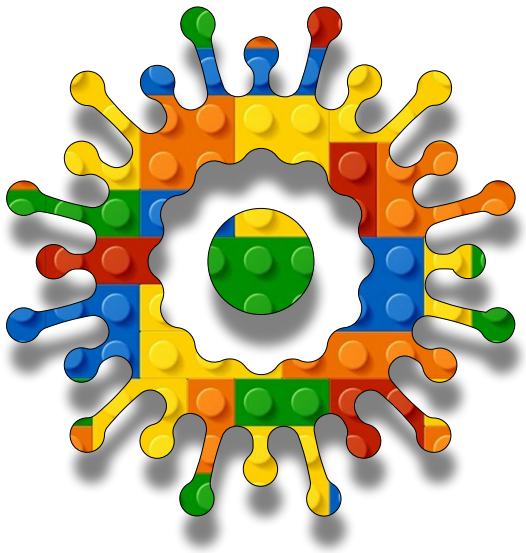
Algorithmic Intelligence Module A Unit #12 pixel predictor



Content

Module A Unit #12 pixel predictor

Introduction to pixel predictor

We will use a webcam from your device. If you don't have one, then you either miss this unit out or attach one manually. The image will be reduced, flipped (mirrored), and then pixelated; this enables the model to be trained on a dataset that isn't too large.

Once it is trained, you will be able to move a circle across the canvas through moving your head from side to side, alternatively holding your left or right hand up. It will use images of you in certain positions and use this data to train the model. The model is pretty similar to before.

index.html

Just a reminder that your index.html file needs the ml5.js added.

```
<!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 A12.1 starting sketch

Give p5.js permission to access your webcam. What you should get instead of the canvas is a video of yourself. It will be reversed (not mirrored).

```
let video

function setup()
{
  noCanvas()
  video = createCapture(VIDEO)
}

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

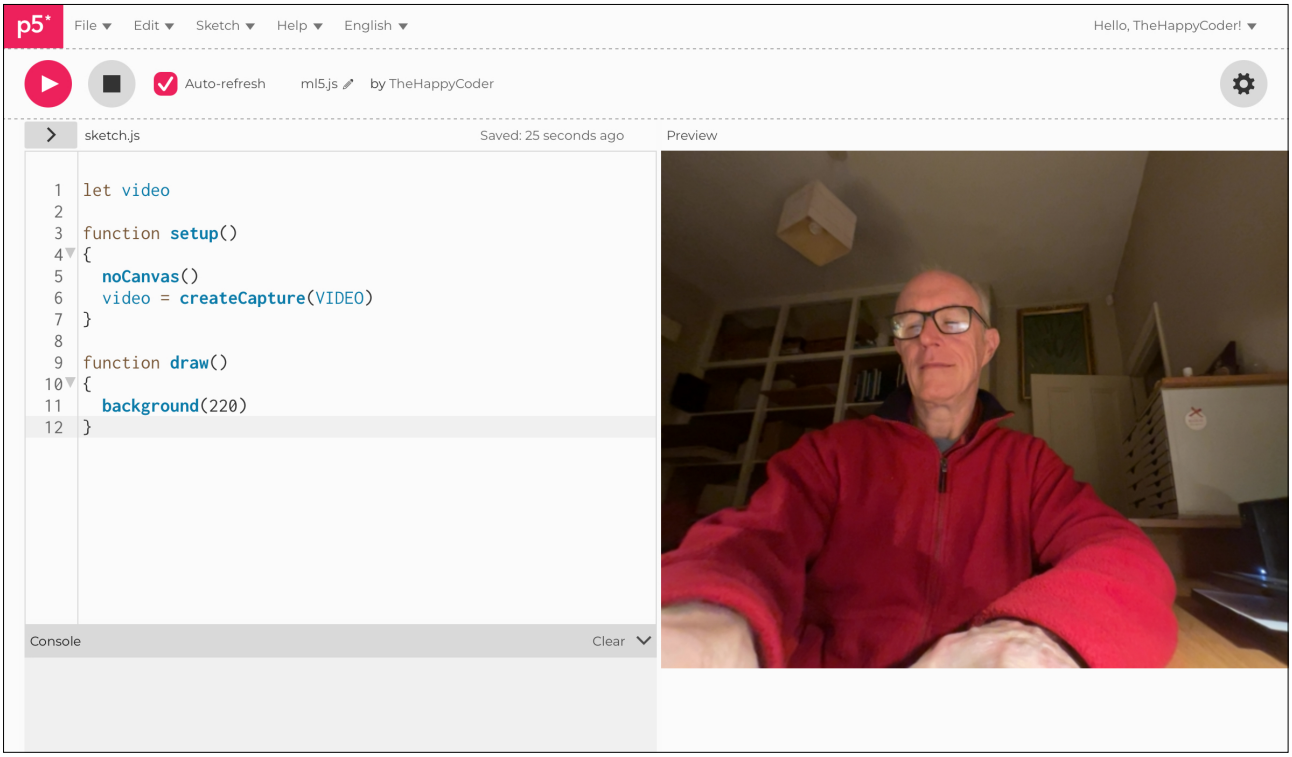
Notes

You should have a nice video image of yourself; the dimensions should be 640 x 480.

Code Explanation

`noCanvas()`	We tell the setup() function we don't want a canvas
`video = createCapture(VIDEO)`	We do want a video from the webcam

Figure A12.1


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 25 seconds ago
Preview
let video
function setup()
{
noCanvas()
video = createCapture(VIDEO)
}
function draw()
{
background(220)
}
Console
Clear

Sketch A12.2 hiding the video

To use the video image, we want it on the canvas. So replace `noCanvas()` with `createCanvas(640, 480)` and hide the video in `setup()`. This will place the video on the canvas.

```
let video

function setup()
{
  createCanvas(640, 480)
  video = createCapture(VIDEO)
  video.hide()
}

function draw()
{
  background(220)
  image(video, 0, 0)
}
```

🗒️ Notes

You should have one video image exactly as before.

🌻 Challenges

1. Comment out `// video.hide()`
2. Change the size of the canvas to see that you have placed the video on the canvas, hence the dimensions.
3. Change `image(video, 0, 0)` to `image(video, 100, 100)`.

🛠️ Code Explanation

`video.hide()`	Hides the video (but not the one drawn on the canvas)
`image(video, 0, 0)`	This places the image, in this case a video at origin (0, 0)

Sketch A12.3 mirrored

We want to flip the video so that it mirrors our movements; it makes it much more intuitive when we come to train the model.

```
let video

function setup()
{
  createCanvas(640, 480)
  video = createCapture(VIDEO, {flipped: true})
  video.hide()
}

function draw()
{
  background(220)
  image(video, 0, 0)
}
```

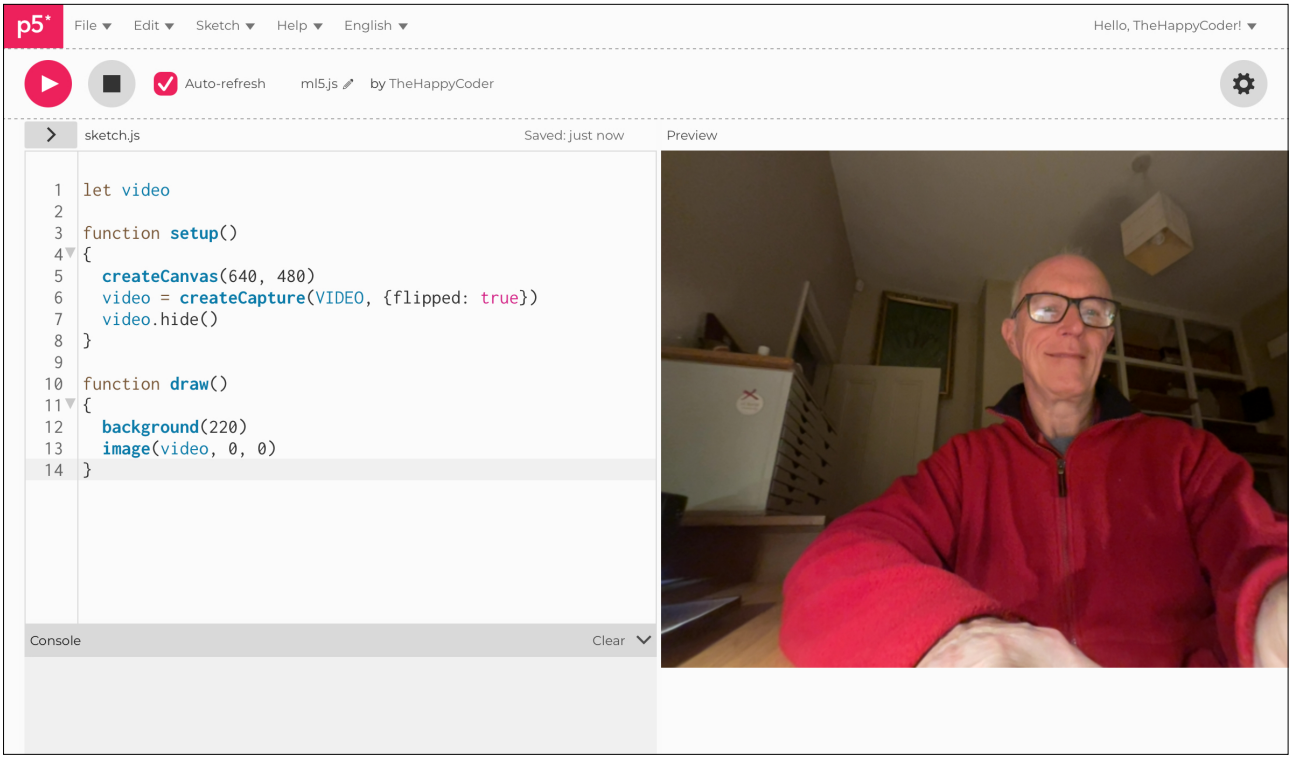
🗒️ Notes

Now you have a mirrored image from the webcam.

