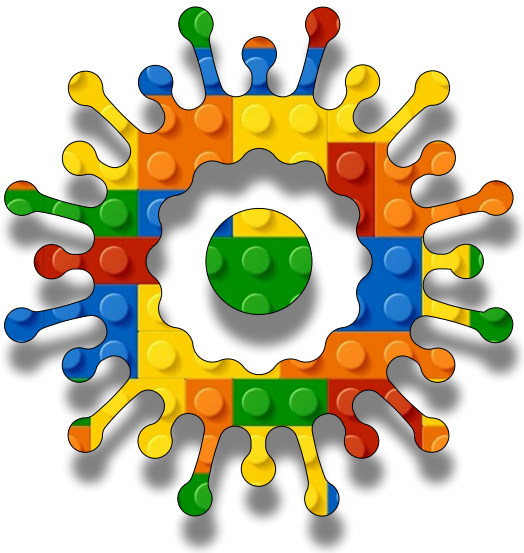
Algorithmic Intelligence Module A Unit #11 colour predictor



Content

Module A Unit #11 colour predictor

Introduction to colour predictor
The index.html file
Sketch A11.1 creating the data
Sketch A11.2 sliders
Sketch A11.3 adding the neural network
Sketch A11.4 adding the data
Sketch A11.5 normalising the data
Sketch A11.6 training the neural network
Sketch A11.7 classifying the colour
Sketch A11.8 the best prediction

Introduction to colour predictor

Another seemingly simple exercise is to train the model to identify `reddish`, `greenish`, and `blueish` colours. For this, we will provide a small synthetic dataset in the form of a `JSON-style` file. We will train the neural network on that data and see if it can learn these three classifications for a variety of combinations of the RGB values.

This is, therefore, a `classification` task as we want it to select, through the built-in softmax function, the most probable colour group. We will use three sliders to change the amount of `red`, `green`, and `blue`.

The index.html

A reminder to add the `ml5.js` line of code if you haven't already.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.11.1/p5.js"></
script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.11.1/addons/
p5.sound.min.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 A11.1 creating the data

We are going to create three labels: `red-ish`, `green-ish`, and `blue-ish`. For each of these three labels, we are going to give them the RGB numbers that are pretty close. For example, `red-ish` is `(255, 0, 0)`, `(254, 0, 0)`, and `(253, 0, 0)`, and do the same for the `green-ish` and `blue-ish`.

```
let data = [
  { r: 255, g: 0, b: 0, colour: "red-ish" },
  { r: 254, g: 0, b: 0, colour: "red-ish" },
  { r: 253, g: 0, b: 0, colour: "red-ish" },
  { r: 0, g: 255, b: 0, colour: "green-ish" },
  { r: 0, g: 254, b: 0, colour: "green-ish" },
  { r: 0, g: 253, b: 0, colour: "green-ish" },
  { r: 0, g: 0, b: 255, colour: "blue-ish" },
  { r: 0, g: 0, b: 254, colour: "blue-ish" },
  { r: 0, g: 0, b: 253, colour: "blue-ish" },
]

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
}
```

🗒️ Notes

This is another small dataset; in reality, you would have a much larger dataset based on people's options (labels), but this is simple enough to be useful.

🌻 Challenge

You could add more data.

🛠️ Code Explanation

`let data = [{r:, g:, b:, colour:},]`	Creating a series of objects in an array called data. These will be the input data

Sketch A11.2 Sliders

We are going to create three sliders to control the colour of the background. We will give it an initial colour of red.

```
let data = [
  { r: 255, g: 0, b: 0, colour: "red-ish" },
  { r: 254, g: 0, b: 0, colour: "red-ish" },
  { r: 253, g: 0, b: 0, colour: "red-ish" },
  { r: 0, g: 255, b: 0, colour: "green-ish" },
  { r: 0, g: 254, b: 0, colour: "green-ish" },
  { r: 0, g: 253, b: 0, colour: "green-ish" },
  { r: 0, g: 0, b: 255, colour: "blue-ish" },
  { r: 0, g: 0, b: 254, colour: "blue-ish" },
  { r: 0, g: 0, b: 253, colour: "blue-ish" },
]

let r
let g
let b
let rSlider
let gSlider
let bSlider

function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 255).position(10, 20)
  gSlider = createSlider(0, 255, 0).position(10, 40)
  bSlider = createSlider(0, 255, 0).position(10, 60)
}

function draw()
{
  r = rSlider.value()
  g = gSlider.value()
  b = bSlider.value()
  background(r, g, b)
```

```
}
```

🗒️ Notes

The slider has three arguments:

🀄 The first is the minimum value,

🀄 The second is the maximum value, and

🀄 The third is the starting value.

