



TECNOLÓGICO
NACIONAL DE MÉXICO



Instituto Tecnológico Superior del Occidente del Estado de Hidalgo
División de ingeniería en Tecnologías de la Información y Comunicaciones



Instituto Tecnológico Superior del Occidente del Estado de Hidalgo

Asignatura: Introducción a las TICs

Carrera: Ingeniera en Tecnologías de la información y comunicación

Grupo y grado: 1C

Ciclo escolar: 2025-2029

Titulo: Prototipo ESPCAM

Alumnos: Jonathan Jovany Monjaraz Perez ,

Brayan Antonio Ordonez Zavala y Gezer Ismael Trejo Cruz

Profesor: Saul Isai Soto Ortiz

Fecha de entrega: 26/10/2025



Índice

Resumen	3
Introducción	3
Procedimiento	4
Resultados.....	7
Código	8
Explicación del Código	19
Conclusión.....	26
Bibliografias.....	27



Resumen

El siguiente reporte presenta el desarrollo del prototipo de una ESPCAM con librerías creadas en Edge Impulse e implementadas en Arduino IDE, lo cual permite reconocer objetos en tiempo real, proyectándolos en una pantalla OLED y encendiendo los leds correspondientes. La implementación demuestra la aplicabilidad de tecnologías IoT y machine learning en entornos educativos, profesionales y de la vida cotidiana.

Introducción

El uso de tecnologías web y móviles ha revolucionado la forma en que interactuamos con dispositivos embebidos. En este contexto, la placa ESPCAM se presenta como una herramienta accesible y versátil para aplicaciones de visión artificial. Junto con Edge Impulse, una plataforma para entrenar modelos de machine learning en el edge, es posible desarrollar sistemas de reconocimiento de objetos sin depender de infraestructura en la nube.

Este reporte documenta el desarrollo de un sistema de reconocimiento de objetos utilizando ESPCAM y Edge Impulse. La ESPCAM, con su cámara integrada y capacidad de procesamiento, se combina con Edge Impulse para crear un sistema de visión artificial accesible y eficiente.

El proyecto busca demostrar cómo los dispositivos embebidos pueden ejecutar modelos de machine learning directamente en el hardware, sin necesidad de conexión constante a la nube. Esto permite aplicaciones en tiempo real con bajo consumo energético y mayor velocidad de respuesta.

Los componentes adicionales enriquecen la funcionalidad del prototipo: la pantalla OLED muestra las etiquetas de los objetos identificados, mientras que los LEDs proporcionan indicadores luminosos para cada categoría detectada. Todo el sistema se programa mediante el IDE de Arduino, utilizando librerías específicas para cada componente.

En esta guía se explican los conceptos esenciales de ESPCAM y reconocimiento de objetos, mostrando el proceso completo desde la configuración inicial hasta los resultados finales. El documento está estructurado para facilitar la comprensión del funcionamiento del prototipo y su potencial aplicación en soluciones prácticas.



Procedimiento

1. Instalación y verificación de ESPCAM en Windows: Se configuró el entorno de desarrollo en el IDE de Arduino, instalando los boards managers de la familia ESP32 y las librerías necesarias para la cámara (ESPCAM Camera) y la pantalla OLED (Adafruit SSD1306 y GFX). Posteriormente, se conectó la ESPCAM mediante un adaptador FTDI, verificando su detección en el sistema y cargando un sketch de prueba para comprobar la funcionalidad de la cámara. También se realizaron pruebas individuales con los LEDs y la pantalla OLED para asegurar su correcto uso.

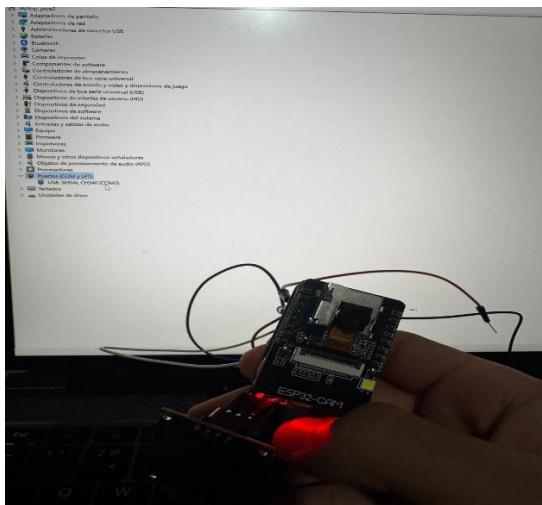
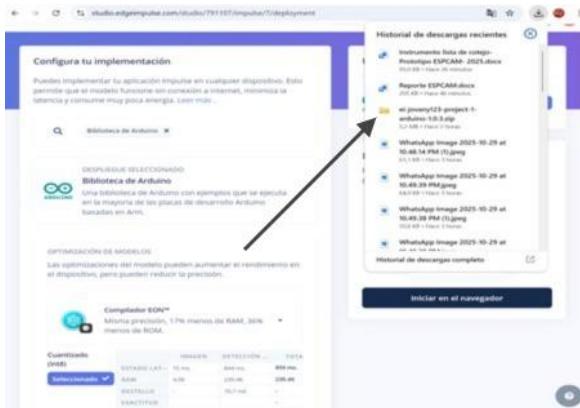


