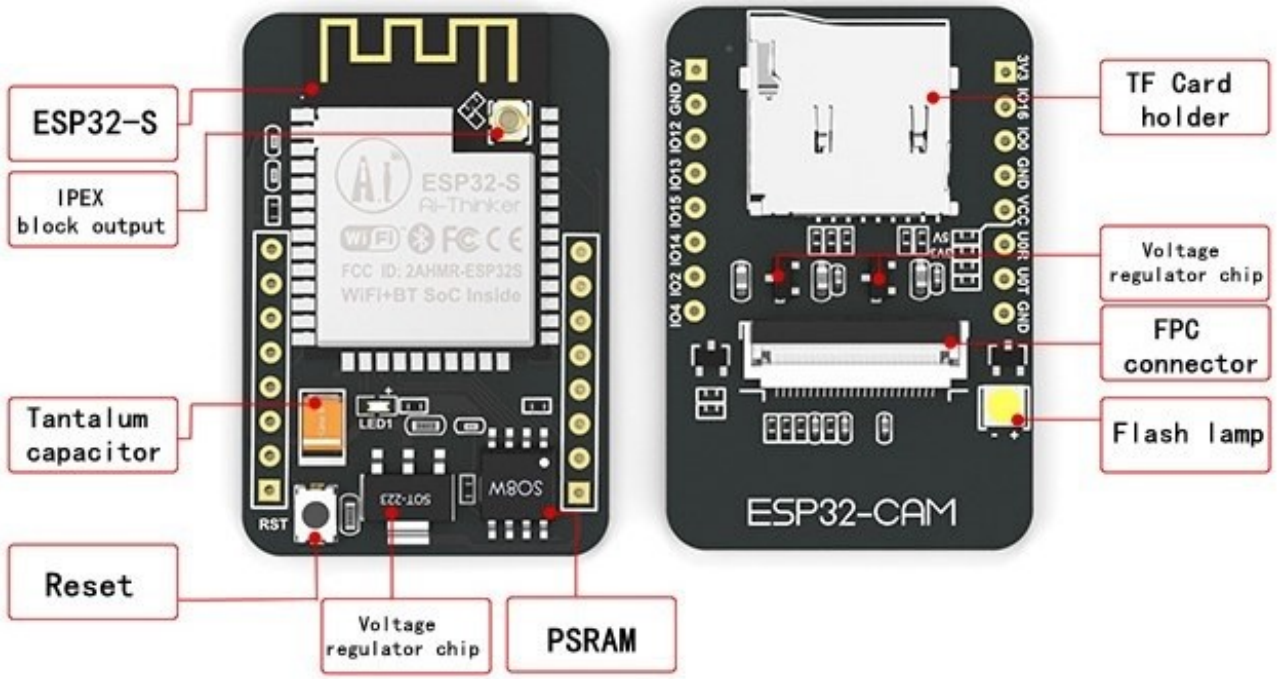
Part 26
-
ESP32-CAM

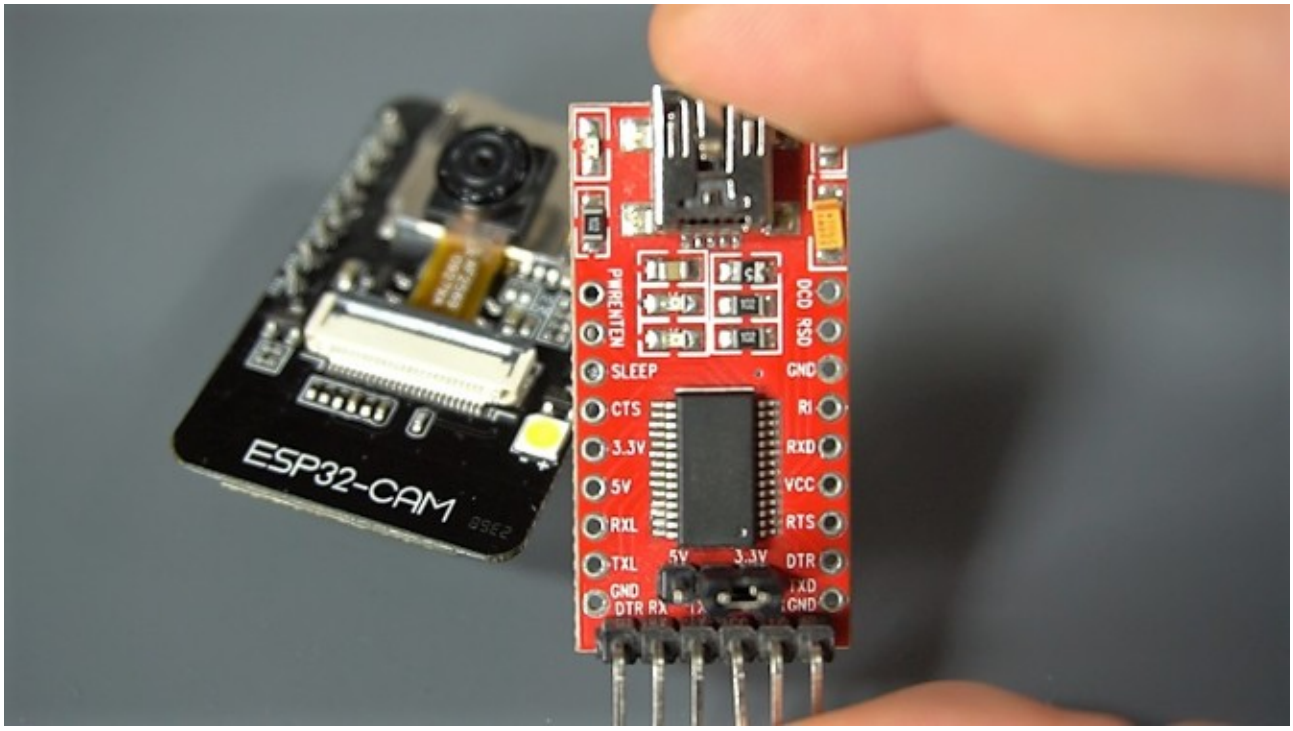
Version: 2021-10-15

Introducing the ESP32-CAM

The ESP32-CAM is a very small camera module with the ESP32-S chip. Besides the OV2640 camera, and several GPIOs to connect peripherals, it also features a microSD card slot that can be useful to store images taken with the camera or to store files to serve to clients.



The ESP32-CAM doesn’t come with a USB connector, so you need an FTDI programmer to upload code through the U0R and U0T pins (serial pins).

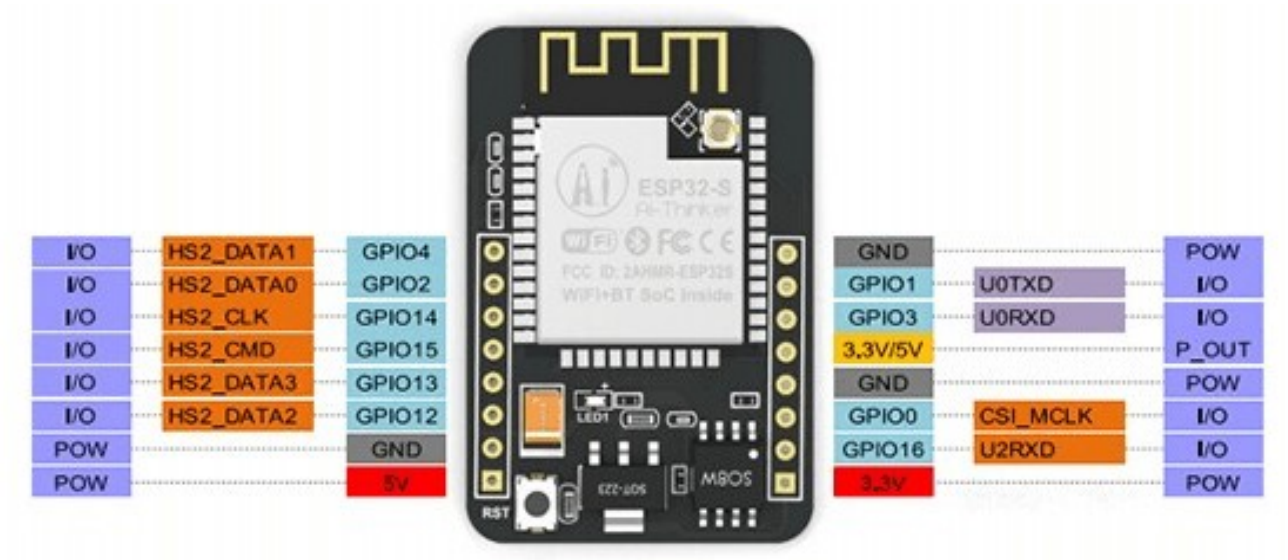


Features

- The smallest 802.11b/g/n Wi-Fi BT SoC module
- Low power 32-bit CPU,can also serve the application processor
- Up to 160MHz clock speed, summary computing power up to 600 DMIPS
- Built-in 520 KB SRAM, external 4MPSRAM
- Supports UART/SPI/I2C/PWM/ADC/DAC
- Support OV2640 and OV7670 cameras, built-in flash lamp
- Support image WiFI upload
- Support TF card
- Supports multiple sleep modes
- Embedded Lwip and FreeRTOS
- Supports STA/AP/STA+AP operation mode
- Support Smart Config/AirKiss technology
- Support for serial port local and remote firmware upgrades (FOTA)

ESP32-CAM Pinout

The following figure shows the ESP32-CAM pinout (AI-Thinker module).



There are three GND pins and two pins for power: either 3.3V or 5V.
GPIO 1 and GPIO 3 are the serial pins. You need these pins to upload code to your board. Additionally, GPIO 0 also plays an important role, since it determines whether the ESP32 is in flashing mode or not. When GPIO 0 is connected to GND, the ESP32 is in flashing mode.
The following pins are internally connected to the microSD card reader:

- GPIO 14 : CLK
- GPIO 15 : CMD
- GPIO 2 : Data 0
- GPIO 4 : Data 1 (also connected to the on-board LED)
- GPIO 12 : Data 2
- GPIO 13 : Data 3

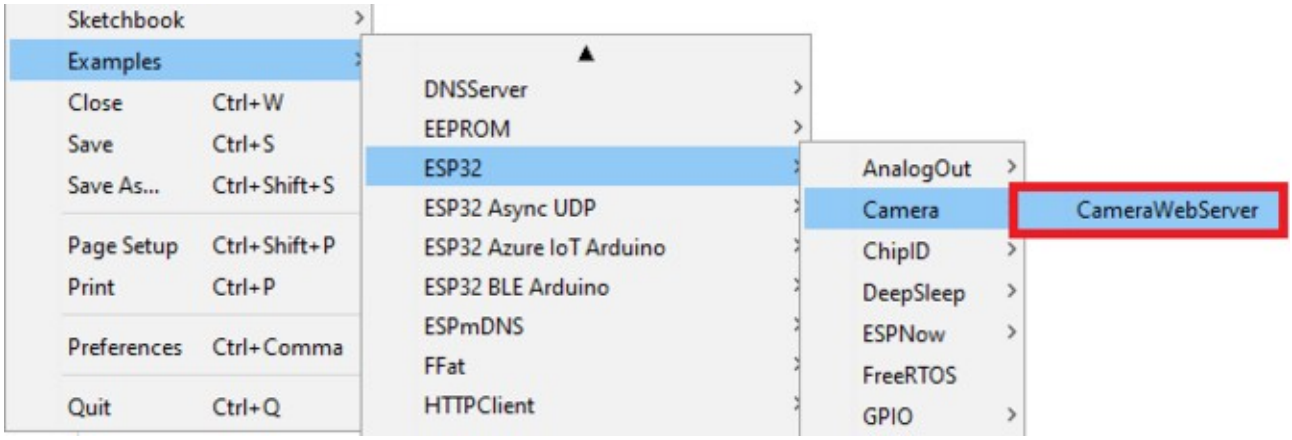
ESP32-CAM Video Streaming and Face Recognition with Arduino IDE

Video Streaming Server

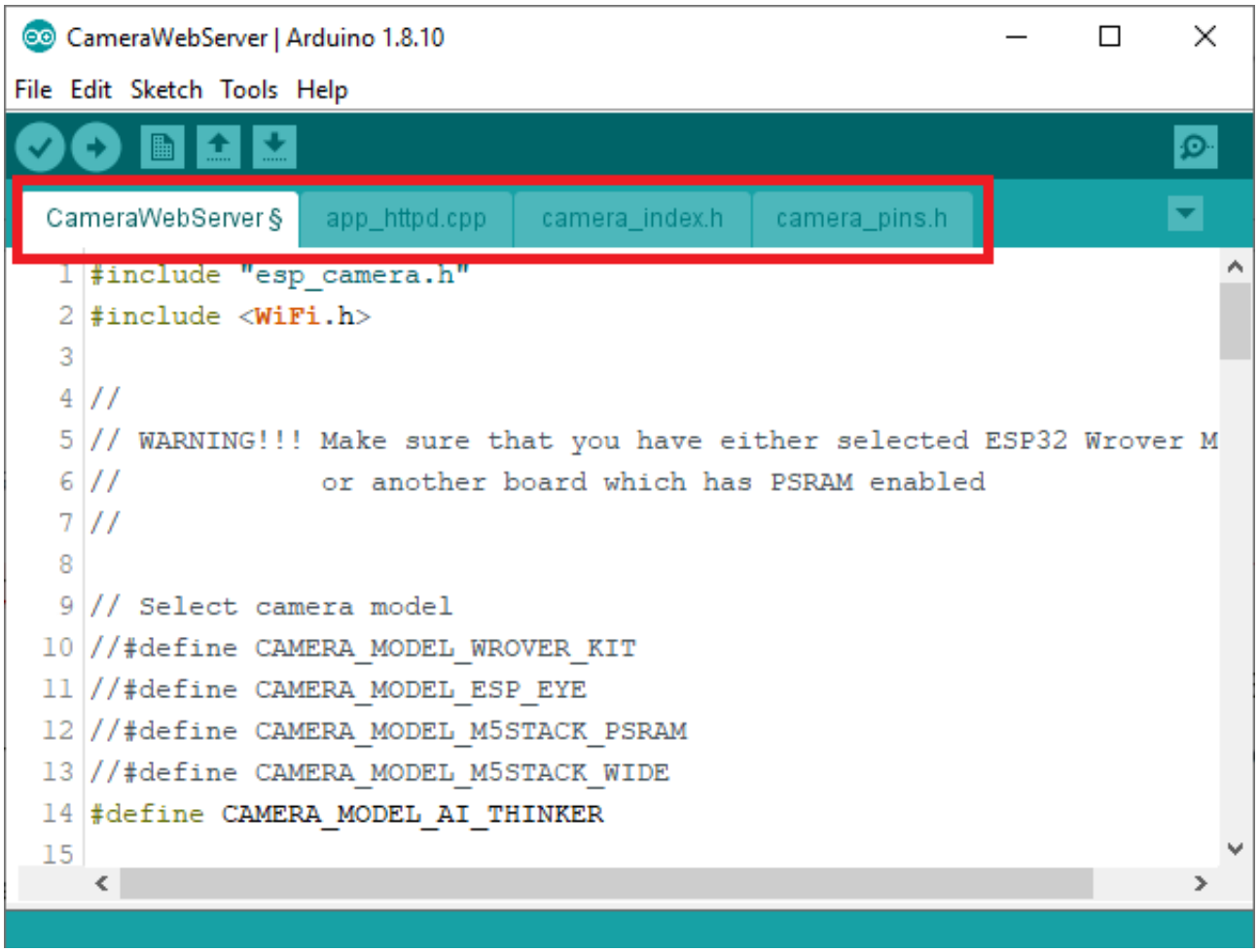
Follow the next steps to build a video streaming web server with the ESP32-CAM that you can access on your local network.