🛠️ Code Explanation

`{flipped: true}`	Flips the video so that it behaves like a mirror

Figure A12.3 mirrored



Sketch A12.4 video ready

Now we want to draw it on the canvas in pixels. However, we want to double-check that the video is working before we do anything else. To do that, we create a boolean variable called `ready` that is either `true` or `false`. The default is `false`, and when the video is ready, we have a callback function called `videoReady()` and set it to `true`. Then, in `draw()`, we display the video image only when `ready` returns `true`.

```
let video
let ready = false

function setup()
{
  createCanvas(640, 480)
  video = createCapture(VIDEO, {flipped: true}, videoReady)
  video.hide()
}

function videoReady()
{
  ready = true
}

function draw()
{
  background(220)
  if (ready)
  {
    image(video, 0, 0)
  }
}
```

🗒️ Notes

Everything should work just as before. This is an important step before we start adding data to the neural network.

🛠️ Code Explanation

`video = createCapture(VIDEO, {flipped: true}, videoReady)`	The videoReady is the callback function
`ready = true`	Returns boolean true when ready
`if (ready)`	Check to see that the video is streaming before displaying it

Sketch A12.5 the size of the video

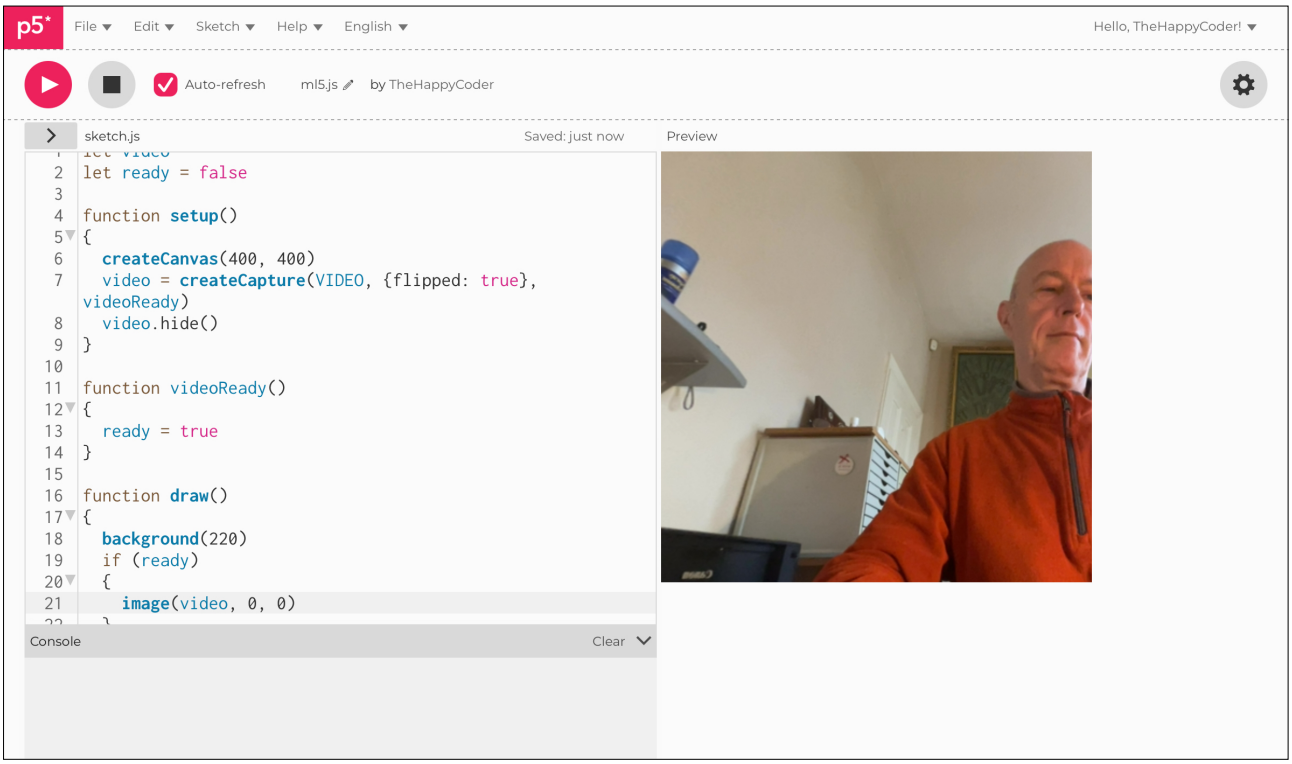
At present, we have an image coming in at 640 x 480 pixels. You are drawing that onto the canvas of the same dimensions (and ratio). If we change the canvas to 400 x 400, we crop the image, losing some of it. We want to use a square image, and then we can simplify the pixelation later. First, let's change the canvas size.

```
let video
let ready = false

function setup()
{
  createCanvas(400, 400)
  video = createCapture(VIDEO, {flipped: true}, videoReady)
  video.hide()
}

function videoReady()
{
  ready = true
}

function draw()
{
  background(220)
  if (ready)
  {
    image(video, 0, 0)
  }
}
```

Notes

We only get part of the image from the camera. We want the full image, even if it is distorted and squashed into a square.

Figure A12.5


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: just now
Preview
let ready = false
function setup()
{
createCanvas(400, 400)
video = createCapture(VIDEO, {flipped: true},
videoReady)
video.hide()
}
function videoReady()
{
ready = true
}
function draw()
{
background(220)
if (ready)
{
image(video, 0, 0)
Console
Clear

Sketch A12.6 resizing the video

We can squash the video to a new size. This will distort the video, but isn't a problem for this exercise. In fact, we will be squashing it a whole lot more. We create a variable called `videoSize` and use the function `video.size()` to change the dimensions. We will set the dimensions to `400` for now.

```
let video
let ready = false
let videoSize = 400

function setup()
{
  createCanvas(400, 400)
  video = createCapture(VIDEO, {flipped: true}, videoReady)
  video.hide()
  video.size(videoSize, videoSize)
}

function videoReady()
{
  ready = true
}

function draw()
{
  background(220)
  if (ready)
  {
    image(video, 0, 0)
  }
}
```

Notes

The video is now fully `400` x `400`, although it is much squashier. Using the same dimensions for the width and height (`videoSize`) is just for convenience. You could have different values. Using a variable means you can alter and change it at any time.

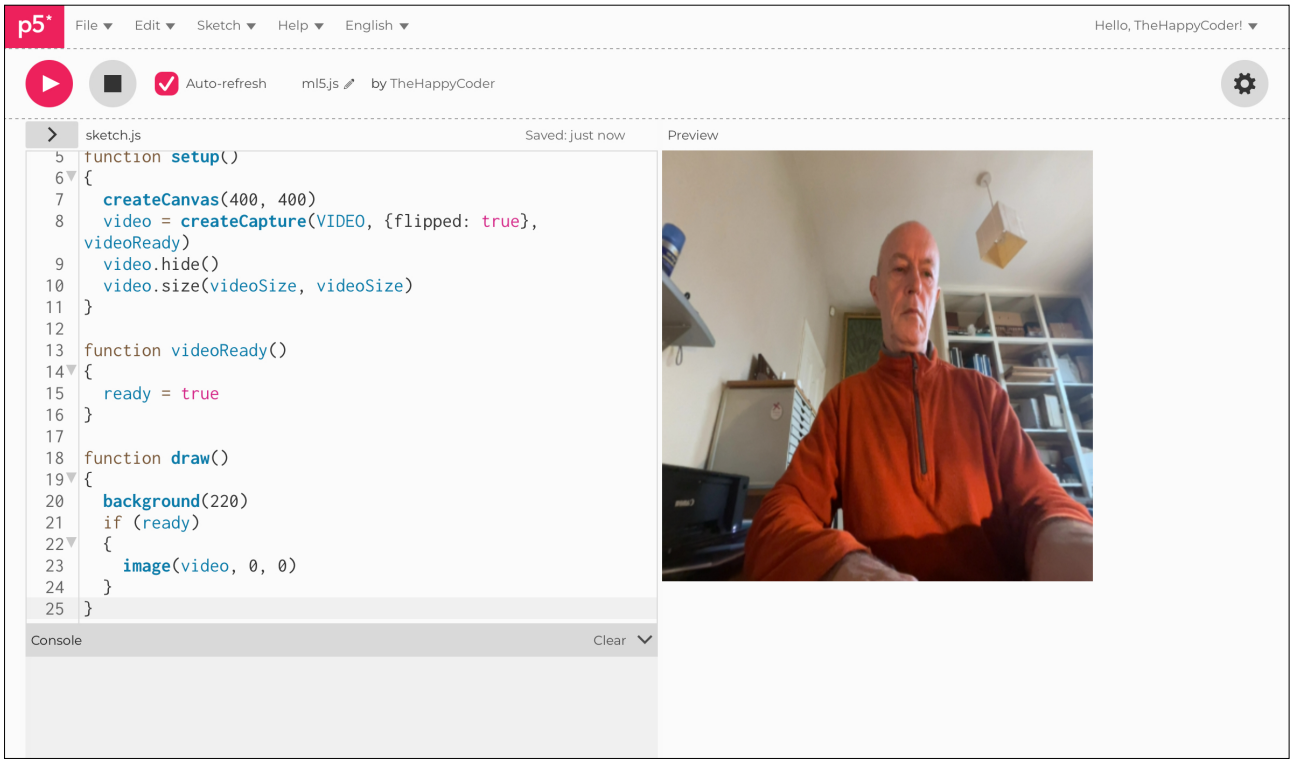
🌻 Challenge

1. Try other dimensions
2. Try 640 x 480

🛠️ Code Explanation

`let videoSize = 400`	Set the dimensions of the video
`video.size(videoSize, videoSize)`	Specifies the dimensions of the video not the canvas drawing it onto

Figure A12.6



Sketch A12.7 even smaller

What we want to do is change the size to 10 x 10, yes, that small! This means the software behind producing the video is going to try and create a 10 x 10 video image from a 640 x 480 video image. We reduce the videoSize to 10.

```
let video
let ready = false
let videoSize = 10

function setup()
{
  createCanvas(400, 400)
  video = createCapture(VIDEO, {flipped: true}, videoReady)
  video.hide()
  video.size(videoSize, videoSize)
}

function videoReady()
{
  ready = true
}

function draw()
{
  background(220)
  if (ready)
  {
    image(video, 0, 0)
  }
}
```

Notes

The video is now really small, almost too small to see; it is now 10 pixels by 10 pixels. The second figure, A9.7b, below shows what it does look like close up. You will notice that it seems grainy and indistinct. This is because it is trying to approximate the larger image.

