# Project: Deploy an object detection model

**Introduction**

Object detection is a complex system that involves identifying potential objects of interest in an image, classifying them, and giving their locations to the user. Identifying objects of interest in a live video stream is a powerful tool in many computer vision applications.

In this project, you will have the opportunity to collect data, train an object detection model, and deploy it to an embedded system.

**Required Hardware**

At this time, object detection models only run on computers (including single board computers) and smartphones. Some of the required TensorFlow operations are not supported in TensorFlow Lite for Microcontrollers. If and when it is possible to run object detection on microcontrollers (e.g. OpenMV Camera), I will update this project.

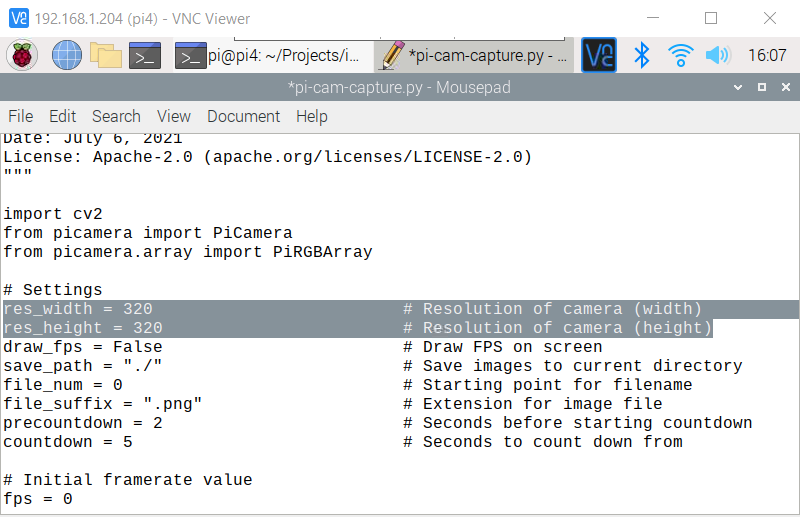
You will need the following hardware setups for this section:

* Raspberry Pi 4, SD card, Pi Camera

**Note:** You are welcome to try other embedded systems not listed here. However, I cannot promise that they will work and I likely will not be able to help you troubleshoot any issues you may come across.

**Data Collection**

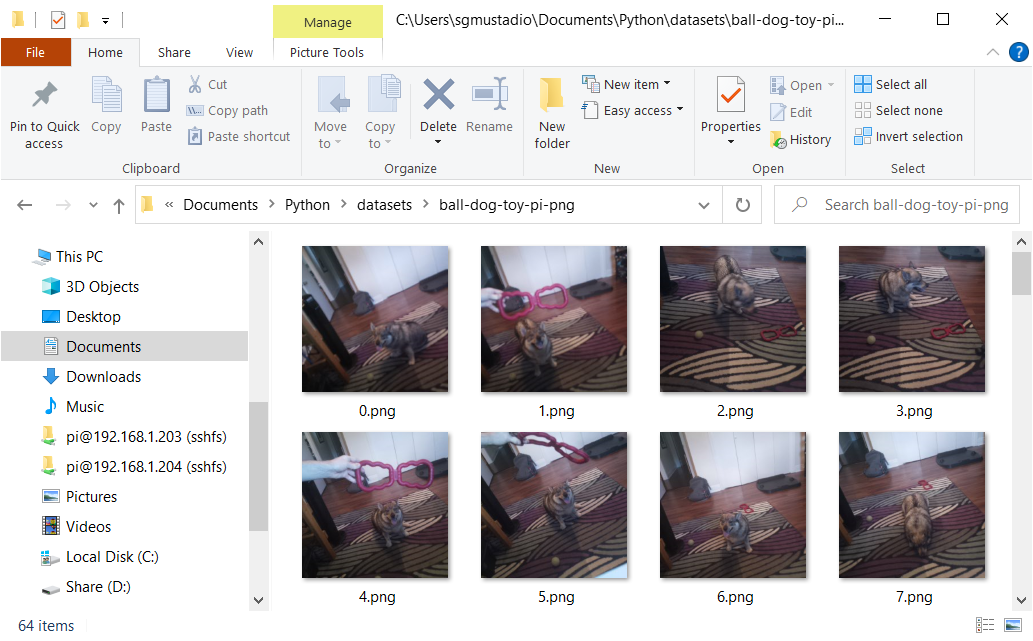
Use the [image capture program](https://github.com/ShawnHymel/computer-vision-with-embedded-machine-learning/tree/master/1.1.3%20-%20Data%20Collection) that we saw in the first module to collect a series of images. Note that you will need to change the image resolution to 320x320, as the object detection model we are using in this project only works with that resolution.



Choose one or more classes that you wish to identify. I recommend starting with something simple, like 3 classes, as it will save you some work having to label everything.

Take photos where one or more such objects are in the photo. For my project, a photo might include just 1 dog. Another photo might include 1 dog, 1 ball, and 2 tug toys. Make sure you take photos in a variety of environments, backgrounds, lighting conditions, angles, etc.

Here is an example of my dataset that includes various instances of my 3 objects: dog, ball, toy.