CameraWebServer Example Code

In your Arduino IDE, go to File > Examples > ESP32 > Camera and open the CameraWebServer example.



The following code should load.



Before uploading the code, you need to insert your network credentials in the following variables:

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

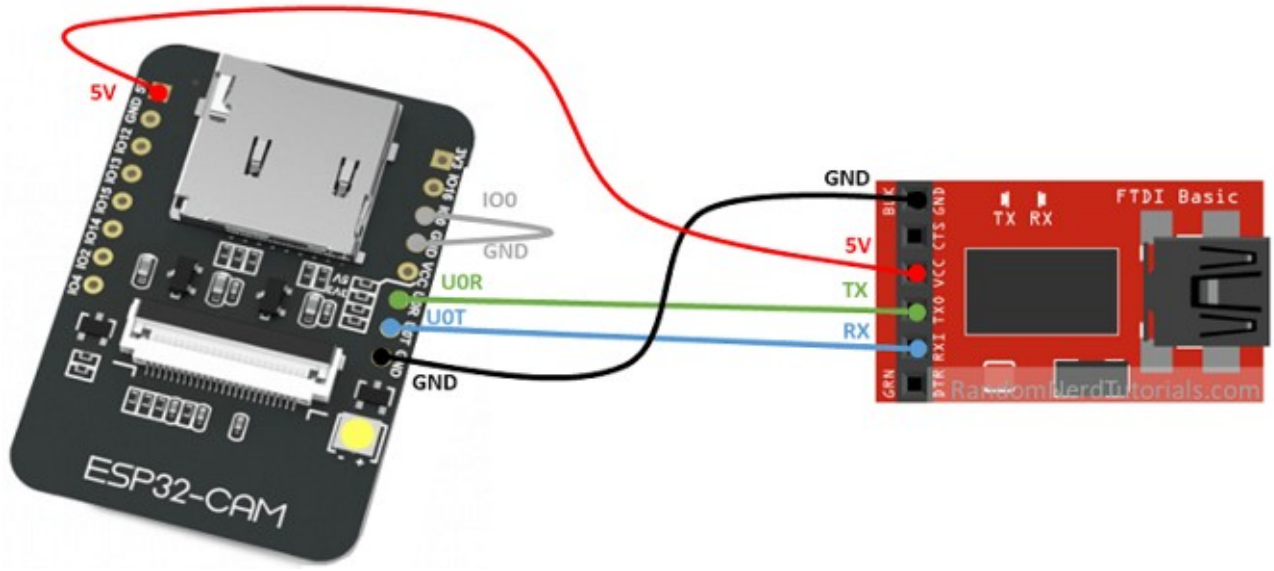
Then, make sure you select the right camera module. In this case, we're using the AI-THINKER Model.

So, comment all the other models and uncomment this one:

```
// Select camera model
//#define CAMERA_MODEL_WROVER_KIT
//#define CAMERA_MODEL_ESP_EYE
//#define CAMERA_MODEL_M5STACK_PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE
#define CAMERA_MODEL_AI_THINKER
```

If none of these correspond to the camera you're using, you need to add the pin assignment for your specific board in the camera_pins.h tab.
Now, the code is ready to be uploaded to your ESP32.

ESP32-CAM Upload Code
Connect the ESP32-CAM board to your computer using an FTDI programmer. Follow the next schematic diagram:



Many FTDI programmers have a jumper that allows you to select 3.3V or 5V. Make sure the jumper is in the right place to select 5V.
Important: GPIO 0 needs to be connected to GND so that you're able to upload code.

ESP32-CAM	FTDI Programmer
GND	GND
5V	VCC (5V)
U0R	TX
U0T	RX
GPIO 0	GND

To upload the code, follow the next steps:

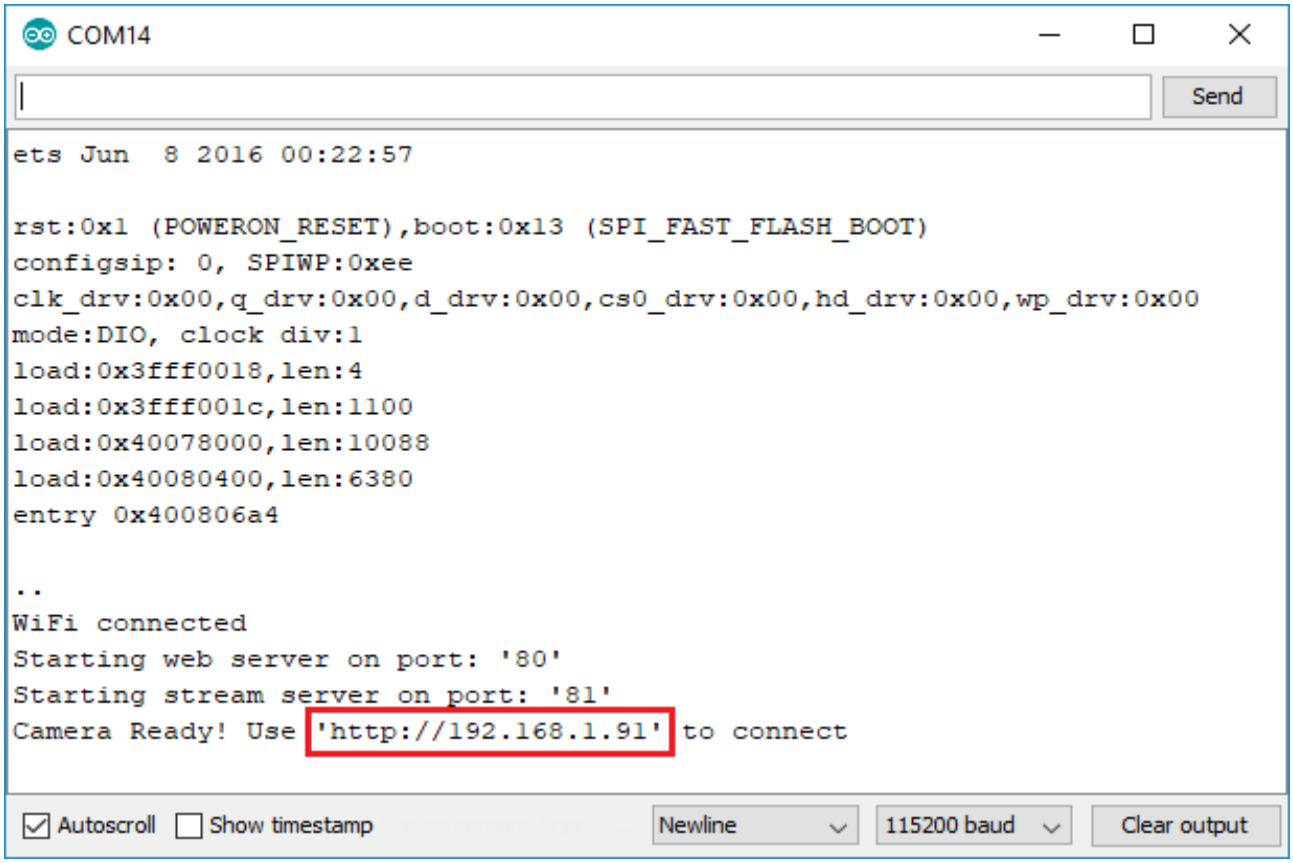
- Go to Tools > Board and select AI-Thinker ESP32-CAM.
- Go to Tools > Port and select the COM port the ESP32 is connected to.
- Then, click the upload button to upload the code.

When you start to see these dots on the debugging window as shown below, press the ESP32-CAM on-board RST button.

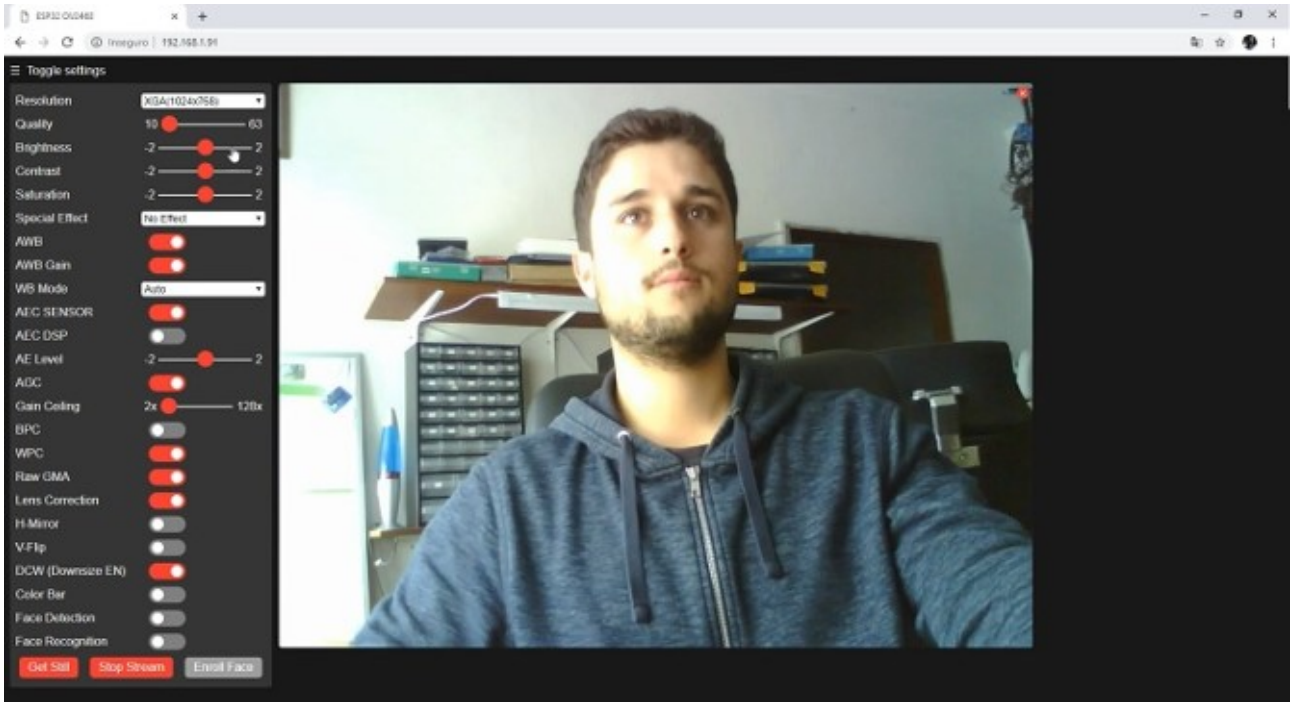
After a few seconds, the code should be successfully uploaded to your board.

Getting the IP address

After uploading the code, disconnect GPIO 0 from GND.
Open the Serial Monitor at a baud rate of 115200. Press the ESP32-CAM on-board Reset button.
The ESP32 IP address should be printed in the Serial Monitor.



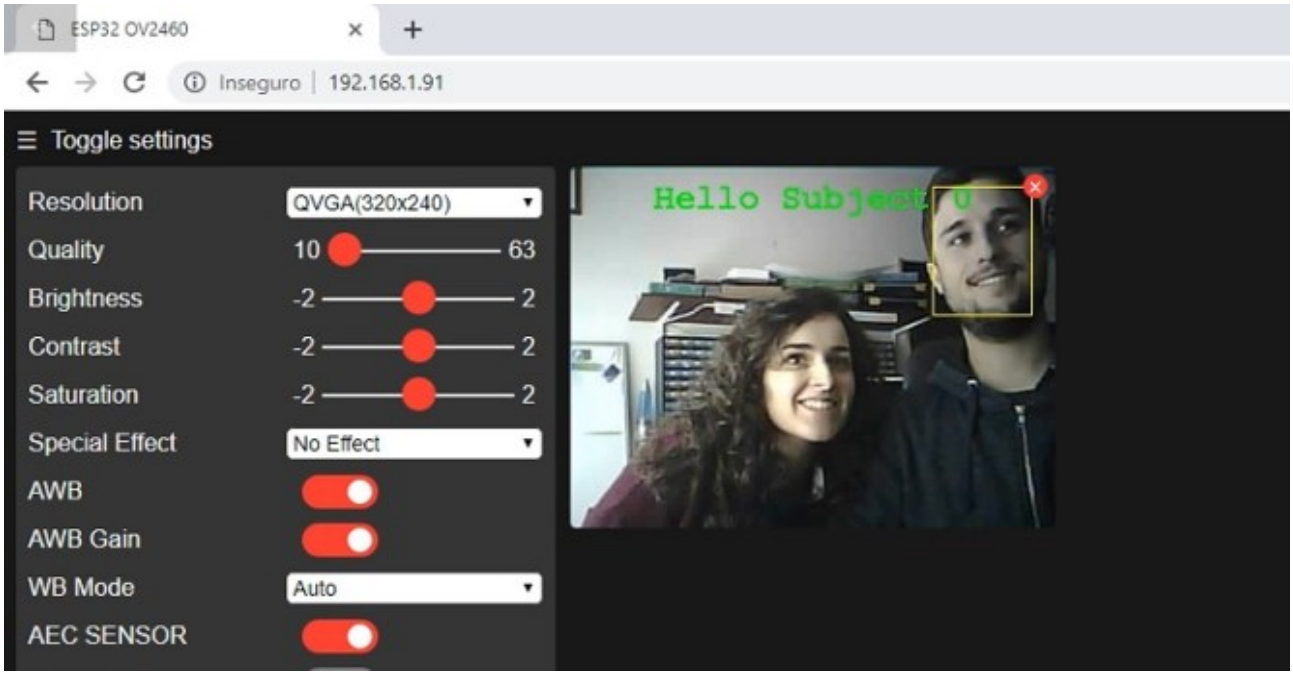
Accessing the Video Streaming Server
Now, you can access your camera streaming server on your local network. Open a browser and type the ESP32-CAM IP address. Press the Start Streaming button to start video streaming.



You also have the option to take photos by clicking the Get Still button. Unfortunately, this example doesn’t save the photos, but you can modify it to use the on board microSD Card to store the captured photos.
There are also several camera settings that you can play with to adjust the image settings.

