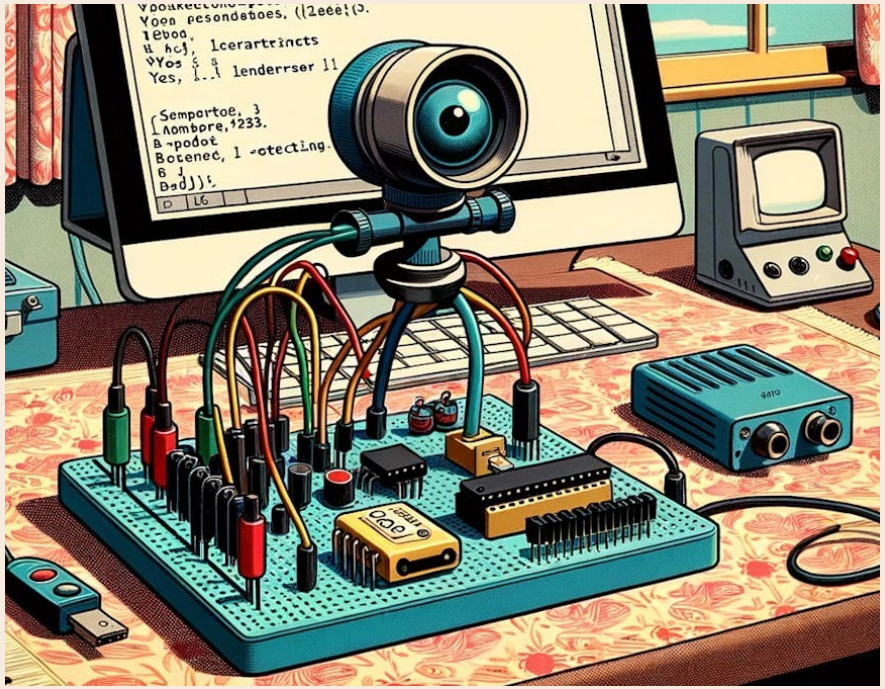
CHAPTER 3 · TINYML ON SEEED

Object Detection

Not just what is in the frame, but where, and how many

FOMO · EDGE IMPULSE · SENSECRAFT

47 slides · 1 lab · XIAO ESP32S3 Sense



01

WHERE WE ARE · PART I

From "what" to "where"

One label per image was enough for a box and a wheel. It is not enough for a conveyor belt.

LEARNING OBJECTIVES

What you will be able to do

01

Tell detection from classification

Know when one label per image stops working, and what a detector returns instead.

02

Explain how FOMO works

A grid of tiny classifiers and a centroid per object, instead of a bounding box.

03

Label and train in Edge Impulse

Bounding boxes, AI-assisted labeling, FOMO MobileNetV2 0.35, and the F1 score.

04

Deploy it two ways

An Arduino library with serial output, and SenseCraft with a live preview.

CLASSIFICATION

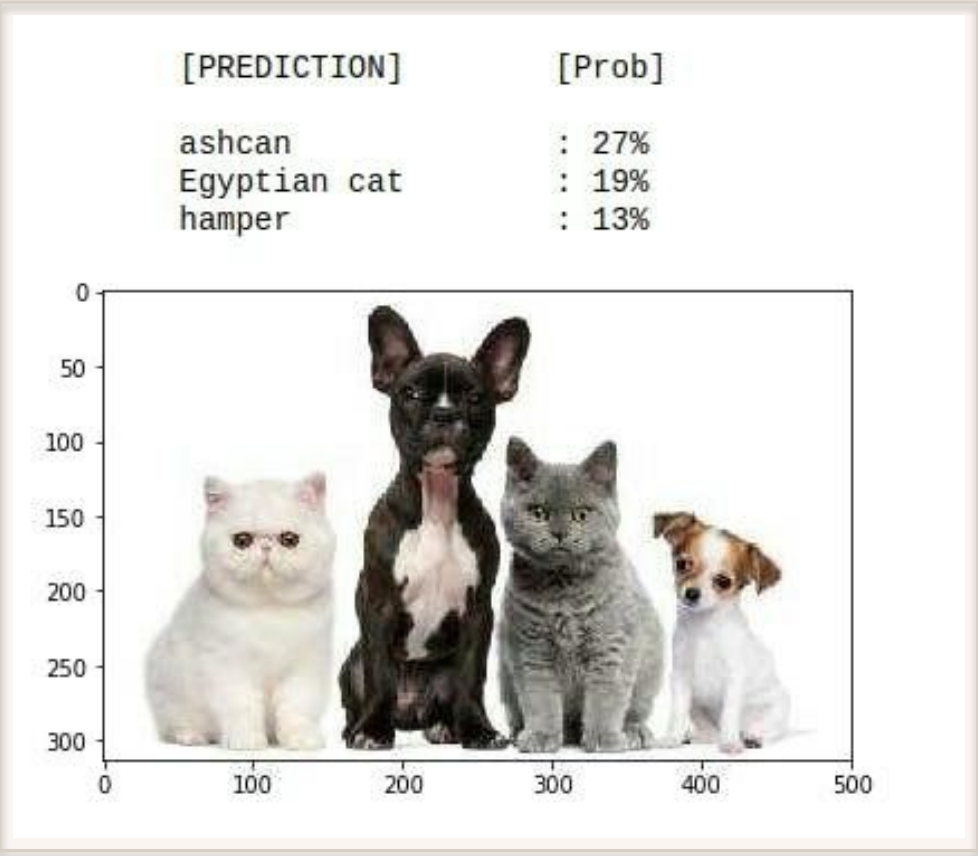
One dominant object, one label



Cat or dog. When one object fills the frame, a single label is the right answer.

WHEN IT BREAKS

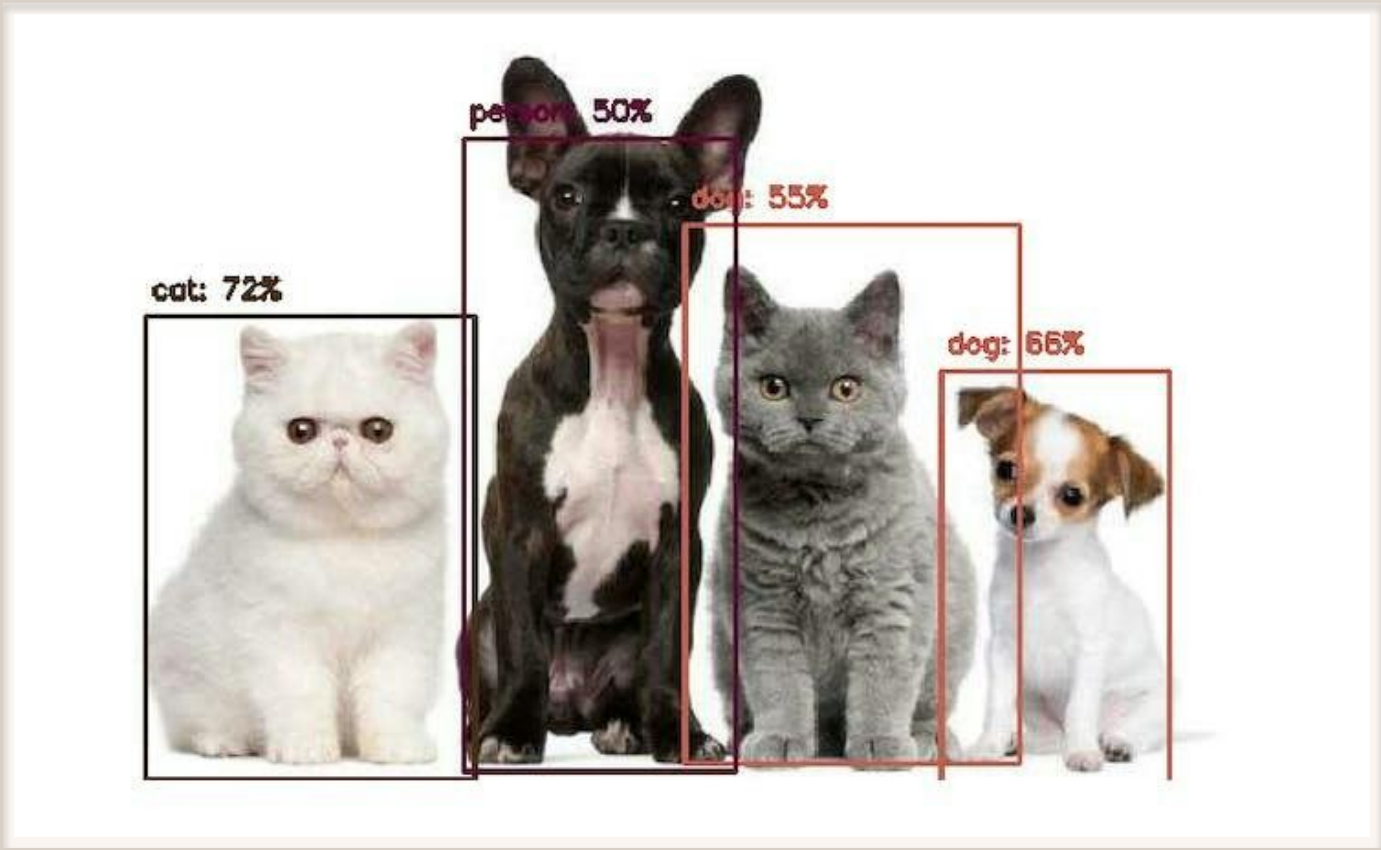
Four animals, no dominant one, and the answer is "ashcan"



MobileNet trained on ImageNet, running on a Raspberry Pi. Top score 27 %, for the wrong class.

OBJECT DETECTION

Every object gets a class, a score, and a box



MobileNetV2 SSD FPN-Lite 320 × 320, trained on COCO, on a Raspberry Pi. Up to 10 objects per frame.

THE PROBLEM

SSD and YOLO models weigh several megabytes. Our board has 512 KB of fast internal RAM.

The SSD above is about 3.7 MB. Fine on a Raspberry Pi; far too big for a microcontroller.

THREE ANSWERS

Three models, three different questions

Classification

What is in this image? One label for the whole frame.

Detection with boxes

What, where, and how big? A class and a bounding box per object. SSD, YOLO.

FOMO

What, where, and how many? A class and a centroid per object. No size.

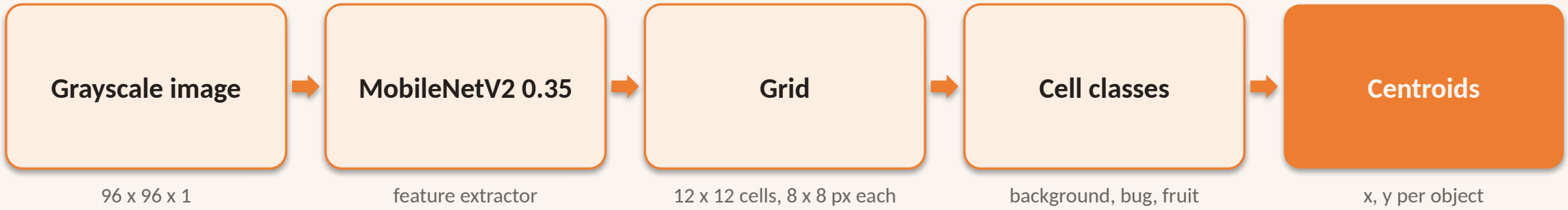
FOMO · FASTER OBJECTS, MORE OBJECTS

Drop the box, keep the center. Up to 30 times less memory and energy than SSD or YOLOv5.

