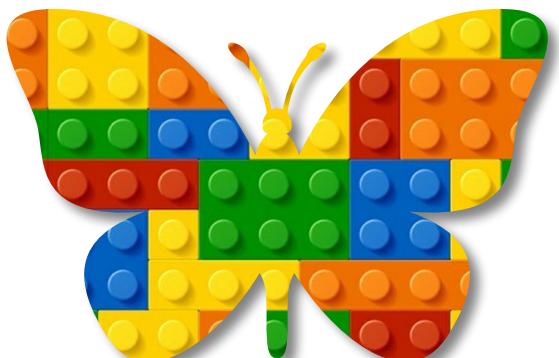


Artificial Intelligence

Module A

Unit #8

colour predictor





Module A Unit #8 colour predictor

Introduction to colour predictor

The index.html file

- Sketch A8.1 creating the data
- Sketch A8.2 sliders
- Sketch A8.3 adding the neural network
- Sketch A8.4 adding the data
- Sketch A8.5 normalising the data
- Sketch A8.6 training the neural network
- Sketch A8.7 classifying the colour
- Sketch A8.8 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 A8.1 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, eg red is **(255, 0, 0)**, **(254, 0, 0)** and **(253, 0, 0)** and do the same for the green and the blue

```
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 A8.2 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.

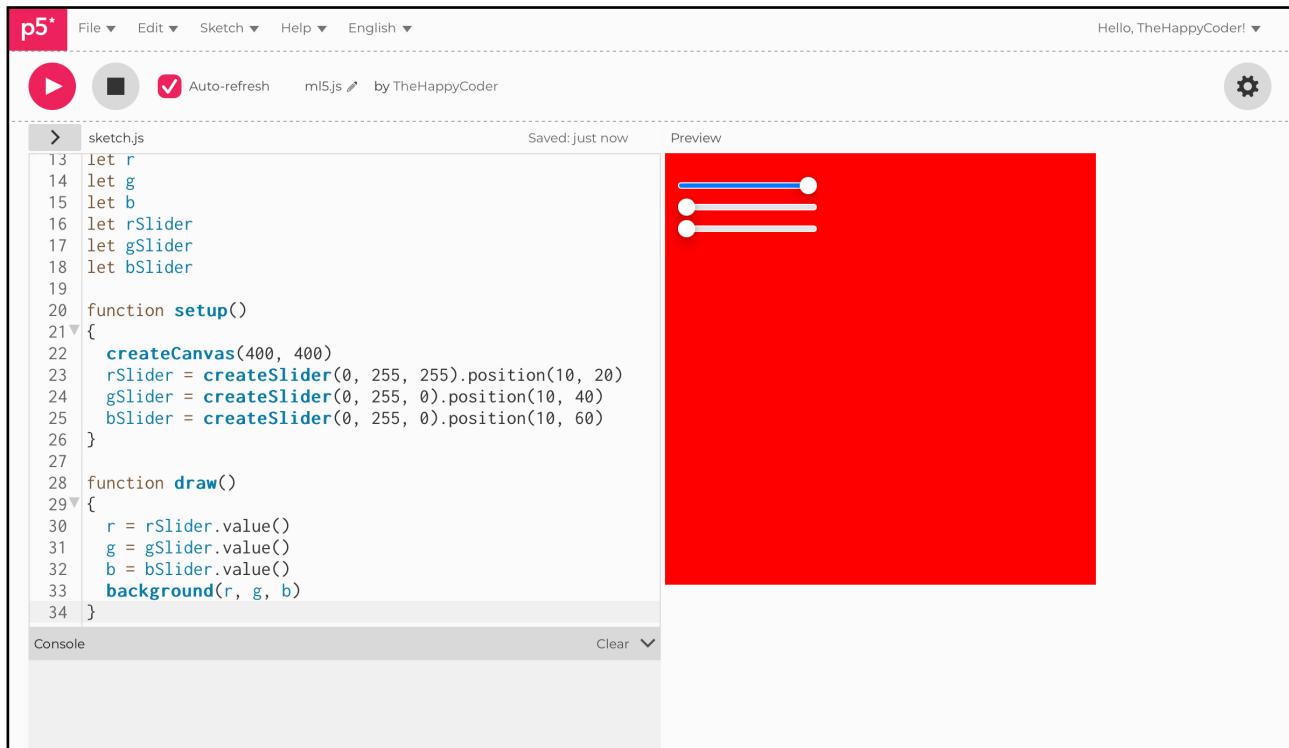
Challenge

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 A8.2



The screenshot shows the p5.js IDE interface. At the top, there's a menu bar with 'File ▾', 'Edit ▾', 'Sketch ▾', 'Help ▾', and 'English ▾'. To the right of the menu is a greeting 'Hello, TheHappyCoder! ▾' and a gear icon for settings. Below the menu, there are buttons for play/pause, stop, and auto-refresh, followed by the file name 'sketch.js' and the author 'by TheHappyCoder'. The status bar indicates the file was 'Saved: just now' and shows 'Preview'. The code editor on the left contains the following P5.js code:

```
13 let r
14 let g
15 let b
16 let rSlider
17 let gSlider
18 let bSlider
19
20 function setup()
21 {
22   createCanvas(400, 400)
23   rSlider = createSlider(0, 255, 255).position(10, 20)
24   gSlider = createSlider(0, 255, 0).position(10, 40)
25   bSlider = createSlider(0, 255, 0).position(10, 60)
26 }
27
28 function draw()
29 {
30   r = rSlider.value()
31   g = gSlider.value()
32   b = bSlider.value()
33   background(r, g, b)
34 }
```

The preview window on the right shows a solid red square. Above the red square, there are three horizontal sliders with black bars and white circular knobs, positioned at approximately y-coordinates 20, 40, and 60.



Sketch A8.3 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 A8.4 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** value 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 reference 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 A8.5 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 A8.6 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")
}

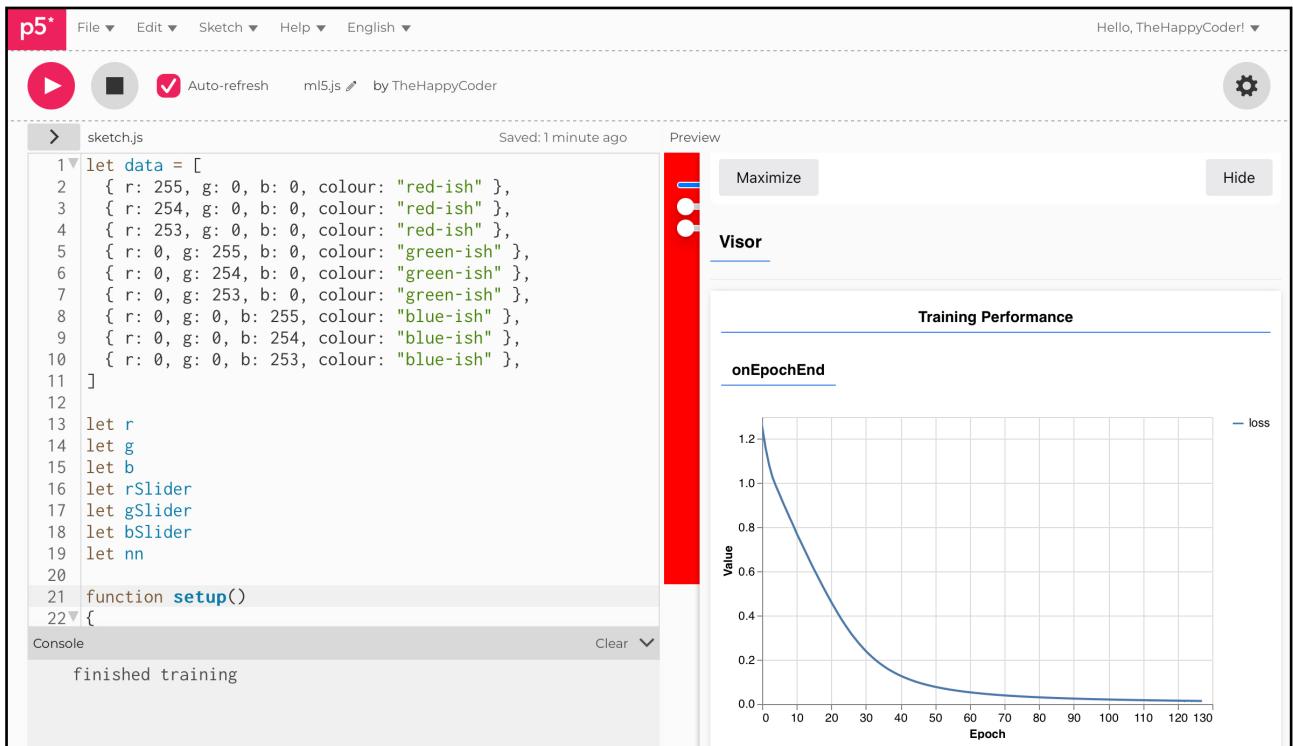
function draw()
{
  r = rSlider.value()
  g = gSlider.value()
  b = bSlider.value()
  background(r, g, b)
}
```