Face recognition and detection.

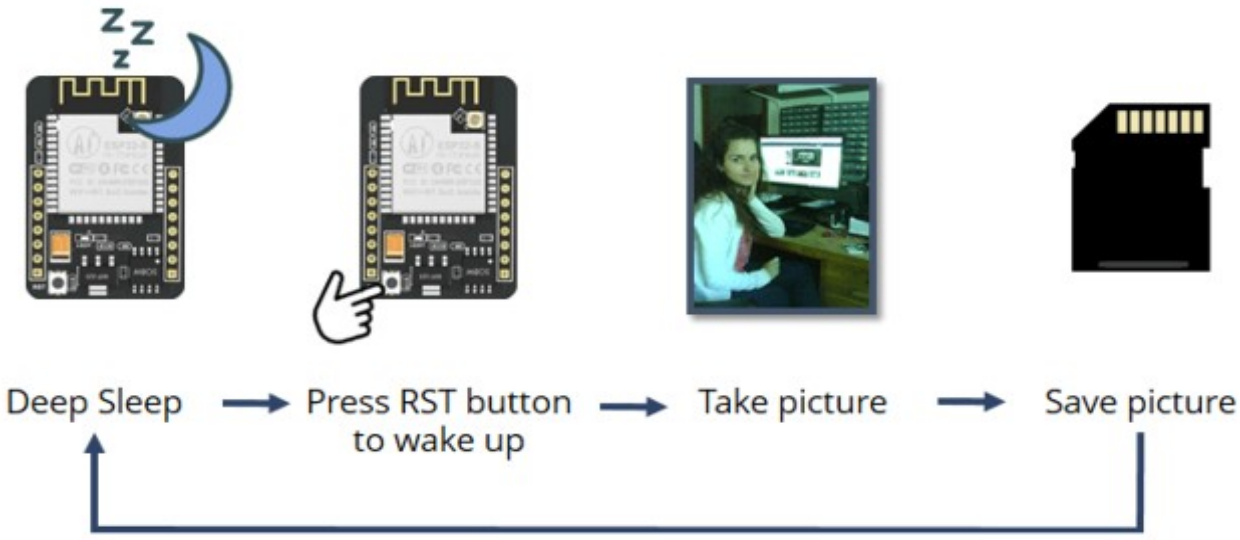
First, you need to enroll a new face. It will make several attempts to save the face. After enrolling a new user, it should detect the face later on (subject 0).



And that’s it. Now you have your video streaming web server up and running with face detection and recognition with the example from the library.

ESP32-CAM Take Photo and Save to MicroSD Card

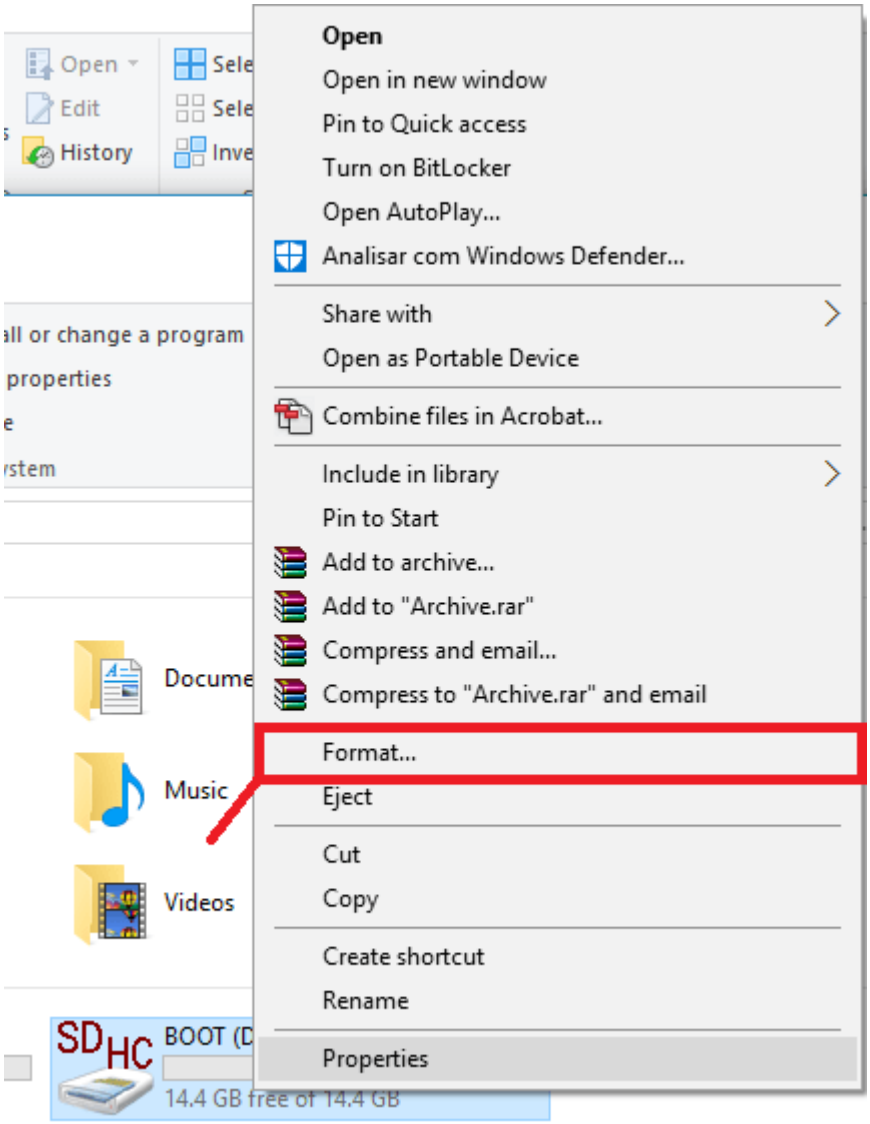
Here is a quick overview on how it works.



- The ESP32-CAM is in deep sleep mode
- Press the RESET button to wake up the board
- The camera takes a photo
- The photo is saved in the microSD card with the name: pictureX.jpg, where X corresponds to the picture number
- The picture number will be saved in the ESP32 flash memory so that it is not erased during RESET and we can keep track of the number of photos taken.

Formatting MicroSD Card
The first thing we recommend doing is formatting your microSD card. You can use the Windows formatter tool or any other microSD formatter software.

Insert the microSD card in your computer. Go to My Computer and right click in the SD card. Select Format as shown in figure below.



A new window pops up. Select FAT32, press Start to initialize the formatting process and follow the onscreen instructions.

Note: according to the product specifications, the ESP32-CAM should only support 4 GB SD cards. However, we've tested with 16 GB SD card and it works well.

Take and Save Photo Sketch

Copy the following code to your Arduino IDE.

```
#include "esp_camera.h"
#include "Arduino.h"
#include "FS.h"                // SD Card ESP32
#include "SD_MMC.h"            // SD Card ESP32
#include "soc/soc.h"           // Disable brownour problems
#include "soc/rtc_cntl_reg.h"  // Disable brownour problems
#include "driver/rtc_io.h"
#include <EEPROM.h>            // read and write from flash memory

// define the number of bytes you want to access
#define EEPROM_SIZE 1

// Pin definition for 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

int pictureNumber = 0;

void setup()
{
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); //disable brownout detector

  Serial.begin(115200);
  //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(psramFound())
  {
    config.frame_size = FRAMESIZE_UXGA; // FRAMESIZE_ + QVGA|CIF|VGA|SVGA|XGA|SXGA|UXGA
    config.jpeg_quality = 10;
    config.fb_count = 2;
  }
  else
  {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }
```

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

  //Serial.println("Starting SD Card");
  if(!SD_MMC.begin())
  {
    Serial.println("SD Card Mount Failed");
    return;
  }

  uint8_t cardType = SD_MMC.cardType();
  if(cardType == CARD_NONE)
  {
    Serial.println("No SD Card attached");
    return;
  }

  camera_fb_t * fb = NULL;

  // Take Picture with Camera
  fb = esp_camera_fb_get();
  if(!fb)
  {
    Serial.println("Camera capture failed");
    return;
  }
  // initialize EEPROM with predefined size
  EEPROM.begin(EEPROM_SIZE);
  pictureNumber = EEPROM.read(0) + 1;

  // Path where new picture will be saved in SD Card
  String path = "/picture" + String(pictureNumber) +".jpg";

  fs::FS &fs = SD_MMC;
  Serial.printf("Picture file name: %s\n", path.c_str());

  File file = fs.open(path.c_str(), FILE_WRITE);
  if(!file)
  {
    Serial.println("Failed to open file in writing mode");
  }
  else
  {
    file.write(fb->buf, fb->len); // payload (image), payload length
    Serial.printf("Saved file to path: %s\n", path.c_str());
    EEPROM.write(0, pictureNumber);
    EEPROM.commit();
  }
  file.close();
  esp_camera_fb_return(fb);

  // Turns off the ESP32-CAM white on-board LED (flash) connected to GPIO 4
  pinMode(4, OUTPUT);
  digitalWrite(4, LOW);
  rtc_gpio_hold_en(GPIO_NUM_4);

  delay(2000);
  Serial.println("Going to sleep now");
  delay(2000);
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}

void loop()
{

}
```

The code starts by including the necessary libraries to use the camera. We also include the libraries needed to interact with the microSD card.

Define the number of bytes you want to access in the flash memory. Here, we'll only use one byte that allows us to generate up to 256 picture numbers.

```
#define EEPROM_SIZE 1
```

Then, define the pins for the AI-THINKER camera module.

```
// Pin definition for 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
```

Note: you might need to change the pin definition depending on the board you're using. Wrong pin assignment will result in a failure to init the camera.

Initialize an int variable called pictureNumber that that will generate the photo name: picture1.jpg, picture2.jpg, and so on.

```
int pictureNumber = 0;
```

All our code is in the setup(). The code only runs once when the ESP32 wakes up (in this case when you press the on-board RESET button). Define the camera settings:

```
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; se the following settings for a camera
with PSRAM (like the one we're using in this tutorial).
```

```
if(psramFound()){
  config.frame_size = FRAMESIZE_UXGA; // FRAMESIZE_ + QVGA|CIF|VGA|SVGA|XGA|
SXGA|UXGA
  config.jpeg_quality = 10;
  config.fb_count = 2;
}
```

If the board doesn't have PSRAM, set the following:

```
else {
  config.frame_size = FRAMESIZE_SVGA;
  config.jpeg_quality = 12;
  config.fb_count = 1;
}
```

Initialize the camera:

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

Initialize the microSD card:

```
//Serial.println("Starting SD Card");
if(!SD_MMC.begin()){
  Serial.println("SD Card Mount Failed");
  return;
}

uint8_t cardType = SD_MMC.cardType();
if(cardType == CARD_NONE){
  Serial.println("No SD Card attached");
  return;
}
```

The following lines take a photo with the camera:

```
camera_fb_t * fb = NULL;

// Take Picture with Camera
fb = esp_camera_fb_get();
if(!fb) {
  Serial.println("Camera capture failed");
  return;
}
```

After that, initialize the EEPROM with the size defined earlier:

```
EEPROM.begin(EEPROM_SIZE);
```

The picture number is generated by adding 1 to the current number saved in the flash memory.

```
pictureNumber = EEPROM.read(0) + 1;
```

To save the photo in the microSD card, create a path to your file. We'll save the photo in the main directory of the microSD card and the file name is going to be (picture1.jpg, picture2.jpg, picture3.jpg, etc…).

```
String path = "/picture" + String(pictureNumber) +".jpg";
```

These next lines save the photo in the microSD card:

```
fs::FS &fs = SD_MMC;
Serial.printf("Picture file name: %s\n", path.c_str());

File file = fs.open(path.c_str(), FILE_WRITE);
if(!file){
  Serial.println("Failed to open file in writing mode");
}
else {
  file.write(fb->buf, fb->len); // payload (image), payload length
  Serial.printf("Saved file to path: %s\n", path.c_str());
  EEPROM.write(0, pictureNumber);
  EEPROM.commit();
}
file.close();
```

After saving a photo, we save the current picture number in the flash memory to keep track of the number of photos taken.

