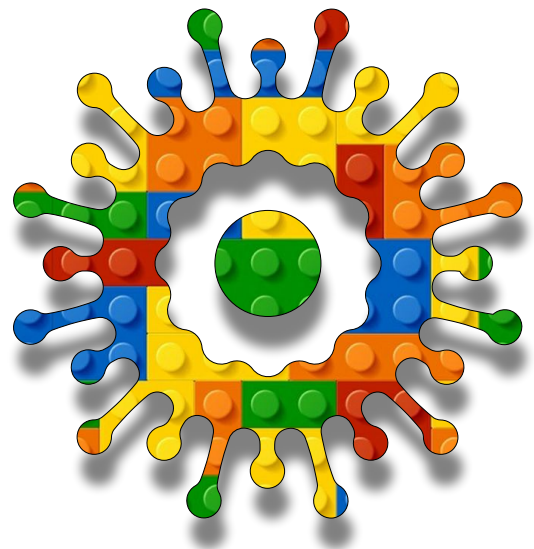


Algorithmic
Intelligence
Module B
Unit #7
mobileNet
Image
Classification





Module B Unit #7 mobileNet image classification

Sketch B7.1	index.html
Sketch B7.2	callback
A banana	
Sketch B7.3	the appearance of the banana
Sketch B7.4	the classification
Sketch B7.5	get the results
Sketch B7.6	display



Introduction to mobileNet image classification

The pre-trained model we are going to use is called **mobileNet**. It is a very handy model that you can use online. It uses a dataset from **ImageNet**, which has nearly 15 million images. So it is quite comprehensive but not complete, as you may well find out when you try to look at objects.

This unit, we just upload images to classify.



Sketch B7.1 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 B7.2 callback

We have now access to another pre-trained model with `ml5.js` called `mobileNet`. We are going to check that everything is working by having a callback. The callback tells us when it is loaded in the console.

```
let mobilenet

function setup()
{
  createCanvas(400, 400)
  mobilenet = ml5.imageClassifier('MobileNet', modelReady)
}

function modelReady()
{
  console.log('model is ready')
}
```



Notes

We get the callback console log: `model is ready`. It may take a few seconds.



A banana

You can use any object you like, but choose something that doesn't have any other object in it. Just make sure it works OK from the start. I have selected a banana image from [Unsplash](#).