We give the red slider `rSlider` a starting value of `255`, which is the maximum.

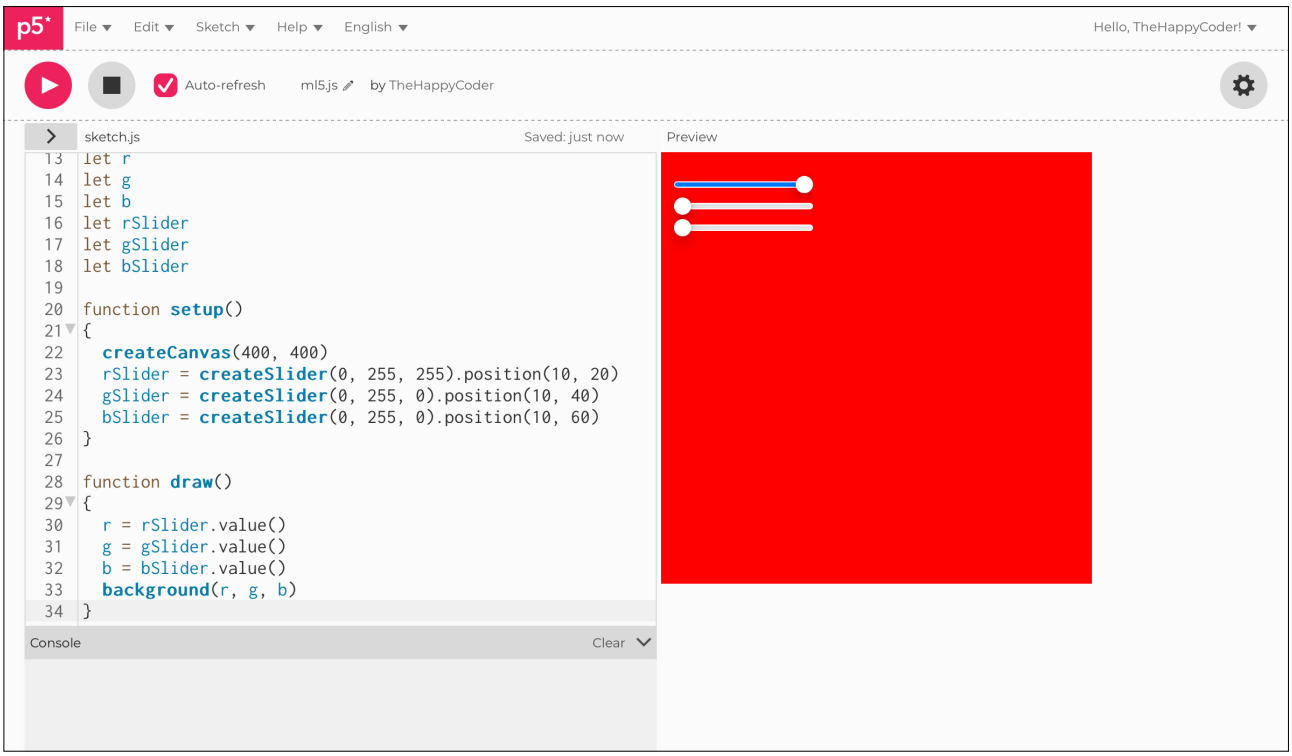
🌻 Challenge

Have one of the others start at a different value.