```
EEPROM.write(0, pictureNumber);
EEPROM.commit();
```

When the ESP32-CAM takes a photo, it flashes the on-board LED. After taking the photo, the LED remains on, so we send instructions to turn it off. The LED is connected to GPIO 4.

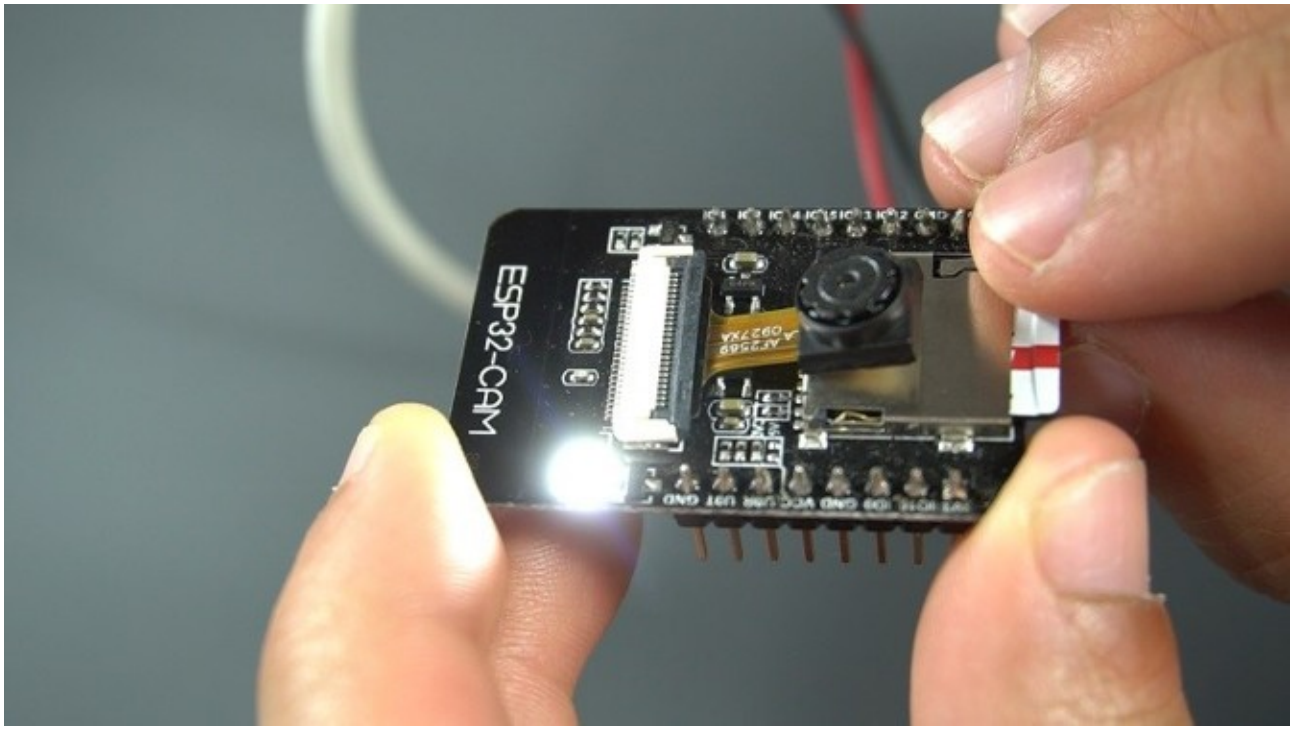
```
pinMode(4, OUTPUT);
digitalWrite(4, LOW);
rtc_gpio_hold_en(GPIO_NUM_4);
```

Finally, we put the ESP32 in deep sleep.

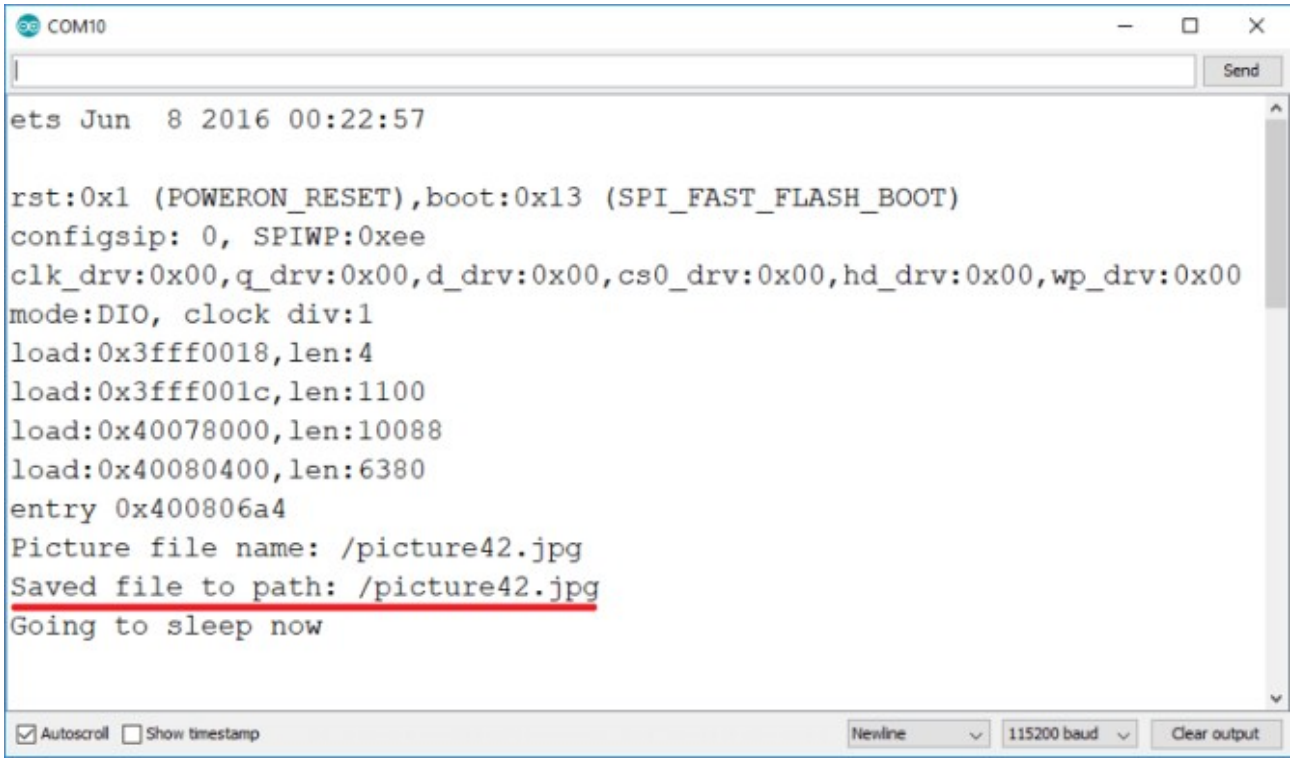
```
esp_deep_sleep_start();
```

Because we don’t pass any argument to the deep sleep function, the ESP32 board will be sleeping indefinitely until RESET.

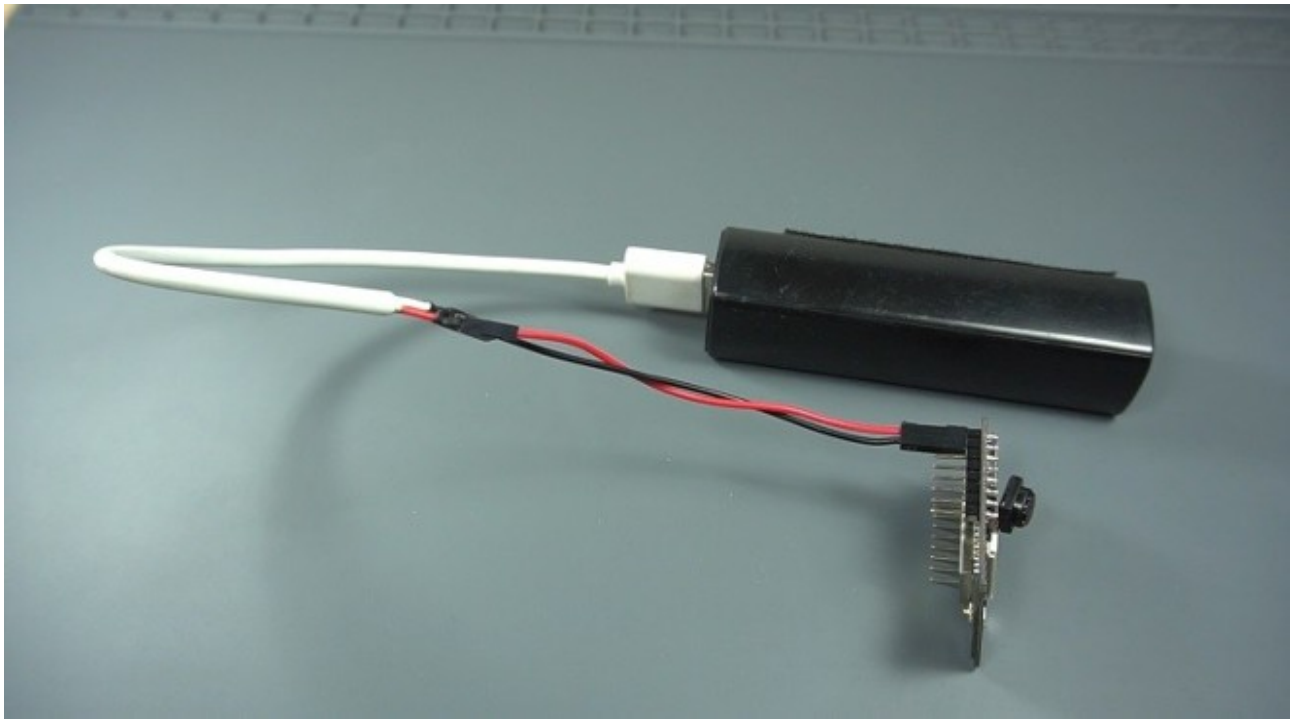
Demonstration
After uploading the code, remove the jumper that connects GPIO 0 from GND.
Open the Serial Monitor at a baud rate of 115200. Press the ESP32-CAM reset button. It should initialize and take a photo. When it takes a photo it turns on the flash (GPIO 4).



Check the Arduino IDE Serial Monitor window to see if everything is working as expected. As you can see, the picture was successfully saved in the microSD card.



After making sure that everything is working as expected, you can disconnect the ESP32-CAM from the FTDI programmer and power it using an independent power supply.



To see the photos taken, remove the microSD card from the microSD card slot and insert it into your computer. You should have all the photos saved.

The quality of your photo depends on your lighting conditions. Too much light can ruin your photos and dark environments will result in many black pixels.

ESP32-CAM Video Streaming Web Server (works with Home Assistant)

The ESP32 camera is going to host a video streaming web server that you can access with any device in your network.

You can integrate this video streaming web server with popular home automation platforms like Home Assistant or Node-RED.

Video Streaming Server
Follow the next steps to build a video streaming web server with the ESP32-CAM that you can access on your local network.

Video Streaming Web Server Code

Copy the code below to your Arduino IDE.

```
#include "esp_camera.h"
#include <WiFi.h>
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
#include "fb_gfx.h"
#include "soc/soc.h" //disable brownout problems
#include "soc/rtc_cntl_reg.h"  //disable brownout problems
#include "esp_http_server.h"

//Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

#define PART_BOUNDARY "123456789000000000000987654321"

// This project was tested with the AI Thinker Model, M5STACK PSRAM Model and M5STACK WITHOUT
PSRAM
#define CAMERA_MODEL_AI_THINKER
//#define CAMERA_MODEL_M5STACK_PSRAM
//#define CAMERA_MODEL_M5STACK_WITHOUT_PSRAM

// Not tested with this model
//#define CAMERA_MODEL_WROVER_KIT

#if defined(CAMERA_MODEL_WROVER_KIT)
  #define PWDN_GPIO_NUM    -1
  #define RESET_GPIO_NUM   -1
  #define XCLK_GPIO_NUM    21
  #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      19
  #define Y4_GPIO_NUM      18
  #define Y3_GPIO_NUM       5
  #define Y2_GPIO_NUM       4
  #define VSYNC_GPIO_NUM   25
  #define HREF_GPIO_NUM    23
  #define PCLK_GPIO_NUM    22

#elif defined(CAMERA_MODEL_M5STACK_PSRAM)
  #define PWDN_GPIO_NUM     -1
  #define RESET_GPIO_NUM    15
  #define XCLK_GPIO_NUM     27
  #define SIOD_GPIO_NUM     25
  #define SIOC_GPIO_NUM     23

  #define Y9_GPIO_NUM       19
  #define Y8_GPIO_NUM       36
  #define Y7_GPIO_NUM       18
  #define Y6_GPIO_NUM       39
  #define Y5_GPIO_NUM        5
  #define Y4_GPIO_NUM       34
  #define Y3_GPIO_NUM       35
  #define Y2_GPIO_NUM       32
  #define VSYNC_GPIO_NUM    22
  #define HREF_GPIO_NUM     26
  #define PCLK_GPIO_NUM     21

#elif defined(CAMERA_MODEL_M5STACK_WITHOUT_PSRAM)
  #define PWDN_GPIO_NUM     -1
  #define RESET_GPIO_NUM    15
  #define XCLK_GPIO_NUM     27
  #define SIOD_GPIO_NUM     25
  #define SIOC_GPIO_NUM     23

  #define Y9_GPIO_NUM       19
  #define Y8_GPIO_NUM       36
  #define Y7_GPIO_NUM       18
  #define Y6_GPIO_NUM       39
  #define Y5_GPIO_NUM        5
  #define Y4_GPIO_NUM       34
```

```
  #define Y3_GPIO_NUM       35
  #define Y2_GPIO_NUM       17
  #define VSYNC_GPIO_NUM    22
  #define HREF_GPIO_NUM     26
  #define PCLK_GPIO_NUM     21

#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

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 stream_httpd = NULL;

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;
  }

  while(true)
  {
    fb = esp_camera_fb_get();
    if (!fb)
    {
      Serial.println("Camera capture failed");
      res = ESP_FAIL;
    } else {
      if(fb->width > 400)
      {
        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)
    {
      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(res == ESP_OK)
    {
      res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY, strlen(_STREAM_BOUNDARY));
    }
    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;
    }
    //Serial.printf("MJPG: %uB\n",(uint32_t)(_jpg_buf_len));
  }
  return res;
}

void startCameraServer()
{
  httpd_config_t config = HTTPD_DEFAULT_CONFIG();
  config.server_port = 80;

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

  //Serial.printf("Starting web 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()
{
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); //disable brownout detector

  Serial.begin(115200);
  Serial.setDebugOutput(false);

  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(psramFound())
  {
    config.frame_size = FRAMESIZE_UXGA;
```

```
    config.jpeg_quality = 10;
    config.fb_count = 2;
  }
  else
  {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }

  // 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;
  }
  // Wi-Fi connection
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  Serial.print("Camera Stream Ready! Go to: http://");
  Serial.print(WiFi.localIP());

