Algorithmic Intelligence
Module B
Unit #9
CoCo SSD Bounding Box Image Classification

Content

Module B Unit #9 CoCoSSD bounding box image classification

We need an image, preferably one that has a few objects in it, not too many and not cluttered. For instance, the one below, I will include it on my website. Upload it the usual way and rename it photo (fruit or whatever). The actual image is quite large, but I have deliberately chosen a square image for simplicity. You can use your own image if you want.

Figure 1: a photo

Sketch B9.1 index.html

Make sure you have the `ml5.js` line of code in your `index.html` file.

```
<!DOCTYPE html>
<html lang="en"><head>
    <script src="https://cdn.jsdelivr.net/npm/p5@2.2.2/lib/p5.js"></script>
    <script src="https://unpkg.com/ml5@1/dist/ml5.min.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <meta charset="utf-8">
  </head>
  <body>
    <main>
    </main>
    <script src="sketch.js"></script>
</body></html>
```

Sketch B9.2 the cocossd model

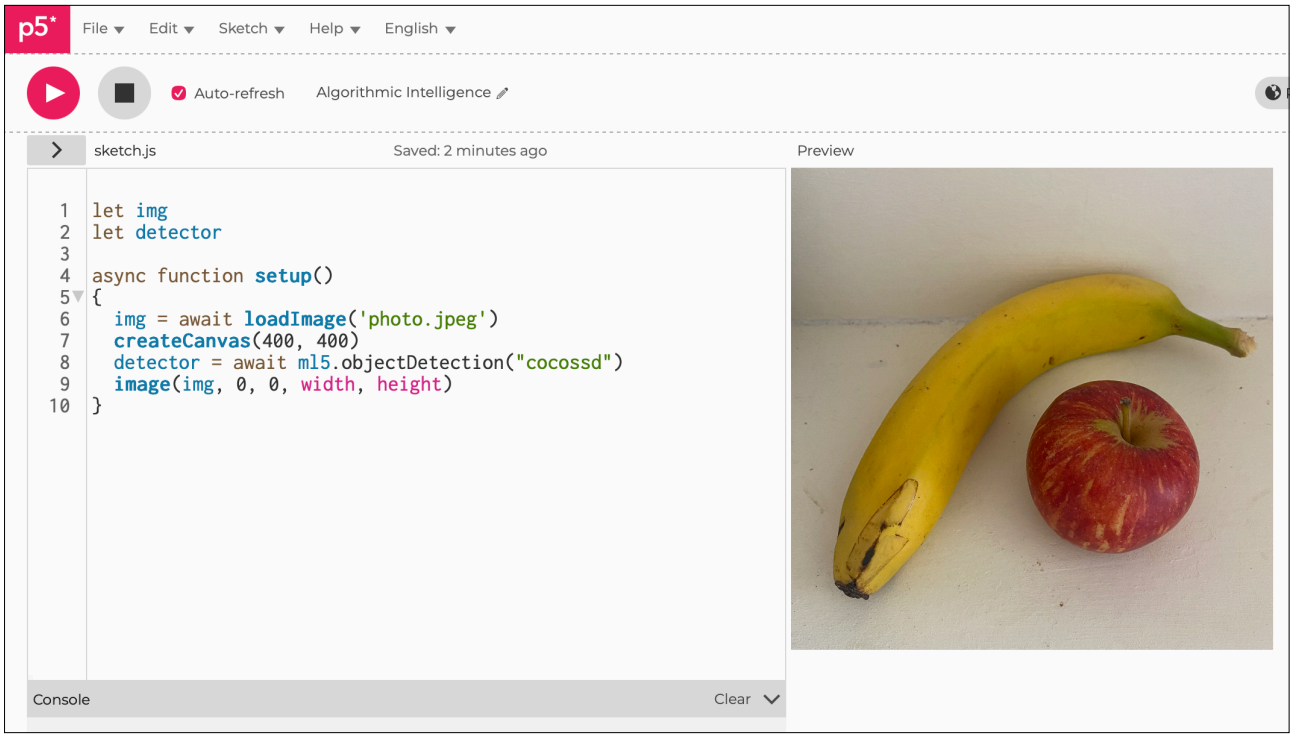
Our starting model, assuming that you have `ml5.js`.

```
let img
let detector

async function setup()
{
  img = await loadImage('photo.jpeg')
  createCanvas(400, 400)
  detector = await ml5.objectDetection("cocossd")
  image(img, 0, 0, width, height)
}
```

