# DEEP LEARNING WITH JAVASCRIPT

## A Hacker's Guide To Getting Started With Neural Networks

KEVIN SCOTT


2019 1st Edition

# Deep Learning With Javascript

## A Hacker's Guide to Getting Started With Neural Networks

Version 1.1

Kevin Scott

# Contents

Hi!

This sample contains two chapters from my book, Deep Learning With Javascript.

The first chapter talks about why you'd want to build a Neural Network in Javascript; the second, how to get started doing Image Classification.

I would love any feedback, good or bad (feedback@dljsbook.com). This field moves incredibly quickly, and I have and will continue to keep this book updated with the latest and greatest techniques.

— Kevin

# Foreword

```
function login(username, password) {
  const user = User.get(username)
  if (!user) {
    throw new Error('Bad login attempt')
  } else if (!user.checkPassword(password)) {
    throw new Error('Bad login attempt')
  } else if (user.expired) {
    throw new Error('Your subscription has expired')
  }
  return user
}
```

You've probably seen code like this before.

This is how most software gets written today. Manually and line by line. You, the programmer, specify everything about how a program should operate, from how a user interacts to the way data is stored and retrieved.

Some have called this approach to code explicit programming, though a more straight forward description would simply be "programming".

When you're building a login form, or any sort of system where you want **this** action to cause **that** outcome, explicitly programming software makes a lot of sense. You can reason through code like this. There's little danger that the software wakes up one day on the wrong side of the bed and decides it's tired of this whole "login business" and why don't we try logging users *out* for a change? Your code will always do exactly what you've told it to.

But let's say you want software that can detect whether a photograph contains a human face or not. How might you approach that?

```
function isThisAHumanFace(pixels) {
  if (
    hasEyes(pixels) &&
    hasEars(pixels) &&
    hasNose(pixels) &&
    hasMouth(pixels)
  ) {
    return true
  }
  return false
}
```

Perhaps you start by reasoning that a human head is made up of eyes, ears, a nose, and a mouth. What if the photograph is from above, and all we see is the top of the person's head? What about from below? What if they're upside down, or half their face is in shadow? What if they're wearing a floppy hat? What if they have a beard or have long hair over their eyes? Trying to define every possibility in a scenario like this becomes very complicated, very quickly.

Deep Learning offers a fundamentally different approach to building software. Instead of explicitly enumerating every possible feature the software may encounter, you provide the computer with examples, and let *the computer* figure out how to get to the right answer.

That's the promise of Deep Learning. Tackle problems that traditionally were hard-to-impossible to solve with regular programming. And in the process, revolutionize how we build software.

◒ ◒ ◒

I wanted a book like this one when I began studying Deep Learning. I'm a hacker at heart, and I learn by doing.

Most books on Deep Learning assume strong mathematics or statistics backgrounds. Traditionally this hasn't been a problem, because anyone looking to enter the field would be expected to tackle problems that required these sorts of backgrounds.

Today, facilitated by better hardware, better frameworks, and new discoveries in research, that's starting to change. We're seeing a whole new range of use cases emerge that are relevant to programmers, designers, writers, artists, and poets. Over the next few years I believe Neural Networks will be incorporated across all software domains, no matter the industry.

The goal of this book is to demystify Neural Networks and prepare you for that future, through hacking and through using them in projects you'd use in the real world.

To get the most out of this book, I assume you're familiar with modern Javascript, or at least confident enough to fake it. If you do need a primer on Javascript, there's a number of great recommendations in the Resources section.

# How This Book is Organized

At the beginning of most chapters is a URL that provides a sandbox for writing your code, with everything set up and ready to go, that looks like:

`https://dljsbook.com/i/inference`

You are of course welcome to run the examples locally if you'd prefer. If you do, install the following libraries via `npm`:

- `@dljsbook/data` - This is the main set of datasets you'll use throughout the book.
- `@dljsbook/ui` - This is the main set of UI components that will support your models.
- `@dljsbook/models` - This contains links to pretrained models.
- `@tensorflow/tfjs` - This is the main Deep Learning framework we'll be using throughout the book.
- `@tensorflow/tfjs-vis` - (Optional) This is a visualization helper for displaying charts, images, and tables in the browser. We'll be using this to log out various items during our coding.