Launched by Edge Impulse in 2022. It runs on microcontrollers with less than 200 KB of RAM.

HOW FOMO WORKS

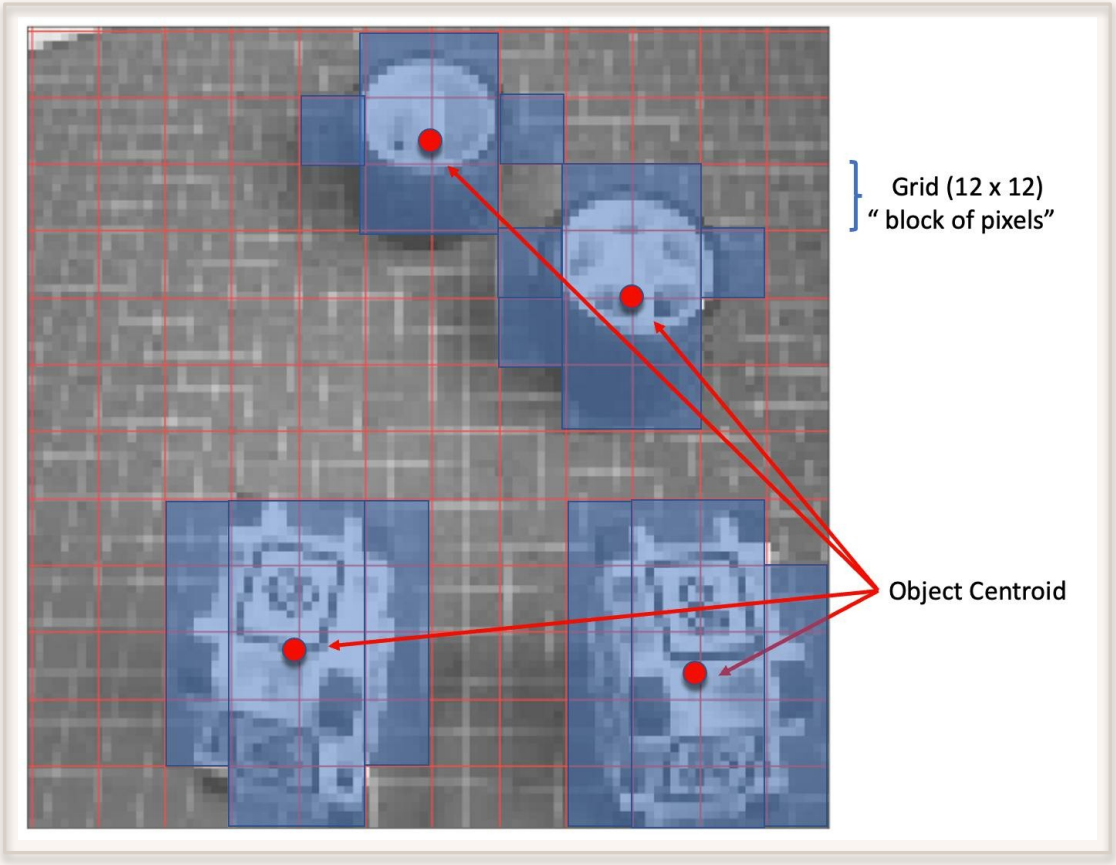
An image becomes a grid, and the grid becomes centroids



Every cell is a tiny classifier. Neighboring cells of the same class merge into one object.

THE GRID, DRAWN

Blue cells hold an object; the red dot is its centroid



1. The red lines are the 12 × 12 grid — each cell is one "block of pixels"
2. Blue cells are the ones the model classified as an object
3. The red dot is the center of each blue region — that is the output
4. Size is not reported. A large object only means more blue cells.

02

DATASET

Fruits, bugs, and the background

Fifty-eight pictures, several objects in each, captured with the board itself.

THE GOAL

Sort and count oranges and frogs on a line



1. Fruit: small oranges
2. Bug: plastic frogs
3. Background: everything else — FOMO adds this class by itself
4. We want the class, the position, and the count of every object

RAW DATA

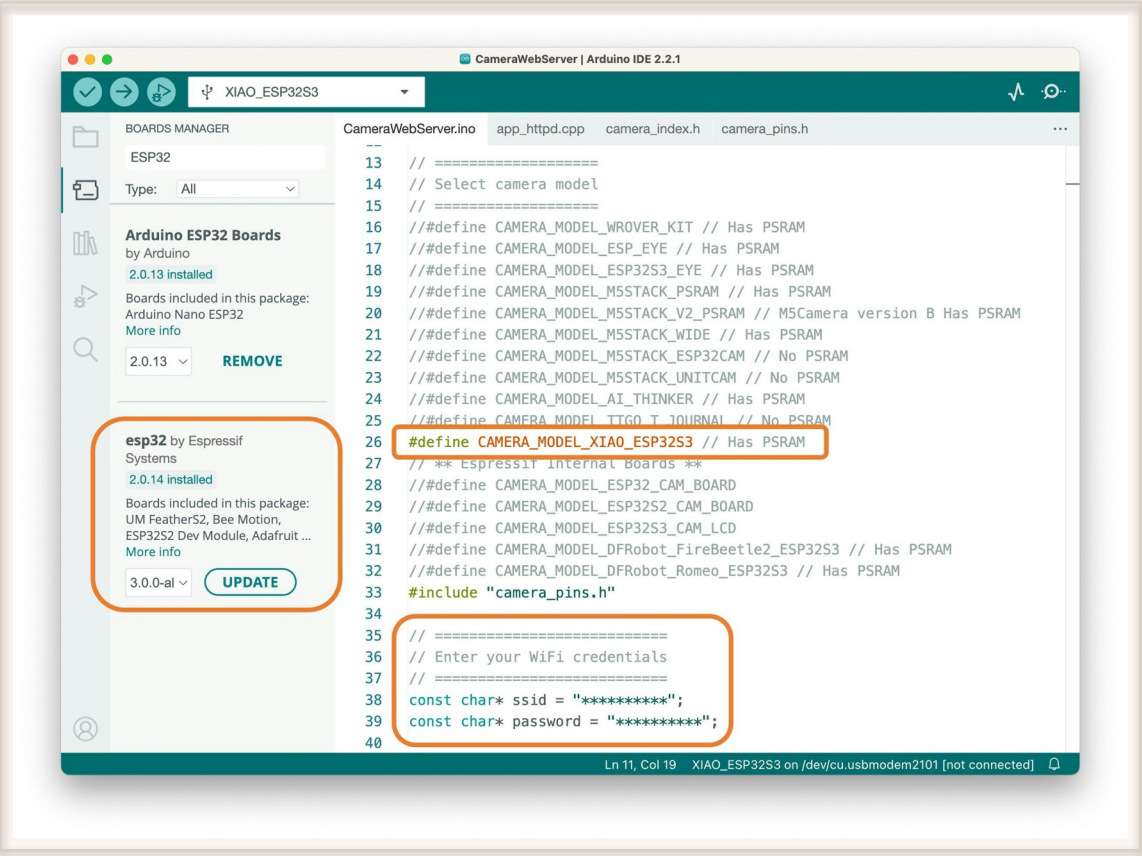
Several objects per image, so no class folders



Unlabeled QVGA frames from the XIAO. Each image holds fruits, bugs, or both, in different numbers.

CAPTURE · ARDUINO IDE

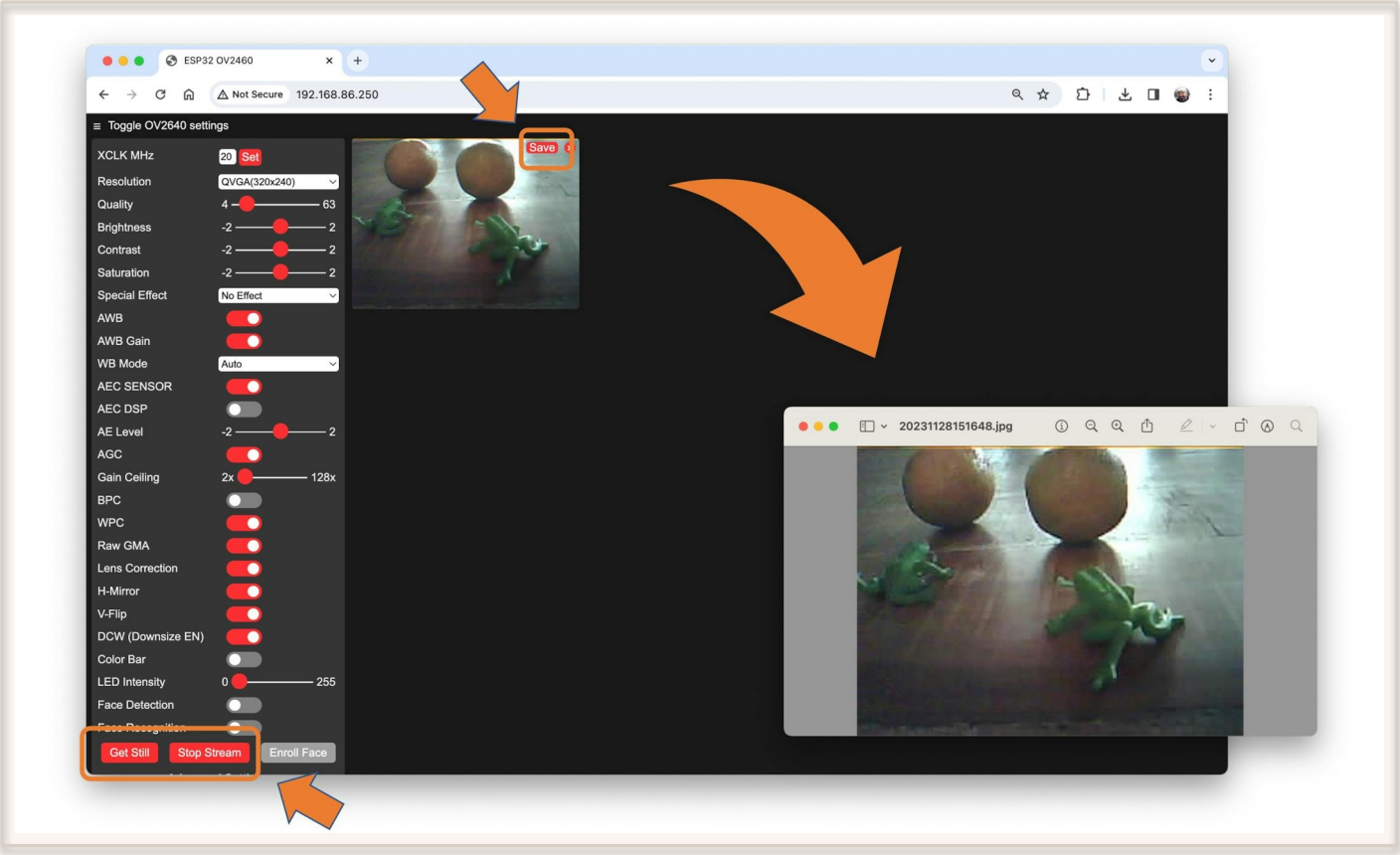
CameraWebServer turns the board into a camera you can save from



1. File › Examples › ESP32 › Camera › CameraWebServer
2. A stable esp32 core (2.0.x) — not a 3.x alpha
3. Keep only CAMERA_MODEL_XIAO_ESP32S3 uncommented
4. Enter your Wi-Fi credentials, enable PSRAM in Tools, and upload

CAPTURE · BROWSER

Open the address, start the stream, save the frames



1. The Serial Monitor prints the board's address — open it in a browser
2. Choose QVGA, 320 × 240, then Start Stream
3. Save stores the current frame in your Downloads folder
4. Stop Stream when you are done, and move the images to one folder

CAPTURE RULES

About fifty images, and make each one different

01

Mix the scene

Vary how many fruits and bugs appear, and combine them in the same frame.

