ESP32-CAM Object Identification

Table of Contents:

Introduction

When trying to keep pace with the dizzying speed of technology, one thing that never ceases to amaze me is how a cutting-edge application can go from costing millions of dollars to costing pennies, all in the space of a few years.

One such technology is Object Detection, the ability of a machine to recognize an object (or several objects) using a video camera. Only a few years ago, "playing" with applications like this demanded huge computers, often with big (and expensive) graphics cards.

We have also used SBCs like the NVIDIA Jetson Nano, which is essentially an AI-dedicated GPU, for these tasks. This sort of hardware is required to run the neural networks needed for machine learning.

But now we have a new generation of microcontrollers that can run a small version of machine language, something called Embedded ML or Tiny ML. We've already seen how well the Kendryte K210 performs in the DFRobot Husky Lens using this technique.

Today we'll work on ESP32 to make capable of simple object detection. If your needs are simple, you may be able to fulfill them with an inexpensive ESP32-CAM board.

Edge Computing

With the reduction in size that comes with microcontrollers also comes a reduction in price, so it's now possible to use object detection as part of an "edge computing" application.

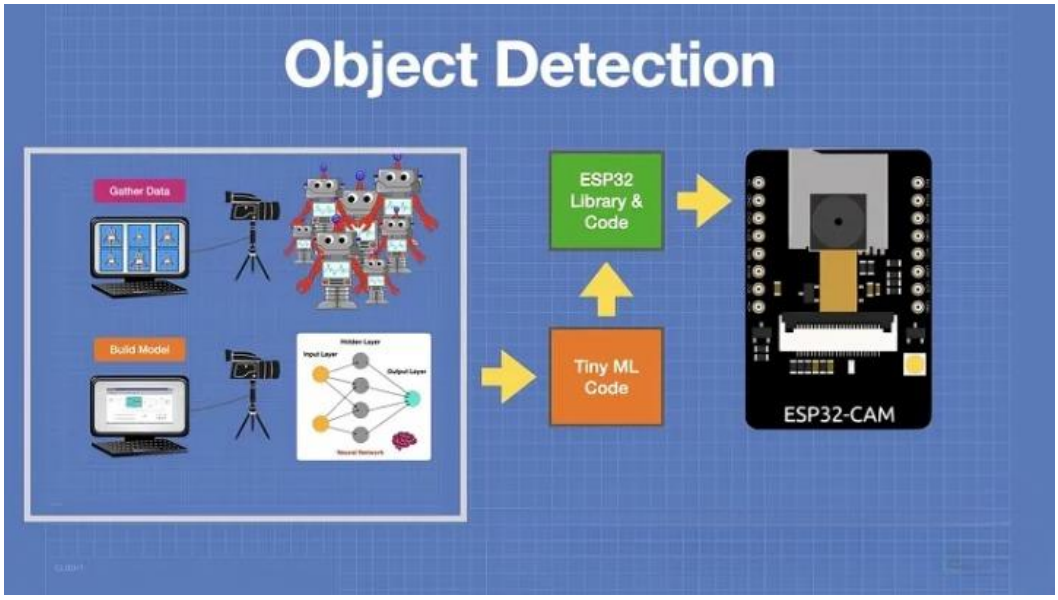
Edge computing is a model of computing that involves processing data closer to its source or at the "edge" of the network, rather than relying on a central location like a main computer or cloud-based service.

By using low-cost cameras and microcontrollers, we can implement a system that can recognize specific objects and report their position back to a central controller or react directly to their presence.

Workflow

We have quite a few tasks to go through in order to deploy an object to the ESP32-CAM.

- **Collect Images** – You'll need to take pictures of your target object from different angles and views. For accurate object detection, you might need 50 or more images for each object.
- **Label Images** – This is the process of drawing bounding boxes around the objects and labeling them. It is a bit time-consuming, but it is an essential step.
- **Train Model** – Now that you have your images labeled, you can train a model using them. You'll be setting up a neural network and running your data through it.
- **Export Model** – Once you have your model properly trained, you can export it to a format that is compatible with the ESP32-CAM board.
- **Run Model** – Run your model on the ESP32-CAM board and see if it works! You can always redo some of the training if the results are not what you had hoped for.



