

INFORME

GUIA DE TRABAJO CON ESP32 Y RECONOCIMIENTO DE IMÁGENES

FULVIO YESID VIVAS CANTERO

**DEPARTAMENTO DE TELEMATICA
FACULTAD DE INGENIERIA ELECTRONICA Y
TELECOMUNICACIONES**

AGOSTO

2021

TinyML ESP32-CAM: Clasificación de Imágenes con Edge Impulse

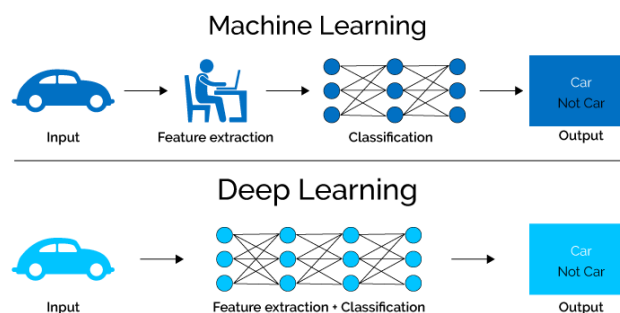
Objetivo: Usar TinyML con ESP32-CAM para reconocer y clasificar imágenes usando aprendizaje profundo que se ejecuta directamente en el dispositivo. Los pasos contemplan: crear un modelo de aprendizaje automático usando Tensorflow lite, seleccionar el modelo y desplegar e implementar el modelo en el ESP32.

GENERALIDADES

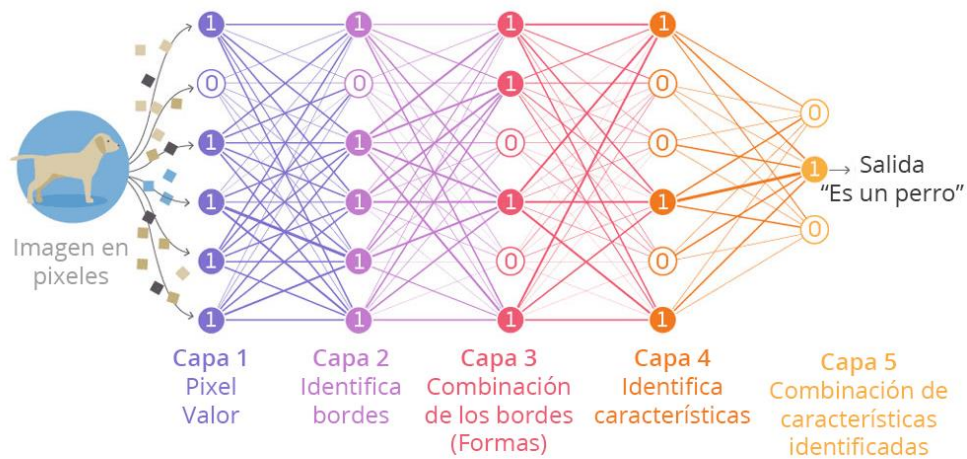
Deep Learning

El Deep Learning es un subconjunto de inteligencia artificial y una pieza de una familia más amplia de aprendizaje artificial. En lugar de humanos que programan aplicaciones informáticas específicas de la tarea, el Deep Learning toma datos no estructurados y los entrena para obtener resultados más precisos en función de los datos de entrenamiento. Las aplicaciones de Deep Learning aprenden y resuelven tareas limitadas sin que se las programe explícitamente para hacerlo.

Aplicaciones: reconocimiento facial para desbloquear teléfonos o identificar amigos en fotos de medios sociales, motores de recomendación en videos en tiempo real y servicios de música o al comprar en sitios de comercio electrónico, procesamiento de imágenes médicas como ayudar a diagnosticar enfermedades como el cáncer, filtros de correo no deseado en el correo electrónico, y detección de fraude con tarjetas de crédito



El Deep Learning se basa en el trabajo de las redes neurales. Cuando una red neural es entrenada, los datos de entrenamiento pasan al nivel inferior, el nivel de entrada, y pasan a través de sucesivos niveles informáticos, multiplicándose y sumándose juntos en formas complejas, hasta que finalmente llegan, radicalmente transformados, al nivel de salida. Es allí cuando el programa determina por qué una imagen es una casa y no un perro, por ejemplo.



Fuente: <https://www.quantamagazine.org/>

Dataset

Este conjunto de datos contiene 4242 imágenes de flores. La recopilación de datos se basa en el archivo de datos, imágenes de Google, imágenes de Yandex. Puede utilizar este dataset para reconocer las plantas de la foto.

Las imágenes se dividen en cinco clases: manzanilla, tulipán, rosa, girasol, diente de león. Para cada clase hay unas 800 fotos. Las fotos no son de alta resolución, aproximadamente 320x240 píxeles. Las fotos no se reducen a un solo tamaño, ¡tienen diferentes proporciones!

<https://www.kaggle.com/alxmamaev/flowers-recognition>

Edge Impulse

Edge Impulse es una plataforma para desarrollar algoritmos de aprendizaje máquina enfocados a implementarse en sistemas embebidos como microcontroladores o computadoras con recursos reducidos. Tiene disponibles diversas herramientas que la hacen adecuada tanto para principiantes como usuarios avanzados. La practicidad de esta herramienta es que no necesitas involucrarte demasiado con el código, puedes implementar tu algoritmo con ingresar tu base de datos, ajustas los hiperparámetros y entrenas el programa (<https://www.edgeimpulse.com/>).

Para implementar un proyecto solo tiene que crear tu cuenta, indicar que tipo de proyecto quieres hacer (detección de objetos (<https://docs.edgeimpulse.com/docs/object-detection>), reconocimiento de voz, o procesar gestos de un acelerómetro) y hacer la adquisición de datos. Cuando tengas tu dataset puedes crear un Impulse, que es la herramienta que procesará y entrenará el modelo con los datos, para finalmente cargar el modelo a tu microcontrolador.

