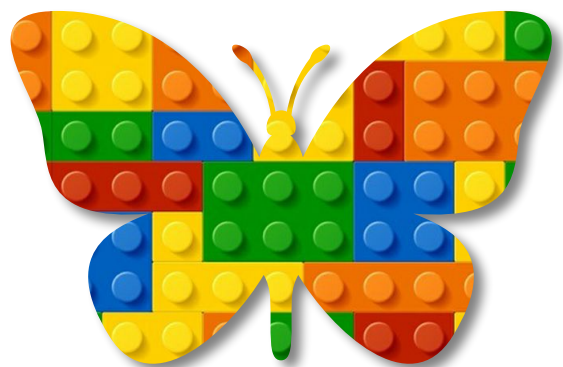


Artificial Intelligence Module A Unit #9 pixel predictor





Module A Unit #9 pixel predictor

Introduction to pixel predictor

The [index.html](#) file

Sketch A9.1	starting sketch
Sketch A9.2	hiding the video
Sketch A9.3	mirrored
Sketch A9.4	video ready
Sketch A9.5	the size of the video
Sketch A9.6	resizing the video
Sketch A9.7	even smaller
Sketch A9.8	loading the pixels
Sketch A9.9	enlarged pixels
Sketch A9.10	neural network
Sketch A9.11	adding the slider output
Sketch A9.12	adding the buttons
Sketch A9.13	the training button
Sketch A9.14	getting the inputs
Sketch A9.15	adding an example
Sketch A9.16	training the model
Sketch A9.17	finished training
Sketch A9.18	predicting the movement
Sketch A9.19	predict ready



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.



Just a reminder of 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 A9.1 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

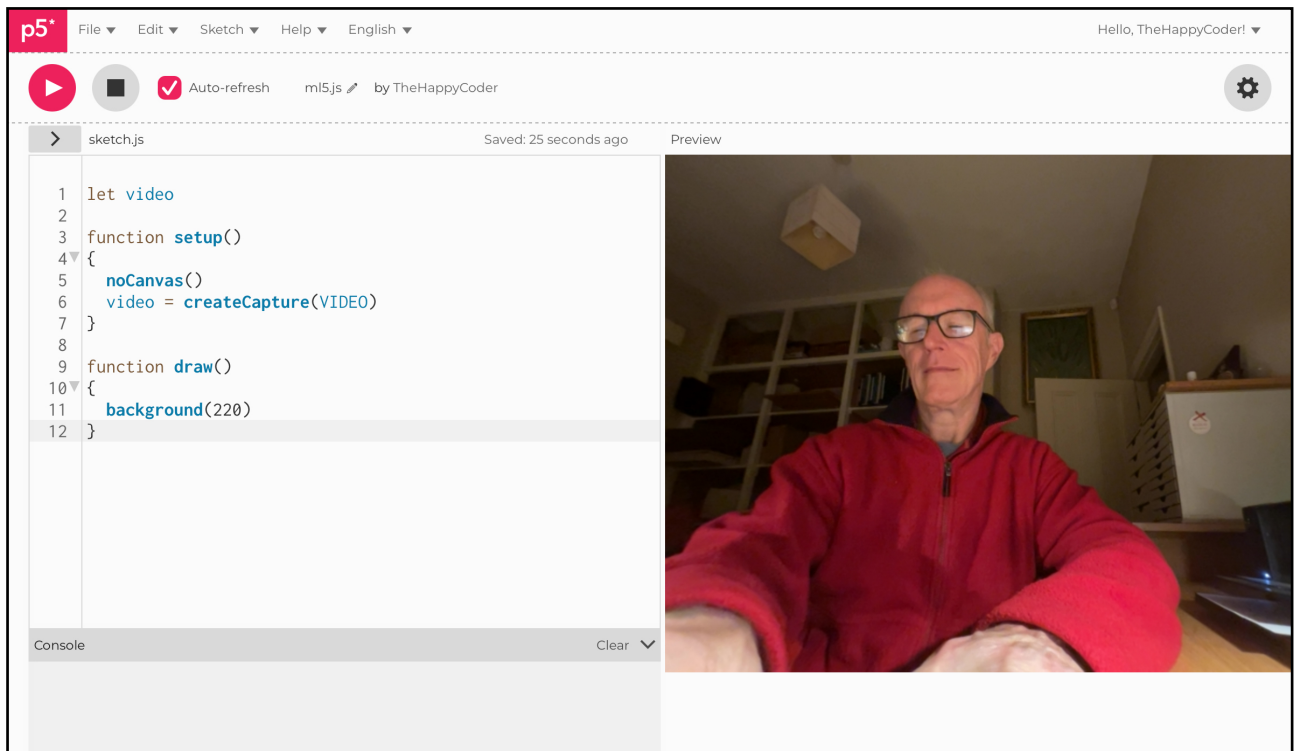
You should have a nice video image of yourself, the dimensions should be 640 x 480