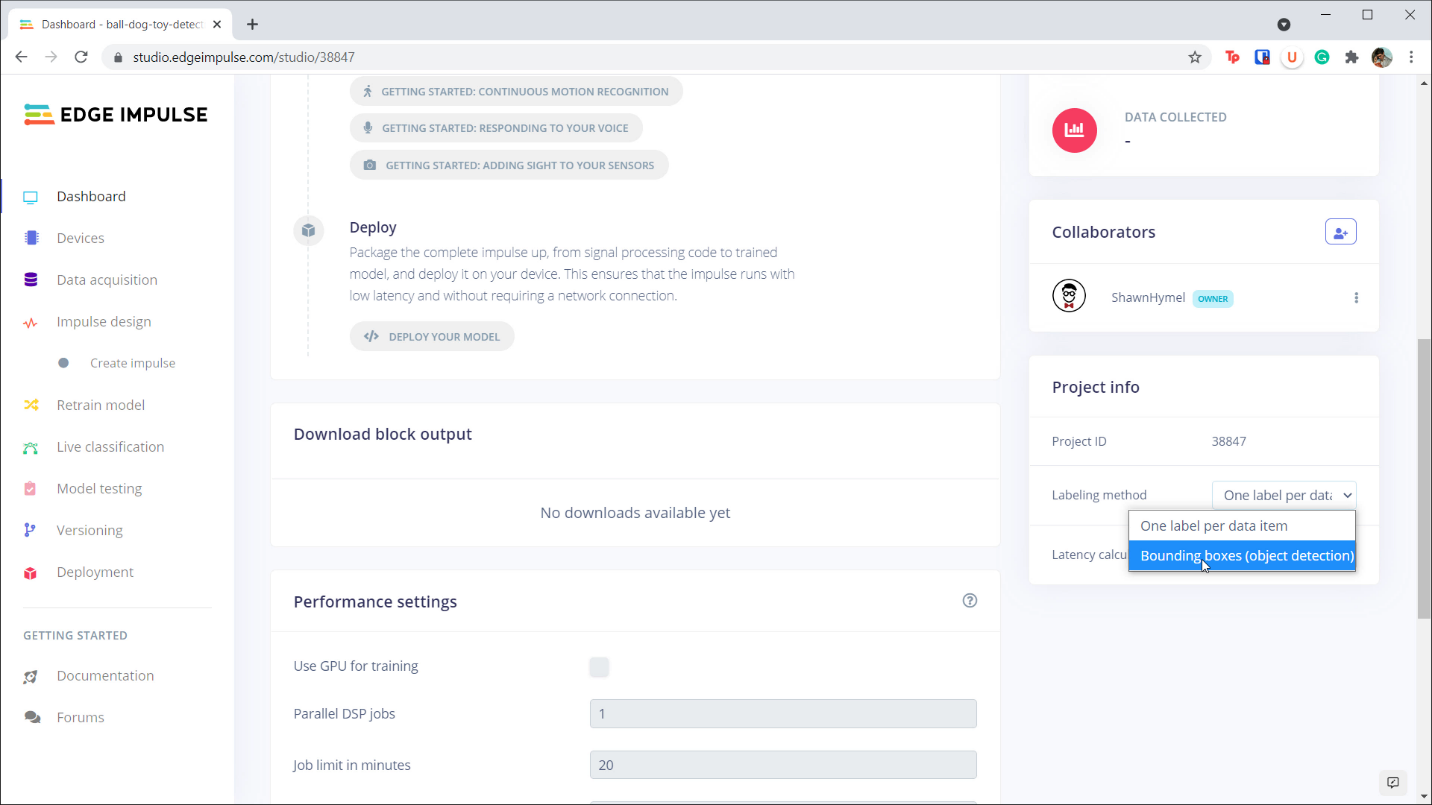
Aim to have enough photos such that you have at least 50 instances of each object class (this could be 50 or more total images, depending on how many objects you have in each image).

You are welcome to [download my dataset here](https://github.com/ShawnHymel/computer-vision-with-embedded-machine-learning/blob/master/Datasets/ball-dog-toy-pi-png.zip). However, don’t expect it to work well in your particular environment, as your background, dog, toy, ball, etc. will likely look different than mine.