🛠️ Code Explanation

`createSlider(0, 255, 255)`	The third argument is the value of the slider
`createSlider(...).position(10, 20)`	The position is the x (10 pixels) and y (20 pixels) co-ordinates of the slider

Figure A11.2



Sketch A11.3 adding the neural network

Let's introduce our neural network, as we have done numerous times before.

```
let data = [
  { r: 255, g: 0, b: 0, colour: "red-ish" },
  { r: 254, g: 0, b: 0, colour: "red-ish" },
  { r: 253, g: 0, b: 0, colour: "red-ish" },
  { r: 0, g: 255, b: 0, colour: "green-ish" },
  { r: 0, g: 254, b: 0, colour: "green-ish" },
  { r: 0, g: 253, b: 0, colour: "green-ish" },
  { r: 0, g: 0, b: 255, colour: "blue-ish" },
  { r: 0, g: 0, b: 254, colour: "blue-ish" },
  { r: 0, g: 0, b: 253, colour: "blue-ish" },
]

let r
let g
let b
let rSlider
let gSlider
let bSlider
let nn

function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 255).position(10, 20)
  gSlider = createSlider(0, 255, 0).position(10, 40)
  bSlider = createSlider(0, 255, 0).position(10, 60)
  let options = {
    task: 'classification',
    debug: true
  }
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
}
```

```
function draw()
{
  r = rSlider.value()
  g = gSlider.value()
  b = bSlider.value()
  background(r, g, b)
}
```

🗒️ Notes

As you can see, we have a `classification` task and set the debug to `true`.

Sketch A11.4 adding the data

Now, to add the data to the neural network. We cycle through the `data` array, giving the set of inputs and the colour label to a variable called `item`. This holds the value of the `red`, `green`, and `blue` as well as the colour label. However, we only pull out the `r`, `g`, and `b` values. The `colour` values are the outputs for each set of corresponding `r`, `g`, `b` values.

```
let data = [
  { r: 255, g: 0, b: 0, colour: "red-ish" },
  { r: 254, g: 0, b: 0, colour: "red-ish" },
  { r: 253, g: 0, b: 0, colour: "red-ish" },
  { r: 0, g: 255, b: 0, colour: "green-ish" },
  { r: 0, g: 254, b: 0, colour: "green-ish" },
  { r: 0, g: 253, b: 0, colour: "green-ish" },
  { r: 0, g: 0, b: 255, colour: "blue-ish" },
  { r: 0, g: 0, b: 254, colour: "blue-ish" },
  { r: 0, g: 0, b: 253, colour: "blue-ish" },
]

let r
let g
let b
let rSlider
let gSlider
let bSlider
let nn

function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 255).position(10, 20)
  gSlider = createSlider(0, 255, 0).position(10, 40)
  bSlider = createSlider(0, 255, 0).position(10, 60)
  let options = {
    task: 'classification',
    debug: true
  }
  ml5.setBackend("webgl")
```

```
  nn = ml5.neuralNetwork(options)
  for (let i = 0; i < data.length; i++)
  {
    let item = data[i]
    let inputs = [item.r, item.g, item.b]
    let outputs = [item.colour]
    nn.addData(inputs, outputs)
  }
}

function draw()
{
  r = rSlider.value()
  g = gSlider.value()
  b = bSlider.value()
  background(r, g, b)
}
```

🗒️ Notes

Using the `item` variable seems a bit redundant, but you are taking them from the complete array. It means you don't accidentally change any values in the dataset. This repeats for the length of the dataset.

🌻 Challenge

You could replace `item` with `data[i]` and remove all references to the `item` variable.

🛠️ Code Explanation

`let item = data[i]`	For each index if the dataset array push it into the item variable
`let inputs = [item.r, item.g, item.b]`	Each set of inputs are added
`let outputs = [item.colour]`	The corresponding output is added
`nn.addData(inputs, outputs)`	Both the inputs and outputs are added to the neural network

Sketch A11.5 normalising the data

Next, we will normalise the data because we don't want very large and very small numbers in the neural network calculations.

```
let data = [
  { r: 255, g: 0, b: 0, colour: "red-ish" },
  { r: 254, g: 0, b: 0, colour: "red-ish" },
  { r: 253, g: 0, b: 0, colour: "red-ish" },
  { r: 0, g: 255, b: 0, colour: "green-ish" },
  { r: 0, g: 254, b: 0, colour: "green-ish" },
  { r: 0, g: 253, b: 0, colour: "green-ish" },
  { r: 0, g: 0, b: 255, colour: "blue-ish" },
  { r: 0, g: 0, b: 254, colour: "blue-ish" },
  { r: 0, g: 0, b: 253, colour: "blue-ish" },
]

let r
let g
let b
let rSlider
let gSlider
let bSlider
let nn

function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 255).position(10, 20)
  gSlider = createSlider(0, 255, 0).position(10, 40)
  bSlider = createSlider(0, 255, 0).position(10, 60)
  let options = {
    task: 'classification',
    debug: true
  }
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
  for (let i = 0; i < data.length; i++)
```

```
  {
    let item = data[i]
    let inputs = [item.r, item.g, item.b]
    let outputs = [item.colour]
    nn.addData(inputs, outputs)
  }
  nn.normalizeData()
}

function draw()
{
  r = rSlider.value()
  g = gSlider.value()
  b = bSlider.value()
  background(r, g, b)
}
```

Sketch A11.6 training the neural network

We will now train our neural network, setting the batch size to 16 and the epochs to 128.