Imagen donde se verifica que la ESPCAM sea detectada en el sistema.

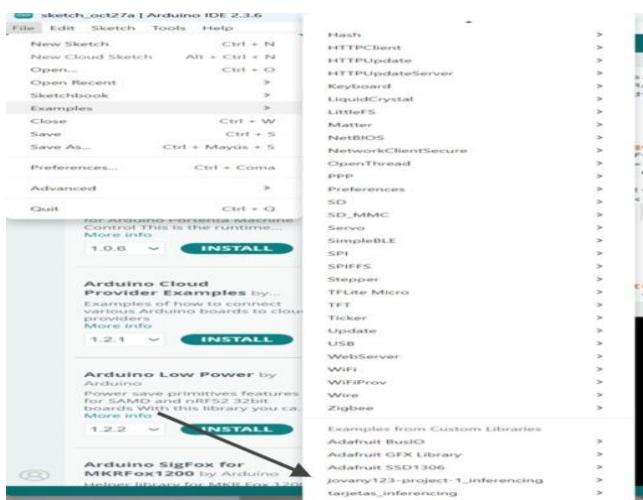
2. Generación de la librería de Edge Impulse: En la plataforma Edge Impulse se recolectó un conjunto de datos de aproximadamente 104 imágenes por clase (2 clases en total) utilizando la cámara de la ESPCAM directamente desde la plataforma. Luego se entrenó un modelo de clasificación de imágenes, logrando una precisión del 88% tras 30 épocas de entrenamiento. Finalmente, el modelo se exportó como librería de Arduino (.zip) lista para integrar en el código.



Librería descargada.

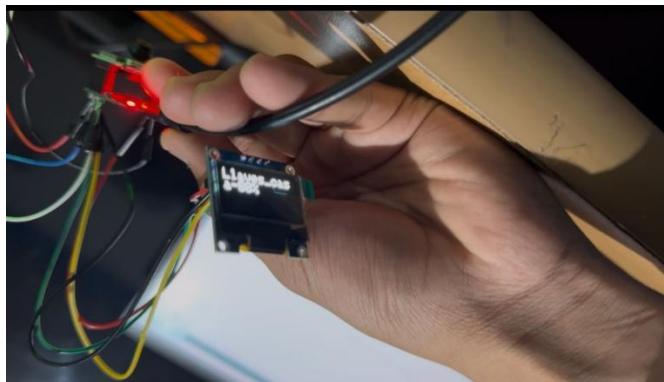


3. Instalación de la librería en ESPCAM: La librería descargada fue añadida al proyecto mediante la opción “Programa → Incluir librería → Añadir biblioteca .ZIP” en el IDE de Arduino. Después se verificó su correcta instalación y se cargó el código generado automáticamente por Edge Impulse, asegurando la compatibilidad con la tarjeta AI Thinker ESP32-CAM.



Librería instalada.