Code Explanation

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

Figure A9.1



Sketch A9.2 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

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



Sketch A9.3 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

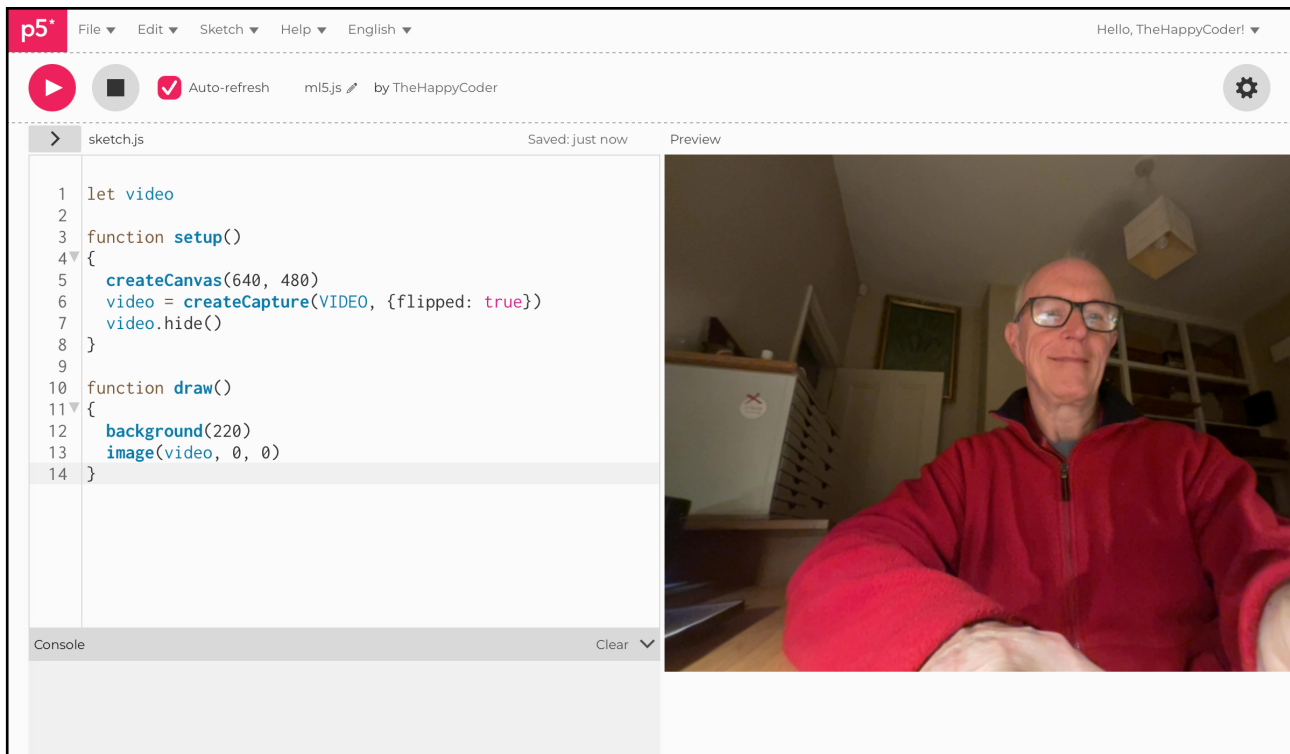
Now you have a mirrored image from the webcam



Code Explanation

<code>{flipped: true}</code>	Flips the video so that it behaves like a mirror
------------------------------	--

Figure A9.3 mirrored





Sketch A9.4 video ready

Now we want to draw it on the canvas in pixels. However, we want to double check that the video to be 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 for when we start adding data to the neural network



Code Explanation

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



Sketch A9.5 the size of the video

At present we have an image coming in at **640x480** pixels. You are drawing that onto the canvas of the same dimensions (and ratio). If we change the canvas to **400x400** we crop the image losing some of the image. We want to use a square image and then we can simplify the pixelations 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
}

