Copyright Notice

```
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"> </script>
<script src="webcam.js"></script>
</head>
<body>
  <div>
    <div>
      <video autoplay playsinline muted id="wc" width="224" height="224"></video>
    </div>
  </div>
</body>

<script src="index.js"></script>
</html>
```

```
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"> </script>
<script src="webcam.js"></script>
</head>
<body>
  <div>
    <div>
      <video autoplay playsinline muted id="wc" width="224" height="224"></video>
    </div>
  </div>
</body>

<script src="index.js"></script>
</html>
```

```html
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"> </script>
<script src="webcam.js"></script>
</head>
<body>
  <div>
    <div>
      <video autoplay playsinline muted id="wc" width="224" height="224"></video>
    </div>
  </div>
</body>

<script src="index.js"></script>
</html>
```

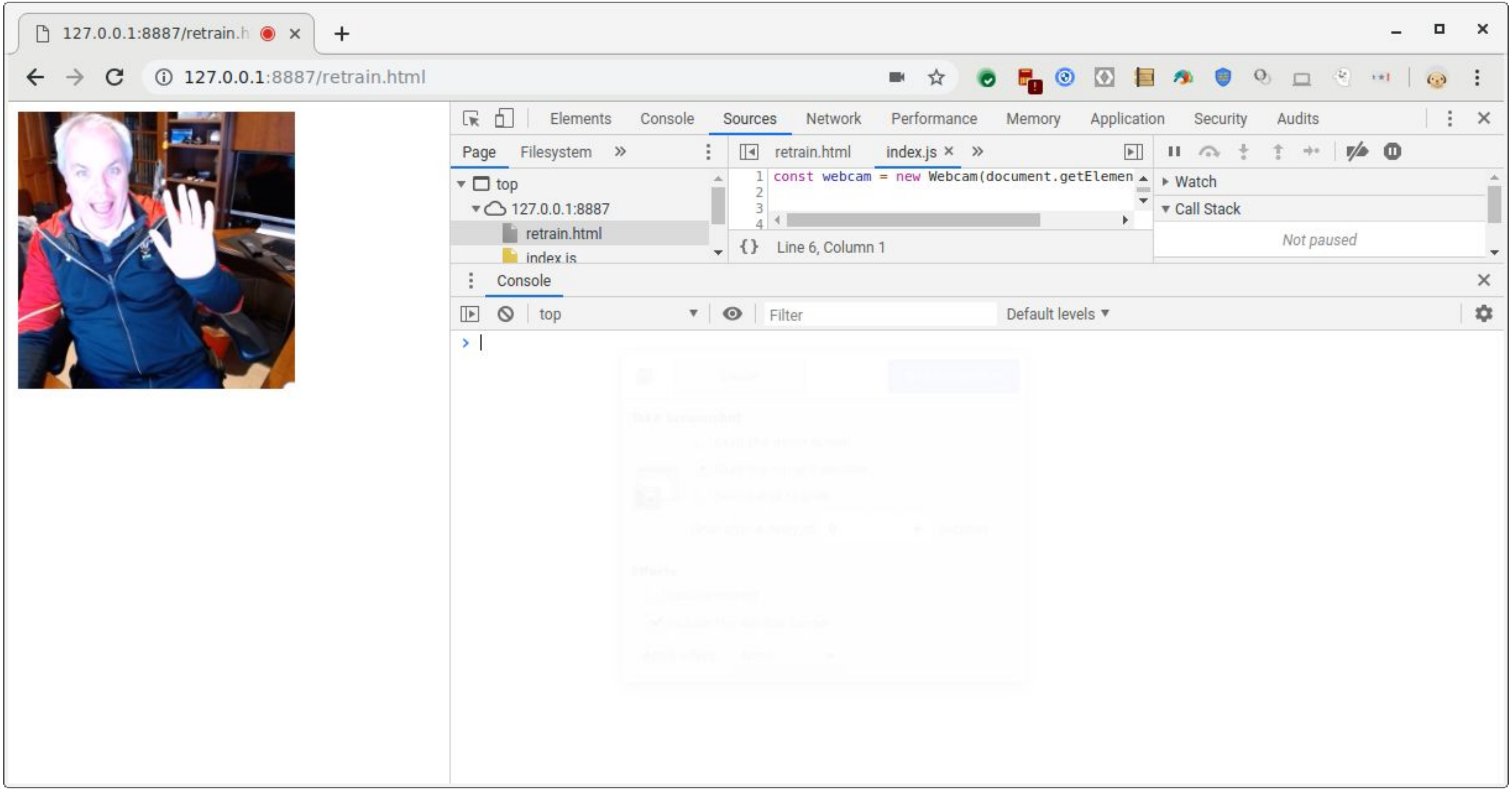
```html
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"> </script>
<script src="webcam.js"></script>
</head>
<body>
  <div>
    <div>
      <video autoplay playsinline muted id="wc" width="224" height="224"></video>
    </div>
  </div>
</body>

<script src="index.js"></script>
</html>
```

```html
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"> </script>
<script src="webcam.js"></script>
</head>
<body>
  <div>
    <div>
      <video autoplay playsinline muted id="wc" width="224" height="224"></video>
    </div>
  </div>
</body>

<script src="index.js"></script>
</html>
```

```
let mobilenet;
let model;
const webcam = new Webcam(document.getElementById('wc'));

async function init(){
    await webcam.setup();
}

init();
```

```
let mobilenet;
let model;
const webcam = new Webcam(document.getElementById('wc'));

async function init(){
    await webcam.setup();
}

init();
```

```javascript
let mobilenet;
let model;
const webcam = new Webcam(document.getElementById('wc'));

async function init(){
    await webcam.setup();
}

init();
```



```
const webcam = new Webcam(document.getElemen
```

Line 6, Column 1

Not paused

```
async function loadMobilenet() {
  const mobilenet = await
               tf.loadLayersModel('https://storage.googleapis.com/tfjs-models
                                           /tfjs/mobilenet_v1_0.25_224/model.json');
  const layer = mobilenet.getLayer('conv_pw_13_relu');
  return tf.model({inputs: mobilenet.inputs, outputs: layer.output});
}
```

```
async function loadMobilenet() {
  const mobilenet = await
               tf.loadLayersModel('https://storage.googleapis.com/tfjs-models
                                           /tfjs/mobilenet_v1_0.25_224/model.json');
  const layer = mobilenet.getLayer('conv_pw_13_relu');
  return tf.model({inputs: mobilenet.inputs, outputs: layer.output});
}
```

```
async function loadMobilenet() {
  const mobilenet = await
               tf.loadLayersModel('https://storage.googleapis.com/tfjs-models
                                           /tfjs/mobilenet_v1_0.25_224/model.json');