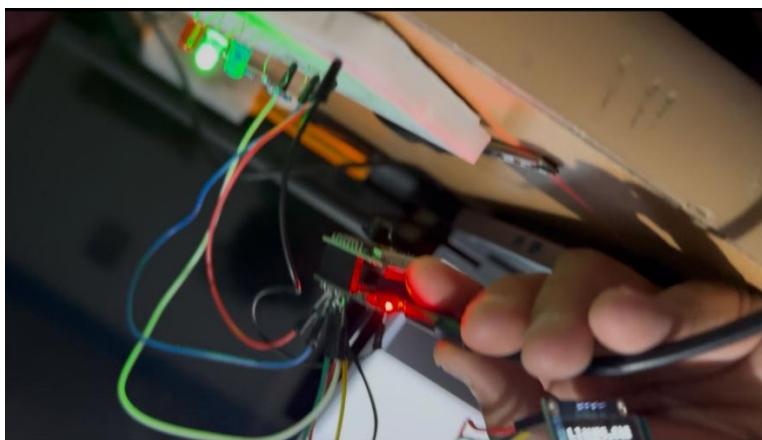
4. Demostración del funcionamiento con la librería: Se integraron los módulos de cámara, display OLED y control de LEDs dentro de un mismo sketch. La ESPCAM capturaba imágenes en tiempo real y procesaba las predicciones del modelo entrenado. Cada vez que el sistema identificaba un objeto, el nombre aparecía en el monitor serie y posteriormente en la pantalla OLED. Este paso confirmó que el modelo respondía de forma correcta a las detecciones previstas.



ESPCAM detectando objeto en pantalla OLED.



5. Activación de LEDS por objeto identificado y visualización en pantalla OLED: El sistema final combinó hardware y software de manera integrada. Cada vez que la cámara reconocía un objeto específico, se encendía un LED correspondiente y se mostraba su etiqueta en la pantalla OLED. Por ejemplo, al detectar “Llaves_casa”, el LED indicador se activaba de inmediato, proporcionando retroalimentación visual. El circuito ensamblado en la protoboard, con conexiones firmes y bien distribuidas, funcionó correctamente durante todas las pruebas, cumpliendo los objetivos de reconocimiento, visualización y respuesta luminosa.



ESPCAM detectando objeto y encendiéndo el LED correspondiente.



Resultados

Resultados del funcionamiento del prototipo ESPCAM con Edge Impulse y display OLED

El sistema desarrollado integró exitosamente los módulos de captura, procesamiento, visualización y retroalimentación visual mediante LEDs. Durante las pruebas, la ESP32-CAM logró ejecutar el modelo de clasificación entrenado en Edge Impulse de manera estable, mostrando los resultados en la pantalla OLED y activando el LED correspondiente a cada objeto detectado. El código implementado permitió la ejecución local del modelo sin conexión a internet, destacando las funciones `run_classifier()` y `ei_camera_capture()`, encargadas de capturar la imagen, procesarla y devolver las predicciones. Estas funciones garantizan la operación autónoma del sistema y una respuesta inmediata ante la presencia de los objetos entrenados. Durante las pruebas de funcionamiento, la cámara capturó las imágenes a una resolución de 320x240 píxeles y las procesó en tiempo real.

Cuando el modelo identificaba una clase específica, se mostraba en el display OLED el nombre del objeto y el porcentaje de confianza, mediante las funciones `display.print()` y `display.display()` de la librería Adafruit_SSD1306. Esto permitió verificar visualmente la detección directamente desde el dispositivo sin necesidad del monitor serie.

Para la respuesta física, se emplearon tres LEDs conectados a los pines GPIO 12, 13 y 2. Cada uno se encendía dependiendo del objeto reconocido, siguiendo las condiciones establecidas en el siguiente bloque de código:

```
if (strcmp(bb.label, "Llaves_casa") == 0) {  
    digitalWrite(LED_AZUL, LOW);  
    digitalWrite(LED_VERDE, HIGH);  
    digitalWrite(LED_ROJO, LOW);  
}  
else if (strcmp(bb.label, "llaves_moto") == 0) {  
    digitalWrite(LED_AZUL, LOW);  
    digitalWrite(LED_VERDE, LOW);  
    digitalWrite(LED_ROJO, HIGH);  
}  
else {  
    digitalWrite(LED_AZUL, HIGH);  
    digitalWrite(LED_VERDE, LOW);  
    digitalWrite(LED_ROJO, LOW);  
}
```



{

Este fragmento fue determinante para vincular las predicciones del modelo con la activación de los LEDs, logrando una retroalimentación visual clara y confiable. En los casos en que no se detectaba ningún objeto, el sistema mostraba “No detecto” en la pantalla OLED y mantenía únicamente el LED azul encendido como estado de espera. El procesamiento del modelo alcanzó tiempos promedio de inferencia entre 250 y 400 ms, demostrando un rendimiento adecuado para aplicaciones en tiempo real. Además, la comunicación I2C con la pantalla OLED fue estable y permitió la actualización continua de la información sin errores ni interrupciones.