PROCEDIMIENTO

Para utilizar el aprendizaje profundo con ESP32-CAM este debe clasificar imágenes realizando los siguientes pasos:

1. Encontrar el conjunto de datos donde entrenar el modelo
2. Manipular el conjunto de datos si es necesario
3. Definir la arquitectura del modelo
4. Entrenar el modelo

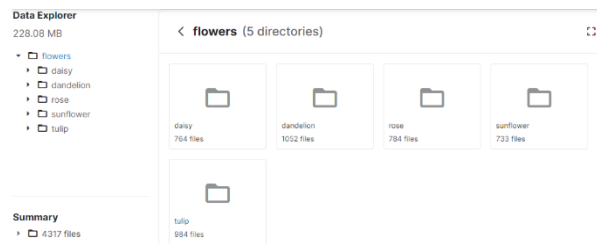
5. Desarrollar el código ESP32-CAM para ejecutar el modelo

Edge Impulse acelera la definición del modelo de aprendizaje profundo y la fase de entrenamiento produciendo un modelo TinyML listo para usar con el ESP32-CAM. Este modelo se basa en Tensorflow lite.

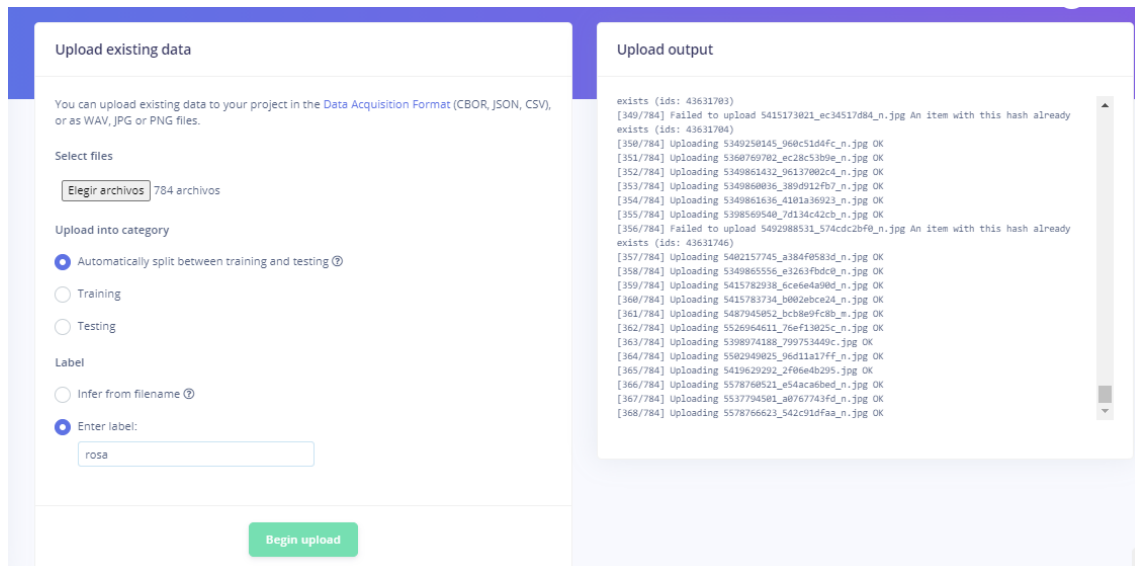
1. Definir el conjunto de datos para entrenar el modelo para usar con ESP32-CAM

Hay varios conjuntos de datos que podemos usar para entrenar nuestro modelo TinyML. Usaremos un conjunto de datos para reconocer y clasificar las flores usando ESP32-CAM y aprendizaje profundo. Por tanto, el dataset contiene varias flores agrupadas en clases. Kaggle es un buen punto de partida cuando busca un repositorio de conjuntos de datos o desea tener información sobre Machine Learning. El dataset Flowers-Recognition contiene 5 clases de flores diferentes:

- Margarita
- Diente de león
- Rosa
- Girasol
- Tulipán



Antes de continuar, es necesario crear una cuenta de Kaggle ('KAGGLE_USERNAME', 'KAGGLE_KEY').



2. Preparación del conjunto de datos para usar con Edge Impulse

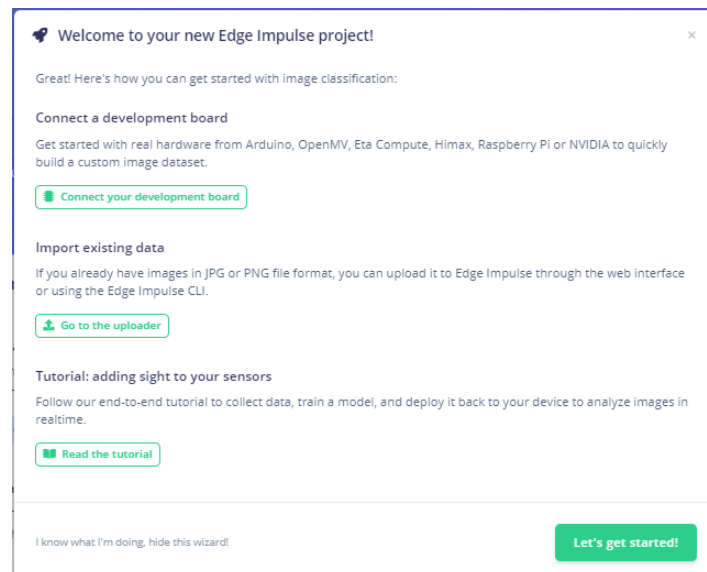
Antes de entrenar el modelo, es necesario cargar los datos en Edge Impulse. Para hacerlo, podemos instalar todo lo que necesite de forma local o puede utilizar Google Colab para hacerlo. Este tutorial usa esta segunda opción. Este es el enlace donde puede descargar el código. En primer lugar, tenemos que descargar el conjunto de datos de Kaggle:

```
!pip install -q kaggle
import os
os.environ['KAGGLE_USERNAME']='your_username'
os.environ['KAGGLE_KEY']='your_key'

# Flower dataset
!kaggle datasets download -d alxmamaev/flowers-recognition
!unzip flowers-recognition.zip
```

```
# Fruit dataset
!kaggle datasets download -d moltean/fruits
!unzip fruits.zip
```

https://github.com/survivingwithandroid/EdgeImpulse-fruit-recognition-ESP32-CAM/blob/main/EdgeImpulse_Fruit_Recognition.ipynb



Observe que descargamos dos conjuntos de datos diferentes: uno que contiene flores y otro que contiene frutas. Cuando entrenamos un modelo, es necesario seleccionar una clase que difiera de la clase que queremos clasificar.