02

Vary the conditions

Different angles, backgrounds, and light. The model should not learn the table.

03

Mind size and overlap

FOMO prefers similar sizes and no overlap — easy with a fixed camera. We break it a little on purpose.

EDGE IMPULSE STUDIO

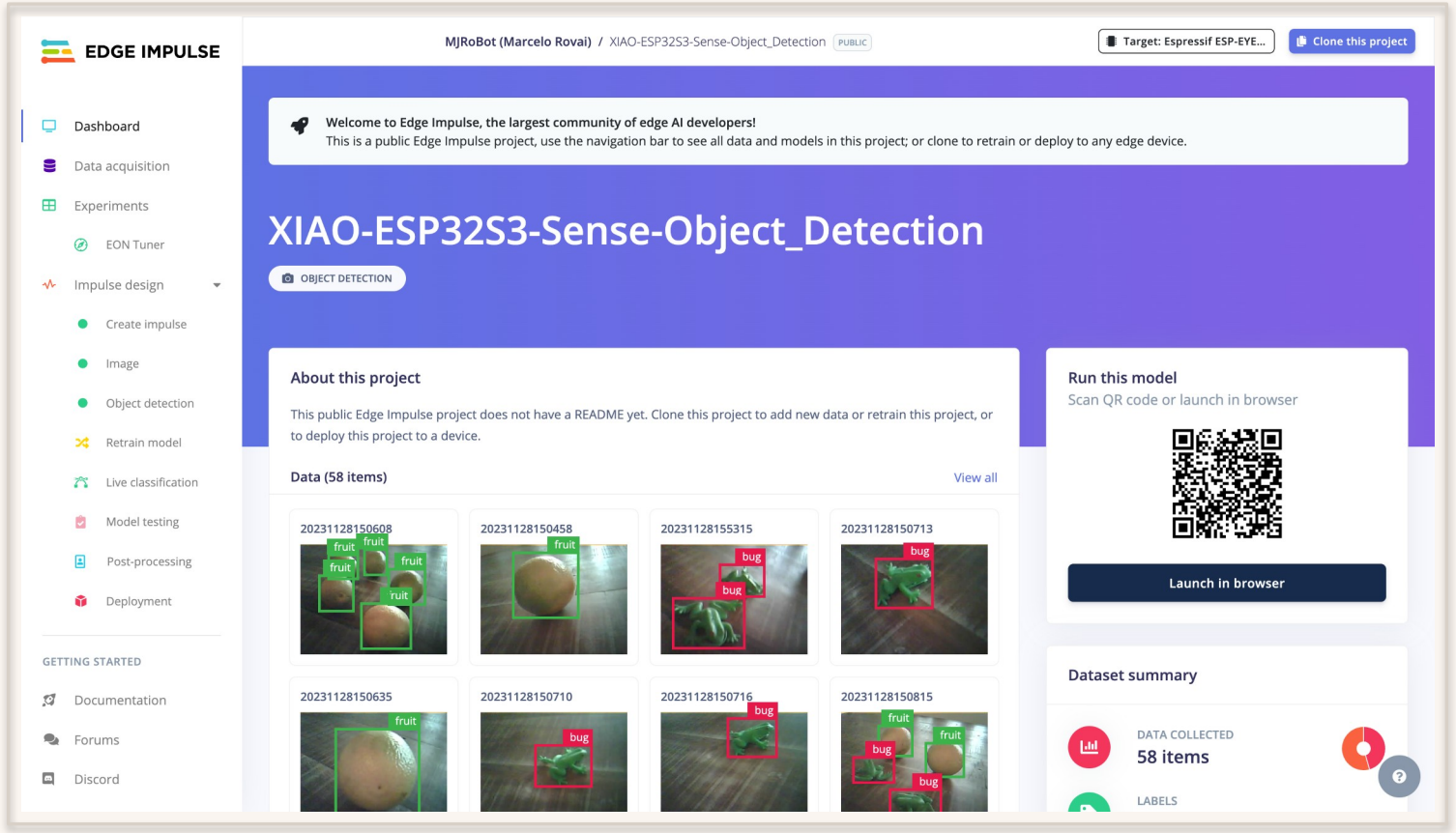
03

Label, train, and test

Boxes go in, centroids come out.

THE PROJECT

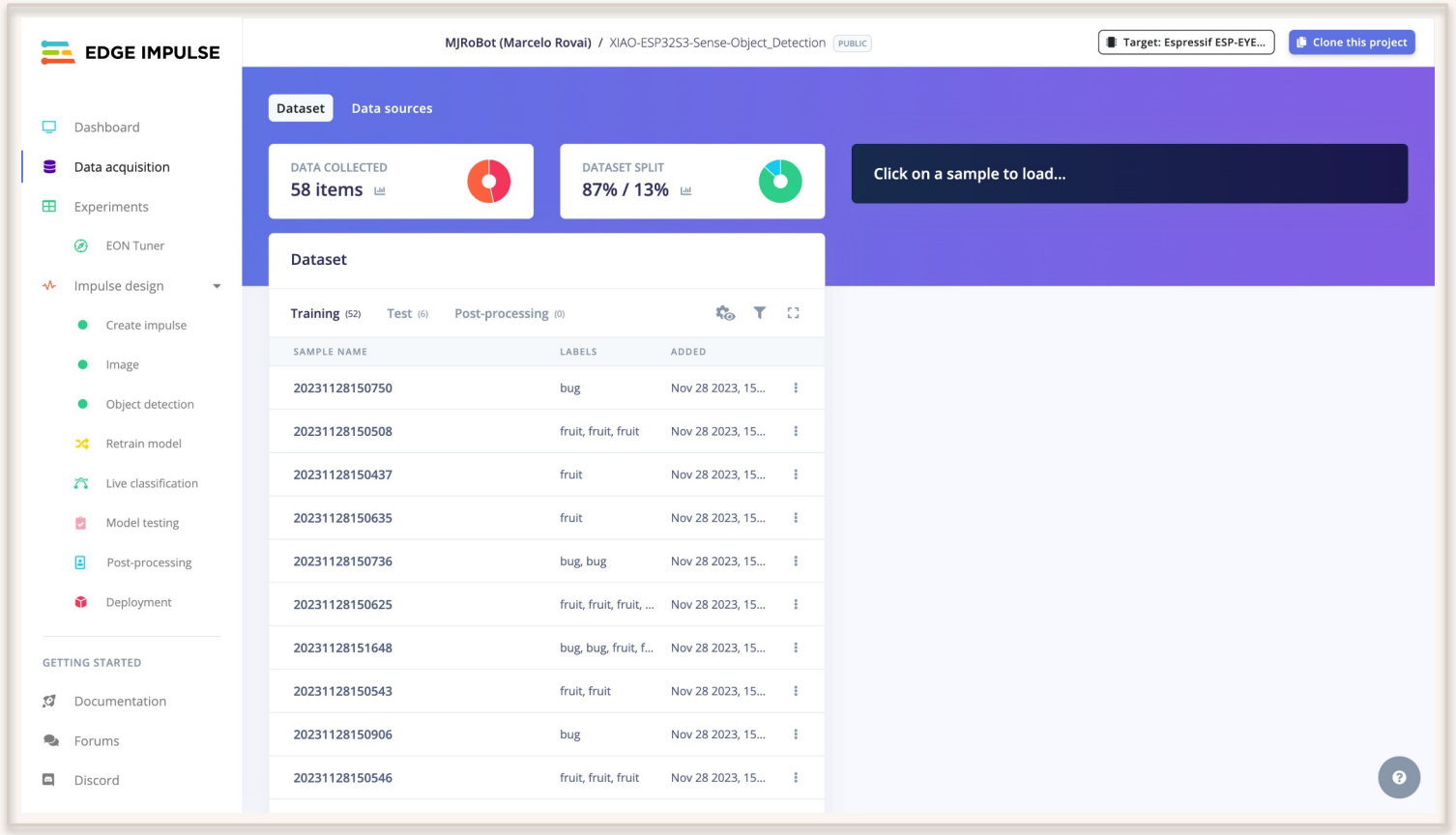
A public project you can clone, targeting the ESP-EYE



1. XIAO-ESP32S3-Sense-Object_Detection — public project 315759
2. Labeling method set to bounding boxes (object detection)
3. Target: Espressif ESP-EYE, the closest device to our board
4. 58 items, already labeled with fruits and bugs

DATA ACQUISITION

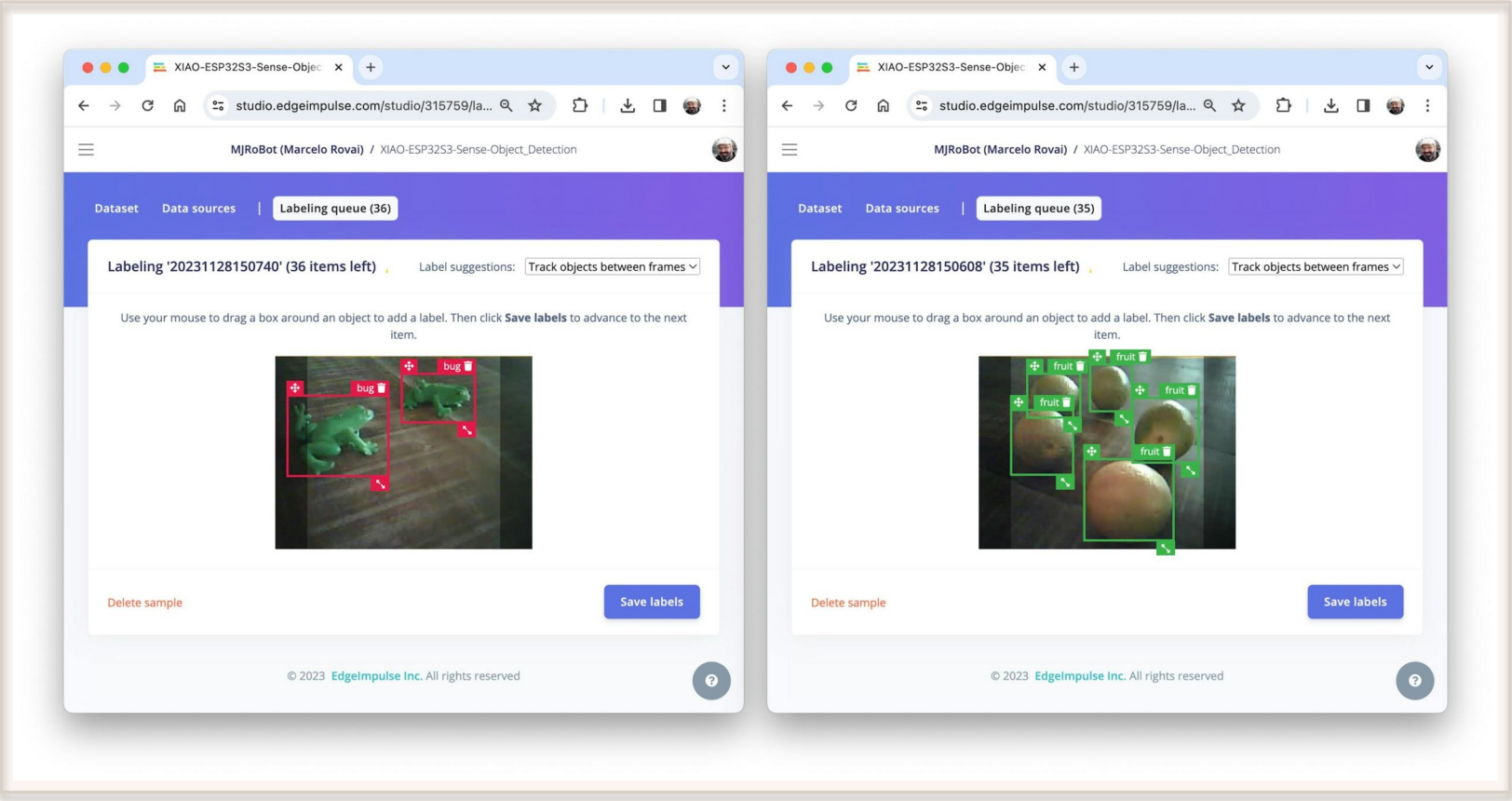
Upload everything as training data, then label