  const layer = mobilenet.getLayer('conv_pw_13_relu');
  return tf.model({inputs: mobilenet.inputs, outputs: layer.output});
}
```

```
async function loadMobilenet() {
  const mobilenet = await
             tf.loadLayersModel('https://storage.googleapis.com/tfjs-models
                                         /tfjs/mobilenet_v1_0.25_224/model.json');
  const layer = mobilenet.getLayer('conv_pw_13_relu');
  return tf.model({inputs: mobilenet.inputs, outputs: layer.output});
}
```

```
async function init(){
    await webcam.setup();
    mobilenet = await loadMobilenet();
    tf.tidy(() => mobilenet.predict(webcam.capture()));
}
```

```
async function init(){
    await webcam.setup();
    mobilenet = await loadMobilenet();
    tf.tidy(() => mobilenet.predict(webcam.capture()));
}
```

```
async function init(){
    await webcam.setup();
    mobilenet = await loadMobilenet();
    tf.tidy(() => mobilenet.predict(webcam.capture()));
}
```

```
async function train() {
  model = tf.sequential({
    layers: [
      tf.layers.flatten({inputShape: mobilenet.outputs[0].shape.slice(1)}),
      tf.layers.dense({ units: 100, activation: 'relu'}),
      tf.layers.dense({ units: 3, activation: 'softmax'})
    ]
  });
}
```

```
async function train() {
  model = tf.sequential({
    layers: [
      tf.layers.flatten({inputShape: mobilenet.outputs[0].shape.slice(1)}),
      tf.layers.dense({ units: 100, activation: 'relu'}),
      tf.layers.dense({ units: 3, activation: 'softmax'})
    ]
  });
}
```

```
async function train() {
  model = tf.sequential({
    layers: [
      tf.layers.flatten({inputShape: mobilenet.outputs[0].shape.slice(1)}),
      tf.layers.dense({ units: 100, activation: 'relu'}),
      tf.layers.dense({ units: 3, activation: 'softmax'})
    ]
  });
}
```

```
async function train() {
  model = tf.sequential({
    layers: [
      tf.layers.flatten({inputShape: mobilenet.outputs[0].shape.slice(1)}),
      tf.layers.dense({ units: 100, activation: 'relu'}),
      tf.layers.dense({ units: 3, activation: 'softmax'})
    ]
  });
}
```

```
async function train() {
  model = tf.sequential({
    layers: [
      tf.layers.flatten({inputShape: mobilenet.outputs[0].shape.slice(1)}),
      tf.layers.dense({ units: 100, activation: 'relu'}),
      tf.layers.dense({ units: 3, activation: 'softmax'})
    ]
  });
}
```

```
const embeddings = mobilenet.predict(img);
const predictions = model.predict(embeddings);
```

```html
<button type="button" id="0" onclick="handleButton(this)" >Rock</button>
<button type="button" id="1" onclick="handleButton(this)" >Paper</button>
<button type="button" id="2" onclick="handleButton(this)" >Scissors</button>
<div id="rocksamples">Rock Samples:</div>
<div id="papersamples">Paper Samples:</div>
<div id="scissorssamples">Scissors Samples:</div>
<button type="button" id="train" onclick="doTraining()" >Train Network</button>
```

```html
<button type="button" id="0" onclick="handleButton(this)" >Rock</button>
<button type="button" id="1" onclick="handleButton(this)" >Paper</button>