Para la creación y carga del conjunto de datos para Edge Impulse seguir el tutorial (<https://docs.edgeimpulse.com/docs/object-detection>). En términos generales, definiremos una serie de muestras que queremos usar para clasificar nuestro modelo de aprendizaje profundo, y el código lo carga en el Edge Impulse.

DATA COLLECTED
3,423 items

LABELS
5

SAMPLE NAME	LABEL	ADDED	LENGTH
32465197894_71dde519...	tulipanes	Today, 15:10:09	-
26685647236_8211cb3e...	tulipanes	Today, 15:10:08	-
26685648806_c76dd583...	tulipanes	Today, 15:10:08	-
25965526231_941b6a21...	tulipanes	Today, 15:10:08	-
26564770956_ac4800ae...	tulipanes	Today, 15:10:08	-
25965548411_dbbe2626...	tulipanes	Today, 15:10:08	-
25429468133_6bfba75d...	tulipanes	Today, 15:10:08	-
25429501953_a1f9ce09e...	tulipanes	Today, 15:10:08	-

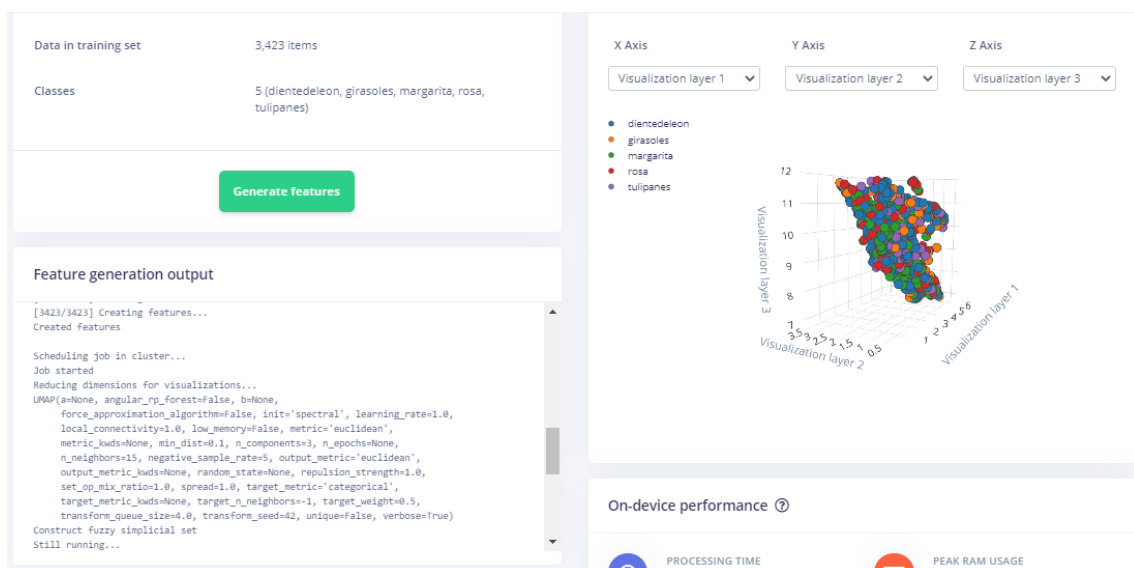
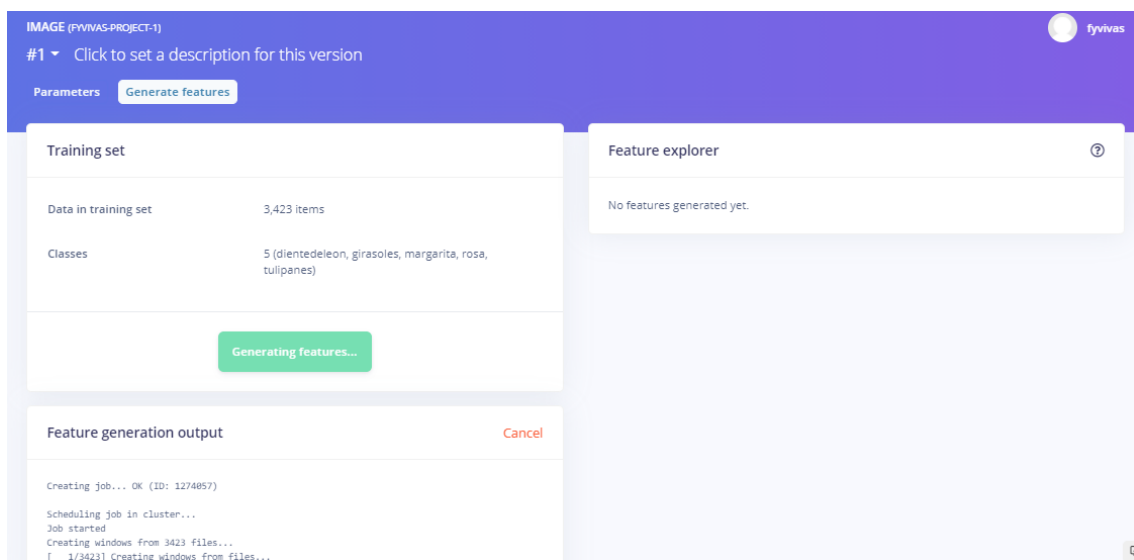
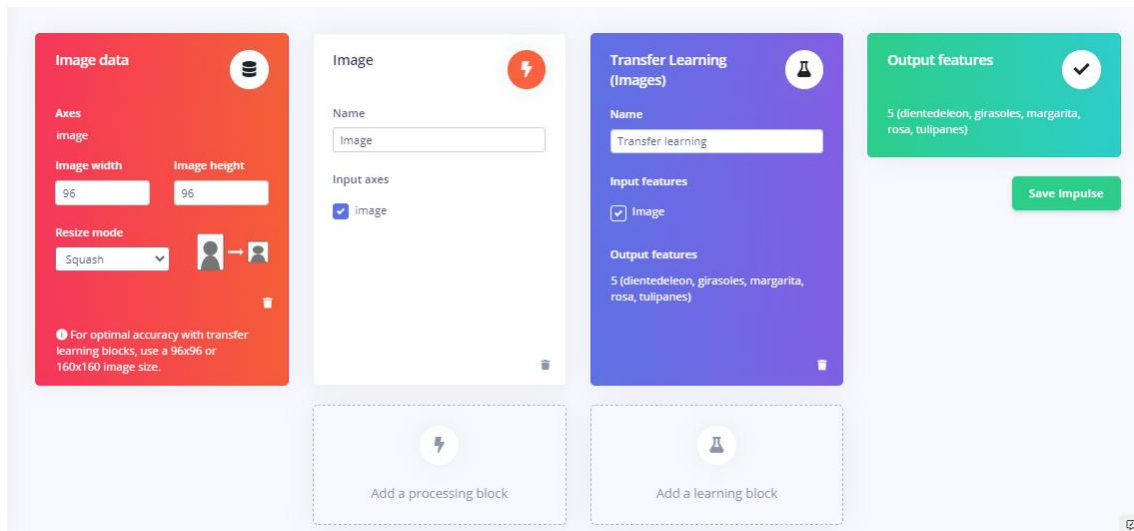
Record new data