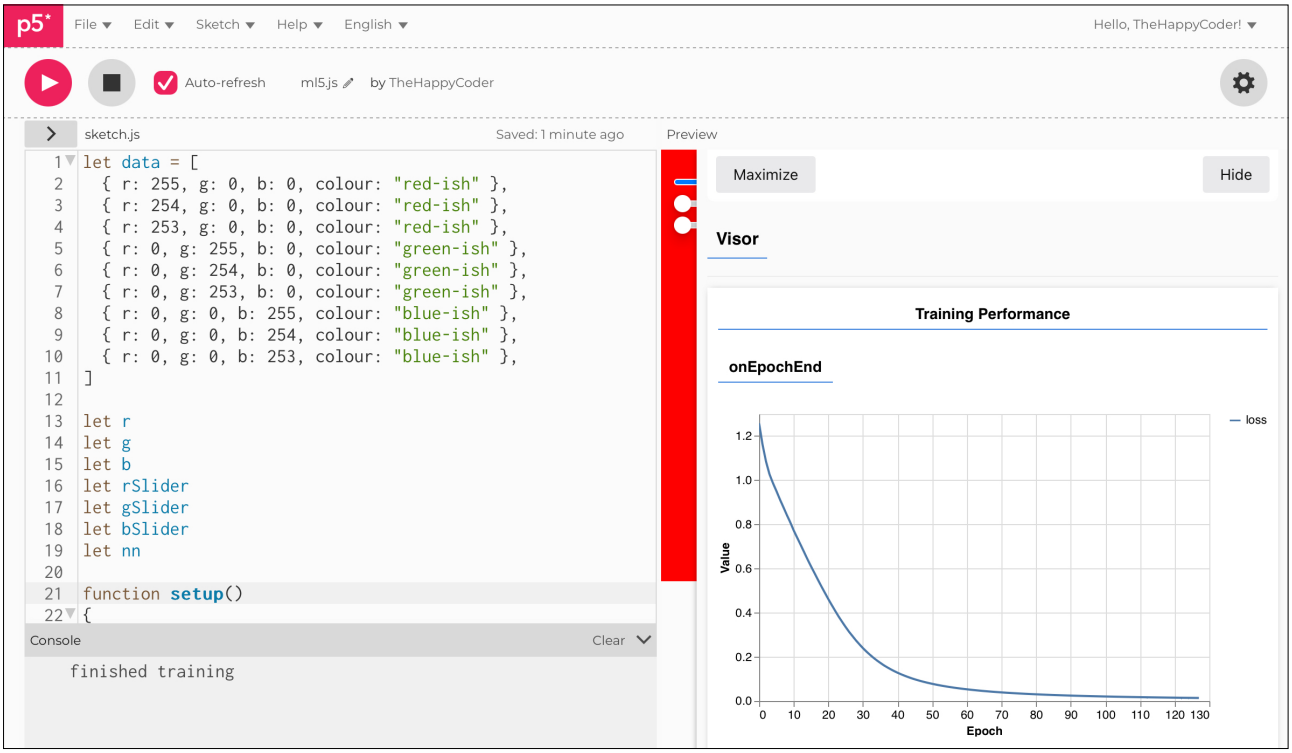
```
let data = [
  { r: 255, g: 0, b: 0, colour: "red-ish" },
  { r: 254, g: 0, b: 0, colour: "red-ish" },
  { r: 253, g: 0, b: 0, colour: "red-ish" },
  { r: 0, g: 255, b: 0, colour: "green-ish" },
  { r: 0, g: 254, b: 0, colour: "green-ish" },
  { r: 0, g: 253, b: 0, colour: "green-ish" },
  { r: 0, g: 0, b: 255, colour: "blue-ish" },
  { r: 0, g: 0, b: 254, colour: "blue-ish" },
  { r: 0, g: 0, b: 253, colour: "blue-ish" },
]

let r
let g
let b
let rSlider
let gSlider
let bSlider
let nn

function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 255).position(10, 20)
  gSlider = createSlider(0, 255, 0).position(10, 40)
  bSlider = createSlider(0, 255, 0).position(10, 60)
  let options = {
    task: 'classification',
    debug: true
  }
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
  for (let i = 0; i < data.length; i++)
```

```
  {
    let item = data[i]
    let inputs = [item.r, item.g, item.b]
    let outputs = [item.colour]
    nn.addData(inputs, outputs)
  }
  nn.normalizeData()
  const trainingOptions = {
    batchSize: 16,
    epochs: 128
  }
  nn.train(trainingOptions, finishedTraining)
}

function finishedTraining()
{
  console.log("finished training")
}

function draw()
{
  r = rSlider.value()
  g = gSlider.value()
  b = bSlider.value()
  background(r, g, b)
}
```