Notes

Just the basics.

Figure B9.2



Sketch B9.3 let's go detecting

We need an array to hold the results from the detection.

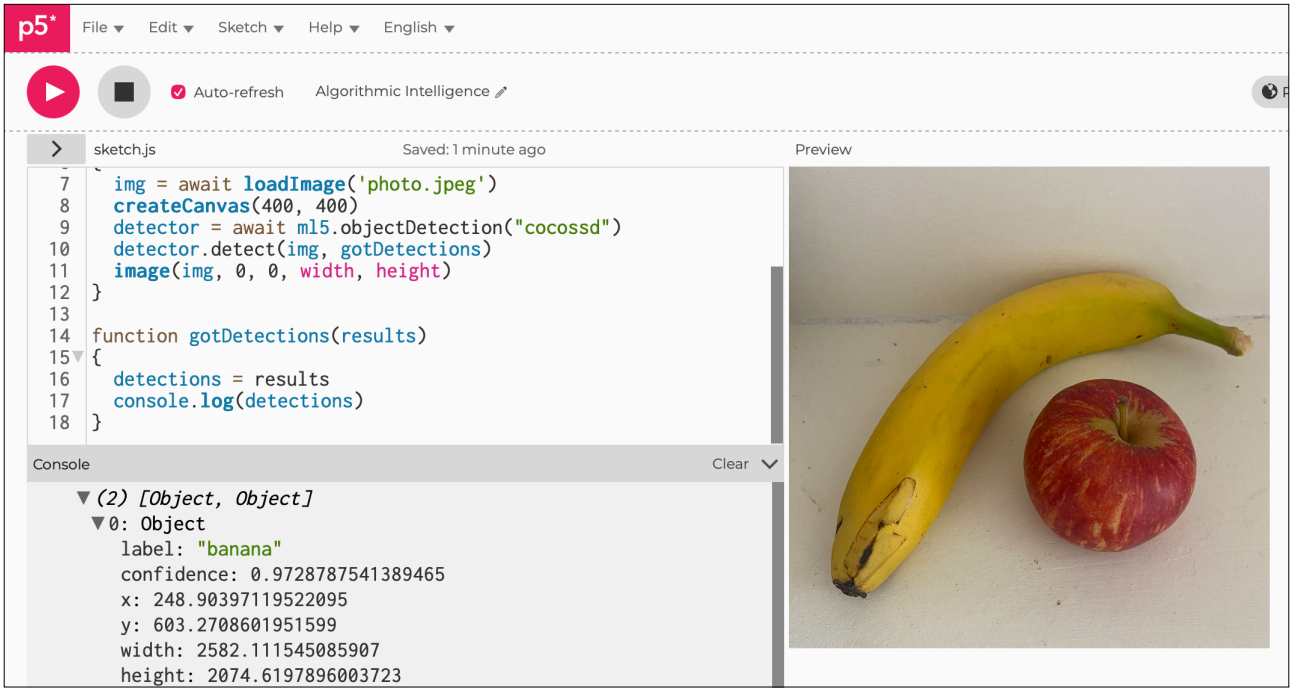
```
let img
let detector
let detections = []

async function setup()
{
  img = await loadImage('photo.jpeg')
  createCanvas(400, 400)
  detector = await ml5.objectDetection("cocossd")
  detector.detect(img, gotDetections)
  image(img, 0, 0, width, height)
}

function gotDetections(results)
{
  detections = results
  console.log(detections)
}
```

Notes

If you look in the console, you will see that it returns two objects, and if you open them up, you will see that they are correctly labelled banana and apple with very high confidence scores.

