# THE EVOLUTION OF OBJECT DETECTION AND ITS CHALLENGES

**INTRODUCTION TO OBJECT DETECTION**

Object detection has been a cornerstone of computer vision, addressing the need to recognize and locate multiple objects within an image or video frame. Its applications span various industries, including **autonomous systems, security, healthcare, manufacturing, and smart surveillance**. Over time, object detection techniques have evolved from basic image processing to sophisticated deep-learning-based models, yet several challenges persist in real-time implementation, particularly on low-power, cost-effective hardware.

**THE EVOLUTION OF OBJECT DETECTION**

**Early Object Detection Techniques (Pre-Deep Learning Era)**

Initial object detection systems relied on **basic image processing algorithms** such as **edge detection, colour segmentation, and contour detection**.

Features were manually extracted using methods like **Histogram of Oriented Gradients (HOG)** and **Scale-Invariant Feature Transform (SIFT)** to differentiate objects.

These methods were computationally **lightweight** but **lacked accuracy** when dealing with cluttered backgrounds, varying lighting conditions, and occlusions.

**The Shift to Machine Learning-Based Approaches**

Machine learning introduced **feature extraction combined with classifiers** like **Support Vector Machines (SMS)** and **Random Forests** to improve object recognition.

These methods were more adaptable than traditional rule-based systems but required extensive feature engineering and struggled with real-time applications.

**The Deep Learning Revolution**

The rise of **Convolutional Neural Networks (CNN’s)** provided a breakthrough in feature extraction, enabling automatic learning of object characteristics.

Models like **R-CNN, Fast R-CNN, and Faster R-CNN** improved accuracy but demanded high computational power, making them impractical for real-time, embedded applications.

Despite advancements, most deep learning models require **GPUs, cloud processing, or expensive edge devices**, limiting their widespread adoption in low-power systems.

**EXISTING SOLUTIONS AND THEIR LIMITATIONS**

**High-End Vision Systems (Industrial & Autonomous Vehicles)**

**Example:** LiDAR-based object detection for self-driving cars.

**Problem:** Expensive sensors, power-hungry computation, and complex calibration requirements.

**Multiple Camera Setups (Surveillance & Security Systems)**

**Example:** AI-powered CCTV networks with multiple camera feeds.

**Problem:** High processing load, network bandwidth requirements, and limited accessibility in low-resource environments.

**Cloud-Based Object Detection (Retail, Smart Cities, and IoT Applications)**

**Example:** Retail checkout systems using cloud AI for product identification.

**Problem:** Latency issues, data privacy concerns, and dependency on internet connectivity.

**Low-Cost Microcontroller-Based Object Detection (Embedded & IoT Systems)**

**Example:** ESP 8266 or Arduino-based image processing modules.

**Problem:** Limited processing power, lower accuracy, and restricted to simple detection tasks.

**WHY IS A NEW APPROACH NEEDED?**

Existing object detection methods often fail to provide a **balanced solution** in terms of cost, power efficiency, and real-time processing. The challenge is to develop a **lightweight, standalone system** that can:  
✔ Perform **real-time object detection** without cloud dependence.  
✔ Use **minimal hardware resources** while maintaining accuracy.  
✔ Be **cost-effective and easy to implement** in various industries.

**WHERE THIS PROJECT CAN BE APPLIED?**

✅ **Smart Surveillance & Security** – Detecting unauthorized access or suspicious activities in restricted areas.  
✅ **Autonomous Robots & Drones** – Enabling obstacle detection and navigation for self-operating systems.  
✅ **Industrial Automation** – Monitoring product defects and ensuring quality control in manufacturing lines.  
✅ **Healthcare & Assistive Technologies** – Helping visually impaired individuals by identifying objects in real time.  
✅ **Agriculture & Smart Farming** – Identifying crops, monitoring livestock, and detecting pests using embedded AI solutions.  
✅ **Traffic Monitoring & Smart Cities** – Enhancing road safety by detecting vehicles and pedestrians in real-time.

**FUTURE ADVANCEMENTS & POSSIBILITIES**

🚀 **Edge AI Integration** – More powerful AI chips like **Google Coral** and **NVIDIA Jetson** can enhance performance.  
🚀 **OG & IoT Connectivity** – Enabling faster, low-latency object detection across connected devices.  
🚀 **Thermal & Infrared Imaging** – Combining visual and heat-based object detection for security and healthcare.  
🚀 **Self-Learning Models** – AI that adapts to environments without needing extensive retraining.  
🚀 **Lightweight AI Models** – Optimized deep-learning models designed for low-power devices, reducing reliance on GPUs.