Challenge

Try something a little bigger, say 50 pixels.

Figure A12.7a can you see it?

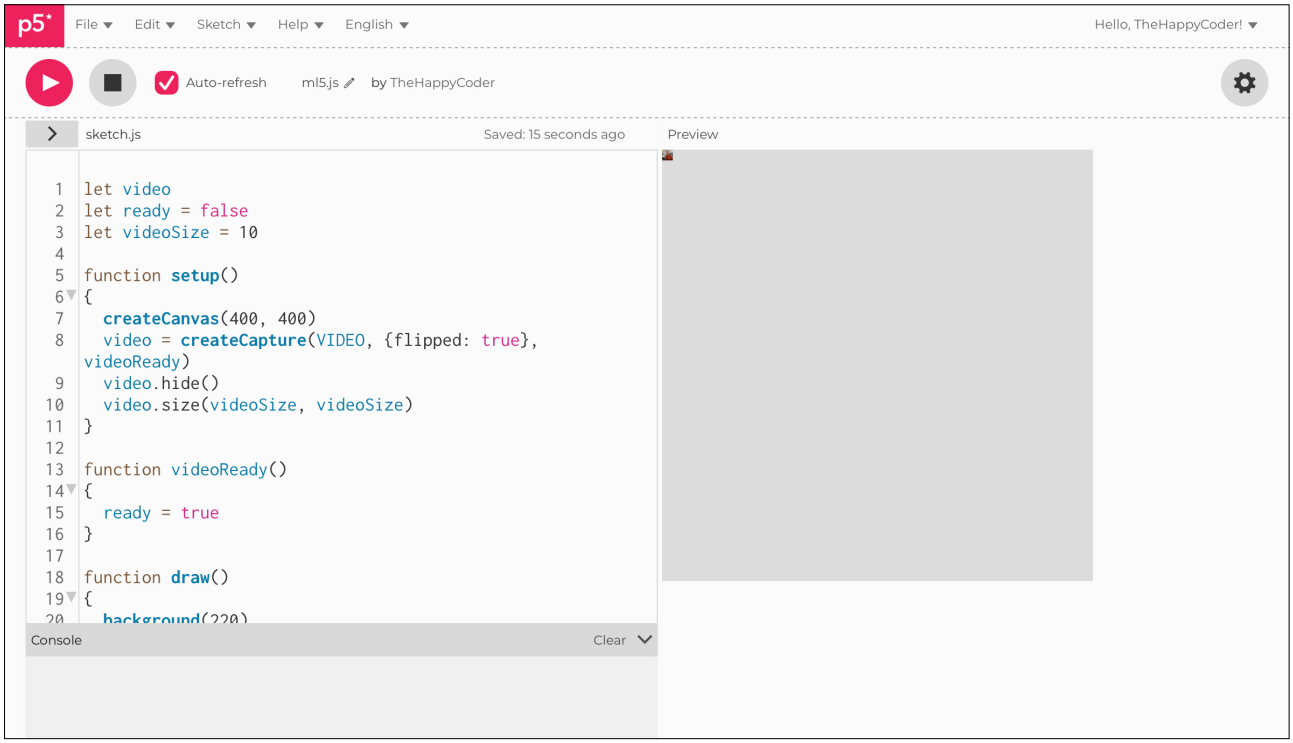
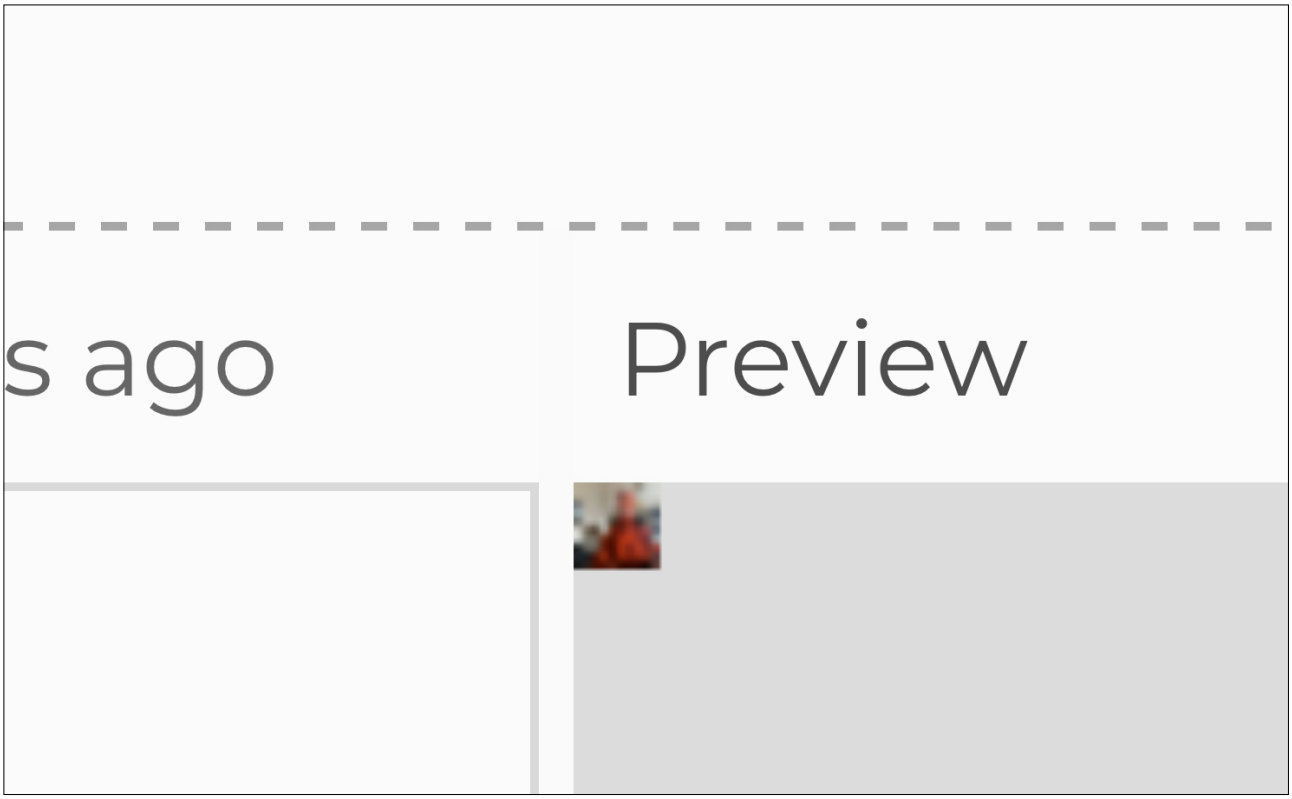


Figure A12.7b yes, it really is me!



Sketch A12.8 loading the pixels

We want to take those pixels which equate to 100 data points (10 by 10), except that there are four channels for each pixel: the red, green, blue, and alpha (transparency), which means we actually have 400 data points, even though we will ignore the alpha.

The function video.loadPixels() does just that; it loads the video into a pixel array which we will access through a nested loop. Remove the image(video, 0, 0) and replace it with the loadPixels(). A full explanation of how we derive the nested loop for getting the values of the pixels in an image is covered in the coding snippets unit. Check back there if you are unsure.

```
let video
let ready = false
let videoSize = 10

function setup()
{
  createCanvas(400, 400)
  video = createCapture(VIDEO, {flipped: true}, videoReady)
  video.hide()
  video.size(videoSize, videoSize)
}

function videoReady()
{
  ready = true
}

function draw()
{
  background(220)
  if (ready)
  {
    video.loadPixels()
    for (let x = 0; x < video.width; x++)
    {
      for (let y = 0; y < video.height; y++)
      {
```

```
        let index = (x + y * video.width) * 4
        let r = video.pixels[index + 0]
        let g = video.pixels[index + 1]
        let b = video.pixels[index + 2]
      }
    }
  }
}
```

🗒️ Notes

This means we have, currently, no video to display; you get a blank canvas. The nested loop takes the r (red) value from the pixel, the g (green) value from that pixel, and the b (blue) value of the pixel from the pixel array. What we can now do is scale or magnify each pixel and colour in a square with those pixel values. In short, each pixel has four elements of the pixel in the array; that is why we multiply by four on each loop, the fourth being the alpha value (which we will not bother to use).