You will also note that the `x`, `y`, `width`, and `height` are extremely large compared to our canvas. That is because the original image is much larger, and we just scaled it to fit conveniently on our canvas.

Figure B9.3


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Intelligence
sketch.js
Saved: 1 minute ago
Preview
7 img = await loadImage('photo.jpeg')
8 createCanvas(400, 400)
9 detector = await ml5.objectDetection("cocossd")
10 detector.detect(img, gotDetections)
11 image(img, 0, 0, width, height)
12 }
13
14 function gotDetections(results)
15 {
16 detections = results
17 console.log(detections)
18 }
Console
Clear
▼(2) [Object, Object]
▼0: Object
label: "banana"
confidence: 0.9728787541389465
x: 248.90397119522095
y: 603.2708601951599
width: 2582.111545085907
height: 2074.6197896003723

Sketch B9.4 resizing

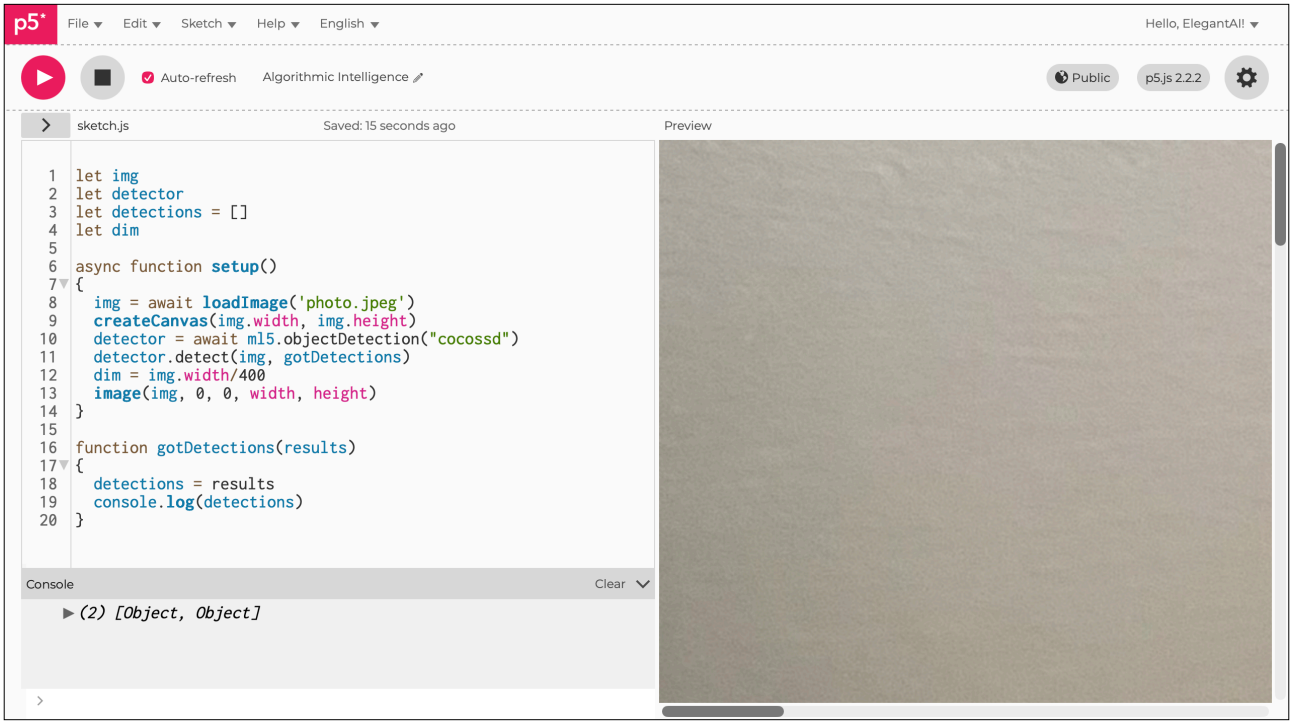
We will have a problem when we come to draw the bounding box. The dimensions in the array for the banana and the apple will most likely be somewhere off to the right, off-screen. We can mitigate this by using a ratio of image to canvas; also, you want to use an image that is not a square, which this will also help to scale in both dimensions.

```
let img
let detector
let detections = []
let dim

async function setup()
{
  img = await loadImage('photo.jpeg')
  createCanvas(img.width, img.height)
  detector = await ml5.objectDetection("cocossd")
  detector.detect(img, gotDetections)
  dim = img.width/400
  image(img, 0, 0, width, height)
}

function gotDetections(results)
{
  detections = results
  console.log(detections)
}
```