<button type="button" id="2" onclick="handleButton(this)" >Scissors</button>
<div id="rocksamples">Rock Samples:</div>
<div id="papersamples">Paper Samples:</div>
<div id="scissorssamples">Scissors Samples:</div>
<button type="button" id="train" onclick="doTraining()" >Train Network</button>
```

```html
<button type="button" id="0" onclick="handleButton(this)" >Rock</button>
<button type="button" id="1" onclick="handleButton(this)" >Paper</button>
<button type="button" id="2" onclick="handleButton(this)" >Scissors</button>
<div id="rocksamples">Rock Samples:</div>
<div id="papersamples">Paper Samples:</div>
<div id="scissorssamples">Scissors Samples:</div>
<button type="button" id="train" onclick="doTraining()" >Train Network</button>
```

```
<button type="button" id="0" onclick="handleButton(this)" >Rock</button>
<button type="button" id="1" onclick="handleButton(this)" >Paper</button>
<button type="button" id="2" onclick="handleButton(this)" >Scissors</button>
<div id="rocksamples">Rock Samples:</div>
<div id="papersamples">Paper Samples:</div>
<div id="scissorssamples">Scissors Samples:</div>
<button type="button" id="train" onclick="doTraining()" >Train Network</button>
```

```
<button type="button" id="0" onclick="handleButton(this)" >Rock</button>
<button type="button" id="1" onclick="handleButton(this)" >Paper</button>
<button type="button" id="2" onclick="handleButton(this)" >Scissors</button>
<div id="rocksamples">Rock Samples:</div>
<div id="papersamples">Paper Samples:</div>
<div id="scissorssamples">Scissors Samples:</div>
<button type="button" id="train" onclick="doTraining()" >Train Network</button>
```

```javascript
function handleButton(elem){
  switch(elem.id){
  case "0":
    rockSamples++;
    document.getElementById("rocksamples").innerText = "Rock samples:" + rockSamples;
    break;
  case "1":
    paperSamples++;
    document.getElementById("papersamples").innerText = "Paper samples:" + paperSamples;
    break;
  case "2":
    scissorsSamples++;
    document.getElementById("scissorssamples").innerText = "Scissors samples:" + scissorsSamples;
    break;
  }
  label = parseInt(elem.id);
  const img = webcam.capture();
  dataset.addExample(mobilenet.predict(img), label);
}
```

```
function handleButton(elem){
  switch(elem.id){
  case "0":
    rockSamples++;
    document.getElementById("rocksamples").innerText = "Rock samples:" + rockSamples;
    break;
  case "1":
    paperSamples++;
    document.getElementById("papersamples").innerText = "Paper samples:" + paperSamples;
    break;
  case "2":
    scissorsSamples++;
    document.getElementById("scissorssamples").innerText = "Scissors samples:" + scissorsSamples;
    break;
  }
  label = parseInt(elem.id);
  const img = webcam.capture();
  dataset.addExample(mobilenet.predict(img), label);
}
```

```
function handleButton(elem){
  switch(elem.id){
  case "0":
    rockSamples++;
    document.getElementById("rocksamples").innerText = "Rock samples:" + rockSamples;
    break;
  case "1":