🛠️ Code Explanation

`video.loadPixels()`	Loads all the pixels (with four elements per pixel) into an array
`for (let x = 0; x < video.width; x++)`	Goes through all 10 values of x
`for (let y = 0; y < video.height; y++)`	For each value of x work through each 10 values of y
`let index = (x + y * video.width) * 4`	Each index starting from the top left hand corner, going across to the bottom right hand corner
`let r = video.pixels[index + 0]`	This is the index for the first element (red) of the pixel colour
`let g = video.pixels[index + 1]`	This is the index for the second element (green) of the pixel colour
`let b = video.pixels[index + 2]`	This is the index for the third element (blue) of the pixel colour

Sketch A12.9 enlarged pixels

We are going to draw 10 by 10 squares on the canvas. Each square will be given the r, g, b values for the corresponding pixel in the 10 x 10 pixel video image. If you remember, it was a bit grainy and blurry; we are effectively going to scale up that tiny image to fill the canvas. The width of the square (w) is simply the width of the canvas divided by the videoSize. This means if you want to make changes to either of those variables, they will calculate them rather than hard-coding the values.

```
let video
let ready = false
let videoSize = 10

function setup()
{
  createCanvas(400, 400)
  video = createCapture(VIDEO, {flipped: true}, videoReady)
  video.hide()
  video.size(videoSize, videoSize)
}

function videoReady()
{
  ready = true
}

function draw()
{
  background(220)
  if (ready)
  {
    let w = width/videoSize
    video.loadPixels()
    for (let x = 0; x < video.width; x++)
    {
      for (let y = 0; y < video.height; y++)
      {
        let index = (x + y * video.width) * 4
        let r = video.pixels[index + 0]
```

```
        let g = video.pixels[index + 1]
        let b = video.pixels[index + 2]
        noStroke()
        fill(r, g, b)
        square(x * w, y * w, w)
      }
    }
  }
}
```

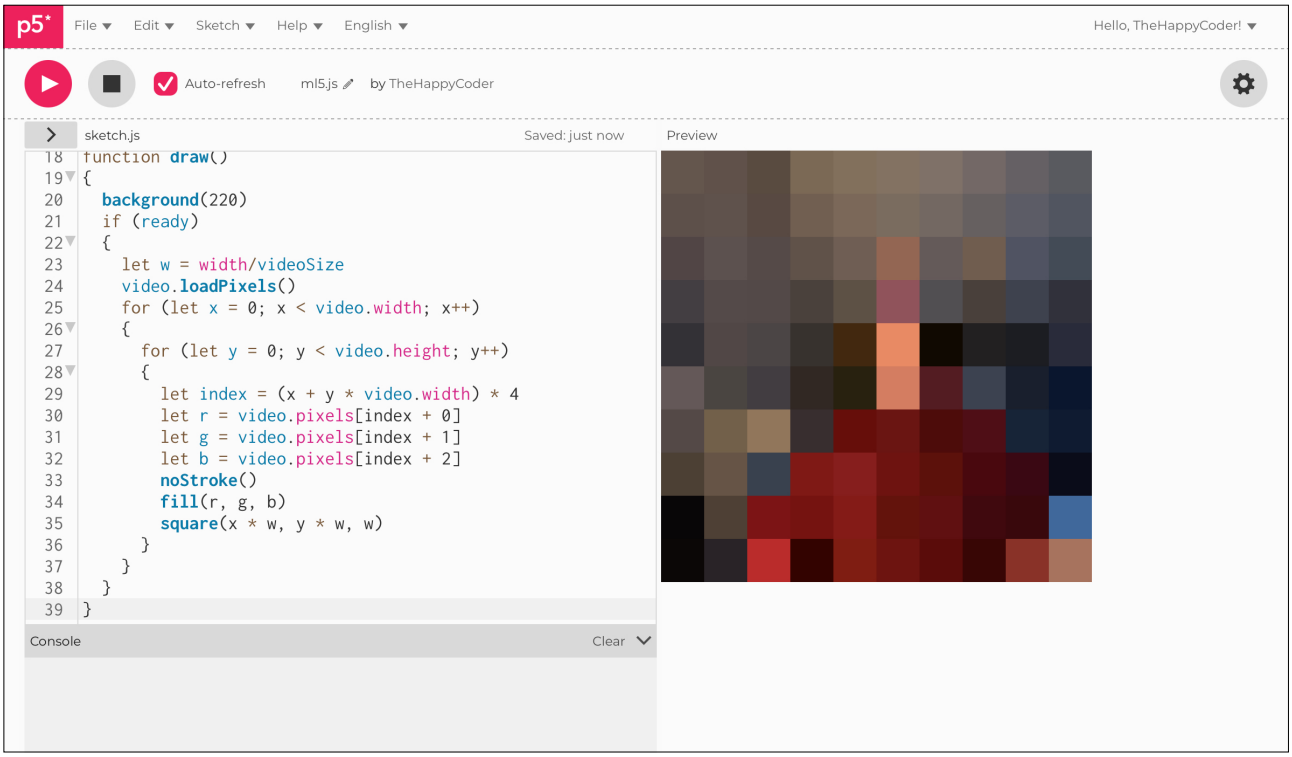
🗒️ Notes

We now have a larger version of that tiny image of 10 x 10 pixels on a canvas of 10 by 10 squares.

🛠️ Code Explanation

`square(x * w, y * w, w)`	Calculating the position and size of each square

Figure A12.9 it really is me



Sketch A12.10 neural network

Now we can add our neural network. The inputs are all the pixels (`totalPixels`) in the image of `videoSize` (`10`), and each pixel has an `RGB` value, hence `videoSize x videoSize x 3`. We have all the pixels as the inputs, hence why we want a very small image. For the output, we have one output (this will become clear later). This will include a learning rate lower than the default. The task is `regression` because the output will be a single value. We will also have the loss function charted.

```
let nn
let video
let ready = false
let videoSize = 10

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  video = createCapture(VIDEO, {flipped: true}, videoReady)
  video.hide()
  video.size(videoSize, videoSize)
  let totalPixels = videoSize * videoSize * 3
  const options = {
    inputs: totalPixels,
    outputs: 1,
    learningRate: 0.01,
    task: 'regression',
    debug: true
  }
  nn = ml5.neuralNetwork(options)
}

function videoReady()
{
  ready = true
}

function draw()
{
```

```
  background(220)
  if (ready)
  {
    let w = width/videoSize
    video.loadPixels()
    for (let x = 0; x < video.width; x++)
    {
      for (let y = 0; y < video.height; y++)
      {
        let index = (x + y * video.width) * 4
        let r = video.pixels[index + 0]
        let g = video.pixels[index + 1]
        let b = video.pixels[index + 2]
        noStroke()
        fill(r, g, b)
        square(x * w, y * w, w)
      }
    }
  }
}
```

🗒️ Notes

The effect hasn't changed, but we have the main elements in place to run this example.

Sketch A12.11 adding the slider output

We want to create a circle which we can control by moving our head from side to side or left hand to right hand. We will have the position of your head (or hand) as the input pixels and the position of the circle as the output. We use a slider to position our circle as we train it. We move the circle to the left and move our head to the left and then save the data/image pixels (several times), repeat by moving the circle to the right and our head to the right (read left hand right hand). Again, repeat saving several examples.

```
let nn
let video
let ready = false
let videoSize = 10
let sliderPos = 0
let slider

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  slider = createSlider(0, width, 0)
  video = createCapture(VIDEO, {flipped: true}, videoReady)
  video.hide()
  video.size(videoSize, videoSize)
  let totalPixels = videoSize * videoSize * 3
  const options = {
    inputs: totalPixels,
    outputs: 1,
    learningRate: 0.01,
    task: 'regression',
    debug: true
  }
  nn = ml5.neuralNetwork(options)
}

function videoReady()
{
  ready = true
```

```
}

function draw()
{
  background(220)
  if (ready)
  {
    let w = width/videoSize
    video.loadPixels()
    for (let x = 0; x < video.width; x++)
    {
      for (let y = 0; y < video.height; y++)
      {
        let index = (x + y * video.width) * 4
        let r = video.pixels[index + 0]
        let g = video.pixels[index + 1]
        let b = video.pixels[index + 2]
        noStroke()
        fill(r, g, b)
        square(x * w, y * w, w)
      }
    }
    sliderPos = slider.value()
    fill(0, 0, 255)
    circle(sliderPos, height/2, 50)
  }
}
```

🗒️ Notes

We draw the circle on the canvas; we also initialise it to zero, which is the left-hand side, obviously. Note we are only moving the blue circle along the x-axis; the y-value is fixed, hence one value is output.

