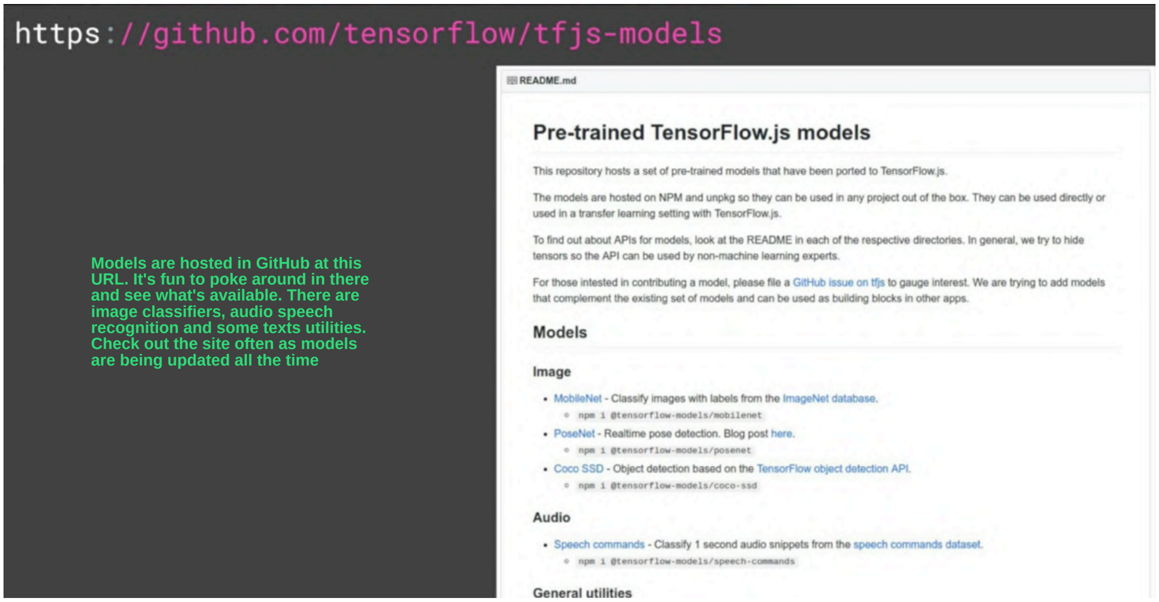
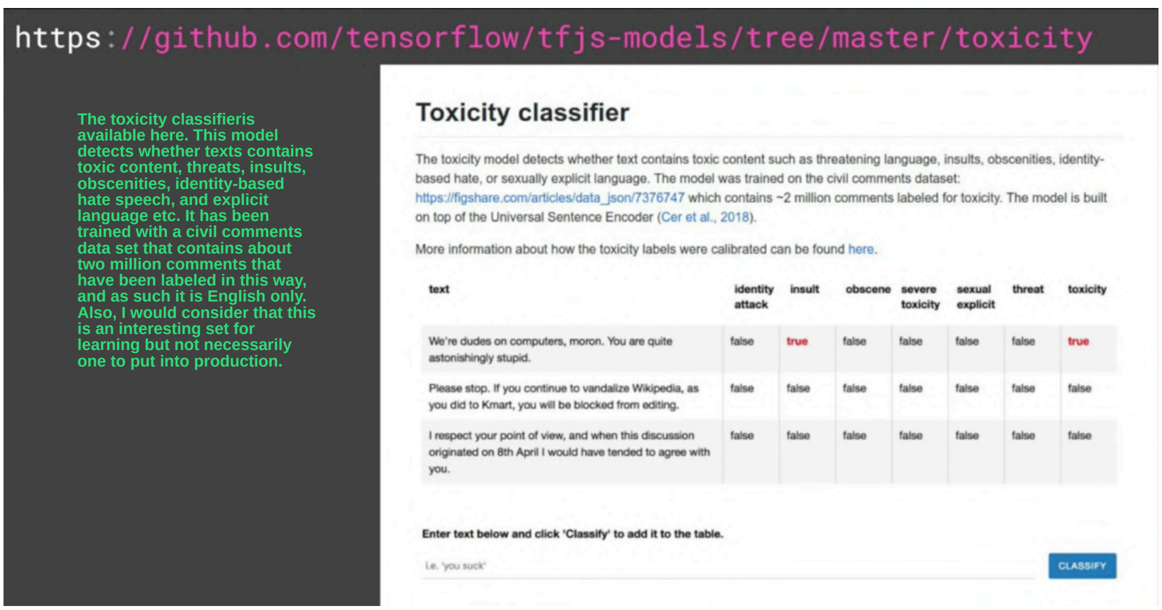




Toxicity Classifier

In the next example, we will use the pre-trained Toxicity model to detect whether a given piece of text contains toxic content such as threatening language, insults, obscenities, identity-based hate, or sexually explicit language.

You can use Brackets to open the **toxicity.html** file and take a look at the code. You can find the **toxicity.html** file in the following folder in the GitHub repository for this course:

[dlaicourse/TensorFlow Deployment/Course 1 - TensorFlow-JS/Week 3/Examples/]()

When you launch the **toxicity.html** file in the Chrome browser make sure to open the Developer Tools to see the output in the Console.

Here are the ones to get tensorflow.js latest and the toxicity model. I often get questions about how one can find these models. It's kind of hard to search for them if you don't know what they are. So the rule of thumb that I would recommend is to take a look at the URL like this, and then just take the name of the model at the end, and then go back to the GitHub we shared earlier and look at the models, and hey they match.

```
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow-models/toxicity"></script>
```

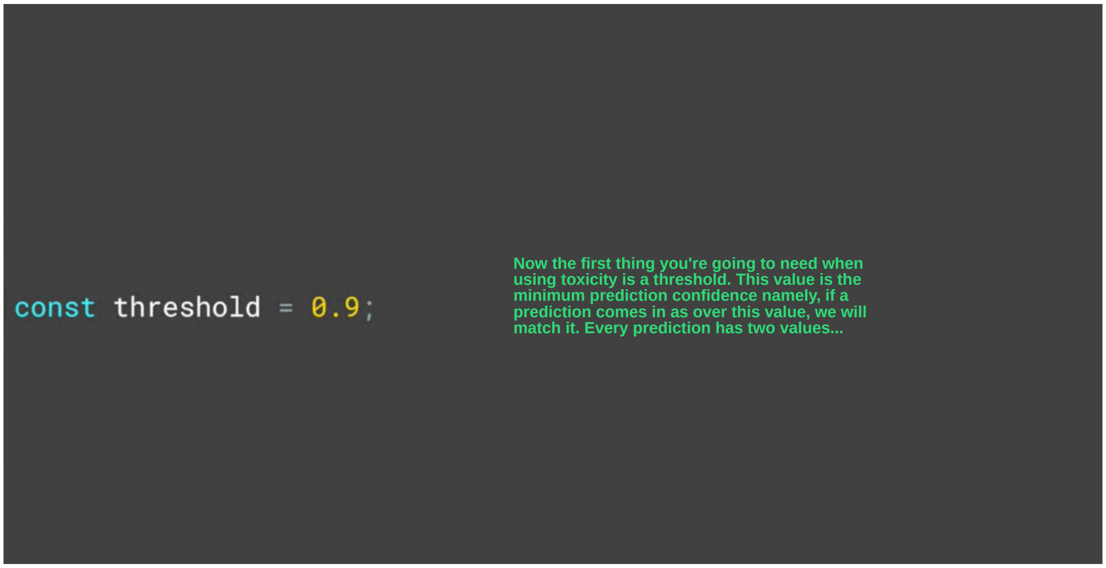
https://github.com/tensorflow/tfjs-models