Figure 1: Source: 'Mockup Graphics' on Unsplash

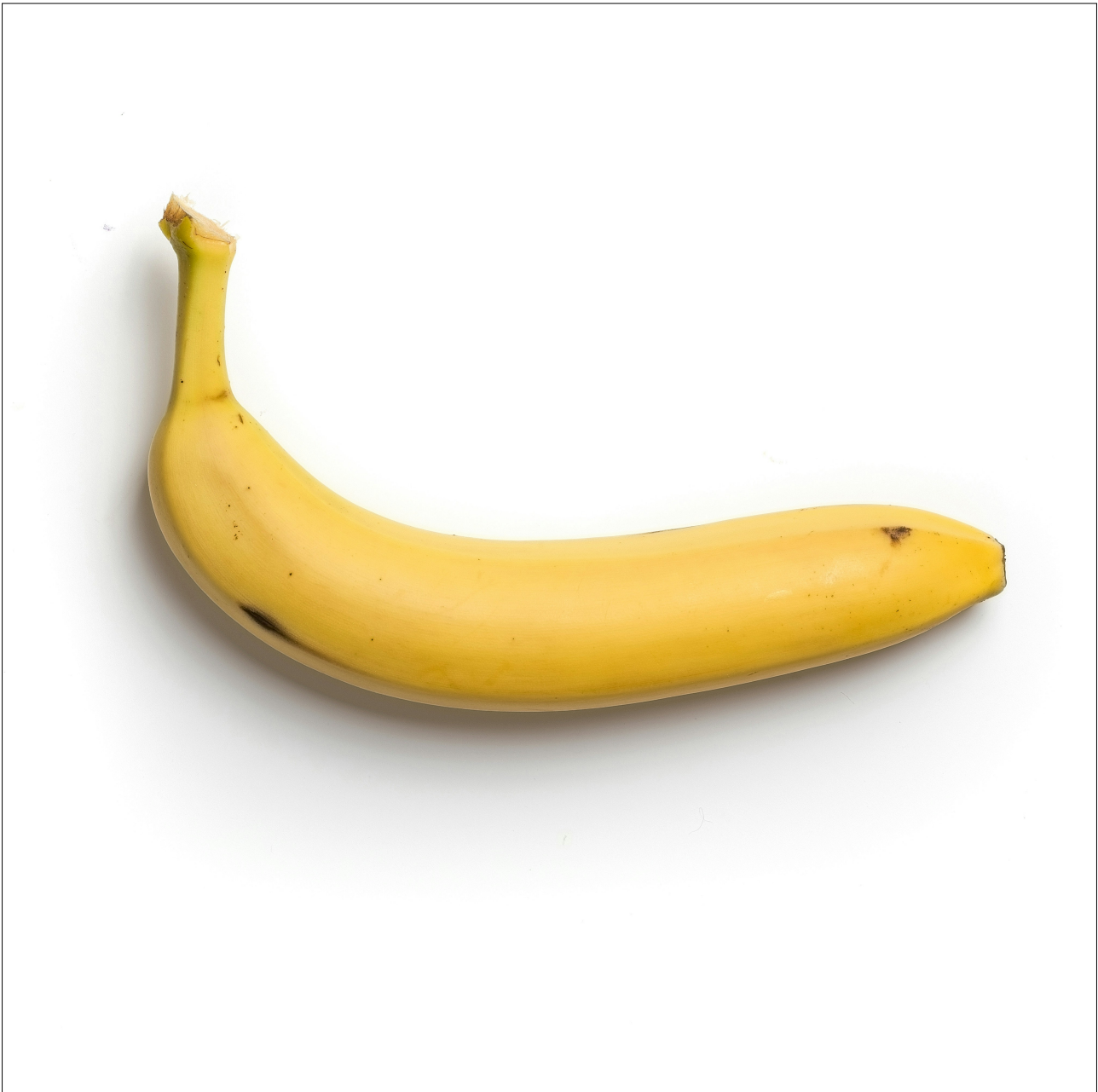