ESP32-CAM

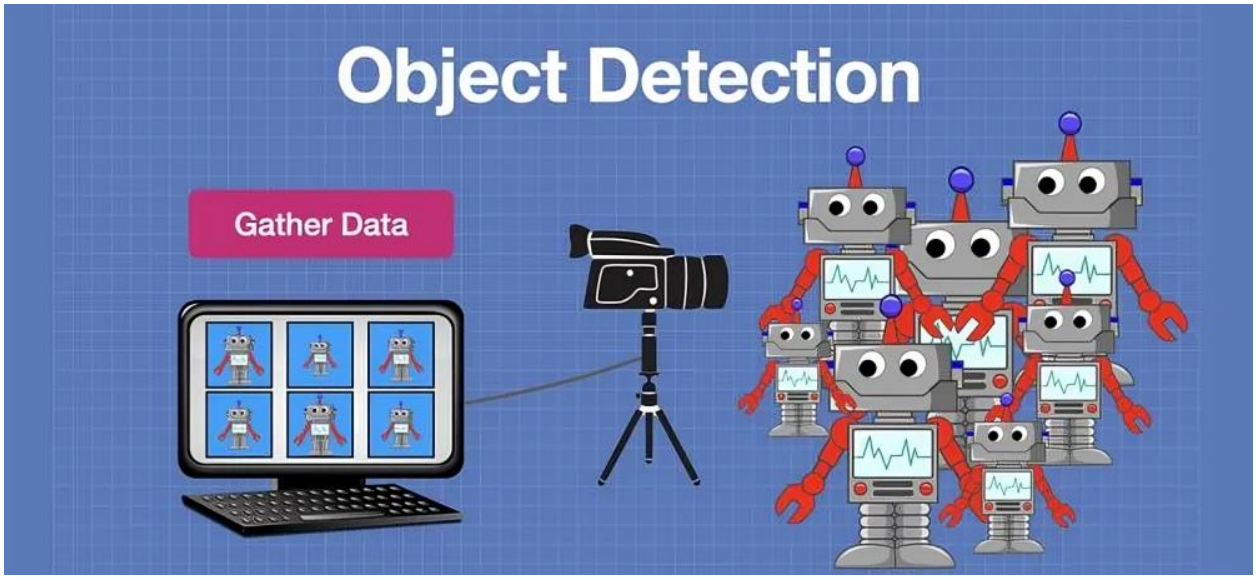
By far, the most popular and probably the most inexpensive board is the ESP32-CAM board.

This board has been around for several years, it's an open-source design, so it is sold under many different names and build qualities.

One disadvantage of this low-cost board is that it has no USB connection, so you will need to use either an FTDI adapter or a USB adapter board. The latter is much easier; your ESP32-CAM board just snaps onto it. These are often sold in conjunction with ESP32-CAM boards.