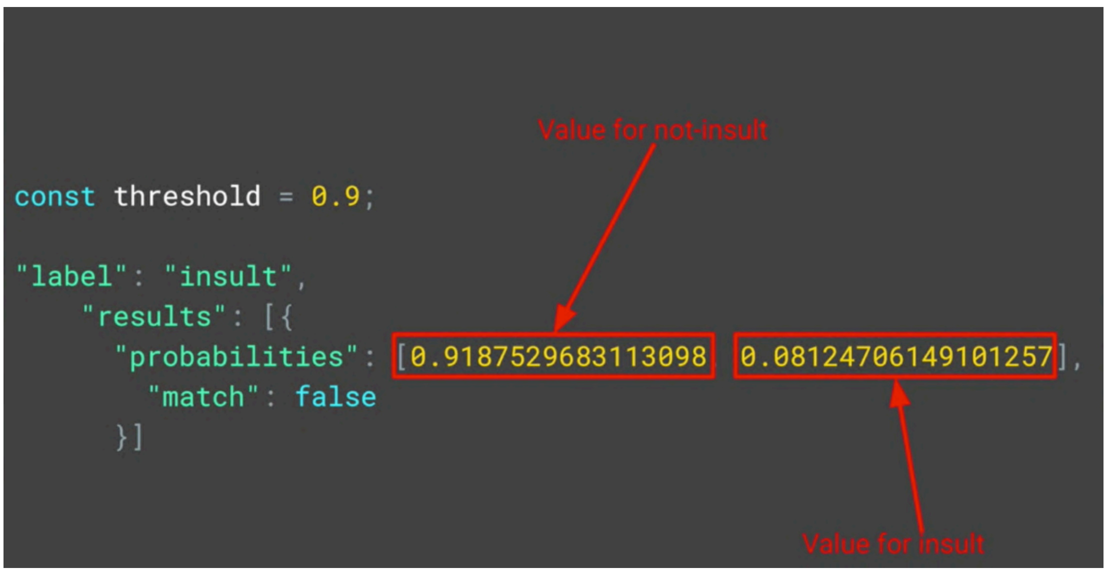
Here's toxicity. Similarly, if you want to look at the universal sentence encoder, we'll know what the URL of the script for that will look like.

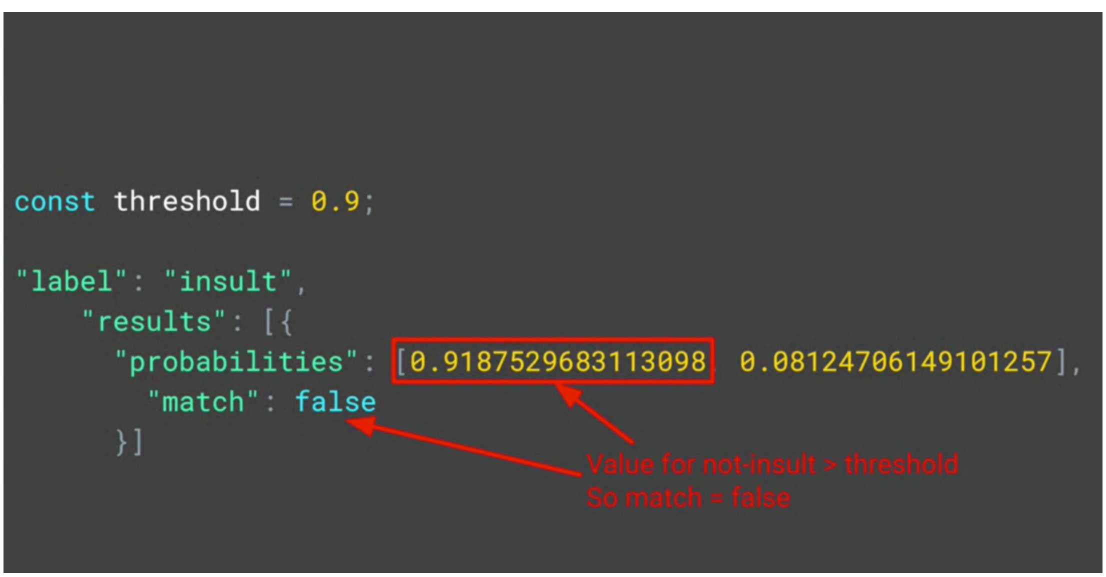
body-pix	Fix typo in source code (#183)	5 days ago
coco-ssd	Update bodypix, cocossd, knn-classifier, posenet to depend on tfjs 1.0 (	a month ago
knn-classifier	Update bodypix, cocossd, knn-classifier, posenet to depend on tfjs 1.0 (	a month ago
mobilenet	Update versions of tfjs and mobilenet in the example code (#174)	18 days ago
posenet	Update bodypix, cocossd, knn-classifier, posenet to depend on tfjs 1.0 (	a month ago
speech-commands	[speech-commands] Fix incorrect metadata field for word labels; v0.3.4 (	an hour ago
toxicity	Update toxicity demo per reviewer feedback. (#172)	22 days ago
universal-sentence-encoder	Depend on tfjs 1.0 in USE. (#164)	a month ago

Here's a super simple HTML page containing the scripts.

```
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow-models/toxicity"></script>
<script>
// Your code here
</script>
</head>
<body></body>
</html>
```


const threshold = 0.9;
Now the first thing you're going to need when using toxicity is a threshold. This value is the minimum prediction confidence namely, if a prediction comes in as over this value, we will match it. Every prediction has two values...


Value for not-insult
const threshold = 0.9;
"label": "insult",
"results": [{
"probabilities": [0.9187529683113098, 0.08124706149101257],
"match": false
}]
Value for insult


const threshold = 0.9;
"label": "insult",
"results": [{
"probabilities": [0.9187529683113098, 0.08124706149101257],
"match": false
}]
Value for not-insult > threshold
So match = false

```
const threshold = 0.9;

"label": "insult",
    "results": [{
      "probabilities": [0.08124706149101257, 0.9187529683113098],
        "match": true
      }]
```

Value for insult > threshold
So match = true

```
const threshold = 0.9;

"label": "insult",
    "results": [{
      "probabilities": [0.5, 0.5],
        "match": null
      }]
```

Neither value > threshold
So match = null

Let's see how to do a prediction on a sentence.
Here's the code and we'll unpack it line by line.