  // Start streaming web server
  startCameraServer();
}

void loop()
{
  delay(1);
}
```

Before uploading the code, you need to insert your network credentials in the following variables:

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

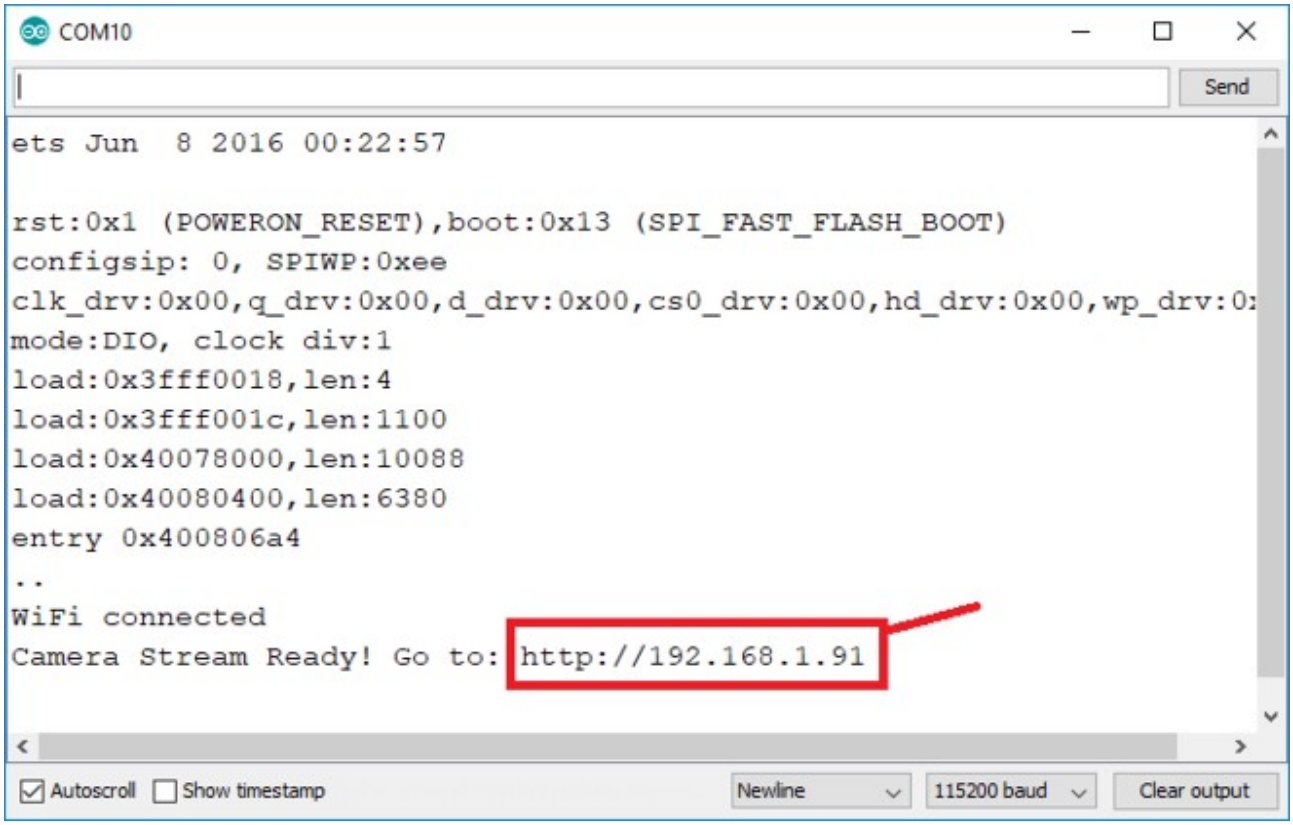
Then, make sure you select the right camera module. In this case, we're using the AI-THINKER Model.

If you're using the same camera module, you don't need to change anything on the code.

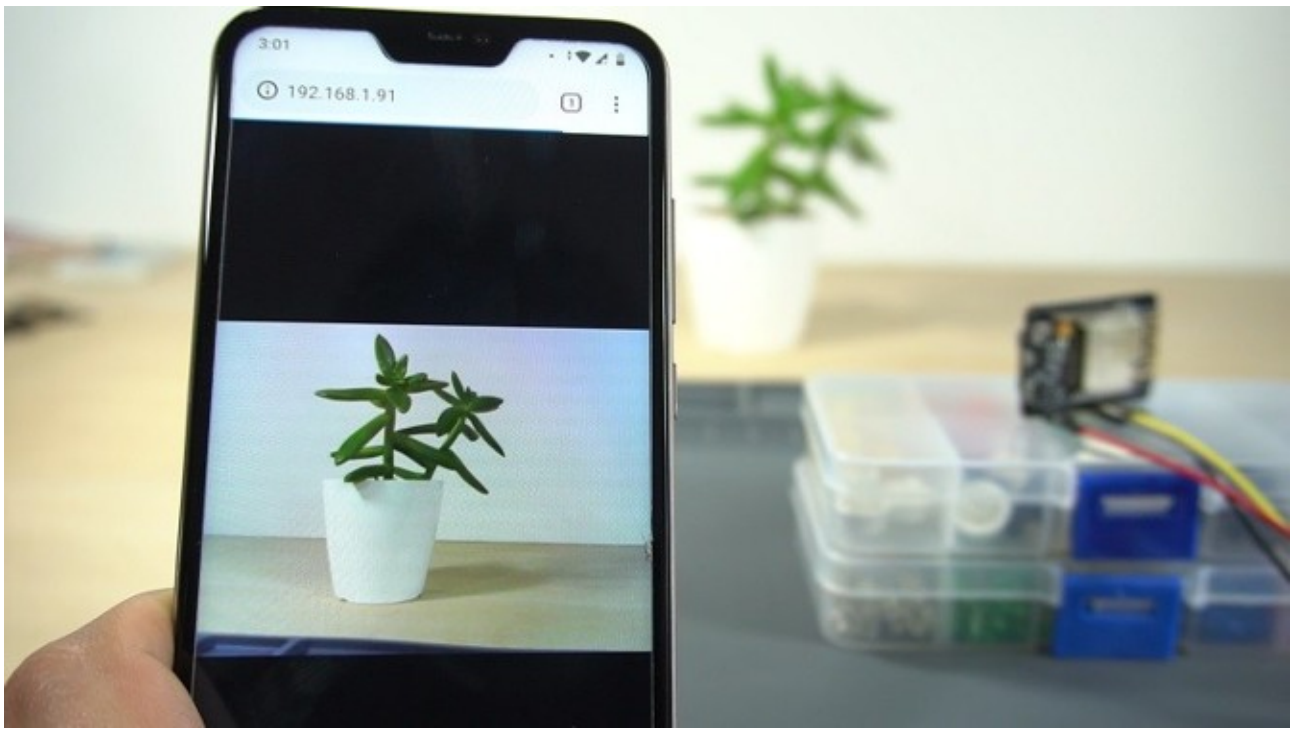
```
#define CAMERA_MODEL_AI_THINKER
```

Now, you can upload the code to your ESP32-CAM board.

Getting the IP address
After uploading the code, disconnect GPIO 0 from GND. Open the Serial Monitor at a baud rate of 115200. Press the ESP32-CAM on-board Reset button.
The ESP32 IP address should be printed in the Serial Monitor.

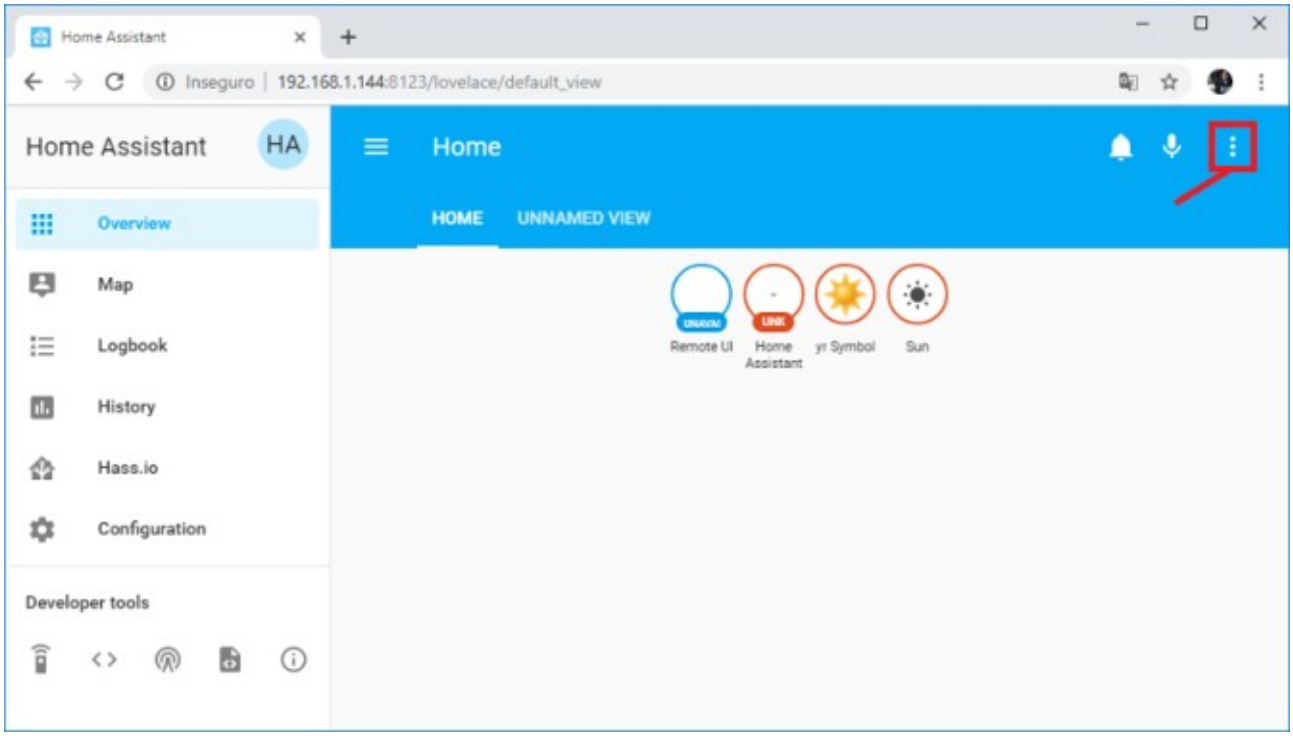


Accessing the Video Streaming Server
Now, you can access your camera streaming server on your local network. Open a browser and type the ESP32-CAM IP address. A page with the current video streaming should load.

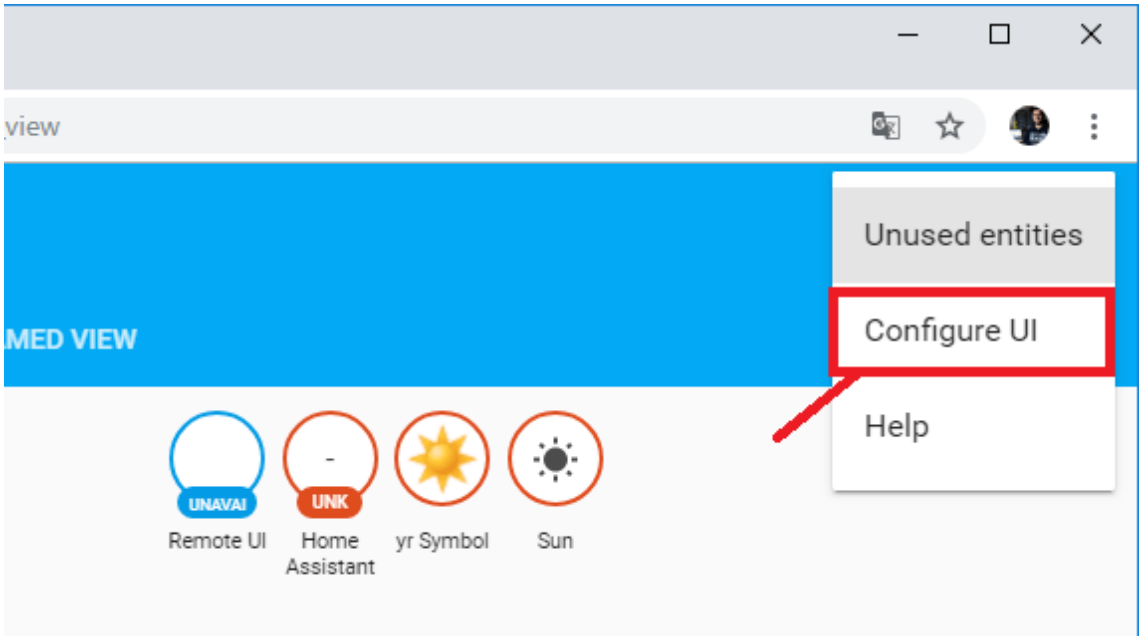


Home Assistant Integration
Having just the ESP32-CAM working via IP might be useful for most people, but you can integrate this project with Home Assistant (or with other home automation platforms). Continue reading to learn how to integrate with Home Assistant.

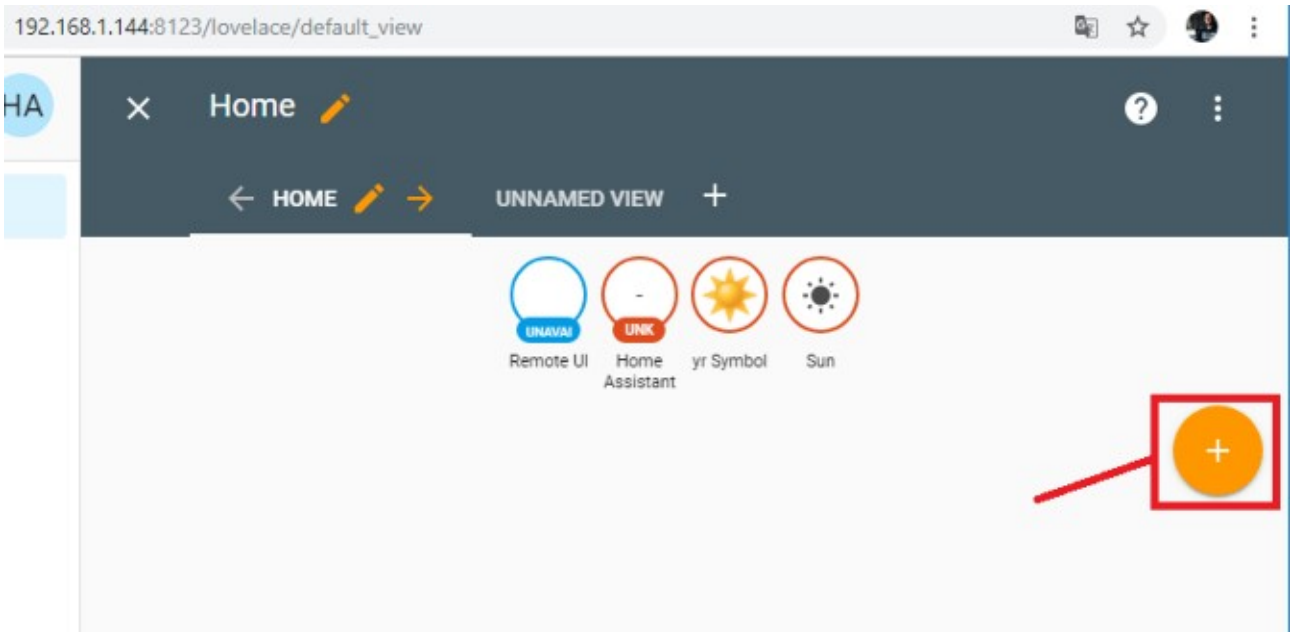
Adding ESP32-CAM to Home Assistant
Open your Home Assistant dashboard and go to the more Settings menu.



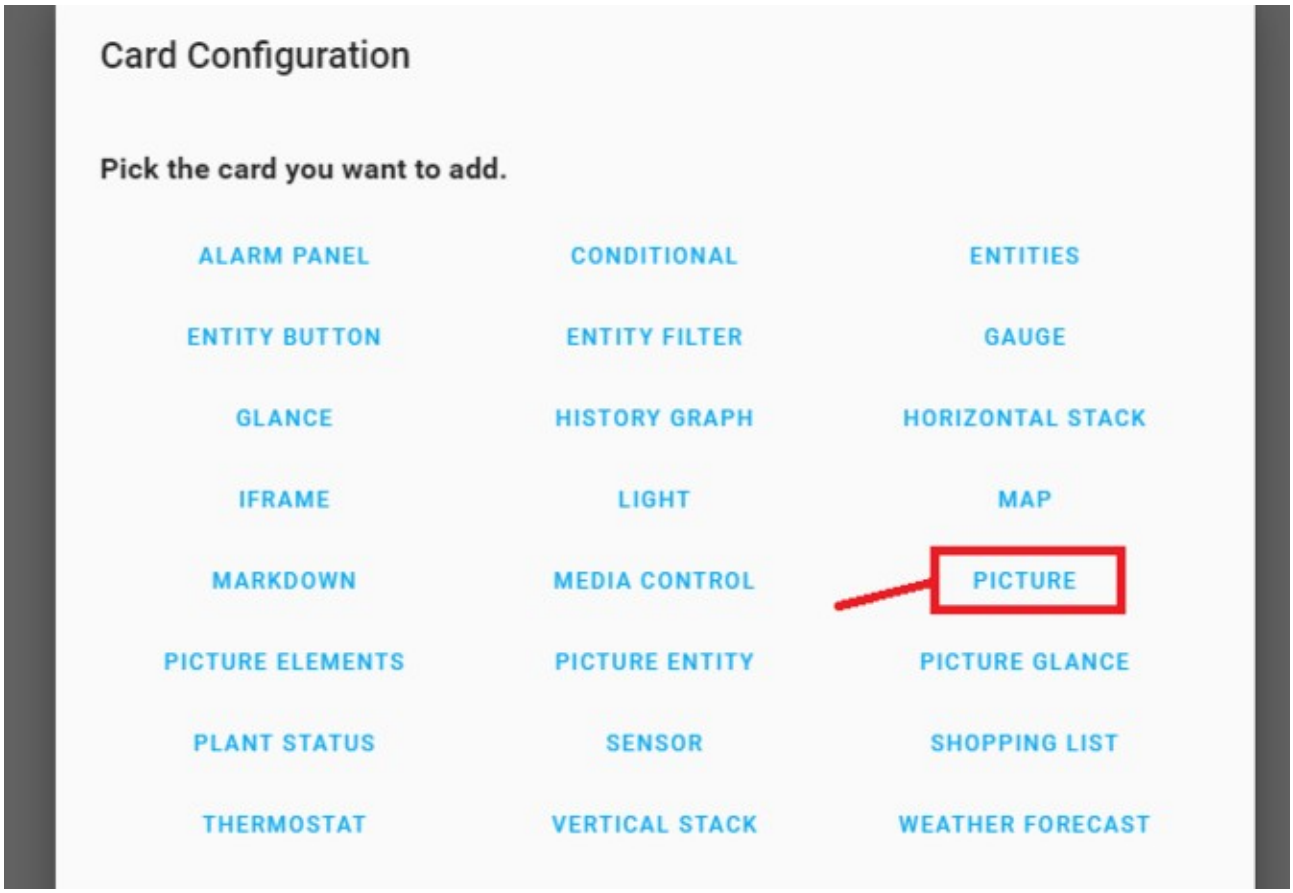
Open Configure UI:



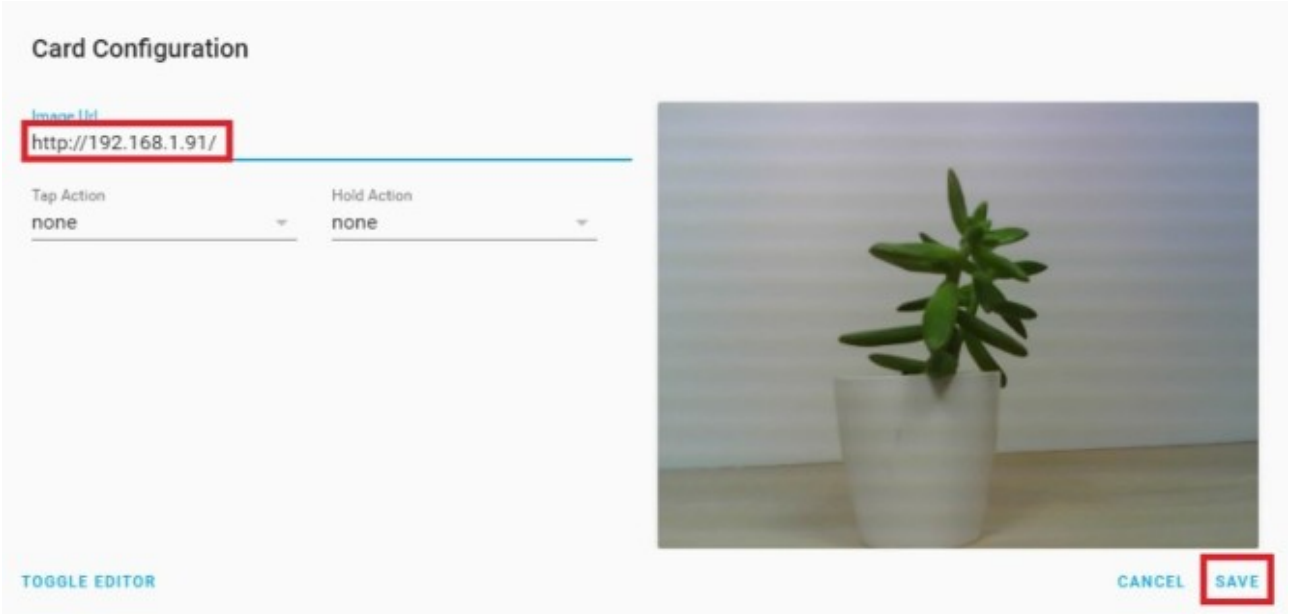
Add a new card to your Dashboard:



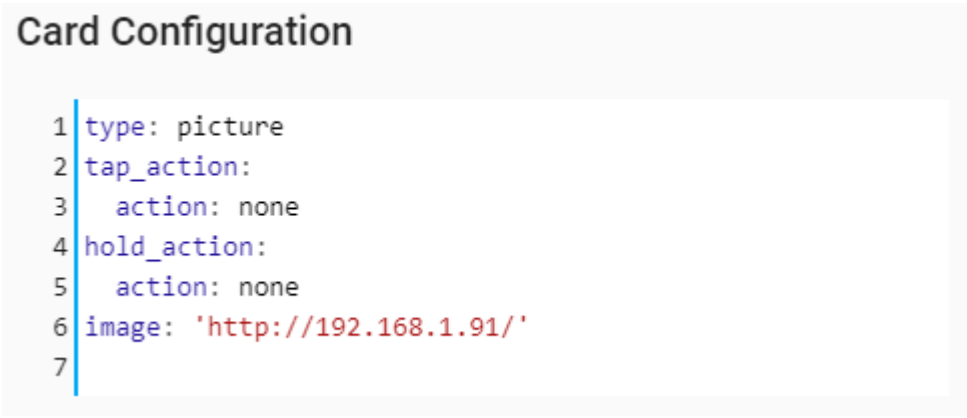
Pick a card of the type Picture.



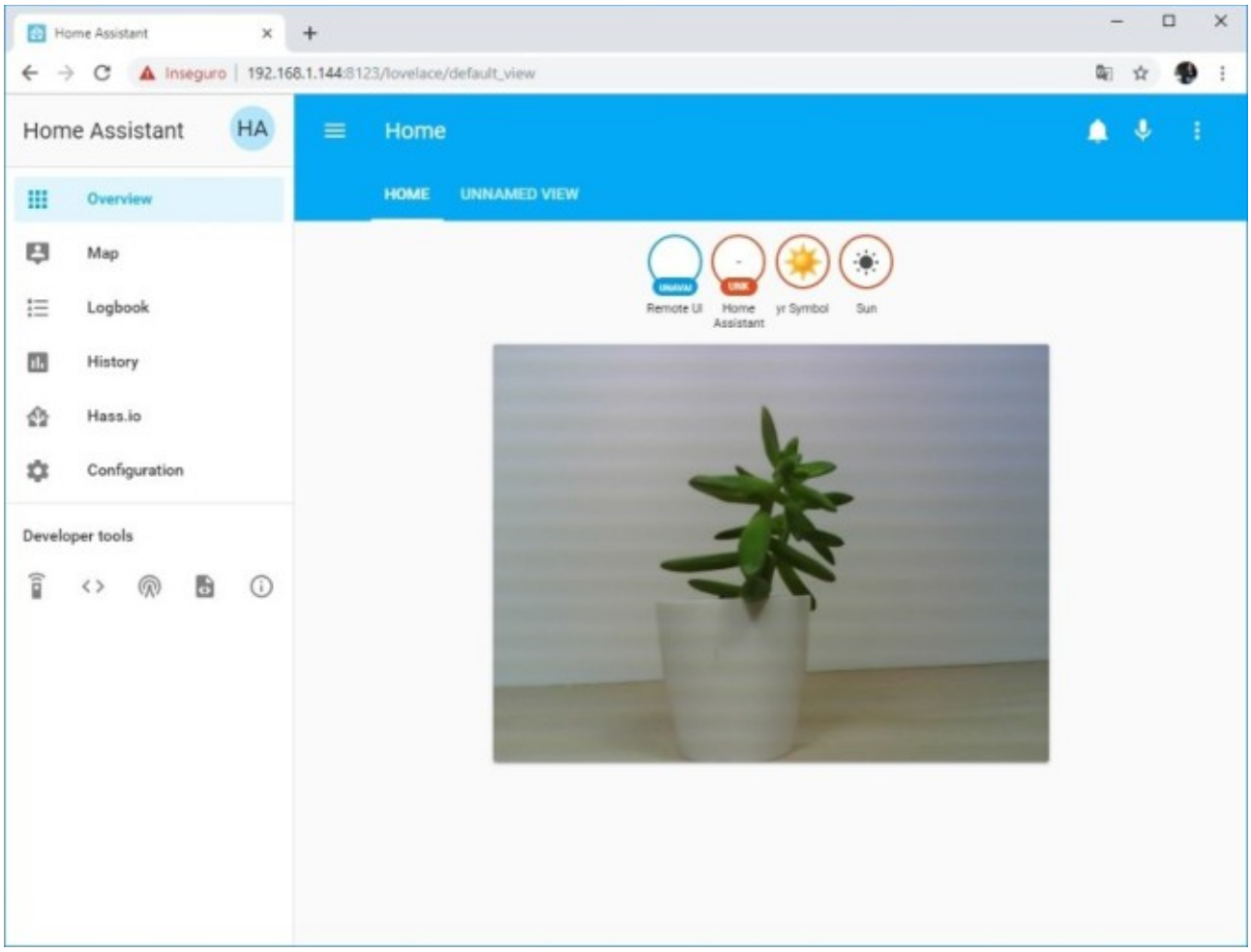
In the Image URL field, enter your ESP32-CAM IP address. Then, click the “SAVE” button and return to the main dashboard.



If you’re using the configuration file, this is what you need to add.



After that, Home Assistant can display the ESP32-CAM video streaming.



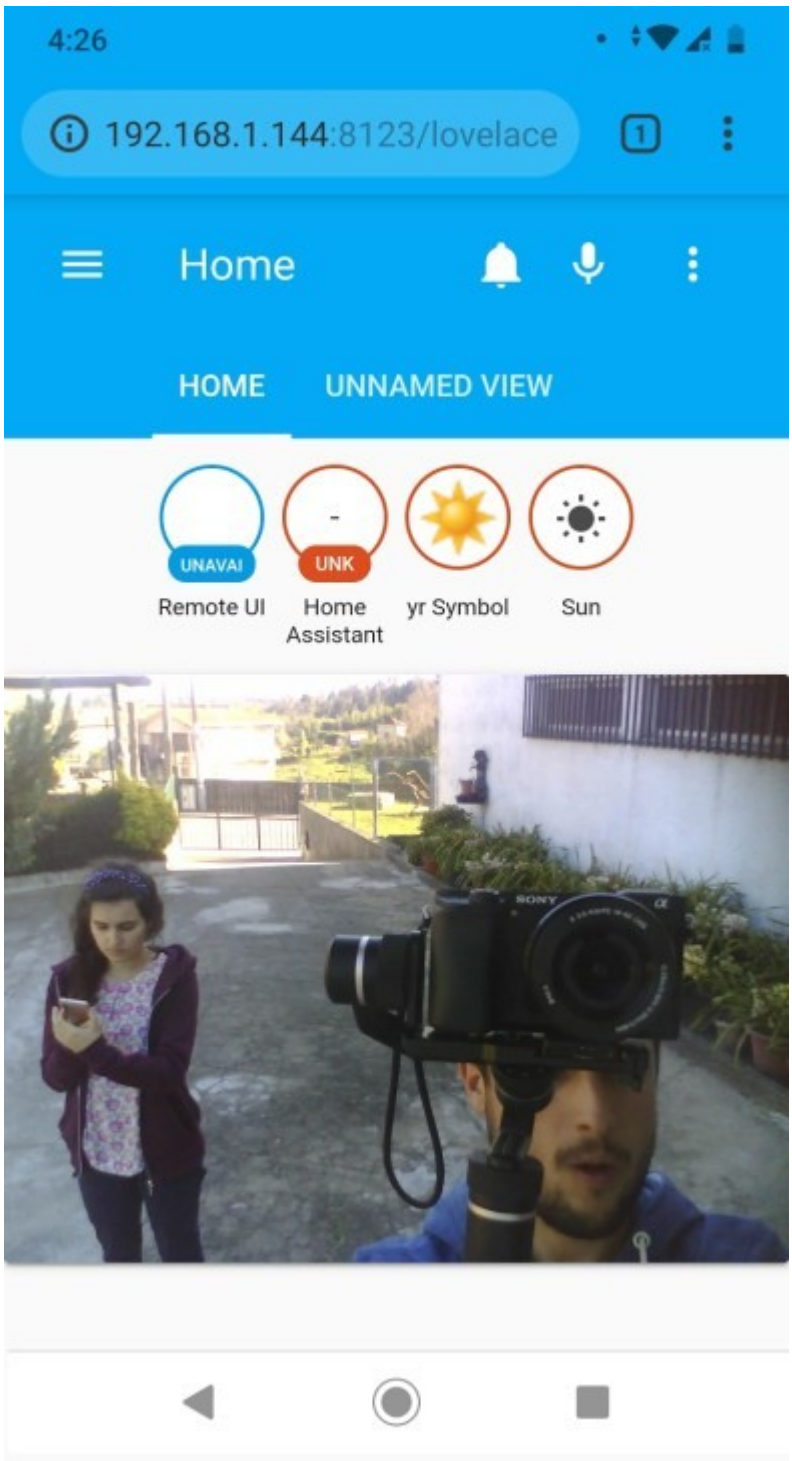
Taking It Further
To take this project further, you can use one fake dummy camera and place the ESP32-CAM inside.

The ESP32-CAM board fits perfectly into the dummy camera enclosure.

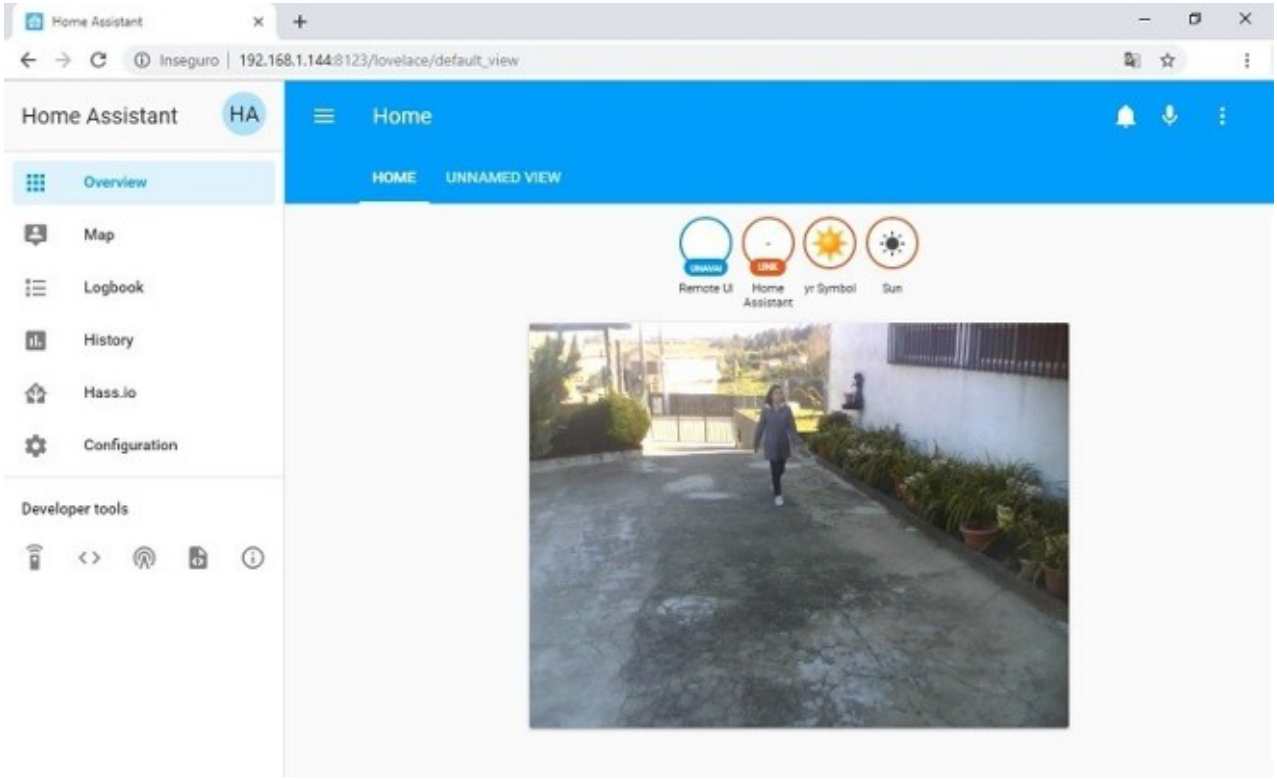
You can power it using a 5V power adapter through the ESP32-CAM GND and 5V pins.

Place the surveillance camera in a suitable place.

After that, go to the camera IP address or to your Home Assistant dashboard and see in real time what's happening. The following image shows us testing the video streaming camera. Sara is taking a screenshot while I'm filming the camera.



It’s impressive what this little $9 ESP32 camera module can do and it’s been working reliably. Now, we can use the surveillance camera to see in real time what’s happening in my front entrance.


Home Assistant
Inseguro | 192.168.1.144:8123/lovelace/default_view
Home Assistant
HA
Overview
Map
Logbook
History
Hass.io
Configuration
Developer tools
Home
HOME