1. The first upload was 47 images, all unlabeled
2. They wait in the labeling queue until each object has a box
3. Each row lists the labels found in that image, like "fruit, fruit, bug"

LABELING

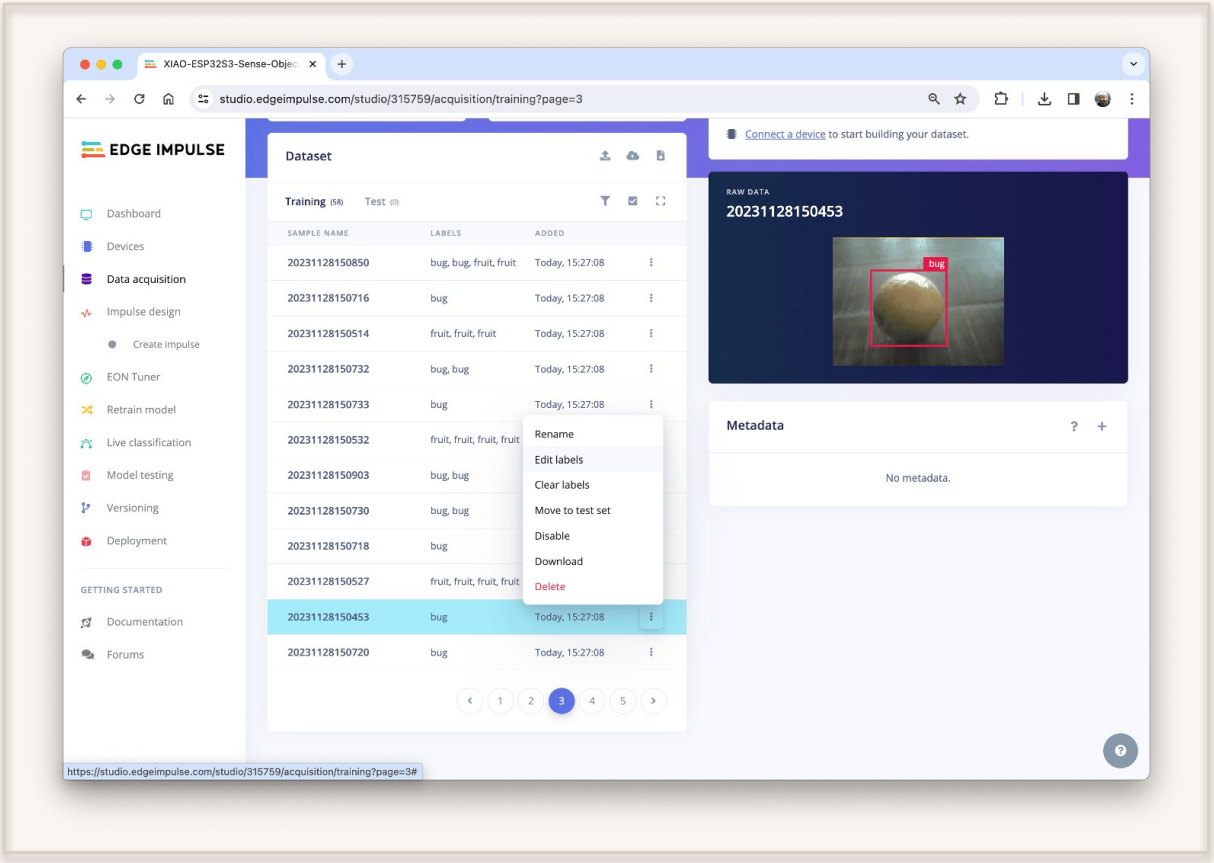
Draw a box once, and let tracking do part of the work



1. Drag a box around each object, type its label, then Save labels
2. Label suggestions: "Track objects between frames"
3. YOLOv5 suggestions only help with COCO objects — not plastic frogs
4. Tracking labels the next image partially — always check it

REVIEW

Fix a wrong label from the three-dot menu



1. Open the three-dot menu at the end of the sample's row
2. Edit labels opens the image with its boxes
3. The same menu moves a sample to the test set

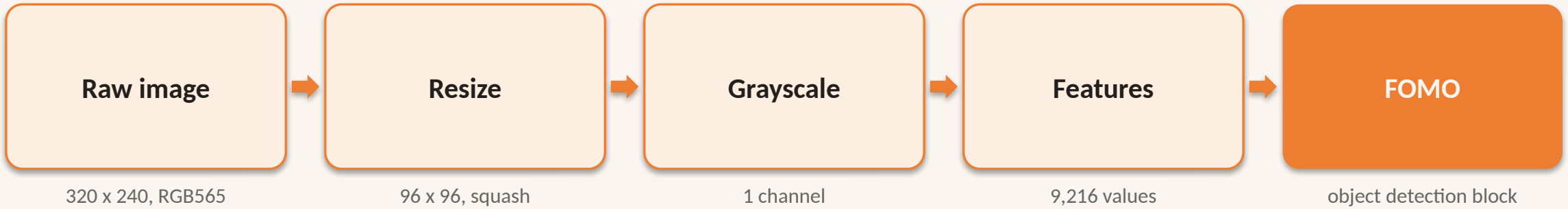
BALANCE AND SPLIT

Fruits outnumbered bugs. Eleven more bug pictures, 58 in total, and 6 held back for testing.

The Studio reports the split as 87 % train, 13 % test. Count objects, not only images.

THE IMPULSE

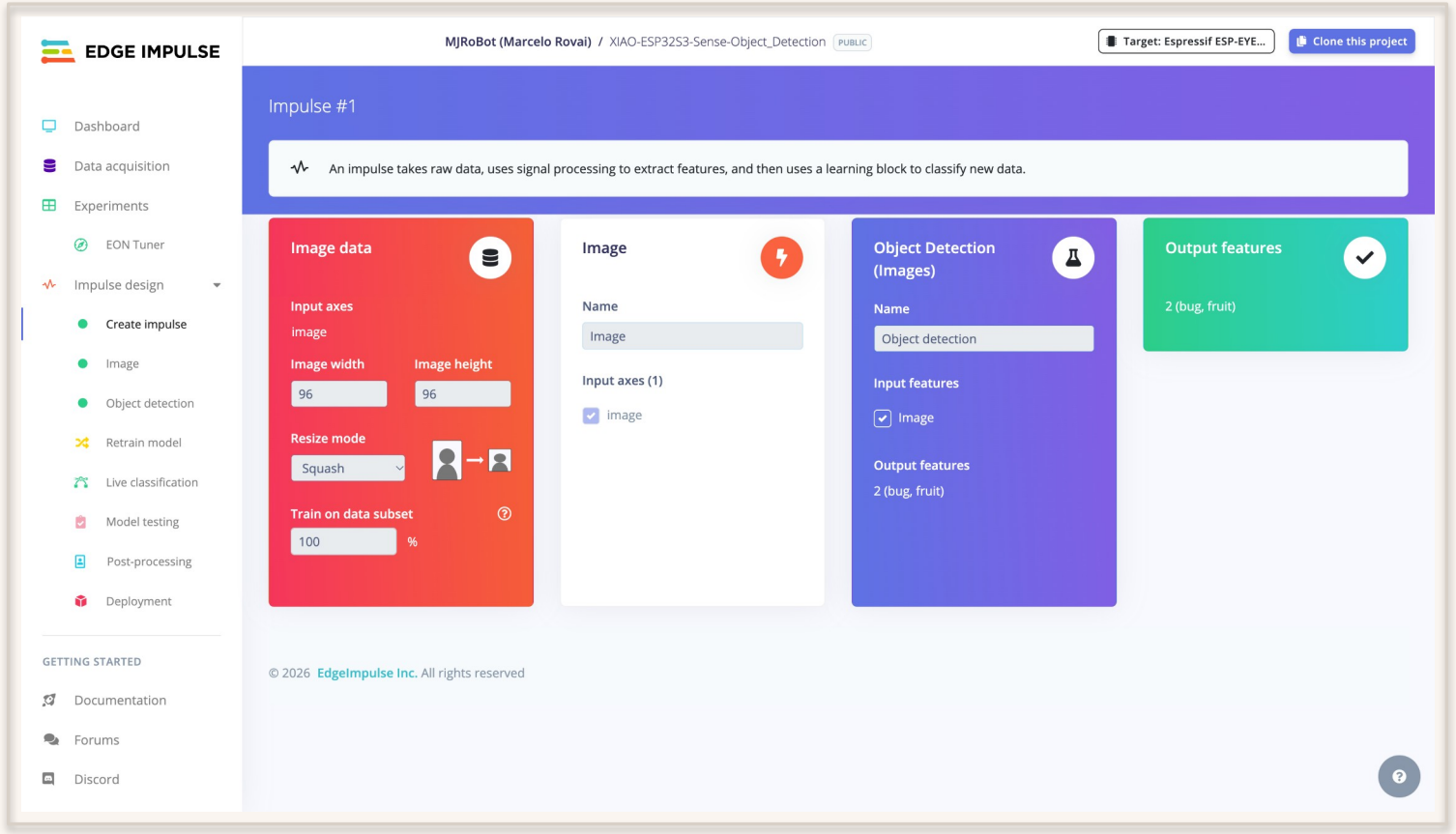
From a QVGA frame to 9,216 features



Squash, not crop — objects near the edges stay in the frame.