🛠️ Code Explanation

`let sliderPos = 0`	The position of the slider
`let slider`	Naming the slider
`slider = createSlider(0, width, 0)`	Creating the slider
`sliderPos = slider.value()`	Gives you the value of the slider position

Figure A12.11a: slider to the left

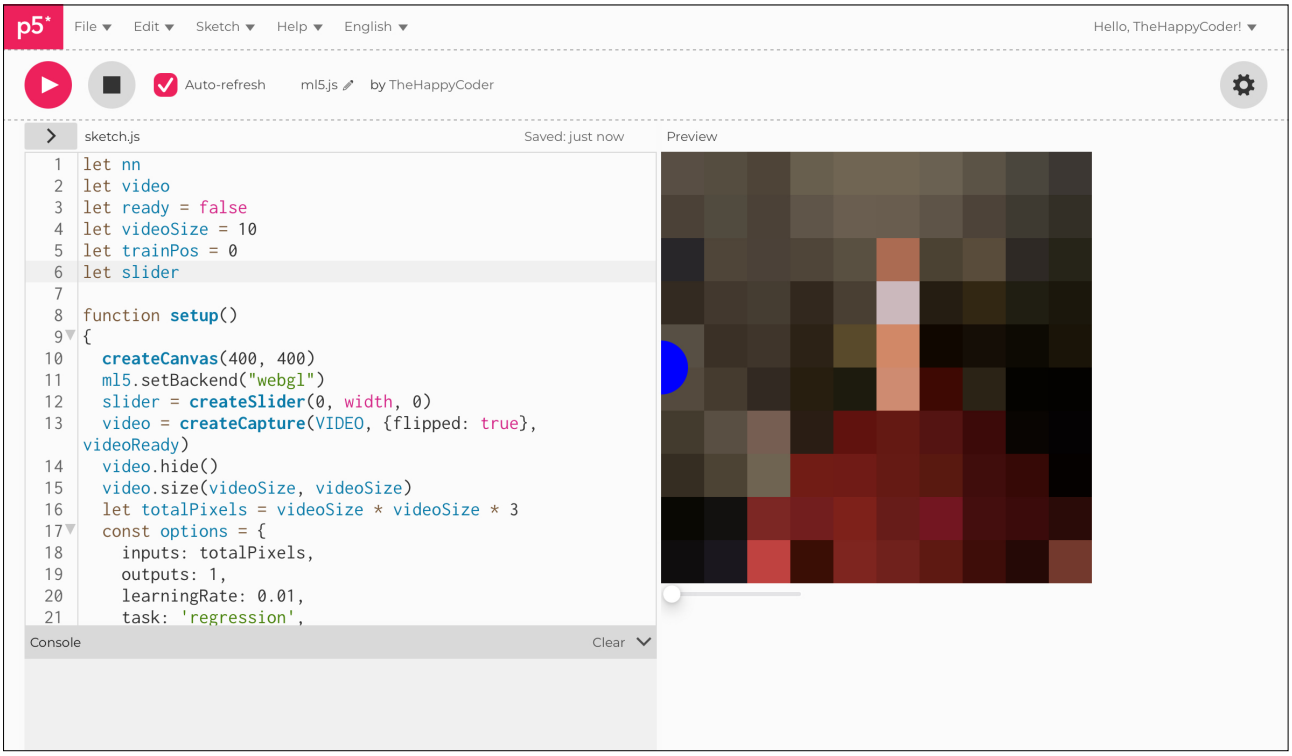
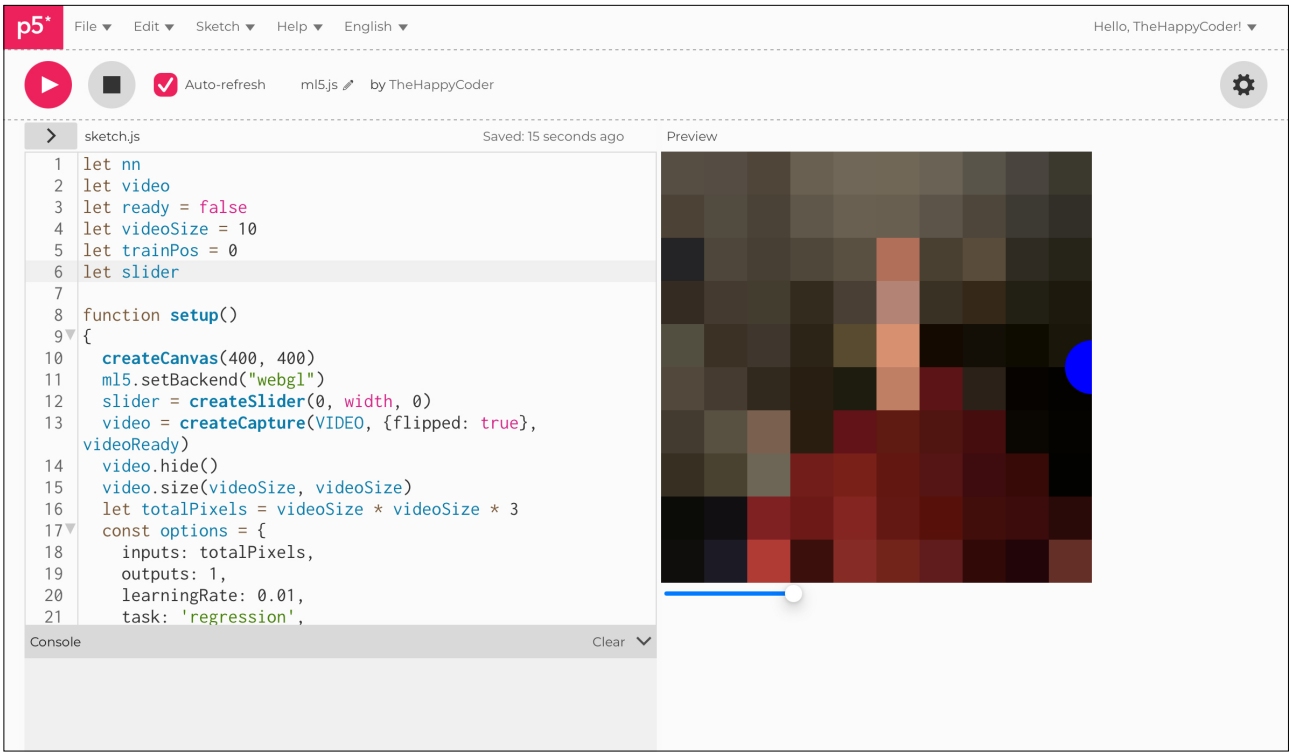


Figure A12.11b: slider to the right



Sketch A12.12 adding the buttons

We need a button so we can add the examples.

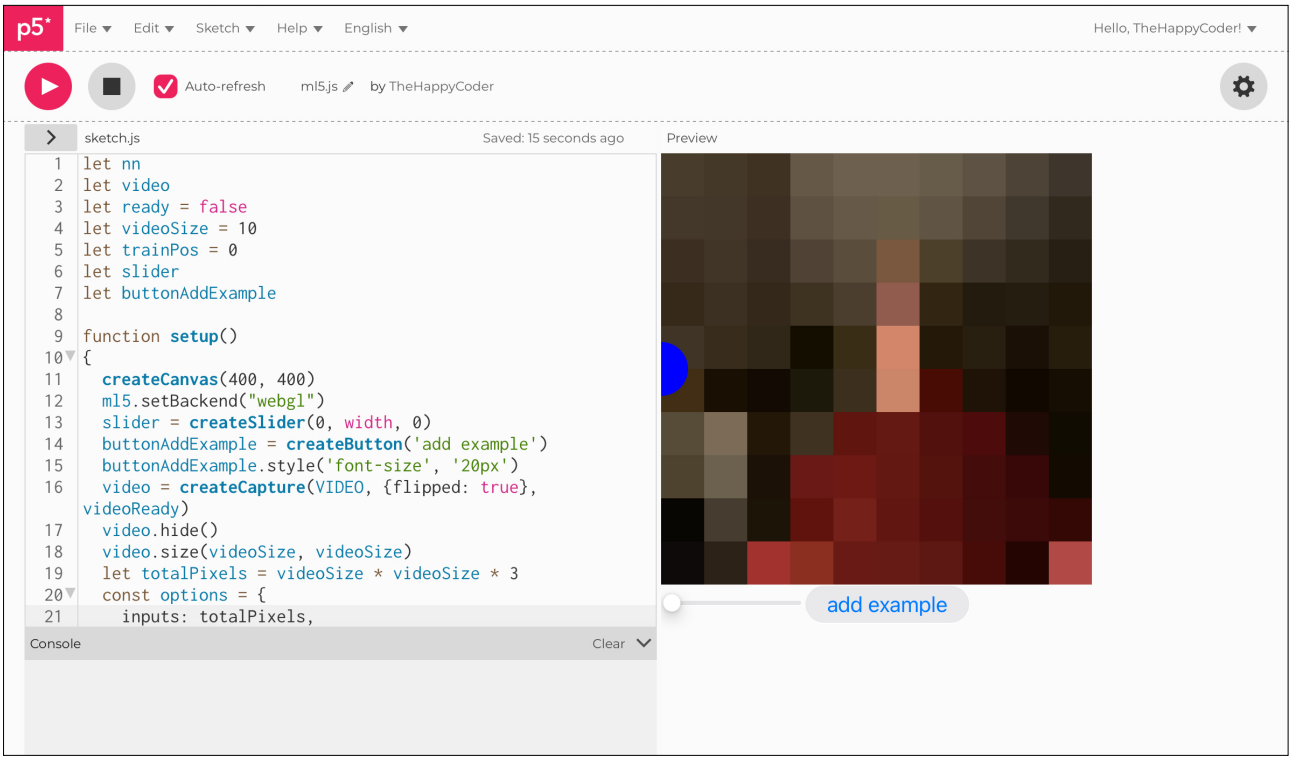
```
let nn
let video
let ready = false
let videoSize = 10
let sliderPos = 0
let slider
let buttonAddExample

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  slider = createSlider(0, width, 0)
  buttonAddExample = createButton('add example')
  buttonAddExample.style('font-size', '20px')
  video = createCapture(VIDEO, {flipped: true}, videoReady)
  video.hide()
  video.size(videoSize, videoSize)
  let totalPixels = videoSize * videoSize * 3
  const options = {
    inputs: totalPixels,
    outputs: 1,
    learningRate: 0.01,
    task: 'regression',
    debug: true
  }
  nn = ml5.neuralNetwork(options)
}

function videoReady()
{
  ready = true
}
```

```
function draw()
{
  background(220)
  if (ready)
  {
    let w = width/videoSize
    video.loadPixels()
    for (let x = 0; x < video.width; x++)
    {
      for (let y = 0; y < video.height; y++)
      {
        let index = (x + y * video.width) * 4
        let r = video.pixels[index + 0]
        let g = video.pixels[index + 1]
        let b = video.pixels[index + 2]
        noStroke()
        fill(r, g, b)
        square(x * w, y * w, w)
      }
    }
    sliderPos = slider.value()
    fill(0, 0, 255)
    circle(sliderPos, height/2, 50)
  }
}
```

🗒️ Notes

No action from the button yet.