🗒️ Notes

We do get a very nice curved loss function.

Figure A11.6


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 1 minute ago
Preview
let data = [
{ r: 255, g: 0, b: 0, colour: "red-ish" },
{ r: 254, g: 0, b: 0, colour: "red-ish" },
{ r: 253, g: 0, b: 0, colour: "red-ish" },
{ r: 0, g: 255, b: 0, colour: "green-ish" },
{ r: 0, g: 254, b: 0, colour: "green-ish" },
{ r: 0, g: 253, b: 0, colour: "green-ish" },
{ r: 0, g: 0, b: 255, colour: "blue-ish" },
{ r: 0, g: 0, b: 254, colour: "blue-ish" },
{ r: 0, g: 0, b: 253, colour: "blue-ish" },
]
let r
let g
let b
let rSlider
let gSlider
let bSlider
let nn
function setup()
{
Console
Clear
finished training
Maximize
Hide
Visor
Training Performance
onEpochEnd
loss
Value
Epoch

Sketch A11.7 classifying the colour

We can now classify the colour as `red-ish`, `green-ish`, or `blue-ish`. We create a `classify()` function where we input the new slider values and classify the output, which is then sent to another function called `gotResults()`. There we can have a look at the results to understand what data we want. To look inside the data, we `console.log` the results.

```
let data = [
  { r: 255, g: 0, b: 0, colour: "red-ish" },
  { r: 254, g: 0, b: 0, colour: "red-ish" },
  { r: 253, g: 0, b: 0, colour: "red-ish" },
  { r: 0, g: 255, b: 0, colour: "green-ish" },
  { r: 0, g: 254, b: 0, colour: "green-ish" },
  { r: 0, g: 253, b: 0, colour: "green-ish" },
  { r: 0, g: 0, b: 255, colour: "blue-ish" },
  { r: 0, g: 0, b: 254, colour: "blue-ish" },
  { r: 0, g: 0, b: 253, colour: "blue-ish" },
]

let r
let g
let b
let rSlider
let gSlider
let bSlider
let nn

function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 255).position(10, 20)
  gSlider = createSlider(0, 255, 0).position(10, 40)
  bSlider = createSlider(0, 255, 0).position(10, 60)
  let options = {
    task: 'classification',
    debug: true
  }
  ml5.setBackend("webgl")
```

```
  nn = ml5.neuralNetwork(options)
  for (let i = 0; i < data.length; i++)
  {
    let item = data[i]
    let inputs = [item.r, item.g, item.b]
    let outputs = [item.colour]
    nn.addData(inputs, outputs)
  }
  nn.normalizeData()
  const trainingOptions = {
    batchSize: 16,
    epochs: 128
  }
  nn.train(trainingOptions, finishedTraining)
}

function finishedTraining()
{
  console.log("finished training")
  classify()
}

function classify()
{
  const input = [r, g, b]
  nn.classify(input, gotResults)
}

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

function draw()
{
  r = rSlider.value()
  g = gSlider.value()
  b = bSlider.value()
```

```
  background(r, g, b)
}
```

🗒️ Notes

In the console, you get the following (see image below):