Notes

As you can see, the image is quite large, and we can only see a small part of it. So, we need to resize the image to fit the canvas.

Figure B9.4


p5*
File
Edit
Sketch
Help
English
Hello, ElegantAI!
Auto-refresh
Algorithmic Intelligence
Public
p5.js 2.2.2
sketch.js
Saved: 15 seconds ago
Preview
let img
let detector
let detections = []
let dim
async function setup()
{
img = await loadImage('photo.jpeg')
createCanvas(img.width, img.height)
detector = await ml5.objectDetection("cocossd")
detector.detect(img, gotDetections)
dim = img.width/400
image(img, 0, 0, width, height)
}
function gotDetections(results)
{
detections = results
console.log(detections)
}
Console
Clear
▶ (2) [Object, Object]

Sketch B9.5 rearrange the image

We now have the image of a decent size. The image has been moved into the `draw()` function and removed from the `setup()` function.

```
let img
let detector
let detections = []
let dim

async function setup()
{
  img = await loadImage('photo.jpeg')
  createCanvas(img.width, img.height)
  detector = await ml5.objectDetection("cocossd")
  detector.detect(img, gotDetections)
  dim = img.width/400
  // image(img, 0, 0, width, height)
}

function draw()
{
  image(img, 0, 0, width/dim, height/dim)
}

function gotDetections(results)
{
  detections = results
  console.log(detections)
}
```