Figure A12.12



Sketch A12.13 the training button

While we are in the business of adding buttons, we will create a training button to use later.

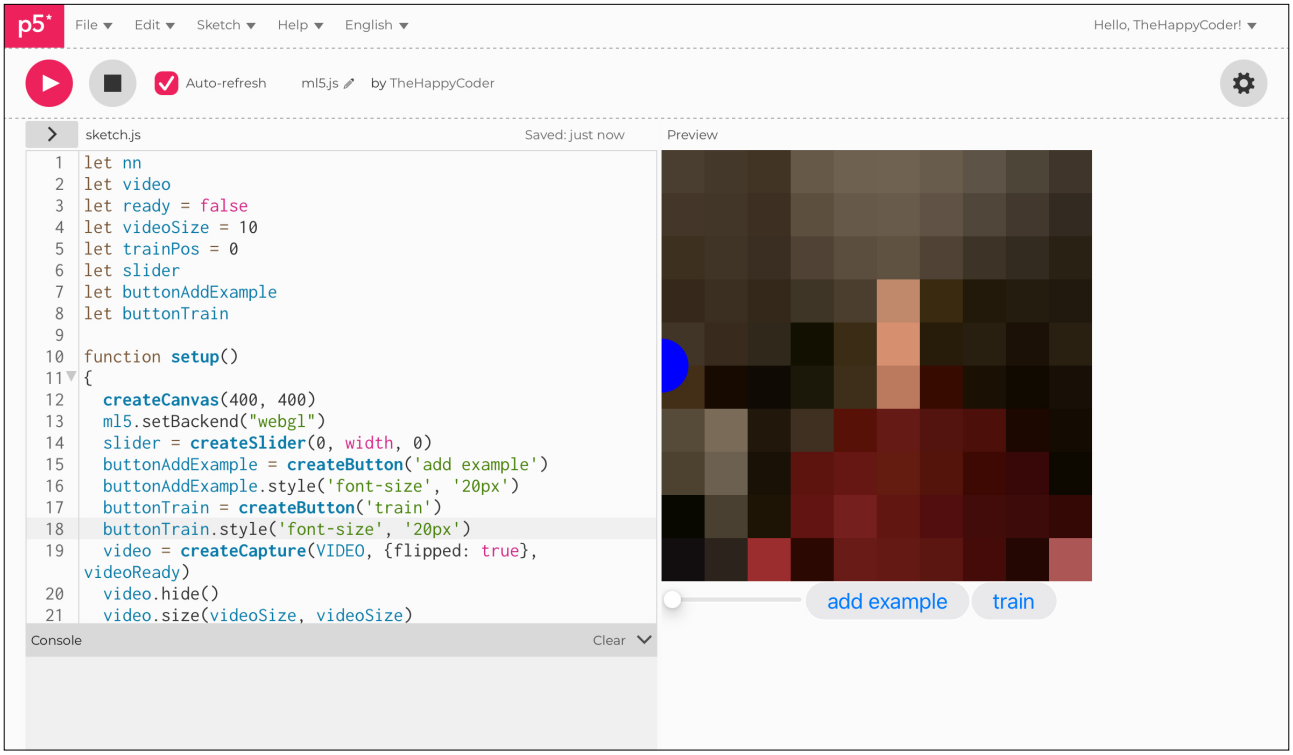
```
let nn
let video
let ready = false
let videoSize = 10
let sliderPos = 0
let slider
let buttonAddExample
let buttonTrain

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  slider = createSlider(0, width, 0)
  buttonAddExample = createButton('add example')
  buttonAddExample.style('font-size', '20px')
  buttonTrain = createButton('train')
  buttonTrain.style('font-size', '20px')
  video = createCapture(VIDEO, {flipped: true}, videoReady)
  video.hide()
  video.size(videoSize, videoSize)
  let totalPixels = videoSize * videoSize * 3
  const options = {
    inputs: totalPixels,
    outputs: 1,
    learningRate: 0.01,
    task: 'regression',
    debug: true
  }
  nn = ml5.neuralNetwork(options)
}

function videoReady()
```

```
{
  ready = true
}

function draw()
{
  background(220)
  if (ready)
  {
    let w = width/videoSize
    video.loadPixels()
    for (let x = 0; x < video.width; x++)
    {
      for (let y = 0; y < video.height; y++)
      {
        let index = (x + y * video.width) * 4
        let r = video.pixels[index + 0]
        let g = video.pixels[index + 1]
        let b = video.pixels[index + 2]
        noStroke()
        fill(r, g, b)
        square(x * w, y * w, w)
      }
    }
    sliderPos = slider.value()
    fill(0, 0, 255)
    circle(sliderPos, height/2, 50)
  }
}
```

🗒️ Notes

Now we have the two buttons and a slider.

Figure A12.13



Sketch A12.14 getting the inputs

Now we have the two buttons and a slider, we can do something with them. The aim is to put the slider to zero, move our head to the left (or left hand), and effectively take several snapshots (adding about five examples each), moving the slider to the other edge of the canvas, and repeat with our head in that position (or right hand). So we end up with ten (or however many) datasets. We will do this in stages; the first stage is to get those inputs by creating a function called `getInputs()`.

We are going to load the pixels; remember, it is taking the tiny (`10` x `10`) video. We need an empty array to store these inputs. There are three values for each pixel: the red (`r`), green (`g`), and blue (`b`) values, which are `index + 0`, `index + 1`, and `index + 2`, respectively, for each pixel.

```
let nn
let video
let ready = false
let videoSize = 10
let sliderPos = 0
let slider
let buttonAddExample
let buttonTrain

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  slider = createSlider(0, width, 0)
  buttonAddExample = createButton('add example')
  buttonAddExample.style('font-size', '20px')
  buttonTrain = createButton('train')
  buttonTrain.style('font-size', '20px')
  video = createCapture(VIDEO, {flipped: true}, videoReady)
  video.hide()
  video.size(videoSize, videoSize)
  let totalPixels = videoSize * videoSize * 3
  const options = {
    inputs: totalPixels,
    outputs: 1,
    learningRate: 0.01,
```

```
    task: 'regression',
    debug: true
  }
  nn = ml5.neuralNetwork(options)
}

function videoReady()
{
  ready = true
}

function draw()
{
  background(220)
  if (ready)
  {
    let w = width/videoSize
    video.loadPixels()
    for (let x = 0; x < video.width; x++)
    {
      for (let y = 0; y < video.height; y++)
      {
        let index = (x + y * video.width) * 4
        let r = video.pixels[index + 0]
        let g = video.pixels[index + 1]
        let b = video.pixels[index + 2]
        noStroke()
        fill(r, g, b)
        square(x * w, y * w, w)
      }
    }
    sliderPos = slider.value()
    fill(0, 0, 255)
    circle(sliderPos, height/2, 50)
  }
}
```

```
function getInputs()
```

```
{
  video.loadPixels()
  let inputs = []
  for (let i = 0; i < video.width * video.height; i++)
  {
    let index = i * 4
    inputs.push(video.pixels[index + 0])
    inputs.push(video.pixels[index + 1])
    inputs.push(video.pixels[index + 2])
  }
  return inputs
}
```

🗒️ Notes

Quite a few lines of code which look very similar to the `draw()` function. We are using the pixels from the tiny video, hence `video.pixels[]` rather than just `pixels[]`.

🛠️ Code Explanation

`let inputs = []`	This is the inputs array
`for (let i = 0; i < video.width * video.height; i++)`	The video.width times the video.height gives us all the pixels
`let index = i * 4`	We jump every four elements in the pixel array to get the next pixel
`inputs.push(video.pixels[index + 0])`	The red value is pushed into the inputs array
`inputs.push(video.pixels[index + 1])`	The green value is pushed into the inputs array
`inputs.push(video.pixels[index + 2])`	The blue value is pushed into the inputs array
`return inputs`	Returns the complete array

Sketch A12.15 adding an example

We want to add the data to the neural network. We want the inputs for the position of the slider. We create another variable called pos, which returns the value of the slider. We want to add it every time we click on the add example button with the mouse.

```
let nn
let video
let ready = false
let videoSize = 10
let sliderPos = 0
let slider
let buttonAddExample
let buttonTrain
let pos = 0

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  slider = createSlider(0, width, 0)
  buttonAddExample = createButton('add example')
  buttonAddExample.style('font-size', '20px')
  buttonTrain = createButton('train')
  buttonTrain.style('font-size', '20px')
  video = createCapture(VIDEO, {flipped: true}, videoReady)
  video.hide()
  video.size(videoSize, videoSize)
  let totalPixels = videoSize * videoSize * 3
  const options = {
    inputs: totalPixels,
    outputs: 1,
    learningRate: 0.01,
    task: 'regression',
    debug: true
  }
  nn = ml5.neuralNetwork(options)
}
```

```
function videoReady()
{
  ready = true
}

function draw()
{
  background(220)
  buttonAddExample.mousePressed(addExample)
  if (ready)
  {
    let w = width/videoSize
    video.loadPixels()
    for (let x = 0; x < video.width; x++)
    {
      for (let y = 0; y < video.height; y++)
      {
        let index = (x + y * video.width) * 4
        let r = video.pixels[index + 0]
        let g = video.pixels[index + 1]
        let b = video.pixels[index + 2]
        noStroke()
        fill(r, g, b)
        square(x * w, y * w, w)
      }
    }
    sliderPos = slider.value()
    fill(0, 0, 255)
    circle(sliderPos, height/2, 50)
  }
}

function getInputs()
{
  video.loadPixels()
  let inputs = []
  for (let i = 0; i < video.width * video.height; i++)
```

```
  {
    let index = i * 4
    inputs.push(video.pixels[index + 0])
    inputs.push(video.pixels[index + 1])
    inputs.push(video.pixels[index + 2])
  }
  return inputs
}
```

```
function addExample()
{
  pos = slider.value()
  let inputs = getInputs()
  nn.addData(inputs, [pos])
}
```

🗒️ Notes

We add the pixel data combined with the slider position for each example.