Durante el ensamblaje final, las conexiones entre la ESP32-CAM, la pantalla OLED y los LEDs se realizaron sobre una protoboard, asegurando una distribución limpia y eficiente de la alimentación. El sistema completo operó correctamente durante las pruebas, reconociendo los objetos entrenados (“Llaves_casa” y “Ilves_moto”) con alta precisión y reaccionando de manera inmediata mediante la visualización y la iluminación de los LEDs.

Los resultados obtenidos confirmaron el correcto funcionamiento del modelo embebido, la comunicación entre módulos y la ejecución estable del código. El prototipo cumplió satisfactoriamente con los requerimientos del laboratorio al lograr detección visual, procesamiento local y respuesta luminosa en tiempo real.

Código

```
//Pin GND-GND
```

```
//3.3 V-VCC
```

```
//14-SCL
```

```
//15-SDA
```

```
#include <jovany123-project-1_inferencing.h>
#include "edge-impulse-sdk/dsp/image/image.hpp"
#include "esp_camera.h"

#define CAMERA_MODEL_AI_THINKER
```



```
#if 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

#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>

#define I2C_SDA 15
#define I2C_SCL 14
TwoWire I2Cbus = TwoWire(0);

#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);

/* ◆ Pines para LEDs de estado */
#define LED_AZUL 12
#define LED_VERDE 13
#define LED_ROJO 2

static bool debug_nn = false;
static bool is_initialised = false;
uint8_t *snapshot_buf;

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_freq_hz = 20000000,  
.ledc_timer = LEDC_TIMER_0,  
.ledc_channel = LEDC_CHANNEL_0,  
  
.pixel_format = PIXFORMAT_JPEG,  
.frame_size = FRAMESIZE_QVGA,  
.jpeg_quality = 12,  
.fb_count = 1,  
.fb_location = CAMERA_FB_IN_PSRAM,  
.grab_mode = CAMERA_GRAB_WHEN_EMPTY,  
};  
  
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);
```

```
void setup() {
```

```
    Serial.begin(115200);
```

```
    I2Cbus.begin(I2C_SDA, I2C_SCL, 1000000);
```

```
    if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
```

```
        Serial.printf("SSD1306 OLED display failed to initialize.\n");
```

```
        while (true);
```

```
}
```

```
    pinMode(LED_AZUL, OUTPUT);
```

```
    pinMode(LED_VERDE, OUTPUT);
```

```
    pinMode(LED_ROJO, OUTPUT);
```

```
    digitalWrite(LED_AZUL, HIGH);
```

```
    digitalWrite(LED_VERDE, LOW);
```

```
    digitalWrite(LED_ROJO, LOW);
```

```
    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 continuous inference in 2 seconds...\n");
display.clearDisplay();
display.setCursor(0, 0);
display.setTextSize(1);
display.setTextColor(SSD1306_WHITE);
display.print("Starting continuous\n inference in\n 2 seconds...");
display.display();
ei_sleep(2000);
display.clearDisplay();

}

void loop() {
display.clearDisplay();

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);

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;  
  
}  
  
  
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;  
  
}  
  
  
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  
  
bool bb_found = result.bounding_boxes[0].value > 0;  
  
bool detecto_Llaves_casa = false;  
  
bool detecto_llves_moto = false;  
  
  
for (size_t ix = 0; ix < result.bounding_boxes_count; ix++) {  
  
    auto bb = result.bounding_boxes[ix];  
  
    if (bb.value == 0) continue;  
  
  
    ei_printf("    %s (%f) [x: %u, y: %u, w: %u, h: %u]\n", bb.label, bb.value, bb.x, bb.y, bb.width,  
        bb.height);  
  
    display.setCursor(0, 20 * ix);
```



```
display.setTextSize(2);

display.setTextColor(SSD1306_WHITE);

display.print(bb.label);

display.print("-");

display.print(int((bb.value)*100));

display.print("%");

display.display();

if (strcmp(bb.label, "Llaves_casa") == 0) detecto_Llaves_casa = true;

if (strcmp(bb.label, "llves_moto") == 0) detecto_llves_moto = true;

}

if (bb_found) {

if (detecto_Llaves_casa) {

digitalWrite(LED_AZUL, LOW);

digitalWrite(LED_VERDE, HIGH);

digitalWrite(LED_ROJO, LOW);

} else if (detecto_llves_moto) {

digitalWrite(LED_AZUL, LOW);

digitalWrite(LED_VERDE, LOW);

digitalWrite(LED_ROJO, HIGH);

} else {

digitalWrite(LED_AZUL, HIGH);

digitalWrite(LED_VERDE, LOW);

digitalWrite(LED_ROJO, LOW);

}

} else {

ei_printf("  Nada detectado\n");
```



```
display.setCursor(0, 20);

display.setTextSize(2);

display.setTextColor(SSD1306_WHITE);

display.print("No detecto");

display.display();

digitalWrite(LED_AZUL, HIGH);

digitalWrite(LED_VERDE, LOW);

digitalWrite(LED_ROJO, LOW);

}

#endif

free(snapshot_buf);

}

bool ei_camera_init(void) {

if (is_initialised) return true;

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();

if (s->id.PID == OV3660_PID) {

s->set_vflip(s, 1);

s->set_brightness(s, 1);

s->set_saturation(s, 0);
```