**REFERENCE PAPERS**

📌 **Fast and Accurate 3D Object Detection for LiDAR-Camera-Based Autonomous Vehicles Using One Shared Voxel-Based Backbone**  
🔗 [IEEE Xplore](https://ieeexplore.ieee.org/document/9340187/)

📌 **Optical Character Recognition (OCR) in Cursive Scripts Using Object Detection Networks**  
🔗 [Tabriz Journal of Electrical Engineering](https://tjee.tabrizu.ac.ir/article_18897.html?lang=en)

📌 **Enhancing Trustworthiness in Real-Time Single Object Detection**  
🔗 [Reference Not Available]

MICRO VISION KIT: AI-DRIVEN OBJECT DETECTION FOR IOT APPLICATIONS

**ABSTRACT:**

In the realm of embedded systems and Internet of Things (IoT), efficient and cost-effective real-time object detection remains a pivotal challenge. Traditional systems often rely on multiple cameras and extensive hardware setups, leading to increased costs and complexity. This paper presents a streamlined approach utilizing the ESP 32-CAM module integrated with the YOLO (You Only Look Once) object detection algorithm, facilitated by the Edge Impulse platform. This configuration offers a low-cost, single-sensor solution capable of rapid image processing and accurate detection of multiple objects in real-time. The proposed system demonstrates significant improvements in deployment simplicity, cost efficiency, and processing speed, making it ideal for various applications, including surveillance, automation, and IoT deployments.

**INTRODUCTION**

Object detection is a cornerstone of modern computer vision applications, enabling systems to identify and locate objects within an environment. Traditional approaches often involve complex hardware setups with multiple cameras and high computational requirements, leading to increased costs and system complexity. These systems face challenges such as latency in data processing, high power consumption, and difficulties in integrating multiple components. The need for a more efficient, cost-effective solution is evident, especially for applications requiring real-time processing and deployment in resource-constrained environments.

**PROBLEM STATEMENT**

Previous object detection systems have relied heavily on separate cameras and extensive hardware configurations, resulting in several challenges:

**High Costs:** Utilizing multiple cameras and associated hardware increases the overall system cost, making it less accessible for budget-constrained projects.

**Complexity in Integration:** The need to synchronize multiple components complicates system design and integration, leading to potential reliability issues.

**Latency in Processing:** Data transmission from multiple cameras to processing units introduces delays, hindering real-time performance.

**Increased Power Consumption:** Operating multiple devices escalates power requirements, posing challenges for battery-operated or energy-efficient systems.

These challenges necessitate a solution that simplifies hardware requirements, reduces costs, and enhances real-time processing capabilities.

**PROPOSED SOLUTION**

To address these challenges, we propose a novel approach that leverages the ESP 32-CAM module integrated with the YOLO object detection algorithm, utilizing the Edge Impulse platform for model training and deployment. This solution offers several advantages:

**Single-Sensor Integration:** The ESP 32-CAM combines a camera and processing unit into a single module, reducing hardware complexity and cost.

**Cost Efficiency:** The module is highly affordable, making it accessible for a wide range of applications and budgets.

**Real-Time Processing:** With onboard processing capabilities, the system can perform object detection with minimal latency, suitable for time-sensitive applications.

**Edge Impulse Platform:** Utilizing Edge Impulse streamlines the process of data collection, model training, and deployment, facilitating rapid development and iteration.

**SYSTEM ARCHITECTURE**

The proposed system architecture comprises the following components:

**ESP 32-CAM Module:** Serves as the primary hardware, capturing images and performing onboard processing.

**YOLO Algorithm:** A state-of-the-art object detection algorithm known for its speed and accuracy, adapted for deployment on the ESP 32-CAM.

**Edge Impulse Platform:** Provides tools for collecting training data, developing machine learning models, and deploying them to the ESP 32-CAM.

**IMPLEMENTATION DETAILS**

**DATA COLLECTION AND ANNOTATION**

Data collection is a critical step in training an effective object detection model. Utilizing the ESP 32-CAM's built-in camera, we capture images of target objects from various angles and under different lighting conditions to create a diverse dataset. These images are then uploaded to the Edge Impulse platform, where they are annotated with bounding boxes to label the objects of interest. This process ensures that the model learns to recognize objects accurately in diverse scenarios.

