RESEARCH WORK
ON

TEACHABLE MACHINE & HOW TO BUILD IT WITH TensorFlow.js

Submitted in partial fulfillment of the requirement for the
Open Source Program

STUDENT CODE-IN

Submitted By
ASHWINI JHA

Under the supervision of
AKHILDAS KS (Project Administrator)

Contents

Chapter 1

1.1 What is a Teachable Machine ?

Teachable Machine is a web-based tool that makes creating machine learning models fast, easy, and accessible to everyone.

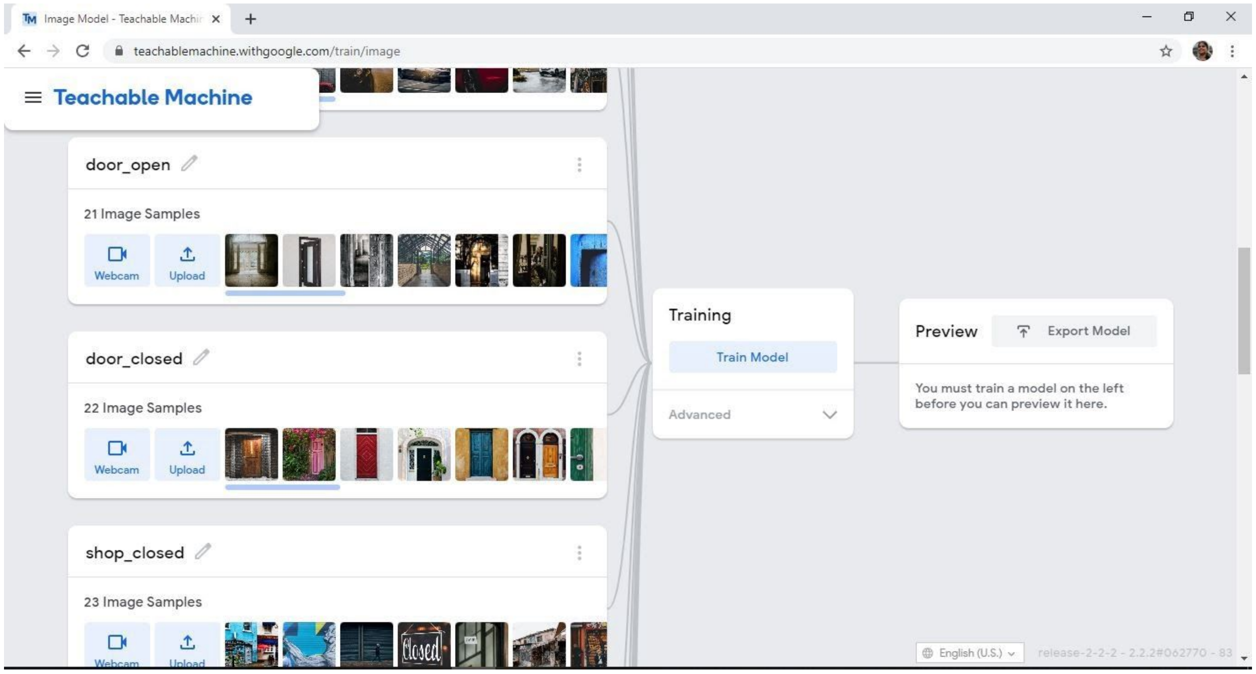
1.2 How does it work?

We train a computer to recognize your images, sounds, and poses without writing any machine learning code. Then, use your model in your own projects, sites, apps, and more.

1.3 How to use it?

- **Gather**

Gather and group your examples into classes, or categories, that you want the computer to learn.