}

```
is_initialised = true;
```

```
return true;
```

```
}
```

```
void ei_camera_deinit(void) {
```

```
esp_err_t err = esp_camera_deinit();
```

```
if (err != ESP_OK) {
```

```
ei_printf("Camera deinit failed\n");
```

```
return;
```

```
}
```

```
is_initialised = false;
```

```
}
```

```
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) {

    size_t pixel_ix = offset * 3;

    size_t pixels_left = length;

    size_t out_ptr_ix = 0;

    while (pixels_left != 0) {

        out_ptr[out_ptr_ix] = (snapshot_buf[pixel_ix + 2] << 16)
```



```
+ (snapshot_buf[pixel_ix + 1] << 8)  
+ snapshot_buf[pixel_ix];  
  
out_ptr_ix++;  
  
pixel_ix += 3;  
  
pixels_left--;  
  
}  
  
return 0;  
}  
  
  
#if !defined(EI_CLASSIFIER_SENSOR) || EI_CLASSIFIER_SENSOR != EI_CLASSIFIER_SENSOR_CAMERA  
#error "Invalid model for current sensor"  
#endif
```

Explicación del Código

1. Comentarios de conexión

//Pin GND-GND, //3.3 V-VCC, //14-SCL, //15-SDA

Solo son notas de cableado: indican cómo conectar el OLED al ESP32-CAM: GND a GND, VCC a 3.3 V, SCL al pin 14 y SDA al pin 15 del ESP32. No afectan al programa

2. Inclusión de librerías principales

#include <jovany123-project-1_inferencing.h>

Incluye el archivo generado por Edge Impulse con tu modelo (clases, dimensiones de entrada, run_classifier, etc.).

#include "edge-impulse-sdk/dsp/image/image.hpp"

Incluye funciones de procesamiento de imagen (recortar, redimensionar, interpolar imágenes RGB).

#include "esp_camera.h"

Incluye la librería de ESP32 para configurar la cámara, inicializarla y capturar imágenes.



3. Selección del modelo de cámara y definición de pines

```
#define CAMERA_MODEL_AI_THINKER
```

Indica que se usa el módulo ESP32-CAM AI Thinker.

```
Bloque #if defined(CAMERA_MODEL_AI_THINKER) ... #endif
```

Dentro de este bloque se mapean todos los pines físicos del ESP32 a las señales de la cámara:

- PWDN_GPIO_NUM, RESET_GPIO_NUM, XCLK_GPIO_NUM para encendido, reset y reloj del sensor.
- SIOD_GPIO_NUM, SIOC_GPIO_NUM para el bus SCCB/I2C de configuración.
- Y2_GPIO_NUM a Y9_GPIO_NUM para los datos de imagen en paralelo.
- VSYNC_GPIO_NUM, HREF_GPIO_NUM, PCLK_GPIO_NUM para sincronización vertical, comienzo de línea y reloj de píxel.

Si no se define un modelo válido, el #error "Camera model not selected" fuerza un error de compilación.

4. Parámetros del búfer de imagen cruda

```
EI_CAMERA_RAW_FRAME_BUFFER_COLS 320, EI_CAMERA_RAW_FRAME_BUFFER_ROWS 240,  
EI_CAMERA_FRAME_BYTE_SIZE 3
```

Definen el tamaño crudo de la imagen de la cámara: 320 x 240 píxeles, con 3 bytes por píxel (RGB).