**MODEL TRAINING WITH EDGE IMPULSE**

The annotated dataset is used to train a custom YOLO-based object detection model on the Edge Impulse platform. The platform provides an intuitive interface for configuring the model architecture, selecting hyperparameters, and monitoring training progress. Transfer learning techniques are employed to fine-tune a pre-trained model, accelerating the training process and enhancing performance. Once trained, the model is optimized for deployment on the ESP 32-CAM's hardware constraints.

**DEPLOYMENT TO ESP 32-CAM**

Edge Impulse facilitates seamless deployment by generating firmware that integrates the trained model with the ESP 32-CAM. This firmware is flashed onto the device, enabling it to perform real-time object detection independently. The ESP 32-CAM processes the captured images, applies the model to detect objects, and outputs the results, such as drawing bounding boxes around detected objects and identifying their classes.

**PERFORMANCE EVALUATION**

To assess the system's effectiveness, we conducted experiments focusing on key performance metrics:

**Detection Accuracy:** The system demonstrated high accuracy in identifying and localizing multiple objects within the frame, with precision and recall rates comparable to more complex setups.

**Processing Speed:** The integration of the YOLO algorithm with the ESP 32-CAM enabled real-time processing, achieving frame rates suitable for applications requiring immediate responses.

**Resource Utilization:** The solution operates efficiently within the ESP 32-CAM's resource constraints, ensuring stable performance without the need for additional hardware.

**COST ANALYSIS**

A significant advantage of the proposed system is its cost-effectiveness. The ESP 32-CAM module is priced at low cost. substantially lower than traditional multi-camera setups that can cost hundreds of dollars. This affordability, combined with reduced hardware requirements, makes the solution ideal for large-scale deployments and projects with limited budgets.

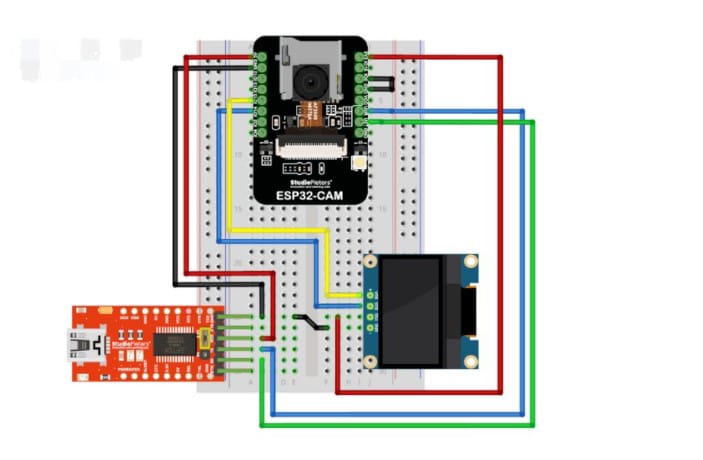
**APPLICATIONS**

The versatility and efficiency of the proposed system open avenues for various applications

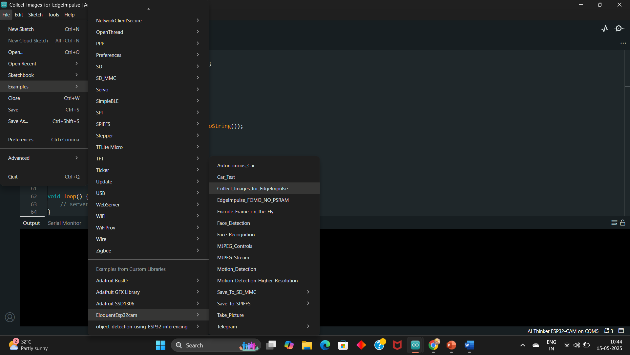
**COMPONENTS REQUIRED FOR ESP 32 CAM OBJECT DETECTION PROJECT:**

* **Hardware Tools**
* **ESP 32-CAM** – Main microcontroller with a built-in camera.
* **USB to Serial Converter** – Required to program the ESP 32-CAM.
* **Breadboard** – For easy circuit connections.
* **0.96” OLED Display (Optional)** – Displays classification results.
* **Jumper Wires** – For wiring connections.
* **SOFTWARE TOOLS:**
* **Edge Impulse**
* **Purpose**: Train and deploy ML models for object detection.
* **Usage**: Collect images, train AI models, and generate firmware for ESP 32-CAM.
* **Arduino IDE**
* **Purpose**: Write, upload, and debug code for ESP 32-CAM.  
   **Usage**: Install ESP 32 board, load ML model, and program ESP 32-CAM.