Notes

We do get a very nice curved loss function.

Figure A8.6



The screenshot shows the p5.js IDE interface. On the left, the code editor displays `sketch.js` with the following content:

```
1 let data = [
2   { r: 255, g: 0, b: 0, colour: "red-ish" },
3   { r: 254, g: 0, b: 0, colour: "red-ish" },
4   { r: 253, g: 0, b: 0, colour: "red-ish" },
5   { r: 0, g: 255, b: 0, colour: "green-ish" },
6   { r: 0, g: 254, b: 0, colour: "green-ish" },
7   { r: 0, g: 253, b: 0, colour: "green-ish" },
8   { r: 0, g: 0, b: 255, colour: "blue-ish" },
9   { r: 0, g: 0, b: 254, colour: "blue-ish" },
10  { r: 0, g: 0, b: 253, colour: "blue-ish" },
11 ]
12
13 let r
14 let g
15 let b
16 let rSlider
17 let gSlider
18 let bSlider
19 let nn
20
21 function setup()
22 {
  finished training
}
```

The right side of the interface features a "Visor" panel titled "Training Performance". It contains a line graph titled "onEpochEnd" showing the "loss" value over 130 epochs. The loss starts at approximately 1.2 and decreases rapidly, reaching near zero by epoch 80.

Epoch	Value (loss)
0	1.2
10	0.8
20	0.4
30	0.2
40	0.15
50	0.1
60	0.08
70	0.06
80	0.04
90	0.03
100	0.02
110	0.01
120	0.01
130	0.01



Sketch A8.7 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 A8.7

The screenshot shows the p5.js IDE interface. At the top, there's a toolbar with icons for play, stop, and refresh, followed by the file name "sketch.js", "Auto-refresh" checked, "ml5.js" by TheHappyCoder!, and a user profile "Hello, TheHappyCoder!". Below the toolbar is the code editor with the following content:

```
sketch.js
50 console.log("finished training")
51 classify()
52 }
53
54 function classify()
55{
56   const input = [r, g, b]
57   nn.classify(input, gotResults)
58 }
59
60 function gotResults(results)
61 {
62 }
```

The "Console" tab is selected, displaying the output of the code:

```
finished training
▼ (3) [Object, Object, Object]
  ▼ 0: Object
    red-ish: 0.9893314838409424
    label: "red-ish"
    confidence: 0.9893314838409424
  ▼ 1: Object
    blue-ish: 0.005957999732345343
    label: "blue-ish"
    confidence: 0.005957999732345343
  ▼ 2: Object
    green-ish: 0.004710541572421789
    label: "green-ish"
    confidence: 0.004710541572421789
```

To the right of the code editor is a "Preview" window showing a red background with three horizontal bars: a blue bar at the top, a white bar in the middle, and a black bar at the bottom.