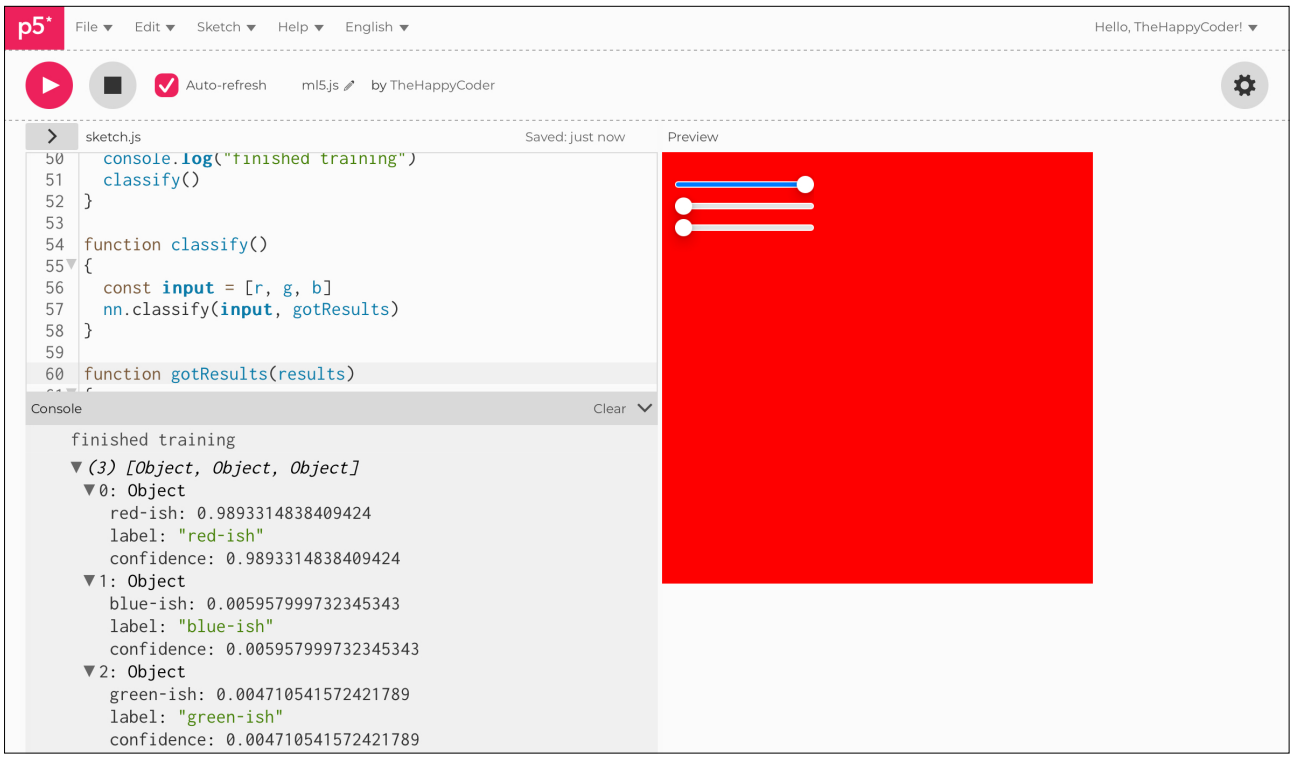
- 🀄 An array with three objects
- 🀄 Each object has a label and confidence score
- 🀄 The highest confidence score is the top one `result[0]`
- 🀄 The next highest is `result[1]`
- 🀄 The third is `result[2]`
- 🀄 We want the top one at index `[0]`

This means we only get the default RGB values; next, we need to get the new values when we move the sliders

🛠️ Code Explanation

`const input = [r, g, b]`	We use the default settings for the initial slider which is red

Figure A11.7



Sketch A11.8 the best prediction

❗ Remove the `console.log()` in the `gotResults()` function as we don't need that information now.

