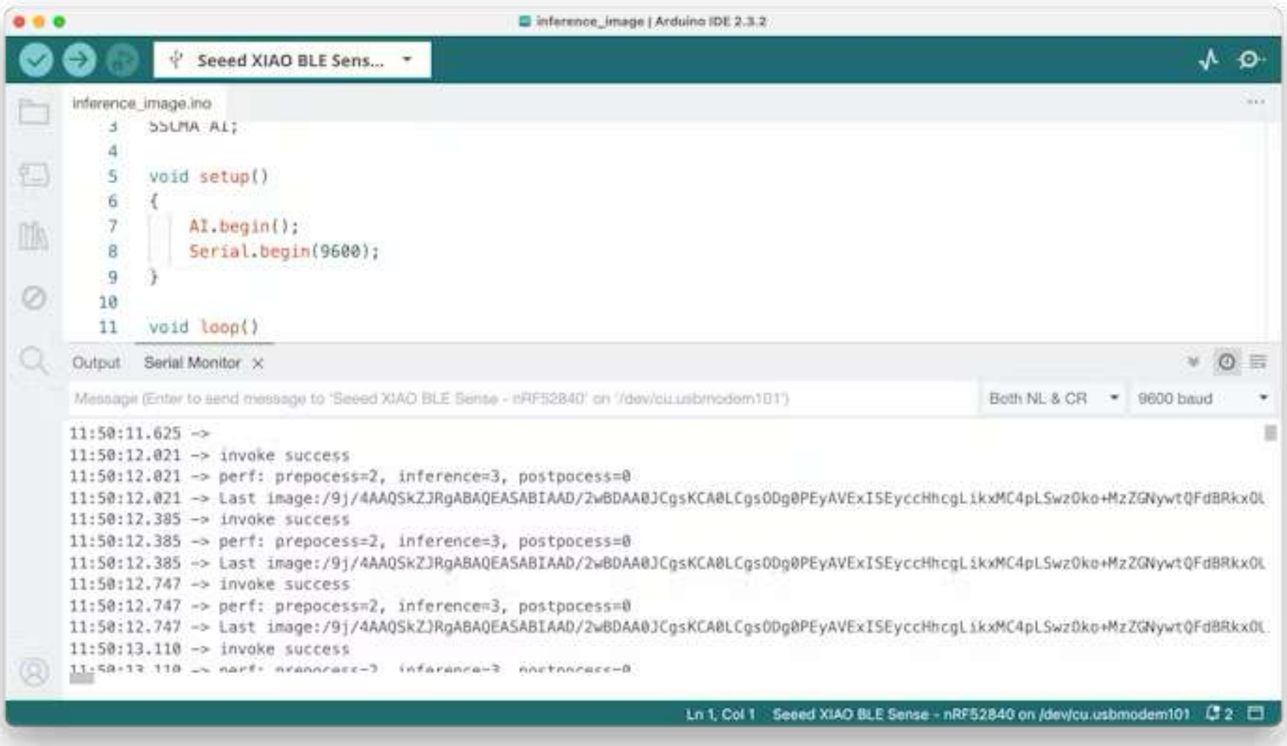
```
11:59:48.540 -> Box[1] target=1, score=97, x=130, y=90, w=20, h=20
11:59:51.639 -> invoke success
11:59:51.639 -> perf: prepocess=2, inference=3, postpocess=0
11:59:51.639 -> Box[0] target=2, score=97, x=50, y=110, w=20, h=20
11:59:51.639 -> Box[1] target=1, score=98, x=130, y=90, w=20, h=20
11:59:51.704 -> invoke success
11:59:51.704 -> perf: prepocess=2, inference=2, postpocess=1
11:59:51.704 -> Box[0] target=2, score=97, x=50, y=110, w=20, h=20
11:59:51.704 -> Box[1] target=1, score=97, x=130, y=90, w=20, h=20
```

- Target 1 is the box label, and 2 is the wheel.

Note that w and h values will always be 20,20. We will get only the centroid coordinates (x,y) with FOMO.

In the examples, you can also have the code: inference_image.ino. With that, you can get the last image as raw data captured by the device.

```
if (AI.last_image().length() > 0)
{
Serial.print("Last image:");
Serial.println(AI.last_image());
}
```


inference_image | Arduino IDE 2.3.2
Seeed XIAO BLE Sens...
inference_image.ino
Output
Serial Monitor
Message (Enter to send message to 'Seeed XIAO BLE Sense - nRF52840' on '/dev/cu.usbmodem101')
Both NL & CR
9600 baud
Ln 1, Col 1
Seeed XIAO BLE Sense - nRF52840 on /dev/cu.usbmodem101
2

```
SSCMA AI;

void setup()
{
    AI.begin();
    Serial.begin(9600);
}

void loop()
```

```
11:50:11.625 ->
11:50:12.021 -> invoke success
11:50:12.021 -> perf: prepocess=2, inference=3, postpocess=0
11:50:12.021 -> Last image:/9j/4AAQSkZJRgABAQEASABIAAD/2wBDAA0JCgsKCA0LCgsODg0PEyAVExISEyccHhcgLikxMC4pLSwzOko+MzZGNywtQFdBRkxOU
11:50:12.385 -> invoke success
11:50:12.385 -> perf: prepocess=2, inference=3, postpocess=0
11:50:12.385 -> Last image:/9j/4AAQSkZJRgABAQEASABIAAD/2wBDAA0JCgsKCA0LCgsODg0PEyAVExISEyccHhcgLikxMC4pLSwzOko+MzZGNywtQFdBRkxOU
11:50:12.747 -> invoke success
11:50:12.747 -> perf: prepocess=2, inference=3, postpocess=0
11:50:12.747 -> Last image:/9j/4AAQSkZJRgABAQEASABIAAD/2wBDAA0JCgsKCA0LCgsODg0PEyAVExISEyccHhcgLikxMC4pLSwzOko+MzZGNywtQFdBRkxOU
11:50:13.110 -> invoke success
11:50:13.110 -> perf: prepocess=2, inference=3, postpocess=0
```

6. Conclusion

In this tutorial, we explored several computer vision (CV) applications using the Seeed Studio *Grove Vision AI Module V2*, a powerful but still tiny device specially developed for applications in embedded machine learning. We could also see that the device is fast, mainly because it is based on the Himax WiseEye2 chip, a processor with a dual-core Arm Cortex-M55 and integrated ARM Ethos-U55 neural network unit, specifically designed to accelerate ML inference in area-constrained embedded and IoT devices.

This device has much more to explore, such as creating from scratch, object detection, and pose estimation projects with YOLO.

Knowing more

If you want to learn more about Embedded Machine Learning (TinyML), please see these references:

- "TinyML - Machine Learning for Embedding Devices" - UNIFEI
- "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
- "Deep Learning with Python" by François Chollet
- “TinyML” by Pete Warden, Daniel Situnayake
- "TinyML Cookbook" by Gian Marco Iodice
- "AI at the Edge" by Daniel Situnayake, Jenny Plunkett
- MACHINE LEARNING SYSTEMS for TinyML - Collaborative effort
- XIAO: Big Power, Small Board - Lei Feng, Marcelo Rovai
- Edge Impulse Expert Network - Collective authors

> On the TinyML4D website, You can find lots of educational materials on TinyML. They are all free and open-source for educational uses – we ask that if you use the material, please cite them! TinyML4D is an initiative to make TinyML education available to everyone globally.

TINYML4D

That's all, **folks!**

As always, I hope this project can help others find their way into the exciting world of AI!

Greetings from the south of the world!

See you at my next project!

Thank you

Marcelo