Figure 2: Upload it to the sketch

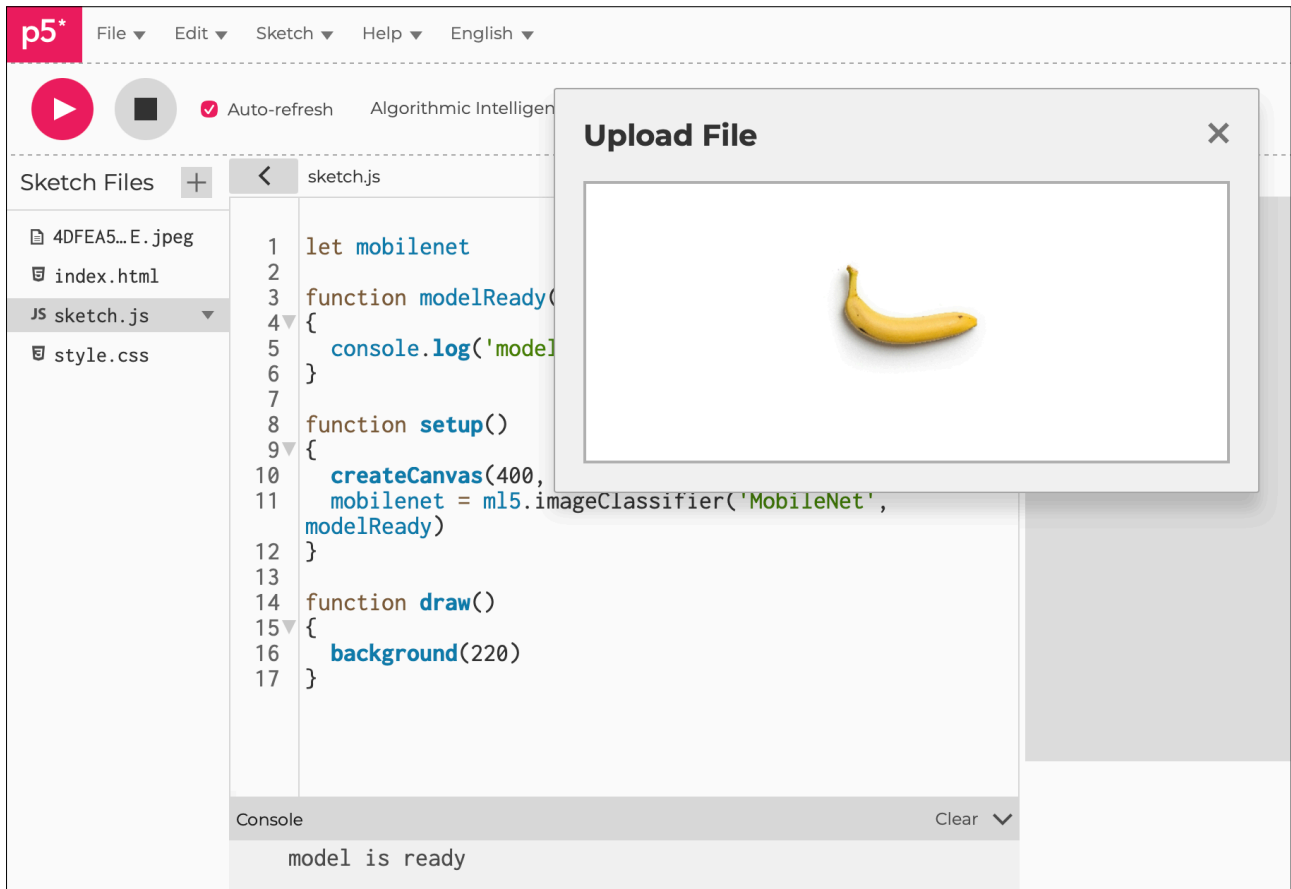
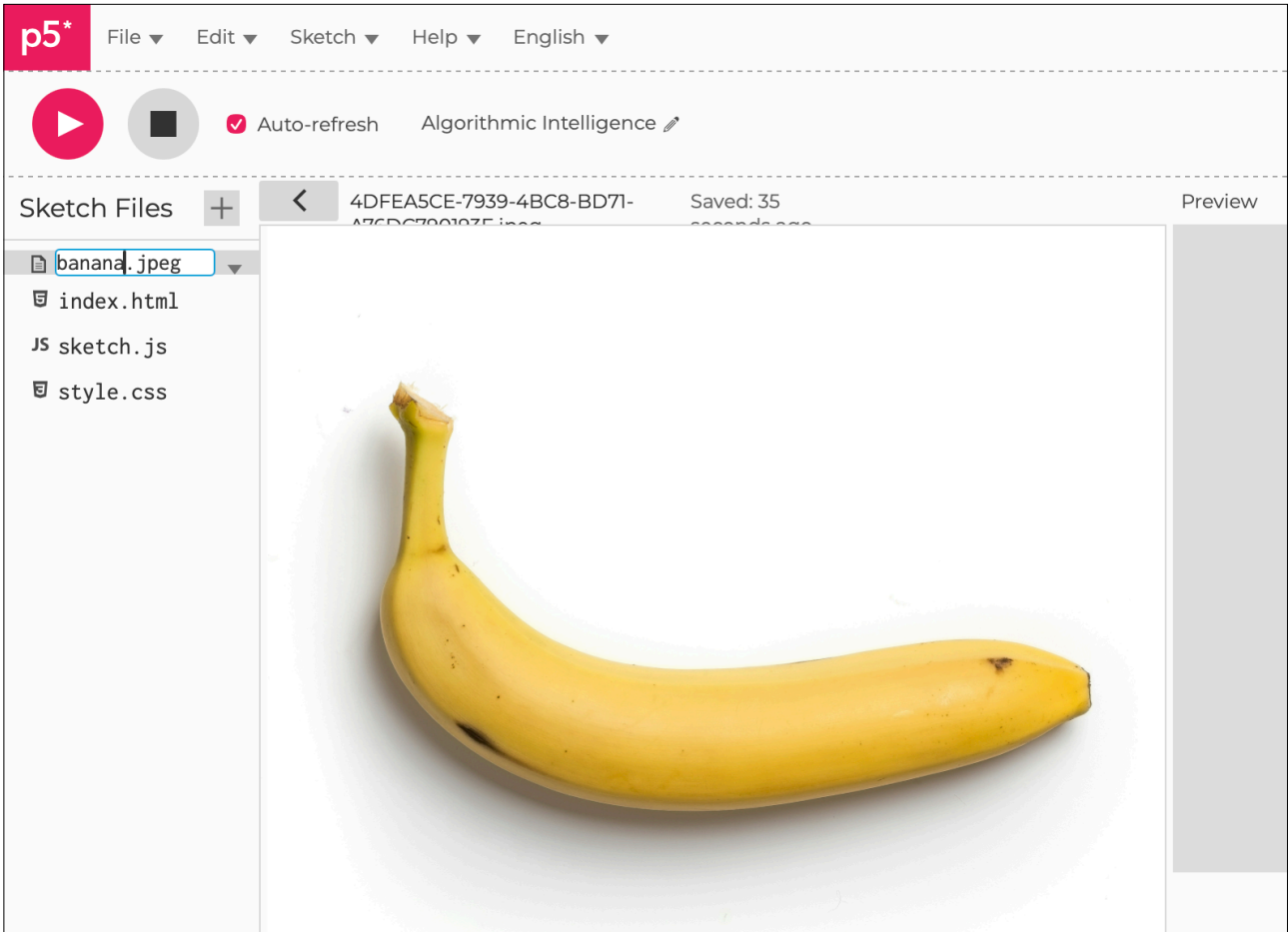