No devices connected to the remote management API.

RAW DATA
Click on a sample to load...

3. Definir el modelo y entrenarlo para clasificar imágenes

Una vez que los datos están listos, podemos definir el modelo que usaremos para reconocer imágenes con ESP32-CAM.



Hay algunos aspectos a tener en cuenta:

- Como entrada del modelo, usaremos una imagen con 48×48 píxeles. Este es un aspecto importante. Tenga en cuenta que el ESP32-CAM (pero en general todos los dispositivos

como el ESP32) tiene una cantidad limitada de memoria. Por tanto, tenemos que reducir el tamaño de la imagen. Si usamos una imagen RGB el número de características que tiene que manejar la ESP32-CAM es $48 \times 48 \times 3$. Puede comprender fácilmente que al aumentar el tamaño de la imagen, el modelo no encajará en la ESP32-CAM.

- Usamos el aprendizaje de transferencia para entrenar el modelo.

4. Entrenamiento del modelo de aprendizaje automático mediante el aprendizaje por transferencia

Una vez extraídas las características, podemos entrenar el modelo. Estos son los parámetros utilizados para entrenar el modelo:

The image displays two screenshots from the TensorFlow Lite Studio application. The left screenshot shows the 'Neural Network settings' dialog box. Under 'Training settings', the 'Number of training cycles' is set to 300, the 'Learning rate' is 0.0005, 'Data augmentation' is checked, and the 'Minimum confidence rating' is 0.60. Under 'Neural network architecture', the 'Input layer' has 6,912 features, and the model is 'MobileNetV2 0.05 (final layer: 10 neurons, 0.1 dropout)'. The right screenshot shows the 'Training output' window. The log indicates: 'Creating job... OK (ID: 1274872)', 'Copying features from processing blocks...', 'Copying features from DSP block...', 'Copying features from DSP block OK', 'Copying features from processing blocks OK', and 'Job started'.

Observe que hemos utilizado MobileNetV2 0.05 porque el modelo debe caber en la memoria del dispositivo. La matriz de confusión se muestra a continuación:

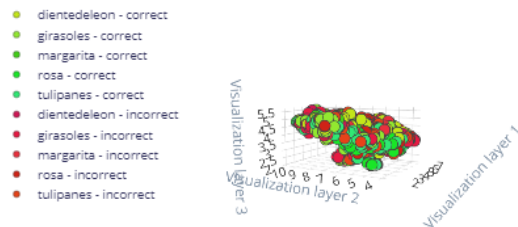
ACCURACY
55.2%

LOSS
1,15

Confusion matrix (validation set)

	DIENTEDELEON	GIRASOLES	MARGARITA	ROSA	TULIPANES
DIENTEDELEON	63.3%	8.9%	14.6%	5.7%	7.6%
GIRASOLES	13.2%	47.1%	12.4%	6.6%	20.7%
MARGARITA	22.5%	3.6%	55.8%	15.2%	2.9%
ROSA	10.5%	4.0%	14.5%	46.8%	24.2%
TULIPANES	7.6%	6.9%	13.2%	12.5%	59.7%
F1 SCORE	0.61	0.54	0.53	0.49	0.57

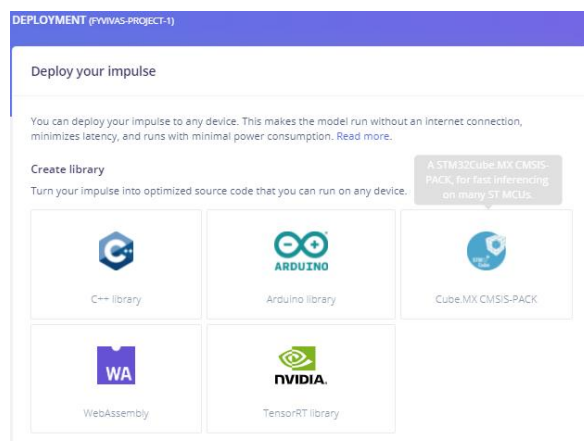
Feature explorer (full training set) ?



La precisión del modelo es del 55%. El último paso es la cuantificación del modelo y finalmente podemos descargar la librería para usarla con la ESP32 CAM. La biblioteca contiene todo lo que necesitamos para ejecutar la clasificación de imágenes usando ESP32 CAM.

5. Desplegar el modelo en la ESP32 CAM

Ahora puedes desplegar tu impulso en cualquier dispositivo. Esto hace que el modelo funcione sin conexión a Internet, minimiza la latencia y funciona con un consumo mínimo de energía. De esta manera se convierte su impulso en código fuente optimizado que puede ejecutar en cualquier dispositivo.