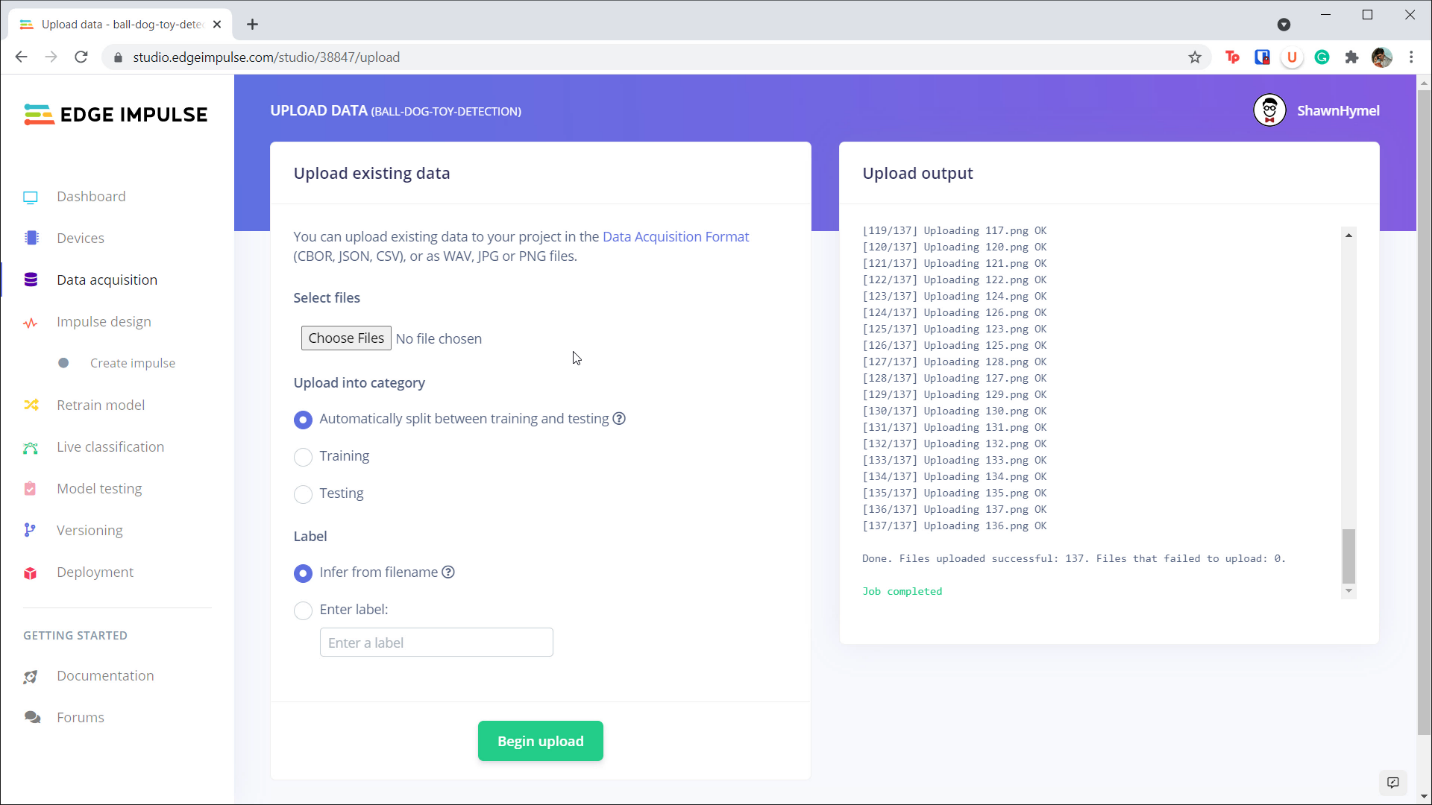
Convert all images to PNG format.

**Train Object Detection Model**

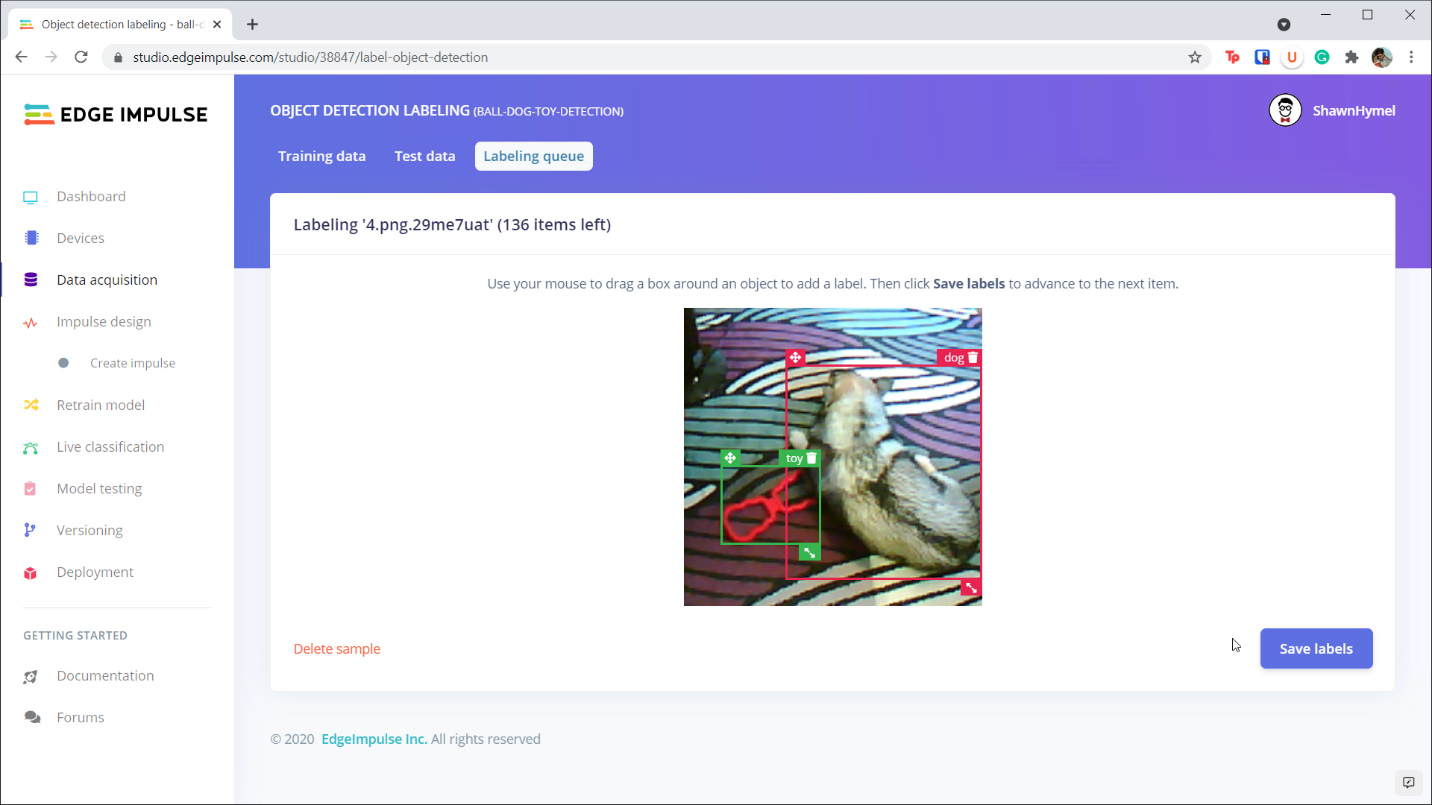
Start a new Edge Impulse project. In **Dashboard**, scroll down to *Project info*. Change *Labeling method* to **Bounding boxes (object detection)**.