**CIRCUIT DIAGRAM FOR ESP 32 CAM IMAGE RECOGNITION**



**WORKING DEMONSTRATION OF ESP 32 CAM OBJECT RECOGNITION PROJECT**



**CODING**

**/\* Edge Impulse Arduino examples**

**\* Copyright (c) 2022 EdgeImpulse Inc.**

**\***

**\* Permission is hereby granted, free of charge, to any person obtaining a copy**

**\* of this software and associated documentation files (the "Software"), to deal**

**\* in the Software without restriction, including without limitation the rights**

**\* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell**

**\* copies of the Software, and to permit persons to whom the Software is**

**\* furnished to do so, subject to the following conditions:**

**\***

**\* The above copyright notice and this permission notice shall be included in**

**\* all copies or substantial portions of the Software.**

**\***

**\* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR**

**\* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,**

**\* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE**

**\* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER**

**\* 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.**

**\*/**

**// These sketches are tested with 2.0.4 ESP32 Arduino Core**

**// https://github.com/espressif/arduino-esp32/releases/tag/2.0.4**

**/\* Includes ---------------------------------------------------------------- \*/**

**#include <object\_detection\_using\_ESP32\_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/CameraWebServer/camera\_pins.h**

**#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**

**#define EI\_CAMERA\_RAW\_FRAME\_BUFFER\_ROWS           240**

**#define EI\_CAMERA\_FRAME\_BYTE\_SIZE                 3**

**#include <Wire.h>**

**#include <Adafruit\_GFX.h>**

**#include <Adafruit\_SSD1306.h>**

**// ESP32-CAM doesn't have dedicated i2c pins, so we define our own. Let's choose 15 and 14**

**#define I2C\_SDA 15**

**#define I2C\_SCL 14**

**TwoWire I2Cbus = TwoWire(0);**

**// Display defines**

**#define SCREEN\_WIDTH 128**

**#define SCREEN\_HEIGHT 64**

**#define OLED\_RESET -1**

**#define SCREEN\_ADDRESS 0x3C**

**Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &I2Cbus, OLED\_RESET);**

**/\* Private variables ------------------------------------------------------- \*/**

**static bool debug\_nn = false; // Set this to true to see e.g. features generated from the raw signal**

**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,**

**.pin\_sscb\_scl = SIOC\_GPIO\_NUM,**

**.pin\_d7 = Y9\_GPIO\_NUM,**

**.pin\_d6 = Y8\_GPIO\_NUM,**

**.pin\_d5 = Y7\_GPIO\_NUM,**

**.pin\_d4 = Y6\_GPIO\_NUM,**

**.pin\_d3 = Y5\_GPIO\_NUM,**

**.pin\_d2 = Y4\_GPIO\_NUM,**

**.pin\_d1 = Y3\_GPIO\_NUM,**

**.pin\_d0 = Y2\_GPIO\_NUM,**

**.pin\_vsync = VSYNC\_GPIO\_NUM,**

**.pin\_href = HREF\_GPIO\_NUM,**

**.pin\_pclk = PCLK\_GPIO\_NUM,**

**//XCLK 20MHz or 10MHz for OV2640 double FPS (Experimental)**

**.xclk\_freq\_hz = 20000000,**

**.ledc\_timer = LEDC\_TIMER\_0,**

**.ledc\_channel = LEDC\_CHANNEL\_0,**

**.pixel\_format = PIXFORMAT\_JPEG, //YUV422,GRAYSCALE,RGB565,JPEG**

**.frame\_size = FRAMESIZE\_QVGA,    //QQVGA-UXGA Do not use sizes above QVGA when not JPEG**

**.jpeg\_quality = 12, //0-63 lower number means higher quality**

**.fb\_count = 1,       //if more than one, i2s runs in continuous mode. Use only with JPEG**

**.fb\_location = CAMERA\_FB\_IN\_PSRAM,**

**.grab\_mode = CAMERA\_GRAB\_WHEN\_EMPTY,**

**};**

**/\* Function definitions ------------------------------------------------------- \*/**

**bool ei\_camera\_init(void);**

**void ei\_camera\_deinit(void);**

**bool ei\_camera\_capture(uint32\_t img\_width, uint32\_t img\_height, uint8\_t \*out\_buf) ;**

**/\*\***

**\* @brief      Arduino setup function**

**\*/**

**void setup()**

**{**

**// put your setup code here, to run once:**

**Serial.begin(115200);**

**//comment out the below line to start inference immediately after upload**

**while (!Serial);**

**Serial.println("Edge Impulse Inferencing Demo");**

**if (ei\_camera\_init() == false) {**

**ei\_printf("Failed to initialize Camera!\r\n");**

**}**

**else {**

**ei\_printf("Camera initialized\r\n");**

**}**

**ei\_printf("\nStarting continious inference in 2 seconds...\n");**

**ei\_sleep(2000);**

**}**

**/\*\***

**\* @brief      Get data and run inferencing**

**\***

**\* @param[in]  debug  Get debug info if true**

**\*/**

**void loop()**

**{**

**// instead of wait\_ms, we'll wait on the signal, this allows threads to cancel us...**

**if (ei\_sleep(5) != EI\_IMPULSE\_OK) {**

**return;**

**}**

**snapshot\_buf = (uint8\_t\*)malloc(EI\_CAMERA\_RAW\_FRAME\_BUFFER\_COLS \* EI\_CAMERA\_RAW\_FRAME\_BUFFER\_ROWS \* EI\_CAMERA\_FRAME\_BYTE\_SIZE);**