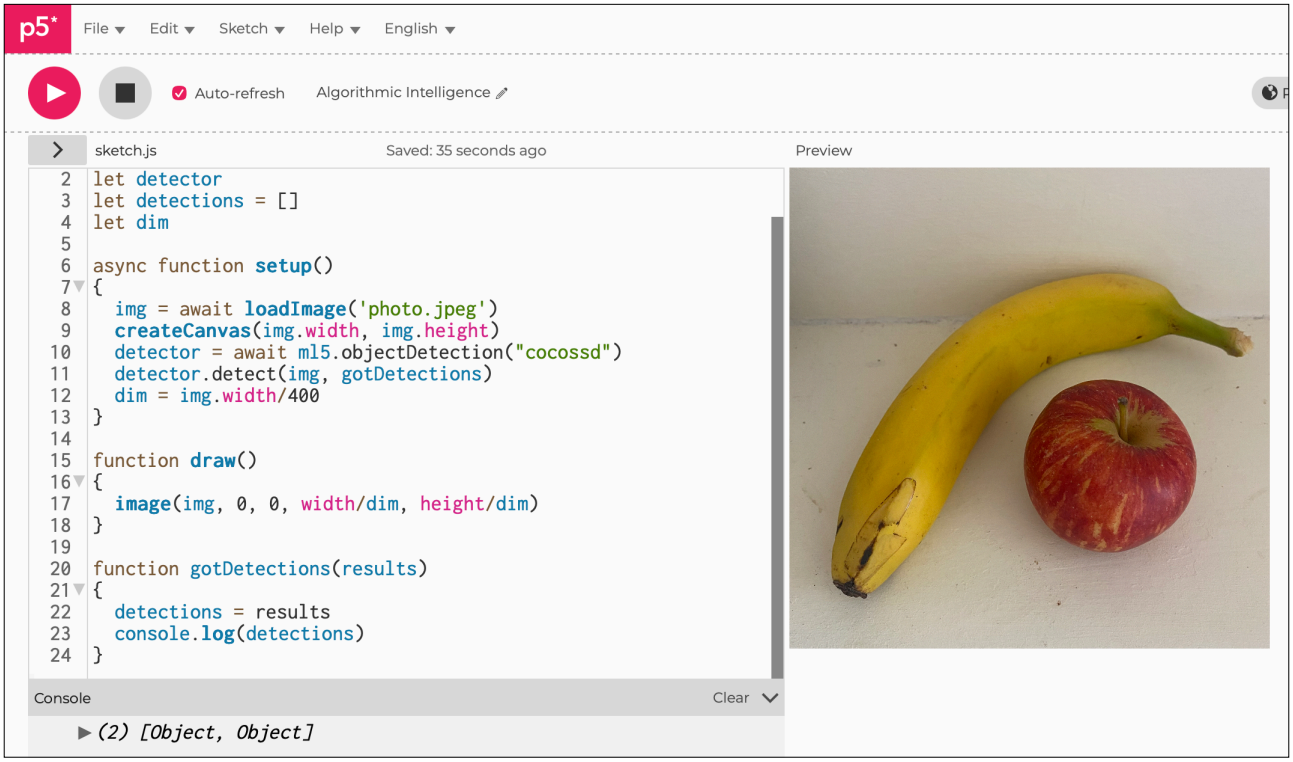
🗒️ Notes

We have divided the width by the dim value to give us the better canvas image.

🌻 Challenges

1. Try other values other than 400.
2. What happens if the image is not a square?

Figure B9.5


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Intelligence
sketch.js
Saved: 35 seconds ago
Preview
let detector
let detections = []
let dim
async function setup()
{
img = await loadImage('photo.jpeg')
createCanvas(img.width, img.height)
detector = await ml5.objectDetection("cocossd")
detector.detect(img, gotDetections)
dim = img.width/400
}
function draw()
{
image(img, 0, 0, width/dim, height/dim)
}
function gotDetections(results)
{
detections = results
console.log(detections)
}
Console
Clear
▶ (2) [Object, Object]