5. Librerías y configuración de I2C + OLED

```
#include <Wire.h>, #include <Adafruit_GFX.h>, #include <Adafruit_SSD1306.h>
```

Librerías para manejar I2C y el display OLED SSD1306, con funciones gráficas (texto, figuras, etc.).

```
#define I2C_SDA 15, #define I2C_SCL 14, TwoWire I2Cbus = TwoWire(0);
```

Configuran los pines 15 y 14 como SDA y SCL de I2C, y crean el objeto I2Cbus para manejar ese bus.

```
SCREEN_WIDTH 128, SCREEN_HEIGHT 64, OLED_RESET -1, SCREEN_ADDRESS 0x3C y  
Adafruit_SSD1306 display(...)
```

Definen tamaño, reset y dirección I2C del OLED, y crean el objeto display que se usará para escribir texto en la pantalla.



6. Pines de LEDs y variables globales

LED_AZUL 12, LED_VERDE 13, LED_ROJO 2

Definen los pines donde se conectan los LEDs de estado (azul, verde y rojo).

static bool debug_nn = false;

Bandera para activar o no mensajes de depuración de la red neuronal (no la usas mucho aquí, pero se pasa a run_classifier).

static bool is_initialised = false;

Indica si la cámara ya fue inicializada para no hacerlo dos veces.

uint8_t *snapshot_buf;

Puntero a un búfer donde se almacenará la imagen cruda capturada de la cámara.

7. Estructura de configuración de la cámara

static camera_config_t camera_config = { ... };

Estructura que agrupa toda la configuración de la cámara:

- Asocia cada pin físico (PWDN_GPIO_NUM, Y2_GPIO_NUM, VSYNC_GPIO_NUM, etc.) con su función en el módulo de cámara.
- Configura parámetros de funcionamiento: reloj XCLK a 20 MHz (xclk_freq_hz), formato de salida JPEG (pixel_format), resolución QVGA (320 x 240, frame_size), calidad JPEG (jpeg_quality = 12), un solo buffer (fb_count = 1), ubicación del buffer en PSRAM (fb_location), y modo de captura (grab_mode = CAMERA_GRAB_WHEN_EMPTY).

8. Prototipos de funciones

ei_camera_init, ei_camera_deinit, ei_camera_capture

Solo anuncian las funciones que luego se implementan para inicializar la cámara, desinicializarla y capturar imágenes.

9. Función setup

void setup()

Esta función se ejecuta una sola vez al encender:

- Serial.begin(115200);

Inicializa el puerto serie a 115200 baudios para poder ver mensajes en el monitor.



- I2Cbus.begin(I2C_SDA, I2C_SCL, 100000);
Inicia el bus I2C con los pines 15 (SDA) y 14 (SCL) a 100 kHz.
- if (!display.begin(...)) { ... }
Intenta inicializar el OLED. Si falla, imprime un error por serie y se queda en un bucle infinito, porque sin display el resto de la interfaz no tendría sentido.
- pinMode(LED_AZUL/VERDE/ROJO, OUTPUT) y digitalWrite iniciales
Configuran los tres LEDs como salidas y encienden el azul (sistema listo) mientras los otros dos se mantienen apagados.
- while (!Serial); y Serial.println("Edge Impulse Inferencing Demo");
Espera a que el puerto serie esté listo y escribe un mensaje identificando el programa.
- if (ei_camera_init() == false) { ... } else { ... }
Llama a la función que inicializa la cámara, avisando por serie si falló o si se inició correctamente.
- ei_printf("\nStarting continuous inference in 2 seconds...\n");
Anuncia por consola que la inferencia comenzará en 2 segundos.
- Bloque del display: display.clearDisplay();, setCursor, setTextSize, setTextColor,
display.print(...), display.display();, ei_sleep(2000);, display.clearDisplay();
Muestra en el OLED el mensaje de "Starting continuous inference in 2 seconds...",
espera 2 segundos y luego borra la pantalla para dejarla lista para el bucle principal.

10. Función loop (flujo general)

void loop()

Se repite continuamente:

- display.clearDisplay();
Limpia la pantalla al inicio de cada iteración.
- if (ei_sleep(5) != EI_IMPULSE_OK) return;
Espera 5 ms de forma compatible con Edge Impulse; si algo va mal, sale de la iteración.
- Reservar memoria:
snapshot_buf = (uint8_t *)malloc(...); + if (snapshot_buf == nullptr) { ... }
Reserva memoria para la imagen cruda (320 x 240 x 3); si no hay memoria, imprime error y sale del loop.
- Preparar el signal de entrada:
ei::signal_t signal;



Se establece su longitud (total_length = ancho * alto del modelo) y la función que proporcionará los datos (signal.get_data = &ei_camera_get_data).

