Browse > Computer Science > Software Development

This course is part of the **TensorFlow: Data and Deployment Specialization**

Browser-based Models with TensorFlow.js

4.7 887 ratings | 95%

Laurence Moroney

Go To Course

Already enrolled

34,000 already enrolled

About this Course

235,102 recent views

Bringing a machine learning model into the real world involves a lot more than just modeling. This Specialization will teach you how to navigate various deployment scenarios and use data more effectively to train your model.

In this first course, you'll train and run machine learning models in any browser using TensorFlow.js. You'll learn techniques for handling data in the browser, and at the end you'll build a computer vision project that recognizes and classifies objects from a webcam.

This Specialization builds upon our TensorFlow in Practice Specialization. If you are new to TensorFlow, we recommend that you take the TensorFlow in Practice Specialization first. To develop a deeper, foundational understanding of how neural networks work, we recommend that you take the Deep Learning Specialization.

WHAT YOU WILL LEARN

- ✓ Train and run inference in a browser
- ✓ Handle data in a browser
- ✓ Build an object classification and recognition model using a webcam

Syllabus - What you will learn from this course

Content Rating 95% (2,638 ratings)

WEEK 1

7 hours to complete

Introduction to TensorFlow.js

Welcome to Browser-based Models with TensorFlow.js, the first course of the TensorFlow for Data and Deployment Specialization. In this first course, we're going to look at how to train machine learning models in the browser and how to use them to perform inference using JavaScript. This will allow you to use machine learning directly in the browser as well as on backend servers like Node.js. In the first week of the course, we are going to build some basic models using JavaScript and we'll execute them in simple web pages.

11 videos (Total 30 min), 7 readings, 5 quizzes **See All**

WEEK 2

5 hours to complete

Image Classification In the Browser

This week we'll look at Computer Vision problems, including some of the unique considerations when using JavaScript, such as handling thousands of images for training. By the end of this module you will know how to build a site that lets you draw in the browser and recognizes your handwritten digits!

8 videos (Total 27 min), 5 readings, 3 quizzes **See All**

WEEK 3

5 hours to complete

Converting Models to JSON Format

This week we'll see how to take models that have been created with TensorFlow in Python and convert them to JSON format so that they can run in the browser using Javascript. We will start by looking at two models that have already been pre-converted. One of them is going to be a toxicity classifier, which uses NLP to determine if a phrase is toxic in a number of categories; the other one is Mobilenet which can be used to detect content in images. By the end of this module, you will train a model in Python yourself and convert it to JSON format using the tensorflow.js converter.

12 videos (Total 28 min), 6 readings, 3 quizzes **See All**

WEEK 4

5 hours to complete

Transfer Learning with Pre-Trained Models

One final work type that you'll need when creating Machine Learned applications in the browser is to understand how transfer learning works. This week you'll build a complete web site that uses TensorFlow.js, capturing data from the web cam, and re-training mobilenet to recognize Rock, Paper and Scissors gestures.

11 videos (Total 26 min), 3 readings, 3 quizzes **See All**

Table of Content

Week 1: Introduction to TensorFlow.js

Welcome to Browser-based Models with TensorFlow.js, the first course of the TensorFlow for Data and Deployment Specialization. In this first course, we're going to look at how to train machine learning models in the browser and how to use them to perform inference using JavaScript. This will allow you to use machine learning directly in the browser as well as on backend servers like Node.js. In the first week of the course, we are going to build some basic models using JavaScript and we'll execute them in simple web pages.

Learning Objectives

- Use TensorFlow.js to build and train simple machine learning models in JavaScript
- Describe the key characteristics of one-hot encoding
- Use TensorFlow.js to load data from a CSV file
- Use Web Server for Chrome to serve web pages from a local folder over the network using HTTP.

1.1 Specialization Introduction

1.1.1 Specialization Introduction, A conversation with Andrew Ng

This Specialization will teach you how to deploy your trained model in TensorFlow. We will learn to run models in the browsers with JavaScript and on the phone.

1.2 Training and Inference using TensorFlow.js in JavaScript

1.2.1 Course Introduction, A conversation with Andrew Ng

TensorFlow JS allows you to train neural network and inference in web browser. Users can upload a picture to a web browser or grab an image from a webcam, and then train or inference a neural network in the web browser without sending that image up to the cloud to be processed by a server. To save time and process user privacy, it can also be deployed on mobile without connection.

1.2.2 Reading: Getting Your System Ready

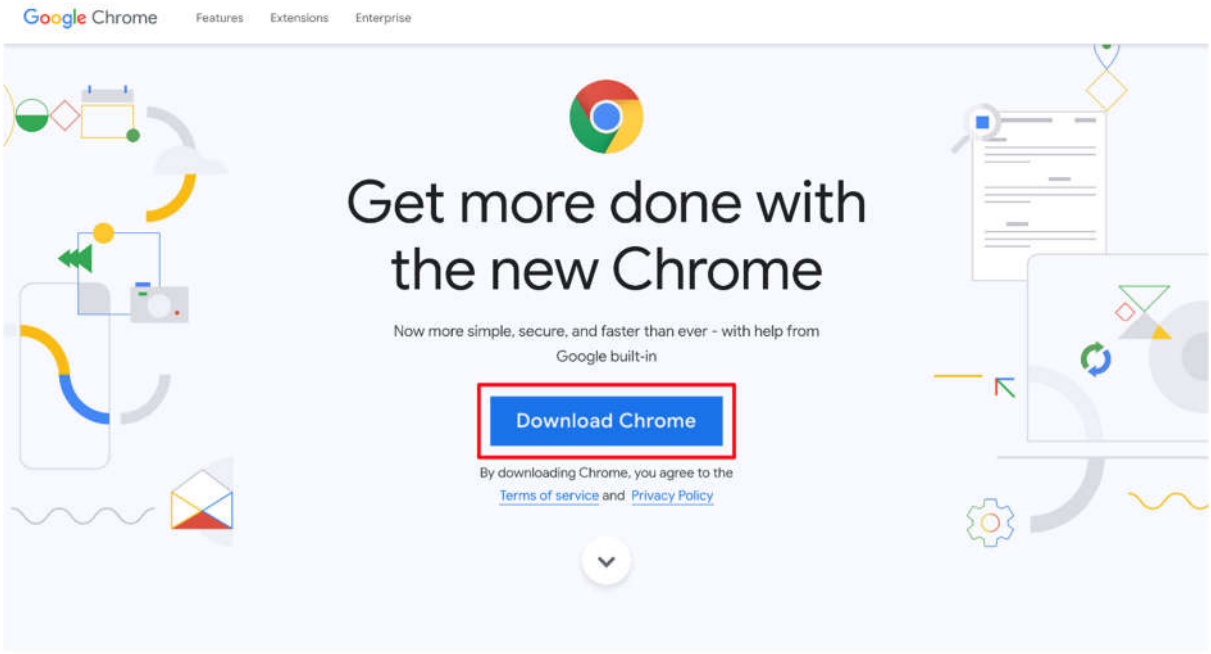
In this course you will be running all the examples and exercises locally on your machine. Consequently, you need an HTML editor, and a web server installed on your machine. Throughout this course we use

- Chrome: internet browser;
- Brackets: HTML editor;
- Web Server for Chrome App: web server.

All of these are free and are available for various platforms including Windows, Mac OS, and Linux. Below, we will go over the installation process of each of these.

Chrome

To install the Chrome browser, simply visit the Chrome home page.



Click the blue "Download Chrome" button to download the installation file to your machine.

Install Chrome on Windows

1. If prompted, click **Run** or **Save**.
2. If you chose **Save**, double-click the downloaded file to start installing.
3. Start Chrome.

Install Chrome on Mac

1. Open the file called **googlechrome.dmg**.
2. In the window that opens, find Chrome .
3. Drag Chrome to the Applications folder.
4. Open Chrome.
5. Open Finder.
6. In the sidebar, to the right of Google Chrome, click Eject.

Install Chrome on Linux

Use the same software that installs programs on your computer to install Chrome. You'll be asked to enter the administrator account password.

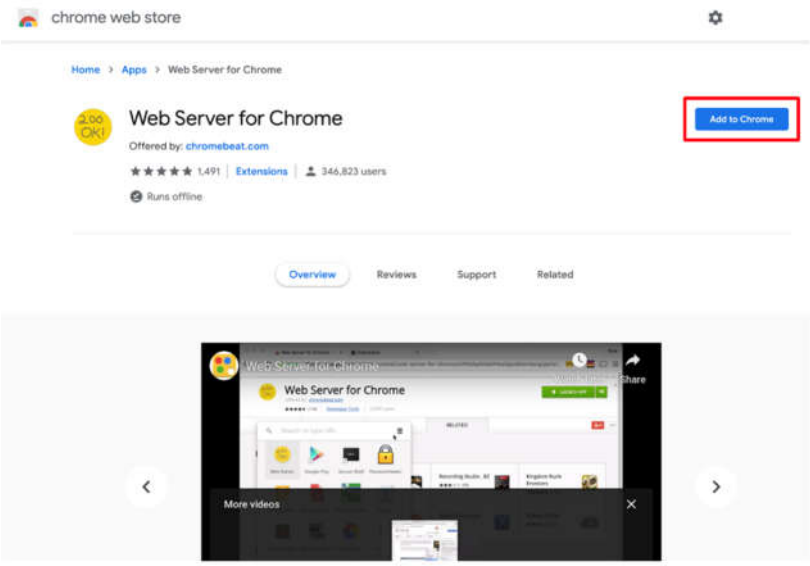
1. To open the package, click **OK**.
2. Click **Install Package**.

Google Chrome will be added to your software manager so it stays up-to-date.

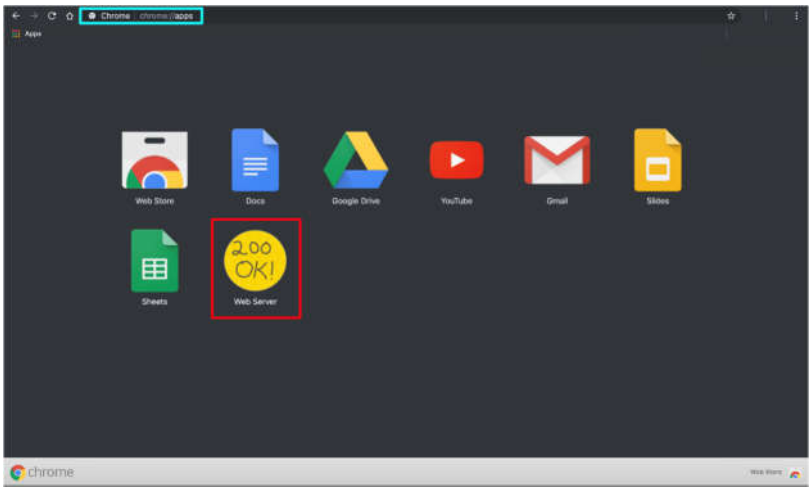
NOTE: It is very important that you use the latest version of Chrome for the exercises to run. Make sure you have the latest version of Chrome installed. The current latest version is: 78.0.3904.108.

Web Server for Chrome

To install the Web Server for Chrome App go to the Web Server for Chrome App page in the chrome web store.

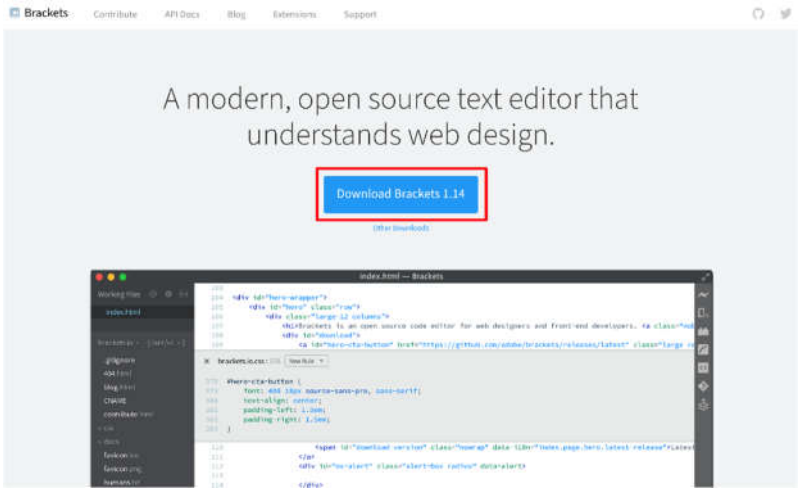


Click on the blue "Add to Chrome" button on the upper right-hand corner. The app will automatically be added to your Chrome Browser. You can find it under the Chrome Apps. Type **chrome://apps/** into Google Chrome to pull up your Chrome Apps.



Brackets

To install the Brackets, simply visit the [Brackets home page](#).



Click on the blue "Download Brackets" button to download the installation file to your machine.

Note: On September 1, 2021, Adobe will end support for Brackets so you might not be able to find the downloadable binaries (Download Brackets button) directly on their website. You can still find them in their [Github repo releases section](#).

For Windows, v1.14.2 download link can be found [here](#).
For Mac, v1.14.2 download link can be found [here](#).

Install Brackets on Windows

1. If prompted, click **Run** or **Save**.
2. If you chose **Save**, double-click the downloaded file to start installing.
3. Start Brackets.
4. When you open Brackets for the first time it will open a Windows Security Alter window. Click on "Allow Access" to run Brackets.

Install Brackets on Mac

1. Open the file called **Brackets.Release.1.14.0.dmg**. Note: The numbers at the end of the filename will change depending on the version that you are installing.
2. Open the downloaded file.
3. Drag Brackets to the Applications folder.

4. Open Brackets.
5. Open Finder.
6. In the sidebar, to the right of Brackets, click Eject.

Install Brackets on Linux

Use the same software that installs programs on your computer to install Brackets. You'll be asked to enter the administrator account password.

1. To open the package, click **OK**.
2. Click **Install Package**.

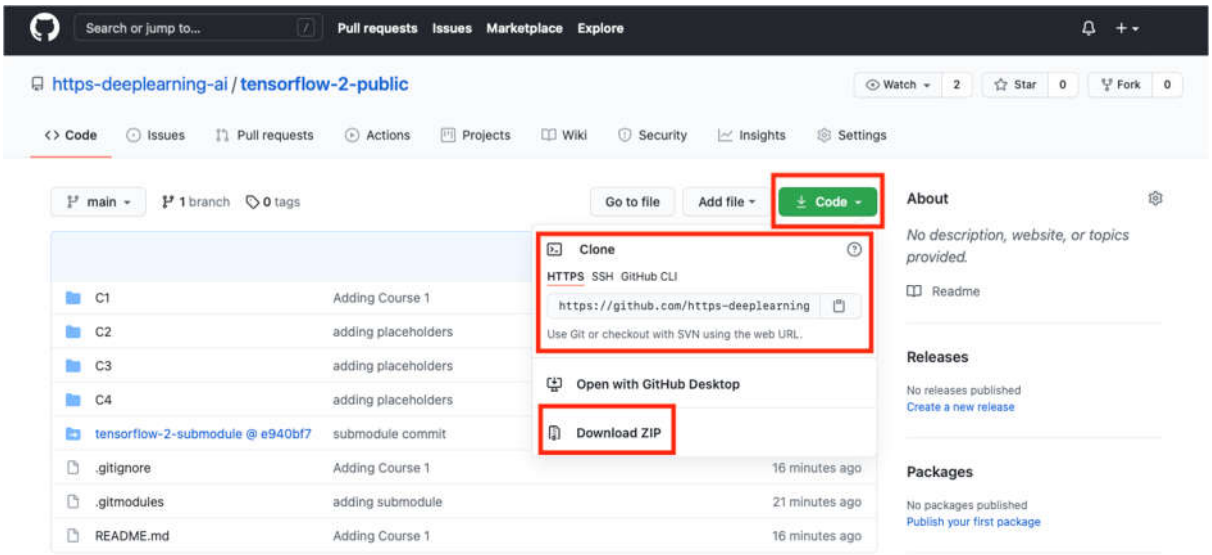
If you run into trouble installing Brackets in your Linux system, make sure to check out the Brackets Linux Guide.

That's it! You should now be all setup to follow along.

1.2.3 **Reading**: Downloading the Ungraded Labs and Programming Assignments

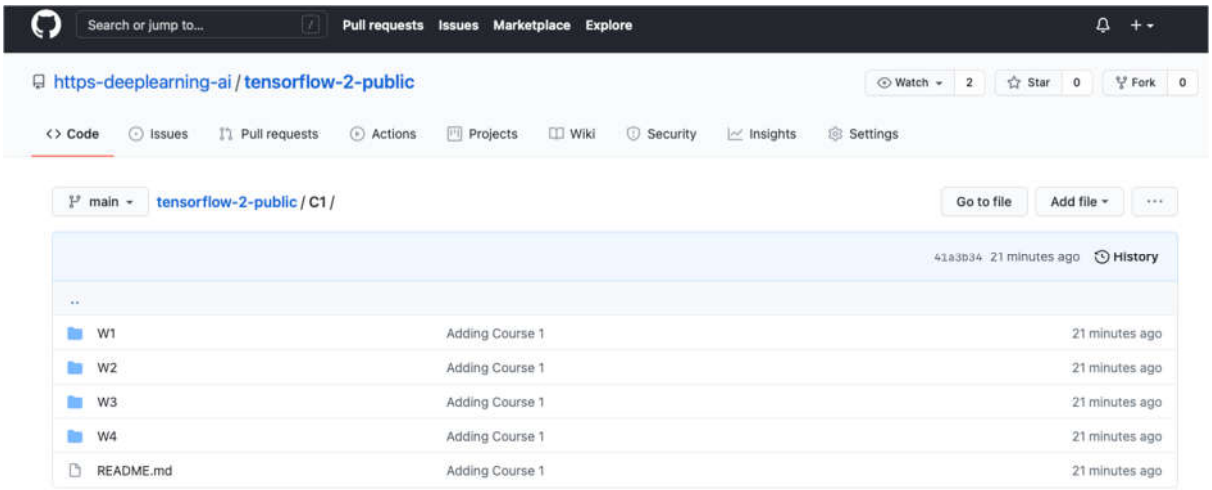
Like we mentioned earlier, in this course you will be running all the coding examples and exercises locally on your machine. We have created this GitHub Repository where you can find all the ungraded labs and programming assignments not only for this course but for the entire TensorFlow for Data and Deployment Specialization.

You can download all the ungraded labs and programming assignments to your computer by cloning or downloading the GitHub Repository.



You can find the corresponding ungraded labs and programming assingments for this course in the following folder in the GitHub repository:

tensorflow-2-public/C1_Browser-based-TF-JS/



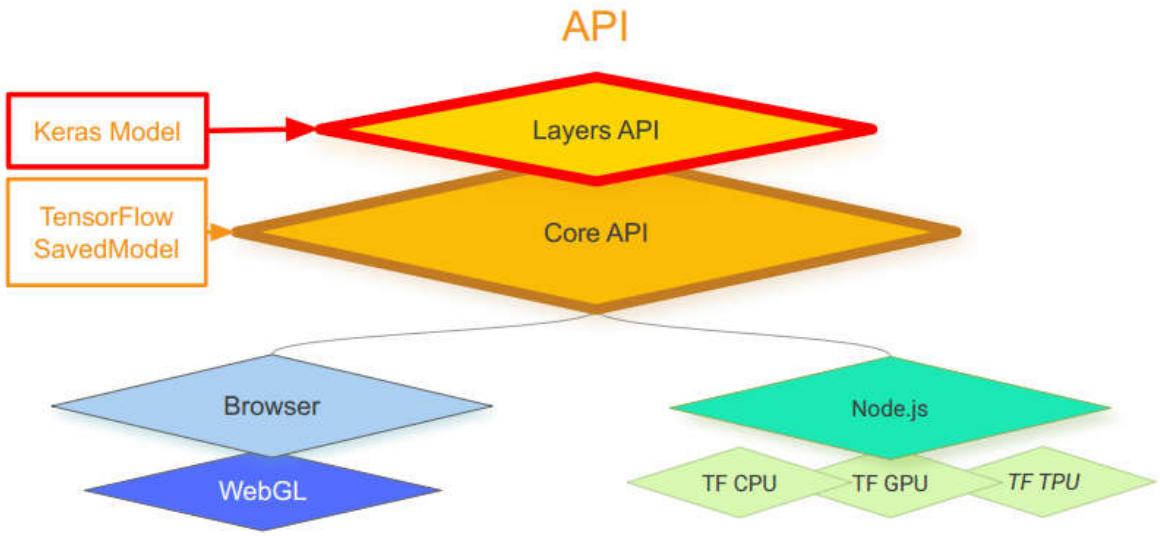
Each folder contains the corresponding ungraded labs and programming assignments for each week of this course.

NOTE: The code in the repository is updated occasionally. Therefore, the code in the repository may vary slightly from the one shown in the videos.

1.2.4 A few Words from Laurence

This week you will build some basic models using JavaScript and execute them in a web page. Here is the architecture of TensorFlow.js.

- **Layers API**: as a high-level API, allows you to program CNN and DNN using JavaScript. Its syntax is slightly different with Python.
- **Core APIs**: a low-level APIs, which can work with a TensorFlow saved model formats. Especially TensorFlow 2.0, which can be used across the Python APIs, the JavaScript ones, and even TensorFlow Lite for mobile and embedded devices.
- **Browser and WebGL**: interact with the Core API for accelerated training and inference.
- **Node.js:** allow you to build server-side or terminal applications using the available CPUs, GPUs, and TPUs.



1.2.5 Building the Model

To build a TensorFlow.js, you first need to build a web page. Here is a simple web page.

```
<html>
<head></head>
<body>
    <h1>First HTML Page</h1>
</body>
</html>
```

Then load the TensorFlow.js file by adding a script tag below the ***head***.

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

To build a model that infers the relationship between two numbers (y=2x-1). Make sure this is above the ***body*** tag in the HTML page.