    paperSamples++;
    document.getElementById("papersamples").innerText = "Paper samples:" + paperSamples;
    break;
  case "2":
    scissorsSamples++;
    document.getElementById("scissorssamples").innerText = "Scissors samples:" + scissorsSamples;
    break;
  }
  label = parseInt(elem.id);
  const img = webcam.capture();
  dataset.addExample(mobilenet.predict(img), label);
}
```

```
function handleButton(elem){
  switch(elem.id){
  case "0":
    rockSamples++;
    document.getElementById("rocksamples").innerText = "Rock samples:" + rockSamples;
    break;
  case "1":
    paperSamples++;
    document.getElementById("papersamples").innerText = "Paper samples:" + paperSamples;
    break;
  case "2":
    scissorsSamples++;
    document.getElementById("scissorssamples").innerText = "Scissors samples:" + scissorsSamples;
    break;
  }
  label = parseInt(elem.id);
  const img = webcam.capture();
  dataset.addExample(mobilenet.predict(img), label);
}
```

```
function handleButton(elem){
  switch(elem.id){
  case "0":
    rockSamples++;
    document.getElementById("rocksamples").innerText = "Rock samples:" + rockSamples;
    break;
  case "1":
    paperSamples++;
    document.getElementById("papersamples").innerText = "Paper samples:" + paperSamples;
    break;
  case "2":
    scissorsSamples++;
    document.getElementById("scissorssamples").innerText = "Scissors samples:" + scissorsSamples;
    break;
  }
  label = parseInt(elem.id);
  const img = webcam.capture();
  dataset.addExample(mobilenet.predict(img), label);
}
```

```
function handleButton(elem){
  switch(elem.id){
  case "0":
    rockSamples++;
    document.getElementById("rocksamples").innerText = "Rock samples:" + rockSamples;
    break;
  case "1":
    paperSamples++;
    document.getElementById("papersamples").innerText = "Paper samples:" + paperSamples;
    break;
  case "2":
    scissorsSamples++;
    document.getElementById("scissorssamples").innerText = "Scissors samples:" + scissorsSamples;
    break;
  }
  label = parseInt(elem.id);
  const img = webcam.capture();
  dataset.addExample(mobilenet.predict(img), label);
}
```

```
function handleButton(elem){
  switch(elem.id){
  case "0":
    rockSamples++;
    document.getElementById("rocksamples").innerText = "Rock samples:" + rockSamples;
    break;
  case "1":
    paperSamples++;
    document.getElementById("papersamples").innerText = "Paper samples:" + paperSamples;
    break;
  case "2":
    scissorsSamples++;
    document.getElementById("scissorssamples").innerText = "Scissors samples:" + scissorsSamples;
    break;
  }
  label = parseInt(elem.id);
  const img = webcam.capture();
  dataset.addExample(mobilenet.predict(img), label);
}
```

```
function handleButton(elem){
  switch(elem.id){
  case "0":
    rockSamples++;
    document.getElementById("rocksamples").innerText = "Rock samples:" + rockSamples;
    break;
  case "1":
    paperSamples++;
    document.getElementById("papersamples").innerText = "Paper samples:" + paperSamples;
    break;
  case "2":