CREATE IMPULSE

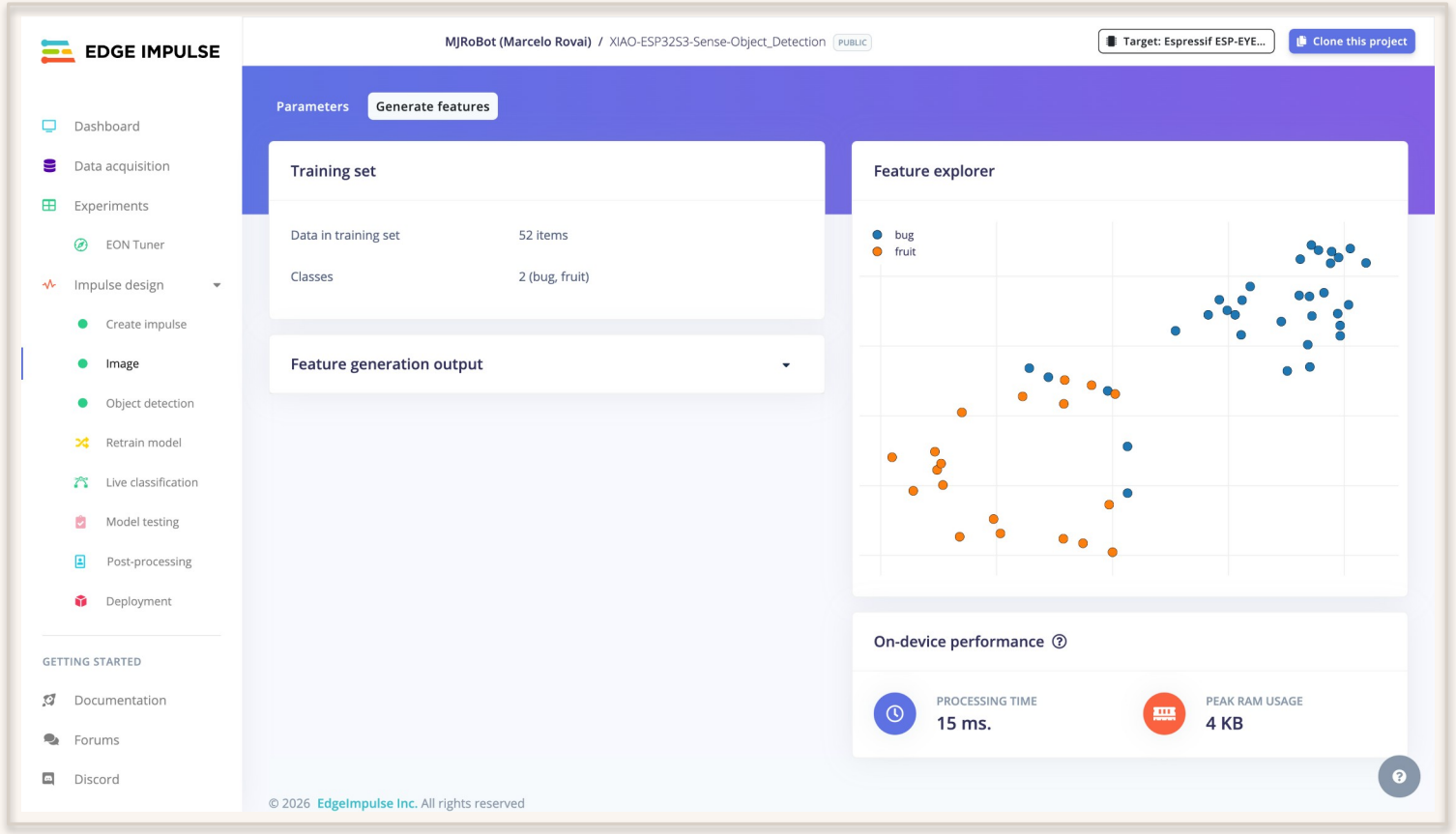
Image data, an image block, and an object detection block



1. Image data at 96 × 96, resize mode Squash
2. The Image processing block
3. Object Detection (Images) as the learning block
4. Output features: 2 — bug and fruit

FEATURES

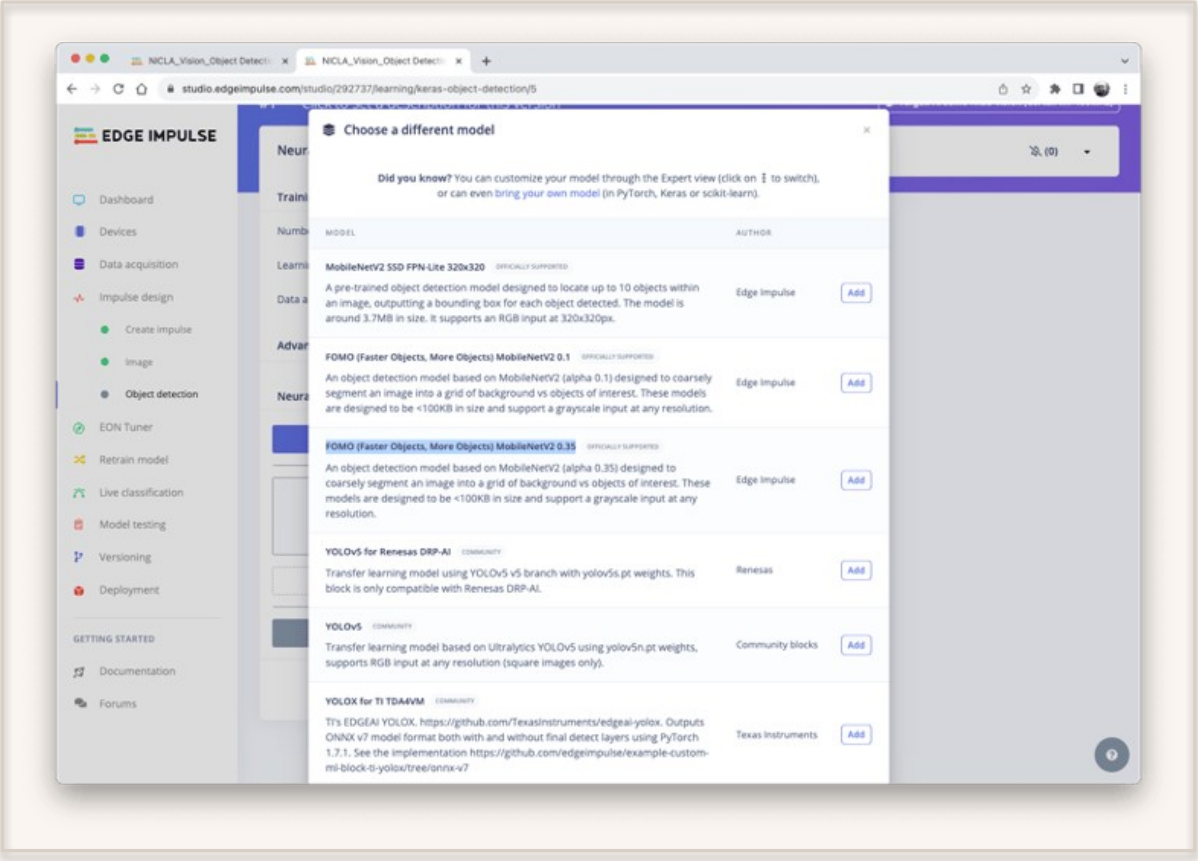
Grayscale features, and two classes that already separate



1. Color depth set to Grayscale in Parameters, then Generate features
2. 52 training items, 2 classes
3. Bugs and fruits form two clusters, with a few points in between
4. Pre-processing cost on the target: 15 ms and 4 KB of RAM

CHOOSE A MODEL

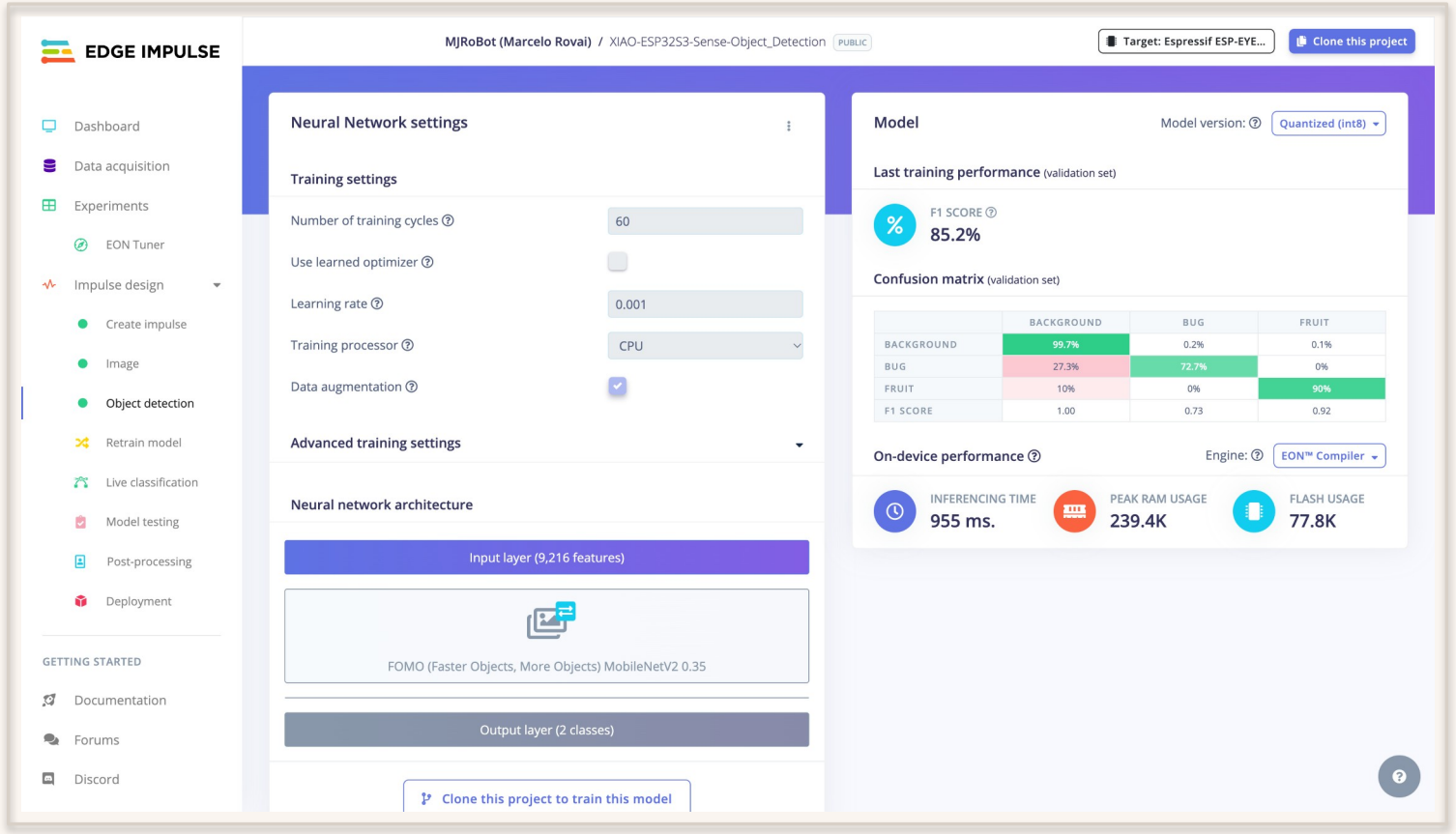
FOMO MobileNetV2 0.35, not a 3.7 MB SSD



1. MobileNetV2 SSD FPN-Lite 320 × 320 — about 3.7 MB, RGB only
2. FOMO MobileNetV2 0.1 — the smallest option
3. FOMO MobileNetV2 0.35 — our choice, about 250 KB of RAM and 80 KB of flash

TRAINING

60 epochs, and an F1 score of 85 %



1. 60 training cycles, learning rate 0.001, augmentation on, 20 % validation
2. Input 9,216 features → FOMO MobileNetV2 0.35 → 2 classes (+ background)
3. F1 score 85.2 % on the validation set
4. Estimate for the ESP-EYE: 955 ms, 239.4 KB of RAM, 77.8 KB of flash

READ THE CONFUSION MATRIX

Bugs disappear into the background 27 % of the time

ACTUAL ↓ / PREDICTED →	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

Validation set. Bugs and fruits are never confused with each other — the errors are all missed objects.