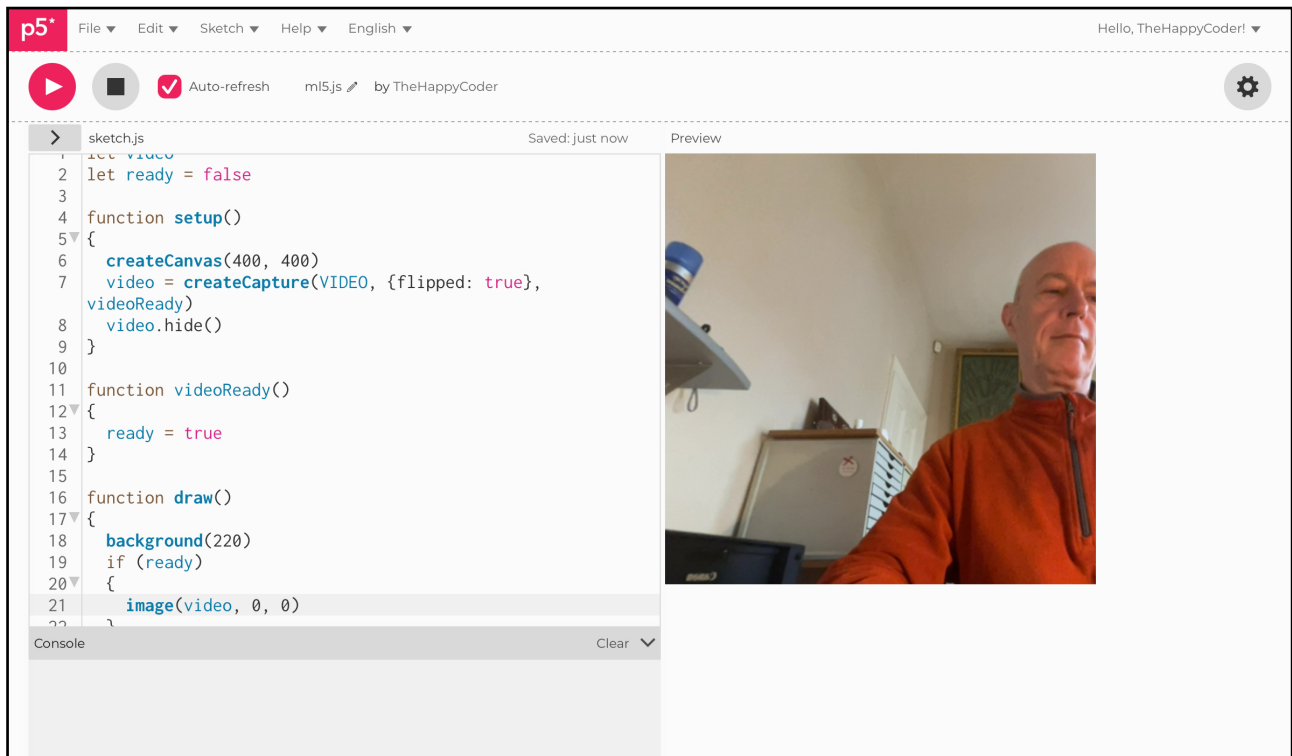
function draw()
{
  background(220)
  if (ready)
  {
    image(video, 0, 0)
  }
}
```



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 A9.5





Sketch A9.6 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 **400x400**, 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

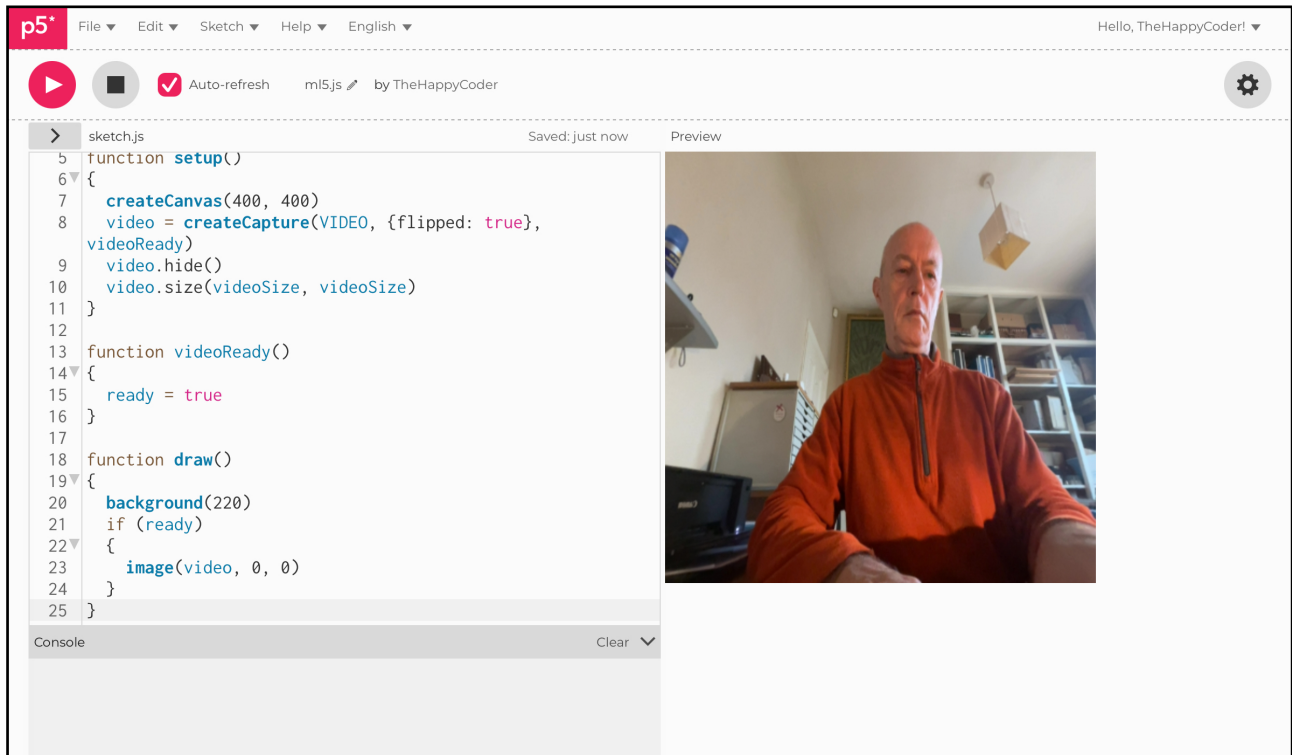
1. Try other dimensions
2. Try 640x480



Code Explanation

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

Figure A9.6





Sketch A9.7 even smaller

What we want to do is change the size to **10x10**, yes, that small! This means the software behind producing the video is going to try and create a **10x10** video image from a **640x480** 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 try to approximate the larger image.