Figure 3: rename 'banana'





Sketch B7.3 the appearance of the banana

Now that you have successfully uploaded an image and renamed it, then we can display it on the canvas.

```
let mobilenet
let img

async function setup()
{
  createCanvas(400, 400)
  img = await loadImage('banana.jpeg')
  image(img, 0, 0, width, height)
  mobilenet = ml5.imageClassifier('MobileNet', modelReady)
}

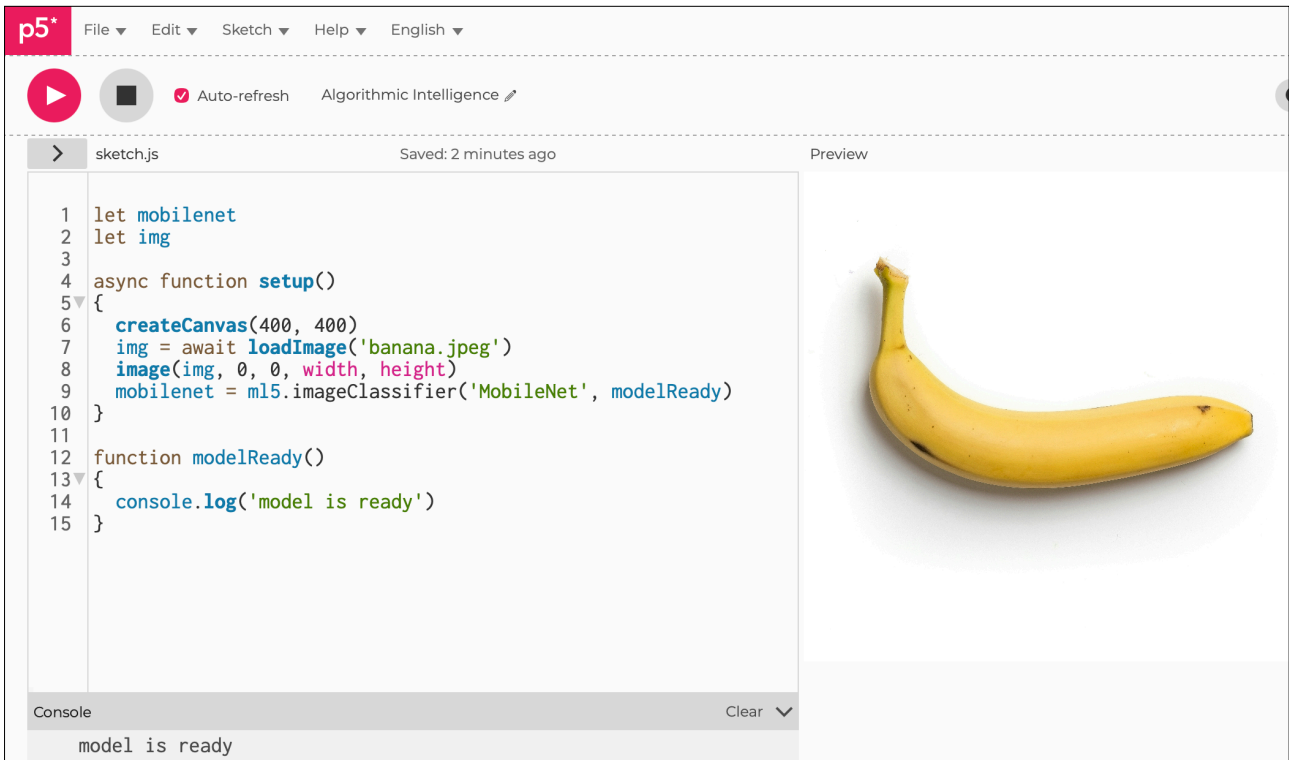
function modelReady()
{
  console.log('model is ready')
}
```



Notes

We use **async** and **await** because we want the image to be downloaded first, before the model is loaded. The banana should magically appear on the canvas. I deliberately chose a square image so there's no squashing or stretching.