- Capturar imagen:

```
if (ei_camera_capture(...) == false) { ... }
```

Toma una foto, la convierte a RGB y, si falla, imprime error, libera snapshot_buf y termina la iteración.

- Ejecutar el modelo:

```
ei_impulse_result_t result = {0};
```

```
EI_IMPULSE_ERROR err = run_classifier(&signal, &result, debug_nn);
```

Ejecuta la red neuronal sobre la imagen capturada. Si err indica fallo, se imprime el error y se termina la iteración.

- Imprimir tiempos de inferencia:

```
ei_printf("Predictions (DSP: %d ms., Classification: %d ms., Anomaly: %d ms.): ..." )
```

Muestra cuánto tardó el procesamiento de señal, la clasificación y el módulo de anomalías.

11. Bloque de detección de objetos y manejo de LEDs/OLED

```
#if EI_CLASSIFIER_OBJECT_DETECTION == 1
```

El código siguiente solo se usa si el modelo es de detección de objetos.

- Variables de estado:

```
bool bb_found = result.bounding_boxes[0].value > 0;
```

Indica si existe al menos una bounding box válida.

```
bool detecto_Llaves_casa = false; y bool detecto_llves_moto = false;
```

Señalan si se detectaron específicamente las clases "Llaves_casa" o "llves_moto".

- Recorrido de detecciones:

```
for (size_t ix = 0; ix < result.bounding_boxes_count; ix++) { ... }
```

Recorre todas las bounding boxes de result y, para cada una con valor distinto de cero:

- Imprime por serie la etiqueta, la probabilidad y las coordenadas de la caja.

- Muestra en el OLED la etiqueta y la confianza en porcentaje, en una fila distinta (setCursor(0, 20 * ix), tamaño de texto 2).

- Compara bb.label con "Llaves_casa" y "llves_moto" para activar las banderas detecto_Llaves_casa y detecto_llves_moto cuando se detectan esas clases.

- Decisión de LEDs cuando sí hay detecciones (if (bb_found) { ... }):

- Si detecto_Llaves_casa es verdadero:

```
digitalWrite(LED_AZUL, LOW); digitalWrite(LED_VERDE, HIGH); digitalWrite(LED_ROJO,
```



LOW);

Se enciende el LED verde y se apagan azul y rojo para indicar detección de llaves de casa.

– En caso contrario, si detecto_llaves_moto es verdadero:

Se enciende solo el LED rojo (azul y verde apagados) para indicar detección de llaves de moto.

– Si hay detecciones pero ninguna de esas dos clases:

Se enciende solo el LED azul para indicar que se detectó algún objeto, pero no las clases de interés.

- Caso sin detecciones (else de if (bb_found)):

– Se imprime “Nada detectado” por consola.

– En el OLED se escribe “No detecto” con tamaño de texto 2.

– Se enciende el LED azul y se apagan verde y rojo, indicando que no se detectó ningún objeto.

#endif

Fin del bloque de detección de objetos.

- free(snapshot_buf);

Libera la memoria de la imagen al final de la iteración del loop para evitar fugas.

12. Función ei_camera_init

bool ei_camera_init(void)

Inicializa la cámara:

- Si is_initialised ya es verdadero, solo devuelve true (ya estaba lista).
- Llama a esp_camera_init(&camera_config) con la estructura de configuración. Si falla, se imprime el código de error y se devuelve false.
- Obtiene el objeto sensor_t *s = esp_camera_sensor_get();. Si el sensor es del tipo OV3660, ajusta orientación (set_vflip), brillo (set_brightness) y saturación (set_saturation).
- Marca is_initialised = true; y devuelve true.

13. Función ei_camera_deinit

void ei_camera_deinit(void)

Apaga / desinicializa la cámara:



- Llama a `esp_camera_deinit()`. Si falla, imprime un mensaje de error y regresa.
- Si todo va bien, pone `is_initialised = false`; para indicar que la cámara ya no está disponible.