Challenge

Try something a little bigger, say 50 pixels

Figure A9.7a can you see it?

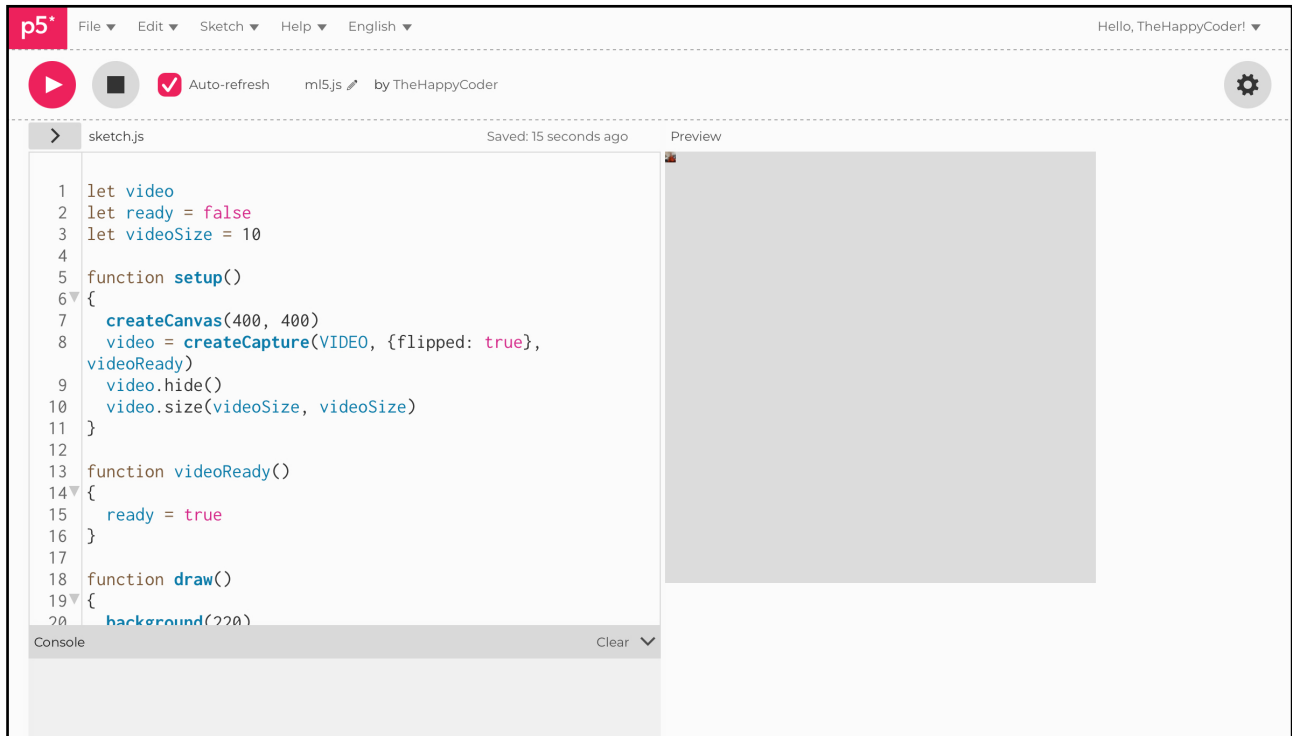
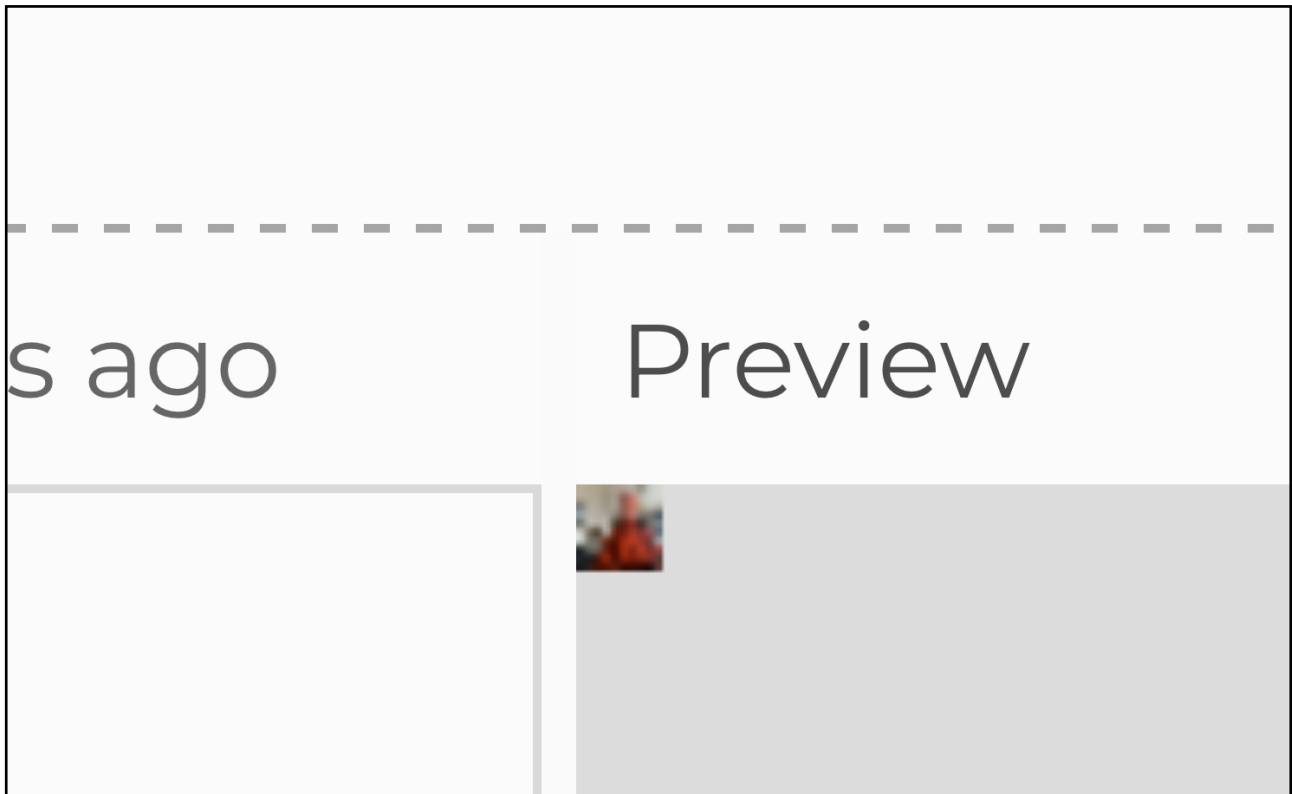


Figure A9.7b yes, it really is me!





Sketch A9.8 loading the pixels

We want to take those pixels which equates 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** datapoints, 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 at 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

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

<pre>let g = video.pixels[index + 1]</pre>	This is the index for the second element (green) of the pixel colour
<pre>let b = video.pixels[index + 2]</pre>	This is the index for the third element (blue) of the pixel colour