Here we will be extracting the top result (`index[0]`) with the highest confidence score and then printing the result as text on the canvas. We initialise the `label` as a string (`let label = " "`). We add the `classify()` function in the `gotResults()` function to check for updates to the slider, so that every time you move the sliders you get an instant update. We only train it once but classify repeatedly. If that is unclear, work through the logic!

```
let data = [
  { r: 255, g: 0, b: 0, colour: "red-ish" },
  { r: 254, g: 0, b: 0, colour: "red-ish" },
  { r: 253, g: 0, b: 0, colour: "red-ish" },
  { r: 0, g: 255, b: 0, colour: "green-ish" },
  { r: 0, g: 254, b: 0, colour: "green-ish" },
  { r: 0, g: 253, b: 0, colour: "green-ish" },
  { r: 0, g: 0, b: 255, colour: "blue-ish" },
  { r: 0, g: 0, b: 254, colour: "blue-ish" },
  { r: 0, g: 0, b: 253, colour: "blue-ish" },
]


let r
let g
let b
let rSlider
let gSlider
let bSlider
let nn
let label = " "

function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 255).position(10, 20)
  gSlider = createSlider(0, 255, 0).position(10, 40)
  bSlider = createSlider(0, 255, 0).position(10, 60)
```

```
  let options = {
    task: 'classification',
    debug: true
  }
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
  for (let i = 0; i < data.length; i++)
  {
    let item = data[i]
    let inputs = [item.r, item.g, item.b]
    let outputs = [item.colour]
    nn.addData(inputs, outputs)
  }
  nn.normalizeData()
  const trainingOptions = {
    batchSize: 16,
    epochs: 128
  }
  nn.train(trainingOptions, finishedTraining)
}

function finishedTraining()
{
  console.log("finished training")
  classify()
}

function classify()
{
  const input = [r, g, b]
  nn.classify(input, gotResults)
}

function gotResults(results)
{
  // console.log(results)
  label = results[0].label
  classify()
```

```
}

function draw()
{
  r = rSlider.value()
  g = gSlider.value()
  b = bSlider.value()
  background(r, g, b)
  textSize(64)
  text(label, width/4, height/2)
}
```

🗒️ Notes

You will perhaps notice that the results aren't perfect despite the loss function looking pretty good. We have only provided a small amount of data, but also we haven't offered any change to most of the other hyperparameters, which is something we could do.

🌻 Challenges

1. Change the number of hidden layers and nodes,
2. Change the batch size,
3. Change the learning rate

🛠️ Code Explanation

`let label = " "`	Define the label as a string
`label = results[0].label`	Pulling the label from the classified results