The `Tensorflow.js` documentation is top-notch and I'll avoid regurgitating it unless it's particularly relevant to the topic at hand.

The field of Deep Learning changes incredibly fast. Things evolve, and quickly. If anything is missing, confusing, or just plain wrong, please let me know at feedback@dljsbook.com so I can fix it for the next version.

Happy hacking!

# 1. What is Deep Learning

When I was growing up, I read a story about a man from Serbia named Kalfa Manojlo. He was a blacksmith's apprentice who dreamt of becoming the first man in human history to fly.

He wasn't the first to try. People had been flapping their arms like birds for centuries with no apparent success, but Manojlo, with the confidence of somebody too young to know what they don't know, believed he could build a pair of wings better than any that had come before. On a chilly November day in 1841 Manojlo took his winged contraption, scaled the town Customs Office, and launched himself into the air above a crowd of bemused onlookers.

You've probably never heard of this guy unless you're Serbian, so you can probably guess what happened: Manojlo landed head first in a nearby snowbank to much amusement from the gathered crowd. (Pretty good entertainment in the 1840s.)

Many early attempts at flight were like this. People thought that to fly like a bird, you had to imitate a bird. It's not an unreasonable assumption; birds have been flying pretty darn well for a pretty long time. But to conquer flight on a human scale requires a fundamentally different approach.

Neural Networks, the engines that power Deep Learning, are inspired by the human brain, in particular its capacity to learn over time. Similar to biological brains, they are composed of cells connected together that change in response to exposure from stimulus. However, Neural Networks are really closer to statistics on steroids than they are to a human brain, and the strategies we'll use to build and train them are diverge pretty quickly from anything related to the animal kingdom.

Neural Networks have been around since at least the '50s, and from the beginning people have asked when we might expect machines to achieve consciousness. Depending on the speaker, the term Artificial Intelligence can mean anything from Logistic Regression to Skynet taking over the world. For this reason, we'll instead refer to the technology we're interested in as **Machine Learning** and **Deep Learning**.

Machine Learning is the act of making predictions. That's it. You put data in, you get data out. This includes a large range of methods not under the Deep Learning umbrella, including many traditional statistical methods. And Deep Learning covers the specific technology we'll study in this book, Neural Networks.

# The Proliferation of Deep Learning

Though the concepts behind Neural Networks have been around since at least the '50s, it's only within the past decade that it's exploded in industry, largely because of advances in hardware, in the volume of data, and in cutting-edge research advancements.

It turns out that asking a machine to make predictions, and giving it the tools to improve those predictions, is applicable to a startlingly diverse set of applications.

You've probably encountered Neural Networks in use through one of the popular virtual assistants, like Siri, Alexa, or Google Home, on the market today. When you use your face or fingerprint to unlock your phone, that's a Neural Network. There's a Neural Network running on your phone's keyboard, predicting which words you're likely to type next, or autosuggesting likely phrases in your email application. Perhaps you've been prompted to tag your friends in uploaded photos, with your friends' faces highlighted and their names autosuggested.

Deep Learning can recognize and classify images to a degree of accuracy that exceeds humans. Deep Learning monitors your inbox for spam and monitors your purchases for fraud. An autonomous agent uses Deep Learning to decide which move to make in a game of Go. An autonomous car uses Deep Learning to decide whether to speed up, slow down, and turn right or left. Hedge funds use Deep Learning to predict what stocks to buy, and stores use it to forecast demand. Magazines and newspapers use Deep Learning to automatically generate summaries of sports, and doctors are using Deep Learning to identify cancerous cells, perform surgery, and sequence genomes. Researchers recently trained a

Neural Network to [diagnose early-stage Alzheimer's disease long before doctors could](https://www.ucsf.edu/news/2018/12/412946/artificial-intelligence-can-detect-alzheimers-disease-brain-scans-six-years
).