    scissorsSamples++;
    document.getElementById("scissorssamples").innerText = "Scissors samples:" + scissorsSamples;
    break;
  }
  label = parseInt(elem.id);
  const img = webcam.capture();
  dataset.addExample(mobilenet.predict(img), label ;
}
```

```
<script src="rps-dataset.js"></script>
```

```
const dataset = new RPSDataset();
```

```
class RPSDataset {
  constructor() {
    this.labels = []
  }
  addExample(example, label) {
    if (this.xs == null) {
      this.xs = tf.keep(example);
      this.labels.push(label);
    } else {
      const oldX = this.xs;
      this.xs = tf.keep(oldX.concat(example, 0));
      this.labels.push(label);
      oldX.dispose();
    }
  }
  encodeLabels(numClasses) {
    ...
  }
}
```

```
class RPSDataset {
  constructor() {
    this.labels = []
  }
  addExample(example, label) {
    if (this.xs == null) {
      this.xs = tf.keep(example);
      this.labels.push(label);
    } else {
      const oldX = this.xs;
      this.xs = tf.keep(oldX.concat(example, 0));
      this.labels.push(label);
      oldX.dispose();
    }
  }
  encodeLabels(numClasses) {
    ...
  }
}
```

```
class RPSDataset {
  constructor() {
    this.labels = []
  }
  addExample(example, label) {
    if (this.xs == null) {
      this.xs = tf.keep(example);
      this.labels.push(label);
    } else {
      const oldX = this.xs;
      this.xs = tf.keep(oldX.concat(example, 0));
      this.labels.push(label);
      oldX.dispose();
    }
  }
  encodeLabels(numClasses) {
    ...
  }
}
```

```
class RPSDataset {
  constructor() {
    this.labels = []
  }
  addExample(example, label) {
    if (this.xs == null) {
      this.xs = tf.keep(example);
      this.labels.push(label);
    } else {
      const oldX = this.xs;
      this.xs = tf.keep(oldX.concat(example, 0));
      this.labels.push(label);
      oldX.dispose();
    }
  }
  encodeLabels(numClasses) {
    ...
  }
}
```

```
class RPSDataset {
  constructor() {
    this.labels = []
  }
  addExample(example, label) {
    if (this.xs == null) {
      this.xs = tf.keep(example);
      this.labels.push(label);
    } else {
      const oldX = this.xs;
      this.xs = tf.keep(oldX.concat(example, 0));
      this.labels.push(label);
      oldX.dispose();
    }
  }
  encodeLabels(numClasses) {
    ...
  }
}
```

```
class RPSDataset {
 constructor() {
  this.labels = []
 }
 addExample(example, label) {
  if (this.xs == null) {
   this.xs = tf.keep(example);
   this.labels.push(label);
  } else {
   const oldX = this.xs;
   this.xs = tf.keep(oldX.concat(example, 0));
   this.labels.push(label);
   oldX.dispose();
  }
 }
 encodeLabels(numClasses) {
  ...
 }
}
```

```
class RPSDataset {
  constructor() {
    this.labels = []
  }
  addExample(example, label) {
    if (this.xs == null) {
      this.xs = tf.keep(example);
      this.labels.push(label);
    } else {
      const oldX = this.xs;
      this.xs = tf.keep(oldX.concat(example, 0));
      this.labels.push(label);
      oldX.dispose();
    }
  }
  encodeLabels(numClasses) {
    ...