Sketch A8.8 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 A8.8a

The screenshot shows the p5.js IDE interface. The top bar includes 'File ▾', 'Edit ▾', 'Sketch ▾', 'Help ▾', and 'English ▾'. On the right, it says 'Hello, TheHappyCoder! ▾'. The main area has tabs for 'sketch.js' and 'Preview'. The code editor contains the following JavaScript code:

```
1 let data = [
2   { r: 255, g: 0, b: 0, colour: "red-ish" },
3   { r: 254, g: 0, b: 0, colour: "red-ish" },
4   { r: 253, g: 0, b: 0, colour: "red-ish" },
5   { r: 0, g: 255, b: 0, colour: "green-ish" },
6   { r: 0, g: 254, b: 0, colour: "green-ish" },
7   { r: 0, g: 253, b: 0, colour: "green-ish" },
8   { r: 0, g: 0, b: 255, colour: "blue-ish" },
9   { r: 0, g: 0, b: 254, colour: "blue-ish" },
10  { r: 0, g: 0, b: 253, colour: "blue-ish" },
11 ]
12
13 let r
14 let g
15 let b
16 let rSlider
17 let gSlider
18 let bSlider
19 let nn
20 let label = " "
21
```

The 'Console' panel at the bottom shows the message 'finished training'.

The 'Preview' window shows a solid red background. At the top left, there are three horizontal sliders: one blue, one white, and one white. To the right of the sliders, the text 'red-ish' is displayed in a large, black, sans-serif font.

Figure A8.8b

The screenshot shows the p5.js IDE interface. At the top, there are navigation menus: File, Edit, Sketch, Help, and English. On the right side, it says "Hello, TheHappyCoder! ▾". Below the menu bar, there are icons for play, stop, and auto-refresh, followed by the text "ml5.js" and "by TheHappyCoder". To the right of these is a gear icon.

The main area has tabs for "sketch.js" and "Preview". The "sketch.js" tab contains the following code:

```
1 let data = [
2   { r: 255, g: 0, b: 0, colour: "red-ish" },
3   { r: 254, g: 0, b: 0, colour: "red-ish" },
4   { r: 253, g: 0, b: 0, colour: "red-ish" },
5   { r: 0, g: 255, b: 0, colour: "green-ish" },
6   { r: 0, g: 254, b: 0, colour: "green-ish" },
7   { r: 0, g: 253, b: 0, colour: "green-ish" },
8   { r: 0, g: 0, b: 255, colour: "blue-ish" },
9   { r: 0, g: 0, b: 254, colour: "blue-ish" },
10  { r: 0, g: 0, b: 253, colour: "blue-ish" },
11 ]
12
13 let r
14 let g
15 let b
16 let rSlider
17 let gSlider
18 let bSlider
19 let nn
20 let label = " "
21
```

The "Preview" tab shows a green square with the text "green-ish" in black. Above the text are three sliders: one for red (value ~254), one for green (value ~253), and one for blue (value ~0). The "Console" tab at the bottom shows the message "finished training".

Figure A8.8c

The screenshot shows the p5.js IDE interface. The top bar includes the p5 logo, file navigation, and a user profile. The main area has tabs for 'sketch.js' and 'Preview'. The code editor contains the following JavaScript code:

```
1 let data = [
2   { r: 255, g: 0, b: 0, colour: "red-ish" },
3   { r: 254, g: 0, b: 0, colour: "red-ish" },
4   { r: 253, g: 0, b: 0, colour: "red-ish" },
5   { r: 0, g: 255, b: 0, colour: "green-ish" },
6   { r: 0, g: 254, b: 0, colour: "green-ish" },
7   { r: 0, g: 253, b: 0, colour: "green-ish" },
8   { r: 0, g: 0, b: 255, colour: "blue-ish" },
9   { r: 0, g: 0, b: 254, colour: "blue-ish" },
10  { r: 0, g: 0, b: 253, colour: "blue-ish" },
11 ]
12
13 let r
14 let g
15 let b
16 let rSlider
17 let gSlider
18 let bSlider
19 let nn
20 let label = " "
21
```

The 'Console' tab shows the message 'finished training'. The 'Preview' window shows a purple background with three sliders at their maximum values (255) and the text 'blue-ish'.