🛠️ Code Explanation

`let pos = 0`	We need a new variable otherwise we end up with two sets of data from the slider position
`buttonAddExample.mousePressed(addExample)`	We call the addExample function
`pos = slider.value()`	The pos variable gets the slider position
`let inputs = getInputs()`	We bring the pixel dataset
`nn.addData(inputs, [pos])`	Adding the data, the inputs (pixels) and the output (pos as an array)

Sketch A12.16 training the model

We have now collected the training data examples, and we want to train the model on that data. So, we create another function called `trainModel()`. This normalises the data and then trains on it. We will train it for `50 epochs`.

❗ If you run it now, you will get an error.

```
let nn
let video
let ready = false
let videoSize = 10
let sliderPos = 0
let slider
let buttonAddExample
let buttonTrain
let pos = 0

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  slider = createSlider(0, width, 0)
  buttonAddExample = createButton('add example')
  buttonAddExample.style('font-size', '20px')
  buttonTrain = createButton('train')
  buttonTrain.style('font-size', '20px')
  video = createCapture(VIDEO, {flipped: true}, videoReady)
  video.hide()
  video.size(videoSize, videoSize)
  let totalPixels = videoSize * videoSize * 3
  const options = {
    inputs: totalPixels,
    outputs: 1,
    learningRate: 0.01,
    task: 'regression',
    debug: true
  }
  nn = ml5.neuralNetwork(options)
```

```
}

function videoReady()
{
  ready = true
}

function draw()
{
  background(220)
  buttonAddExample.mousePressed(addExample)
  if (ready)
  {
    let w = width/videoSize
    video.loadPixels()
    for (let x = 0; x < video.width; x++)
    {
      for (let y = 0; y < video.height; y++)
      {
        let index = (x + y * video.width) * 4
        let r = video.pixels[index + 0]
        let g = video.pixels[index + 1]
        let b = video.pixels[index + 2]
        noStroke()
        fill(r, g, b)
        square(x * w, y * w, w)
      }
    }
    sliderPos = slider.value()
    fill(0, 0, 255)
    circle(sliderPos, height/2, 50)
  }
}

function getInputs()
{
  video.loadPixels()
  let inputs = []
```

```
  for (let i = 0; i < video.width * video.height; i++)
  {
    let index = i * 4
    inputs.push(video.pixels[index + 0] / 255)
    inputs.push(video.pixels[index + 1] / 255)
    inputs.push(video.pixels[index + 2] / 255)
  }
  return inputs
}

function addExample()
{
  pos = slider.value()
  let inputs = getInputs()
  nn.addData(inputs, [pos])
  buttonTrain.mousePressed(trainModel)
}

function trainModel()
{
  nn.normalizeData()
  nn.train(finishedTraining)
}
```

🗒️ Notes

In the `train()` function, we have a callback function for when the training is finished, called `finishedTraining`. We don't have the `finishedTraining()` function so you will get an error.

🛠️ Code Explanation

`buttonTrain.mousePressed(trainModel)`	When the button to train has been pressed it calls the trainModel function
`nn.normalizeData()`	Data is then normalised
`nn.train(finishedTraining)`	Trained over 50 epochs and then the callback

Sketch A12.17 finished training

When we have finished training the model, we want to predict the result. To do that, we have a function and call it `predict()`.

❗ If you run it now, you will still get an error.

```
let nn
let video
let ready = false
let videoSize = 10
let sliderPos = 0
let slider
let buttonAddExample
let buttonTrain
let pos = 0

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  slider = createSlider(0, width, 0)
  buttonAddExample = createButton('add example')
  buttonAddExample.style('font-size', '20px')
  buttonTrain = createButton('train')
  buttonTrain.style('font-size', '20px')
  video = createCapture(VIDEO, {flipped: true}, videoReady)
  video.hide()
  video.size(videoSize, videoSize)
  let totalPixels = videoSize * videoSize * 3
  const options = {
    inputs: totalPixels,
    outputs: 1,
    learningRate: 0.01,
    task: 'regression',
    debug: true
  }
  nn = ml5.neuralNetwork(options)
}
```

```
function videoReady()
{
  ready = true
}

function draw()
{
  background(220)
  buttonAddExample.mousePressed(addExample)
  if (ready)
  {
    let w = width/videoSize
    video.loadPixels()
    for (let x = 0; x < video.width; x++)
    {
      for (let y = 0; y < video.height; y++)
      {
        let index = (x + y * video.width) * 4
        let r = video.pixels[index + 0]
        let g = video.pixels[index + 1]
        let b = video.pixels[index + 2]
        noStroke()
        fill(r, g, b)
        square(x * w, y * w, w)
      }
    }
    sliderPos = slider.value()
    fill(0, 0, 255)
    circle(sliderPos, height/2, 50)
  }
}

function getInputs()
{
  video.loadPixels()
  let inputs = []
  for (let i = 0; i < video.width * video.height; i++)
```

```
  {
    let index = i * 4
    inputs.push(video.pixels[index + 0] / 255)
    inputs.push(video.pixels[index + 1] / 255)
    inputs.push(video.pixels[index + 2] / 255)
  }
  return inputs
}

function addExample()
{
  pos = slider.value()
  let inputs = getInputs()
  nn.addData(inputs, [pos])
  buttonTrain.mousePressed(trainModel)
}

function trainModel()
{
  nn.normalizeData()
  nn.train(finishedTraining)
}

function finishedTraining()
{
  predict()
}
```

🗒️ Notes

Finished training, which means we are now ready to predict. No `predict()` function hence the error.

Sketch A12.18 predicting the movement

Now we need to predict the result after training. We use the pixels from the video, so that when we move our heads from left to right (left or right hand), it can predict the position of the circle based on our trained model. Here we create the `predict()` function.

! If you run it, you will still get an error.

```
let nn
let video
let ready = false
let videoSize = 10
let sliderPos = 0
let slider
let buttonAddExample
let buttonTrain
let pos = 0

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  slider = createSlider(0, width, 0)
  buttonAddExample = createButton('add example')
  buttonAddExample.style('font-size', '20px')
  buttonTrain = createButton('train')
  buttonTrain.style('font-size', '20px')
  video = createCapture(VIDEO, {flipped: true}, videoReady)
  video.hide()
  video.size(videoSize, videoSize)
  let totalPixels = videoSize * videoSize * 3
  const options = {
    inputs: totalPixels,
    outputs: 1,
    learningRate: 0.01,
    task: 'regression',
    debug: true
  }
  nn = ml5.neuralNetwork(options)
```

```
}

function videoReady()
{
  ready = true
}

function draw()
{
  background(220)
  buttonAddExample.mousePressed(addExample)
  if (ready)
  {
    let w = width/videoSize
    video.loadPixels()
    for (let x = 0; x < video.width; x++)
    {
      for (let y = 0; y < video.height; y++)
      {
        let index = (x + y * video.width) * 4
        let r = video.pixels[index + 0]
        let g = video.pixels[index + 1]
        let b = video.pixels[index + 2]
        noStroke()
        fill(r, g, b)
        square(x * w, y * w, w)
      }
    }
    sliderPos = slider.value()
    fill(0, 0, 255)
    circle(sliderPos, height/2, 50)
  }
}

function getInputs()
{
  video.loadPixels()
  let inputs = []
```

```
  for (let i = 0; i < video.width * video.height; i++)
  {
    let index = i * 4
    inputs.push(video.pixels[index + 0] / 255)
    inputs.push(video.pixels[index + 1] / 255)
    inputs.push(video.pixels[index + 2] / 255)
  }
  return inputs
}

function addExample()
{
  pos = slider.value()
  let inputs = getInputs()
  nn.addData(inputs, [pos])
  buttonTrain.mousePressed(trainModel)
}

function trainModel()
{
  nn.normalizeData()
  nn.train(finishedTraining)
}

function finishedTraining()
{
  predict()
}
```

```
function predict()
{
  let inputs = getInputs()
  nn.predict(inputs, gotPosition)
}
```

🗒️ Notes

In the `predict()` function, we collect our head (or hand) movement and feed it into the neural network, and have a callback for the results called `gotPosition`. An error because of the lack of `gotPosition()` function.