Figure B7.3



The screenshot shows the p5.js IDE interface. At the top, there is a menu bar with 'File', 'Edit', 'Sketch', 'Help', and 'English'. Below the menu bar, there are control buttons for play, stop, and auto-refresh, along with the text 'Algorithmic Intelligence'. The main workspace is divided into two panels: a code editor on the left and a preview window on the right. The code editor shows the following JavaScript code:

```
1 let mobilenet
2 let img
3
4 async function setup()
5 {
6   createCanvas(400, 400)
7   img = await loadImage('banana.jpeg')
8   image(img, 0, 0, width, height)
9   mobilenet = ml5.imageClassifier('MobileNet', modelReady)
10 }
11
12 function modelReady()
13 {
14   console.log('model is ready')
15 }
```

The preview window displays a yellow banana on a white background. Below the code editor, there is a console window showing the output 'model is ready'.



Sketch B7.4 the classification

We can make a few changes to the sketch to get the model to **classify** (predict) what the object is in the image. The first change is that we need the model to load before we start to classify. We get rid of the callback and change its name (to **gotResults**) so we can repurpose it for collecting the results of the classification.

```
let mobilenet
let img

async function setup()
{
  createCanvas(400, 400)
  img = await loadImage('banana.jpeg')
  image(img, 0, 0, width, height)
  mobilenet = await ml5.imageClassifier('MobileNet')
  mobilenet.classify(img, gotResults)
}

function gotResults(results)
{
  console.log(results)
}
```

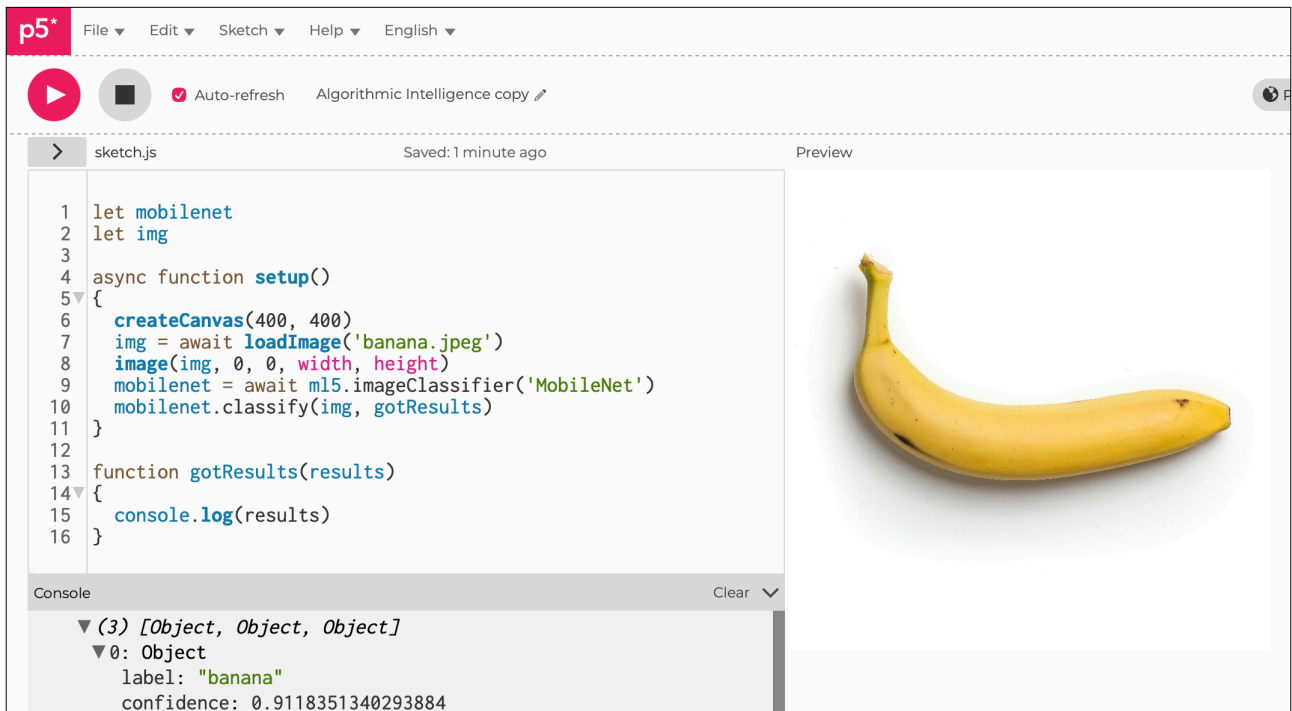


Notes

You will notice that we get the top three guesses (predictions), and they are scored as a probability. Clearly, this is a banana with almost **100% (0.91)** score; the other two are interesting. A slug, I could just about get!