- ***tf.sequential***: defines the model to be a sequential.
- ***model.add***: add one dense layer with one unit to the sequence.
- ***model.compile.***: compile the model with a loss function (*MeanSquaredError*) and optimizer (*sgd*).
- ***model.summary***: outputs the summary of the model.

```
<script lang="js">
    const model = tf.sequential();
    model.add(tf.layers.dense({units: 1, inputShape: [1]}));
    model.compile({loss:'meanSquaredError',
                   optimizer:'sgd'});
    model.summary();
</script>
```

Here are the console outputs. It is a simple neural network with 2 parameters (a weight and a bias).

```
_________________________________________________________________
Layer (type)                 Output shape              Param #
=================================================================
dense_Dense1 (Dense)         [null,1]                  2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

The data format for training the neural network is slightly different with Python. In python, you define data as a NumPy array. JavaScript doesn't have NumPy array, it uses *tensor2d* (a two-dimensional array or two one-dimensional arrays).

Here, the training values are in one array, and the shape of training values in second array。

```
const xs = tf.tensor2d([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], [6, 1]);
const ys = tf.tensor2d([-3.0, -1.0, 2.0, 3.0, 5.0, 7.0], [6, 1]);
```

1.2.6 Training the Model

Training is an asynchronous function, because it will take an indeterminate time to compete, and we don't want to lock the browser while this is going on.

- ***doTraining:*** is an asynchronous function. It trains and passes the model just created.
- ***model.predict:*** predict model output. The tensor will be fed into the model and the model will produce an output and be displayed in the alert box.

```
doTraining(model).then(() => {
    alert(model.predict(tf.tensor2d([10], [1,1])));
});
```

Here is the complete asynchronous function for training the model. This code should go at the top of script block. Using *model.fit* function to train the model and pass the xs and ys parameters from it. In javaScript, you need to *await* the result while training the model. The rest of the parameters are a JSON list. It is enclosed in braces with each list item denoted by a name, followed by a colon, followed by a value.

- ***epochs:*** defines the number times of training the entire data.
- ***callbacks***: as a list, it lists the onEpochEnd function.
- ***onEpochEnd***: gets the epoch and logs as parameters and prints out the epoch and loss for that epoch.

```
async function doTraining(model){
    const history =
          await model.fit(xs, ys,
                { epochs: 500,
                  callbacks:{
                      onEpochEnd: async(epoch, logs) =>{
                          console.log("Epoch:" + epoch + " Loss:" + logs.loss);
                      }
                  }
                });
}
```

1.2.7 Intake Survey

We would love to learn more about your experience with DeepLearning.AI and with AI in general. Please consider completing the short 3-question survey so we can better serve you!

This course uses a third-party tool, Intake Survey, to enhance your learning experience. The tool will reference basic information like your name, email, and Coursera ID.

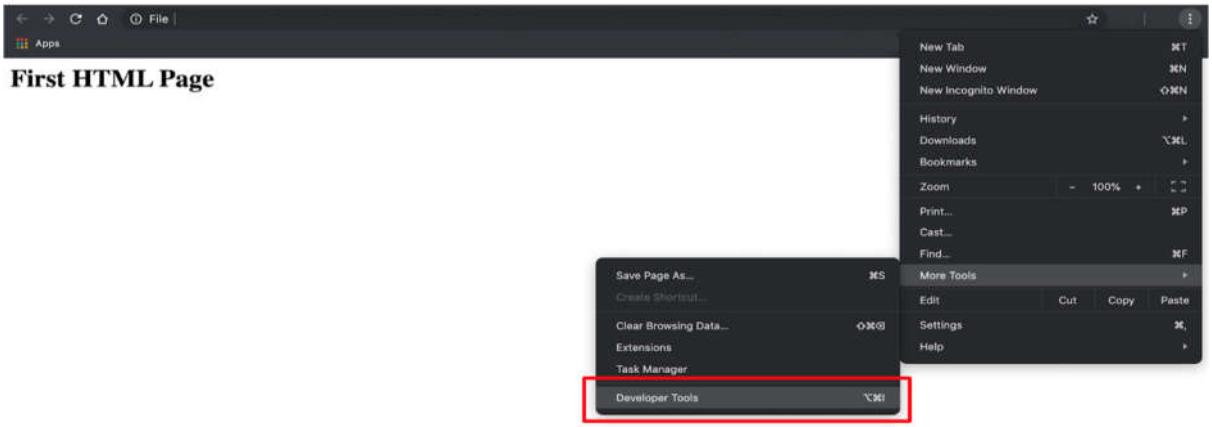
1.2.8 Ungraded External Tool: Join us on Discourse!

1.2.9 Reading: Your First Model

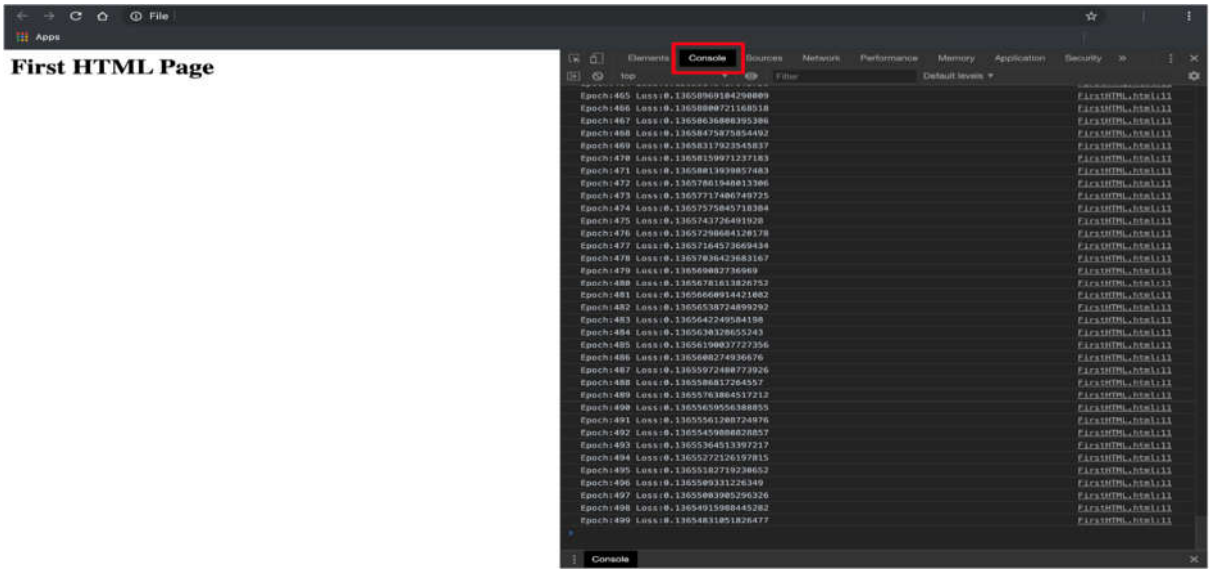
We start at a simple example. You can use Brackets to open the **C1_W1_Lab_1_FirstHTML.html** and find the **C1_W1_Lab_1_FirstHTML.html** file in the following folder in the GitHub repository:

tensorflow-2-public/C1_Browser-based-TF-JS/W1/ungraded_labs/

When you launch the **C1_W1_Lab_1_FirstHTML.html** file in the Chrome browser make sure to open the Developer Tools to see the training progress.



After opening the Developer Tools you should see the training progress in the Console: (ctrl+shift+j)



1.2.10 First Example in Code

```
<html>
<head></head>
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
    <script lang="js">
        async function doTraining(model){
            const history =
                await model.fit(xs, ys,
                    { epochs: 500,
                      callbacks:{
                          onEpochEnd: async(epoch, logs) =>{
                              console.log("Epoch:"
                                          + epoch
                                          + " Loss:"
                                          + logs.loss);

                          }
                      }
                    });
        }
        const model = tf.sequential();
        model.add(tf.layers.dense({units: 1, inputShape: [1]}));
        model.compile({loss:'meanSquaredError',
                       optimizer:'sgd'});
        model.summary();
        const xs = tf.tensor2d([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], [6, 1]);
        const ys = tf.tensor2d([-3.0, -1.0, 2.0, 3.0, 5.0, 7.0], [6, 1]);
        doTraining(model).then(() => {
            alert(model.predict(tf.tensor2d([10], [1,1])));
        });
    </script>
<body>
    <h1>First HTML Page</h1>
</body>
</html>
```

1.2.11 Quiz 1

Question 1

What is the name of the API at the heart of TensorFlow.js which allows things like layers to be used?

- TFJS API
- Core TF API
- JS API
- Core API

Question 2

How does TensorFlow.js use GPU acceleration in the browser?

- It works natively through GPU libraries in TensorFlow
- You access GPU through WebGL in the browser
- It doesn't
- You must install GPU runtimes for each browser, and explicitly use them

Question 3

How can you use a TPU with TensorFlow.js?

- You have to serve your JS from a GCP instance
- You can use Node.js on GCP and access TPU instances
- Only using Colab
- You can't

Which of the following lines of code will correctly add a single dense layer containing a single neuron that takes a numeric input to a model using JavaScript?

- `model.add(tf.layers.dense({units: 1, inputShape: [1]}));`
- `model.add(tf.layers({units: 1, inputShape: [1,1]}))`
- `model.add(tf.layers.dense({units= 1, inputShape: [1]}));`
- `model.add(tf.layers.dense({units: 1, inputShape:= [1]}));`

Question 5

When creating data to input to a model using Python you could use a numpy array. How would you do it in JavaScript?

- Use a tensor2d containing the data
- Use a tensor2d containing the data and the shape of the data
- Use a numpyjs array
- Use a tensor2d contining a numpyjs array

Question 6

If I train a model to detect a linear relationship (i.e. Y=2X-1), what line of code would output a prediction from that model for Y where X=10?

- `alert(model.predict(tf.tensor2d([10])));`
- `alert(model.predict(10));`
- `alert(model.predict(tf.tensor2d([10], [1,1])));`

```
alert(model.predict([10], [1,1]));
```

Question 7

When training a model, if I want to log training status at the end of an epoch, what is the name of the callback event you want to capture?

onEpochEnd

EpochEnd

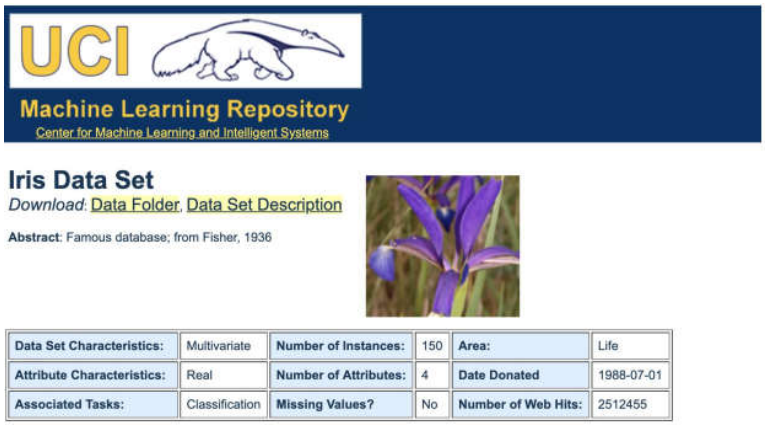
EpochEnded

OnEpochEnded

1.3 Training Models with CSV Files

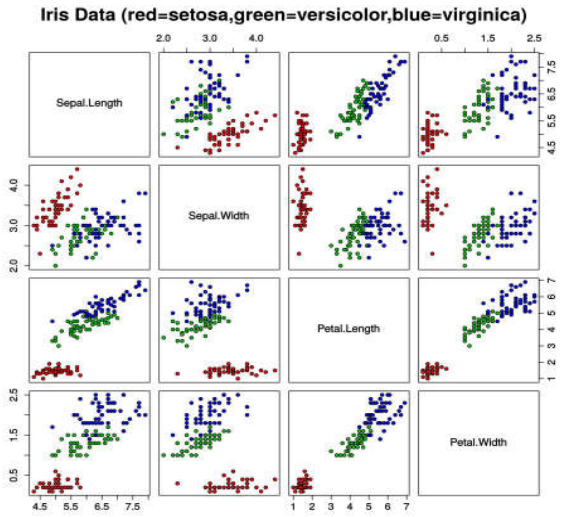
1.3.1 The Iris Dataset

Tensorflow.js provides some tools to read data from a CSV file and use it to train a multi-class classifier. We use Iris dataset as example.



https://archive.ics.uci.edu/ml/datasets/iris

This dataset includes 150 samples taken from 3 types of iris flower with 50 from each type. This plot by Nicoguaro shows the potential to build a classifier.



By Nicoguaro - Own work, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=46257808

1.3.2 **Reading**: Iris Dataset Documentation

To know more about the Iris dataset please visit the [Iris dataset home page](#).

1.3.3 Reading the Data

In this SCV file, the first line contains the column name definition. The first four columns are features and the last column *species* is the label. Note, they are all in text, and code will remix this into number to map the species to a neuron in the output layer.

```
sepal_length,sepal_width,petal_length,petal_width,species
5.1,3.5,1.4,0.2,setosa
4.9,3,1.4,0.2,setosa
4.7,3.2,1.3,0.2,setosa
4.6,3.1,1.5,0.2,setosa
5,3.6,1.4,0.2,setosa
5.4,3.9,1.7,0.4,setosa
4.6,3.4,1.4,0.3,setosa
5,3.4,1.5,0.2,setosa
```

We'll start by putting an asynchronous function into a JavaScript block, which can await some values for example when we're training.

```
async function run() {
}
```

First, the CSV is at a URL. It is loading from the same directory as the web page hosting and using the HTTP stack to get the file, not loading from the file system directly. We will use the *brackets IDE* with a built-in web server.

- ***tf.data.csv:*** manages CSV file by calling it and passing the URL.
- ***isLabel***: identifies the label from the column name.

```
const csvUrl = 'iris.csv';
const trainingData = tf.data.csv(csvUrl, {
    columnConfigs: {
        species: {
            isLabel: true
        }
    }
});
```

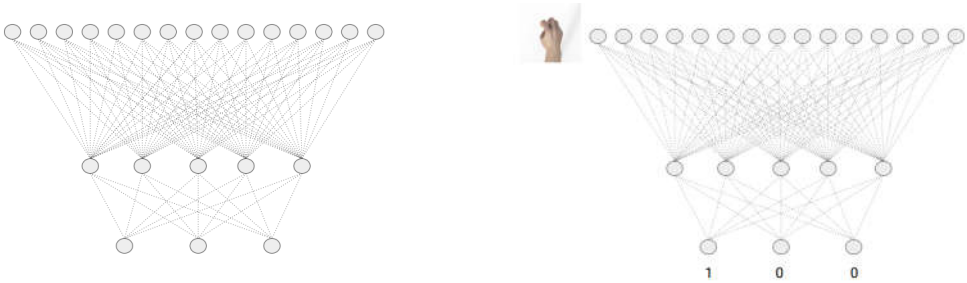
For training, we convert the dictionaries (return from *tf.data.csv*) into array by calling the map method. We covert the label into one hot encoded array in the set of *ys* in the *labels.*

- ***xs:Object.values(xs):*** return the features in array.
- ***ys:Object.values(labels):***: returns on-hot encoded labels in array.

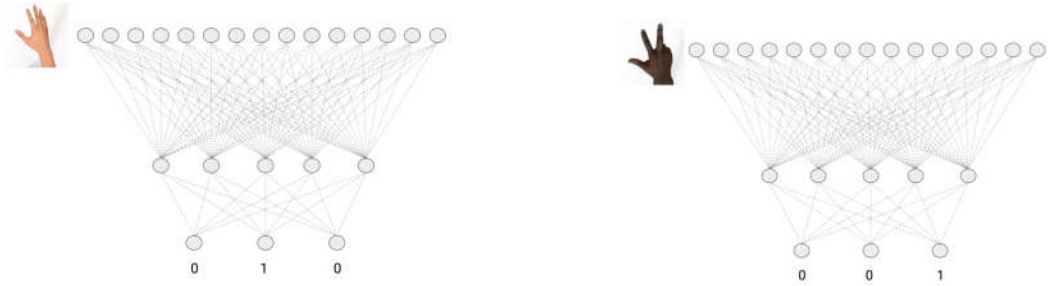
```
const convertedData =
  trainingData.map(({xs, ys}) => {
    const labels = [
      ys.species == "setosa" ? 1 : 0,
      ys.species == "virginica" ? 1 : 0,
      ys.species == "versicolor" ? 1 : 0 ]

    return{ xs: Object.values(xs), ys:Object.values(labels)};

  }).batch(10);
```

1.3.4 One-hot Encoding

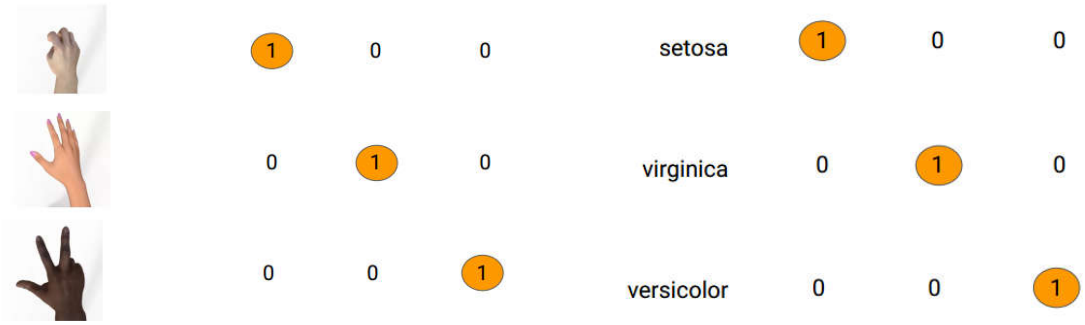
Imagine a neural network that performs three classifications using three output neurons like this: the rock, paper, and scissors. You want to have the first neuron close to one and the others close to zero representing a rock.



And, the second neuron close to one and the others close to zero representing paper. Finally in this case for scissors, you would want the third neuron close to one and the others closer to zero.



Thus, if we feed labels into the neural network when training it that represent the desired outputs, we would encode them in the representation. That's one-hot encoding, i.e. one of the values in the array is the hot value, In this case, we're doing the same thing with the three species of iris.



The *const* labels is an array with three elements. First element is one if the species is *setosa* and zero otherwise.

The second element is one if the species is *virginica* and zero otherwise.

```
const convertedData =
  trainingData.map(({xs, ys}) => {
    const labels = [
      ys.species == "setosa" ? 1 : 0,
      ys.species == "virginica" ? 1 : 0,
      ys.species == "versicolor" ? 1 : 0 ]

    return{ xs: Object.values(xs), ys:Object.values(labels)};

  }).batch(10);
```

```
const convertedData =
  trainingData.map(({xs, ys}) => {
    const labels = [
      ys.species == "setosa" ? 1 : 0,
      ys.species == "virginica" ? 1 : 0,
      ys.species == "versicolor" ? 1 : 0 ]

    return{ xs: Object.values(xs), ys:Object.values(labels)};

  }).batch(10);
```

Finally, the third element is one if the species is versicolor, and zero otherwise.

```
const convertedData =
  trainingData.map(({xs, ys}) => {
    const labels = [
      ys.species == "setosa" ? 1 : 0,
      ys.species == "virginica" ? 1 : 0,
      ys.species == "versicolor" ? 1 : 0 ]

    return{ xs: Object.values(xs), ys:Object.values(labels)};

  }).batch(10);
```

1.3.5 **Practice Quiz**: One-Hot Encoding

Question 1

Given a set of Xs and Ys, how would you one-hot encode a label based on the text or ‘rock’, ‘paper’, ‘scissors’ ?

```
trainingData.map(({xs, ys}) =>{
                    labels=[ys.label == ? 1 : 0, "rock",
                    ys.label == ? 1 : 0, "paper",
                    ys.label == ? 1 : 0, "scissors"]}
```

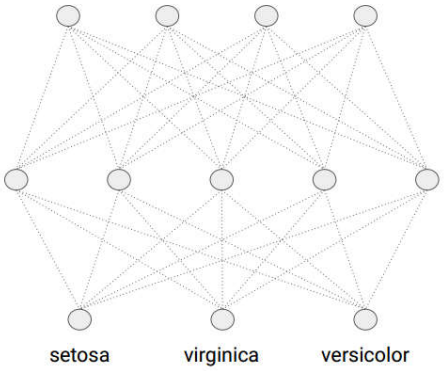
```
trainingData.map(({xs, ys}) => {
    const labels = [
    ys.label == "rock",
                    "paper",
                    scissors"]}
```

```
trainingData.map(({xs, ys}) => {
    const labels = [
            ys.label == "rock" == 1,
            ys.label == "paper" == 1,
            ys.label == "scissors" == 1 ]}
```

```
trainingData.map(({xs, ys}) => {
    const labels = [
            ys.label == "rock" ? 1 : 0,
            ys.label == "paper" ? 1 : 0,
            ys.label == "scissors" ? 1 : 0 ]}
```

1.3.6 Designing the NN

Here is the neural network. At the top are the four features, in the middle is a hidden layer with five nodes, and at the bottom are the three nodes that we'll use for the classification.



Here is the code:

- ***tf.sequential***: defines the model to be a sequential.
- ***model.add 1***: add one hidden dense layer with five neurons by specifying the input shape with the number of features.
- ***model.add 2***: add last dense layer with three neurons activating them with a Softmax function to get the probability of the flower class.
- ***model.compile.***: compile the model with a loss function (*categorocalCrossentropy*) and optimizer (*adam =0.06*).
- ***model.summary***: outputs the summary of the model

```
const model = tf.sequential();

model.add(tf.layers.dense({
      inputShape: [numOfFeatures],
      activation: "sigmoid", units: 5}))

model.add(tf.layers.dense({activation: "softmax", units: 3}));
```

```
model.compile({
    loss: "categoricalCrossentropy",
    optimizer: tf.train.adam(0.06)}
```

To tarin the model, we use *model.fitDataset* by passing the convertedData at the first parameter. Then you pass a list of JSON style name values, such as epochs, callbacks.