Figure A11.8a

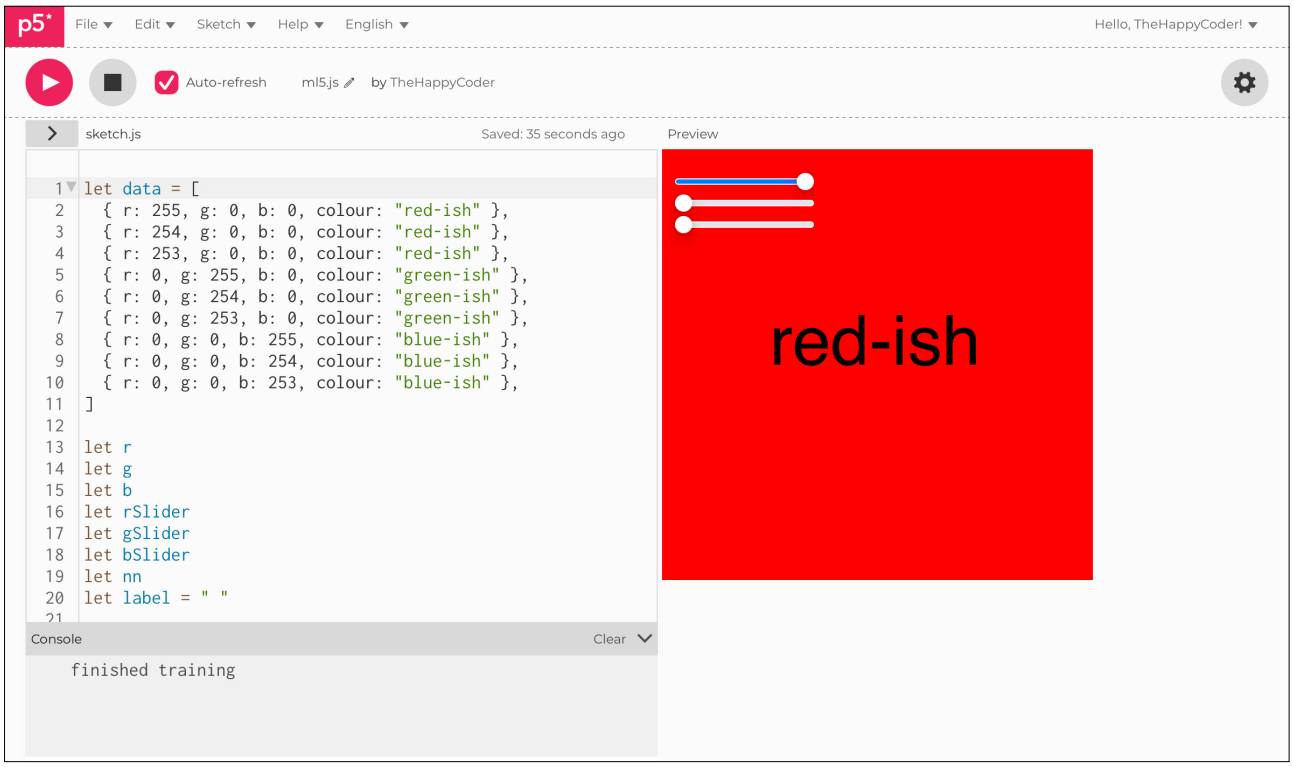


Figure A11.8b

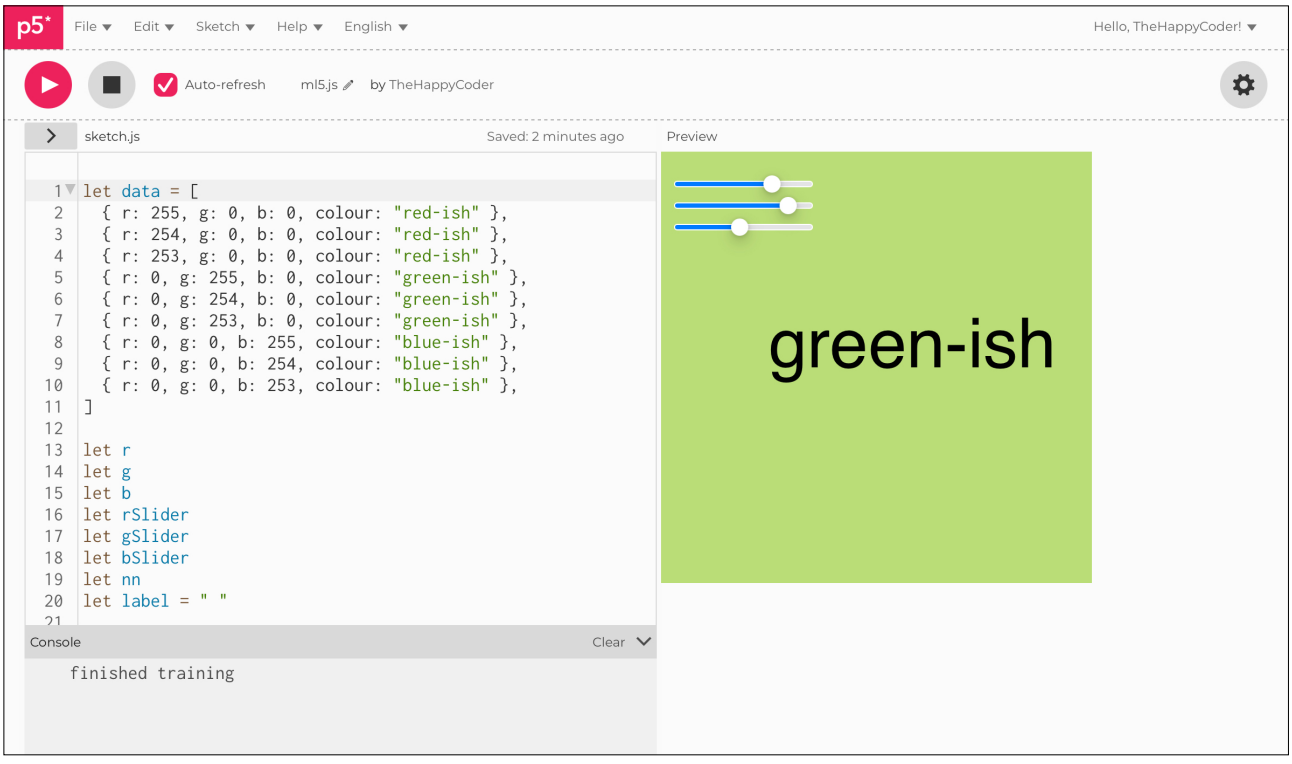


Figure A11.8c