Go to **Data acquisition** and upload all of your images. You can leave *Automatically split between training and testing* as well as *Infer from filename* (for *Label*) selected, as we will be supplying our own labels through bounding boxes.

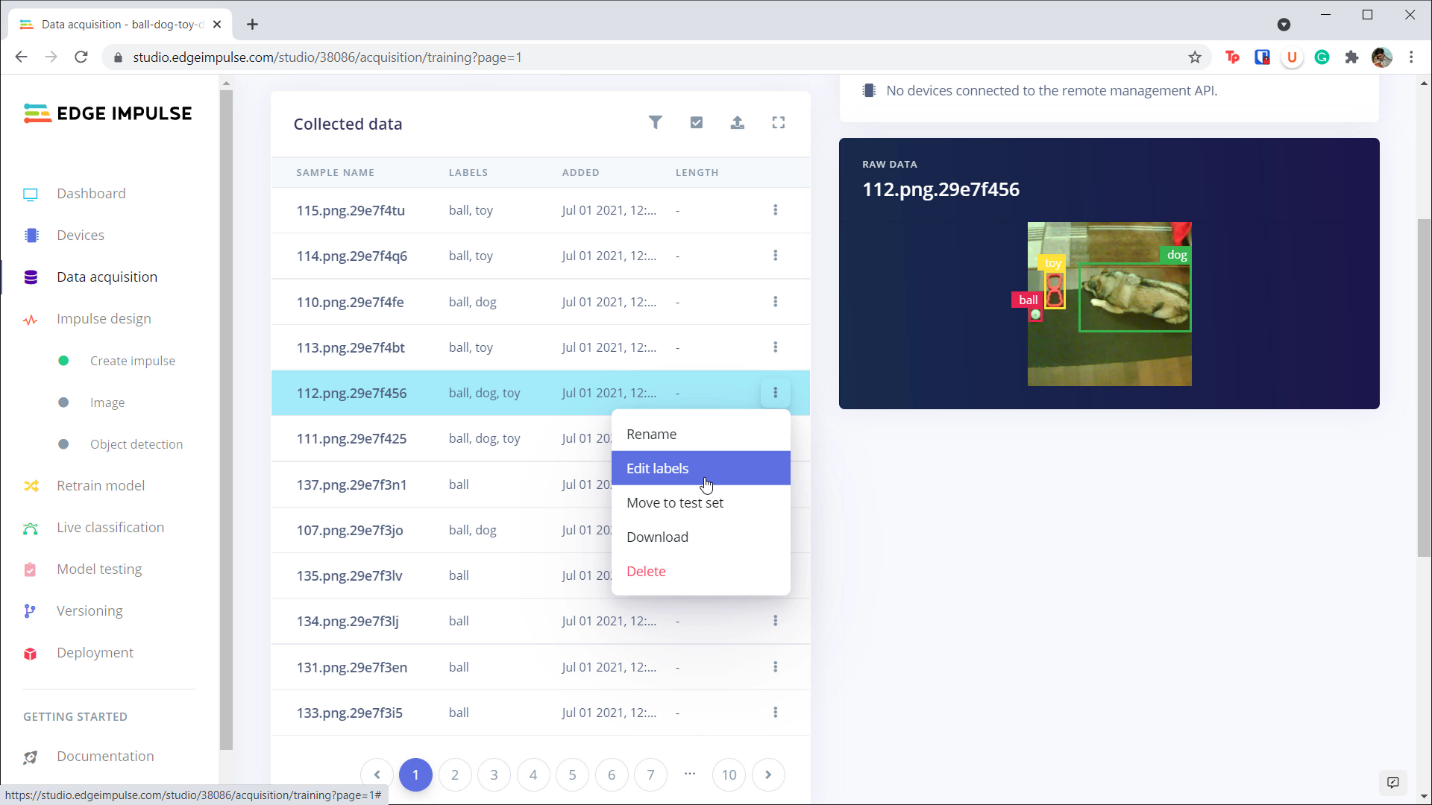


Once uploading is done, click on **Data acquisition** again. Click on **Labeling queue** at the top, which will walk you through creating bounding boxes for your images. Click and drag on the image to create a bounding box (and fill in the label when asked).