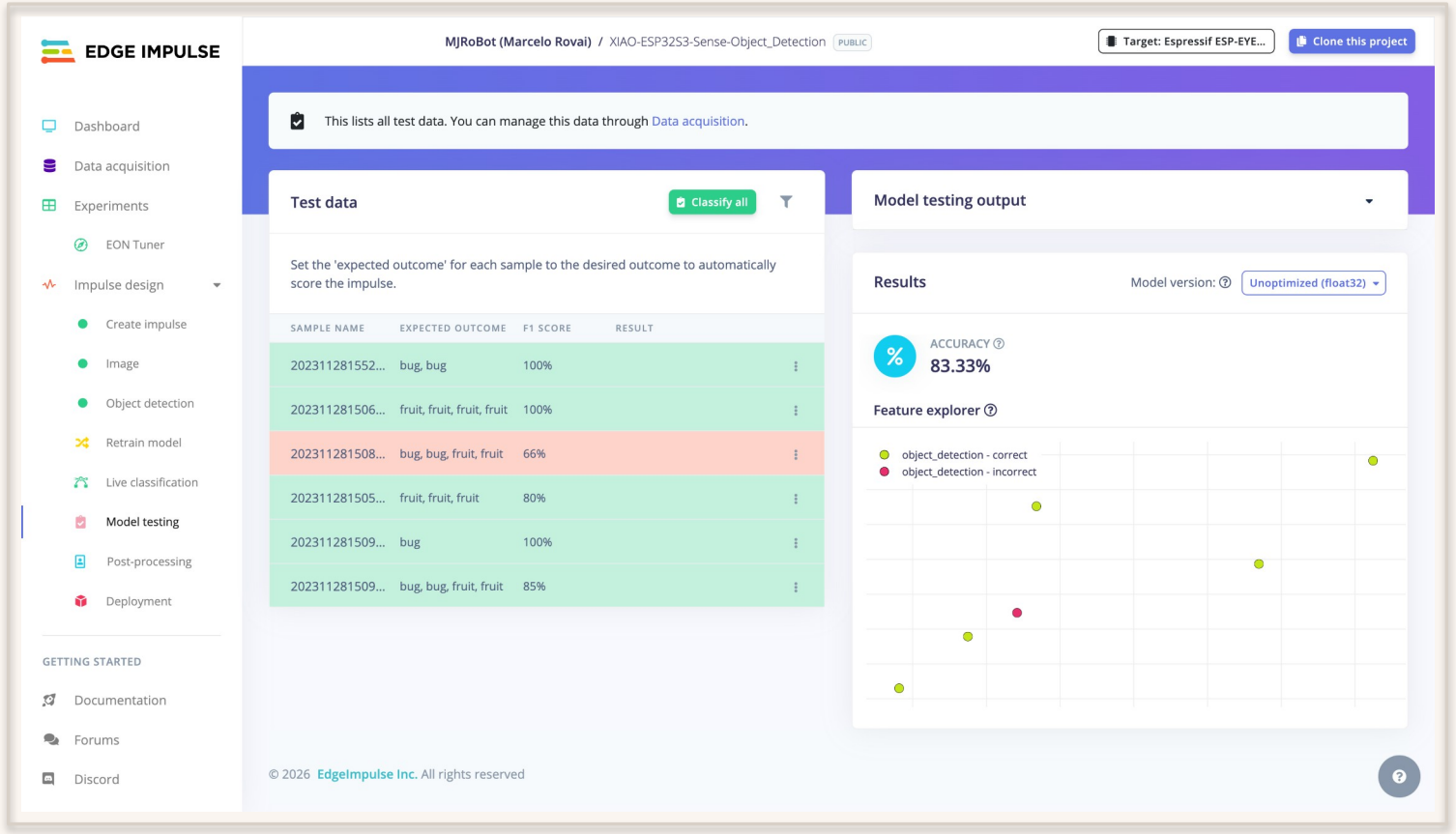
WHY F1, NOT ACCURACY

Most of the 144 cells are background. A model that never finds anything still looks accurate.

F1 combines precision and recall per class, so missed objects and false alarms both cost.

MODEL TESTING

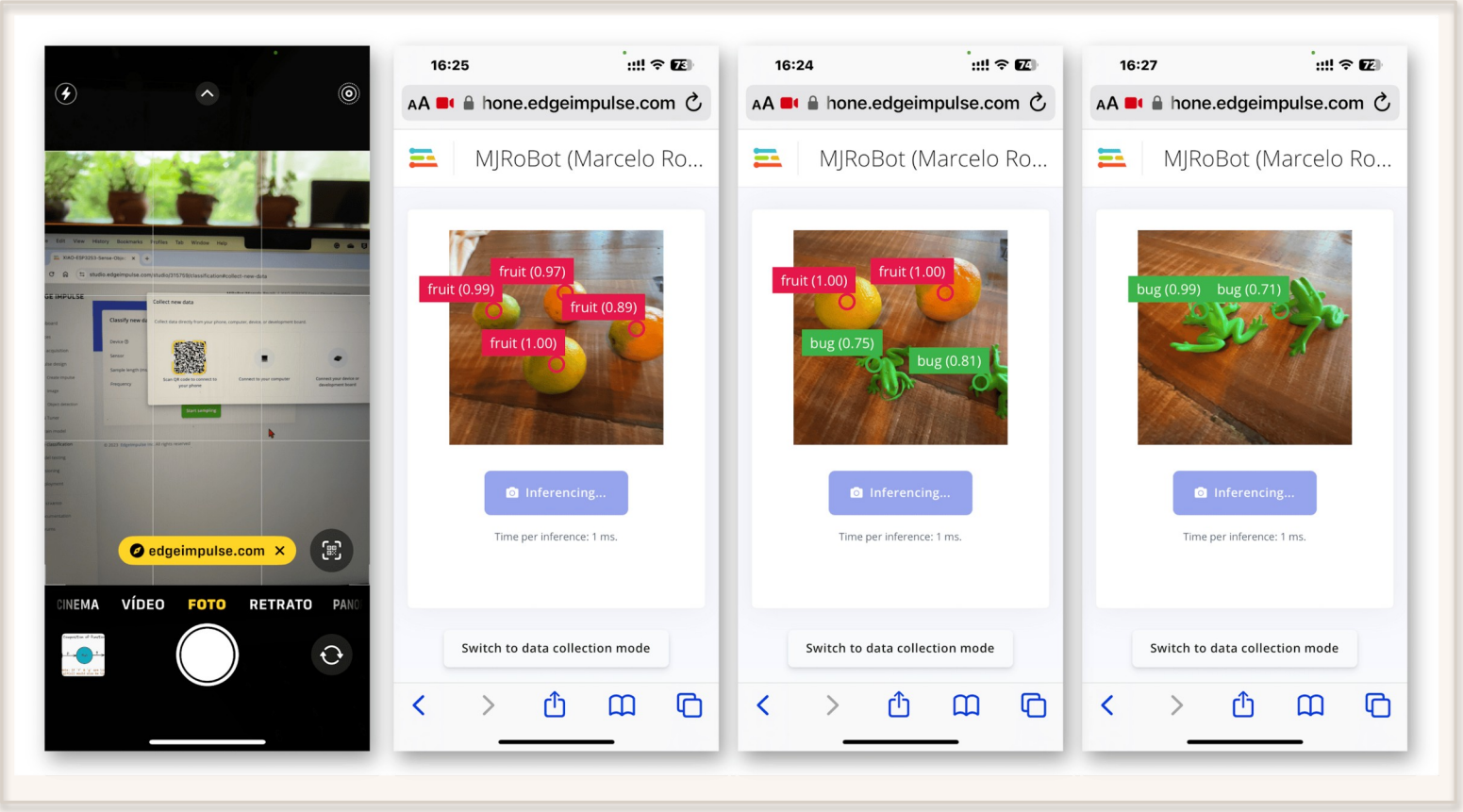
83 % on six images the model never saw



1. Six test images, each with several objects
2. Per-image F1 from 66 % to 100 % — the 66 % one holds two bugs and two fruits
3. Overall 83.33 %, close to the 85 % on validation

LIVE CLASSIFICATION

Test with your phone before you flash anything



1. Scan the QR code in Live classification with your phone
2. Take a photo, and each detected object is marked with its class and score
3. False positives? Raise the confidence threshold to 0.8 or more

04

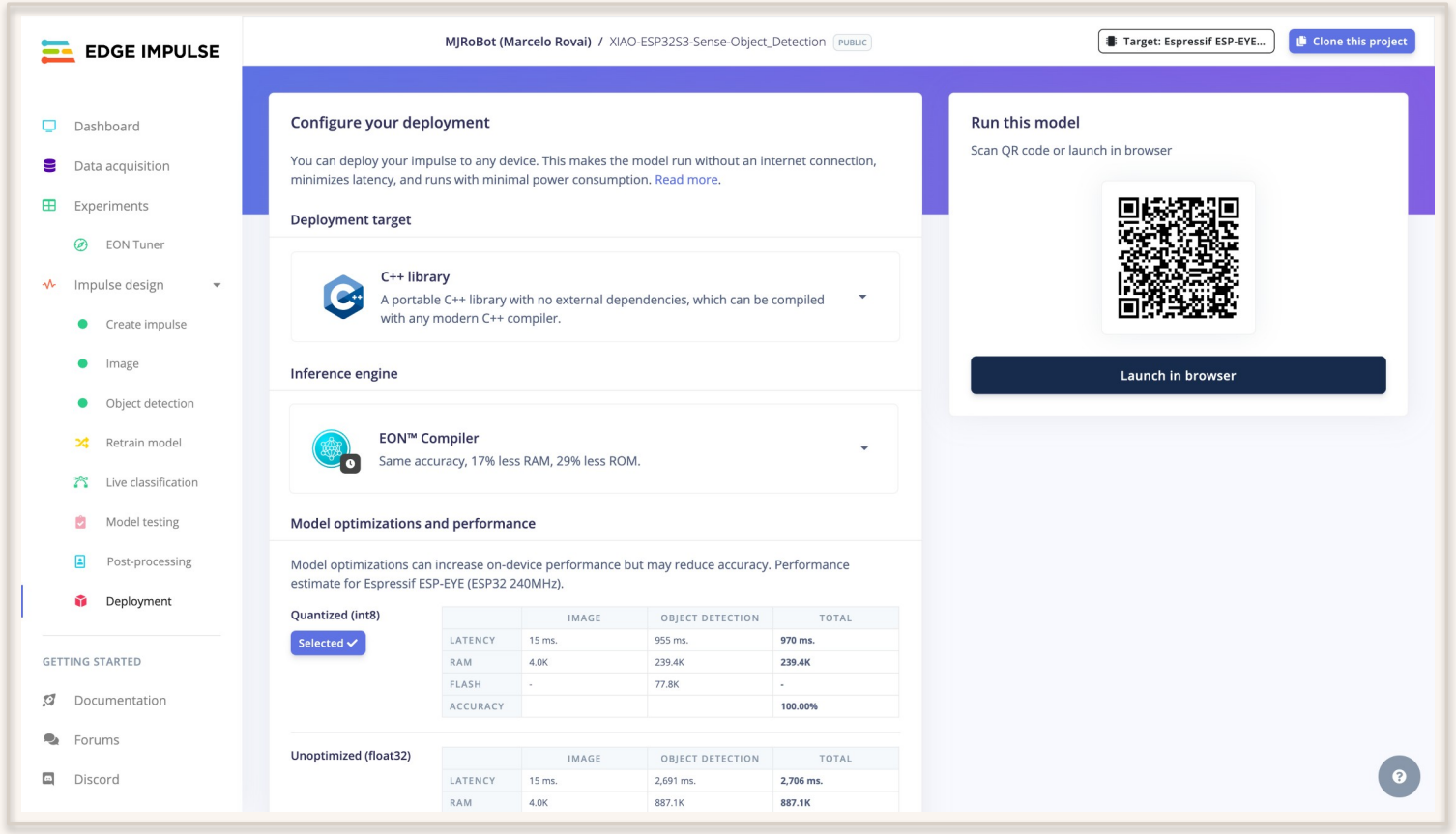
DEPLOY AND RUN

Two ways onto the board

An Arduino library you control, and SenseCraft so you can see what the camera sees.