UNNAMED VIEW
Remote UI
Home Assistant
yr Symbol
Sun

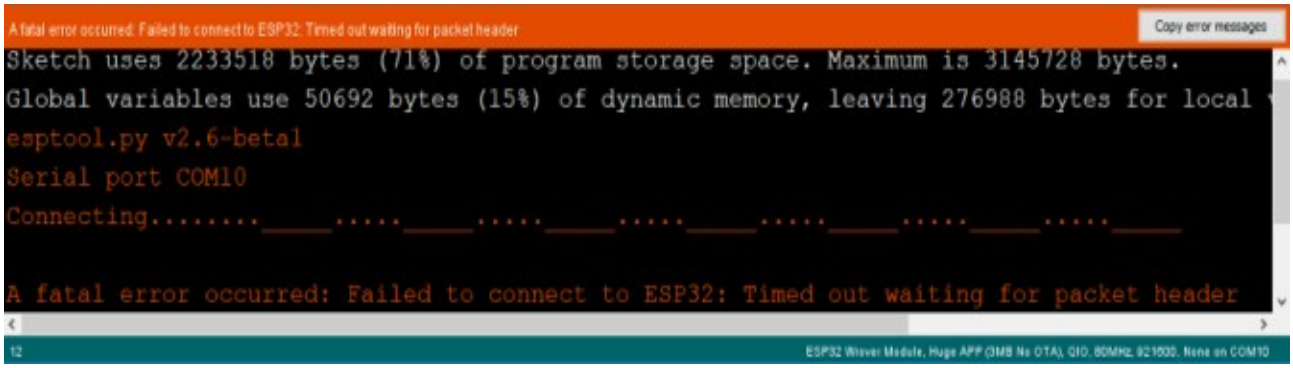
ESP32-CAM Troubleshooting Guide: Most Common Problems Fixed

If you have a different problem or a different solution to these issues, you can share your tips by writing a comment below.

Most common errors:

1. Failed to connect to ESP32: Timed out waiting for packet header
2. Camera init failed with error 0x20001 or similar
3. Brownout detector or Guru meditation error
4. Sketch too big error – Wrong partition scheme selected
5. Board at COMX is not available – COM Port Not Selected
6. Psram error: GPIO isr service is not installed
7. Weak Wi-Fi Signal
8. No IP Address in Arduino IDE Serial Monitor
9. Can't open web server
10. The image lags/shows lots of latency
11. esp_camera_fb_get(): Failed to get the frame on time!

Failed to connect to ESP32: Timed out waiting for packet header

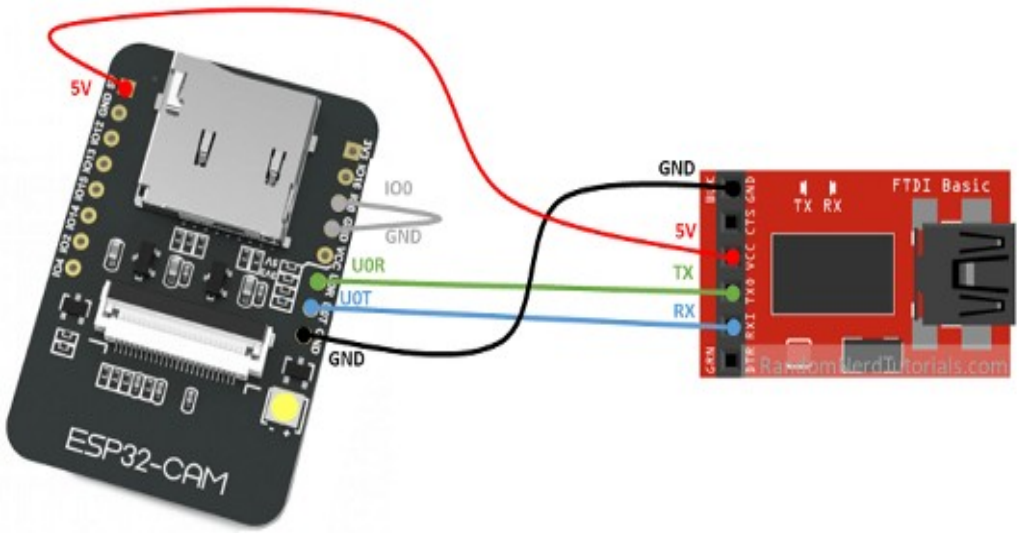


This error means that the ESP32-CAM is not in flashing mode or it is not connected properly to the FTDI programmer.

Double-check the steps to upload code

Double-check that you've followed the exact steps to put your ESP32-CAM in flashing mode. Failing to complete one of the steps may result in that error. Here's the steps you need to follow:

Connect the ESP32-CAM board to your computer using an FTDI programmer. Follow the next schematic diagram:

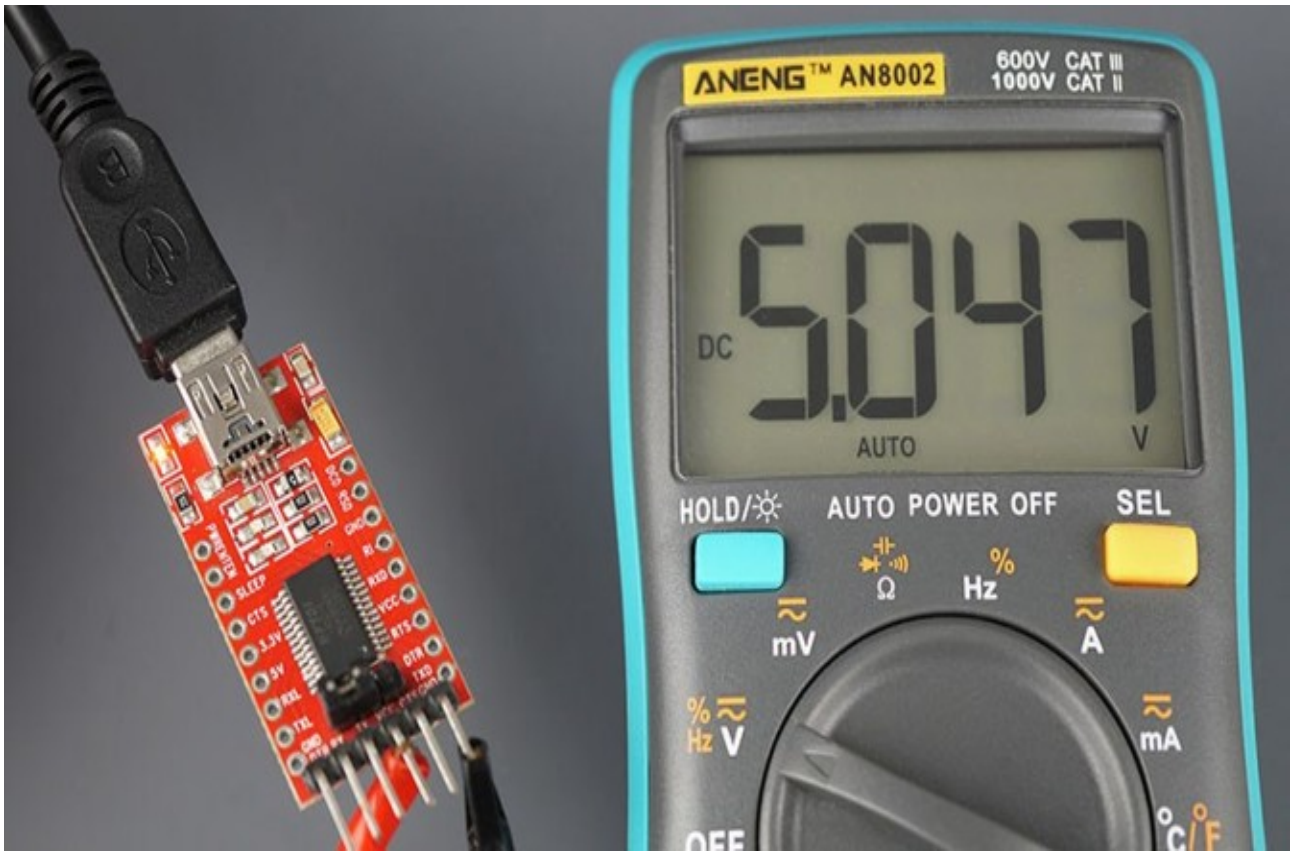


Many FTDI programmers have a jumper that allows you to select 3.3V or 5V. Make sure the jumper is in the right place to select 5V.

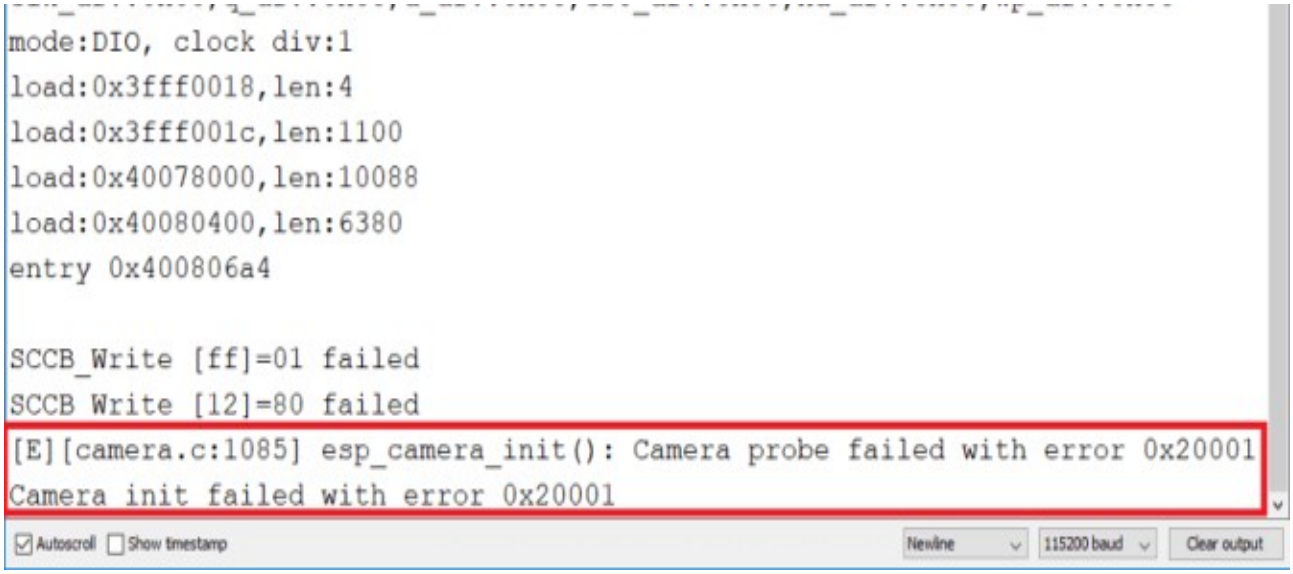
Important: if you can't upload the code, double-check that GPIO 0 is connected to GND and that you selected the right settings in the **Tools** menu. You should also press the on-board Reset button to restart your ESP32 in flashing mode. Also, check that you have the FTDI programmer jumper cap set to 5V.

Power the ESP32-CAM with 5V
Some of our readers reported that they could only upload code when the ESP32 was powered with 5V. So, power the ESP32-CAM with 5V.

FTDI Programmer 5V
Measure the output voltage of your FTDI programmer (VCC and GND) using a Multimeter to ensure it's providing 5V to your ESP32-CAM.



Camera init failed with error 0x20001 or similar



If you get this exact error, it means that your camera OVX is not connected properly to your ESP32 board or you have the wrong pin assignment in the code.
Sometimes, unplugging and plugging the FTDI programmer multiple times or restart the board multiple times, might solve the issue.

Camera not connected properly
The camera has a tiny connector and you must ensure it's connected in the the right away and with a secure fit, otherwise it will fail to establish a connection.

Wrong pin assignment in the code
When you get this error, it might also mean that you didn't select the right board in the define section or the pin definition is wrong for your board.
Make sure you select the right camera module in your projects. You just need to uncomment the right camera module and comment all the others:

```
//#define CAMERA_MODEL_WROVER_KIT
//#define CAMERA_MODEL_M5STACK_PSRAM
#define CAMERA_MODEL_AI_THINKER
```

In this example, we're using the CAMERA_MODEL_AI_THINKER, so it's the one that is enabled. Otherwise, it will fail the pin assignment and the camera will fail to init.
There are many esp32-cam boards being released ("fake boards") that the wiring between the ESP32 and the OV camera might be different, so selecting the camera module, might not be enough. You might need to check each gpio declaration with your board pinout.
For example, M5Stack board *without* PSRAM has a different pin assignment than the M5STACK *with* PSRAM (defined on the code by default). So, you need to change the pin definition in the code accordingly to the board pinout.

Not enough power through USB source
If you're powering your ESP32 through a USB port on your computer, it might not be supplying enough power.
The camera/connector is broken
If you get this error, it might also mean that your camera or the camera ribbon is broken. If that is the case, you may get a new OV2640 camera probe.

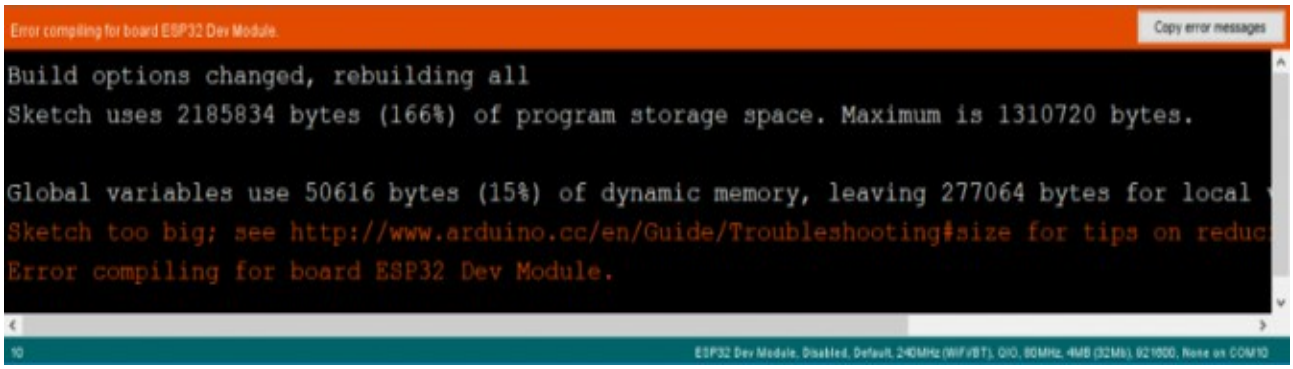
Brownout detector or Guru meditation error
When you open your Arduino IDE Serial monitor and the error message "Brownout detector was triggered" is constantly being printed over and over again. It means that there's some sort of hardware problem.
It's often related to one of the following issues:

- Poor quality USB cable;
- USB cable is too long;
- Board with some defect (bad solder joints);
- Bad computer USB port;
- Or not enough power provided by the computer USB port.

Solution:

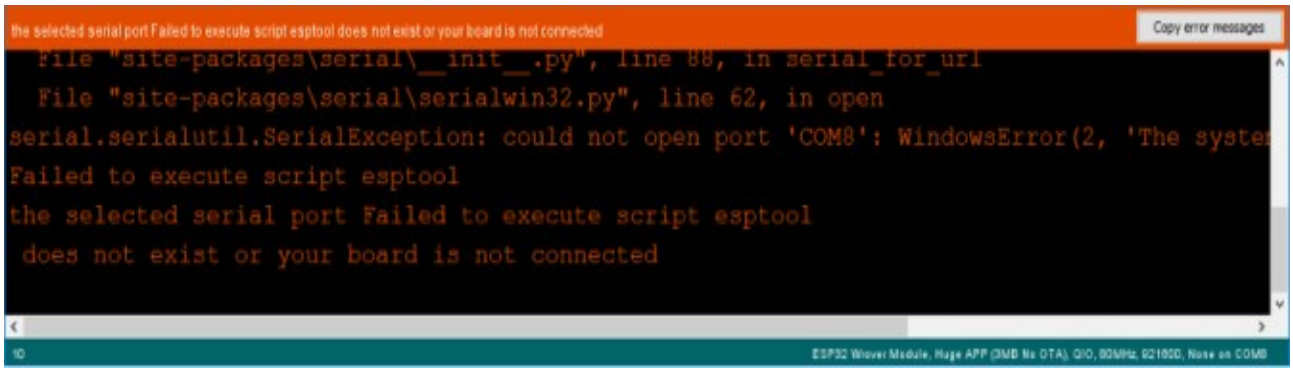
- try a different shorter USB cable (with data wires)
- use a different computer USB port or use a USB hub with an external power supply
- some readers reported that when powering the ESP32-CAM with 5V, the issue was fixed.
- Also, follow the suggestions described in issue 2.

Sketch too big error – Wrong partition scheme selected



When you get the following error:
Sketch too big; see http://www.arduino.cc/en/Guide/Troubleshooting#size for tips on reducing it.
Error compiling for board ESP32 Dev Module.
It means that you haven't selected the right partition scheme. Make sure you select the right partition scheme. In your Arduino IDE, go to **Tools** > **Partition Scheme**, select "**Huge APP (3MB No OTA)**".

Board at COMX is not available – COM Port Not Selected



If you get the following error or similar:

```
serial.serialutil.SerialException: could not open port 'COM8':
WindowsError(2, 'The system cannot find the file specified.')
Failed to execute script esptool
the selected serial port Failed to execute script esptool
 does not exist or your board is not connected
Board at COM8 is not available
```

It means that you haven't selected the COM port in the Tools menu. In your Arduino IDE, go to **Tools** > **Port** and select the COM port the ESP32 is connected to.
It might also mean that the ESP32-CAM is not establishing a serial connection with your computer or it is not properly connected to the USB connector.

Psram error: GPIO isr service is not installed

You are using a board without PSRAM and you get the following error or similar:

```
E (161) gpio: gpio_isr_handler_remove(380): GPIO isr service is not
installed, call gpio_install_isr_service() first
Camera init failed with error 0x101
```

when the board was initialized with the following settings:

```
config.frame_size = FRAMESIZE_UXGA;
config.jpeg_quality = 10;
config.fb_count = 2;
```

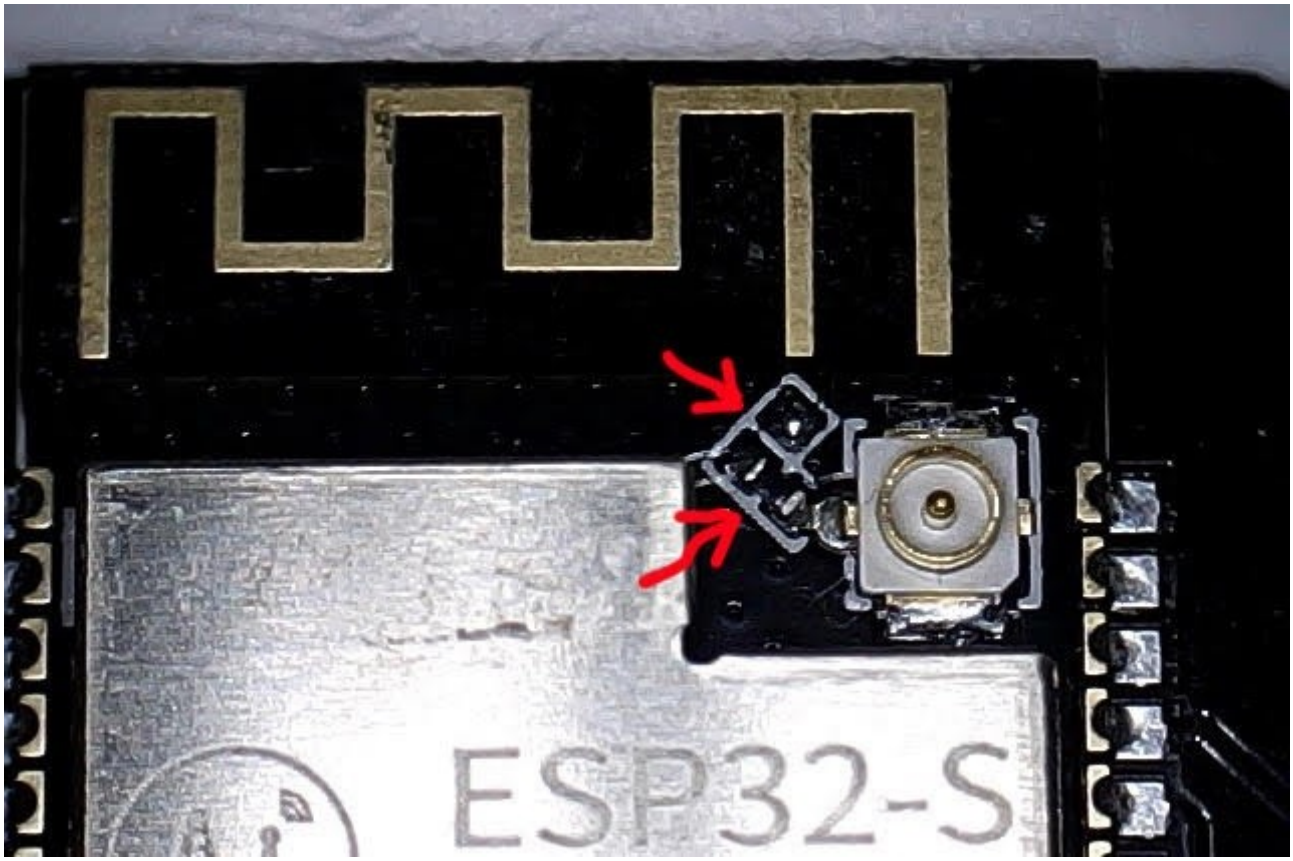
Adding the following fixes the issues (it lowers the image resolution so it won't need so much space to store images. However, as a result, you cannot get some high resolution formats due to the limited memory):

```
if(psramFound())
{
  config.frame_size = FRAMESIZE_UXGA;
  config.jpeg_quality = 10;
  config.fb_count = 2;
}
else
{
  config.frame_size = FRAMESIZE_SVGA;
  config.jpeg_quality = 12;
  config.fb_count = 1;
}
```

Note: face recognition and detection doesn't work with boards without PSRAM. However, you can still use all the other functionalities of the board. For example, although you can't use the face recognition and detection features, you can still play with the example and explore the board features as long as you have the right pin assignment in the code.

Weak Wi-Fi Signal

The ESP32-CAM has the option to use either the built-in antenna or an external antenna. If your ESP32-CAM AI-Thinker has no Wi-Fi connection or poor connection, it might have the external antenna enabled. **If you connect an external antenna to the connector, it should work fine.**
Check if the jumper 0K resistor by the antenna connector is in the proper position for the desired antenna. There are 3 little white squares laid out like a "<" with the middle position being common.



With board turned so the the PCB antenna is up:

- To use the PCB antenna, the resistor must be on the top position, like this: /
- For the antenna connector, the resistor must be on the bottom position, like this: \

So, to enable the on-board antenna:

- Unsolder the resistor that goes to the antenna, it's in this position \
- And solder together the two connections to enable the on-board antenna.

No IP Address in Arduino IDE Serial Monitor

If you just see dots printed in the serial monitor (……), it means that your ESP32-CAM is not establishing a Wi-Fi connection with your router.

Double-check your network credentials
You need to make sure that you've typed your exact network credentials (SSID and password) in the following variables:

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Select the right baud rate in the Arduino IDE Serial Monitor
If you don't select the right baud rate in the Arduino IDE Serial Monitor, you won't get your board IP address or you'll just get garbage on the screen.
Make sure you select the right baud rate. In our examples with the ESP32-CAM, we use 115200 baud rate.

Reset the board multiple times
You might also need to press the ESP32-CAM on-board RESET button multiple times to restart your ESP and print the IP address during boot.

RX and TX swapped
Double-check the connections between your ESP32 board and the FTDI programmer. RX goes to TX and TX goes to RX. If these connections are swapped, the ESP32-CAM is not able to establish a serial communication with your computer.
Wi-Fi Range
If the router is far away from your ESP32 board, it might not be able to catch the Wi-Fi signal. Ensure that your ESP32-CAM is fairly close to your router.

Can’t open web server
If the ESP32-CAM is printing the IP address in your Arduino IDE Serial Monitor, but when you try to open the web server in your web browser you see a blank screen, it usually means that you are trying to access the ESP32-CAM web server with multiple web browser tabs.
At the moment, these ESP32-CAM sketches only work with one client connected at a time.

T**he image lags/shows lots of latency**
Having some latency is normal for such a small and cheap camera. Some readers have suggested the following to reduce latency:

- Power the ESP32-CAM with a standalone 5V power supply
- Reduce the frame size with the following in your code:
  config.frame_size = FRAMESIZE_SVGA or config.frame_size = FRAMESIZE_VGA
- Use an external antenna.

esp_camera_fb_get(): Failed to get the frame on time!
We’ve personally never faced this issue. However, many readers are getting this error with their ESP32-CAM boards.
One of our readers (Fibula) suggested the following to solve this issue:

> “I’m using the ESP32-CAM Module 2MP OV2640 Camera sensor Module Type-C USB module from Aliexpress. Although not mentioned, It doesn’t have the extra PSRAM the other M5 models do, and the camera has one changed IO pin.
> See here: https://github.com/m5stack/m5stack-cam-psram/blob/master/README.md and scroll down to Interface Comparison.
> The CameraWebServer Arduino example we’re probably all using doesn’t have this ESP32-CAM model defined.
> You need to add it yourself in the main tab add:
>
> ```
> #define CAMERA_MODEL_M5STACK_NO_PSRAM
> ```
>
> And in the *camera_pins.h* tab add the following:
>
> ```
> #elif defined(CAMERA_MODEL_M5STACK_NO_PSRAM)
> #define PWDN_GPIO_NUM -1
> #define RESET_GPIO_NUM 15
> #define XCLK_GPIO_NUM 27
> #define SIOD_GPIO_NUM 25
> #define SIOC_GPIO_NUM 23
> #define Y9_GPIO_NUM 19
> #define Y8_GPIO_NUM 36
> #define Y7_GPIO_NUM 18
> #define Y6_GPIO_NUM 39
> #define Y5_GPIO_NUM 5
> #define Y4_GPIO_NUM 34
> #define Y3_GPIO_NUM 35
> #define Y2_GPIO_NUM 17
> #define VSYNC_GPIO_NUM 22
> #define HREF_GPIO_NUM 26
> #define PCLK_GPIO_NUM 21
> ```
>
> And you’re good to go.
> Also note that the max resolution of the bare ESP32-CAM Module is XGA 1024×768, I assume also because of the lack of PSRAM. “

We hope this suggestion solves your issue

Using larger microSD card sizes
According to he datasheet, the ESP32-CAM should only supports 4GB microSD cards. However, we've tested with 16GB microSD card and it works well.
You might not be able to store more than 4GB, even though you have 16GB. We haven't tested storing more than 4GB, so we're not sure about this.