```
await model.fitDataset(
    convertedData,
    {
        epochs:100,
        callbacks:{
        onEpochEnd: async(epoch, logs) =>{
          console.log("E: " + epoch + " Loss: " + logs.loss);
        }
    }
});
```

To do inference and get a prediction, we can create an input tensor with feature values and pass it to the prediction method.

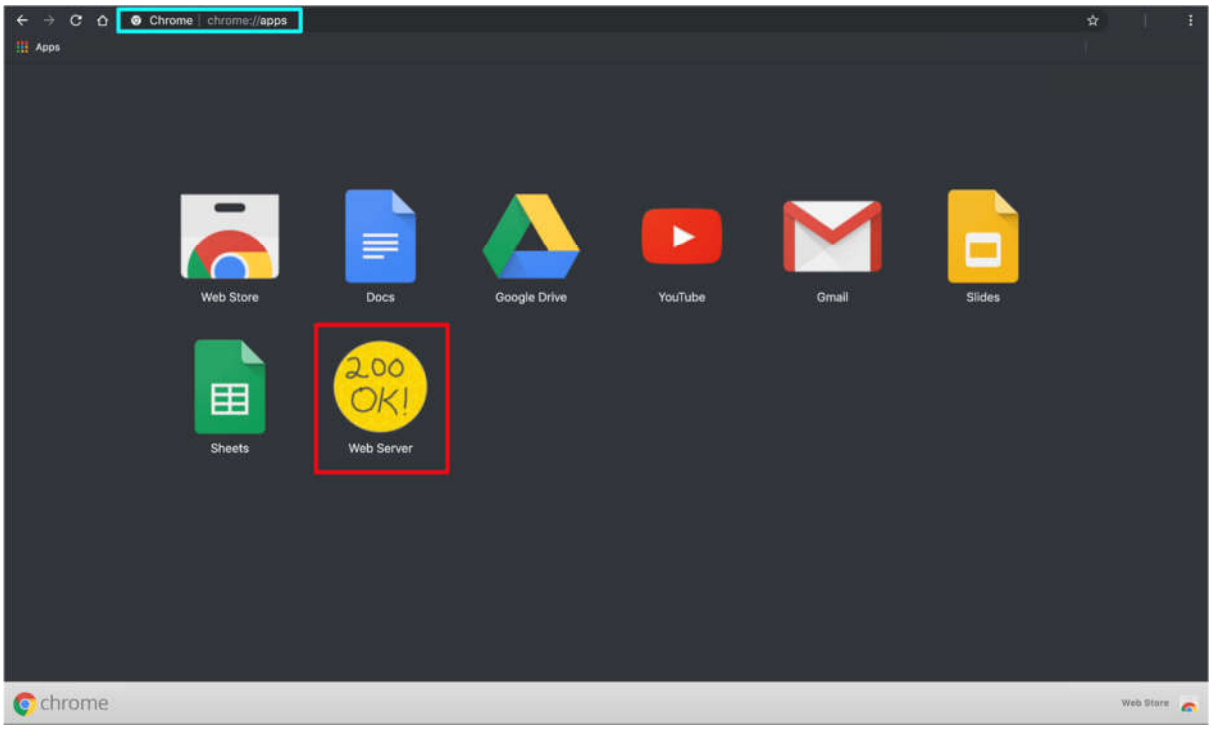
- ***tf.tensor2d***: declares the four feature values and the shape of that tensor.
- ***model.predict***: gets a Tensor back with a prediction in it after pass the test value to it.

```
const testVal = tf.tensor2d([5.8, 2.7, 5.1, 1.9], [1, 4]);
const prediction = model.predict(testVal);
alert(prediction);
```

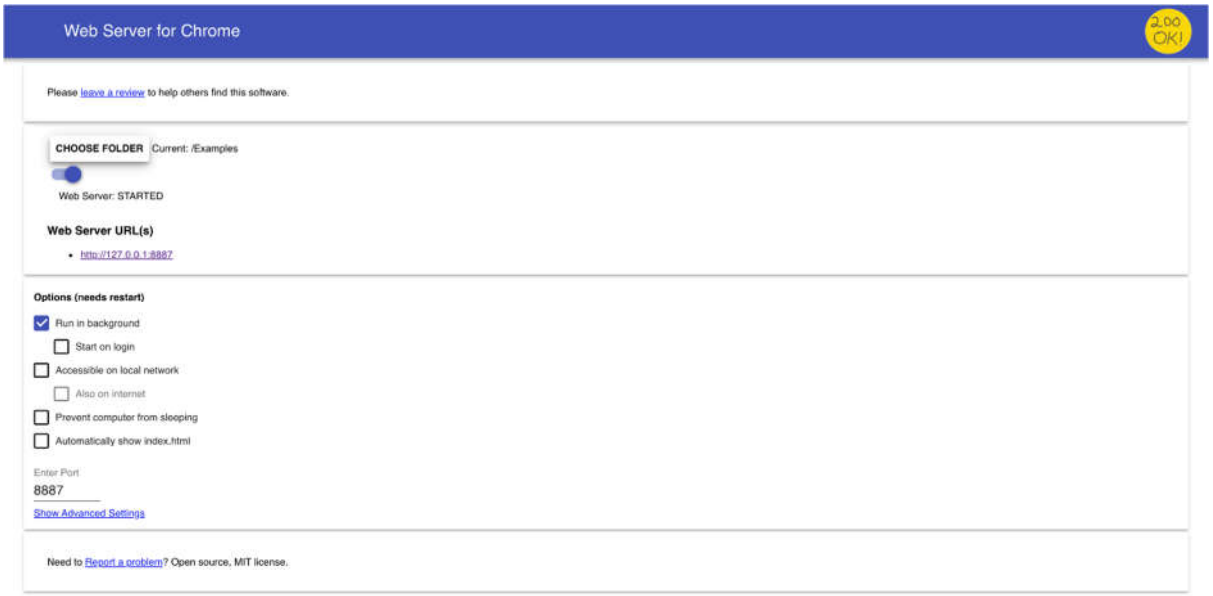
1.3.7 **Reading**: Using the Web Server

For the remaining exercises you will run a web server locally on your machine. In javascript every call needs to be done through and HTTP request, even if you have your files locally, you can't just load them directly, you need to make an HTTP request to those files. This is what the Web Server for Chrome App is used for.

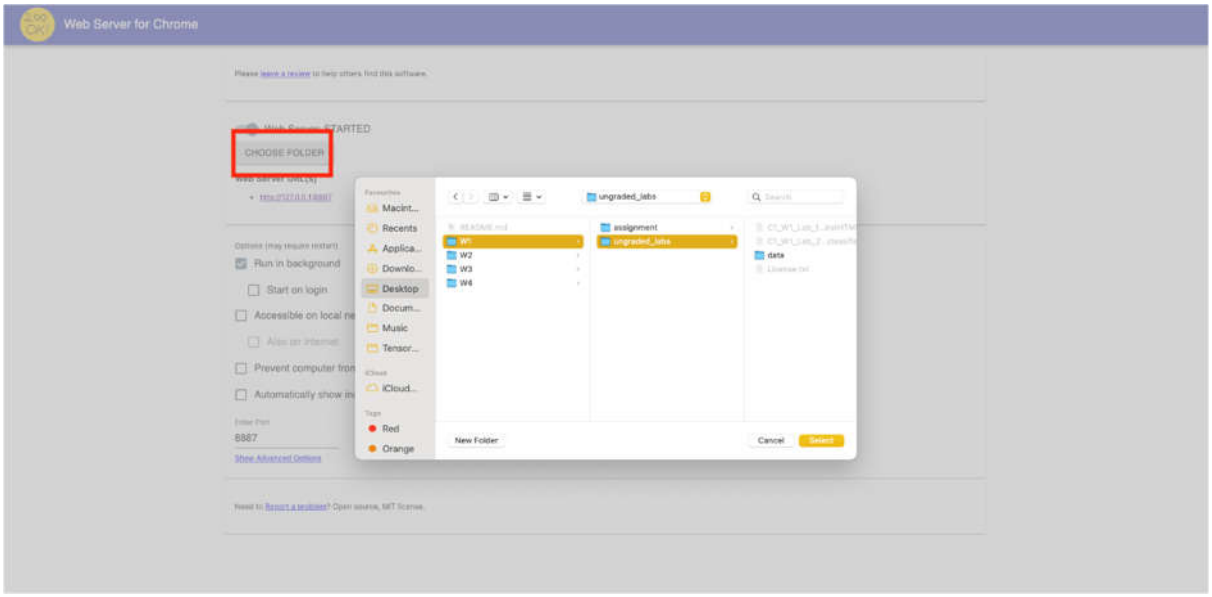
1. Open the Chrome browser and go to the Chrome apps (chrome://apps/):



2. Click on the Web Server icon to launch it.



3. Click on "CHOOSE FOLDER" and select the folder that contains the ungraded lab(s) or programming assignment. For this example, we will run the **C1_W1_Lab_2_iris_classifier.html** file. On my computer, this file is in the following folder:
tensorflow-2-public/C1_Browser-based-TF-JS/W1/ungraded_labs/
Therefore, I will select that folder.



4. Once you have chosen the correct folder, you can click on the Web Server URL (http://127.0.0.1:8887).

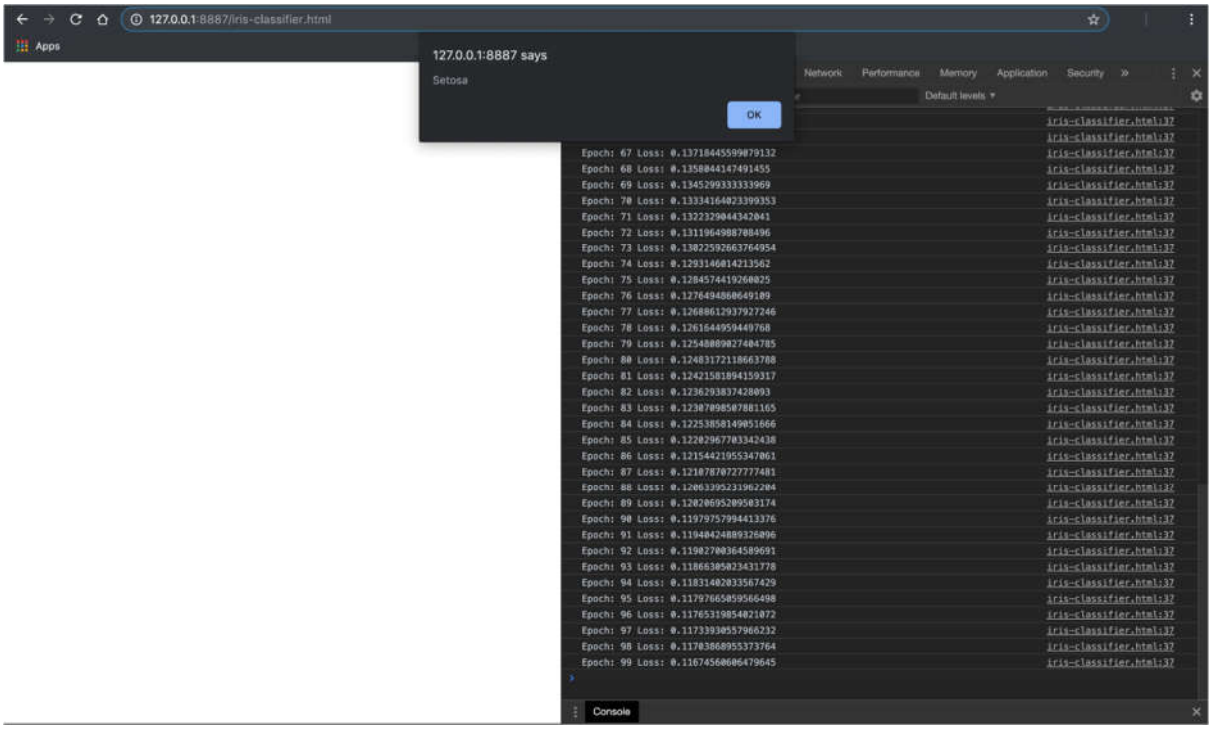
5. Once you click on the Web Server URL, this will open a new tab in your Chrome browser (if Chrome is your default browser). You can now click on the **html** file you want to run. In this case, we are going to run the **C1_W1_Lab_2_iris_classifier.html** file.

1.3.8 Iris Classifier

In the next example, we will create a neural network that can classify the 3 types of Iris plants found in the Iris dataset. You can use Brackets to open the **C1_W1_Lab_2_iris_classifier.html** file. You can find the **C1_W1_Lab_2_iris_classifier.html** file in the following folder in the GitHub repository for this course:

tensorflow-2-public/C1_Browser-based-TF-JS/W1/ungraded_labs/

When you launch the **C1_W1_Lab_2_iris_classifier.html** file in the Chrome browser (using the Web Server) make sure to open the Developer Tools to see the output in the Console. After training has finished, the code will alert the name of the predicted Iris plant. As you can see below, in this example, the predicted Iris plant is a Setosa.



1.3.9 Iris Classifier in Code

```
<html>
<head></head>
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
    <script lang="js">
        async function run(){
            const csvUrl = '/data/iris.csv';
            const trainingData = tf.data.csv(csvUrl, {
                columnConfigs: {
                    species: {
                        isLabel: true
                    }
                }
            });

            const numOfFeatures = (await trainingData.columnNames()).length - 1;
            const numOfSamples = 150;
            const convertedData =
                  trainingData.map(({xs, ys}) => {
                      const labels = [
                            ys.species == "setosa" ? 1 : 0,
                            ys.species == "virginica" ? 1 : 0,
                            ys.species == "versicolor" ? 1 : 0
                      ]
                      return{ xs: Object.values(xs), ys: Object.values(labels)};
                  }).batch(10);

            const model = tf.sequential();
            model.add(tf.layers.dense({inputShape: [numOfFeatures], activation: "sigmoid", units: 5}))
            model.add(tf.layers.dense({activation: "softmax", units: 3}));

            model.compile({loss: "categoricalCrossentropy", optimizer: tf.train.adam(0.06)});

            await model.fitDataset(convertedData,
                             {epochs:100,
                              callbacks:{
                                  onEpochEnd: async(epoch, logs) =>{
                                      console.log("Epoch: " + epoch + " Loss: " + logs.loss);
                                  }
                              }});

            // Test Cases:

            // Setosa
            const testVal = tf.tensor2d([4.4, 2.9, 1.4, 0.2], [1, 4]);

            // Versicolor
            // const testVal = tf.tensor2d([6.4, 3.2, 4.5, 1.5], [1, 4]);

            // Virginica
            // const testVal = tf.tensor2d([5.8,2.7,5.1,1.9], [1, 4]);

            const prediction = model.predict(testVal);
            const pIndex = tf.argMax(prediction, axis=1).dataSync();

            const classNames = ["Setosa", "Virginica", "Versicolor"];

            // alert(prediction)
            alert(classNames[pIndex])

        }
        run();
    </script>
<body>
</body>
</html>
```

1.3.10 **Reading**: Week 1 Wrap up

This week you got an introduction to Machine Learning in the browser with TensorFlow.js. You saw how to build your first basic model -- an iris classifier -- by reading data from a CSV, one hot encoding the labels, and then training a model to recognize future data as a type of Iris plant. Next week you'll take this further by using Convolutional Neural Networks in the browser that can recognize images of handwritten digits!

1.4 Lecture Notes (Optional)

1.4.1 **Ungraded External Tool**: Lecture Notes W1

You can download this week's lecture notes from this post in our Discourse community.

1.5 Graded Exercise -Breast Cancer Classification

1.51 Programming Assignment: Week 1-Breast Cancer Classification

```
<html>
<head></head>
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
    <script lang="js">

        async function run(){
            const trainingUrl = 'wdbc-train.csv';

            // Take a look at the 'wdbc-train.csv' file and specify the column
            // that should be treated as the label in the space below.
            // HINT: Remember that you are trying to build a classifier that
            // can predict from the data whether the diagnosis is malignant or benign.
            const trainingData = tf.data.csv(trainingUrl, {

                // YOUR CODE HERE
                columnConfigs: {
                    diagnosis: {
                        isLabel: true
                    }
                }

            });

            const numOfFeatures = (await trainingData.columnNames()).length - 1
            console.log(numOfFeatures);

            // Convert the training data into arrays in the space below.
            // Note: In this case, the labels are integers, not strings.
            // Therefore, there is no need to convert string labels into
            // a one-hot encoded array of label values like we did in the
            // Iris dataset example.
            const convertedTrainingData = trainingData.map(
                ({xs, ys}) => {
                    return {xs:Object.values(xs), ys:Object.values(ys)};
                }

            ).batch(10);


            // console.log(convertedTrainingData);

            const testingUrl = 'wdbc-test.csv';

            // Take a look at the 'wdbc-test.csv' file and specify the column
            // that should be treated as the label in the space below..
            // HINT: Remember that you are trying to build a classifier that
            // can predict from the data whether the diagnosis is malignant or benign.
            const testingData = tf.data.csv(testingUrl, {

                // YOUR CODE HERE
                columnConfigs: {
                    diagnosis: {
                        isLabel: true
                    }
                }

            });

            // Convert the testing data into arrays in the space below.
            // Note: In this case, the labels are integers, not strings.
            // Therefore, there is no need to convert string labels into
            // a one-hot encoded array of label values like we did in the
            // Iris dataset example.
            const convertedTestingData = testingData.map(
                ({xs, ys}) => {
                    return {xs:Object.values(xs), ys:Object.values(ys)};
                }

            ).batch(10);


            // Specify the number of features in the space below.
            // HINT: You can get the number of features from the number of columns
            // and the number of labels in the training data.
            // console.log(numOfFeatures);

            // In the space below create a neural network that predicts 1 if the diagnosis is malignant
            // and 0 if the diagnosis is benign. Your neural network should only use dense
            // layers and the output layer should only have a single output unit with a
            // sigmoid activation function. You are free to use as many hidden layers and
            // neurons as you like.
            // HINT: Make sure your input layer has the correct input shape. We also suggest
            // using ReLu activation functions where applicable. For this dataset only a few
            // hidden layers should be enough to get a high accuracy.
            const model = tf.sequential();

            // YOUR CODE HERE
            model.add(tf.layers.dense({inputShape: [numOfFeatures],units: 32, activation: "relu"}));
            model.add(tf.layers.dense({units: 1, activation: "sigmoid"}));


            console.log(model.summary());

            // Compile the model using the binaryCrossentropy loss,
            // the rmsprop optimizer, and accuracy for your metrics.
            model.compile({loss: "binaryCrossentropy", optimizer: tf.train.rmsprop(0.05), metrics: ['accuracy']});


            await model.fitDataset(convertedTrainingData,
                                {epochs:100,
                                 validationData: convertedTestingData,
                                 callbacks:{
                                     onEpochEnd: async(epoch, logs) =>{
                                         console.log("Epoch: " + epoch + " Loss: " + logs.loss + " Accuracy: " + logs.acc);
                                     }
                                 }});
            await model.save('downloads://my_model');

        }
        run();
    </script>
<body>
</body>
</html>
```

Week 2: Image Classification in the Browser

This week we'll look at Computer Vision problems, including some of the unique considerations when using JavaScript, such as handling thousands of images for training. By the end of this module, you will know how to build a site that lets you draw in the browser and recognizes your handwritten digits!

Learning Objectives

- Use tf-vis to visualize the output of callbacks
- Use a convolutional neural network to build a handwriting classifier
- Use a sprite sheet to train a classifier

2.1 Creating Convolutional Neural Networks in JavaScript

2.1.1 Introduction, A conversation with Andrew Ng

This week, you will build a handwriting recognition model using MNIST dataset and CNNs in a web browser. That allows to draw a character yourself to predict result in the web browser. You will learn to use spritesheet to load image data for model training in the web browser.

2.1.2 Creating a Convolutional Net with JavaScript

- ***tf.sequential:*** defines a sequential model
- ***tf.layers.conv2d:*** is a 2D convolutional network.
  - *inputShape*: defines the shape of the input; *[28,28,1]* means the image size is 28*28.
  - *kernelSize*: defines the size of convolutional filter. 3 means 3*3 filters.
  - *filters:* try to learn the convolutions from.
  - *activation: 'relu'* means to filter out values that less than 0.
- ***tf.layers.maxPooling2d:*** *poolSize:[2,2]* means 2*2 pool.
- ***tf.layers.flatten:*** defines the flatten layer. It takes pixels (28*28) flatten them out.
- ***tf.layers.dense:*** defines dense layer.
  - *units*: defines the number of neurals.
  - *activity*: defines the activity function.