DEPLOYMENT

Arduino library, quantized int8, EON Compiler on



1. Search for and select Arduino library as the deployment target
2. Keep the EON Compiler on — less RAM and less flash, same accuracy
3. int8: 970 ms and 239.4 KB of RAM — float32: 2,706 ms and 887.1 KB
4. Build, and a .zip library downloads

READ THE PROFILER

The Studio estimates 955 ms for the model. On the XIAO, it runs in 143 ms.

The estimate is for an ESP-EYE, an older ESP32. The ESP32-S3 has vector instructions the optimized kernels use.

THE SKETCH

Replace the camera pins with the XIAO's

esp32_camera.ino — replaces lines 32 to 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
```

— Add the .zip with Sketch › Include Library › Add .ZIP Library.

— Open Examples › your project › esp32 › esp32_camera.

— The example ships with ESP-EYE pins. These are the XIAO ESP32S3 Sense pins.

SERIAL MONITOR

4 ms to prepare, 143 ms to detect

```
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: 64, width: 16, height: 16 ]
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 ]
```

DSP

4 ms

MODEL

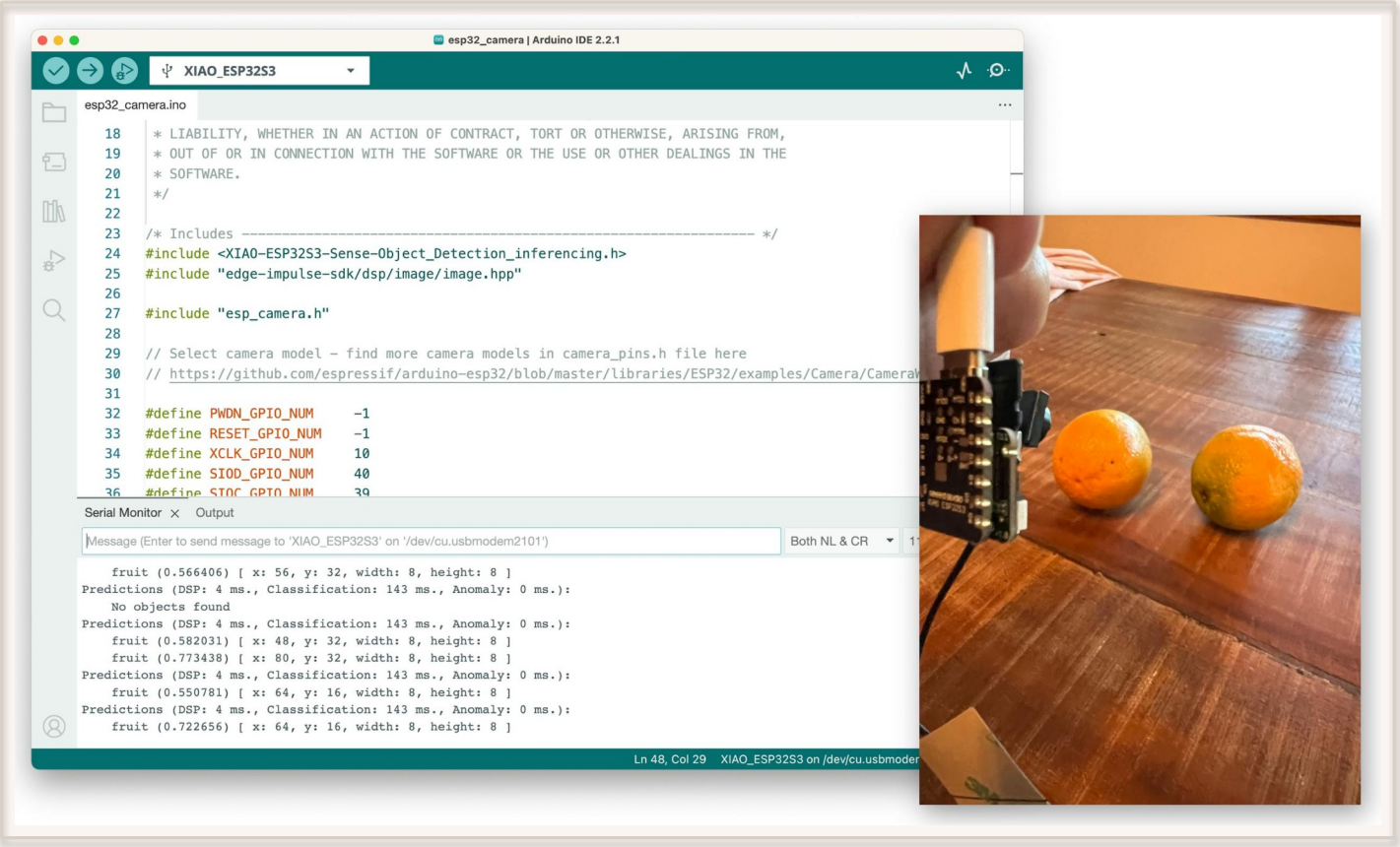
143 ms

ON PAPER

~6.8 FPS

READ THE OUTPUT

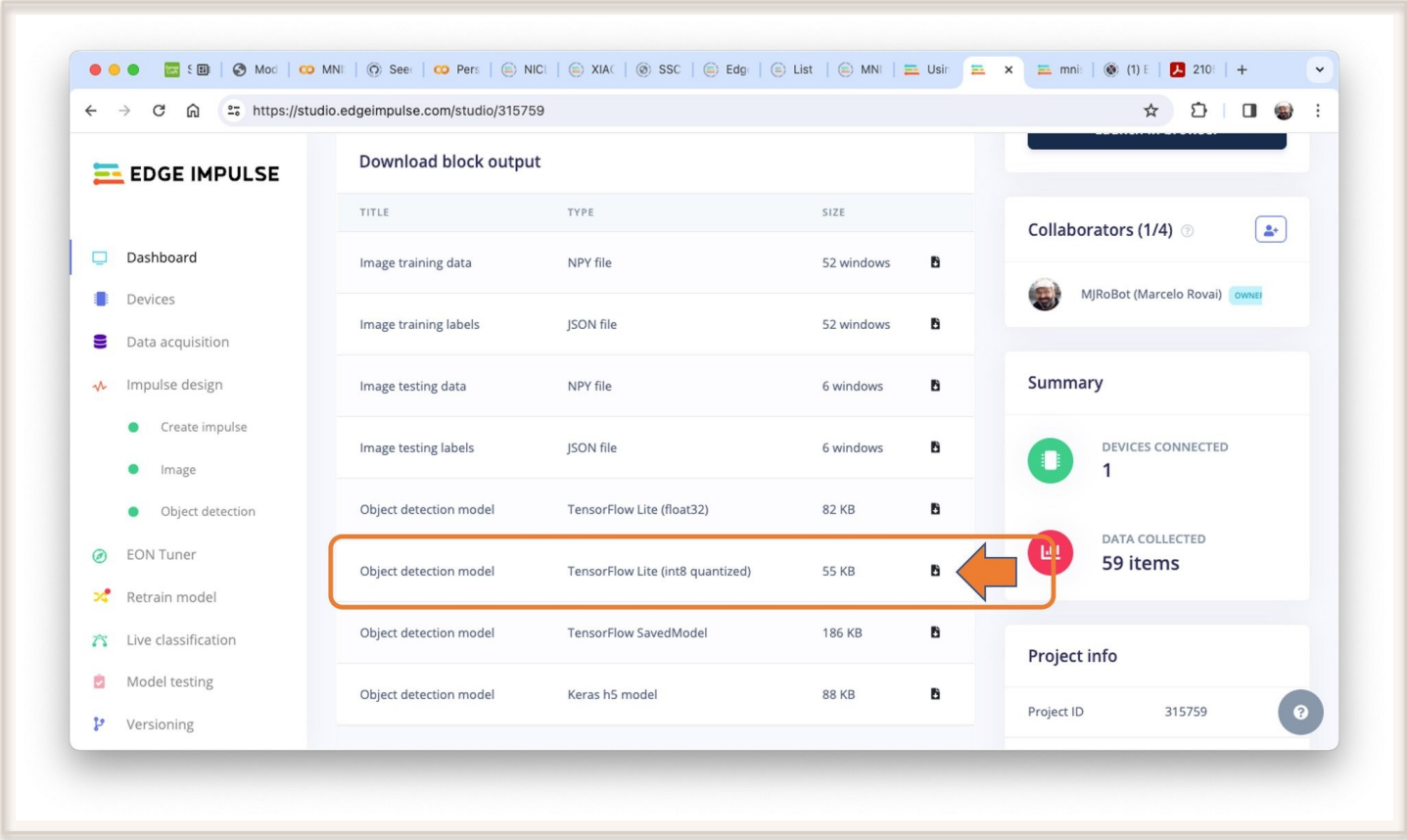
Each line is a class, a score, and a cell, not a box



1. Two oranges in view, two fruit lines in the same prediction
2. x and y are in 96 × 96 input pixels, on multiples of 8 — the grid
3. Width and height of 8 means one cell, not the object's size
4. Count the lines and you have the count of objects

SENSECRAFT · UPLOAD

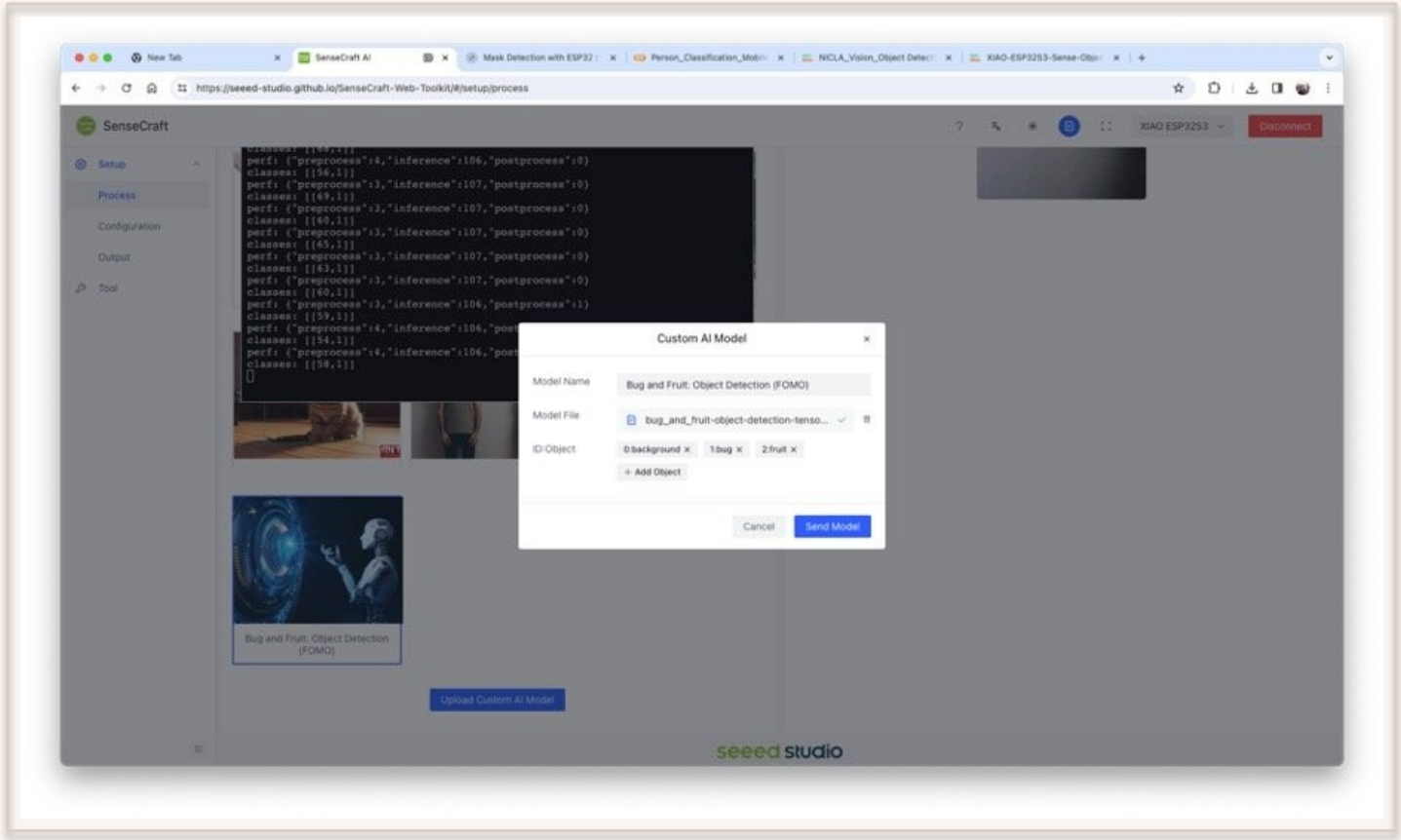
Download the int8 .tflite, then upload a custom model



1. Dashboard › Download block output
2. Object detection model, TensorFlow Lite (int8 quantized) — 55 KB
3. The float32 version is 82 KB; the board gets the int8 one

SENSECRAFT · CUSTOM MODEL

Connect, then enter the labels in alphabetical order

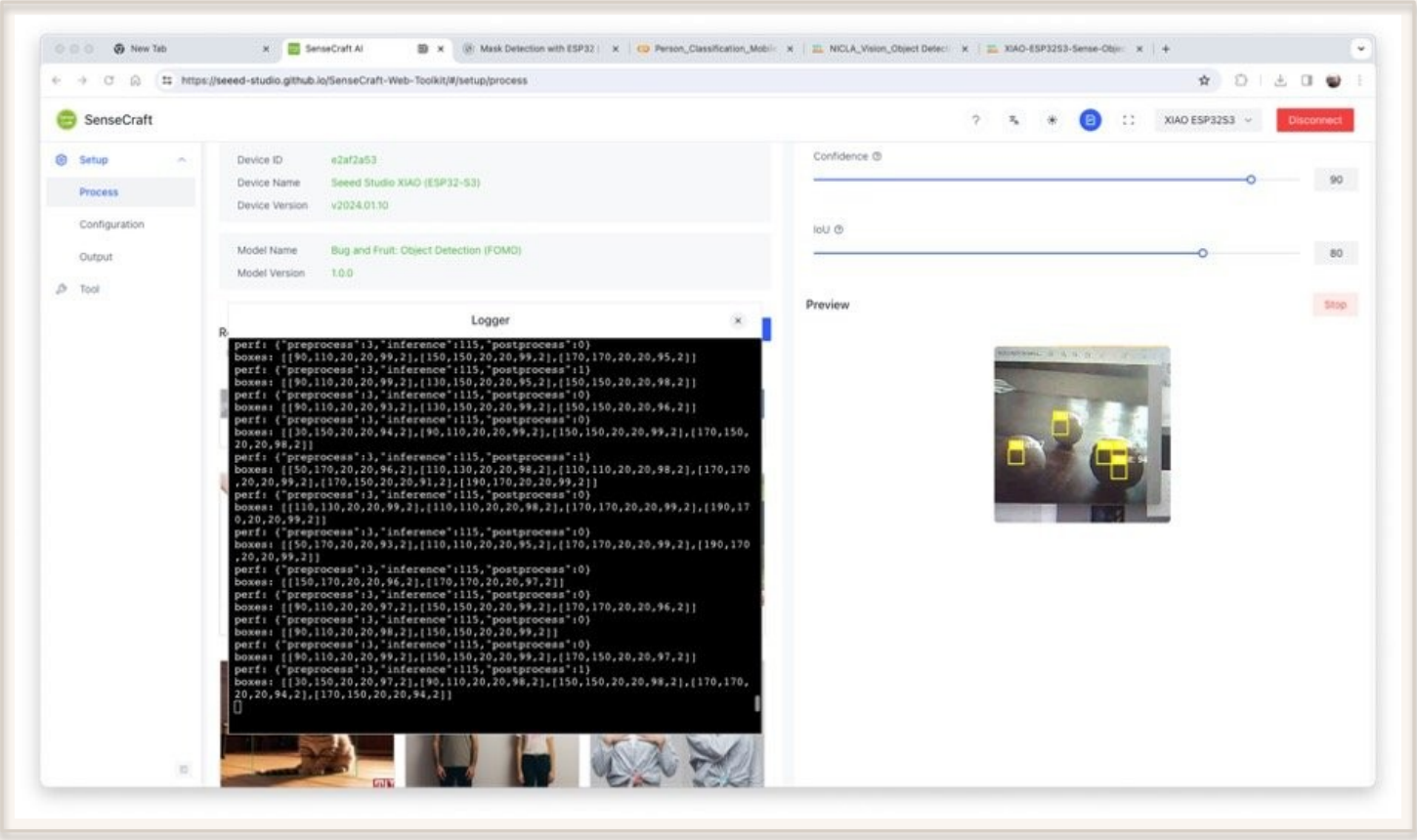


1. Connect the XIAO ESP32S3 in Chrome, and pick its serial port
2. Upload Custom AI Model, at the bottom of the page
3. Choose the .tflite file and give the model a name
4. Labels by ID, in alphabetical order: 0 background, 1 bug, 2 fruit

The book used the SenseCraft Web Toolkit. Today it is part of SenseCraft AI, the workspace from Chapters 1 and 2 — screens may differ.

SENSECRAFT · PREVIEW

Live preview, centroids, and two sliders



1. The preview marks each detected centroid on the live image
2. Confidence sets the minimum score to report an object
3. IoU decides how much two boxes must overlap to count as one
4. Device Log opens the same stream of results as a serial monitor

DEVICE LOG

3 + 115 + 1 milliseconds, and five boxes