First, we load the model, passing it the threshold value that we just specified to initialize it.

```
toxicity.load(threshold).then(model => {
  const sentences = ['you suck!'];
  model.classify(sentences).then(predictions => {
  // Handle Results
  });
});
```

Once it's loaded, we'll have a model.

```
toxicity.load(threshold).then(model => {
  const sentences = ['you suck!'];
  model.classify(sentences).then(predictions => {
  // Handle Results
  });
});
```

We'll then call model.classify parsing it the sentences.

```
toxicity.load(threshold).then(model => {
  const sentences = ['you suck!'];
  model.classify(sentences).then(predictions => {
  // Handle Results
  });
});
```

Then we'll get a set of predictions back that we can handle.

```
toxicity.load(threshold).then(model => {
  const sentences = ['you suck!'];
  model.classify(sentences).then(predictions => {
  // Handle Results
  });
});
```

```
▼ (7) [{…}, {…}, {…}, {…}, {…}, {…}, {…}]
  ▶ 0: {label: "identity_attack", results: Array(1)}
  ▶ 1: {label: "insult", results: Array(1)}
  ▶ 2: {label: "obscene", results: Array(1)}
  ▶ 3: {label: "severe_toxicity", results: Array(1)}
  ▶ 4: {label: "sexual_explicit", results: Array(1)}
  ▶ 5: {label: "threat", results: Array(1)}
  ▶ 6: {label: "toxicity", results: Array(1)}
    length: 7
  ▶ __proto__: Array(0)
```

```
▼ 1:
    label: "insult"
  ▼ results: Array(1)
    ▼ 0:
        match: true
      ▶ probabilities: Float32Array(2) [0.05890671908855438, 0.94109326601028…
      ▶ __proto__: Object
      length: 1
    ▶ __proto__: Array(0)
  ▶ __proto__: Object
```

```
predictions[1].label;                               "label": "insult",
predictions[1].results[0].probabilities[0];           "results": [{
predictions[1].results[0].probabilities[1];             "probabilities": [0.5, 0.5],
predictions[1].results[0].match;                          "match": null
                                                        }]
```

let's write some codes to iterate across all labels and report back on the ones with a match was true.

```
for(i=0; i<7; i++){
    if(predictions[i].results[0].match){
        console.log(predictions[i].label + " was found with a probability of " +
                    predictions[i].results[0].probabilities[1]);
    }
}
```

```
insult was found with a probability of 0.9410932660102844
toxicity was found with a probability of 0.9766321778297424
```