There's almost no industry that won't realize huge changes from Deep Learning, and these changes are coming not in decades, but *today* and over the next few years.

Andrew Ng, co-founder of Google Brain, often refers to this technology as "the new electricity": a technology that will become so ubiquitous as to be embedded in every device, everywhere around us. This oncoming sea change has huge implications for how we build applications and craft software. Andrej Karpathy, Director of AI at Tesla, calls it "Software 2.0":

> It turns out that a large portion of real-world problems have the property that it is significantly easier to collect the data (or more generally, identify a desirable behavior) than to explicitly write the program. In these cases, the programmers will split into two teams. The 2.0 programmers manually curate, maintain, massage, clean and label datasets; each labeled example literally programs the final system because the dataset gets compiled into Software 2.0 code via the optimization. Meanwhile, the 1.0 programmers maintain the surrounding tools, analytics, visualizations, labeling interfaces, infrastructure, and the training code. — Andrej Karpathy

# Intelligent Devices

Traditionally, Neural Networks have been run exclusively on servers, massive computers with the computing horsepower to support them. That's beginning to change.

Companies are investing huge sums of money in improving consumer-grade hardware to run Neural Networks. Apple's recently released NPU chip - a dedicated neural processing unit embedded in every new iPhone - saw an astounding *9x* increase over the previous year's model. As Moore's Law slows down for traditional CPUs, manufacturers are finding that they can eke out big

gains by focusing on more specialized domains, like Neural Networks. This means that, increasingly, consumer-level hardware will feature powerful specialized hardware tuned to efficiently run Neural Networks.

And just in time, because there's compelling reasons to run Neural Networks directly on the device.

A big one is privacy. Consumers and governments are increasingly asking question about how data is being collected, used, and stored. If you run your Neural Network on-device, the data never needs to leave the device, and all processing can happen locally. While this is a compelling argument to users tired of hearing about yet another breach of their data in the news, it also opens up new use cases that demand increased respect for privacy, like smart devices expanding into the more intimate spaces of our homes.

Another reason is latency. If you're in a self-driving car, you can't rely on a cloud connection to detect whether pedestrians are in front of you. Even with a good connection, it's hard to do real time analysis on a 60 FPS video or audio stream if you're processing it on the server; that goes double for cutting-edge AR and VR applications. Processing directly on the device avoids the round trip.

Consumer devices have direct access to a wide array of sensors - proximity sensors, motion sensors, ambient light sensors, moisture sensors - that Neural Networks can hook into, and the devices in turn provide a rich surface for enabling direct interactivity with Neural Networks in ways that servers can't match.

> I believe we're currently at an inflection point, and mobile devices are soon going to dominate inference. The growth of the entire AI ecosystem is going to be fueled by mobile inference capabilities. Thanks to the scale of the mobile ecosystem, the world will become filled with amazing experiences enabled by mainstream AI technology ... Tasks that were once only practical in the cloud are now being pushed to the edge as mobile devices become more powerful. — Dan Abdinoor, CTO of Fritz.ai

As more companies face the decision of whether to deploy their Neural Network on a server or directly on the device, increasingly that answer will be the device. It just makes sense.

# Javascript and Neural Networks

While there's no shortage of domain-specific languages for on-device Neural Networks, I think Javascript is well positioned to garner market share for *on-device* Neural Networks going forward.

Javascript boasts a chameleon-like ability to exist on every platform, everywhere. It starts out with an enormous installed base of every browser on every phone and desktop, but it's also a popular environment for building native mobile application development (with React Native), native desktop development (Electron), or server-side development (Node.js).

In most AI frameworks (for instance, Tensorflow) the framework serves as a translation layer between high level abstractions and the actual mathematical operations farmed out to the GPU. Since the underlying math layer is C++, software developers have the freedom to build abstractions in whatever language is most comfortable.