  }
}
```

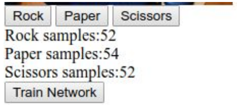
```
async function train() {
  dataset.ys = null;
  dataset.encodeLabels(3);
  model = tf.sequential({
    layers: [
      tf.layers.flatten({inputShape: mobilenet.outputs[0].shape.slice(1)}),
      tf.layers.dense({ units: 100, activation: 'relu'}),
      tf.layers.dense({ units: 3, activation: 'softmax'})
    ]
  });
  const optimizer = tf.train.adam(0.0001);
  model.compile({optimizer: optimizer, loss: 'categoricalCrossentropy'});
  let loss = 0;
  model.fit(dataset.xs, dataset.ys, {
    epochs: 10,
    callbacks: {
      onBatchEnd: async (batch, logs) => {
        loss = logs.loss.toFixed(5);
        console.log('LOSS: ' + loss);
        }
      }
   });
}
```

```
async function train() {
  dataset.ys = null;
  dataset.encodeLabels(3);
  model = tf.sequential({
    layers: [
      tf.layers.flatten({inputShape: mobilenet.outputs[0].shape.slice(1)}),
      tf.layers.dense({ units: 100, activation: 'relu'}),
      tf.layers.dense({ units: 3, activation: 'softmax'})
    ]
  });
  const optimizer = tf.train.adam(0.0001);
  model.compile({optimizer: optimizer, loss: 'categoricalCrossentropy'});
  let loss = 0;
  model.fit(dataset.xs, dataset.ys, {
    epochs: 10,
    callbacks: {
      onBatchEnd: async (batch, logs) => {
        loss = logs.loss.toFixed(5);
        console.log('LOSS: ' + loss);
        }
      }
   });
}
```

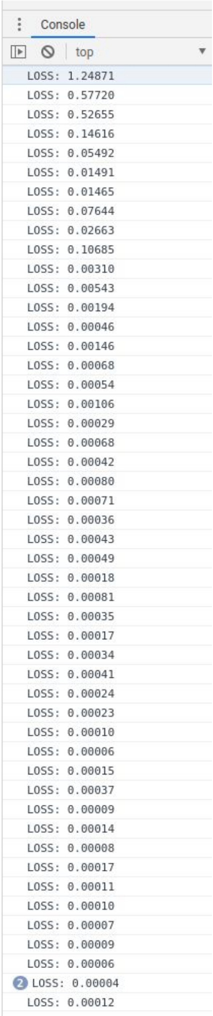
```
async function train() {
  dataset.ys = null;
  dataset.encodeLabels(3);
  model = tf.sequential({
    layers: [
      tf.layers.flatten({inputShape: mobilenet.outputs[0].shape.slice(1)}),
      tf.layers.dense({ units: 100, activation: 'relu'}),
      tf.layers.dense({ units: 3, activation: 'softmax'})
    ]
  });
  const optimizer = tf.train.adam(0.0001);
  model.compile({optimizer: optimizer, loss: 'categoricalCrossentropy'});
  let loss = 0;
  model.fit(dataset.xs, dataset.ys, {
    epochs: 10,
    callbacks: {
      onBatchEnd: async (batch, logs) => {
        loss = logs.loss.toFixed(5);
        console.log('LOSS: ' + loss);
        }
      }
   });
}
```

```
async function train() {
  dataset.ys = null;
  dataset.encodeLabels(3);
  model = tf.sequential({
    layers: [
      tf.layers.flatten({inputShape: mobilenet.outputs[0].shape.slice(1)}),
      tf.layers.dense({ units: 100, activation: 'relu'}),
      tf.layers.dense({ units: 3, activation: 'softmax'})
    ]
  });
  const optimizer = tf.train.adam(0.0001);
  model.compile({optimizer: optimizer, loss: 'categoricalCrossentropy'});
  let loss = 0;
  model.fit(dataset.xs, dataset.ys, {
    epochs: 10,
    callbacks: {
      onBatchEnd: async (batch, logs) => {
        loss = logs.loss.toFixed(5);
        console.log('LOSS: ' + loss);
        }
      }
   });
}
```

```
async function train() {
  dataset.ys = null;
  dataset.encodeLabels(3);
  model = tf.sequential({
    layers: [
      tf.layers.flatten({inputShape: mobilenet.outputs[0].shape.slice(1)}),
      tf.layers.dense({ units: 100, activation: 'relu'}),
      tf.layers.dense({ units: 3, activation: 'softmax'})
    ]
  });
  const optimizer = tf.train.adam(0.0001);
  model.compile({optimizer: optimizer, loss: 'categoricalCrossentropy'});
  let loss = 0;