🛠️ Code Explanation

`let inputs = getInputs()`	The pixels are imputed from the video
`nn.predict(inputs, gotPosition)`	The callback receives the result (output)

Sketch A12.19 predict ready

When we have a prediction, we then need to draw a different circle (in red). We don't want to draw anything while we are getting data or training; hence, the boolean `predictReady` is set to `false` until we have done the prediction. In the `draw()` function, we need to remove the blue circle (used for training) and replace it with the red circle (for prediction) instead. We set `predictReady` to be `true`, get the result, and send the red circle to the predicted position (`pos`). We predict again because we want to see the red circle move when we move our head (or hand) from left to right. The circle should respond to the changes.

```
let nn
let video
let ready = false
let videoSize = 10
let sliderPos = 0
let slider
let buttonAddExample
let buttonTrain
let pos = 0
let predictReady = false

function setup()
{
  createCanvas(400, 400)
  ml5.setBackend("webgl")
  slider = createSlider(0, width, 0)
  buttonAddExample = createButton('add example')
  buttonAddExample.style('font-size', '20px')
  buttonTrain = createButton('train')
  buttonTrain.style('font-size', '20px')
  video = createCapture(VIDEO, {flipped: true}, videoReady)
  video.hide()
  video.size(videoSize, videoSize)
  let totalPixels = videoSize * videoSize * 3
  const options = {
    inputs: totalPixels,
    outputs: 1,
    learningRate: 0.01,
    task: 'regression',
```

```
    debug: true
  }
  nn = ml5.neuralNetwork(options)
}

function videoReady()
{
  ready = true
}

function draw()
{
  background(220)
  buttonAddExample.mousePressed(addExample)
  if (ready)
  {
    let w = width/videoSize
    video.loadPixels()
    for (let x = 0; x < video.width; x++)
    {
      for (let y = 0; y < video.height; y++)
      {
        let index = (x + y * video.width) * 4
        let r = video.pixels[index + 0]
        let g = video.pixels[index + 1]
        let b = video.pixels[index + 2]
        noStroke()
        fill(r, g, b)
        square(x * w, y * w, w)
      }
    }
    if (predictReady == true)
    {
      fill(255, 0, 0)
      circle(pos, height/2, 50)
    }
    else
    {
```

```
      sliderPos = slider.value()
      fill(0, 0, 255)
      circle(sliderPos, height/2, 50)
    }
  }
}

function getInputs()
{
  video.loadPixels()
  let inputs = []
  for (let i = 0; i < video.width * video.height; i++)
  {
    let index = i * 4
    inputs.push(video.pixels[index + 0])
    inputs.push(video.pixels[index + 1])
    inputs.push(video.pixels[index + 2])
  }
  return inputs
}

function addExample()
{
  pos = slider.value()
  let inputs = getInputs()
  nn.addData(inputs, [pos])
  buttonTrain.mousePressed(trainModel)
}

function trainModel()
{
  nn.normalizeData()
  nn.train(finishedTraining)
}

function finishedTraining()
{
  predict()
```

```
}

function predict()
{
  let inputs = getInputs()
  nn.predict(inputs, gotPosition)
}
```

```
function gotPosition(results)
{
  predictReady = true
  pos = (results[0].value)
  predict()
}
```

🗒️ Notes

We introduce the `if()...else` so we only draw the red circle when we are ready and not before; otherwise, we draw the blue.

🌻 Challenges

1. Make the video size smaller (smaller pixels = more of them).
2. Add other hyperparameters.
3. Make other things happen in response to your training.

🛠️ Code Explanation

`let predictReady = false`	Initialise the predictReady as false
`function gotPosition(results)`	Argument is the result
`predictReady = true`	Boolean true
`pos = (results[0].value)`	Get the current result from the output (pos) array
`predict()`	Repeat
`if (predictReady == true)`	If the boolean is true then draw the red circle at the predicted position otherwise (else) continue to draw the blue circle

Figure A12.19a: loss chart

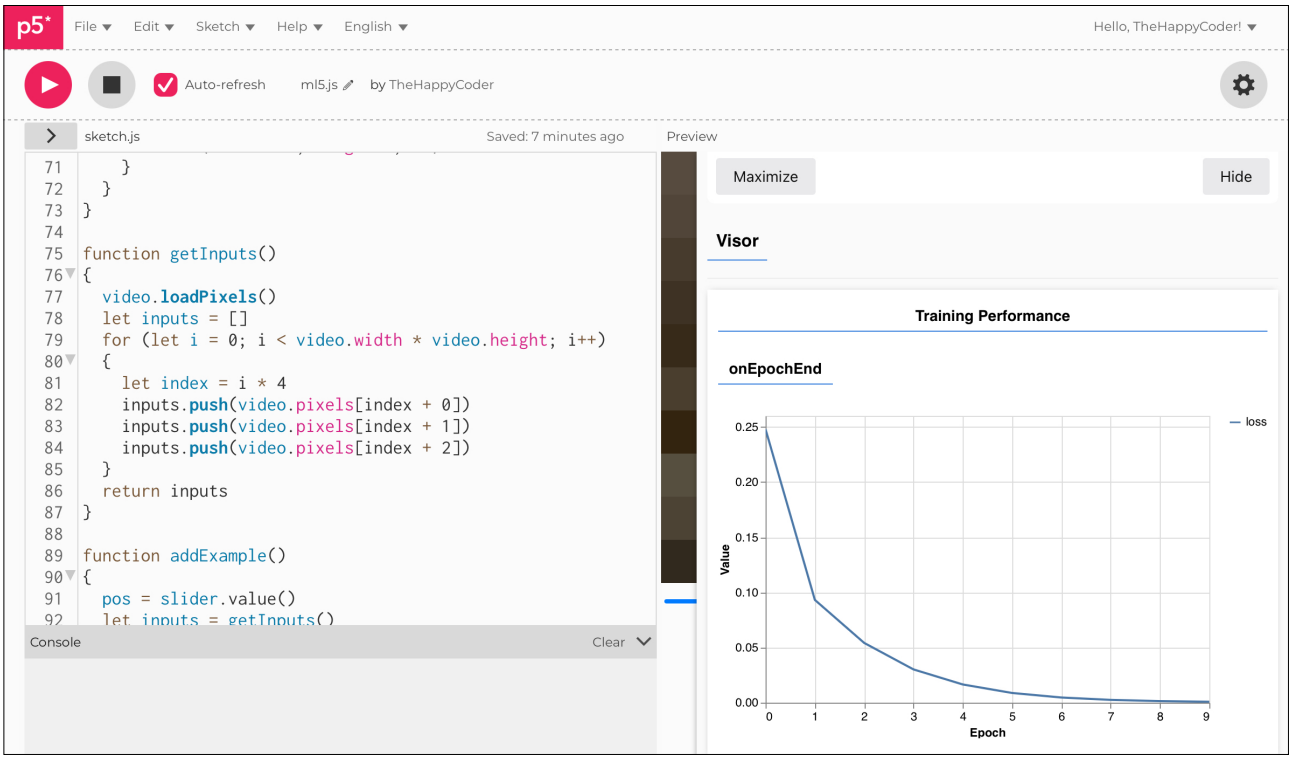


Figure A12.19b: to the left

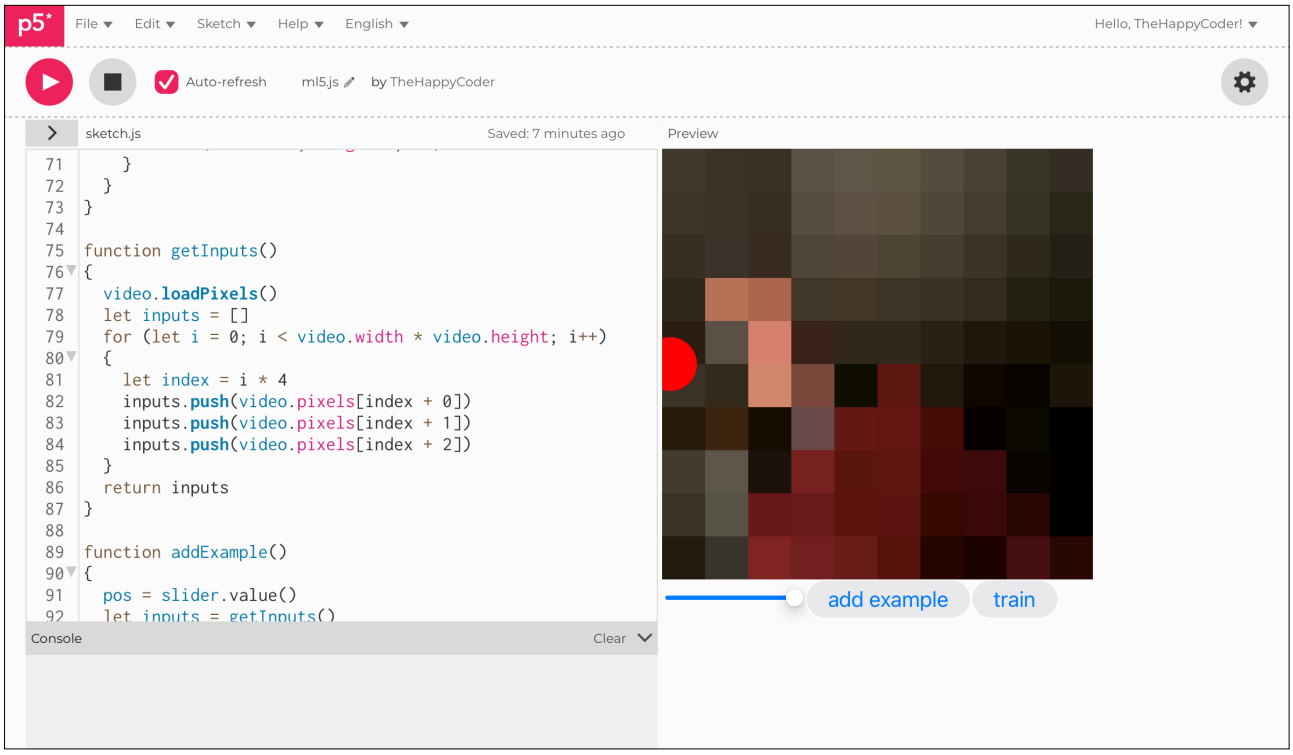


Figure A12.19c: to the right