Module 8
Pixel Predictor
ml5.js

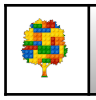
Content

Module 8: Pixel Predictor

Introduction to pixel predictor with ml5.js

We will use a webcam from your PC, Mac, tablet, Chromebook or even phone. The image will be flipped and then pixelated, this means that it is being trained on data that doesn't overwhelm what is basically a simple (as in not huge) model.

Once it is trained you will be able to move a circle across the canvas through moving your head from the left to the right and back again. It will take 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 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/1.10.0/p5.js"></script>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.10.0/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 8.1 starting sketch

For this you will need to allow access to your webcam. You will need to give p5.js permission. What you should get instead of the canvas is a video of yourself. It will be reversed (not mirrored) but we will change that later. The dimensions should be 640 x 480.

```
let video

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

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

Notes

You should have a nice video image of yourself

Sketch 8.2 hiding the video

To make good use of the video we want it on the canvas. So remove: noCanvas() and let's hide the video in setup (otherwise you will have two videos) and put it in 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

Sketch 8.3 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. This won't take affect straight away but will once we use it in draw.

```
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

Sketch 8.4 video ready

Now we want to draw it on the canvas in pixels. However, we want the video to be working before anything else happens so we have a boolean variable that is either true or false. This gives us some control when things happen and in the right order. Default is false and when the video is ready we call a function and set it to 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)
  image(video, 0, 0)
}
```

Sketch 8.5 pixel image

We don't want the video image what we do want is the approximate in larger pixels. Remove: image(video, 0, 0) and instead we are going to draw lots of squares and colour them in. We use a nested loop to get all the pixels.

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

function setup()
{
  createCanvas(640, 480)
  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

There is a lot to take in all in one go but the topic is covered in more depth in the 'learning to Code with p5.js' unit #x video.

Your pixelated image should look like this

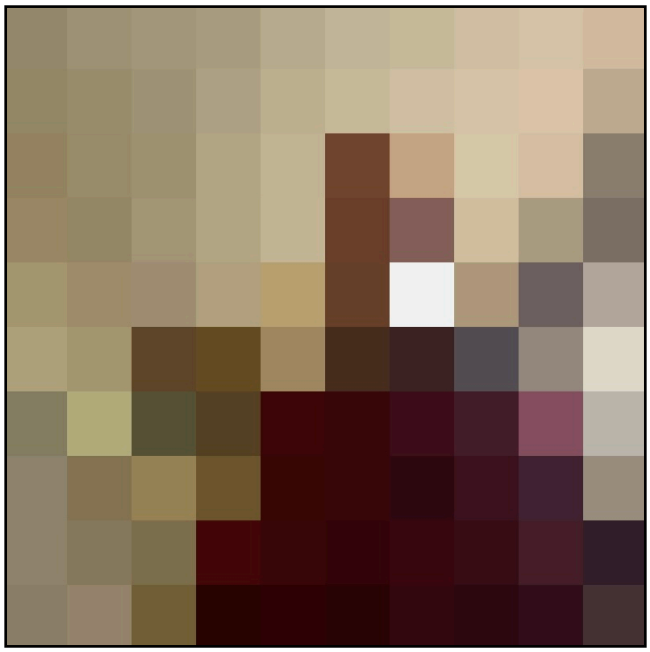
Sketch 8.6 neural network

Now we can add our neural network. The inputs are all the squares, and each square has an RGB value hence x3. In effect we are creating a video with 10 by 10 pixels each with an r, g and b value. So for each input we have 300 values and the output is either 0 or 400 (width). I have changed the canvas size so we now have 10 by 10 pixels showing.

```
let nn
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)
  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)
      }
    }
  }
}
```

Now canvas is (400, 400) and so we have
10 pixels by 10 pixels



Sketch 8.7 buttons

We are going to add a slider for our output. The output is the position of the circle. We use the slider to position the circle and then train on its position. In our example we will train it on the left hand edge and then on the right hand edge (everything will become clear). Plus a button to add images for those two positions. And another button to train.

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

function setup()
{
  createCanvas(400, 400)
  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',
```

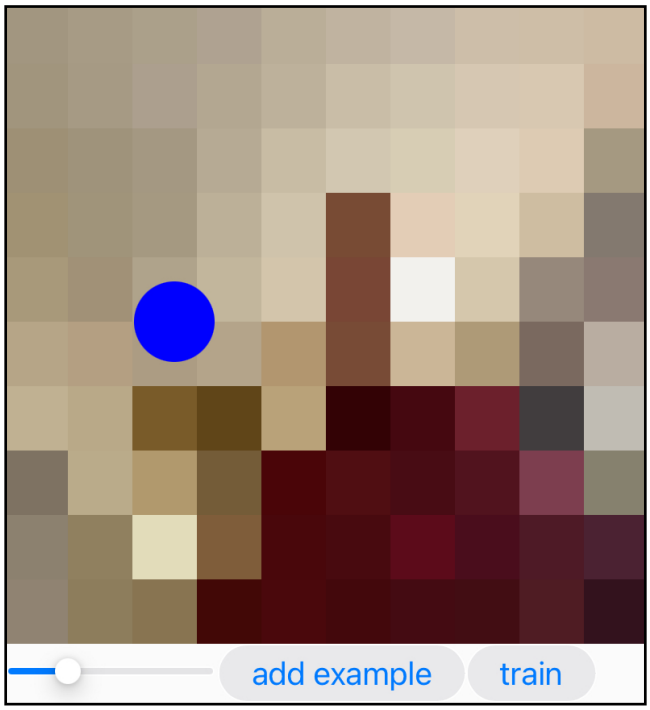
```
    debug: true
  }
  nn = ml5.neuralNetwork(options)
}

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