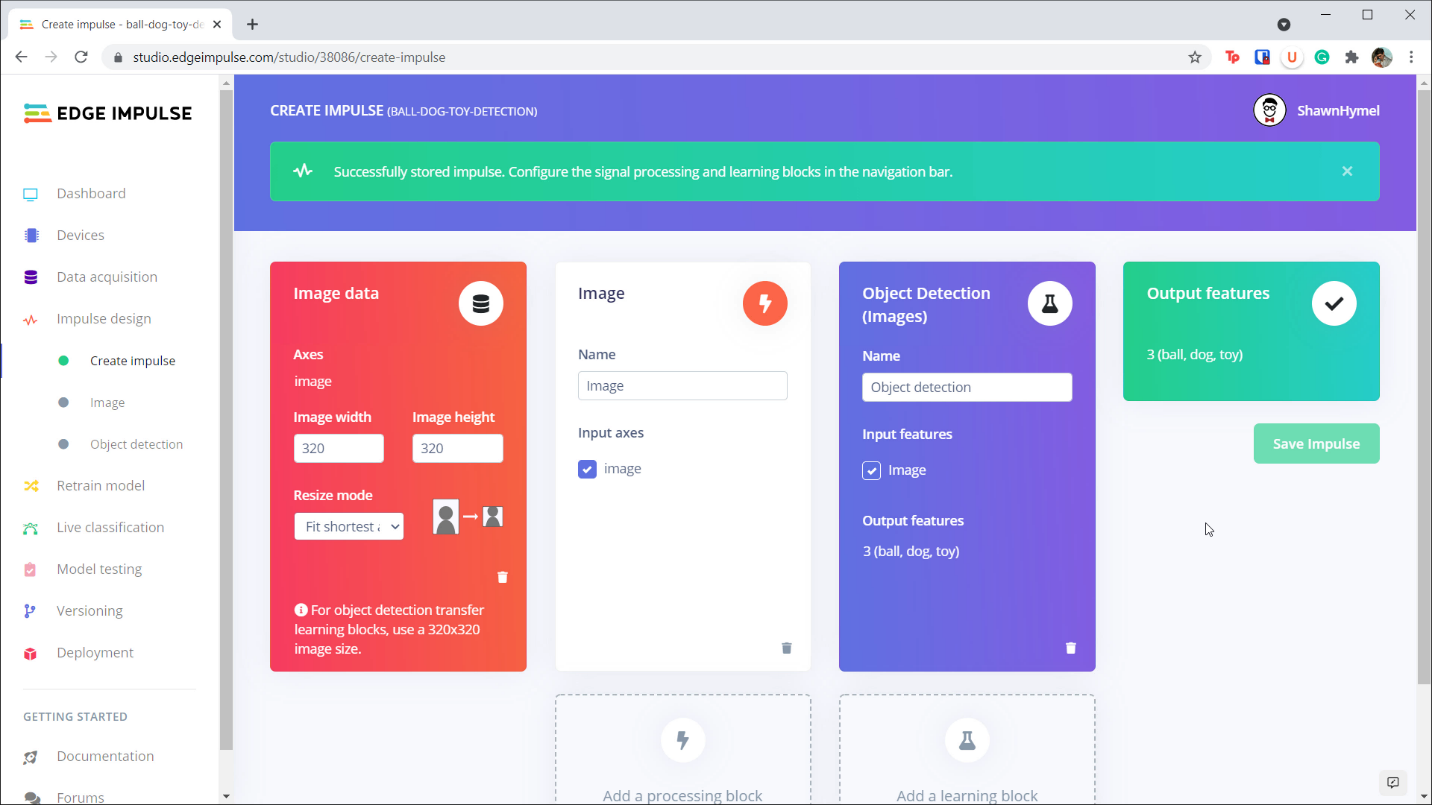
Create bounding boxes for all objects in your images.

If you make a mistake, keep going until the end. When you’re done, click on the **Training data** or **Testing data** tab to find the image with the error. Find the image, click the **3 dots** to the side of the image name, and click **Edit labels**. You will be presented with a pop-up window that allows you to edit the bounding box and label information.