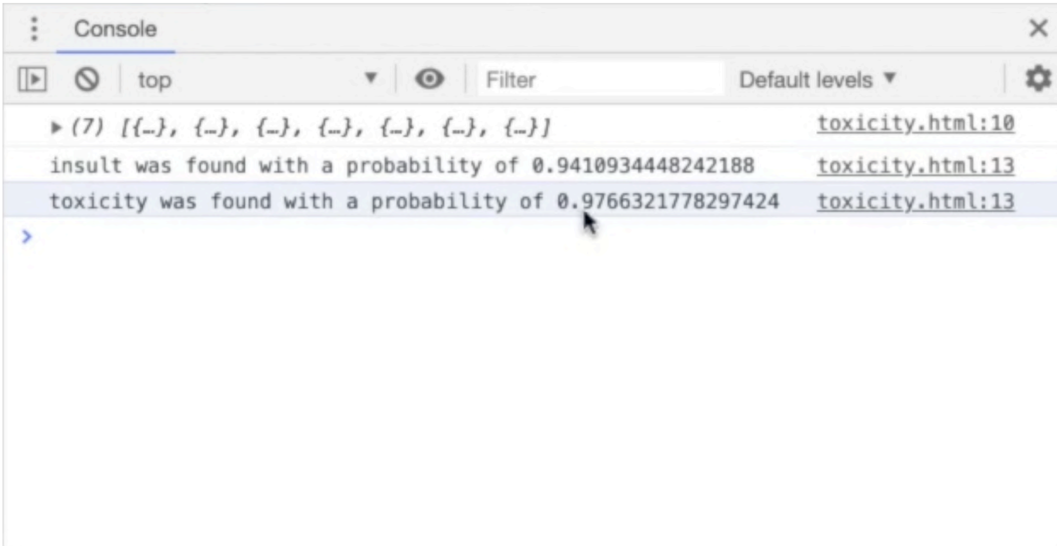
Working Files

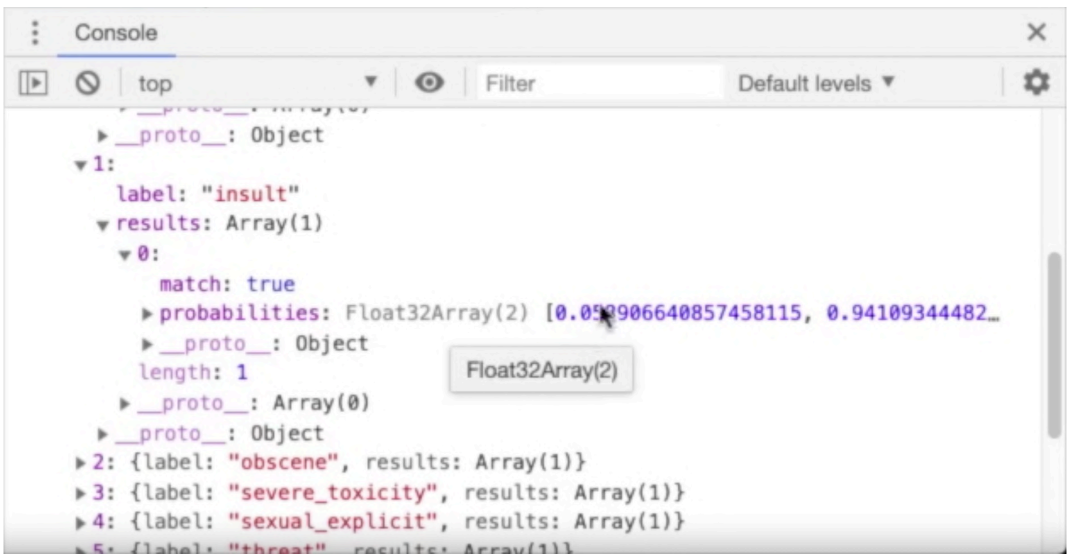
toxicity.html

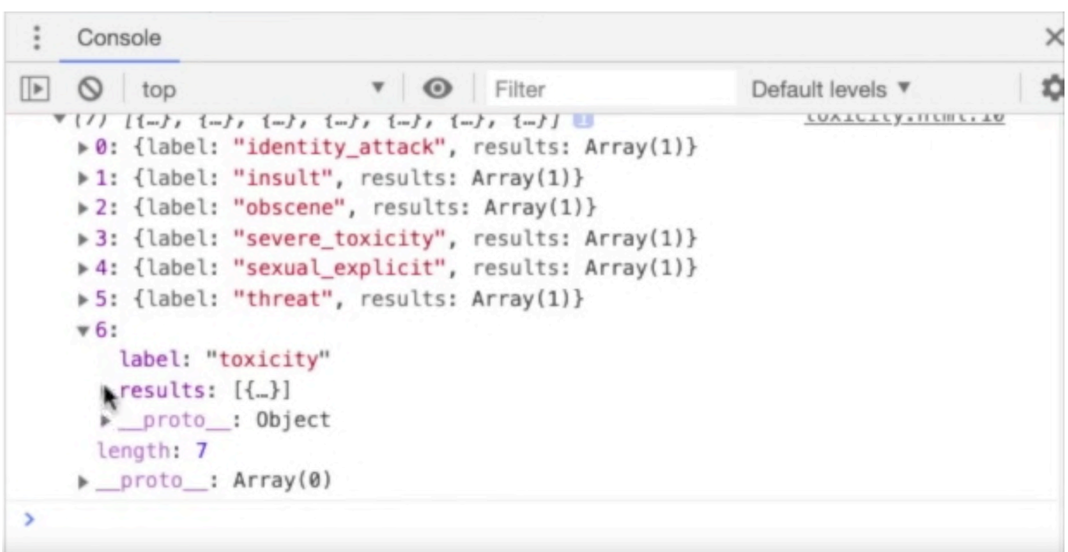
week3

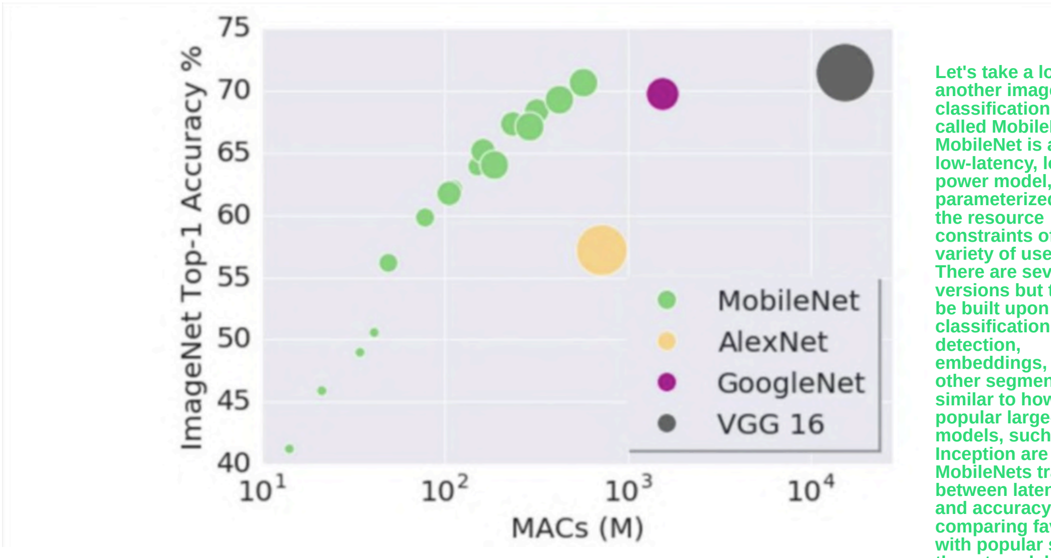
toxicity.html

```
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow-models/toxicity"></script>
<script>
const threshold = 0.9;
toxicity.load(threshold).then(model => {
    const sentences = ['you suck!'];
    model.classify(sentences).then(predictions => {
        console.log(predictions);
        for(i=0; i<7; i++){
            if(predictions[i].results[0].match){
                console.log(predictions[i].label +
                    " was found with a probability of " +
                    predictions[i].results[0].probabilities[1]);
            }
        }
    });
});
</script>
</head>
<body></body>
</html>
```


Console
top
Filter
Default levels
(7) [{…}, {…}, {…}, {…}, {…}, {…}, {…}]
toxicity.html:10
insult was found with a probability of 0.9410934448242188
toxicity.html:13
toxicity was found with a probability of 0.9766321778297424
toxicity.html:13


Console
top
Filter
Default levels
__proto__: Object
1:
label: "insult"
results: Array(1)
0:
match: true
probabilities: Float32Array(2) [0.05890664085745811 5, 0.94109344482…
__proto__: Object
length: 1
Float32Array(2)
__proto__: Array(0)
__proto__: Object
2: {label: "obscene", results: Array(1)}
3: {label: "severe_toxicity", results: Array(1)}
4: {label: "sexual_explicit", results: Array(1)}


Console
top
Filter
Default levels
0: {label: "identity_attack", results: Array(1)}
1: {label: "insult", results: Array(1)}
2: {label: "obscene", results: Array(1)}
3: {label: "severe_toxicity", results: Array(1)}
4: {label: "sexual_explicit", results: Array(1)}
5: {label: "threat", results: Array(1)}
6:
label: "toxicity"
results: [{…}]
__proto__: Object
length: 7
__proto__: Array(0)



Let's take a look at another image classification library called MobileNet. MobileNet is a small low-latency, low-power model, parameterized to meet the resource constraints of a variety of use cases. There are several versions but they can be built upon for classification, detection, embeddings, and other segmentation similar to how other popular large-scale models, such as Inception are used. MobileNets trade-off between latency, size, and accuracy while comparing favorably with popular state of the art models.

Image Classification Using MobileNet

In the next example, we will use the pre-trained MobileNet model to classify images in the browser.

You can use Brackets to open the **mobilenet.html** file and take a look at the code. You can find the **mobilenet.html** file in the following folder in the GitHub repository for this course:

dlaicourse/TensorFlow Deployment/Course 1 - TensorFlow-JS/Week 3/Examples/

When you launch the **mobilenet.html** file in the Chrome browser make sure to open the Developer Tools to see the output in the Console.

Include the script for TensorFlow.js, and in this case, you will also include the script for the mobilenet.

```html
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@1.0.1"> </script>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow-models/mobilenet@1.0.0"> </script>
```

In this example, I'm going to use mobilenet to classify the contents of an image and write them out to the page.

So in the body of my page, I'll need an image tag and a div to contain the output text. I've provided a few images in the download including the coffee one that's referenced here.