```
model = tf.sequential();

model.add(tf.layers.conv2d({inputShape: [28, 28, 1],
          kernelSize: 3, filters: 8, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.conv2d({filters: 16,
          kernelSize: 3, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.flatten());

model.add(tf.layers.dense({units: 128, activation: 'relu'}));

model.add(tf.layers.dense({units: 10, activation: 'softmax'}));
```

Compile model by specifying a loss function, an optimizer, and metrics in a dictionary using name colon value. Note: parameters are passed in using a JavaScript hence the braces.

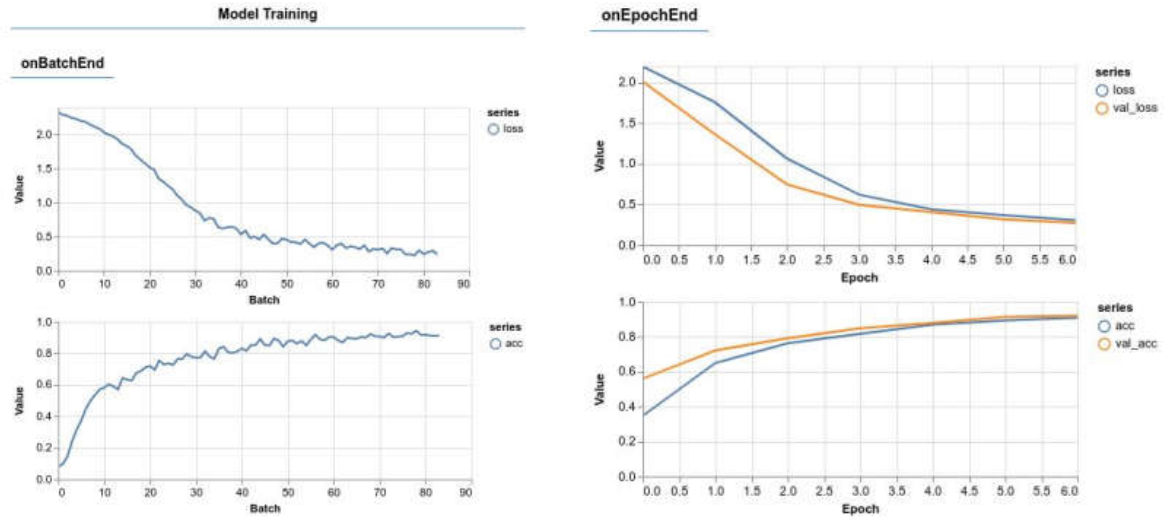
```
model.compile(
    {  optimizer: tf.train.adam(),
       loss: 'categoricalCrossentropy',
       metrics: ['accuracy']
    });
```

Train model using *model.fit* by passing training data, labels, and a dictionary of parameters:

- ***batchSize:*** train data using a subset
- ***validationData:*** train and validate the model to report the accuracy.
- ***epochs:*** specify the number of epochs
- ***shuffle:*** randomize data to prevent over-fitting of the multiple similar classes in the same batch.
- ***callbacks:*** update the user on training status in the training cycle.

```
model.fit(trainXs, trainYs, {
    batchSize: BATCH_SIZE,
    validationData: [testXs, testYs],
    epochs: 20,
    shuffle: true,
    callbacks: fitCallbacks
});
```

In tenSorFlow.js, *tf.vis* library can be used to render the outputs of the callbacks.



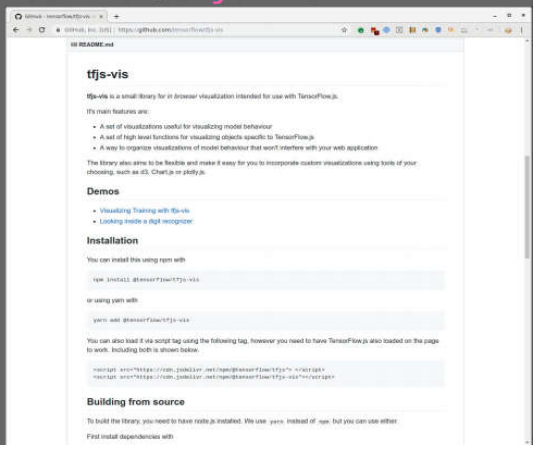
2.1.3 Visualizing the Training Process

tf.vis is one of the tools in javaScript to visualize training process. To include the library called *tf.vis*, using this code in the script.

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

The library is open source, and you can access it at this Github address.

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



In *model.fit* method, the callbacks parameter is configured to an object called ***fitCallBacks***.

```
model.fit(trainXs, trainYs, {
    batchSize: BATCH_SIZE,
    validationData: [testXs, testYs],
    epochs: 20,
    shuffle: true,
    callbacks: fitCallbacks
});
```

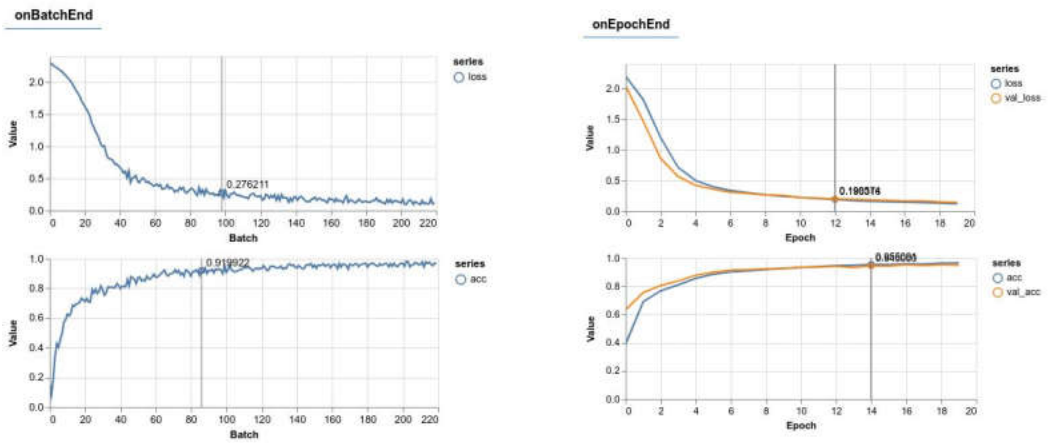
Declare *tfvis.show.fitCallbacks by* passing a container and metrics and return to *fitCallbacks.*

```
const fitCallbacks = tfvis.show.fitCallbacks(container, metrics);
```

Metrics can capture the *loss, val_loss, acc, val_acc; container* can set a name and any required style. The visualization library will create the DOM elements to render the details.

```
const metrics = ['loss', 'val_loss', 'acc', 'val_acc'];
const container = { name: 'Model Training', styles: { height: '1000px' } };
const fitCallbacks = tfvis.show.fitCallbacks(container, metrics);
```

When you are training, the callback will create a container and draw the feedback as this.



2.1.4 **Reading**: tfjs-vis Documentation

To learn more about tfjs-vis, please visit the tfjs-vis GitHub repository.

2.2 Using a Sprite Sheet

2.2.1 What Is a Sprite Sheet?

A sprite sheet is an image that consists of several smaller images (sprites) and/or animations. Combining the small images in one big image improves the game performance, reduces the memory usage, and speeds up the start-up and loading time of the game.

2.2.2 **Reading**: MNIST Sprite Sheet

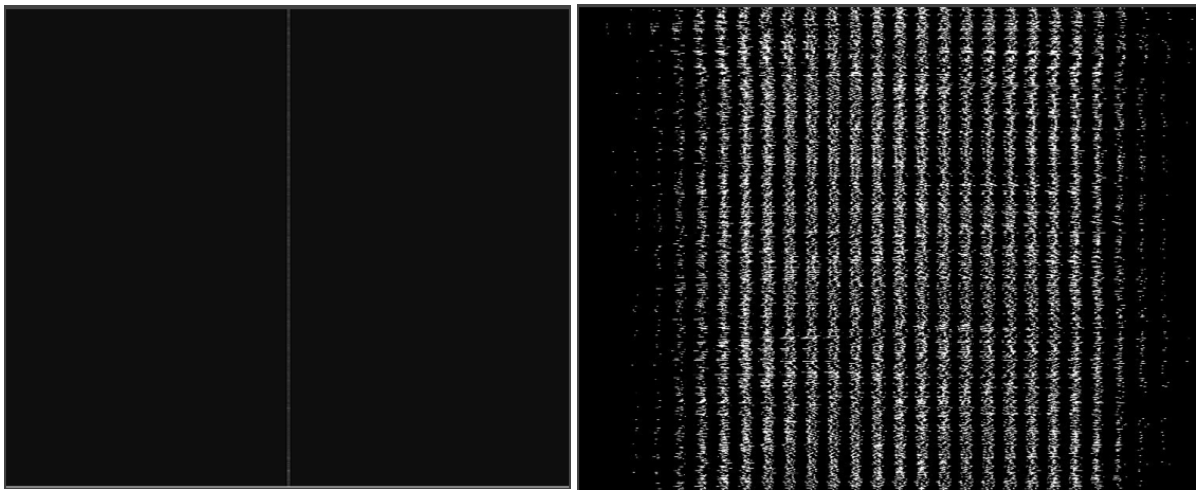
You can find the MNIST Sprite Sheet in this link.

2.2.3 Using the Sprite Sheet

Here's the URL hosting the sprite sheet with all the MNIST images in it.

```
https://storage.googleapis.com/learnjs-data/model-builder/mnist_images.png
```

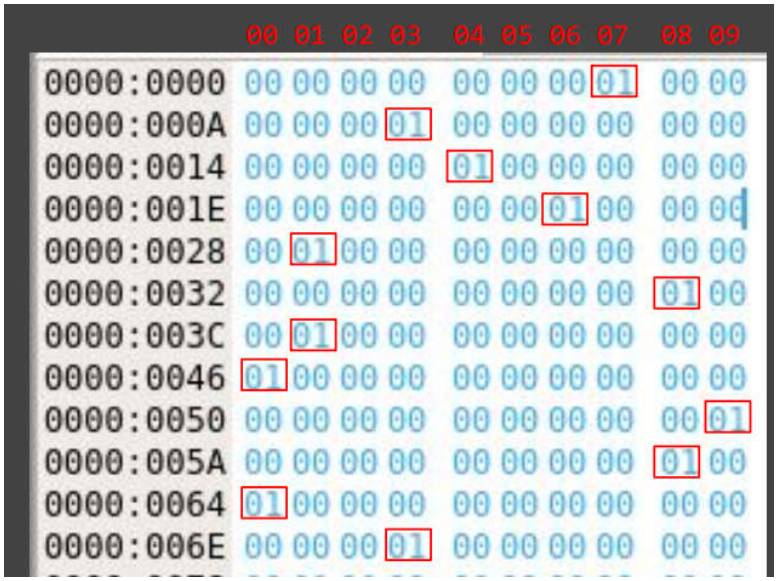
If you visit that URL, you'll see something like this. That's an image that 65,000 pixels tall with one pixel for every image in the database. If you zoom in, you'll see something like this. Remember, every row of pixels here is the flattened version of a 28 by 28 image.



The label can be found at this URL. When you visit it, nothing will render in the browser, but a file will download. The file is 650,000 bytes in size, which means that it is 10 bytes per image.

```
https://storage.googleapis.com/learnjs-data/model-builder/mnist_labels_uint8
```

You need a hex viewer to understand the contents. With the 10 bytes per label, you have nine bytes: 00 and one byte: 01 (the label value). Here: the first image is 7, then 3, and then 4, etc. It's a very inefficient coding from a file size perspective, but it's very easy to transform it into a one-hot encoding in memory because it's a serialized one-hot encode already.



Class *MnistData* in data.js includes three methods: *load(), nextTrainBatch(), nextTestBatch()*.

- ***load()*:** download sprite sheet and labels and use *nextBatch* to decode them.
- ***nextTrainBatch()*:** gets the next batch of training data according to the desired batch size. Note: it keeps the data as 1*768 pixels and then resizes to 28*28 by the calling function.
- ***nextTestBatch()*:** gets the next batch of training data.

```
export class MnistData {
...
    async load() {
    // Download the sprite and slice it
    // Download the labels and decode them
    }

    nextTrainBatch(){
    // Get the next training batch
    }

    nextTestBatch(){
    // Get the next test batch
    }
}
```

Using this code to initialize the data class and load the sprite.

```
const data = new MnistData();
await data.load();
```

Get the batch and resize them to the desired shape of 28*28.

```
const [trainXs, trainYs] = tf.tidy(() => {
    const d = data.nextTrainBatch(TRAIN_DATA_SIZE);
    return [
        d.xs.reshape([TRAIN_DATA_SIZE, 28, 28, 1]),
        d.labels
    ];
});
```

2.3 MNIST Classifier

2.3.1 Using tf.tidy() to Save Memory

This function will create an array containing the set of traingXs and trainYs

- ***nextTrainBatch*:** gets the next train batch from data source. By default, with MNIST, the train data size is 5,500, so it's basically getting 5,500 lines of 784 bites.
- ***d.xs.reshape***: reshapes the data using into a 4D tensor with 5500 in the first dimension, then 28 *28 representing the image, and then one representing the color depth. It will return that as the first element in the array, mapping to train Xs.
- ***d.labels***: returns labels as the second element in the array, because the labels are already one-hot encoded.
- ***tf.tidy:*** cleans up all those intermediate tensors (except those returns) once the execution is done to save memory.

```
const [trainXs, trainYs] = tf.tidy(() => {
    const d = data.nextTrainBatch(TRAIN_DATA_SIZE);
    return [
        d.xs.reshape([TRAIN_DATA_SIZE, 28, 28, 1]),
        d.labels
    ];
});
```

2.3.2 A Few Words from Laurence

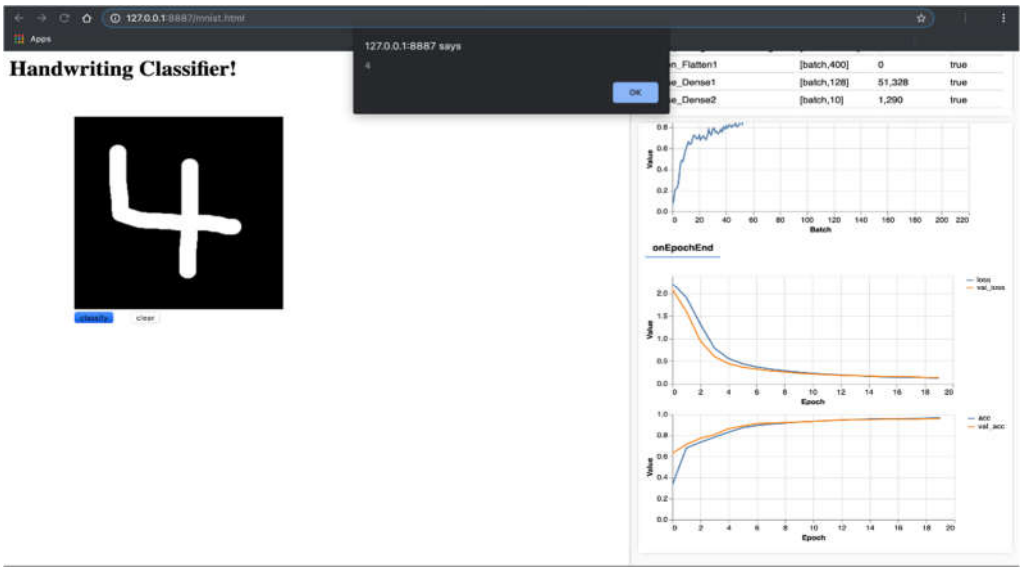
In these lessons, you saw how convolutional neural networks work in JavaScript, but you also got a tour of how to build a complete application that is trained to recognize handwriting.

2.3.3 **Reading**: MNIST Classifier

In the next example, we will create a neural network that can classify the images of handwritten digits from the MNIST dataset. You can use Brackets to open the **script.js** file and find the **script.js** file in the following folder in the GitHub repository for this course:

tensorflow-2-public/C1_Browser-based-TF-JS/W2/ungraded_lab/

When you launch the **C1_W2_Lab_1_mnist.html** file in the Chrome browser (using the Web Server), tfjs-vis will automatically display the model architecture and the training progress. Once training has finished, you can draw digits on the black rectangle to be classified. After drawing a digit, and pressing the "classify" button, the code will alert the predicted digit. As you can see below, in this example, the predicted digit is a 4.



2.3.4 MNIST Classifier in CODE

File 1: nmist.html

- line 3: the TensorFlow JS latest;
- line 4: the TensorFlow JS visualization (report the process of the training);
- line 9: draw something;
- line 10: convert to image;
- line 11-12: two buttons (one for classifying and one for clearing the canvas);
- line 13: data.js (has the Mnist class)
- line 14: script.js (handle the Canvas, inference, training)

```
1  <html>
2  <head>
3      <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
4      <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs-vis"></script>
5
6  </head>
7  <body>
8      <h1>Handwriting Classifier!</h1>
9      <canvas id="canvas" width="280" height="280" style="position:absolute;top:100;left:100;border:8px solid;"></canvas>
10     <img id="canvasimg" style="position:absolute;top:10%;left:52%;width=280;height=280;display:none;">
11     <input type="button" value="classify" id="sb" size="48" style="position:absolute;top:400;left:100;">
12     <input type="button" value="clear" id="cb" size="23" style="position:absolute;top:400;left:180;">
13     <script src="data.js" type="module"></script>
14     <script src="script.js" type="module"></script>
15 </body>
16 </html>
17
```

File 2: data.js

Downloading the image sprite, labels to put them into an array in memory, From the array index, it slices the data and take slices out of return to the training.

- line 27-28: the image path
- line 29-30: the label path

```
27  const MNIST_IMAGES_SPRITE_PATH =
28      'https://storage.googleapis.com/learnjs-data/model-builder/mnist_images.png';
29  const MNIST_LABELS_PATH =
30      'https://storage.googleapis.com/learnjs-data/model-builder/mnist_labels_uint8';
```

- line 44-69: download image, slice out and puts it into an array in memory.

```
44  async load() {
45    // Make a request for the MNIST sprited image.
46    const img = new Image();
47    const canvas = document.createElement('canvas');
48    const ctx = canvas.getContext('2d');
49    const imgRequest = new Promise((resolve, reject) => {
50      img.crossOrigin = '';
51      img.onload = () => {
52        img.width = img.naturalWidth;
53        img.height = img.naturalHeight;
54
55        const datasetBytesBuffer =
56            new ArrayBuffer(NUM_DATASET_ELEMENTS * IMAGE_SIZE * 4);
57
58        const chunkSize = 5000;
59        canvas.width = img.width;
60        canvas.height = chunkSize;
61
62        for (let i = 0; i < NUM_DATASET_ELEMENTS / chunkSize; i++) {
63          const datasetBytesView = new Float32Array(
64              datasetBytesBuffer, i * IMAGE_SIZE * chunkSize * 4,
65              IMAGE_SIZE * chunkSize);
66          ctx.drawImage(
67              img, 0, i * chunkSize, img.width, chunkSize, 0, 0, img.width,
68              chunkSize);
69
```

- line 85-87: download label (one-hot encoded).

```
85      const labelsRequest = fetch(MNIST_LABELS_PATH);
86      const [imgResponse, labelsResponse] =
87          await Promise.all([imgRequest, labelsRequest]);
```

- line 106-113: take the batch size to slice it and return a set of images of the training set
- line 115-120: take the batch size to slice it and return a set of images of the test set
- line 123-137: manage the index within the data array to get the slice index for the next batch

```
106  nextTrainBatch(batchSize) {
107    return this.nextBatch(
108        batchSize, [this.trainImages, this.trainLabels], () => {
109          this.shuffledTrainIndex =
110              (this.shuffledTrainIndex + 1) % this.trainIndices.length;
111          return this.trainIndices[this.shuffledTrainIndex];
112        });
113  }
114
115  nextTestBatch(batchSize) {
116    return this.nextBatch(batchSize, [this.testImages, this.testLabels], () => {
117      this.shuffledTestIndex =
118          (this.shuffledTestIndex + 1) % this.testIndices.length;
119      return this.testIndices[this.shuffledTestIndex];
120    });
121  }
122
123  nextBatch(batchSize, data, index) {
124    const batchImagesArray = new Float32Array(batchSize * IMAGE_SIZE);
125    const batchLabelsArray = new Uint8Array(batchSize * NUM_CLASSES);
126
127    for (let i = 0; i < batchSize; i++) {
128      const idx = index();
129
130      const image =
131          data[0].slice(idx * IMAGE_SIZE, idx * IMAGE_SIZE + IMAGE_SIZE);
132      batchImagesArray.set(image, i * IMAGE_SIZE);
133
134      const label =
135          data[1].slice(idx * NUM_CLASSES, idx * NUM_CLASSES + NUM_CLASSES);
136      batchLabelsArray.set(label, i * NUM_CLASSES);
137    }
```

File 3: script.js

- line 113: when load the document, it will call the run() function.

```
113   document.addEventListener('DOMContentLoaded', run);
```

- line 104: create an object – a new instance of MnistData
- line 105: download the images and get them into memory
- line 106: call getModel() function
- line 107: show the architecture.
- line 108: an asynchronous function. It calls train() function by passing the model and data.
- line 109: call init() function

```
103 async function run() {
104       const data = new MnistData();
105       await data.load();
106       const model = getModel();
107       tfvis.show.modelSummary({name: 'Model Architecture'}, model);
108       await train(model, data);
109       init();
110       alert("Training is done, try classifying your handwriting!");
111   }
112
```

- line 8: create a new sequential model
- line 10: add the layer to that convolutional 2D
- line 11: add layer to pooling 2D
- line 14: Flatten
- line 15: the hidden layer of dense with 128 units
- line 16: the output layer with 10 units
- line 18: compile the model by specifying the optimizer, loss, and metrics.
- line 20: return model back to the caller

```
7  function getModel() {
8      model = tf.sequential();
9
10     model.add(tf.layers.conv2d({inputShape: [28, 28, 1], kernelSize: 3, filters: 8, activation: 'relu'}));
11     model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));
12     model.add(tf.layers.conv2d({filters: 16, kernelSize: 3, activation: 'relu'}));
13     model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));
14     model.add(tf.layers.flatten());
15     model.add(tf.layers.dense({units: 128, activation: 'relu'}));
16     model.add(tf.layers.dense({units: 10, activation: 'softmax'}));
17
18     model.compile({optimizer: tf.train.adam(), loss: 'categoricalCrossentropy', metrics: ['accuracy']});
19
20     return model;
21 }
```

- line 23-48: received the model and data to train the model
- line 24-26: setup the callbacks
    - line 24: load measurements (loss, validation loss, accuracy, validation accuracy) to a list
    - line 25: call model training
    - line 26: specify how it called, pass the container and metrics
- line 28-30: specify batch size, training data size, and test data size
- line 32-38: create the list of train xs and ys
    - line 33: get the next training batch from the data
    - line 35: reshape image size
- line 40-46: create a list of test xs and ys.
    - line 41: get the next test batch from the data
    - line 46: tidy up to clear the memory
- line 48-55: train the model by giving it the train xs and ys

- line 50: specify the validation data to be the test xs and ys
- line 51: set the epochs to be 20
- line 52: shuffle data
- line 53: call fit.callbacks

```
23  async function train(model, data) {
24      const metrics = ['loss', 'val_loss', 'acc', 'val_acc'];
25      const container = { name: 'Model Training', styles: { height: '640px' } };
26      const fitCallbacks = tfvis.show.fitCallbacks(container, metrics);
27
28      const BATCH_SIZE = 512;
29      const TRAIN_DATA_SIZE = 5500;
30      const TEST_DATA_SIZE = 1000;
31
32      const [trainXs, trainYs] = tf.tidy(() => {
33          const d = data.nextTrainBatch(TRAIN_DATA_SIZE);
34          return [
35              d.xs.reshape([TRAIN_DATA_SIZE, 28, 28, 1]),
36              d.labels
37          ];
38      });
39
40      const [testXs, testYs] = tf.tidy(() => {
41          const d = data.nextTestBatch(TEST_DATA_SIZE);
42          return [
43              d.xs.reshape([TEST_DATA_SIZE, 28, 28, 1]),
44              d.labels
45          ];
46      });
47
48      return model.fit(trainXs, trainYs, {
49          batchSize: BATCH_SIZE,
50          validationData: [testXs, testYs],
51          epochs: 20,
52          shuffle: true,
53          callbacks: fitCallbacks
```

- line 87-100: set up the UI
- line 88: UI has the Canvas to draw on
- line 93-95: mouse move, down, enter to draw with mouse
- line 96-97: save button to do inference
- line 98-99: clear button to clear the canvas.