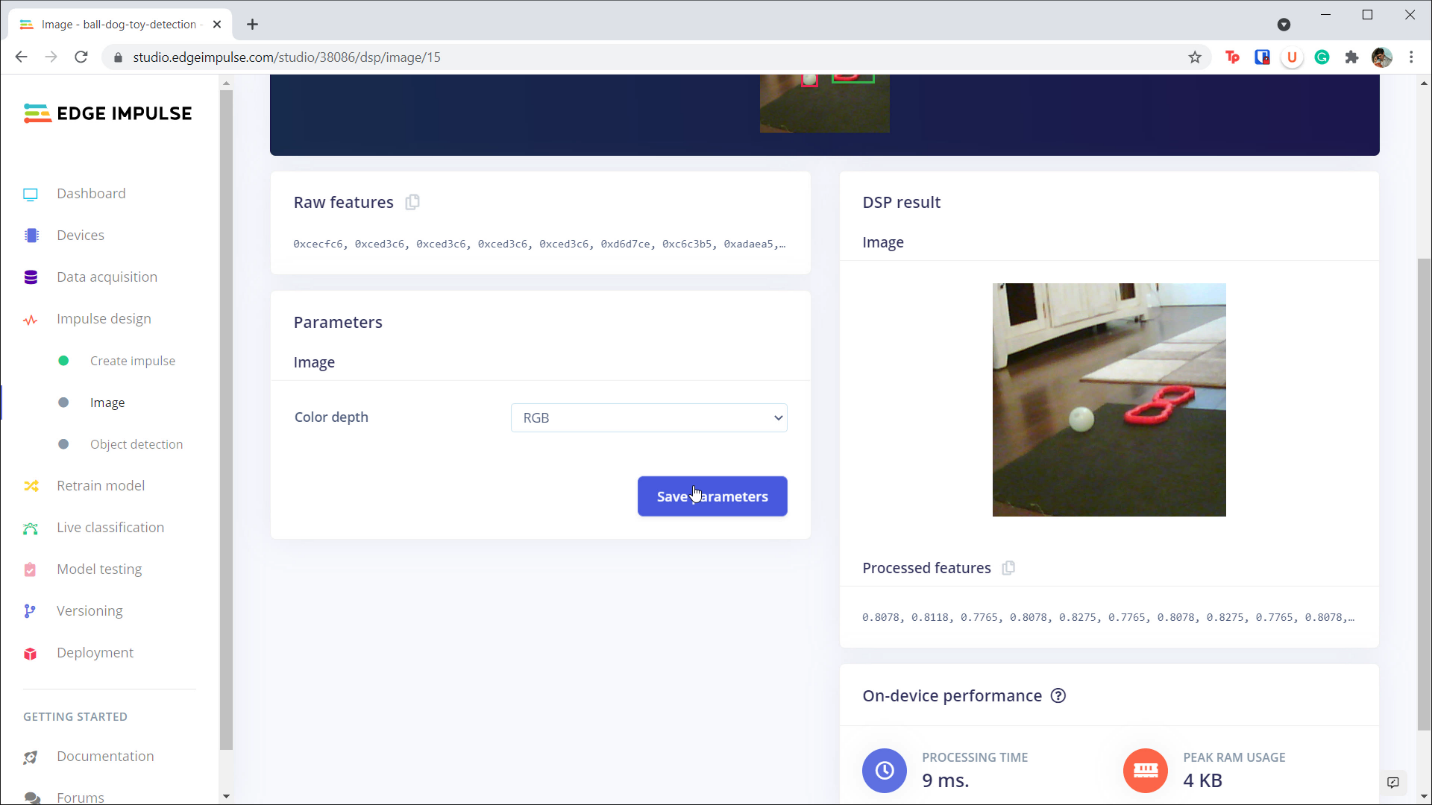


Go to **Impulse design**. Change the *Image data* to have a resolution of **320x320** (at this time, Edge Impulse only supports one object detection model, which requires a 320x320 input resolution).

Add an **Image** block for your processing block and **Object Detection (Images)** for your learning block. Click **Save Impulse**.