You can try other images to see how mobilenet classifies them for yourself if you like.

```html
<body>
    <img id="img" src="coffee.jpg"></img>
    <div id="output" style="font-family:courier;font-size:24px;height=300px"></div>
</body>
```

Next, you'll need the script that passes the image to mobilenet and gets a set of classifications back. Here's the basic version which we'll build on in a moment to make it a little more user-friendly.

Note that this code should execute after the page has loaded. So at the very least, you should have it at the bottom of the page after the closing body tag, or if you are familiar with the DOM model, you can call it when the DOM has finished loading.

For now, I'm going to keep it simple and just put it off to the closing body tag. The first thing it will do is create a variable representing the image tag on the page that we created earlier.

If this script runs before the DOM has loaded, this line will crash.

```javascript
const img = document.getElementById('img');
mobilenet.load().then(model => {
  model.classify(img).then(predictions => {
    console.log(predictions);
  });
});
```

```
const img = document.getElementById('img');
mobilenet.load().then(model => {
  model.classify(img).then(predictions => {
    console.log(predictions);
  });
});
```

Next, it will load mobilenet. Because it's stored in JSON, it's as easy as this to load the object asynchronously

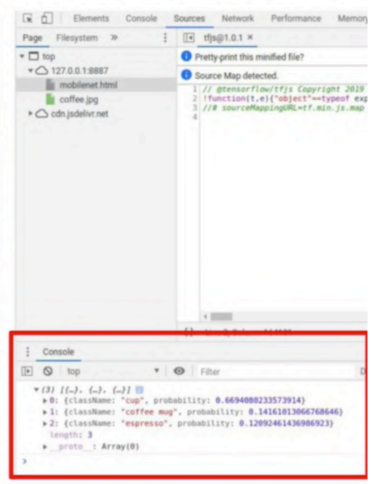
```
const img = document.getElementById('img');
mobilenet.load().then(model => {
  model.classify(img).then(predictions => {
    console.log(predictions);
  });
});
```

```
const img = document.getElementById('img');
mobilenet.load().then(model => {
  model.classify(img).then(predictions => {
    console.log(predictions);
  });
});
```

To use it, we pass the image as a parameter to the mode's classify function.

```
const img = document.getElementById('img');
mobilenet.load().then(model => {
  model.classify(img).then(predictions => {
    console.log(predictions);
  });
});
```

Then we'll get back a set of predictions which we can write out to the console. When we do this, we'll see a result like this in the browser with DevTools running




```
▼(3) [{…}, {…}, {…}]
  ▶0: {className: "cup", probability: 0.6694080233573914}
  ▶1: {className: "coffee mug", probability: 0.14161013066768646}
  ▶2: {className: "espresso", probability: 0.12092461436986923}
   length: 3
  ▶__proto__: Array(0)
```

```
const img = document.getElementById('img');
const outp = document.getElementById('output');
mobilenet.load().then(model => {
  model.classify(img).then(predictions => {
    console.log(predictions);
    for(var i = 0; i<predictions.length; i++){
      outp.innerHTML += "<br/>" + predictions[i].className
                        + " : " + predictions[i].probability;
    }
  });
});
```

Let's take a look at making this a little more user-friendly, and here's the code.

In our HTML, we had a div called output. So let's create a reference to that here and call it outp.

```
const img = document.getElementById('img');
const outp = document.getElementById('output');
mobilenet.load().then(model => {
  model.classify(img).then(predictions => {
    console.log(predictions);
    for(var i = 0; i<predictions.length; i++){
      outp.innerHTML += "<br/>" + predictions[i].className
                        + " : " + predictions[i].probability;
    }
  });
});
```

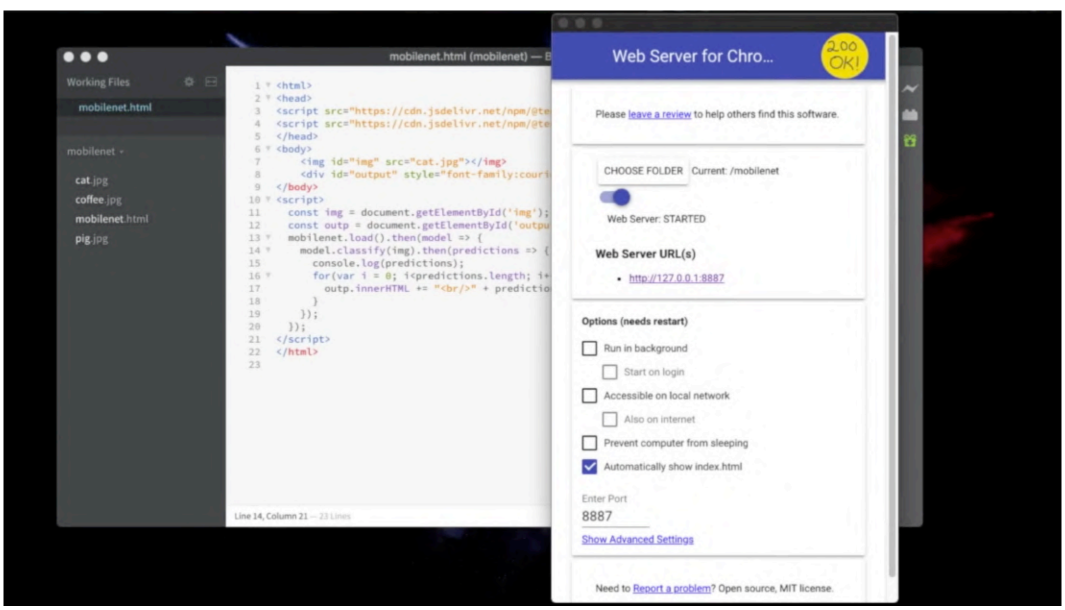
Once we get our predictions back, we can loop through them with a for loop from zero up to less than the predictions.length

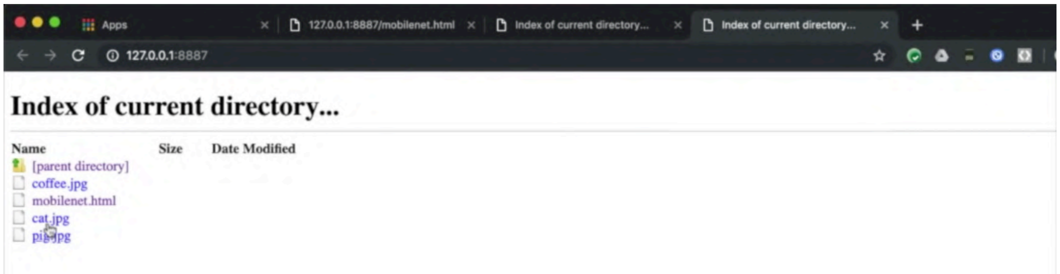
```
const img = document.getElementById('img');
const outp = document.getElementById('output');
mobilenet.load().then(model => {
  model.classify(img).then(predictions => {
    console.log(predictions);
    for(var i = 0; i<predictions.length; i++){
      outp.innerHTML += "<br/>" + predictions[i].className
                        + " : " + predictions[i].probability;
    }
  });
});
```