Javascript is ideal for our purposes in this book, because you already have it installed (through your web browser) and it excels as a language in handling rich interactive experiences. Though all the Networks we'll write in this book are designed to run in a browser, you may wish to tackle larger datasets requiring more computation in the future; if so, all examples can be ported to run in Node.js and can take advantage of whatever server-side GPUs you have at your disposal.

> Any application that can be written in JavaScript, will eventually be written in JavaScript. — Jeff Atwood, co-founder of StackOverflow

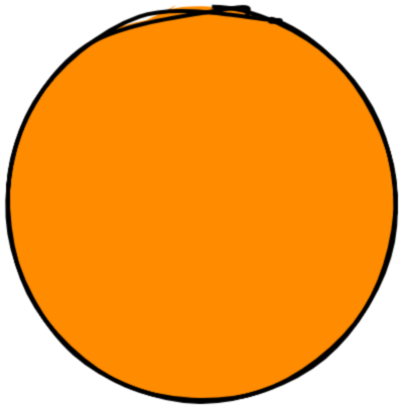
The biggest drawback I see for using Javascript for Deep Learning is the nascent ecosystem. `npm` still lags Python's tools in the breadth and depth of packages supporting Deep Learning, and a huge amount of resources, tutorials, and books demonstrate AI concepts in Python or R. To me, this presents an opportunity as a community to step up and contribute the next generation of tools. Javascript is a wily language and I have no doubt developers will fill in the gaps soon.

# Neural Networks

We've talked about why Neural Networks are important, but what exactly *is* a Neural Network?

Earlier we said Machine Learning is the act of making predictions. More precisely, you put numbers in, those numbers get put through some number of mathematical functions, and new numbers come out.

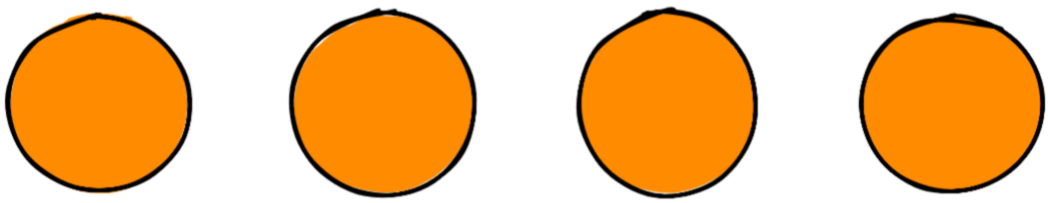
The fundamental building block of a Neural Network is a **Neuron**. In code, Neurons are commonly referred to as **units**.



**A lone Neuron**

The Neuron takes in a single number and transforms it. You'll specify the nature of this transformation when you architect your Network.

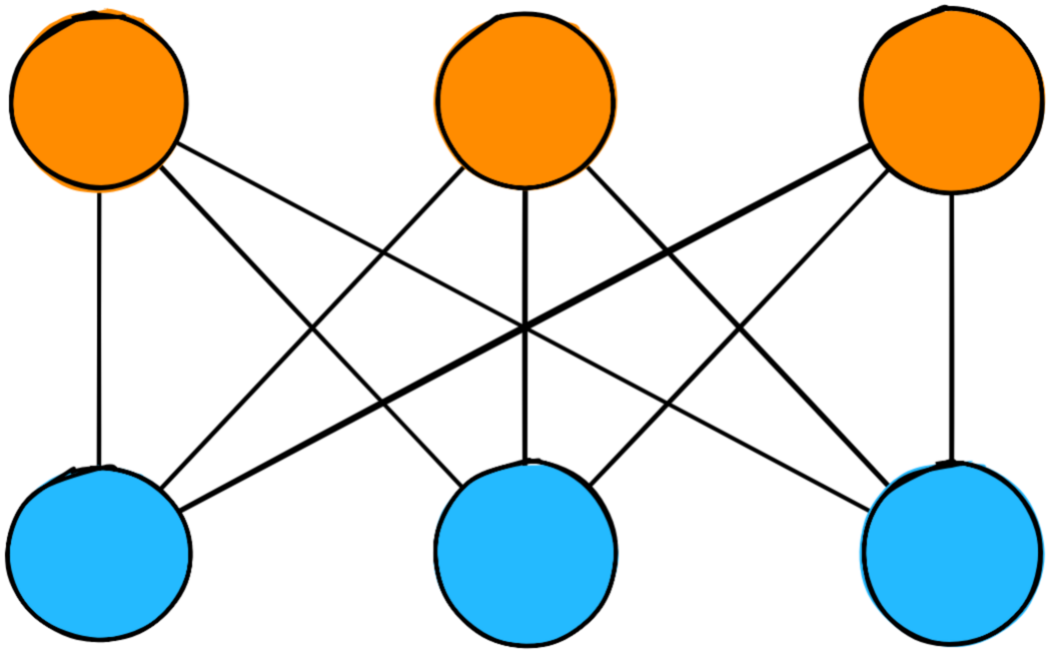
A collection of Neurons comprise a **Layer**.