Sketch A9.9 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 **10x10** 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 code 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

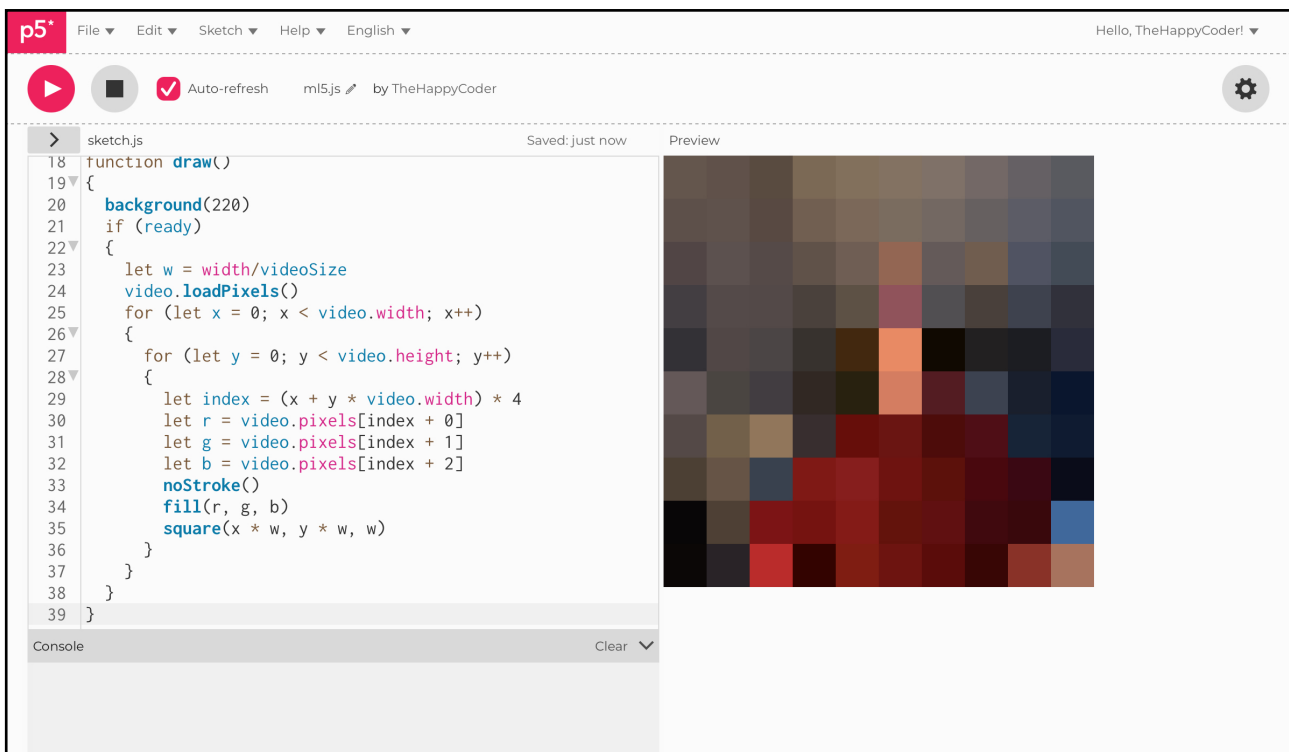
We now have a larger version of that tiny image of **10x10** pixels on a canvas of **10 by 10** squares.



Code Explanation

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

Figure A9.9 it really is me





Sketch A9.10 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 A9.11 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 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 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 output.



Code Explanation