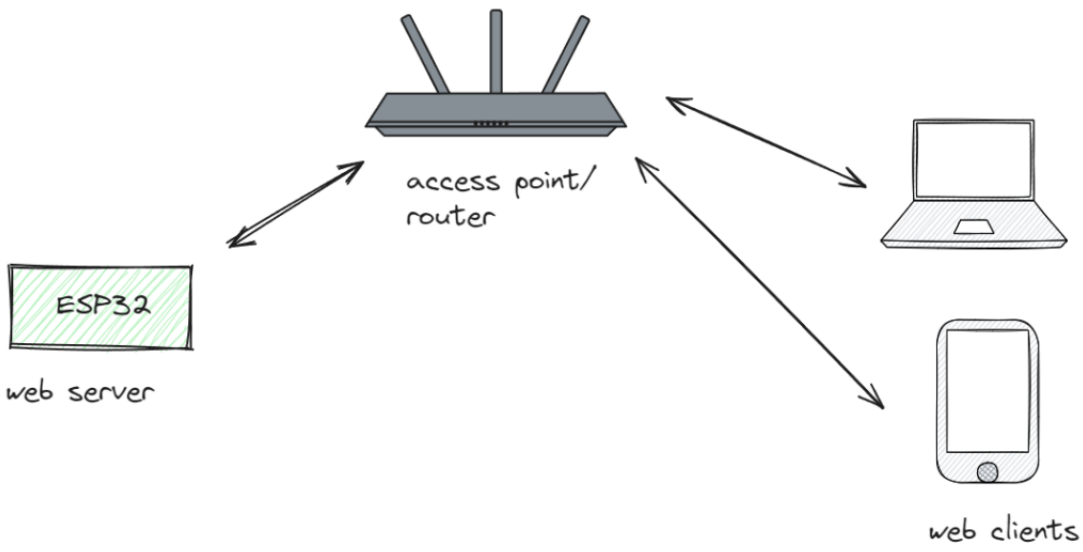
Features

1. **Image Capture:** Utilize the ESP32-CAM's camera module to capture images.
2. **Image Processing:** Use machine learning algorithms to process the captured images.
3. **Object Identification:** Identify objects in the images using the processed data.
4. **Real-time Detection:** Implement real-time object detection capabilities for live video feeds.
5. **User Interface:** Develop a user-friendly interface for interacting with the ESP32-CAM and viewing the object identification results.
6. **Data Storage:** Store captured images and identification results for future reference or analysis.
7. **Remote Access:** Enable remote access to the ESP32-CAM for monitoring and control purposes.
8. **Integration:** Integrate the ESP32-CAM with other devices or systems for enhanced functionality.
9. **Security:** Implement security measures to protect the ESP32-CAM and its data from unauthorized access.
10. **Scalability:** Design the project to be easily scalable, allowing for future expansion and addition of new features.



Technologies used

1. **ESP32-CAM Module:** The core hardware platform for capturing images and running the object identification system.
2. **Arduino IDE:** The development environment for programming the ESP32-CAM module.
3. **Camera Module:** The camera component of the ESP32-CAM used for capturing images.
4. **Machine Learning Framework:** TensorFlow Lite or other lightweight machine learning frameworks for running object detection models on the ESP32-CAM.
5. **Python:** For scripting and model deployment tasks, especially if you're training your machine learning models on a separate machine.
6. **OpenCV:** A computer vision library that can be used for image processing tasks, such as resizing, filtering, and feature extraction.
7. **PlatformIO:** An open-source ecosystem for IoT development that can be used with the ESP32-CAM.
8. **WiFi:** Utilize the ESP32-CAM's WiFi capabilities for remote access and data transfer.



Training Data

Object detection systems work by building up a “model” of the object(s) they are designed to detect. They then use this model to analyze live videos for patterns that match this model.

In order to construct an object detection system, you’ll need to build a model and deploy it. You’ll have to go through the following steps:

Gathering Data

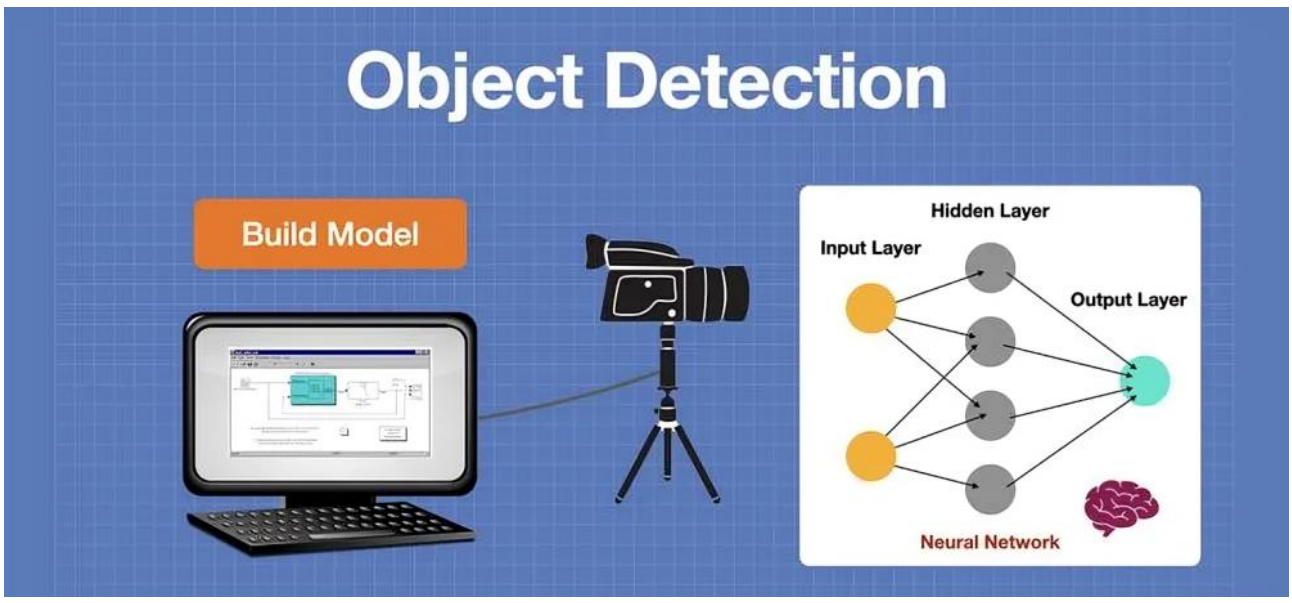
We’ll need pictures, lots of pictures, of the object you are trying to detect. You want to train the model on different views of the object from different angles, different distances, and under different lighting.

Some object detection systems are trained with thousands of images, but we can get pretty good results and keep it under a hundred, which is more manageable.

Not only will you need to gather pictures, but you’ll also need to go through each one and apply a label. This involves tracing out the object using a bounding box.

As you might expect, data gathering is the most time-consuming part of building a custom object detection system.

Building a Model



After you have gathered and organized your data, you will need to run it through a neural network to build your model.

A Neural Network is a computing model that's inspired by the structure of the human brain. It's built with layers of nodes, commonly called neurons, which can be understood like the logic gates in electronics: they take one or more inputs, perform some computation, and generate an output.

The real power of Neural Networks, though, is in the connections between these neurons and how they're weighted or prioritized. These networks are capable of learning from data and adapting their connections, which makes them excellent for complex tasks like pattern recognition or data prediction.

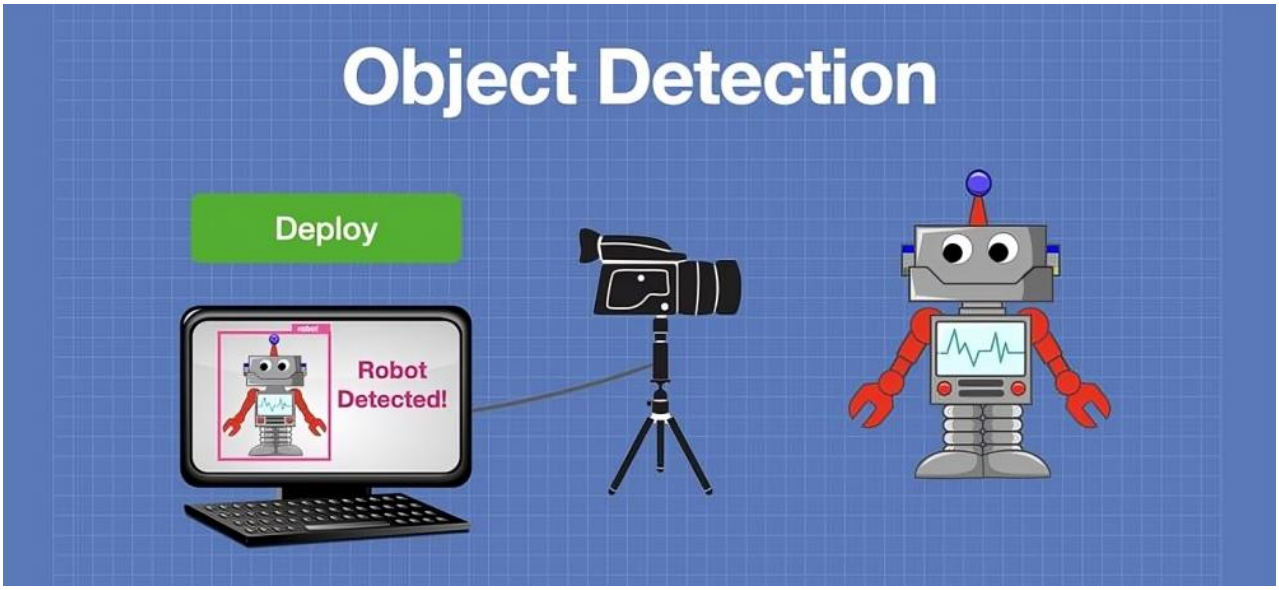
As you might imagine, a neural network takes a fair bit of computing resources. Many times, GPUs are used to take advantage of their incredible parallel-processing abilities.