A Layer of Neurons

Since a layer is a collection of neurons acting in concert, it is capable of much richer computation than a single Neuron.

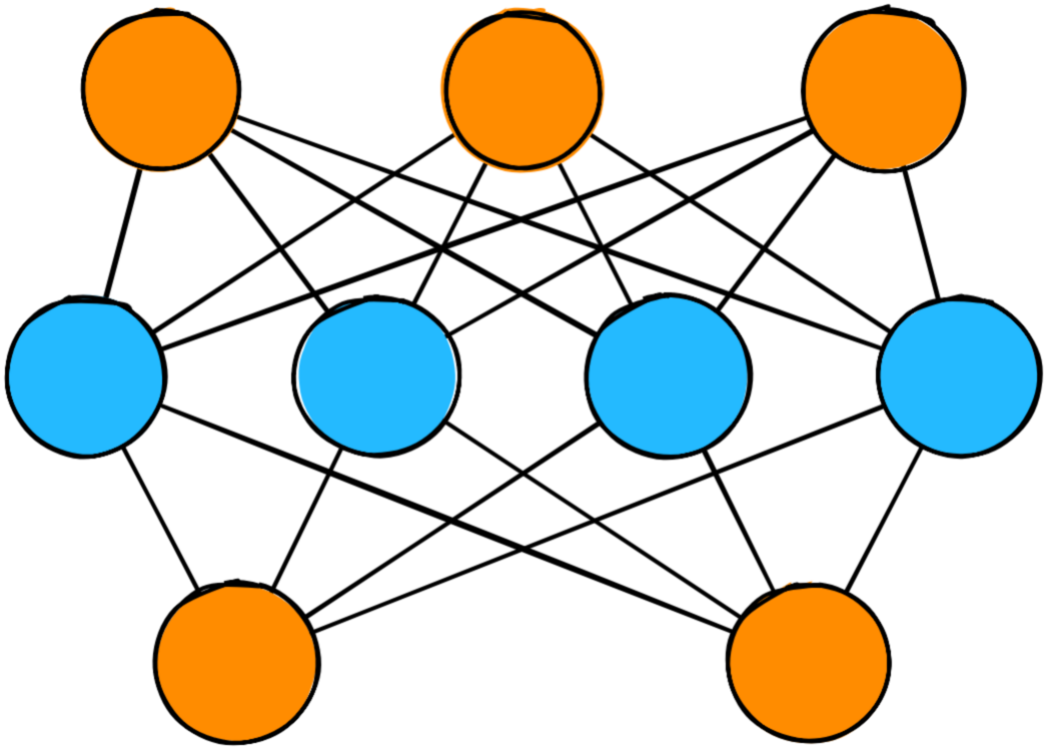
Neurons are connected to other Neurons via **Weights**.



Neurons connected by Weights

Weights describe the strength of the connection between a given pair of Neurons. A bigger weight implies a stronger connection between one Neuron and another, increasing the influence that Neuron will have on the final prediction.