6. Cómo ejecutar la clasificación de imágenes en ESP32-CAM mediante aprendizaje profundo

Este es el momento de implementar el código en el dispositivo ESP32-CAM para ejecutar el modelo de clasificación mediante el aprendizaje profundo. Para hacerlo, podemos comenzar desde el ejemplo de búfer estático enviado con la biblioteca. Es necesario modificar el código de muestra para que podamos:

- Adquirir la imagen
- Adaptar el tamaño de la imagen al conjunto de datos

- Ejecutar el proceso de clasificación

El código se muestra a continuación (Basic-Image-Classification.ino):

```
#include <Arduino.h>
#include <WiFi.h>

#include "esp_http_server.h"
#include "esp_timer.h"

#include "img_converters.h"

#include "esp_camera.h"
#define CAMERA_MODEL_AI_THINKER // Has PSRAM

#include "camera_pins.h"

#include <-image_inference.h>

#include "esp_camera.h"
#define CAMERA_MODEL_AI_THINKER // Has PSRAM

#include "camera_pins.h"

// raw frame buffer from the camera
#define FRAME_BUFFER_COLS 240
#define FRAME_BUFFER_ROWS 240

#define CUTOOUT_COLS EI_CLASSIFIER_INPUT_WIDTH
#define CUTOOUT_ROWS EI_CLASSIFIER_INPUT_HEIGHT
const int cutout_row_start = (FRAME_BUFFER_ROWS - CUTOOUT_ROWS) / 2;
const int cutout_col_start = (FRAME_BUFFER_COLS - CUTOOUT_COLS) / 2;

#define PART_BOUNDARY "1234567890000000000000987654321"

static const char* _STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;boundary=" PART_BOUNDARY;
static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
static const char* _STREAM_PART = "Content-Type: image/jpeg\r\nContent-Length: %u\r\n\r\n";

httpd_handle_t camera_httpd = NULL;
httpd_handle_t stream_httpd = NULL;

const char* ssid = "your_ssid";
const char* password = "your_wifi_pwd";

camera_fb_t * fb = NULL;
uint8_t * _jpg_buf = NULL;

void r565_to_rgb(uint16_t color, uint8_t *r, uint8_t *g, uint8_t *b) {
    *r = (color & 0xF800) >> 8;
    *g = (color & 0x07E0) >> 3;
    *b = (color & 0x1F) << 3;
}

int cutout_get_data(size_t offset, size_t length, float *out_ptr) {
    // so offset and length naturally operate on the *cutout*, so we need to cut it out from the real framebuffer
    size_t bytes_left = length;
    size_t out_ptr_ix = 0;

    // read byte for byte
    while (bytes_left != 0) {
        // find location of the byte in the cutout
        size_t cutout_row = floor(offset / CUTOOUT_COLS);
        size_t cutout_col = offset - (cutout_row * CUTOOUT_COLS);

        // then read the value from the real frame buffer
        size_t frame_buffer_row = cutout_row + cutout_row_start;
```

```

    size_t frame_buffer_col = cutout_col + cutout_col_start;

    uint16_t pixelTemp = fb->buf[(frame_buffer_row * FRAME_BUFFER_COLS) + frame_buffer_col];

    uint16_t pixel = (pixelTemp>>8) | (pixelTemp<<8);

    uint8_t r, g, b;
    r565_to_rgb(pixel, &r, &g, &b);
    float pixel_f = (r << 16) + (g << 8) + b;
    out_ptr[out_ptr_ix] = pixel_f;

    out_ptr_ix++;
    offset++;
    bytes_left--;
}

// and done!
return 0;
}

void classify() {
    ei_printf("Edge Impulse standalone inferencing (Arduino)\n");

    ei_impulse_result_t result = { 0 };

    // Convert to RGB888
    //fmt2rgb888(fb->buf, fb->len, PIXFORMAT_RGB888, _jpg_buf);

    //Serial.println("Signal...");
    // Set up pointer to look after data, crop it and convert it to RGB888
    signal_t signal;
    signal.total_length = CUTOUT_COLS * CUTOUT_ROWS;
    signal.get_data = &cutout_get_data;

    // Feed signal to the classifier
    EI_IMPULSE_ERROR res = run_classifier(&signal, &result, false /* debug */);

    // Returned error variable "res" while data object.array in "result"
    ei_printf("run_classifier returned: %d\n", res);
    if (res != 0) return;

    // print the predictions
    ei_printf("Predictions ");
    ei_printf("(DSP: %d ms., Classification: %d ms., Anomaly: %d ms.)",
        result.timing.dsp, result.timing.classification, result.timing.anomaly);
    ei_printf(": \n");

    // Print short form result data
    ei_printf("[");
    for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
        ei_printf("%.5f", result.classification[ix].value);
        #if EI_CLASSIFIER_HAS_ANOMALY == 1
            ei_printf(", ");
        #else
            if (ix != EI_CLASSIFIER_LABEL_COUNT - 1) {
                ei_printf(", ");
            }
        #endif
    }
    ei_printf("]");

    #if EI_CLASSIFIER_HAS_ANOMALY == 1
        ei_printf("%.3f", result.anomaly);
    #endif
    ei_printf("\n");

    // human-readable predictions
    for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
        ei_printf("  %s: %.5f\n", result.classification[ix].label, result.classification[ix].value);
    }
    #if EI_CLASSIFIER_HAS_ANOMALY == 1

```

```

    ei_printf("  anomaly score: %.3f\n", result.anomaly);
#endif
}

static esp_err_t capture_handler(httpd_req_t *req){
    Serial.println("Capture image");

    esp_err_t res = ESP_OK;
    fb = esp_camera_fb_get();
    if (!fb) {
        Serial.println("Camera capture failed");
        httpd_resp_send_500(req);
        return ESP_FAIL;
    }

    classify();

    httpd_resp_set_type(req, "image/jpeg");
    httpd_resp_set_hdr(req, "Content-Disposition", "inline; filename=capture.jpg");
    httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");

    res = httpd_resp_send(req, (const char *)fb->buf, fb->len);
    esp_camera_fb_return(fb);

    return res;
}

static esp_err_t page_handler(httpd_req_t *req) {

    httpd_resp_set_type(req, "text/html");
    httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
    // httpd_resp_send(req, page, sizeof(page));
}

static esp_err_t stream_handler(httpd_req_t *req){
    camera_fb_t * fb = NULL;
    esp_err_t res = ESP_OK;
    size_t _jpg_buf_len = 0;
    uint8_t * _jpg_buf = NULL;
    char * part_buf[64];

    res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
    if(res != ESP_OK){
        return res;
    }

    httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");

    while(true){
        fb = esp_camera_fb_get();
        if (!fb) {
            Serial.println("Camera capture failed");
            res = ESP_FAIL;
        } else {
            if(fb->format != PIXFORMAT_JPEG){
                bool jpeg_converted = frame2jpg(fb, 80, &_jpg_buf, &_jpg_buf_len);
                esp_camera_fb_return(fb);
                fb = NULL;
                if(!jpeg_converted){
                    Serial.println("JPEG compression failed");
                    res = ESP_FAIL;
                }
            } else {
                _jpg_buf_len = fb->len;
                _jpg_buf = fb->buf;
            }
        }
        if(res == ESP_OK){
            res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY, strlen(_STREAM_BOUNDARY));
        }
    }
}