```
  }
}
```

Notes

The blue circle should move with the slider from the left hand side (0) to the right hand side (400)

The buttons don't do anything yet but the slider should move the blue circle across the canvas



Sketch 8.8 getting the inputs

As we have the buttons and slider we can do something with them. The aim is to put the slider at zero, move our head to the left and effectively take several snapshots (adding about five examples), moving the slider to the other edge of the canvas and repeating. So we end up with ten (or however many) datasets. We will do this in stages, first stage is to get those inputs.

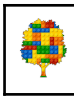
We are going to load the pixels, remember it is taking the pixelated (10x10) video. We need an empty array to store these inputs. There are three values for each pixel the red, green and blue values which are index + 0, index + 1 and index + 2 for each square pixel.

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

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

```
      }
    }
    trainPos = slider.value()
    fill(0, 0, 255)
    circle(trainPos, 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
}
```



Sketch 8.9 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 button with the mouse.

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

function setup()
{
  createCanvas(400, 400)
  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)
      }
    }
    trainPos = slider.value()
```

```
    fill(0, 0, 255)
    circle(trainPos, 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])
}
```

Sketch 8.10 training the model

We have the data examples and now we want to train the model

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

function setup()
{
  createCanvas(400, 400)
  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)
      }
    }
    trainPos = slider.value()
    fill(0, 0, 255)
    circle(trainPos, 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({epochs: 50}, finishedTraining)
}
```

Sketch 8.11 finished

When we have finished train we want to predict

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

function setup()
{
  createCanvas(400, 400)
  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)
      }
    }
    trainPos = slider.value()
    fill(0, 0, 255)
    circle(trainPos, 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({epochs: 50}, finishedTraining)
}

function finishedTraining()
{
  predict()
```

```
}
```

Sketch 8.12 predict

Now we need to predict the result after training. We use the inputs from the video, so that when we move our heads from left to right it can predict the position of the circle based on our trained model.

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

function setup()
{
  createCanvas(400, 400)
  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)
      }
    }
    trainPos = slider.value()
    fill(0, 0, 255)
```

```
    circle(trainPos, 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({epochs: 50}, finishedTraining)
}

function finishedTraining()
```

```
{
  predict()
}
```

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

Sketch 8.13 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 draw we need to abandon the blue circle and use the red one instead. We set the predictReady to be true, we get the top result (best confidence) and send the red circle to that position (pos). We predict again because we want to see the red circle move when we move our head from left to right. The circle should follow out movements.

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

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

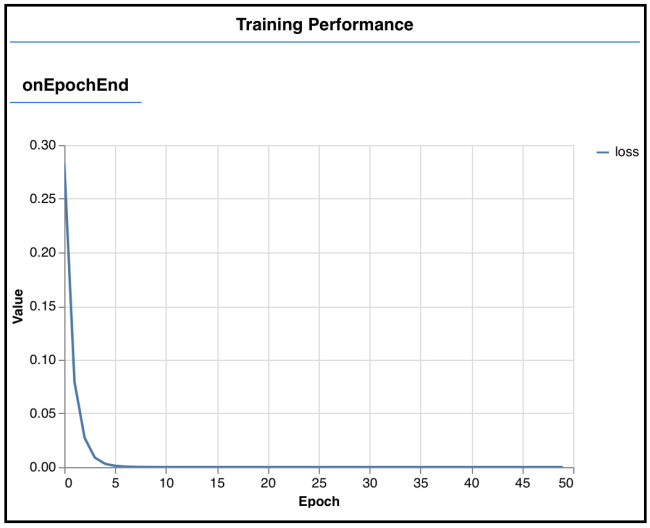
```
        square(x * w, y * w, w)
      }
    }
    if (predictReady == true)
    {
      fill(255, 0, 0)
      circle(pos, height/2, 50)
    }
    else
    {
      trainPos = slider.value()
      fill(0, 0, 255)
      circle(trainPos, height/2, 50)
    }
  }
}

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({epochs: 50}, finishedTraining)
}

function finishedTraining()
{
  predict()
}

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

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

Challenges

1. Make the videoSize smaller (smaller pixels = more of them), add other hyperparameters etc
2. Make other things happen in response to your training

You could get away with a lot fewer epochs but it is still pretty fast



The end result, not bad