```
87  function init() {
88      canvas = document.getElementById('canvas');
89      rawImage = document.getElementById('canvasimg');
90      ctx = canvas.getContext("2d");
91      ctx.fillStyle = "black";
92      ctx.fillRect(0,0,280,280);
93      canvas.addEventListener("mousemove", draw);
94      canvas.addEventListener("mousedown", setPosition);
95      canvas.addEventListener("mouseenter", setPosition);
96      saveButton = document.getElementById('sb');
97      saveButton.addEventListener("click", save);
98      clearButton = document.getElementById('cb');
99      clearButton.addEventListener("click", erase);
100 }
101
102
103 async function run() {
104     const data = new MnistData();
105     await data.load();
106     const model = getModel();
107     tfvis.show.modelSummary({name: 'Model Architecture'}, model);
108     await train(model, data);
109     init();
110     alert("Training is done, try classifying your handwriting!");
111 }
```

- line 80-85: do the inference for us

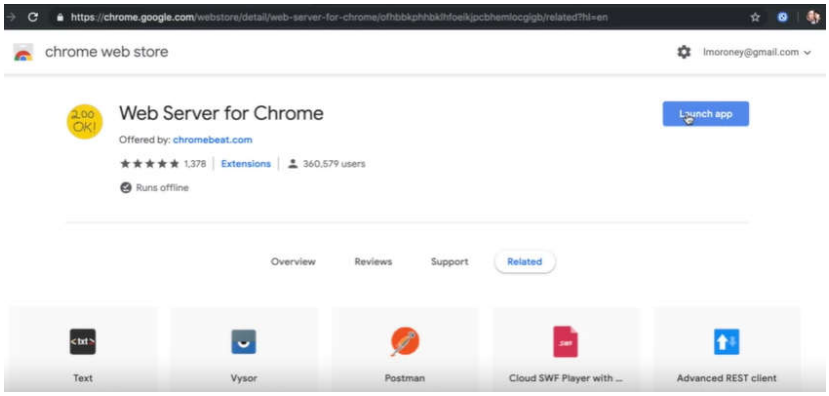
- line 81: do inference by passing it the raw image. 1 means one color channel black and white.
- line 82: resize image just created into 28*28
- line 83: specify the tensor to resize it by adding another dimension
- line 84: pass the Tensor result

```
80 function save() {
81     var raw = tf.browser.fromPixels(rawImage,1);
82     var resized = tf.image.resizeBilinear(raw, [28,28]);
83     var tensor = resized.expandDims(0);
84     alert(model.predict(tensor));
85 }
86
87 function init() {
88     canvas = document.getElementById('canvas');
89     rawImage = document.getElementById('canvasimg');
90     ctx = canvas.getContext("2d");
91     ctx.fillStyle = "black";
92     ctx.fillRect(0,0,280,280);
93     canvas.addEventListener("mousemove", draw);
94     canvas.addEventListener("mousedown", setPosition);
95     canvas.addEventListener("mouseenter", setPosition);
96     saveButton = document.getElementById('sb');
97     saveButton.addEventListener("click", save);
98     clearButton = document.getElementById('cb');
99     clearButton.addEventListener("click", erase);
100 }
```

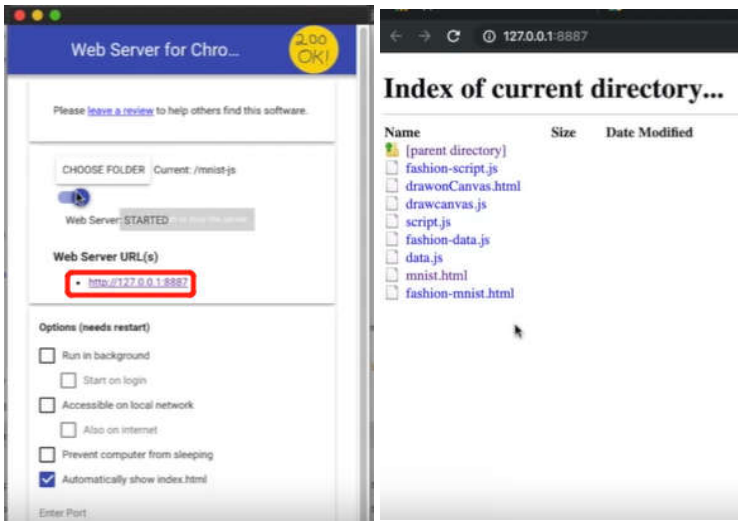
- line 62-73: move the mouse around and draw on the Canvas
- line 72: call raw image and copy the contents of the Canvas as a PNG

```
62 function draw(e) {
63     if(e.buttons!=1) return;
64     ctx.beginPath();
65     ctx.lineWidth = 24;
66     ctx.lineCap = 'round';
67     ctx.strokeStyle = 'white';
68     ctx.moveTo(pos.x, pos.y);
69     setPosition(e);
70     ctx.lineTo(pos.x, pos.y);
71     ctx.stroke();
72     rawImage.src = canvas.toDataURL('image/png');
73 }
74
75 function erase() {
76     ctx.fillStyle = "black";
77     ctx.fillRect(0,0,280,280);
78 }
79
80 function save() {
81     var raw = tf.browser.fromPixels(rawImage,1);
82     var resized = tf.image.resizeBilinear(raw, [28,28]);
83     var tensor = resized.expandDims(0);
84     alert(model.predict(tensor));
85 }
```

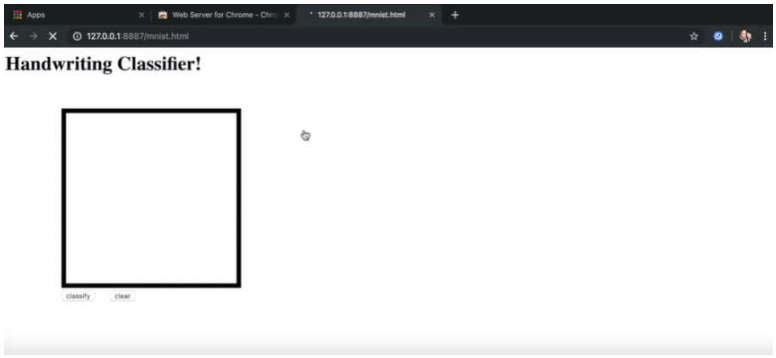
To run the code, you need a web server. To install the web server for Chrome.



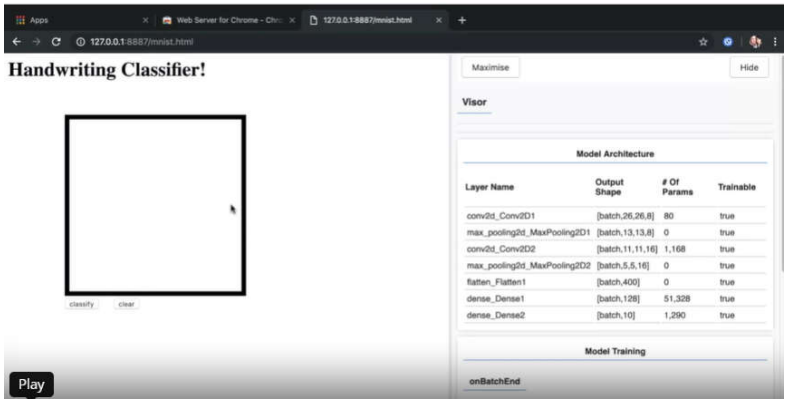
This server allows you to choose a folder on your directory. Select the URL, you will get the directory.



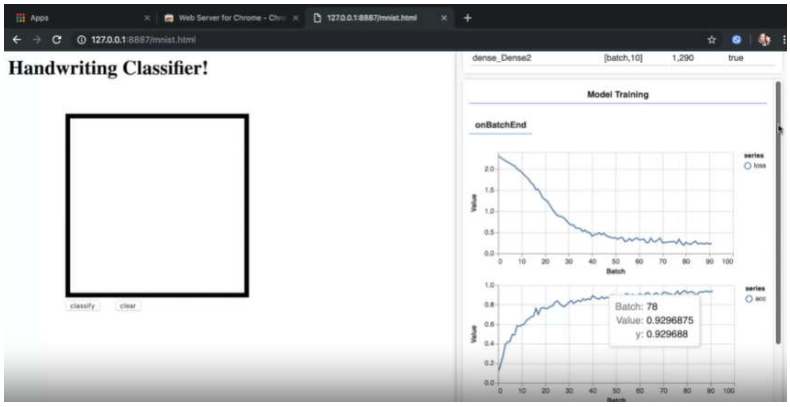
Click mnist.html to launch the server. This is the handweriting classifier. The canvas is currently whie.

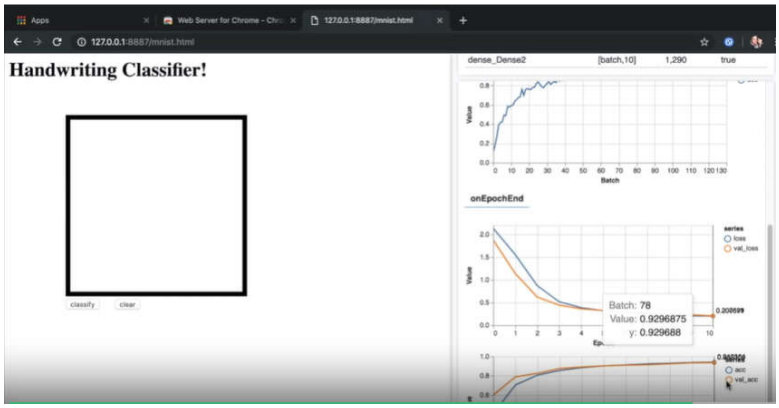


Now, you can start to do the training. Here is the model visualization and model summary.

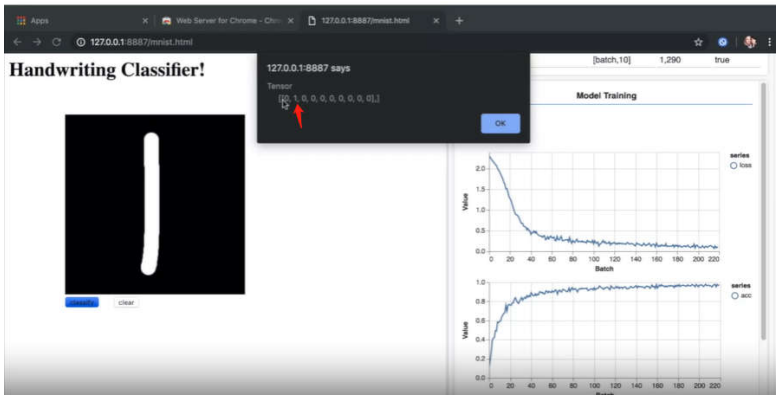


Here, you can see the progress of the training.





After the training finished, you can use Canvas to draw a digit to make prediction.



2.3.5 **Quiz**: Week 2 Quiz

Question 1

What is the correct syntax for the first layer in a convolutional neural network that takes an MNIST (28x28 monochrome) input?

```
model.add(tf.layers.conv2d({inputShape: [28, 28, 1], kernelSize: 3, filters: 8, activation: 'relu'}));
```

```
model.add(tf.layers.conv({inputShape: (28, 28, 1), kernelSize: 3, filters: 8, activation: 'relu'}));
```

```
model.add(tf.layers.conv2d({inputShape: [28, 28], kernelSize: 3, filters: 8, activation: 'relu'}));
```

```
model.add(tf.layers.conv({inputShape: [28, 28, 1], kernelSize: 3, filters: 8, activation: 'relu'}));
```

Question 2

What is the correct syntax for adding a maxPooling2D layer to a Convolutional neural network in JavaScript?

```
model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));
```

```
model.add(tf.layers.maxPooling2d({poolSize = [2, 2]}));
```

```
model.add(tf.layers.maxPooling2D({poolSize: [2, 2]}));
```

```
model.add(tf.layers.maxPooling2D({poolSize =  [2, 2]}));
```

Question 3

What is the correct syntax for compiling a model with an optimizer, loss function and metrics?

```
model.compile({  optimizer: tf.train.adam(), loss: 'categoricalCrossentropy', metrics: ['accuracy']});
```

```
model.compile({  optimizer: tf.train.adam(); loss: 'categoricalCrossentropy'; metrics: ['accuracy']});
```

```
model.compile({  tf.optimizer: tf.train.adam(), tf.loss: 'categoricalCrossentropy', tf.metrics: ['accuracy']});
```

```
model.compile({  optimizer = tf.train.adam(), loss = 'categoricalCrossentropy', metrics = ['accuracy']});
```

Question 4

How do you correctly pass a set of validation data called textXs and testYs to the model.fit method in JavaScript?

Use validationData= [testXs, testYs] and pass it to the model.fit method

Use validationData: [testXs, testYs] in the list of parameters sent as the third parameter to model.fit

Use validationData = [testXs, testYs] in the list of parameters to model.fit

Use validationData: [testXs, testYs] in the list of parameters to model.fit

Question 5

How do you get the built in callbacks visualizer with TensorFlow.js?

Include the tfjs-vis script, set a callback in model.fit, and set it to a const that called show.fitCallbacks() on the tfvis object

Include the tfjs-vis script, call show.fitCallbacks() on the tfvis object

Include the tfjs-vis script and it will work automatically

Include the tfjs-vis script, set a callback in model.fit and it will work automatically

Question 6

If you want to see loss, validation loss, accuracy and validation accuracy on each epoch while training, how do you do this?

Create a list containing text values [“loss=true”, “val_loss=true”, “acc=true”, “val_acc=true”] and pass it to fitCallbacks() as a parameter

Create a list containing text values with the names of the analytics you want to capture, i.e. [‘loss’, ‘val_loss’, ‘acc’, ‘val_acc’] and pass it to fitCallbacks() as a parameter

Create a list setting loss=true, val_loss=true, acc=true, val_acc=true, and pass it to the fitCallbacks() as a parameter

Create a list containing [1, 1, 1, 1] indicating that you want those 4 values to be true and pass it to the fitCallbacks() as a parameter

Question 7

When using a dataset like MNIST or FashionMNIST, why is it advisable to use a sprite sheet containing all the images?

It keeps the data in the native JS format

It prevents excessive multiple HTTP calls to download the data

It doesn’t require any additional pre-processing

It makes the data more secure

Question 8

What is the role of tf.tidy() in TensorFlow.js?

When it is executed, it cleans up all intermediate tensors allocated by a function except those returned by the function

When it is executed, it removes everything tensorflow from the browser memory and cache

It shuts down tensorflow when done, cleaning up all memory

When it is executed, it clears memory for new tensors

2.3.6 **Reading**: Week 2 Wrap up

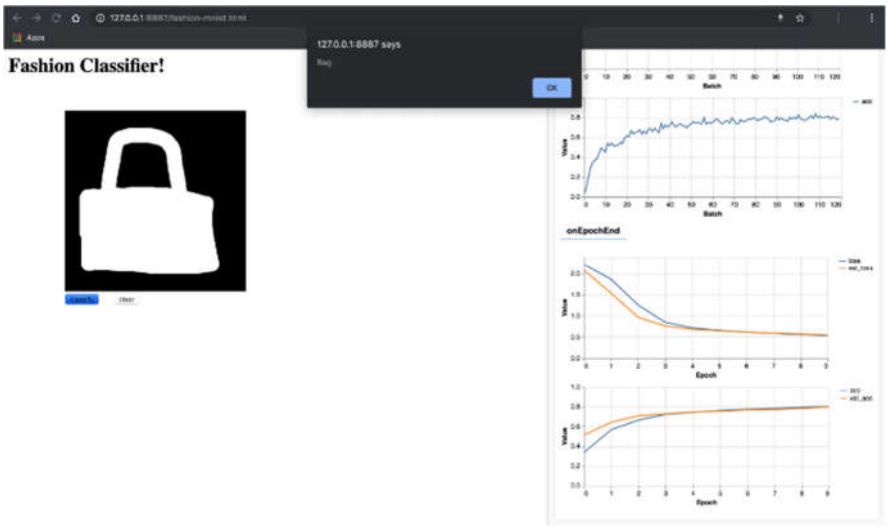
2.4 Lecture Notes (Optional)

2.4.1 **Ungraded External Tool**: Lecture Notes W2

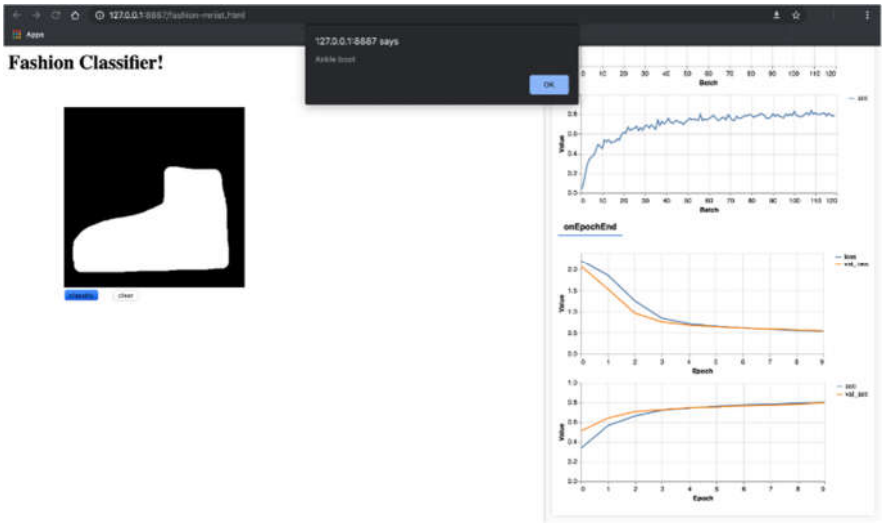
2.5 Graded Exercise – Fashion MNIST Classifier

2.5.1 **Reading**: Exercise Description

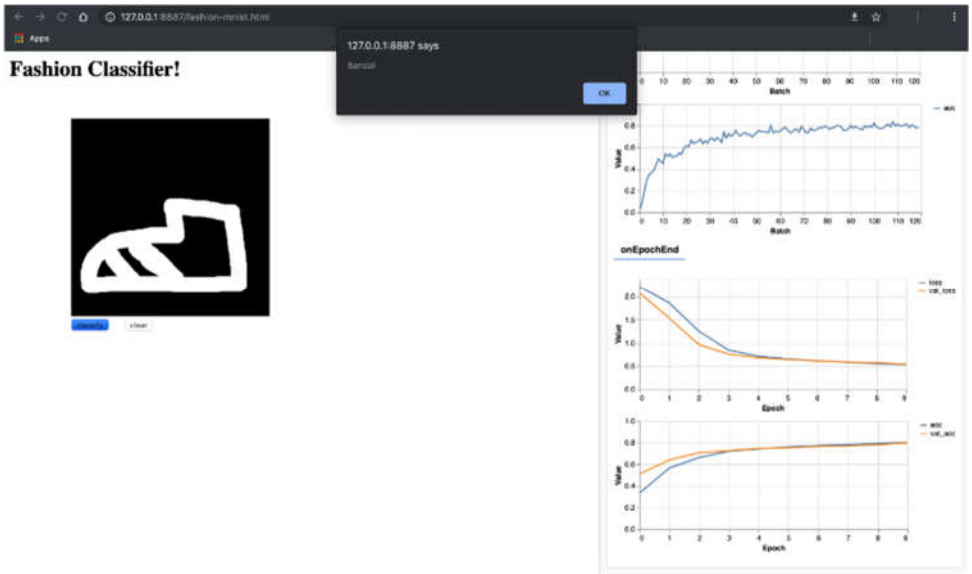
In this week's exercise, you will try to build and train a neural network that can classify the images in the Fashion MNIST dataset. You can use Brackets to open the **C1_W2_Assignment.js** file. You can find the **C1_W2_Assignment.js** file in the following folder in the GitHub repository for this course:
tensorflow-2-public/C1_Browser-based-TF-JS/W2/assignment/
Follow the instructions to complete the exercise. Once you have filled-in the missing code, run the **fashion-mnist.html** file in the Chrome browser using the Web Server. When you launch the **fashion-mnist.html** file, tfjs-vis will automatically display the model architecture and the training progress. Once training has finished, you draw any of the 10 articles of clothing from the Fashion MNIST dataset on the black rectangle to be classified. After drawing a clothing article, and pressing the "classify" button, the code will alert the predicted article of clothing. Below are some examples:
1. In this example, the predicted article of clothing is a Bag.



2. In this example, the predicted article of clothing is an Ankle Boot.



3. In this example, the predicted article of clothing is a Sandal.



2.5.2 **Programming Assignment**: Week 2-Fashion MNIST Classifier

fashion-mnist.html

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

</head>
<body>
    <h1>Fashion Classifier!</h1>
    <canvas id="canvas" width="280" height="280" style="position:absolute;top:100;left:100;border:8px solid;"></canvas>
    <img id="canvasimg" style="position:absolute;top:10%;left:52%;width:280;height:280;display:none;">
    <input type="button" value="classify" id="sb" size="48" style="position:absolute;top:400;left:100;">
    <input type="button" value="clear" id="cb" size="23" style="position:absolute;top:400;left:180;">
    <script src="fashion-data.js" type="module"></script>
    <script src="C1_W2_Assignment.js" type="module"></script>
    <!-- <script src="C1_W2_Assignment_Solution.js" type="module"></script> -->
</body>
</html>