```

```

    if(res == ESP_OK){
        size_t hlen = snprintf((char *)part_buf, 64, _STREAM_PART, _jpg_buf_len);
        res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
    }
    if(res == ESP_OK){
        res = httpd_resp_send_chunk(req, (const char *)_jpg_buf, _jpg_buf_len);
    }
    if(fb){
        esp_camera_fb_return(fb);
        fb = NULL;
        _jpg_buf = NULL;
    } else if(_jpg_buf){
        free(_jpg_buf);
        _jpg_buf = NULL;
    }
    if(res != ESP_OK){
        break;
    }
}
return res;
}

```

```

void startCameraServer(){
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();

```

```

    httpd_uri_t index_uri = {
        .uri      = "/",
        .method   = HTTP_GET,
        .handler  = stream_handler,
        .user_ctx = NULL
    };

```

```

    httpd_uri_t capture_uri = {
        .uri      = "/capture",
        .method   = HTTP_GET,
        .handler  = capture_handler,
        .user_ctx = NULL
    };

```

```

    Serial.printf("Starting web server on port: %d\n", config.server_port);
    if (httpd_start(&camera_httpd, &config) == ESP_OK) {
        httpd_register_uri_handler(camera_httpd, &capture_uri);
        //httpd_register_uri_handler(camera_httpd, &page_uri);
    }

```

```

    // start stream using another webserver
    config.server_port += 1;
    config.ctrl_port += 1;
    Serial.printf("Starting stream server on port: %d\n", config.server_port);
    if (httpd_start(&stream_httpd, &config) == ESP_OK) {
        httpd_register_uri_handler(stream_httpd, &index_uri);
    }

```

```

}

```

```

void setup() {
    Serial.begin(9600);
    Serial.setDebugOutput(true);
    Serial.println();

```

```

    camera_config_t config;
    config.ledc_channel = LEDC_CHANNEL_0;
    config.ledc_timer = LEDC_TIMER_0;
    config.pin_d0 = Y2_GPIO_NUM;
    config.pin_d1 = Y3_GPIO_NUM;
    config.pin_d2 = Y4_GPIO_NUM;
    config.pin_d3 = Y5_GPIO_NUM;
    config.pin_d4 = Y6_GPIO_NUM;
    config.pin_d5 = Y7_GPIO_NUM;
    config.pin_d6 = Y8_GPIO_NUM;

```

```

config.pin_d7 = Y9_GPIO_NUM;
config.pin_xclk = XCLK_GPIO_NUM;
config.pin_pclk = PCLK_GPIO_NUM;
config.pin_vsync = VSYNC_GPIO_NUM;
config.pin_href = HREF_GPIO_NUM;
config.pin_sscb_sda = SIOD_GPIO_NUM;
config.pin_sscb_scl = SIOC_GPIO_NUM;
config.pin_pwdn = PWDN_GPIO_NUM;
config.pin_reset = RESET_GPIO_NUM;
config.xclk_freq_hz = 20000000;
config.pixel_format = PIXFORMAT_JPEG;

// if PSRAM IC present, init with UXGA resolution and higher JPEG quality
//           for larger pre-allocated frame buffer.
if(psramFound()){
  config.frame_size = FRAMESIZE_240X240;
  config.jpeg_quality = 10;
  config.fb_count = 2;
} else {
  config.frame_size = FRAMESIZE_240X240;
  config.jpeg_quality = 12;
  config.fb_count = 1;
}

#ifdef CAMERA_MODEL_ESP_EYE
pinMode(13, INPUT_PULLUP);
pinMode(14, INPUT_PULLUP);
#endif

// camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
  Serial.printf("Camera init failed with error 0x%x", err);
  return;
}

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
}
// drop down frame size for higher initial frame rate
s->set_framesize(s, FRAMESIZE_240X240);

#ifdef CAMERA_MODEL_M5STACK_WIDE
s->set_vflip(s, 1);
s->set_hmirror(s, 1);
#endif

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");

startCameraServer();

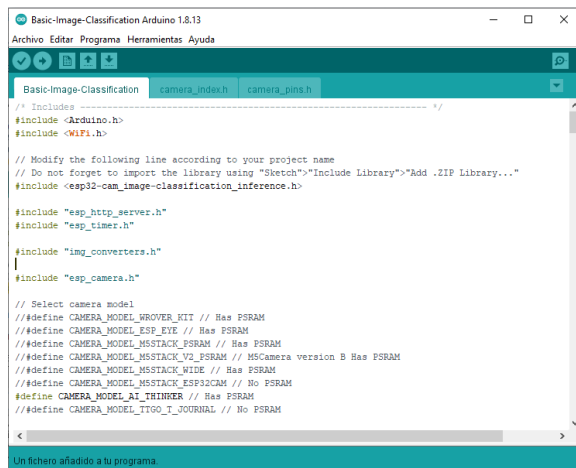
Serial.print("Camera Ready! Use 'http://");
Serial.print(WiFi.localIP());
Serial.println("' to connect");
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(10);
}

```

El código incluye un servidor web para que pueda ejecutar la clasificación desde una interfaz web (el código fuente HTML no está incluido). Además, implementa un servidor de transmisión de video que envía video a la página web.