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

Figure A9.11 slider to the right

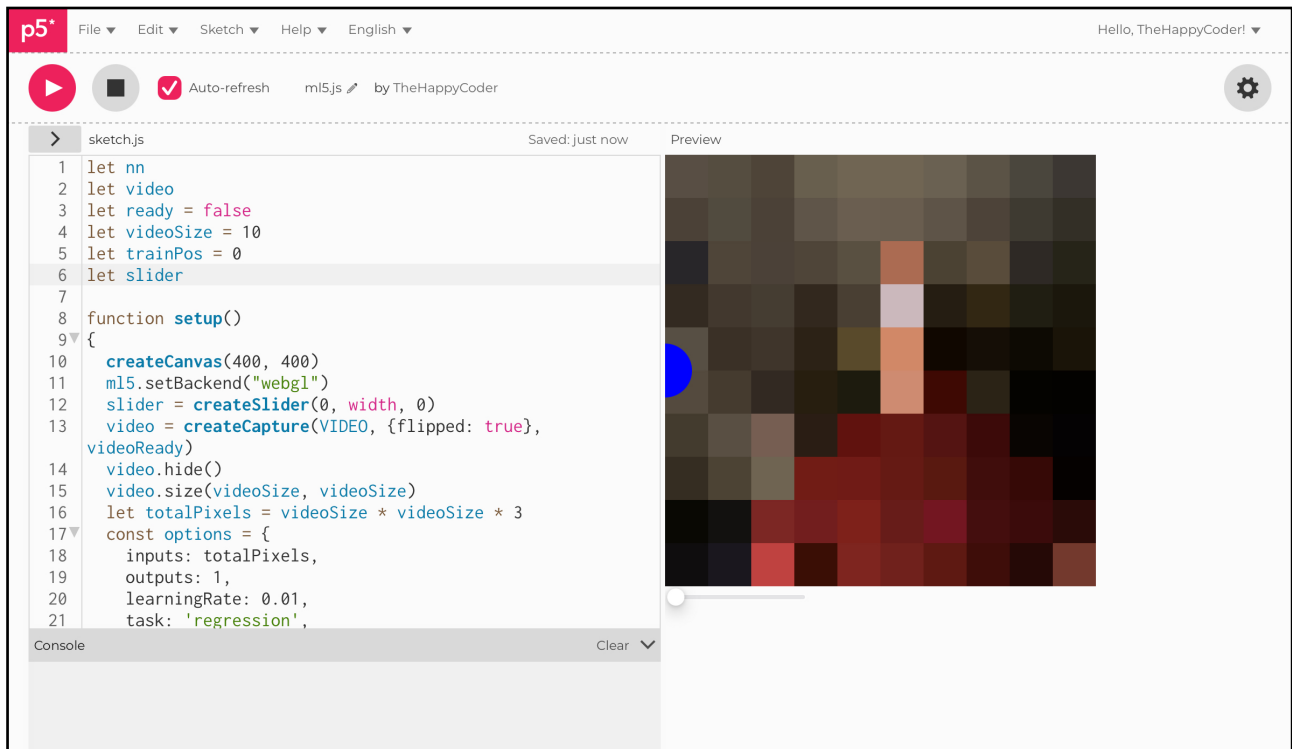
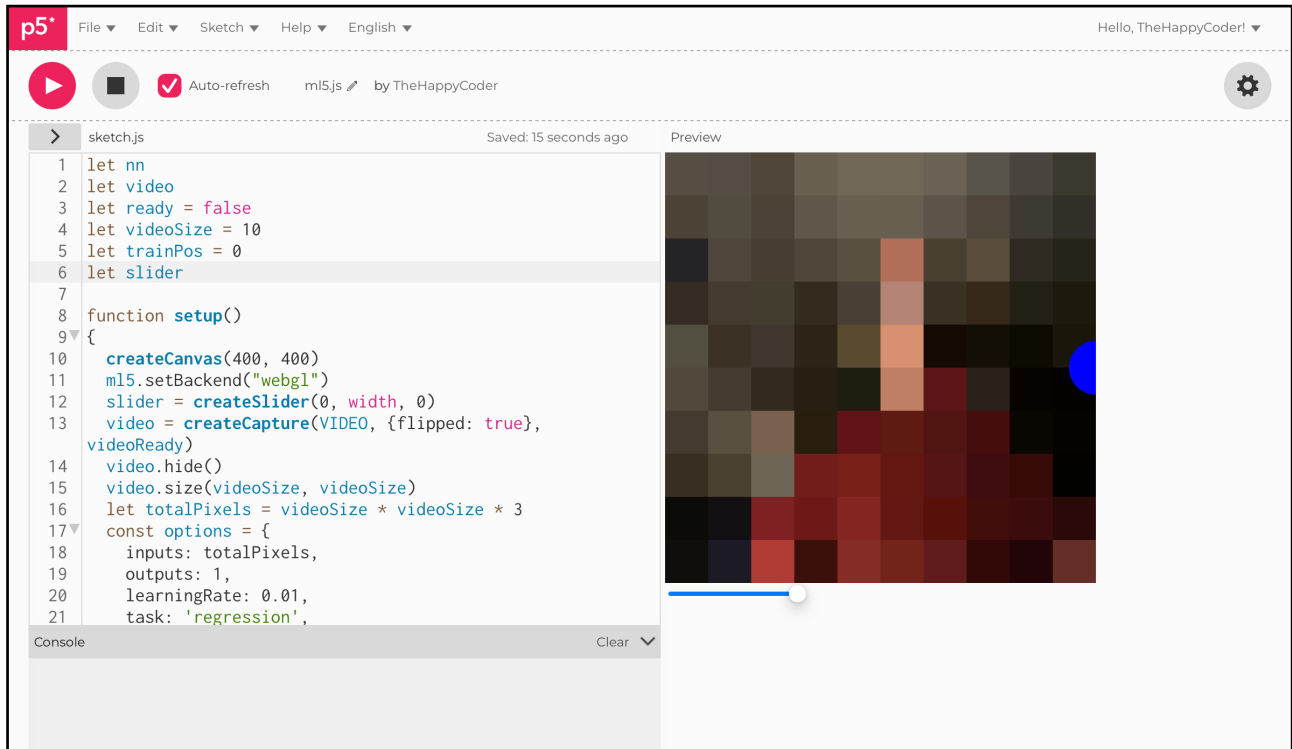


Figure A9.11 slider to the right





Sketch A9.12 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
}

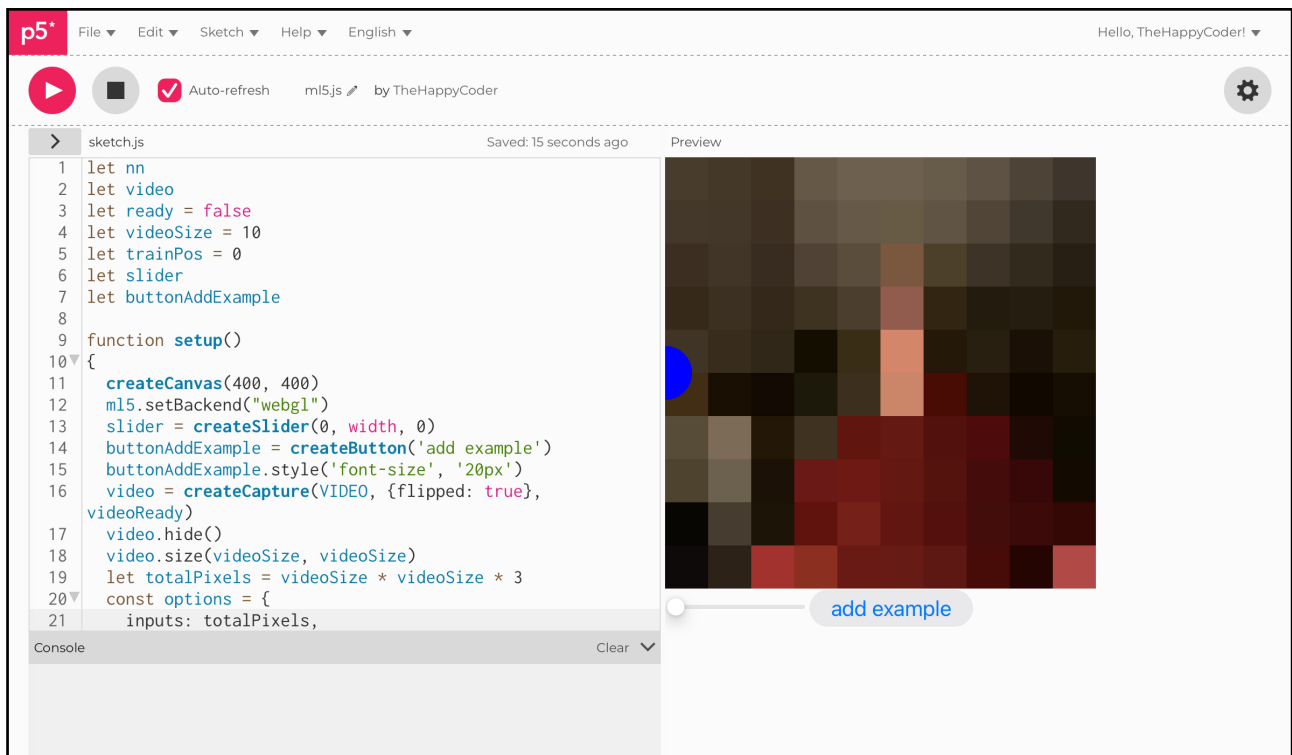
function draw()
{
  background(220)
  if (ready)
  {
    let w = width/videoSize
    video.loadPixels()
    for (let x = 0; x < video.width; x++)
    {
      for (let y = 0; y < video.height; y++)
      {
        let index = (x + y * video.width) * 4
        let r = video.pixels[index + 0]
        let 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 A9.12





Sketch A9.13 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)
  }
}