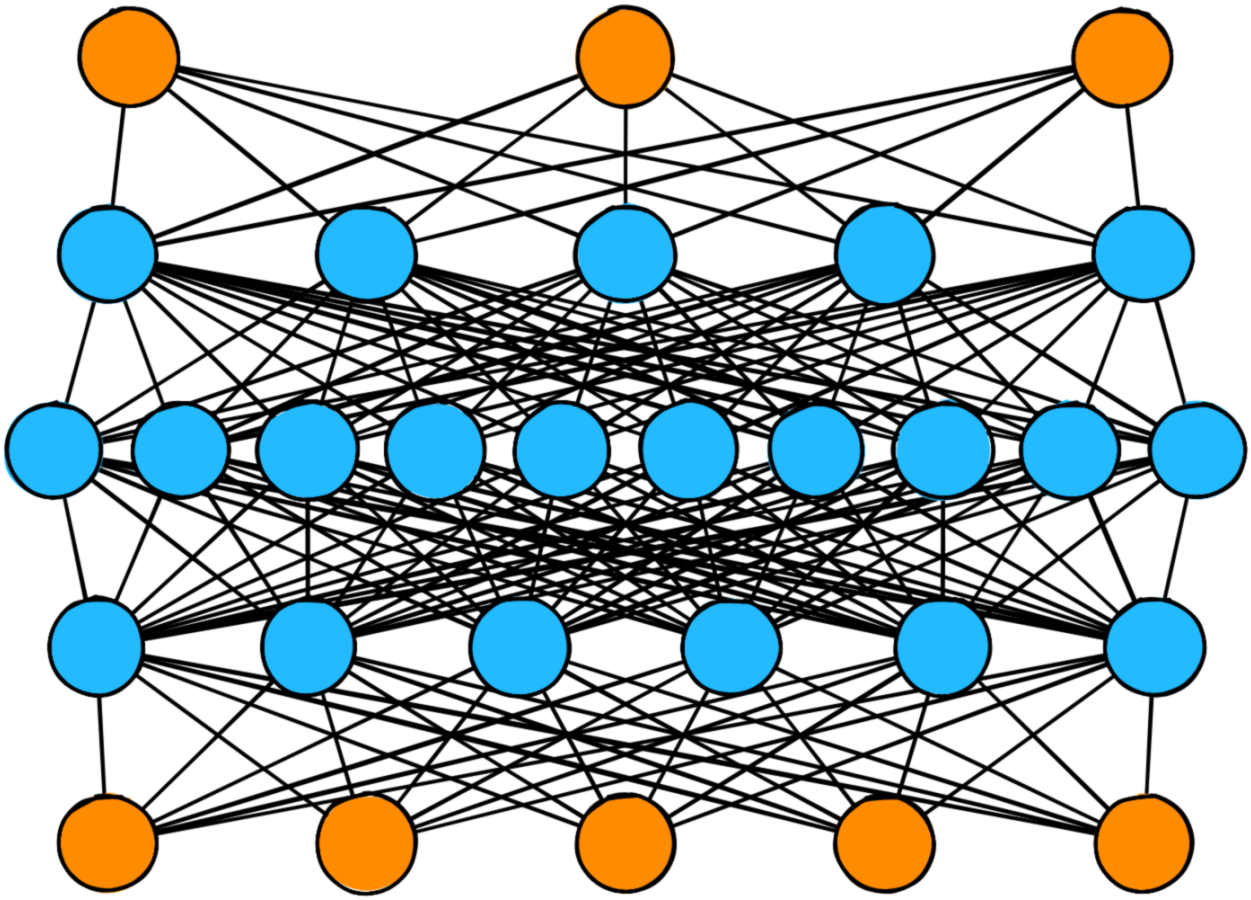
A Neural Network is made up of some number of layers.



**Layers in a Neural Network**

This diagram shows a three-layer Neural Network. The first layer is the **input layer**. This is where data enters the Network. The last layer is the **output layer**, responsible for emitting the transformed values from the Network.