- a. Abra el archivo Basic-Image-Classification.ino en la carpeta / Basic-Image-Classification.
- b. Importe la biblioteca .zip que ha descargado de Edge Impulse Studio



```
Basic-Image-Classification Arduino 1.8.13
Archivo Editor Programa Herramientas Ayuda

/* Includes */
#include <Arduino.h>
#include <WiFi.h>

// Modify the following line according to your project name
// Do not forget to import the library using "Sketch">"Include Library">"Add .ZIP Library..."
#include <esp32-cam_image-classification_inference.h>

#include "esp_http_server.h"
#include "esp_timer.h"

#include "img_converters.h"
#include "esp_camera.h"

// Select camera model
// #define CAMERA_MODEL_WROVER_KIT // Has PSRAM
// #define CAMERA_MODEL_ESP_EYE // Has PSRAM
// #define CAMERA_MODEL_MSSTACK_PSRAM // Has PSRAM
// #define CAMERA_MODEL_MSSTACK_V2_PSRAM // M5Camera version B Has PSRAM
// #define CAMERA_MODEL_MSSTACK_WIDE // Has PSRAM
// #define CAMERA_MODEL_MSSTACK_ESP32CAM // No PSRAM
// #define CAMERA_MODEL_AI_THINKER // Has PSRAM
// #define CAMERA_MODEL_TIGRO_T_JOURNAL // No PSRAM

Un fichero añadido a tu programa.
```

- c. Cambie la línea `#include <esp32-cam_image-Classification_inference.h>` de acuerdo con el nombre de su proyecto.
- d. Compile e implemente el código en su tarjeta
- e. Abra el monitor serial y use la IP proporcionada para capturar una imagen y ejecutar la inferencia

7. Cómo alimentar el modelo con imágenes ESP32-CAM

Lo primero es adaptar la imagen captada por el ESP32-CAM para que podamos pasarla al modelo que hemos entrenado anteriormente. Hay dos aspectos a considerar:

- El tamaño de la imagen
- La codificación de colores

Las imágenes del conjunto de datos son 48×48 . La mejor resolución es 240×240 . En la función `cutout_get_data`, la imagen cambia de tamaño a 48×48 desde el tamaño de la imagen 240×240 y se convierte a RGB888.

8. Implementar el modelo de clasificación en el ESP32-CAM

El último paso es ejecutar el proceso de clasificación en el ESP32-CAM. En el ejemplo anterior, la clasificación se activa desde la interfaz web, pero puede modificar fácilmente el código si no desea utilizarlo:

- `signal_t signal;`
- `signal.total_length = CUTOUT_COLS * CUTOUT_ROWS;`
- `signal.get_data = & cutout_get_data;`
- Lenguaje de código: C ++ (cpp)

Aquí es donde invocamos el método que adapta el tamaño y el color de la imagen.

ESP32 CAM



classify



```
capture image
Edge Impulse standalone inferencing (Arduino)
run_classifier returned: 0
Predictions (DSP: 3 ms., Classification: 848 ms., Anomaly: 0 ms.):
[0.00391, 0.71094, 0.28125, 0.00000]
  rose: 0.00391
  sunflower: 0.71094
  tulip: 0.28125
  unknown: 0.00000
```

ESP32-CAM Image Classification using Machine Learning

El ESP32-CAM tiene la capacidad de adquirir videos e imágenes, usaremos esta capacidad para clasificar imágenes usando el aprendizaje automático. Combinando la capacidad de visión ESP32-CAM con el aprendizaje automático en la nube, en este tutorial, llevaremos el poder de la visión por computadora a un dispositivo diminuto.

La clasificación de imágenes de Machine Learning es la tarea de extraer información de una imagen utilizando un modelo entrenado.

Para clasificar una imagen, el ESP32-CAM se conectará a una plataforma de aprendizaje automático en la nube llamada Clarifai.com (puede crear una cuenta gratis).

Cómo funciona la clasificación de imágenes ESP32-CAM

- 1) Adquirir imágenes usando ESP32-CAM
- 2) Codificar la imagen en base64
- 3) Invocar una API expuesta por la plataforma en la nube de aprendizaje automático, enviando la imagen adquirida por el ESP32
- 4) Analizar la respuesta y extraer la información

La ventaja de este método es que no es necesario entrenar un modelo para clasificar las imágenes por nosotros mismos, pero el ESP32-CAM utiliza un modelo previamente entrenado construido por Clarifai (<https://www.clarifai.com/models/ai-food-recognition>). Este modelo de aprendizaje automático es capaz de identificar y clasificar más de 10,000 conceptos. A partir de las imágenes captadas por el ESP32-CAM, aplicando el reconocimiento de imágenes, es posible extraer información como:

- si hay una persona o no
- interior o exterior
- objetos
- estados de ánimo

INICIALIZANDO EL ESP32-CAM

Primero, es necesario seleccionar su tipo de CAM. Cámbielo de acuerdo con su ESP32-CAM. Luego, tenemos que configurar el ssid wifi y el password wifi para que se conecte al wifi.