```

fashion-data.js

```
const IMAGE_SIZE = 784;
const NUM_CLASSES = 10;
const NUM_DATASET_ELEMENTS = 70000;

const TRAIN_TEST_RATIO = 1 / 7;

const NUM_TRAIN_ELEMENTS = Math.floor(TRAIN_TEST_RATIO * NUM_DATASET_ELEMENTS);
const NUM_TEST_ELEMENTS = NUM_DATASET_ELEMENTS - NUM_TRAIN_ELEMENTS;

const MNIST_IMAGES_SPRITE_PATH =
    'https://storage.googleapis.com/learnjs-data/model-builder/fashion_mnist_images.png';
const MNIST_LABELS_PATH =
    'https://storage.googleapis.com/learnjs-data/model-builder/fashion_mnist_labels_uint8';

/**
 * A class that fetches the sprited MNIST dataset and returns shuffled batches.
 *
 * NOTE: This will get much easier. For now, we do data fetching and
 * manipulation manually.
 */
export class FMnistData {
  constructor() {
    this.shuffledTrainIndex = 0;
    this.shuffledTestIndex = 0;
  }

```

```
  async load() {
    // Make a request for the MNIST sprited image.
    const img = new Image();
    const canvas = document.createElement('canvas');
    const ctx = canvas.getContext('2d');
    const imgRequest = new Promise((resolve, reject) => {
      img.crossOrigin = '';
      img.onload = () => {
        img.width = img.naturalWidth;
        img.height = img.naturalHeight;

        const datasetBytesBuffer =
            new ArrayBuffer(NUM_DATASET_ELEMENTS * IMAGE_SIZE * 4);

        const chunkSize = 5000;
        canvas.width = img.width;
        canvas.height = chunkSize;

        for (let i = 0; i < NUM_DATASET_ELEMENTS / chunkSize; i++) {
          const datasetBytesView = new Float32Array(
              datasetBytesBuffer, i * IMAGE_SIZE * chunkSize * 4,
              IMAGE_SIZE * chunkSize);
          ctx.drawImage(
              img, 0, i * chunkSize, img.width, chunkSize, 0, 0, img.width,
              chunkSize);

          const imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);

          for (let j = 0; j < imageData.data.length / 4; j++) {
            // All channels hold an equal value since the image is grayscale, so
            // just read the red channel.
            datasetBytesView[j] = imageData.data[j * 4] / 255;
          }
        }
        this.datasetImages = new Float32Array(datasetBytesBuffer);

        resolve();
      };
      img.src = MNIST_IMAGES_SPRITE_PATH;
    });

    const labelsRequest = fetch(MNIST_LABELS_PATH);
    const [imgResponse, labelsResponse] =
        await Promise.all([imgRequest, labelsRequest]);

    this.datasetLabels = new Uint8Array(await labelsResponse.arrayBuffer());

    // Create shuffled indices into the train/test set for when we select a
    // random dataset element for training / validation.
    this.trainIndices = tf.util.createShuffledIndices(NUM_TRAIN_ELEMENTS);
    this.testIndices = tf.util.createShuffledIndices(NUM_TEST_ELEMENTS);

    // Slice the the images and labels into train and test sets.
    this.trainImages =
        this.datasetImages.slice(0, IMAGE_SIZE * NUM_TRAIN_ELEMENTS);
    this.testImages = this.datasetImages.slice(IMAGE_SIZE * NUM_TRAIN_ELEMENTS);
    this.trainLabels =
        this.datasetLabels.slice(0, NUM_CLASSES * NUM_TRAIN_ELEMENTS);
    this.testLabels =
        this.datasetLabels.slice(NUM_CLASSES * NUM_TRAIN_ELEMENTS);
  }

  nextTrainBatch(batchSize) {
    return this.nextBatch(
        batchSize, [this.trainImages, this.trainLabels], () => {
          this.shuffledTrainIndex =
              (this.shuffledTrainIndex + 1) % this.trainIndices.length;
          return this.trainIndices[this.shuffledTrainIndex];
        });
  }

```

```
  nextTrainBatch(batchSize) {
    return this.nextBatch(
        batchSize, [this.trainImages, this.trainLabels], () => {
          this.shuffledTrainIndex =
              (this.shuffledTrainIndex + 1) % this.trainIndices.length;
          return this.trainIndices[this.shuffledTrainIndex];
        });
  }

  nextTestBatch(batchSize) {
    return this.nextBatch(batchSize, [this.testImages, this.testLabels], () => {
      this.shuffledTestIndex =
          (this.shuffledTestIndex + 1) % this.testIndices.length;
      return this.testIndices[this.shuffledTestIndex];
    });
  }

  nextBatch(batchSize, data, index) {
    const batchImagesArray = new Float32Array(batchSize * IMAGE_SIZE);
    const batchLabelsArray = new Uint8Array(batchSize * NUM_CLASSES);

    for (let i = 0; i < batchSize; i++) {
      const idx = index();

      const image =
          data[0].slice(idx * IMAGE_SIZE, idx * IMAGE_SIZE + IMAGE_SIZE);
      batchImagesArray.set(image, i * IMAGE_SIZE);

      const label =
          data[1].slice(idx * NUM_CLASSES, idx * NUM_CLASSES + NUM_CLASSES);
      batchLabelsArray.set(label, i * NUM_CLASSES);
    }

    const xs = tf.tensor2d(batchImagesArray, [batchSize, IMAGE_SIZE]);
    const labels = tf.tensor2d(batchLabelsArray, [batchSize, NUM_CLASSES]);

    return {xs, labels};
  }
}

```

C1_W2_Assignment.js

```
import {FMnistData} from './fashion-data.js';
var canvas, ctx, saveButton, clearButton;
var pos = {x:0, y:0};
var rawImage;
var model;

```

```
function getModel() {
    model = tf.sequential();
    model.add(tf.layers.conv2d({inputShape: [28, 28, 1],
                                kernelSize: 3,
                                filters: 32,
                                activation: 'relu',
                                kernel_initializer: 'he_uniform'}));
    model.add(tf.layers.conv2d({kernelSize: 3,
                                filters: 32,
                                activation: 'relu',
                                kernel_initializer: 'he_uniform'}));
    model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));
    model.add(tf.layers.flatten());
    model.add(tf.layers.dense({units: 64, activation: 'relu'}));
    model.add(tf.layers.dense({units: 10, activation: 'softmax'}));
    model.compile({optimizer: tf.train.momentum(0.01, 0.9),
                   loss: 'categoricalCrossentropy',
                   metrics: ['accuracy']});
    return model;

}

```

```
async function train(model, data) {
    const metrics = ['loss', 'val_loss', 'accuracy', 'val_accuracy'];
    const container ={ name: 'Model Training', styles: { height: '1000px' } };
    const fitCallbacks = tfvis.show.fitCallbacks(container, metrics);
    const BATCH_SIZE = 256;
    const TRAIN_DATA_SIZE = 6000;
    const TEST_DATA_SIZE = 1000;

    const [trainXs, trainYs] = tf.tidy(() => {
        const d = data.nextTrainBatch(TRAIN_DATA_SIZE);
        return [
            d.xs.reshape([TRAIN_DATA_SIZE, 28, 28, 1]),
            d.labels
        ];
    });

    const [testXs, testYs] = tf.tidy(() => {
        const d = data.nextTestBatch(TEST_DATA_SIZE);
        return [
            d.xs.reshape([TEST_DATA_SIZE, 28, 28, 1]),
            d.labels
        ];
    });

    return model.fit(trainXs, trainYs, {
        batchSize: BATCH_SIZE,
        validationData: [testXs, testYs],
        epochs: 10,
        shuffle: true,
        callbacks: fitCallbacks
    });
}
```

```
function setPosition(e){
    pos.x = e.clientX-100;
    pos.y = e.clientY-100;
}

function draw(e) {
    if(e.buttons!=1) return;
    ctx.beginPath();
    ctx.lineWidth = 24;
    ctx.lineCap = 'round';
    ctx.strokeStyle = 'white';
    ctx.moveTo(pos.x, pos.y);
    setPosition(e);
    ctx.lineTo(pos.x, pos.y);
    ctx.stroke();
    rawImage.src = canvas.toDataURL('image/png');
}

function erase() {
    ctx.fillStyle = "black";
    ctx.fillRect(0,0,280,280);
}
```

```
function save() {
    var raw = tf.browser.fromPixels(rawImage,1);
    var resized = tf.image.resizeBilinear(raw, [28,28]);
    var tensor = resized.expandDims(0);

    var prediction = model.predict(tensor);
    var pIndex = tf.argMax(prediction, 1).dataSync();

    var classNames = ["T-shirt/top", "Trouser", "Pullover",
                      "Dress", "Coat", "Sandal", "Shirt",
                      "Sneaker",  "Bag", "Ankle boot"];


    alert(classNames[pIndex]);
}

function init() {
    canvas = document.getElementById('canvas');
    rawImage = document.getElementById('canvasimg');
    ctx = canvas.getContext("2d");
    ctx.fillStyle = "black";
    ctx.fillRect(0,0,280,280);
    canvas.addEventListener("mousemove", draw);
    canvas.addEventListener("mousedown", setPosition);
    canvas.addEventListener("mouseenter", setPosition);
    saveButton = document.getElementById('sb');
    saveButton.addEventListener("click", save);
    clearButton = document.getElementById('cb');
    clearButton.addEventListener("click", erase);
}


async function run() {
    const data = new FMnistData();
    await data.load();
    const model = getModel();
    tfvis.show.modelSummary({name: 'Model Architecture'}, model);
    await train(model, data);
    await model.save('downloads://my_model');
    init();
    alert("Training is done, try classifying your drawings!");
}

document.addEventListener('DOMContentLoaded', run);
```

Week 3: Converting Models to JSON Format

This week we'll see how to take models that have been created with TensorFlow in Python and convert them to JSON format so that they can run in the browser using Javascript. We will start by looking at two models that have already been pre-converted. One of them is going to be a toxicity classifier, which uses NLP to determine if a phrase is toxic in several categories; the other one is Mobilenet which can be used to detect content in images. By the end of this module, you will train a model in Python yourself and convert it to JSON format using the tensorflow.js converter.

Learning Objectives

- Use the tensorflow.js converter to convert a Keras model to JSON format
- Use a toxicity model to determine if a phrase is toxic in a number of categories
- Use Mobilenet to detect objects in images

3.1 Toxicity Classifier

3.1.1 A conversation with Andrew Ng

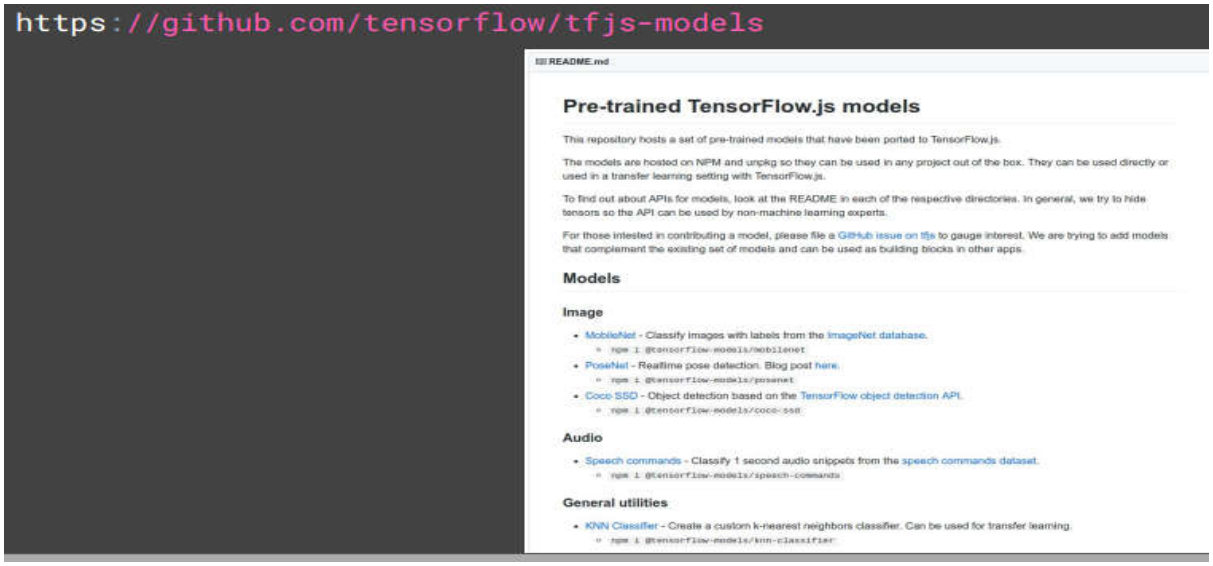
In this week, you'll learn how to take a model that's been trained in TensorFlow in Python and run that in a web browser using TensorFlow.js. This includes both a model that you could train yourself and convert, or taking a model that someone else has trained and just downloading and using that.

3.1.2 A Few Words from Laurance

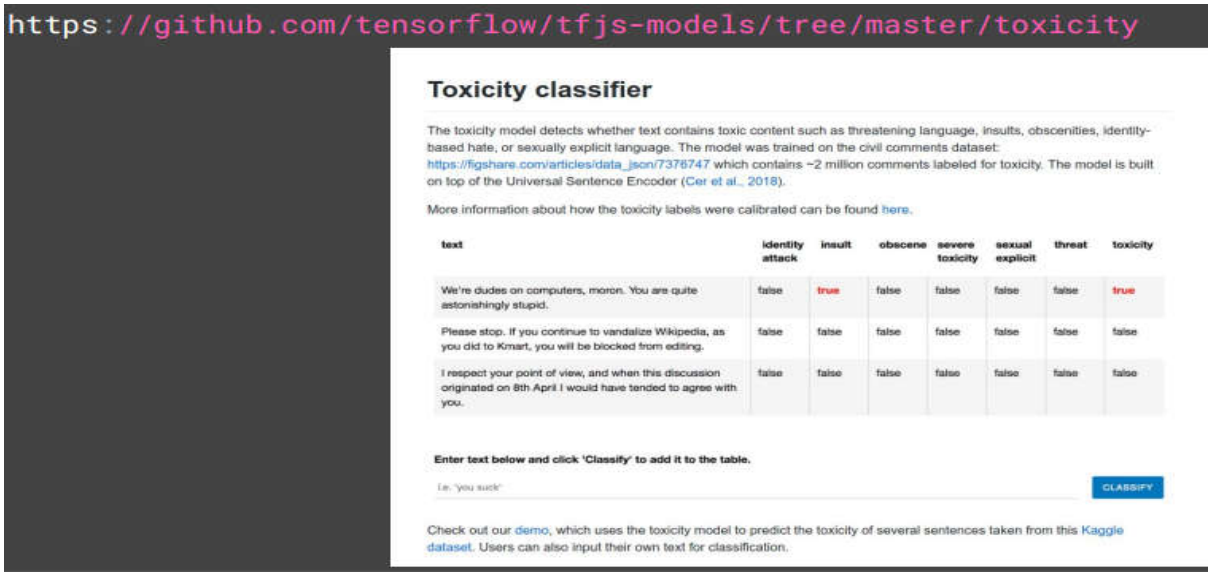
We will use existing trained model in the browser. We will learn how to convert pre-trained models into the format that can be usable in JavaScript. Let's start with some text-based models.

3.1.3 Pre-Trained TensorFlow.js Models

The models are hosted in GitHub at this URL. There are image classifiers, audio speech recognition and some texts utilities, etc.



We'll use the toxicity classifier which is available here. This model detects whether texts contain toxic content, threats, insults, obscenities, identity-based hate speech, and explicit language etc. It has been trained with a civil comments data set that contains about two million labelled comments.



3.1.4 **Reading**: Important Links

Please find here the links to the GitHub repository of pre-trained TensorFlow.js models and the toxicity classifier.

3.1.5 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 **C1_W3_Lab_1_toxicity_classifier.html** file. You can find the **C1_W3_Lab_1_toxicity_classifier.html** file in the following folder in the GitHub repository for this course:

tensorflow-2-public/C1_Browser-based-TF-JS/W3/ungraded_labs/

When you launch the **C1_W3_Lab_1_toxicity_classifier.html** file in the Chrome browser make sure to open the Developer Tools to see the output in the Console.

3.1.6 Toxicity Classifier

Using those scripts to get tensorflow.js latest and toxicity model.

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

You can take the name of the model at the end and go to the Github to check the model.

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

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 is a simple example, we add another script block to put the code.

```
<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>
```

First, you need to set the threshold value. This value is a minimum prediction confidence. If a prediction over this value, we will match it.

```
const threshold = 0.9;
```

Every prediction has two values, such as the insult and not-insults. If the not-insults prediction is greater than the threshold, toxicity will report not-insults by setting matched to false.

```
const threshold = 0.9;

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

Value for not-insult > threshold
So match = false

Similarly, if the insult is greater than the threshold, toxicity will report insult by setting match to true.

```
const threshold = 0.9;

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

Value for insult > threshold
So match = true

If neither is greater, then toxicity will set match to null

```
const threshold = 0.9;

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

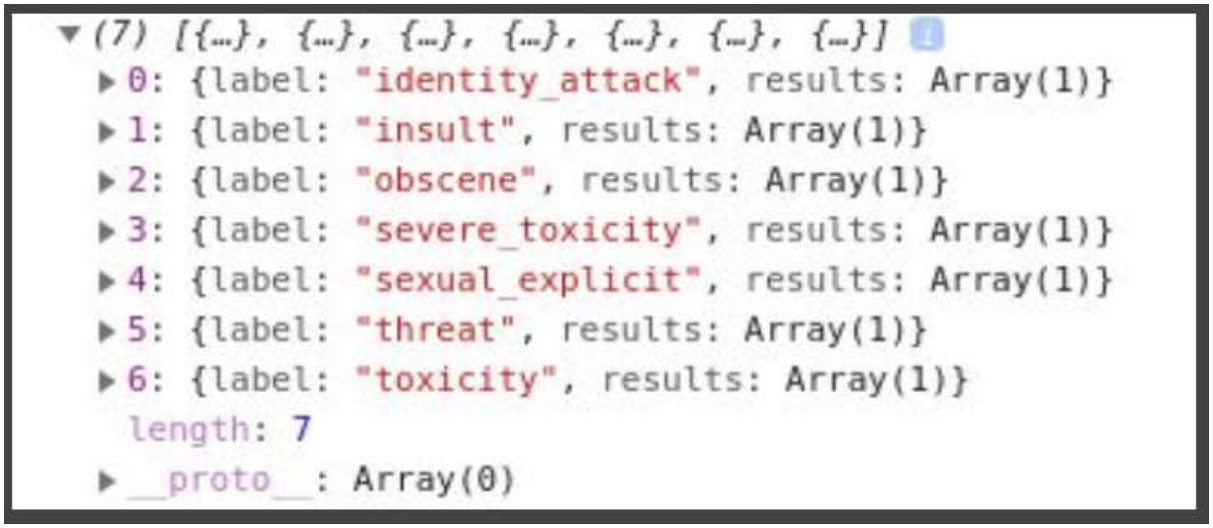
Neither value > threshold
So match = null

Here is the code to do a prediction on a sentence.

- Load the model, passing it the threshold value.
- Have the loaded model
- Create an array of sentences ("you suck!") for classification.
- Call *model.classify* passing it the sentences.
- Get a set of predictions back

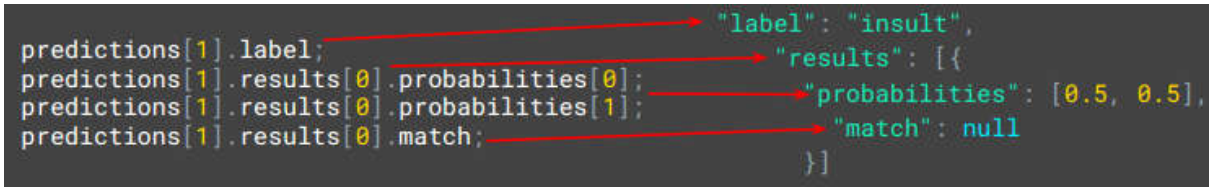
```
toxicity.load(threshold).then(model => {
  const sentences = ['you suck!'];
  model.classify(sentences).then(predictions => {
  // Handle Results
  });
});
```

Here is the console.log predictions result. It has seven different labels of prediction, and each of these contains a set of results.



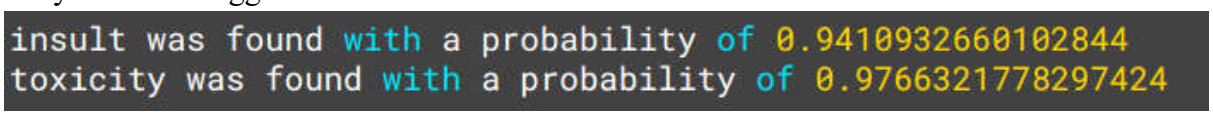
For the insult label, you can see it matches true (probabilities of 0.06 for not-insults and 0.94 for insults), which clearly shows the text was insulting.

To extract the label, we use the *label* property and the *results* array (contains an array of probabilities and match value), as shown in hard-coded of insults.



Using this code to iterate across all labels and report back on the ones with a match was true.

When I run it, the output in the console is like this. To classify multiple sentences, the results array would be bigger than one element.



3.1.7 Toxicity Classifier in Code

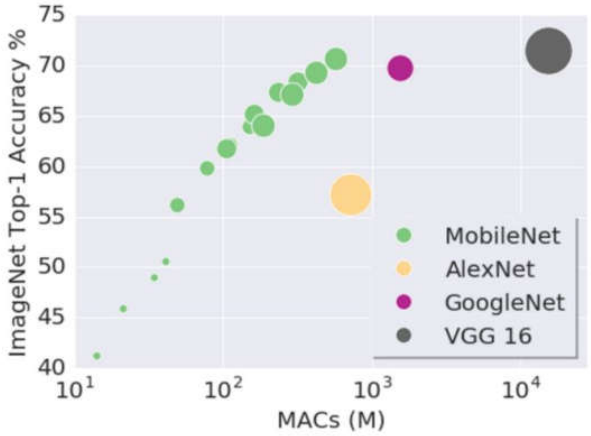
- line 3: load Tensorflow.js
- line 4: load toxicity model
- line 6: set up threshold value
- line 7: load toxicity model by passing the threshold
- line 8: define a sentence
- line 9: pass the sentence to classify for prediction
- line 10: logout those predictions
- line 11-17: iterate through the full list of 7 predictions
- line 12: match is true means a positive hit on the toxic behaviour
- line 13-15: logout the toxic behaviour the label and their actual probability

```
1  <html>
2  <head>
3  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
4  <script src="https://cdn.jsdelivr.net/npm/@tensorflow-models/toxicity"></script>
5  <script>
6  const threshold = 0.9;
7  toxicity.load(threshold).then(model => {
8      const sentences = ['you suck!'];
9      model.classify(sentences).then(predictions => {
10         console.log(predictions);
11         for(i=0; i<7; i++){
12             if(predictions[i].results[0].match){
13                 console.log(predictions[i].label +
14                     " was found with a probability of " +
15                     predictions[i].results[0].probabilities[1]);
16             }
17         }
18     });
19 });
20 </script>
21 </head>
22 <body></body>
23 </html>
```

3.2 Image Classification Using MobileNet

3.2.1 MobileNet

MobileNet is a small low-latency, low-power image classification library, which are trained to recognize a thousand classes. There are several versions, which are mainly used for classification, detection, embeddings, and other segmentation. This URL lists some the supported classes.