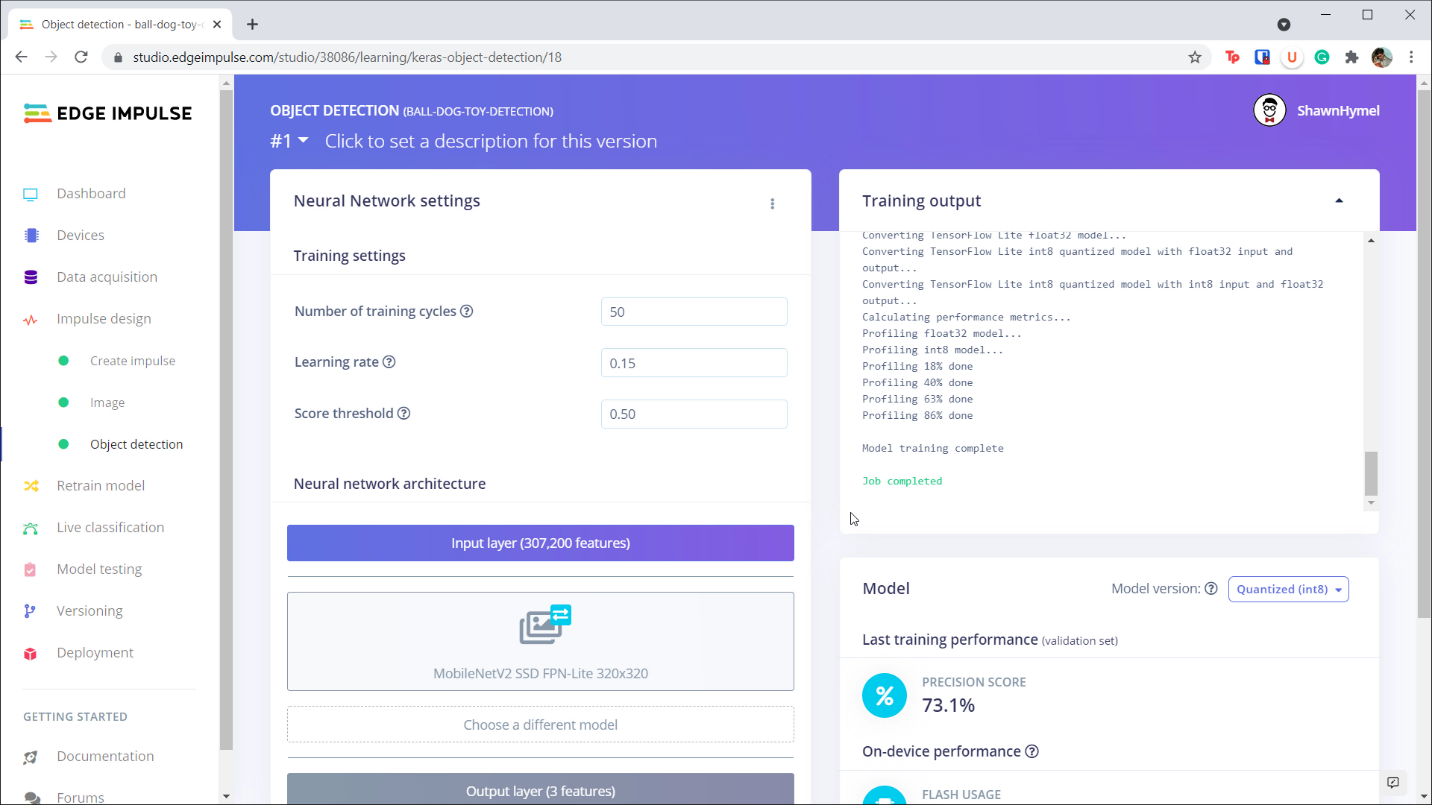


Click on **Image** in the navigation bar on the left side of the screen. Make sure that *Color depth* is set to **RGB**. Click **Save parameters** and then click **Generate Features** on the next screen.

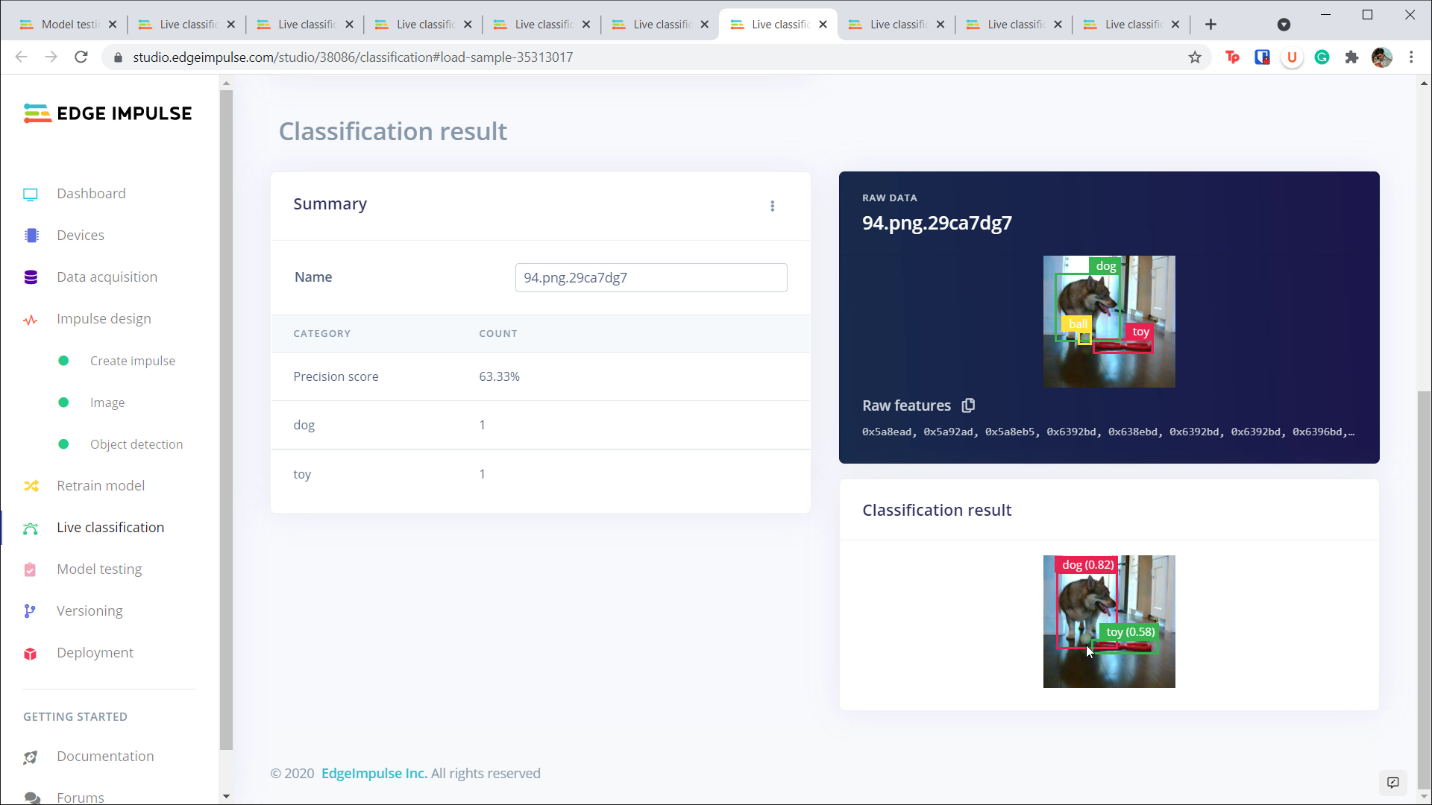


Once features have been extracted, click on **Object detection** in the navigation bar. At this time, you do not have many options for object detection models. Use the default *MobileNetV2 SSD FPN-Lite 320x320* model.

I recommend changing the *Number of training cycles* to **50**. Click **Start training**.



You are welcome to go to **Model testing** to see how well your model performs on your test data. As you can see from my example, the model had a hard time identifying small objects, like the ball in this test image:



When you are happy with the performance of your model, you can deploy it to your embedded system.