**// check if allocation was successful**

**if(snapshot\_buf == nullptr) {**

**ei\_printf("ERR: Failed to allocate snapshot buffer!\n");**

**return;**

**}**

**ei::signal\_t signal;**

**signal.total\_length = EI\_CLASSIFIER\_INPUT\_WIDTH \* EI\_CLASSIFIER\_INPUT\_HEIGHT;**

**signal.get\_data = &ei\_camera\_get\_data;**

**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;**

**}**

**// Run the classifier**

**ei\_impulse\_result\_t result = { 0 };**

**EI\_IMPULSE\_ERROR err = run\_classifier(&signal, &result, debug\_nn);**

**if (err != EI\_IMPULSE\_OK) {**

**ei\_printf("ERR: Failed to run classifier (%d)\n", err);**

**return;**

**}**

**// print the predictions**

**ei\_printf("Predictions (DSP: %d ms., Classification: %d ms., Anomaly: %d ms.): \n",**

**result.timing.dsp, result.timing.classification, result.timing.anomaly);**

**#if EI\_CLASSIFIER\_OBJECT\_DETECTION == 1**

**ei\_printf("Object detection bounding boxes:\r\n");**

**for (uint32\_t i = 0; i < result.bounding\_boxes\_count; i++) {**

**ei\_impulse\_result\_bounding\_box\_t bb = result.bounding\_boxes[i];**

**if (bb.value == 0) {**

**continue;**

**}**

**ei\_printf("  %s (%f) [ x: %u, y: %u, width: %u, height: %u ]\r\n",**

**bb.label,**

**bb.value,**

**bb.x,**

**bb.y,**

**bb.width,**

**bb.height);**

**}**

**// Print the prediction results (classification)**

**#else**

**ei\_printf("Predictions:\r\n");**

**for (uint16\_t i = 0; i < EI\_CLASSIFIER\_LABEL\_COUNT; i++) {**

**ei\_printf("  %s: ", ei\_classifier\_inferencing\_categories[i]);**

**ei\_printf("%.5f\r\n", result.classification[i].value);**

**}**

**#endif**

**// Print anomaly result (if it exists)**

**#if EI\_CLASSIFIER\_HAS\_ANOMALY**

**ei\_printf("Anomaly prediction: %.3f\r\n", result.anomaly);**

**#endif**

**#if EI\_CLASSIFIER\_HAS\_VISUAL\_ANOMALY**

**ei\_printf("Visual anomalies:\r\n");**

**for (uint32\_t i = 0; i < result.visual\_ad\_count; i++) {**

**ei\_impulse\_result\_bounding\_box\_t bb = result.visual\_ad\_grid\_cells[i];**

**if (bb.value == 0) {**

**continue;**

**}**

**ei\_printf("  %s (%f) [ x: %u, y: %u, width: %u, height: %u ]\r\n",**

**bb.label,**

**bb.value,**

**bb.x,**

**bb.y,**

**bb.width,**

**bb.height);**

**}**

**#endif**

**free(snapshot\_buf);**

**}**

**/\*\***

**\* @brief   Setup image sensor & start streaming**

**\***

**\* @retval  false if initialisation failed**

**\*/**

**bool ei\_camera\_init(void) {**

**if (is\_initialised) return true;**

**#if defined(CAMERA\_MODEL\_ESP\_EYE)**

**pinMode(13, INPUT\_PULLUP);**

**pinMode(14, INPUT\_PULLUP);**

**#endif**

**//initialize the camera**

**esp\_err\_t err = esp\_camera\_init(&camera\_config);**

**if (err != ESP\_OK) {**

**Serial.printf("Camera init failed with error 0x%x\n", err);**

**return false;**

**}**

**sensor\_t \* s = esp\_camera\_sensor\_get();**

**// initial sensors are flipped vertically and colors are a bit saturated**

**if (s->id.PID == OV3660\_PID) {**

**s->set\_vflip(s, 1); // flip it back**

**s->set\_brightness(s, 1); // up the