14. Función `ei_camera_capture`

```
bool ei_camera_capture(uint32_t img_width, uint32_t img_height, uint8_t *out_buf)
```

Captura y prepara una imagen para el modelo:

- Verifica que la cámara esté inicializada; si no, imprime error y devuelve `false`.
- Llama a `esp_camera_fb_get()` para obtener un frame buffer `fb` con una imagen JPEG. Si `fb` es nulo, indica error y devuelve `false`.
- Convierte el JPEG a formato RGB888 dentro de `snapshot_buf` usando `fmt2rgb888(...)`. Después devuelve el buffer a la cámara con `esp_camera_fb_return(fb)`. Si la conversión falla, imprime error y devuelve `false`.
- Comprueba si el tamaño deseado (`img_width, img_height`) coincide con 320 x 240. Si no coinciden, marca que hay que redimensionar.
- Si es necesario redimensionar, usa `ei::image::processing::crop_and_interpolate_rgb888(...)` para recortar y escalar la imagen al tamaño que el modelo necesita, guardando el resultado en `out_buf`.
- Si todo fue bien, devuelve `true`.

15. Función `ei_camera_get_data`

```
static int ei_camera_get_data(size_t offset, size_t length, float *out_ptr)
```

Función de callback que Edge Impulse usa para leer partes de la imagen:

- Calcula el índice de byte en `snapshot_buf` correspondiente al píxel `offset` (`pixel_ix = offset * 3`).
- Recorre `length` píxeles y, para cada uno, toma 3 bytes consecutivos (RGB) desde `snapshot_buf`, los combina en un entero de 24 bits y lo guarda como `float` en `out_ptr`.
- Avanza índices (`out_ptr_ix, pixel_ix`) y decrementa `pixels_left` hasta terminar.
- Devuelve 0 para indicar que la lectura se hizo correctamente.



16. Verificación del tipo de sensor del modelo

```
#if !defined(EI_CLASSIFIER_SENSOR) || EI_CLASSIFIER_SENSOR !=  
EI_CLASSIFIER_SENSOR_CAMERA
```

Comprueba en tiempo de compilación que el modelo de Edge Impulse está configurado para sensor de tipo cámara.

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

Si no es así, produce un error de compilación para evitar usar un modelo incompatible.

```
#endif
```

Fin de la verificación.

Conclusión

La integración exitosa de ESPCAM, Arduino IDE, Edge Impulse, pantalla OLED y LEDs demostró la viabilidad de crear sistemas de visión artificial accesibles y funcionales. El prototipo cumplió consistentemente con su objetivo: capturar imágenes, procesarlas localmente mediante un modelo de machine learning, mostrar los resultados en la OLED y activar los LEDs correspondientes al objeto identificado.

A lo largo del proyecto, se logró integrar exitosamente todos los componentes en un sistema cohesivo y funcional. La ESPCAM probó ser una plataforma robusta para aplicaciones de visión artificial, capaz de capturar imágenes y ejecutar inferencias de machine learning de manera eficiente. A través de Edge Impulse, se pudo entrenar, validar e implementar un modelo de clasificación de imágenes que, una vez cargado en la placa, operó de forma autónoma y en tiempo real.

El uso de Arduino IDE fue fundamental para programar y coordinar el comportamiento de todos los elementos del sistema. Desde la inicialización de la cámara y la configuración de los pines, hasta la gestión de las inferencias y la actualización de la interfaz, el entorno de desarrollo permitió un control preciso sobre cada aspecto del funcionamiento del prototipo.

La pantalla OLED cumplió un papel crucial como interfaz de usuario, mostrando de manera inmediata y legible la etiqueta del objeto identificado. Esta retroalimentación visual permitió verificar el correcto funcionamiento del modelo de IA y facilitó la depuración durante las etapas de desarrollo.

Por su parte, los LEDs añadieron una capa adicional de interacción, proporcionando una señal luminosa “inequívoca” que complementaba la información mostrada en la pantalla. Esta



funcionalidad resulta especialmente útil en escenarios donde una confirmación rápida y a distancia es necesaria.

En conclusión, este proyecto no solo resultó en un prototipo funcional de reconocimiento de objetos, sino que proporcionó un marco de referencia completo para el desarrollo de sistemas embebidos inteligentes, demostrando que con los componentes y herramientas adecuadas, es posible materializar aplicaciones de IA que hasta hace poco parecían exclusivas de entornos industriales o de investigación avanzada.

Bibliografias

Edge Impulse Documentation, "Getting Started with ESP-EYE", 2024. [En línea]. Disponible: <https://docs.edgeimpulse.com>

Espressif Systems, "ESP32-CAM Datasheet", 2023.

A. Gupta, "Embedded Machine Learning for IoT Devices", IEEE IoT Journal, vol. 10, no. 4, pp. 3200-3210, 2023.

M. López, "Introducción a la visión artificial con microcontroladores", Revista Iberoamericana de Electrónica, vol. 12, no. 2, 2022.