**Object Detection on the Raspberry Pi**

Create a folder to hold your program and model file:

**mkdir -p Projects/object-detection**

**cd Projects/object-detection**

Download the model file for your project:

**edge-impulse-linux-runner --clean --download modelfile.eim**

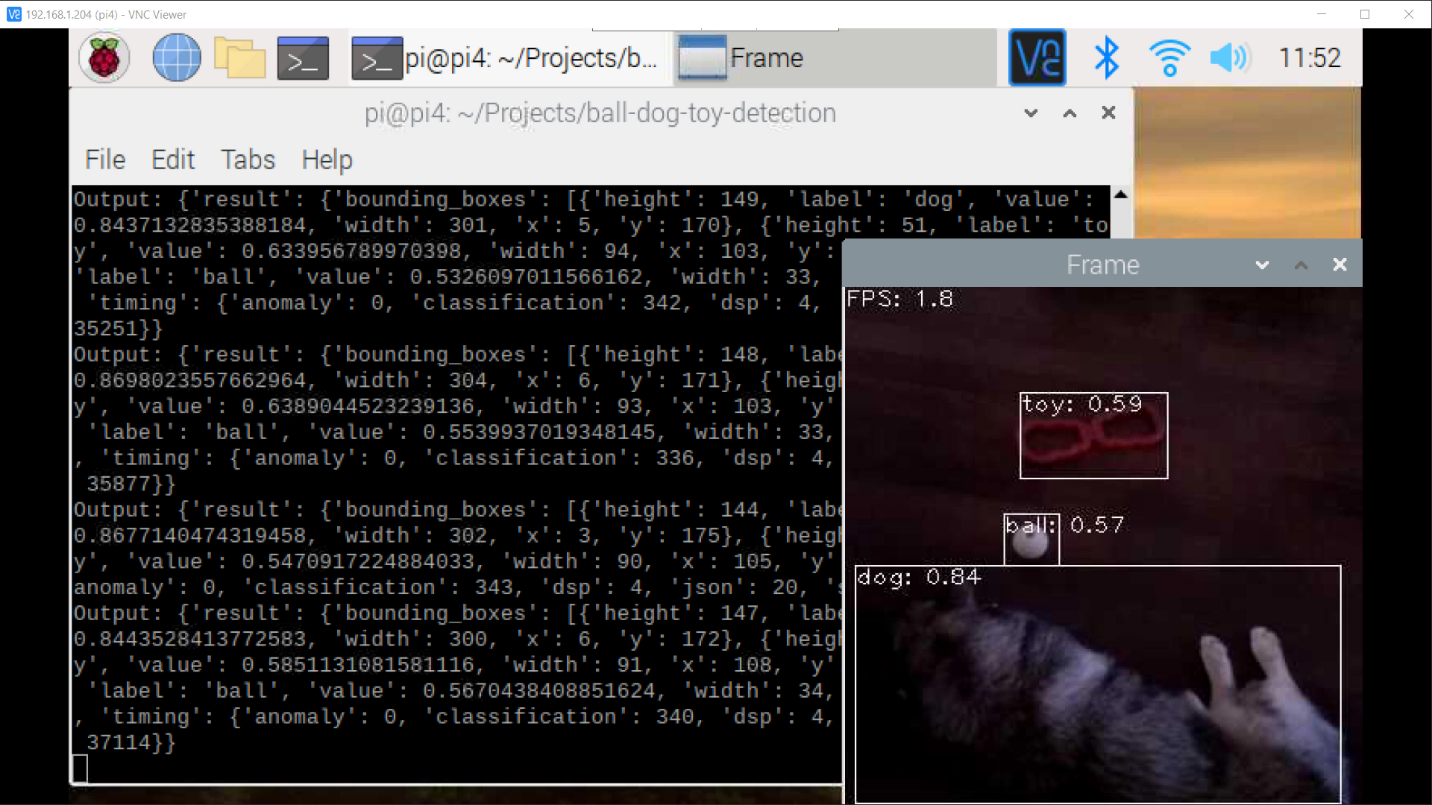
When asked, choose the project you wish to download the model from. Choose the project that you just created with the object detection model.

Create a Python program that captures an image from the camera and performs inference to locate all objects. Note that bounding box information will be output from the Edge Impulse library. You can save them in a list as follows:

bboxes = res[‘result’][‘bounding\_boxes’]

You may also reference the *Deploy Object Detection to Single Board Computer* lecture where I walk you through the code needed to complete this project. The code from that lecture can be [found here](https://github.com/ShawnHymel/computer-vision-with-embedded-machine-learning/blob/master/3.3.1%20-%20Deploy%20Object%20Detection%20Model%20(Raspberry%20Pi)/live-detection-pi-cam.py).

When you run the program, point the camera at various objects, and bounding boxes should be drawn on the preview window. Additionally, the inference results, including bounding box and class probabilities, should be printed to the console.



**Conclusion**

Object detection is a great start to many computer vision projects, including autonomous vehicles, animal or people trackers, robot vision, etc. However, you might notice that it can be painfully slow. Object detection requires a huge amount of computing resources, and so you should expect 1-5 frames per second using this particular object detection model on a Raspberry Pi 4.

You might be able to speed up inference using a more powerful computer (such as the [Nvidia Jetson Nano](https://www.youtube.com/watch?v=bcM5AQSAzUY)) or specialized hardware accelerators (such as the [Google Coral USB Accelerator](https://www.youtube.com/watch?v=qJMwNHQNOVU)).