  model.fit(dataset.xs, dataset.ys, {
    epochs: 10,
    callbacks: {
      onBatchEnd: async (batch, logs) => {
        loss = logs.loss.toFixed(5);
        console.log('LOSS: ' + loss);
        }
      }
   });
}
```



Rock samples:52
Paper samples:54
Scissors samples:52



```
LOSS: 1.24871
LOSS: 0.57720
LOSS: 0.52655
LOSS: 0.14616
LOSS: 0.05492
LOSS: 0.01491
LOSS: 0.01465
LOSS: 0.07644
LOSS: 0.02663
LOSS: 0.10685
LOSS: 0.00310
LOSS: 0.00543
LOSS: 0.00194
LOSS: 0.00046
LOSS: 0.00146
LOSS: 0.00068
LOSS: 0.00054
LOSS: 0.00106
LOSS: 0.00029
LOSS: 0.00068
LOSS: 0.00042
LOSS: 0.00080
LOSS: 0.00071
LOSS: 0.00036
LOSS: 0.00043
LOSS: 0.00049
LOSS: 0.00018
LOSS: 0.00081
LOSS: 0.00035
LOSS: 0.00017
LOSS: 0.00034
LOSS: 0.00041
LOSS: 0.00024
LOSS: 0.00023
LOSS: 0.00010
LOSS: 0.00006
LOSS: 0.00015
LOSS: 0.00037
LOSS: 0.00009
LOSS: 0.00014
LOSS: 0.00008
LOSS: 0.00017
LOSS: 0.00011
LOSS: 0.00010
LOSS: 0.00007
LOSS: 0.00009
LOSS: 0.00006
2 LOSS: 0.00004
LOSS: 0.00012
```

```
<div id="dummy">Once training is complete, click 'Start Predicting' to see
predictions, and 'Stop Predicting' to end</div>

<button type="button" id="startPredicting" onclick="startPredicting()" >
    Start Predicting</button>

<button type="button" id="stopPredicting" onclick="stopPredicting()" >
    Stop Predicting</button>


<div id="prediction"></div>
```

```html
<div id="dummy">Once training is complete, click 'Start Predicting' to see
predictions, and 'Stop Predicting' to end</div>

<button type="button" id="startPredicting" onclick="startPredicting()" >
    Start Predicting</button>

<button type="button" id="stopPredicting" onclick="stopPredicting()" >
    Stop Predicting</button>


<div id="prediction"></div>
```

```
<div id="dummy">Once training is complete, click 'Start Predicting' to see
predictions, and 'Stop Predicting' to end</div>

<button type="button" id="startPredicting" onclick="startPredicting()" >
    Start Predicting</button>

<button type="button" id="stopPredicting" onclick="stopPredicting()" >
    Stop Predicting</button>


<div id="prediction"></div>
```

```html
<div id="dummy">Once training is complete, click 'Start Predicting' to see
predictions, and 'Stop Predicting' to end</div>

<button type="button" id="startPredicting" onclick="startPredicting()" >
    Start Predicting</button>

<button type="button" id="stopPredicting" onclick="stopPredicting()" >
    Stop Predicting</button>

<div id="prediction"></div>
```

```
function startPredicting(){
    isPredicting = true;
    predict();
}
```

```
function startPredicting(){
    isPredicting = true;
    predict();
}
```

```
function startPredicting(){
    isPredicting = true;
    predict();
}
```

```
function stopPredicting(){
    isPredicting = false;
    predict();
}
```

```
function stopPredicting(){
    isPredicting = false;
    predict();
}
```

```
async function predict() {
  while (isPredicting) {
    // Do stuff
    }
    predictedClass.dispose();
    await tf.nextFrame();
  }
}
```

```
while (isPredicting) {
    // Step 1: Get Prediciton


    // Step 2: Evaluate Prediction and Update UI


    // Step 3: Cleanup


  }
```

```
while (isPredicting) {
    // Step 1: Get Prediciton

    // Step 2: Evaluate Prediction and Update UI

    // Step 3: Cleanup

  }
```

```