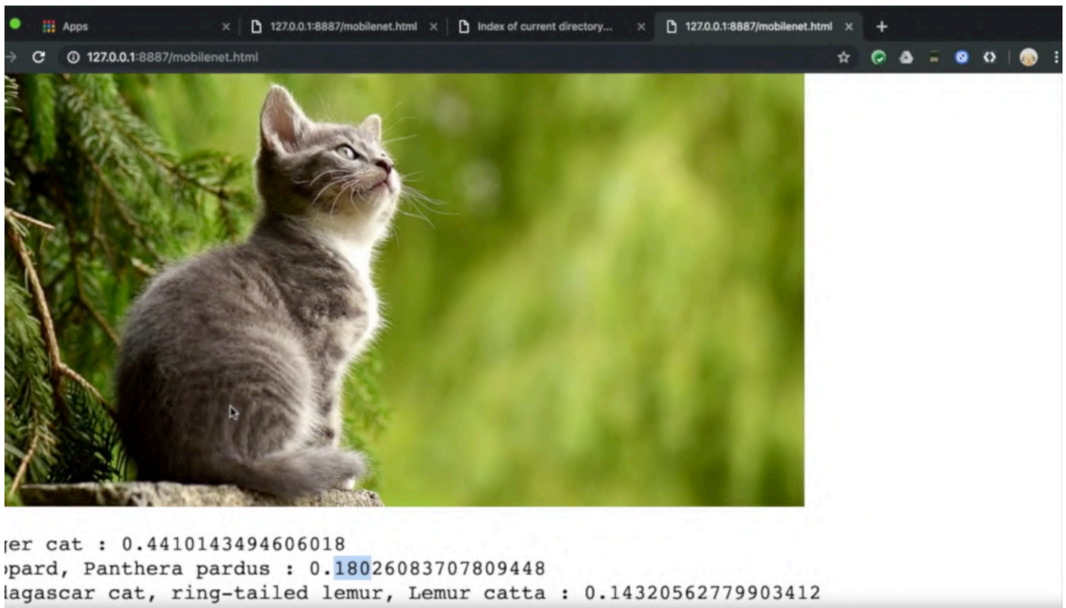
Then within the loop, we can add a break character, the prediction className, and the probability for that className to the innerHTML of the div, and when we run this, we'll see something like this.

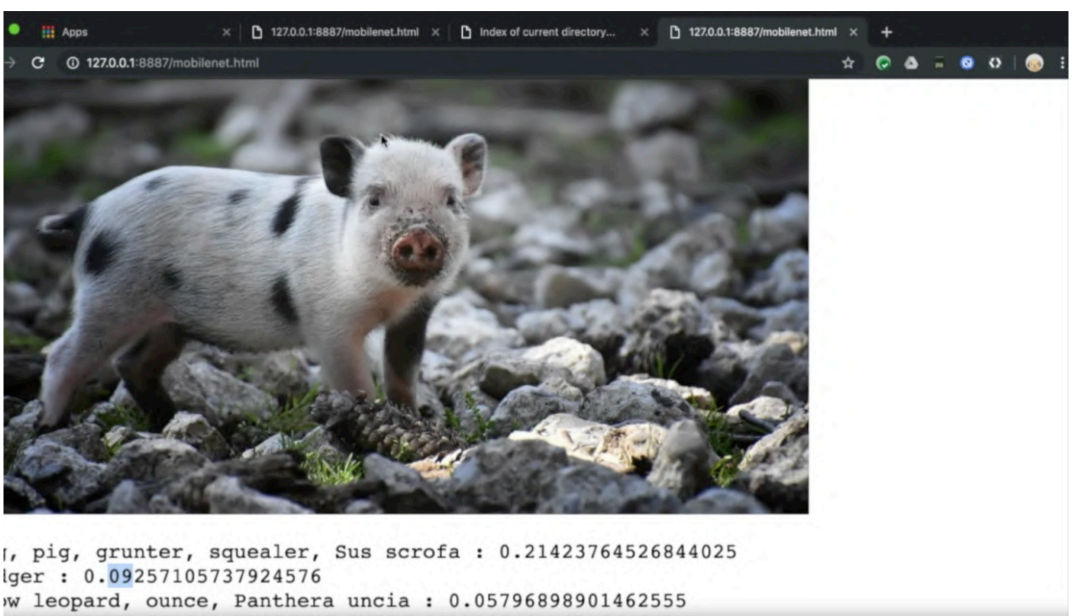

tiger cat : 0.44101303815841675
leopard, Panthera pardus : 0.18026068806648254
Madagascar cat, ring-tailed lemur, Lemur catta : 0.14320671558380127


Working Files
mobilenet.html
mobilenet
cat.jpg
coffee.jpg
mobilenet.html
pig.jpg
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"> </script>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow-models/mobilenet@1.0.0"> </script>
</head>
<body>
<img id="img" src="cat.jpg"></img>
<div id="output" style="font-family:courier;font-size:24px;height=300px"></div>
</body>
<script>
const img = document.getElementById('img');
const outp = document.getElementById('output');
mobilenet.load().then(model => {
model.classify(img).then(predictions => {
console.log(predictions);
for(var i = 0; i<predictions.length; i++){
outp.innerHTML += "
" + predictions[i].className + " : " + predictions[i].probability;
}
});
});
</script>
</html>


mobilenet.html (mobilenet)
Web Server for Chro...
200 OK!
Please leave a review to help others find this software.
CHOOSE FOLDER
Current: /mobilenet
Web Server: STARTED
Web Server URL(s)
http://127.0.0.1:8887
Options (needs restart)
Run in background
Start on login
Accessible on local network
Also on internet
Prevent computer from sleeping
Automatically show index.html
Enter Port
8887
Show Advanced Settings
Need to Report a problem? Open source, MIT license.
Line 14, Column 21 — 23 Lines


Apps
127.0.0.1:8887/mobilenet.html
Index of current directory...
127.0.0.1:8887
Index of current directory...
Name
Size
Date Modified
[parent directory]
coffee.jpg
mobilenet.html
cat.jpg


Apps
127.0.0.1:8887/mobilenet.html
Index of current directory...
127.0.0.1:8887/mobilenet.html
ger cat : 0.4410143494606018
opard, Panthera pardus : 0.18026083707809448
lagascar cat, ring-tailed lemur, Lemur catta : 0.14320562779903412


Apps
127.0.0.1:8887/mobilenet.html
Index of current directory...
127.0.0.1:8887/mobilenet.html
, pig, grunter, squealer, Sus scrofa : 0.21423764526844025
lger : 0.09257105737924576
w leopard, ounce, Panthera uncia : 0.05796898901462555

Linear Model

In the next example, we will train a linear model in Python and then convert it into JSON format using the TensorFlow.js converter.