Sketch B9.6 bounding box

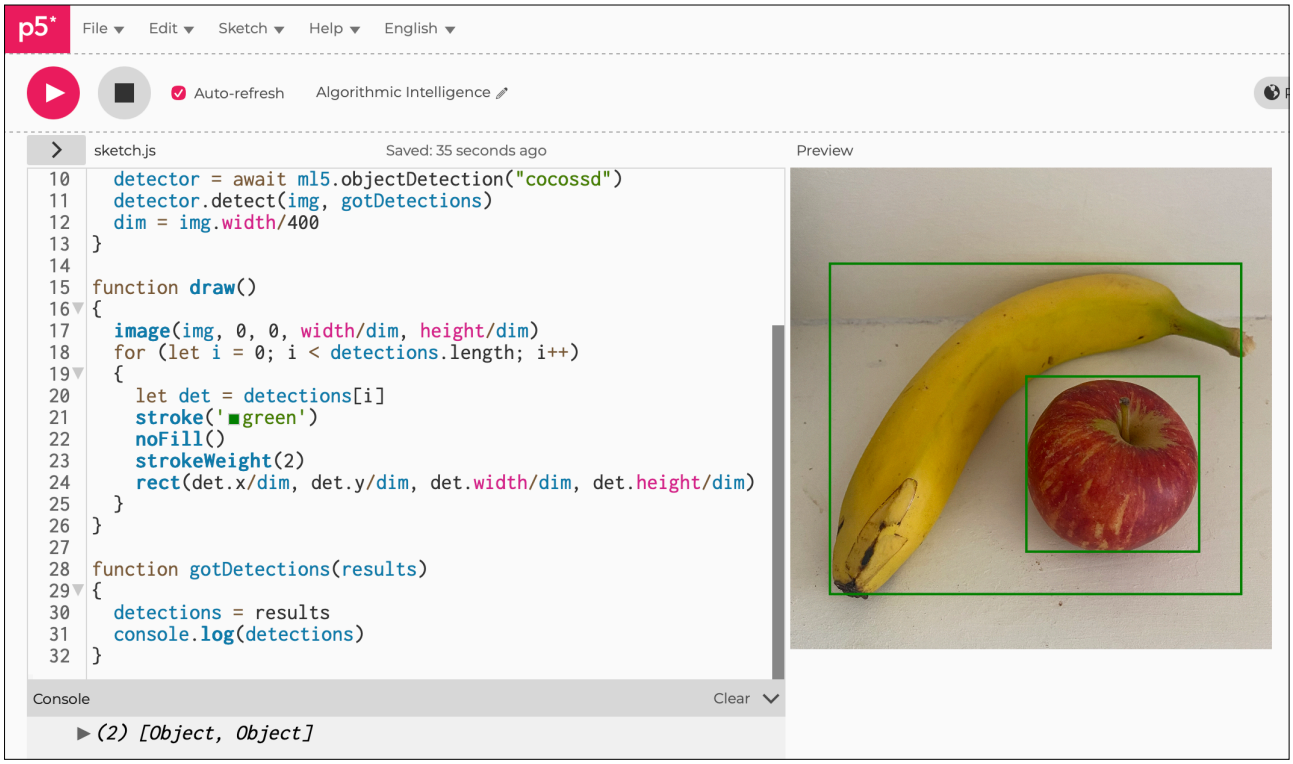
Now we have detected the objects in the image and a means of scaling things, we can use the data to draw a box around said objects. Because we have more than one object, we need to use a loop to work through all the objects (you might have many more than two).

```
let img
let detector
let detections = []
let dim

async function setup()
{
  img = await loadImage('photo.jpeg')
  createCanvas(img.width, img.height)
  detector = await ml5.objectDetection("cocossd")
  detector.detect(img, gotDetections)
  dim = img.width/400
}

function draw()
{
  image(img, 0, 0, width/dim, height/dim)
  for (let i = 0; i < detections.length; i++)
  {
    let det = detections[i]
    stroke('green')
    noFill()
    strokeWeight(2)
    rect(det.x/dim, det.y/dim, det.width/dim, det.height/dim)
  }
}

function gotDetections(results)
{
  detections = results
  console.log(detections)
}
```

🗒️ Notes

We get a nice green box for each object.

Figure B9.6



Sketch B9.7 labelling the objects

We can also insert the names of each object inside their respective bounding boxes.

```
let img
let detector
let detections = []
let dim

async function setup()
{
  img = await loadImage('photo.jpeg')
  createCanvas(img.width, img.height)
  detector = await ml5.objectDetection("cocossd")
  detector.detect(img, gotDetections)
  dim = img.width/400
}

function draw()
{
  image(img, 0, 0, width/dim, height/dim)
  for (let i = 0; i < detections.length; i++)
  {
    let det = detections[i]
    stroke('green')
    noFill()
    strokeWeight(2)
    rect(det.x/dim, det.y/dim, det.width/dim, det.height/dim)

    noStroke()
    fill('white')
    textSize(15)
    text(det.label, det.x/dim + 5, det.y/dim + 15)
  }
}

function gotDetections(results)
{
```

```
  detections = results
  console.log(detections)
}
```

🗒️ Notes

Clearly labelled and bounded. Suggest removing the `console.log()` if not needed.

🌻 Challenge

Try other colours, maybe add a background to the text so that it can be seen against a mixed colour background.

Figure B9.7