while (isPredicting) {
    // Step 1: Get Prediciton

    // Step 2: Evaluate Prediction and Update UI

    // Step 3: Cleanup

  }
```

```
while (isPredicting) {
    // Step 1: Get Prediciton

    // Step 2: Evaluate Prediction and Update UI

    // Step 3: Cleanup

  }
```

```
while (isPredicting) {
    // Step 1: Get Prediciton

    // Step 2: Evaluate Prediction and Update UI

    // Step 3: Cleanup

  }
```

```
const predictedClass = tf.tidy(() => {
  const img = webcam.capture();
  const activation = mobilenet.predict(img);
  const predictions = model.predict(activation);
  return predictions.as1D().argMax();
});
```

```
const predictedClass = tf.tidy(() => {
  const img = webcam.capture();
  const activation = mobilenet.predict(img);
  const predictions = model.predict(activation);
  return predictions.as1D().argMax();
});
```

```
const predictedClass = tf.tidy(() => {
  const img = webcam.capture();
  const activation = mobilenet.predict(img);
  const predictions = model.predict(activation);
  return predictions.as1D().argMax();
});
```

```
const predictedClass = tf.tidy(() => {
  const img = webcam.capture();
  const activation = mobilenet.predict(img);
  const predictions = model.predict(activation);
  return predictions.as1D().argMax();
});
```

```
const predictedClass = tf.tidy(() => {
  const img = webcam.capture();
  const activation = mobilenet.predict(img);
  const predictions = model.predict(activation);
  return predictions.as1D().argMax();
});
```

```
const predictedClass = tf.tidy(() => {
  const img = webcam.capture();
  const activation = mobilenet.predict(img);
  const predictions = model.predict(activation);
  return predictions.as1D().argMax();
});
```

```javascript
const classId = (await predictedClass.data())[0];
var predictionText = "";
switch(classId){
    case 0:
        predictionText = "I see Rock";
        break;
    case 1:
        predictionText = "I see Paper";
        break;
    case 2:
        predictionText = "I see Scissors";
        break;
}
document.getElementById("prediction").innerText = predictionText;
```

```
const classId = (await predictedClass.data())[0];
var predictionText = "";
switch(classId){
    case 0:
        predictionText = "I see Rock";
        break;
    case 1:
        predictionText = "I see Paper";
        break;
    case 2:
        predictionText = "I see Scissors";
        break;
}
document.getElementById("prediction").innerText = predictionText;
```

```
const classId = (await predictedClass.data())[0];
var predictionText = "";
switch(classId){
    case 0:
        predictionText = "I see Rock";
        break;
    case 1:
        predictionText = "I see Paper";
        break;
    case 2:
        predictionText = "I see Scissors";
        break;
}
document.getElementById("prediction").innerText = predictionText;
```

```
const classId = (await predictedClass.data())[0];
var predictionText = "";
switch(classId){
    case 0:
        predictionText = "I see Rock";
        break;
    case 1:
        predictionText = "I see Paper";
        break;
    case 2:
        predictionText = "I see Scissors";
        break;
}
document.getElementById("prediction").innerText = predictionText;
```

```
predictedClass.dispose();
await tf.nextFrame();
```