```

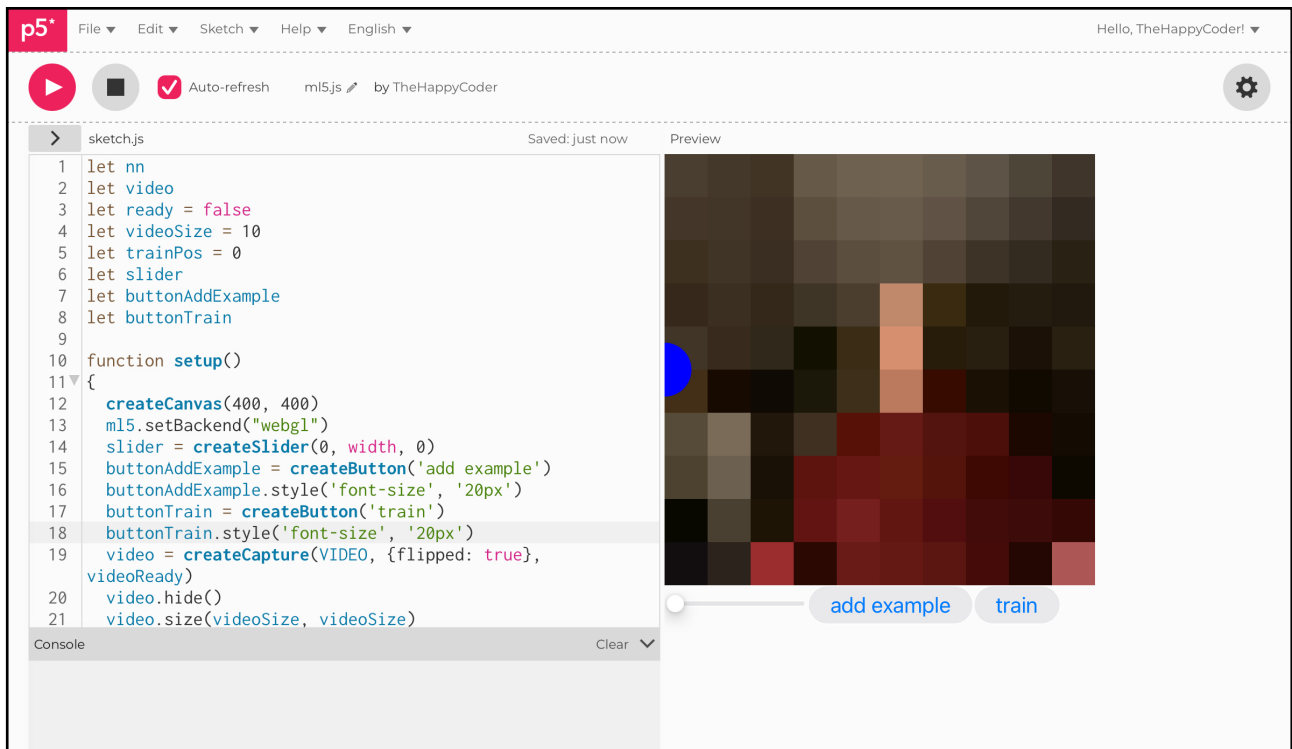
```
}
```



Notes

Now we have the two buttons and a slider

Figure A9.13





Sketch A9.14 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, 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 (10x10) 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` respectfully 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 the tiny video hence `video.pixels[]` hence not just `pixels[]`.



Code Explanation

<code>let inputs = []</code>	This is the inputs array
<code>for (let i = 0; i < video.width * video.height; i++)</code>	The video.width times the video.height gives us all the pixels

<code>let index = i * 4</code>	We jump every four elements in the pixel array to get the next pixel
<code>inputs.push(video.pixels[index + 0])</code>	The red value is pushed into the inputs array
<code>inputs.push(video.pixels[index + 1])</code>	The green value is pushed into the inputs array
<code>inputs.push(video.pixels[index + 2])</code>	The blue value is pushed into the inputs array
<code>return inputs</code>	Returns the complete array



Sketch A9.15 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

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



Sketch A9.16 training the model

We have now collected the training data examples and now 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**.

```
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

! if you run it now you will get an error

In the `train()` function we have a callback function for when the training is finished, called `finishedTraining`.



Code Explanation

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



Sketch A9.17 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()`

```
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

! if you run it now you will still get an error
Finished training which means we are now ready to predict.