! If you get an error message, just check you have removed the redundant code.

Figure B7.4



The screenshot shows the p5.js IDE interface. The top menu bar includes 'File', 'Edit', 'Sketch', 'Help', and 'English'. Below the menu bar, there are control buttons for play, stop, and auto-refresh, along with the text 'Algorithmic Intelligence copy'. The main workspace is divided into two sections: a code editor on the left and a preview window on the right.

The code editor contains the following JavaScript code:

```
1 let mobilenet
2 let img
3
4 async function setup()
5 {
6   createCanvas(400, 400)
7   img = await loadImage('banana.jpeg')
8   image(img, 0, 0, width, height)
9   mobilenet = await ml5.imageClassifier('MobileNet')
10  mobilenet.classify(img, gotResults)
11 }
12
13 function gotResults(results)
14 {
15   console.log(results)
16 }
```

The preview window displays a high-resolution image of a single yellow banana on a white background.

Below the code editor is a console window with the following output:

```
▼ (3) [Object, Object, Object]
  ▼ 0: Object
    label: "banana"
    confidence: 0.9118351340293884
```



Sketch B7.5 get the results

We want the top result, which is the `label` at `index [0]`, the first one. This is also true for probability (confidence). We create two variables to hold these values as strings, as we will want to display them on the canvas as a fixed value or name.

! Remove the console log.

```
let mobilenet
let img
let label = ''
let confidence = ''

async function setup()
{
  createCanvas(400, 400)
  img = await loadImage('banana.jpeg')
  image(img, 0, 0, width, height)
  mobilenet = await ml5.imageClassifier('MobileNet')
  mobilenet.classify(img, gotResults)
}

function gotResults(results)
{
  label = results[0].label
  confidence = results[0].confidence
}
```



Notes

Nothing new to see.



Sketch B7.6 display

! Move the image into the `draw()` function from `setup()`.

We multiply the confidence by `100` (to get the percentage) and take the floor value. We then print the text onto the canvas. We use the `draw()` function for this.

```
let mobilenet
let img
let label = ''
let confidence = ''

async function setup()
{
  createCanvas(400, 400)
  img = await loadImage('banana.jpeg')
  mobilenet = await ml5.imageClassifier('MobileNet')
  mobilenet.classify(img, gotResults)
}

function draw()
{
  image(img, 0, 0, width, height)
  textSize(20)
  text('label: ' + label, 100, 350)
  text('confidence: ' + floor(confidence * 100) + '%', 100, 380)
}

function gotResults(results)
{
  label = results[0].label
  confidence = results[0].confidence
}
```

Notes

A very confident banana

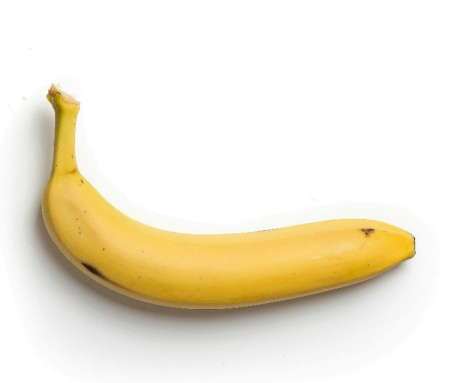
Figure B7.6

p5* File Edit Sketch Help English

Auto-refresh Algorithmic Intelligence

sketch.js Saved: 35 seconds ago Preview

```
5
6 async function setup()
7 {
8   createCanvas(400, 400)
9   img = await loadImage('banana.jpeg')
10  mobilenet = await ml5.imageClassifier('MobileNet')
11  mobilenet.classify(img, gotResults)
12 }
13
14 function draw()
15 {
16   image(img, 0, 0, width, height)
17   textSize(20)
18   text('label: ' + label, 100, 350)
19   text('confidence: ' + floor(confidence * 100) + '%', 100,
20 380)
21 }
22
23 function gotResults(results)
24 {
25   label = results[0].label
26   confidence = results[0].confidence
27 }
```



label: banana
confidence: 91%

Console Clear