```
http://bit.ly/mobilenet-labels

                        00: background
                        01: tench
                        02: goldfish
                        03: great white shark
                        04: tiger shark
                        05: hammerhead
                        06: electric ray
                        07: stingray
                        08: cock
                        09: hen
                        10: ostrich
                        11: brambling
                        12: goldfinch
                        13: house finch
                        ...
```

3.2.2 **Reading**: Classes Supported by MobileNet

You can find a list of the classes supported by MobileNet in this link.

3.2.3 **Reading**: 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 **C1_W3_Lab_2_mobilenet.html** file. You can find the **C1_W3_Lab_2_mobilenet.html** file in the following folder in the GitHub repository:
tensorflow-2-public/C1_Browser-based-TF-JS/W3/ungraded_labs/
When you launch the **C1_W3_Lab_2_mobilenet.html** file in the Chrome browser make sure to open the Developer Tools to see the output in the Console.

3.2.4 Using MobileNet

To use the pre-trained mobilenet in JavaScript, you first need to include the script for TensorFlow.js and the mobilenet.

```
<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, we will use *mobilenet* model to classify the image and write result out. So in the *body* of my page, we need an image tag and a div tag (contain the output text).

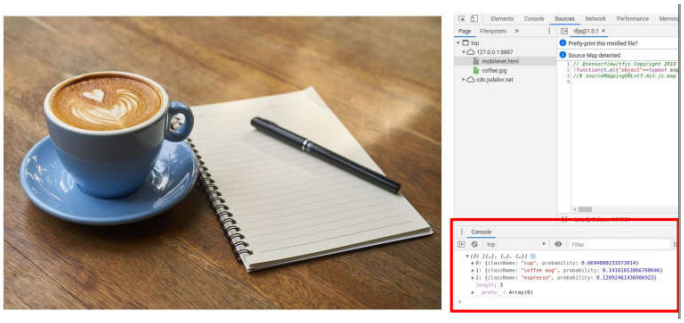
```
<body>
    <img id="img" src="coffee.jpg"></img>
    <div id="output" style="font-family:courier;font-size:24px;height=300px"></div>
</body>
```

Next, you'll pass the image to *mobilenet* and gets a set of classifications back. Note: this code should execute after the page has loaded (put it at the bottom of the page after the closing body tag, or call it when the DOM has finished loading):

- ***document.getElementById('img'):*** create a variable representing the image tag
- ***mobilenet.load:*** load mobilenet.
- ***then(model=>:*** after loaded model, it will be ready to use.
- ***model.classify(img):*** pass the image as a parameter to the mode's classify function
- ***then(predictions=>:*** then we'll get back a set of predictions
- ***console.log(predictions):*** write out the predictions to the console.

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

Then, you will see a result like this in the browser with DevTools running.



3.2.5 Results

If you zoom in the results, you can see the className, and the probability that the image matches that class.

```
▼ (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)
```

This HTML have a div called *output*, which used to create a reference to *outp*. Once getting the predictions, we can iterate though them use a for loop (range from zero to the *predictions.length*). Within the loop, we can add a break character, the prediction className, and the probability for that className to *outp.innerHTML*. When you run the code, you will see something like this.

```
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;
    }
  });
});
```

3.2.6 Example in Code

- line 3: load Tensorflow.js
- line 4: load MobileNet model
- line 7: specify an image
- line 8: design the font style
- line 11: create an *img* pointing at the *img* div
- line 12: create the *outp* pointing at this *output* div

- line 13: load the mobilenet
- line 14: use model to classify the passing image
- line 15: get predictions
- line 16: loop through the length of the predictions
- line 17: create an HTML in the output including the className, a colon, the probability

```
1  <html>
2  <head>
3  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"> </script>
4  <script src="https://cdn.jsdelivr.net/npm/@tensorflow-models/mobilenet@1.0.0"> </script>
5  </head>
6  <body>
7      <img id="img" src="cat.jpg"></img>
8      <div id="output" style="font-family:courier;font-size:24px;height=300px"></div>
9  </body>
10 <script>
11   const img = document.getElementById('img');
12   const outp = document.getElementById('output');
13   mobilenet.load().then(model => {
14     model.classify(img).then(predictions => {
15       console.log(predictions);
16       for(var i = 0; i<predictions.length; i++){
17         outp.innerHTML += "<br/>" + predictions[i].className + " : " + predictions[i].probability;
18       }
19     });
20   });
21 </script>
22 </html>
```

3.3 Converting Mobiles to JSON Format

3.3.1 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 **C1_W3_Lab_3A_linear_to_JavaScript.ipynb** Jupyter notebook found in the following folder in the GitHub repository:

tensorflow-2-public/C1_Browser-based-TF-JS/W3/ungraded_labs/

To run this notebook, you need to install Jupyter with Python 3, TensorFlow 2.0, Tensorflow.js, and NumPy.

After run the Jupyter Notebook, you will end up with a single JSON file named **model.json** and a **.bin** file named **group1-shard1of1.bin.**

Then, you can launch the **C1_W3_Lab_3B_linear.html** file in the Chrome browser and open the Developer Tools to see the output in the Console.

3.3.2 Models to JavaScript

You have learnt how to use the existing pre-trained models in JavaScript in the browser. Now you will learn to build a model in Python and convert it into JavaScript. You start with a basic neural network (y=2x-1).

Step 1. Install tensorflowjs in Python. (Push the restart button after the install in Colab)

```
!pip install tensorflowjs
```

Step 2. Create and train the network. (It's a single layer with a single unit, trained with stochastic gradient descent optimizer and a mean squared error loss. We feed it with six xs and six ys, and train it for 500 epochs.)

```
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)
```

Step 3. Predict a value.

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

Step 4. Save out the model by using a directory with timestamp (/tmp/saved_models/ followed by the timestamp). Note: TensorFlow 2.0 uses *tf.keras.experimental.export_saved_model*, and TensorFlow 1.x uses *tf.keras.saved_model.saved_keras_model*.

```
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)
```

When we run it, the output will show this directory name.

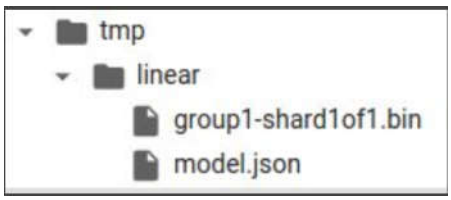
```
INFO:tensorflow:SavedModel written to: /tmp/saved_models/1554528640/1554528642/saved_model.pb
b'/tmp/saved_models/1554528640/1554528642'
```

Step 5. Convert the saved model.

- ***input_format***: use the keras_saved_model;
- ***tmp/saved_models/../..***: specify the directory containing the saved model (a timestamp-based directory).
- ***tmp/linear***: is the output directory where the JSON to be saved.

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

Here is the Saved model files (*model.json* and *group1-shared 1of1.bin*). You need to download these files and put them in the same directory as the HTML page that will host them.



Step 6. Execute in a script block.

- ***MODEL_URL***: load model using URL over HTTP. (in the same directory as the HTML).
- ***await if.loadLayeysModel:*** get the JSON and turn it into a model by passing the MODEL_URL.
- ***model.summary***: inspect the model.
- ***tf.tensor2d***: create input tensor to predict value 10. (a 2D-tensor with the first dimension being the value to classify and the second being the shape of that value, in this case 1*1).
- ***model.predict:*** get the result by passing the inputs.
- ***alert:*** alert the result.

```
<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>
```

3.3.3 Models to JavaScript in Code

This Colab will train the neural network with a linear equation and then convert it into a JavaScript model.

Step1. install the tensorflowjs tools.

```
!pip install tensorflowjs
```

Step2. train the model with 500 epochs.

```
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)
```

Step3. predict the value for 10.

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

Step4. save the model.

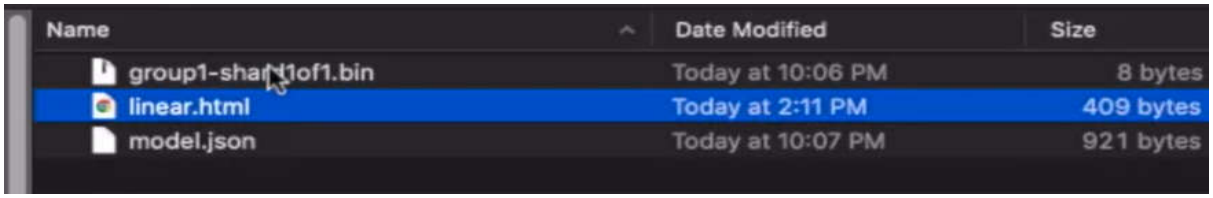
```
import  time
saved_model_path  =  "./{}.h5".format(int(time.time()))

model.save(saved_model_path)
```

Step6. convert the model using the saved model path.

```
!tensorflowjs_converter  --input_format=keras  {saved_model_path}  ./
```

3.3.4 Example in Code

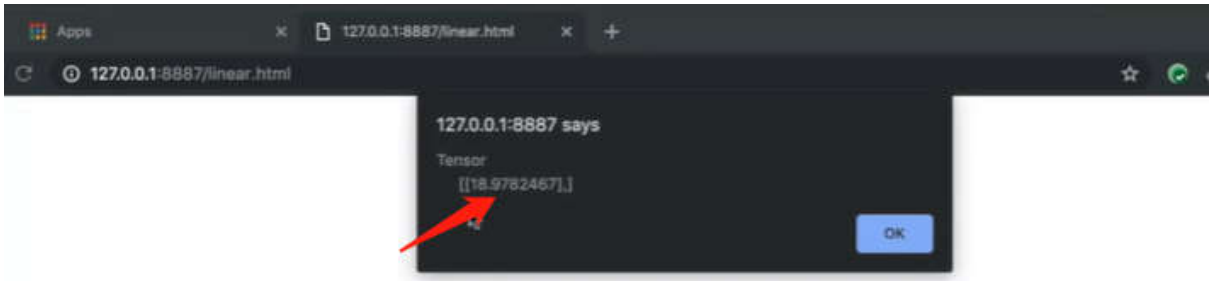
The folder will be serving the web page includes the linear.html, model.json and group1-shard1of1.bin.



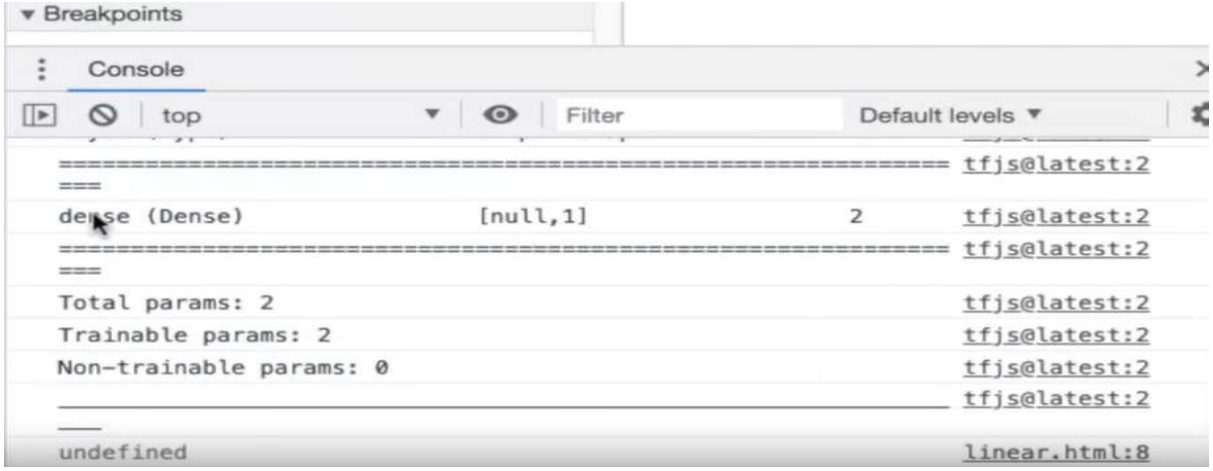
Here is the linear.html file. It has a script tag loading the latest tfjs and a script to run code. The run function is an asynchronous function, which can await TensorFlow to load model. Pass the model URL (model.json) to load the model using tf.loadLayersModel. Then print out model summary in the console log and predict the value for 10 to alert the result.

```
<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>

```

The web page will alter a value 18.9782.



We can also print the model summary of the model out the console using the developer tools.



3.3.5 Quiz: Week 3 Quiz

Question 1

When using the toxicity library, a statement will be labelled with 2 probabilities. What are they?

The first is the probability value for whether or not the phrase is not an insult, and the second is the probability for whether or not it is

The first is the probability value for whether or not the phrase is an insult, and the second is the probability for whether or not it is not

The first is the probability value for whether or not the phrase is not an insult, and the second is the threshold

The first is the probability value for whether or not the phrase is an insult, and the second is the threshold

Question 2

If toxicity returns a probabilities list with values of [0.8, 0.2], what does that mean?

The phrase does not contain an insult

The phrase contains an insult

There's an error

We don’t know. The answer depends on something else

Question 3

How do you determine what type of toxicity is contained in a result from toxicity?

When you call the API you specify what type of toxicity you are looking for with a parameter (i.e. ‘threat’)

There’s no way to determine type of toxicity, either a sentence is toxic or it isn’t

When you call the API you send it a list of specific toxicity types you want it to look for (i.e. ([‘threat’, ‘obscene’])

It returns an array of answers, each one corresponding to a different type of toxicity

Question 4

When using mobilenet in js to classify an image, it can recognize up to 1000 types. How many predictions does it return by default?

All non-zero predictions

3

1000

All that are above a threshold, set by the threshold parameter

Question 5

When converting Python-trained models to JSON to use in tensorflow.js, what is the package that you need to ‘pip install’ (assuming you already have installed tensorflow)

tensorflow-js

Tensorflowjs

None, it’s built into TensorFlow

tensorflow-javascript

Question 6

How do you convert a Python-trained model to JSON?

Save it as a TensorFlow Saved Model, then use the tensorflowjs_convertor script in Python

Save it as a TensorFlow Saved Model, then use the tensorflowjs_convertor script in JavaScript

Simply save it as JSON

Save it as a TensorFlow Saved Model, then import that as a JSON object

Question 7

If you have a model that you've converted to JSON how do you load it into JavaScript?

```
const model = await tf.loadSavedModel(MODEL_URL)
```

```
const model = tf.loadSavedModel(MODEL_URL)
```

```
const model = tf.loadLayersModel(MODEL_URL)
```

```
const model = await tf.loadLayersModel(MODEL_URL)
```

Question 8

When you convert a Python-based model to JSON, how many files will you get?

At least two: the model file, and a sharded collection of binary weight files that can have one or more files

One, the model file itself

Two, the model file and a metadata file

Two, the model file and a snapshot of binary weights

3.3.6 **Reading**: Week 3 Wrap up

To train models in JavaScript, you can use other people's models. The process of converting Machine Learned models from their binary format when created with TensorFlow to JSON, which can be loaded with TensorFlow.js and used for inference.

Next week we'll wrap the course by looking at one other important skill in Machine Learning -- and that's transfer learning, which you'll do by creating a complete site from scratch!

3.4 Lecture Notes (Optional)

3.4.1 **External Tool**: Lecture Notes W3

3.5 Graded Exercise – Converting a Python Model to JavaScript

3.5.1 **Programming Assignment**: Week 3-Converting a Python to JavaScript

Week 4: Transfer Learning with Pre-Trained Models

One final work type that you'll need when creating Machine Learned applications in the browser is to understand how transfer learning works. This week you'll build a complete web site that uses TensorFlow.js, capturing data from the web cam, and re-training mobilenet to recognize Rock, Paper and Scissors gestures.

Learning Objectives

- Train a model in your web browser by using images captured via a webcam
- Apply transfer learning to train a model to recognize hand gestures of rock, paper, and scissors
- Apply transfer learning to train a model to recognize hand gestures of rock, paper, scissors, lizard, and Spock.

4.1 Retraining the MobileNet Model

4.1.1 Introduction, A conversation with Andrew Ng

To train an image classification model using transfer learning in the web browser and interact with a webcam to classify the rock, paper, or scissors.

4.1.2 A Few Words from Laurance

This week you'll use transfer learning to build an image classify model. In this can, you will get an existing pre-trained model mobile net and freeze some of its layers. Training a new neural network using the features that you've extracted from the mobile net. You'll capture images in the browser using the webcam, sort these into your desired classes. And then with transfer learning, you'll build a new model that classifies only those images.

4.1.3 Building a Simple Web Page

First, we build a web page that contains a video div to render the webcam. This page will render a live stream of the webcam.

- ***..tfjs@latest***: load the latest tensorflow.js;
- ***webcam.js***: manage the webcam in the browser by capturing images and converting them to tensors;
- ***video autoplay:*** define a video area on the page given the id ***wc;***
- ***index.js:*** create and initialize the webcam.

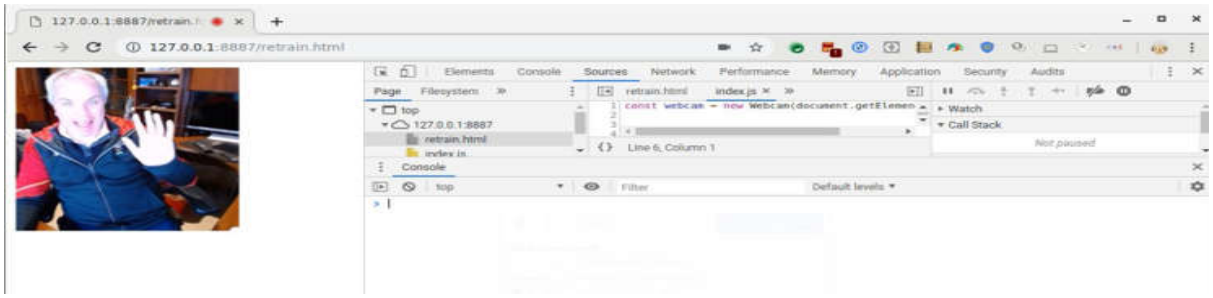
```
<html>
<head>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"> </script>
<script src="webcam.js"></script>
</head>
<body>
  <div>
    <div>
      <video autoplay playsinline muted id="wc" width="224" height="224"></video>
    </div>
  </div>
</body>

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

Here is the **index.js** file.

- ***let mobilenet/ let model***: Declare the mobilenet and model to share across other function in the script.
- ***new Webcam***: create a constant for webcam object, stored in webcam.js, initializing it by point it at the video element in the hosting page that we call **wc** for webcam,
- ***init():*** sets up the webcam by calling webcam.setup.

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

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

init();
```

When you run the page, you will see the webcam in the upper left-hand corner.



4.1.4 Retraining the MobileNet Model

Now, you need to get the mobilenet model.

- ***tf.loadLayersModel*** : Load the JSON model from its hosted URL
- ***mobilenet.getLayer:*** get one of the output layers from the preloaded mobilenet. The one above this layer will freeze.
- ***tf.model:*** make a new model. Its constructor will take the mobilenet inputs and *conv_pw_13_relu* as 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});
}
```

- ***loadMobilenet()*** : load model.
- ***mobilenet.predict(webcam.capture()):*** initialize the model. ***webcam.capture()*** get a tensor and ask mobilenet to predict it. ***tf.tidy*** throws away any unneeded tensors to clear the memory.

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

4.1.5 The Training Function

We create a new model with its input shape being the output shape of the desired mobilenet layer.

- ***tf.sequential*** : define a sequential model.

- ***tf.layers.flatten:*** the first layer is the flattened output from the mobilenet model by truncating the full mobilenet.
- ***tf.layers.dense:*** create a 100-unit hidden layer and then a 3-unit output layer. (transfer learning from the 1000 class mobilenet model to handle 3 specific classes)

```
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'})
    ]
  });
}
```

At prediction time, we'll then get a prediction from our truncated mobilenet up to the layer that we wanted to give us a set of embeddings. We'll then pass those embeddings through the new model to get a prediction that the new model was trained on.

- ***mobilenet.predict*** : get a set of embeddings by passing in the image.
- ***mobile.predict:*** take these embeddings and pass them to the new model to get a prediction photo.

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

4.2 Capturing the Data

4.2.1 Capturing the Data

To capture the samples, encode them for training and then train the new neural network with the transferred features from mobilenet. Here is the code for capturing data.

- ***button***: It creates three buttons, one for each type of sample to capture, which share the same ***handleButton*** as their onClick event handler. Then, another button to start training.
- ***div***: It has three output divs to render the number of samples that have been captured for each.