Sketch A9.18 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.

```
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

! if you run it you will still get an error

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`



Code Explanation

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



Sketch A9.19 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 the `predictReady` to be `true`, we 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 lift our 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 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 videoSize smaller (smaller pixels = more of them),
2. Add other hyperparameters
3. Make other things happen in response to your training



Code Explanation

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

Figure A9.19a loss chart

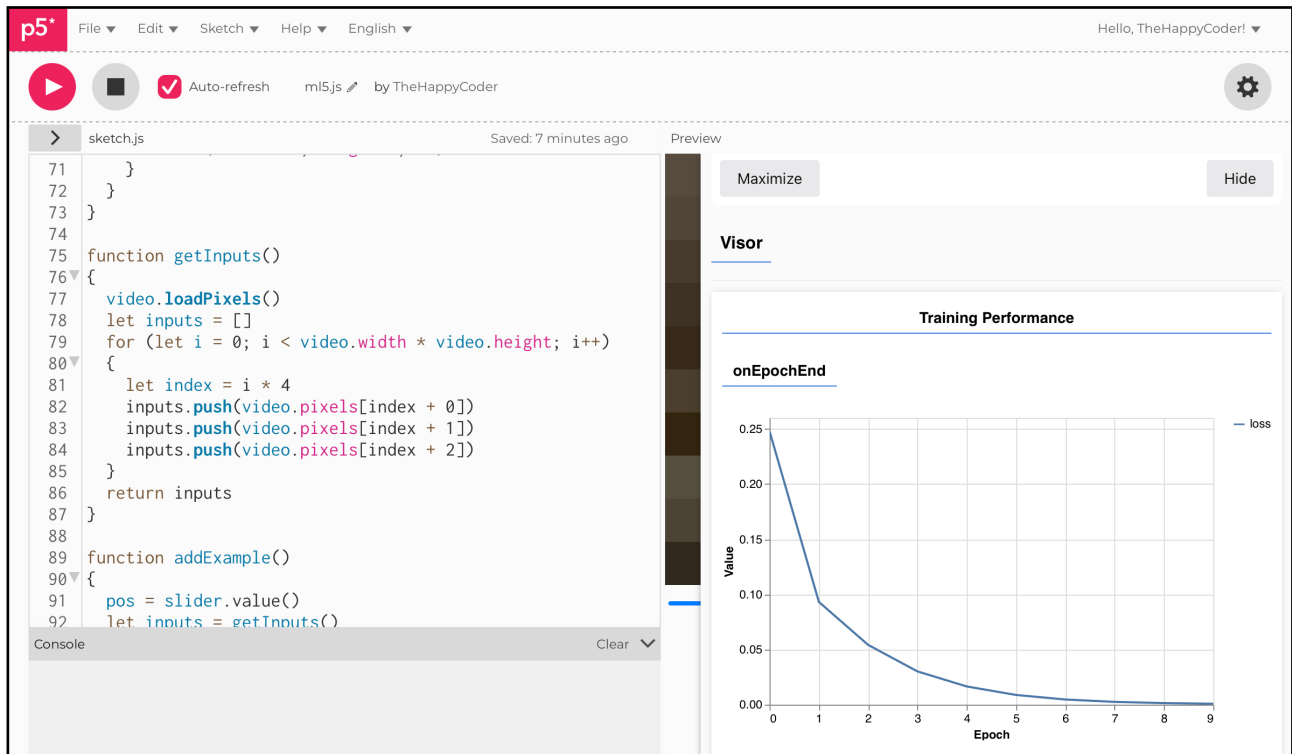


Figure A9.19b to the left

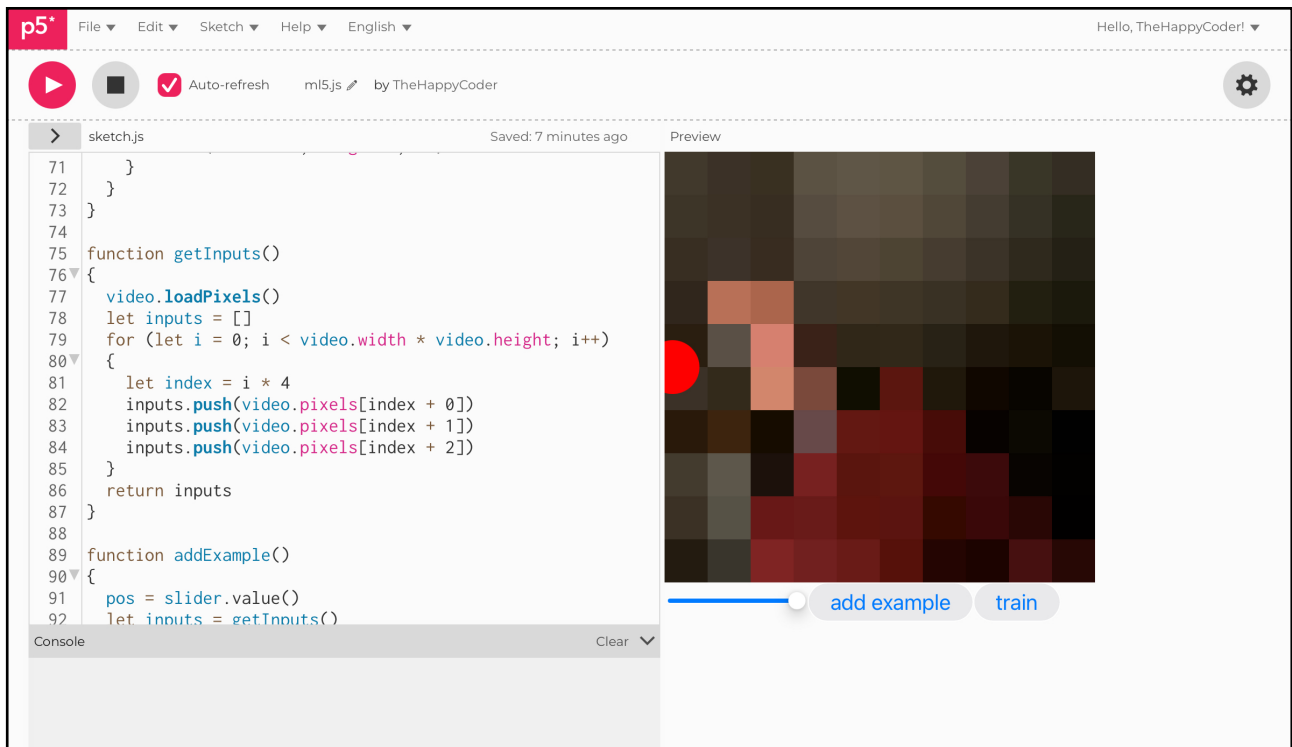


Figure A9.19c to the right