```
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]]
```

PRE-PROCESS

3 ms

INFERENCE

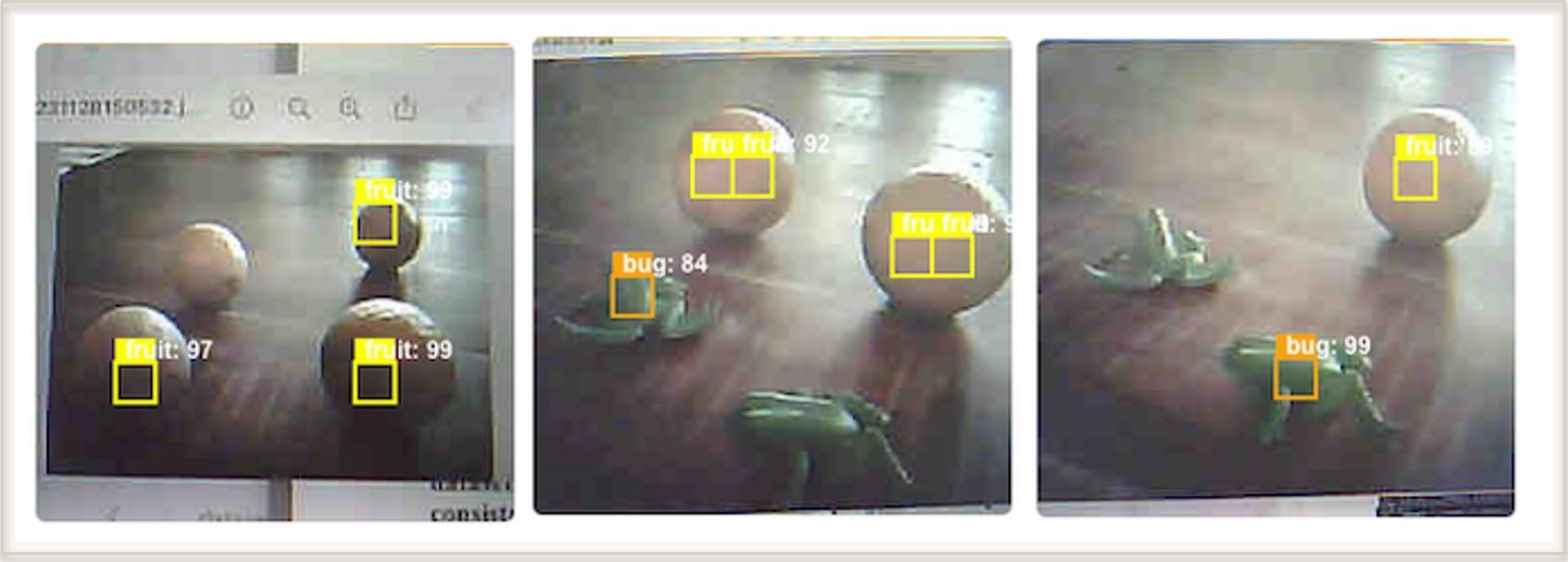
115 ms

BOX

x, y, w, h, score, class

POST-PROCESSING

More boxes than oranges, so merge neighbors



1. A large object can light up several neighboring cells
2. Each region becomes its own centroid, so one orange counts twice
3. Merge centroids closer than a set distance, then count

THE BUDGET

A detector that is faster than our classifier

CONFIGURATION	INPUT	PRE-PROCESS	MODEL	OUTPUT
Ch. 2 classifier, XIAO (measured)	96^2, RGB	4 ms	199 ms	1 label
FOMO, Studio estimate (ESP-EYE)	96^2, gray	15 ms	955 ms	—
FOMO, XIAO, Arduino	96^2, gray	4 ms	143 ms	centroids
FOMO, XIAO, SenseCraft	96^2, gray	3 ms	115 ms	centroids
SSD FPN-Lite, Raspberry Pi 4	320^2, RGB	—	~5× FOMO	boxes, ~1.5 FPS

Same board, same input size. One channel instead of three, and an early cut of MobileNet, make FOMO the cheaper model.

SUMMARY

Three things to remember

01 FOMO trades size for speed

A 12 × 12 grid of tiny classifiers gives a class and a centroid per object, but no size.

02 Labels are boxes, outputs are cells

You draw boxes to train; the board answers in 8-pixel cells. Merge neighbors before you count.

03 Read F1 per class

85 % overall hid a bug class at 73 %. Every error was a missed object, not a confusion.

CHECK YOURSELF

The board prints "fruit [x: 80, y: 32, width: 8, height: 8]". What does width: 8 tell you?

A The orange is 8 pixels wide in the 96 × 96 image

B The detection covers one 8 × 8 cell of the grid — FOMO does not measure size

C The orange is 8 % of the frame width

FOMO classifies cells of 8 × 8 pixels. The width and height describe the cell region around the centroid, not the object.

The interactive deck gives per-answer feedback.

Questions?

Next — Keyword Spotting, where the sensor is a microphone and the image is a spectrogram.