brightness just a bit**

**s->set\_saturation(s, 0); // lower the saturation**

**}**

**#if defined(CAMERA\_MODEL\_M5STACK\_WIDE)**

**s->set\_vflip(s, 1);**

**s->set\_hmirror(s, 1);**

**#elif defined(CAMERA\_MODEL\_ESP\_EYE)**

**s->set\_vflip(s, 1);**

**s->set\_hmirror(s, 1);**

**s->set\_awb\_gain(s, 1);**

**#endif**

**is\_initialised = true;**

**return true;**

**}**

**/\*\***

**\* @brief      Stop streaming of sensor data**

**\*/**

**void ei\_camera\_deinit(void) {**

**//deinitialize the camera**

**esp\_err\_t err = esp\_camera\_deinit();**

**if (err != ESP\_OK)**

**{**

**ei\_printf("Camera deinit failed\n");**

**return;**

**}**

**is\_initialised = false;**

**return;**

**}**

**/\*\***

**\* @brief      Capture, rescale and crop image**

**\***

**\* @param[in]  img\_width     width of output image**

**\* @param[in]  img\_height    height of output image**

**\* @param[in]  out\_buf       pointer to store output image, NULL may be used**

**\*                           if ei\_camera\_frame\_buffer is to be used for capture and resize/cropping.**

**\***

**\* @retval     false if not initialised, image captured, rescaled or cropped failed**

**\***

**\*/**

**bool ei\_camera\_capture(uint32\_t img\_width, uint32\_t img\_height, uint8\_t \*out\_buf) {**

**bool do\_resize = false;**

**if (!is\_initialised) {**

**ei\_printf("ERR: Camera is not initialized\r\n");**

**return false;**

**}**

**camera\_fb\_t \*fb = esp\_camera\_fb\_get();**

**if (!fb) {**

**ei\_printf("Camera capture failed\n");**

**return false;**

**}**

**bool converted = fmt2rgb888(fb->buf, fb->len, PIXFORMAT\_JPEG, snapshot\_buf);**

**esp\_camera\_fb\_return(fb);**

**if(!converted){**

**ei\_printf("Conversion failed\n");**

**return false;**

**}**

**if ((img\_width != EI\_CAMERA\_RAW\_FRAME\_BUFFER\_COLS)**

**|| (img\_height != EI\_CAMERA\_RAW\_FRAME\_BUFFER\_ROWS)) {**

**do\_resize = true;**

**}**

**if (do\_resize) {**

**ei::image::processing::crop\_and\_interpolate\_rgb888(**

**out\_buf,**

**EI\_CAMERA\_RAW\_FRAME\_BUFFER\_COLS,**

**EI\_CAMERA\_RAW\_FRAME\_BUFFER\_ROWS,**

**out\_buf,**

**img\_width,**

**img\_height);**

**}**

**return true;**

**}**

**static int ei\_camera\_get\_data(size\_t offset, size\_t length, float \*out\_ptr)**

**{**

**// we already have a RGB888 buffer, so recalculate offset into pixel index**

**size\_t pixel\_ix = offset \* 3;**

**size\_t pixels\_left = length;**

**size\_t out\_ptr\_ix = 0;**

**while (pixels\_left != 0) {**

**// Swap BGR to RGB here**

**// due to https://github.com/espressif/esp32-camera/issues/379**

**out\_ptr[out\_ptr\_ix] = (snapshot\_buf[pixel\_ix + 2] << 16) + (snapshot\_buf[pixel\_ix + 1] << 8) + snapshot\_buf[pixel\_ix];**

**// go to the next pixel**

**out\_ptr\_ix++;**

**pixel\_ix+=3;**

**pixels\_left--;**

**}**

**// and done!**

**return 0;**

**}**

**#if !defined(EI\_CLASSIFIER\_SENSOR) || EI\_CLASSIFIER\_SENSOR != EI\_CLASSIFIER\_SENSOR\_CAMERA**

**#error "Invalid model for current sensor"**

**#endif**

**OUTPUT:**