Open the **Linear-to-JavaScript.ipynb** Jupyter notebook found in the following folder in the GitHub repository:

[dlaicourse/TensorFlow Deployment/Course 1 - TensorFlow-JS/Week 3/Exercise/]()

To run this notebook you will need have installed Jupyter with Python 3, TensorFlow 2.0, Tensorflow.js, and NumPy.

After you run the Jupyter Notebook, you will end up with a single JSON file named **model.json** and a **.bin** file named **group1-shard1of1.bin** (you can also find these files in the above folder of the GitHub repository).

After you have the **model.json** and the **group1-shard1of1.bin** files, you can launch the **linear.html** file in the Chrome browser. Don't forget to open the Developer Tools to see the output in the Console.

```
!pip install tensorflowjs
```

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
print(tf.__version__)
model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')
xs = np.array([-1.0,  0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)
model.fit(xs, ys, epochs=500)
```

```
print(model.predict([10.0]))
```

Saved Model

We'll start by generating a directory to save the file in, and we do that using a timestamp. So we'll import time, get the current time stamp, and save the model and the path /tmp/saved_models/ followed by the timestamp.

```
import time
saved_model_path = "/tmp/saved_models/{}".format(int(time.time()))

# For TensorFlow 2.0 use this:
# tf.keras.experimental.export_saved_model(model, saved_model_path)

# For TensorFlow 1.x use this:
tf.contrib.saved_model.save_keras_model(model, saved_model_path)
```

```
import time
saved_model_path = "/tmp/saved_models/{}".format(int(time.time()))

# For TensorFlow 2.0 use this:
# tf.keras.experimental.export_saved_model(model, saved_model_path)

# For TensorFlow 1.x use this:
tf.contrib.saved_model.save_keras_model(model, saved_model_path)
```

```python
import time
saved_model_path = "/tmp/saved_models/{}".format(int(time.time()))

# For TensorFlow 2.0 use this:
# tf.keras.experimental.export_saved_model(model, saved_model_path)

# For TensorFlow 1.x use this:
tf.contrib.saved_model.save_keras_model(model, saved_model_path)
```

```
INFO:tensorflow:SavedModel written to /tmp/saved_models/1554528640/1554528642/saved_model.pb
b'/tmp/saved_models/1554528640/1554528642'
```

Here's the command convert a saved model that was previously saved into the tensorflow.js formats and it's called model.json.

The input format parameter takes a number of different values. But to use a saved model you use this one; keras saved model. model.

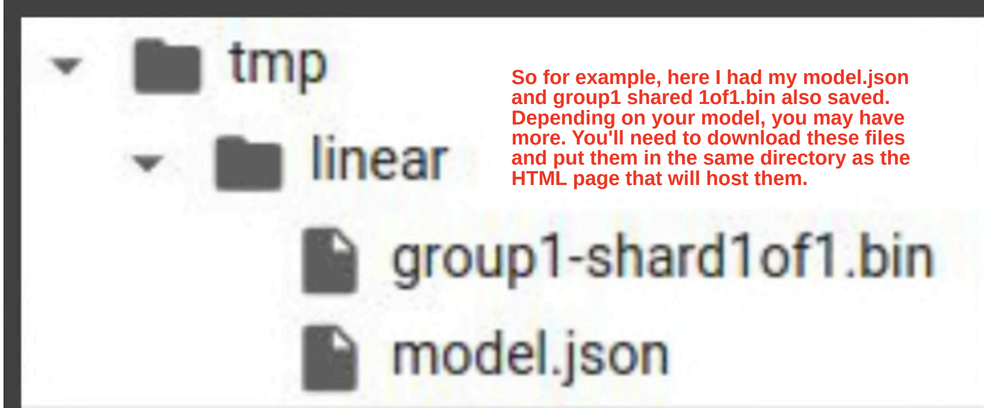
```
!tensorflowjs_converter \
    --input_format=keras_saved_model \
    /tmp/saved_models/1554528640/1554528642 \
    /tmp/linear
```

Next, you'll specify the directory containing the saved model, and this is the timestamp-based directory that you found a few moments ago.

```
!tensorflowjs_converter \
    --input_format=keras_saved_model \
    /tmp/saved_models/1554528640/1554528642 \
    /tmp/linear
```

Finally, is the output directory where you want the JSON to be saved. You'll need to keep a close eye on this directory as more than just the JSON may be written there, you'll need all of the files.

```
!tensorflowjs_converter \
    --input_format=keras_saved_model \
    /tmp/saved_models/1554528640/1554528642 \
    /tmp/linear
```



So for example, here I had my model.json and group1 shared 1of1.bin also saved. Depending on your model, you may have more. You'll need to download these files and put them in the same directory as the HTML page that will host them.

```
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
<script>
    async function run(){
        const MODEL_URL = 'http://127.0.0.1:8887/model.json';
        const model = await tf.loadLayersModel(MODEL_URL);
        console.log(model.summary());
        const input = tf.tensor2d([10.0], [1, 1]);
        const result = model.predict(input);
        alert(result);
    }
    run();
</script>
<body>
</body>
</html>
```

Let's look at an HTML page with the model hosted in it.

First of all is the URL of the model. It has to be loaded over HTTP. So in this case, while it's in the same directory as the HTML, I still use the URL path to it. Be sure to get this part right.

```
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
<script>
    async function run(){
        const MODEL_URL = 'http://127.0.0.1:8887/model.json';
        const model = await tf.loadLayersModel(MODEL_URL);
        console.log(model.summary());
        const input = tf.tensor2d([10.0], [1, 1]);
        const result = model.predict(input);
        alert(result);
    }
    run();
</script>
<body>
</body>
</html>
```

To get the JSON and turn it into a model I can use, I'll call await tf.loadlayersModel passing it that URL. Once this completes, I'll have a trained model available to me.