The layer in the middle is called a **hidden layer**. Hidden layers make up the bulk of Neural Networks, and it's Networks with a lot of hidden layers that give rise to the phrase "Deep Learning": those Networks are "Deep". There's no limit to the number of hidden layers you can use.



**A complicated Neural Network**

A Neural Network exists in one of two phases: **Inference** or **Training**.

**Inference** describes the flow of data moving forward in one direction through the Network, from the input layer to the output layer. You can think of this as the network *predicting* the expected value, based on the given input values. For instance, you might feed it a picture of an animal, and the Network might answer "dog".

When you do this, specific Neurons fire based on the presence or absence of certain characteristics: Does it have fur? Does it have floppy ears? Is its tongue hanging out at an odd angle? Based on the answers to these questions, the Network might answer "Yes, I think this is a dog!"

**I believe this is probably a dog and I am 99.9% sure**

Inference is usually, though not always, how your users will interact with your Neural Network.

**Training** describes the flow of data forward *and* backward through the network. Based on the accuracy of the Network's predictions, changes ripple backwards, adjusting weights so that the network can improve and produce more accurate predictions in the future. You might feed the network a hundred photos of dogs, allowing the network to figure out - on its own - that all dogs tend to have fur and oddly angled tongues.

You may see the terms **forward propagation** and **backpropagation**. These are formal terms for describing Inference and an element of Training, respectively.

Often these two phases - Inference and Training - are approached separately, and in this book that's how we'll tackle them. We'll start by looking at **Inference**, including how to interact with pretrained Neural Networks, how to pass them data, and how to interpret predictions. After that, we'll discuss **Training**, including how to build Neural Networks from scratch, and how to train them to return accurate predictions.

We'll start by learning how to load a Neural Network in our browser and use it.

# Inference


**The Predictor, 1916, Giorgio de Chirico**

# 2. Making Predictions

```
>_ https://dljsbook.com/i/inference
```

> *It may be surprising to the academic community to know that only a tiny fraction of the code in many machine learning systems is actually doing "machine learning". When we recognize that a mature system might end up being (at most) 5% machine learning code and (at least) 95% glue code, reimplementation rather than reuse of a clumsy API looks like a much better strategy.*
>
> — **D. Sculley et all**

Have you ever built an app that relies on users to accurately tag the things they upload? If so, you've probably found you can't rely on users to tag things accurately, or tag things at all.

Part of the problem is it's a hassle. Remember Google+? One of its major innovations was prompting you to categorize your friends into "Circles" to give you a more relevant feed, but nobody had time for that.

To make it easier to collect accurate tags from our users, we can build a Neural Network that will automatically suggest tags when an image is uploaded.

For this example and subsequent chapters, we'll use a Neural Network called **MobileNet**. (Not all Neural Networks have names, but this one does). MobileNet is a Neural Network developed by Google. It's designed to be small and efficient, perfect for usage in a Browser.

All Neural Networks are trained on some collection of data, and if you have access to the original dataset, you can expect the Network to perform at or near its theoretical best. MobileNet was trained on a dataset called **ImageNet**. Its breadth and quality of classification (1000 categories to choose from), along with its size (an average of 500 images per category) have established it as a sort of "gold standard" for establishing Image Classification benchmarks.