```
<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>
```

Here is how the ***handleButton*** click event works. This will capture a frame from the camera for training.

- ***switch(elem.id):*** switch on its ID. Call the function by passing the parameter *elem* (the reference to the button, each button had an ID: 0,1 or 2.).
- ***case:*** get the case through ID. For example, in case 0, it will find the div called “rock samples” and increase rack samples number, and then update the texts of rock samples and its number.
- ***parseInt(elem.id):*** extract the label from ID by converting it into an int (label is 0,1,2 not one-hot encoded).
- ***webcam.capture():*** capture the content from the webcam to extract the features.
- ***dataset.addExample(mobilenet.predict(img),label):*** add the prediction of that image from the mobilenet and the label to the dataset.

```
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);
}
```

4.2.2 The Dataset Class

In the html file, we create a script tag to load the dataset class called "*rps-dataset.js*".

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

In the index.js, we declare the dataset like this.

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

In the RPS dataset.js, we have RPSDataset class.

- ***constructor(){this.labels=[]}:*** initialize the class and set the labels array to empty.
- ***addExample(example,label):*** take an example and a label. (*example*: the output of the prediction for the image from the truncated mobilenet; *label*: 0,1,2);
- ***if(this.xs = null):*** For the first example, the *xs* is null,
    - ***this.xs=tf.keep(example):*** set the *xs* to the *tf.keep(example)*; *tf.keep* tells the *tf.tidy* to keep the tensor;
    - ***this.label.push(label):*** push the *label* to the label array.
- ***else{}:*** append the new example to the old for all subsequent samples.
    - ***const oldX = this.xs:*** create temp variable for the old set of xs (oldX);
    - ***tf.keep(oldX.concat(example,0)):*** concat the new example to the oldX;
    - ***this.label.push(label):*** push the *label* to the label array.
    - ***oldX.dispose***: dispose of the oldX.
- ***encodeLabels(numClasses):*** takes the array of labels and one-hot encoded it for training (keep list of labels, and only create the much large list of one-hot encoded ones for memory efficient.)

```
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) {
    ...
  }
}
```

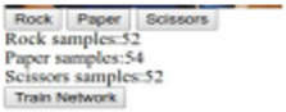
4.2.3 Training the Network with the Captured Data

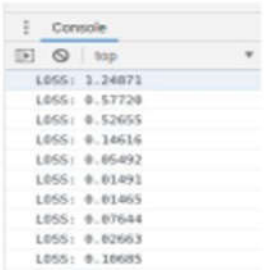
Here is the code for training the network:

- ***dataset.encodeLabels(3):*** one-hot encode the labels in the dataset and put the results into dataset.ys.
- ***tf.sequential{}:*** define the layers for the model.
- ***model.compile:*** compile model with the Adam optimizer and categorical cross entropy as its training on multiple categories.
- ***model.fit:*** train the model for ten epochs by giving the dataset xs and ys. On each batch end, we will console.log the 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);
        }
      }
    });
}
```

By this point, you can capture a few samples of each of rock, paper or scissors and train the neural network to recognize them.





4.3 Performing Inference from the Webcam Feed

4.3.1 Performing Inference

You've created a page that hosts TensorFlow.js, downloaded the Mobilenet model, set it up for transfer learning, and gathering data with which you can retrain the model. Now, you can poll frames from the webcam, and pass them to the model for inference to classify rock, paper, or scissors.

Here is the HTML page.

- ***button id='startPrediction':*** start the inference. You need to create a start predictions function.
- ***button id='stopPrediction':*** stop the inference. You need to create a stop predictions function.
- ***div id='prediction'***: these methods will output to the div with predictions named "prediction"

```
<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>
```

In index.js, you create the startPredicting() method.

- ***ispredicting = true:*** turns on a Boolean called *isPredicting* and lets us to continual predictions.
- ***predict():*** call the predict method.

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

In index.js, you create the stopPredicting() method.

- ***ispredicting = false:*** turns on a Boolean called *isPredicting* and stop predictions.
- ***predict():*** call the predict method.

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

Here is the skeleton of the predict method. It will do all the inference while isPredicting is set to true.

- ***step1 get prediction:*** get the prediction by reading a frame from the webcam and classifying it.

- ***step2 evaluate prediction and update UI:*** evaluate the prediction and update the UI with, "I see rock," "I see paper," "I see scissors," etc.
- ***step 3 cleanup:*** clean up so that it's a good browser citizen.

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

    // Step 2: Evaluate Prediction and Update UI

    // Step 3: Cleanup

  }
```

Here is the code to get prediction. It reads a frame from the webcam, uses mobilenet to get activation, and get prediction from the retrained model, then argmax this and return a 1D tensor containing the prediction.

- ***tf.tidy:*** tidy up to prevent memory leaks;
- ***webcam.capture:*** call webcam.capture and pass the results to IMG
- ***mobilenet.predict(img):*** pass the image to the truncated Mobilenet to get its' set of activations.
- ***mobile.predict(activation):*** pass these activations to the model which was trained on rock paper and scissors classes to get a prediction back.
- ***predictions.as1D().argMax():*** arg max and turn one-hot encoded results of the three classes into a value of zero, one, or two to return it as a classification.

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

After classification, we need to update the user-interface to reflect what we've seen.

- ***classId***: get a class ID from the webcam.
- ***predictionText***: fed class ID into a switch statement to generate text for, "I see rock," "I see paper," "I see scissors."
- ***getElementById***: the prediction div on the page is populated with that text.

```
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 = predictionT
```

Then we tidy up by disposing of the predicted class to trigger the tf.tidy that we mentioned earlier. We also call this tf.nextFrame, which is a TensorFlow function that prevents us from locking up the UI thread so that our page can stay responsive.

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

That's it. You've end to end encoding up an in-browser classifier that takes content from a webcam, retrains a Mobilenet model, and gives you custom classifications.

4.3.2 **Reading**: Rock Paper Scissors

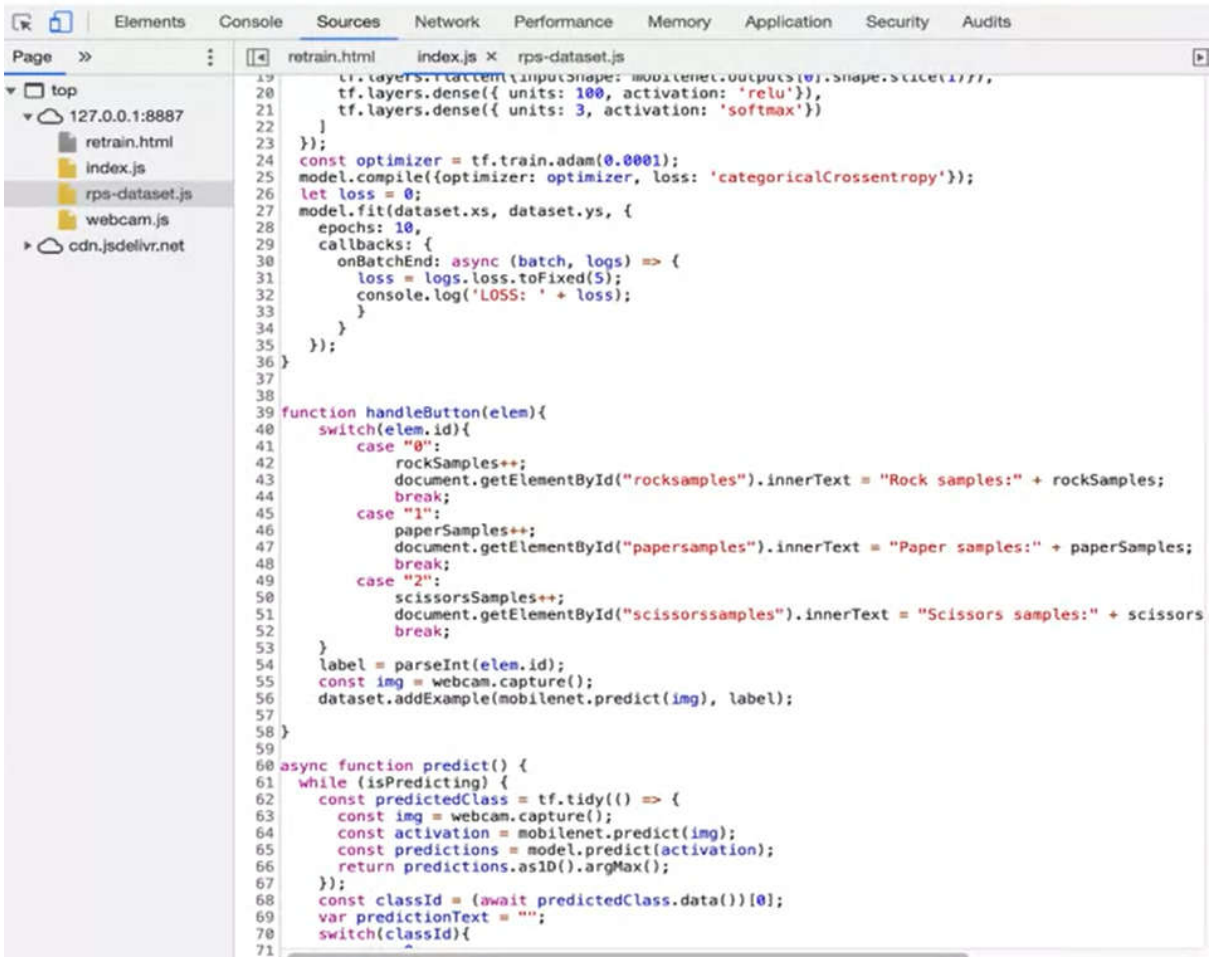
In the next example, we will use a pre-trained MobileNet model to classify hand gestures of Rock, Paper, and Scissors captured by a webcam.

You can use Brackets to open the **index.js** file. You can find the **index.js** file in the following folder in the GitHub repository for this course:

tensorflow-2-public/C1_Browser-based-TF-JS/W4/ungraded_lab/

When you launch the **C1_W4_Lab_1_retrain.html** file in the Chrome browser make sure to open the Developer Tools to see the output in the Console.

4.3.3 Rock Paper Scissors in Code



4.3.4 **Quiz**: Week 4 Quiz

Question 1

What HTML5 tag is used to show the contents of a webcam?

<pre>

<video>

<webcam>

Question 2

If I initialize a webcam object like this:

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

Which code will then start the webcam feed to render in the page?

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

```
async function init(){await webcam.start();}
```

```
async function init(){await webcam.initialize();}
```

```
async function init(){await webcam.go();}async function init(){ await webcam.go();}
```

Question 3

If I want to create a model that uses transfer learning, with everything in mobilenet up to layer 'foo', and my layers afterwards, how do I do it? Assume this code was used to find layer 'foo'

```
const layer = mobilenet.getLayer('foo');
```

```
return tf.model({inputs: mobilenet.inputs, outputs: layer.output});
```

```
return tf.model({inputs: mobilenet.inputs, outputs: layer.outputs});
```

```
return tf.model({inputs: mobilenet.input, outputs: layer.outputs});
```

```
return tf.model({inputs: mobilenet, outputs: layer});
```

Question 4

If I am transfer learning from a mobilenet, and I want to use my own dense layers after the mobilenet ones, what is the correct syntax to use at <INSERT CODE HERE>

```
model = tf.sequential({
  layers: [
   tf.layers.flatten(<INSERT CODE HERE>),
   tf.layers.dense({ units: 100, activation: 'relu'}),
   tf.layers.dense({ units: 3, activation: 'softmax'})
  ]
 });
```

```
{inputShape: mobilenet.outputs[0].shape.slice(1)}
```

```
{inputShape: mobilenet.outputs[0].slice(1)}
```

```
{inputShape: mobilenet.outputs[1].slice(0)}
```

```
{inputShape: mobilenet.outputs[1].shape.slice(0)}
```

Question 5

If I am using a mobilenet with my own DNN for transfer learning in TensorFLow.js, how do I get a prediction for an image?

Just pass the prediction to mobilenet, because you've already added your layers to it

Just pass the prediction to your own model, it already includes the mobilenet layers

Get a set of prediction embeddings from mobilenet and pass them to your model

Get a set of prediction embeddings from your model and pass them to mobilenet

Question 6

If you have a set of predictions returned from model.predict(something) and you want to take the one with the largest probability, how do you do it?

predictions[0] contains the one with the largest probability

predictions.as1D().argMax(), then look at the 0th element

predictions.sort() then look at the 0th element

predictions.argMax() then look at the 0th element

Question 7

If you already have a function called predict() in a class called 'foo' which captures a frame from the webcam and predicts it, what's the best way to call it, particularly if you plan to do continuous predictions?

```
foo.predict(); tf.tidy();
```

```
tf.tidy(foo.predict());
```

```
foo.predict(tf.tidy());
```

```
tf.tidy(() => foo.predict());
```

Question 8

Why is transfer learning a huge advantage, particularly when training in the browser?

It lets you skip training altogether

It allows you to use already-learned convolutions for distinguishing features, saving space

It allows you to use already-learned convolutions for distinguishing features, saving training time

It gives you a smaller model

4.4 Lecture Notes (Optional)

4.4.1 Ungraded External Tool: Lecture Notes W4

4.5 Graded Exercise – Rock Paper Scissors

4.5.1 **Reading**: Exercise Description

Rock, Paper, Scissors, Lizard, Spock.

In this week's exercise, you will train your model in the browser using your webcam. You will train your model in the same way as in the Rock, Paper, Scissors example but now you will include the Lizard and Spock hand gestures.

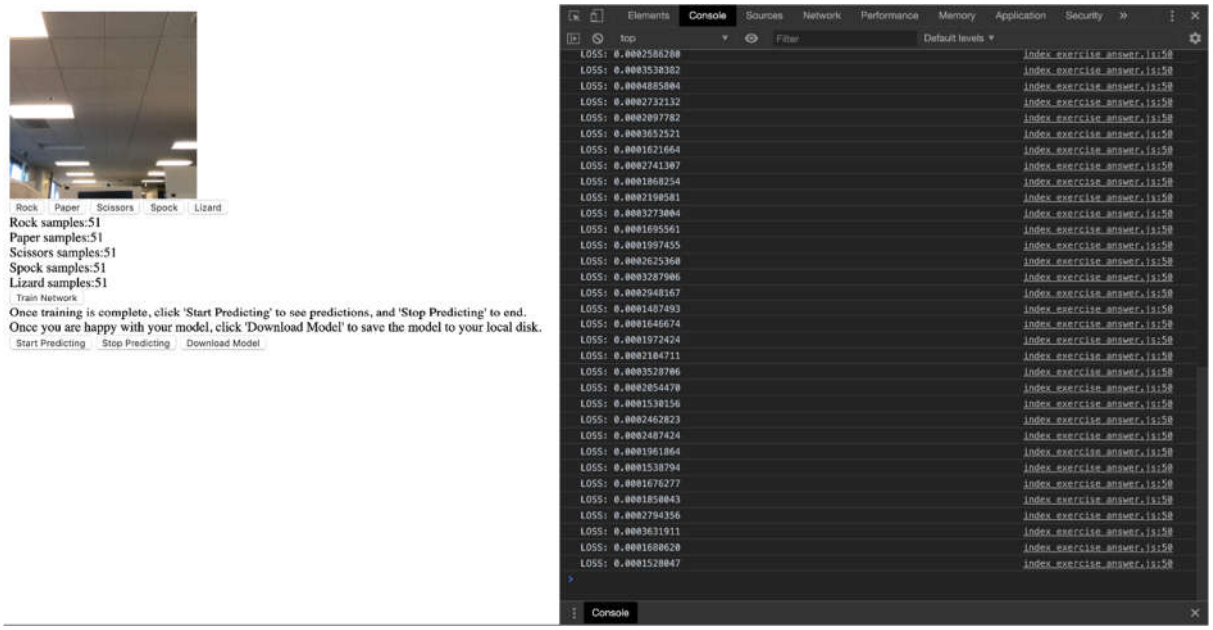
Below are a few tips that can help you get started.

1. To prevent overfitting do not add a lot a of dense layers. Using the same layers as in the Rock, Paper, Scissors example should be good enough.

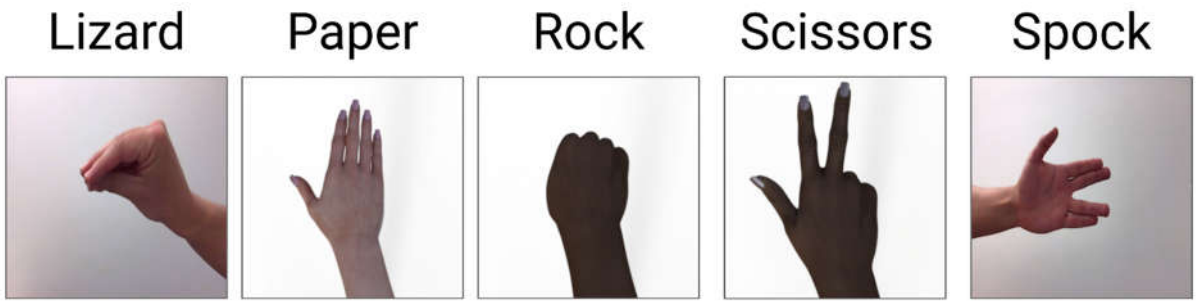
```
async function train() {
  dataset.ys = null;
  dataset.encodeLabels(5);

  // In the space below create a neural network that can classify hand gestures
  // corresponding to rock, paper, scissors, lizard, and spock. The first layer
  // of your network should be a flatten layer that takes as input the output
  // from the pre-trained MobileNet model. Since we have 5 classes, your output
  // layer should have 5 units and a softmax activation function. You are free
  // to use as many hidden layers and neurons as you like.
  // HINT: Take a look at the Rock-Paper-Scissors example. We also suggest
  // using ReLu activation functions where applicable.
  model = tf.sequential({
    layers: [

      // YOUR CODE HERE
      tf.layers.flatten({inputShape: mobilenet.outputs[0].shape.slice(1)}),
      tf.layers.dense({ units: 100, activation: 'relu'}),
      tf.layers.dense({ units: 5, activation: 'softmax'})
    ]
  });
```

2. Collecting between 50 to100 images for each hand gesture should be good enough for most models. However, feel free to experiment and try to train your model with less or more images.

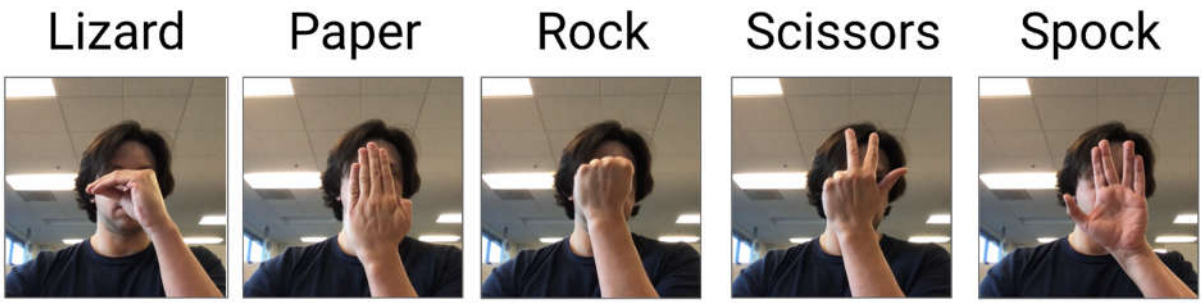
3. **Wait until Training has Finished before you Download the Model**. To make sure training has completed, open the Developer Tools and look at the Console output. When the browser alerts that "Training is Done!" click Ok. After you click Ok, you will see the value of the LOSS being printed in Console. Once the LOSS values stop being printed, you can go ahead and click the "Download Model" button to download the model and its weights.



4. Your model will be graded based on how it performs on our test set. Below are a couple of images from our testing set:



5. To train your model, don't stand too far away from the webcam and it's much better if you do not wear sleeves. Below are some sample training images.



Possible Issues

Because some of the weights of your model are initialized randomly, your model might get stuck in a local minimum during training. When this happens, your model may not perform optimally and may cause your submission to fail. If you are confident, you did everything correctly and your submission didn't pass, try re-training the model and re-submitting it.

4.5.2 **Programming Assignment**: Week 4- Rock Paper Scissors

Your exercise this week is to adapt the Rock, Paper, Scissors code to a much more interesting game of Rock Paper Scissors Lizard Spock from the TV Show “The Big Bang Theory” .

In this exercise, you will train your model in the browser using your webcam. You will train your model in the same way as in the Rock, Paper, Scissors example but now you will include the Lizard and Spock hand gestures. Once you are confident that you have a good model, you will be able to download it. **Note**: In this exercise, it can be a bit tricky to train a model well. So, don't get frustrated if you don't get it right the first time.

Use Brackets to open the **C1_W4_Assignment.js** file found in the following folder in the GitHub repository:

tensorflow-2-public/C1_Browser-based-TF-JS/W4/assignment/

Follow the instructions given in the code and fill-in the missing code in the parts labeled:

// Your Code Here

Once you have filled-in the missing code, run the **rpsls.html** file in the Chrome browser using the Web Server. Once you have trained your model, and ran the code successfully, the code will automatically download your model and its weights to your downloads folder. **It is important to note that your browser might prompt you to allow multiple files to be downloaded. If this happens, allow the browser to download multiple files.**

Once the files have been downloaded, you must upload these two files in a **single Zip file** to be graded. This single Zip file must only contain 2 files:

1. **my_model.json**: Contains the model architecture, i.e. the type and size of each layer.
2. **my_model.weights.bin**: Contains the weights of the model.

Good Luck!