```
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
<script>
    async function run(){
        const MODEL_URL = 'http://127.0.0.1:8887/model.json';
        const model = await tf.loadLayersModel(MODEL_URL);
        console.log(model.summary());
        const input = tf.tensor2d([10.0], [1, 1]);
        const result = model.predict(input);
        alert(result);
    }
    run();
</script>
<body>
</body>
</html>
```

```
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
<script>
    async function run(){
        const MODEL_URL = 'http://127.0.0.1:8887/model.json';
        const model = await tf.loadLayersModel(MODEL_URL);
        console.log(model.summary());
        const input = tf.tensor2d([10.0], [1, 1]);
        const result = model.predict(input);
        alert(result);
    }
    run();
</script>
<body>
</body>
</html>
```

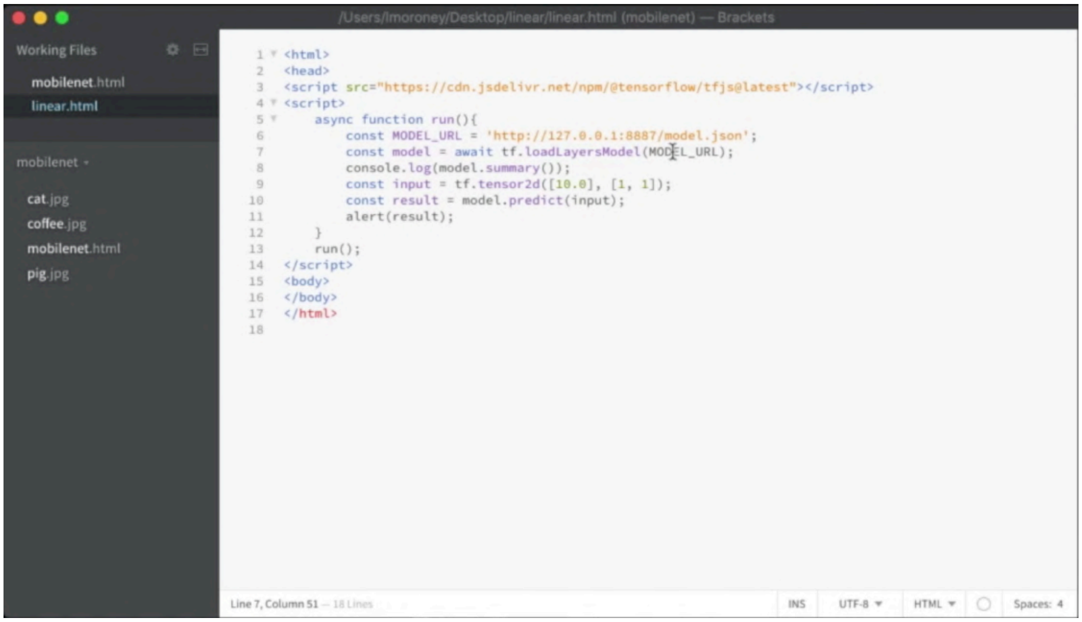
I'll create my input tensor like this. I want to predict the value for 10. So to do that, I have a two-dimensional tensor with the first dimension being the value to classify and the second being the dimension of that value, in this case one by one.

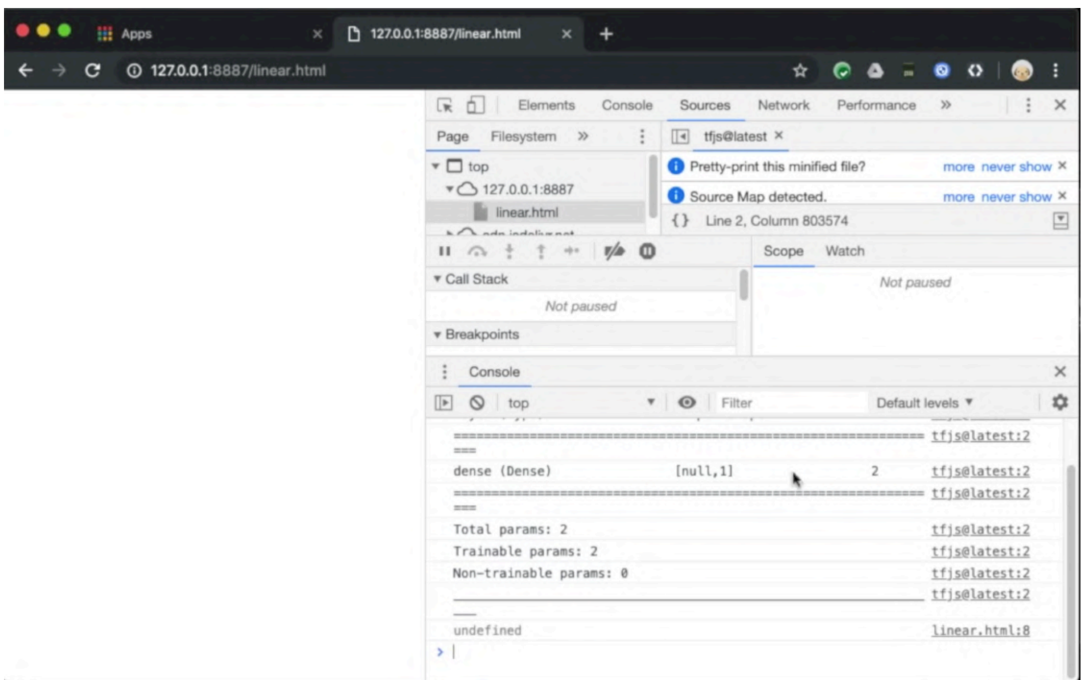
```
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
<script>
    async function run(){
        const MODEL_URL = 'http://127.0.0.1:8887/model.json';
        const model = await tf.loadLayersModel(MODEL_URL);
        console.log(model.summary());
        const input = tf.tensor2d([10.0], [1, 1]);
        const result = model.predict(input);
        alert(result);
    }
    run();
</script>
<body>
</body>
</html>
```

We then get the results by calling model.predict and passing it the inputs.

```
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
<script>
    async function run(){
        const MODEL_URL = 'http://127.0.0.1:8887/model.json';
        const model = await tf.loadLayersModel(MODEL_URL);
        console.log(model.summary());
        const input = tf.tensor2d([10.0], [1, 1]);
        const result = model.predict(input);
        alert(result);
    }
    run();
</script>
<body>
</body>
</html>
```

Then we can alert the result.


/Users/lmoroney/Desktop/linear/linear.html (mobilenet) — Brackets
Working Files
mobilenet.html
linear.html
mobilenet
cat.jpg
coffee.jpg
mobilenet.html
pig.jpg
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
<script>
async function run(){
const MODEL_URL = 'http://127.0.0.1:8887/model.json';
const model = await tf.loadLayersModel(MODEL_URL);
console.log(model.summary());
const input = tf.tensor2d([10.0], [1, 1]);
const result = model.predict(input);
alert(result);
}
run();
</script>
<body>
</body>
</html>
Line 7, Column 51 — 18 Lines
INS
UTF-8
HTML
Spaces: 4


Apps
127.0.0.1:8887/linear.html
127.0.0.1:8887/linear.html
Elements
Console
Sources
Network
Performance
Page
Filesystem
tfjs@latest
top
127.0.0.1:8887
linear.html
Pretty-print this minified file?
more never show
Source Map detected.
more never show
Line 2, Column 803574
Scope
Watch
Call Stack
Not paused
Not paused
Breakpoints
Console
top
Filter
Default levels
dense (Dense) [null,1] 2
Total params: 2
Trainable params: 2
Non-trainable params: 0
undefined
tfjs@latest:2
linear.html:8