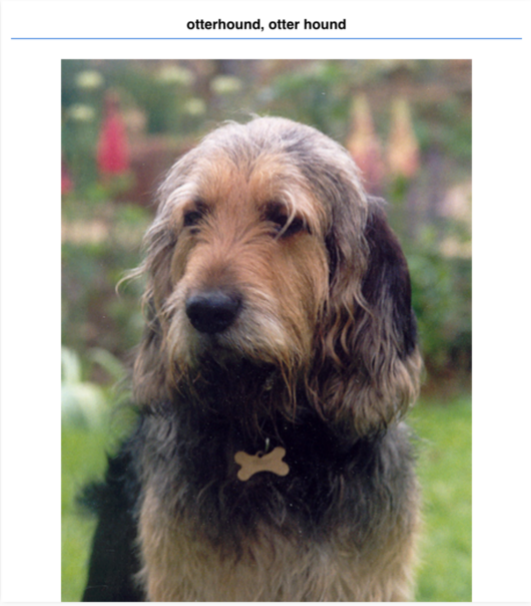

**Samples of ImageNet**

Let's jump into the dataset and see what things look like. The `@dljsbook/data` package provides a handy interface for loading images from ImageNet. You can load an image with:

```
import {ImageNet} from '@dljsbook/data'
const imageNet = new ImageNet()
imageNet.getImage().then(data => data.print())
```

This will print an image to tfvis if it's available, or your browser console.

Any call to `print` from `@dljsbook` accepts an optional HTML element as the first argument, like `print(document.getElementById('root'))`.




**Printing an image from ImageNet to your browser**

Now that you've seen the kind of data we're dealing with, let's see an example of **Inference** in action. We'll load MobileNet and feed it a random image, and see how accurate it is. The code to perform Inference with MobileNet is:

```
import * as tf from '@tensorflow/tfjs'
import {ImageNet} from '@dljsbook/data'
import {MobileNet} from '@dljsbook/models'
const imageNet = new ImageNet()
imageNet
  .ready()
  .then(() => {
    // load the model
    return tf.loadModel(MobileNet.url).then(neuralNetwork => {
      // get a random image
      return imageNet.getImage().then(data => {
        // make a prediction
        const prediction = neuralNetwork.predict(data.image)
        // print the image
        data.print()
        // print the prediction
        imageNet.translatePrediction(prediction)
      })
    })
  })
  .catch(err => console.error(err))
```

If we open up tfvis or your console log, we should see an image and its associated category. Run the code again to load a new, random image. We'll step through this code line by line.

The first line loads a Neural Network from the web into your browser with a single line of code:

```
tf.loadModel('path_to_a_network')
```

`model` is often used interchangeably to refer to a Neural Network.

`tf.loadModel` returns a promise that resolves with the Neural Network. Once the Network has loaded, we can load a random image from the ImageNet dataset:

```
const image = ImageNet.getRandomImage()
```

And send this random image through the Neural Network:

```
const prediction = neuralNetwork.predict(data.image)
imageNet.print(prediction)
```

This returns a category. Let's next look at the `prediction` variable. To translate `prediction` into a normal array, we can write:

```
const pred = prediction.dataSync()
```

This will return 1000 numbers, corresponding to a thousand categories.

If you write `console.log(prediction)` directly, you'll see some strange output. This is because `prediction` is a **Tensor**, not a number. To see the output of `prediction` directly, write `prediction.print()`. **Tensors** are a type of data container we'll cover in depth in Chapter 3.

We can find the category with the highest prediction with this snippet:

```
const pred = prediction
  .as1D()
  .argMax()
  .dataSync()[0]
```

This pulls the highest number, corresponding to the category the Network is most confident matches the image. We can map the value of prediction to a string that matches the category:

```
console.log(ImageNet.classes[pred])
// Dog
```

Why does the Network return a number instead of the name of the category directly?

During the previous chapter, we discussed how Inference was like the Network looking at a picture of a dog and asking: "Does it have floppy ears? A tongue? It's a dog!" In reality, it's more accurate to say that a Network receives data as a series of numbers - 11010 - each representing a pixel value, and the Network emits a number - 61, for example - that must in turn be interpreted as "dog" or not.

Neural Networks deal entirely in numbers, and all the data you send through and collect from the Network needs to be converted. While converting data that comes out of the network is usually pretty straightforward, preparing and converting the data *before* it goes into the model can be trickier.

Let's see how to do that.

◒ ◒ ◒

Thanks for reading!

If you're interesting in purchasing a copy, head on over to https://dljsbook.com.

Again, I'd love to hear any feedback you have to offer, good or bad. You can write to me at:

feedback@dljsbook.com

Hope to hear from you!

— Kevin