Es importante notar que hemos reducido la resolución de la cámara porque la codificación base64 requiere mucha memoria. De todos modos, no necesitamos una resolución mayor para reconocer la imagen.

```
#include "Arduino.h"
#include "esp_camera.h"
#include <WiFi.h>

// Select camera model
// #define CAMERA_MODEL_WROVER_KIT // Has PSRAM
// #define CAMERA_MODEL_ESP_EYE // Has PSRAM
// #define CAMERA_MODEL_M5STACK_PSRAM // Has PSRAM
// #define CAMERA_MODEL_M5STACK_WIDE // Has PSRAM
#define CAMERA_MODEL_AI_THINKER // Has PSRAM
// #define CAMERA_MODEL_TTGO_T_JOURNAL // No PSRAM
```



```
#include "camera_pins.h"
```

```
const char* ssid = "your_ssid";  
const char* password = "wifi_password";
```

```
void setup() {  
  Serial.begin(9600);  
  Serial.setDebugOutput(true);  
  Serial.println();
```

```
  camera_config_t config;  
  config.ledc_channel = LEDC_CHANNEL_0;  
  config.ledc_timer = LEDC_TIMER_0;  
  config.pin_d0 = Y2_GPIO_NUM;  
  config.pin_d1 = Y3_GPIO_NUM;  
  config.pin_d2 = Y4_GPIO_NUM;  
  config.pin_d3 = Y5_GPIO_NUM;  
  config.pin_d4 = Y6_GPIO_NUM;  
  config.pin_d5 = Y7_GPIO_NUM;  
  config.pin_d6 = Y8_GPIO_NUM;  
  config.pin_d7 = Y9_GPIO_NUM;  
  config.pin_xclk = XCLK_GPIO_NUM;  
  config.pin_pclk = PCLK_GPIO_NUM;  
  config.pin_vsync = VSYNC_GPIO_NUM;  
  config.pin_href = HREF_GPIO_NUM;  
  config.pin_sscb_sda = SIOD_GPIO_NUM;  
  config.pin_sscb_scl = SIOC_GPIO_NUM;  
  config.pin_pwdn = PWDN_GPIO_NUM;  
  config.pin_reset = RESET_GPIO_NUM;  
  config.xclk_freq_hz = 20000000;  
  config.pixel_format = PIXFORMAT_JPEG;
```

```
  // if PSRAM IC present, init with UXGA resolution and higher JPEG quality  
  //      for larger pre-allocated frame buffer.  
  if(psramFound()){  
    config.frame_size = FRAMESIZE_QVGA;  
    config.jpeg_quality = 10;  
    config.fb_count = 2;  
  } else {  
    config.frame_size = FRAMESIZE_QVGA;  
    config.jpeg_quality = 12;  
    config.fb_count = 1;  
  }  
}
```

```
#if defined(CAMERA_MODEL_ESP_EYE)  
  pinMode(13, INPUT_PULLUP);  
  pinMode(14, INPUT_PULLUP);  
#endif
```

```
  // camera init  
  esp_err_t err = esp_camera_init(&config);  
  if (err != ESP_OK) {  
    Serial.printf("Camera init failed with error 0x%x", err);  
    return;  
  }
```

```
#if defined(CAMERA_MODEL_M5STACK_WIDE)  
  s->set_vflip(s, 1);  
  s->set_hmirror(s, 1);  
#endif
```

```

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");

classifyImage();

}

```

ADQUIRIR UNA IMAGEN USANDO ESP32-CAM

camera_fb_t contiene la información de la imagen y los datos que representan la imagen capturada. Usando el método esp_camera_fb_get (), la ESP32-CAM captura la imagen. Finalmente, codificamos en base64 la imagen. fb-> buf contiene los datos y fb-> len es el tamaño del búfer.

```

void classifyImage() {

// Capture picture
camera_fb_t * fb = NULL;
fb = esp_camera_fb_get();

if(!fb) {
  Serial.println("Camera capture failed");
  return;
}

size_t size = fb->len;
String buffer = base64::encode((uint8_t *) fb->buf, fb->len);
....
}

```

APLICACIÓN DE RECONOCIMIENTO DE IMÁGENES MEDIANTE ESP32-CAM

Una vez que se captura la imagen, el siguiente paso es reconocer la imagen y extraer información de ella. Incluso si es posible utilizar el modelo de aprendizaje automático que se ejecuta en ESP32, queremos utilizar una plataforma de aprendizaje automático en la nube que utilice modelos previamente entrenados. Para lograrlo, es necesario invocar una API y enviar la imagen codificada. En el método classifyImage agregue las siguientes líneas:

```

String payload = "{\"inputs\": [{\"data\": {\"image\": {\"base64\": \"\" + buffer + \"\"}}}]\"}";

buffer = "";
// Uncomment this if you want to show the payload
// Serial.println(payload);

esp_camera_fb_return(fb);

// Generic model
String model_id = "aaa03c23b3724a16a56b629203edc62c";

HTTPClient http;
http.begin("https://api.clarifai.com/v2/models/" + model_id + "/outputs");
http.addHeader("Content-Type", "application/json");
http.addHeader("Authorization", "Key your_key");
int response_code = http.POST(payload);

```

El código muestra la identificación del modelo que queremos usar para reconocer la imagen y la clave de autorización.

AÑADIENDO CAPACIDAD DE VISIÓN POR COMPUTADORA AL ESP32-CAM

Después de enviar la imagen base64 a la plataforma en la nube de aprendizaje automático, obtenemos la respuesta con todos los conceptos extraídos de la imagen. Los conceptos son etiquetas que se utilizan para clasificar la imagen y reconocerla. Usando las etiquetas, obtenemos una descripción de la imagen. Cada etiqueta tiene una probabilidad.

```
/ Parse the json response: Arduino assistant
const int jsonSize = JSON_ARRAY_SIZE(1) + JSON_ARRAY_SIZE(20) + 3*JSON_OBJECT_SIZE(1) +
6*JSON_OBJECT_SIZE(2) + JSON_OBJECT_SIZE(3) + 20*JSON_OBJECT_SIZE(4) + 2*JSON_OBJECT_SIZE(6);
DynamicJsonDocument doc(jsonSize);
deserializeJson(doc, response);
```

```
for (int i=0; i < 10; i++) {
  const name = doc["outputs"][0]["data"]["concepts"][i]["name"];
  const float p = doc["outputs"][0]["data"]["concepts"][i]["value"];
```

```
  Serial.println("=====");
  Serial.print("Name:");
  Serial.println(name[i]);
  Serial.print("Prob:");
  Serial.println(p);
  Serial.println();
}
```

PRUEBA EL RECONOCIMIENTO DE IMÁGENES

Estamos listos para probar cómo funciona la clasificación de imágenes con nuestro ESP32-CAM. Después de cargar el sketch en la ESP32-CAM, debe presionar el botón de reinicio para iniciar el proceso de reconocimiento de imagen. Para visualizar la imagen capturada, puede usar un decodificador de base64 a imagen pasando el flujo de bytes codificado que representa la imagen.



```
=====
Name: flower
Prob: 0.99
=====
Name: no person
Prob: 0.98
=====
Name: flora
Prob: 0.97
=====
Name: nature
Prob: 0.96
=====
Name: leaf
Prob: 0.95
=====
```



```
=====
Name: no person
Prob: 1.00
=====
Name: ball
Prob: 1.00
=====
Name: recreation
Prob: 0.99
=====
Name: one
Prob: 0.99
=====
Name: soccer
Prob: 0.98
=====
```