Deploying the Model

Once the model is trained and built, it will need to be deployed so that it can be put to use.

The target system can be either the system the model was built on or another system with suitable video capabilities.

If the model is found to be inaccurate, it can be trained further using additional images and different neural network parameters.



Machine Learning & Tiny ML

The model we are creating is coded for use with Machine Learning (ML), or more specifically, TinyML.

Machine Learning, a key branch of Artificial Intelligence, involves algorithms and statistical models that enable computer systems to improve their performance on tasks over time without being explicitly programmed to do so.

These systems “learn” from the data they encounter, adapting and optimizing their actions based on patterns and insights they glean. In effect, they “learn” similarly to humans.

TinyML, or Tiny Machine Learning, is an emerging field that brings machine learning capabilities to ultra-low power microcontroller-based devices. This would include the ESP32.

Loading to ESP32-CAM

If you have followed along up to this point, you should now have a ZIP file with a name similar to your project name. Keep track of this file and proceed to the next step.

Importing Library

You’ll need the Arduino IDE for the nex, we are using IDE Version 2 as an example, but you should also be able to use the older version 1.8 if you wish.

I assume you have the ESP32 Boards Manager installed. I used the AIThinker ESP32-CAM board for both the ESP32-CAM and ESP-EYE, and it worked properly. If this paragraph hasn’t made much sense to you so far, you should probably check out my article on the ESP32-CAM and ESP32, as you need to know how to work with these boards in the Arduino IDE.

The ZIP file we trained can be opened as an Arduino Library, and it can be installed as you would any ZIP library in the Arduino IDE:

- Open the Arduino IDE.
- Go to the Sketch entry in the top menu
- Select Include Library. A submenu will open.
- Select Add ZIP Library..

The library should install into your IDE. You’ll get a message when it is done.

We are now ready to deploy the file to our ESP32-CAM board.

Usage

1. **Home Security**: Use the ESP32-CAM to detect intruders or motion in your home and send alerts or trigger alarms.

2. **Smart Agriculture**: Monitor plant health by detecting diseases or pests on crops using image recognition.

3. **Smart Retail**: Implement automated checkout systems or track customer movements in stores for analytics.

4. **Environmental Monitoring**: Detect and track wildlife in natural habitats or monitor environmental changes.

5. **Industrial Automation**: Use for quality control in manufacturing or monitoring processes.

6. **Traffic Monitoring**: Detect and analyze traffic flow, vehicle types, or parking space availability.

7. **Healthcare**: Monitor patient movements or detect falls in elderly care facilities.

8. **Education**: Use for robotics projects, teaching computer vision concepts, or creating interactive displays.

9. **Artificial Intelligence**: Experiment with AI models for more complex object detection tasks.

10. **Personal Projects**: Create custom projects like a smart doorbell, wildlife camera, or pet monitoring system.

Conclusion

In conclusion, the ESP32-CAM offers a wide range of possibilities for object detection applications. Whether you're looking to enhance home security, improve agricultural practices, or innovate in retail and healthcare, the ESP32-CAM's capabilities make it a versatile platform for various projects. With its affordability and ease of use, it's an excellent choice for hobbyists, students, and professionals alike to explore the exciting world of computer vision and IoT.