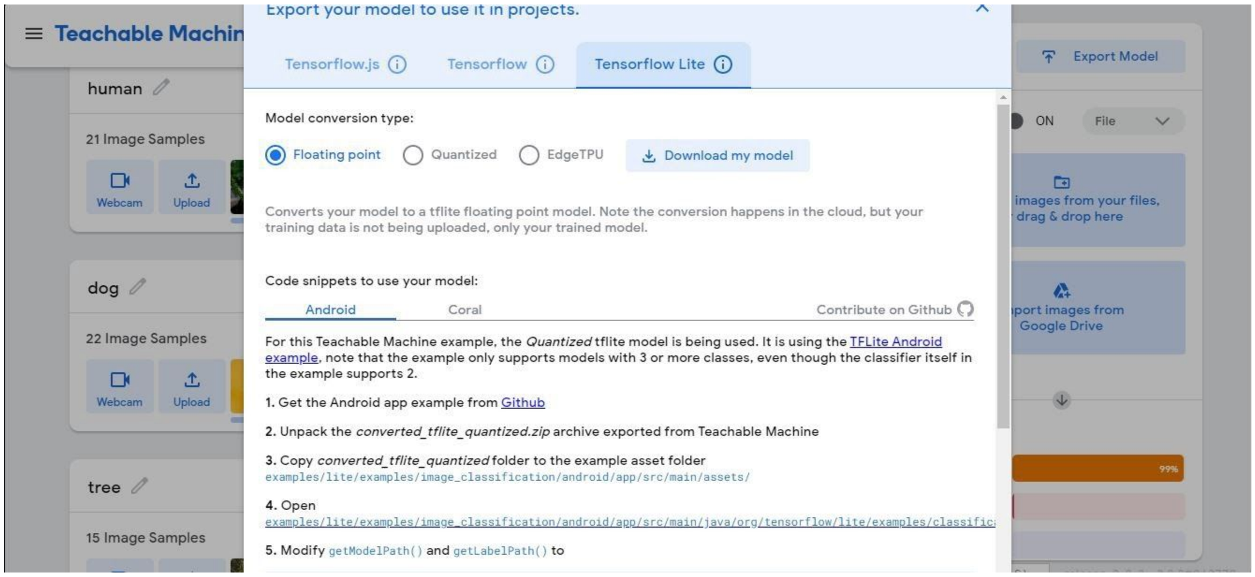


- **Train**

Train your model, then instantly test it out to see whether it can correctly classify new examples.

- **Export**

Export your model for your projects: sites, apps, and more. You can download your model or host it online for free.



Export your model to use it in projects.

Tensorflow.js | Tensorflow | Tensorflow Lite

Model conversion type:

[✓ Floating point] Quantized EdgeTPU Download my model

Converts your model to a tflite floating point model. Note the conversion happens in the cloud, but your training data is not being uploaded, only your trained model.

Code snippets to use your model:

Android | Coral | Contribute on Github

For this Teachable Machine example, the *Quantized* tflite model is being used. It is using the TFLite Android example, note that the example only supports models with 3 or more classes, even though the classifier itself in the example supports 2.

1. Get the Android app example from Github

2. Unpack the *converted_tflite_quantized.zip* archive exported from Teachable Machine

3. Copy *converted_tflite_quantized* folder to the example asset folder `examples/lite/examples/image_classification/android/app/src/main/assets/`

4. Open `examples/lite/examples/image_classification/android/app/src/main/java/org/tensorflow/lite/examples/classific`

5. Modify `getModelPath()` and `getLabelPath()` to

1.4 Works With...

- **TensorFlow.js**
- **ml5.js**
- **p5.js**
- **Coral**
- **Framer**
- **Node.js**
- **Glitch**

1.4.1 TensorFlow.js

TensorFlow.js is a JavaScript Library for training and deploying machine learning models in the browser and in Node.js. Comfortable with concepts like Tensors, Layers, Optimizers and Loss Functions (or willing to get comfortable with them)? TensorFlow.js provides flexible building blocks for neural network programming in JavaScript.Want to get started with Machine Learning but not worry about any low level details like Tensors or Optimizers? Built on top of TensorFlow.js, the ml5.js library provides access to machine learning algorithms and models in the browser with a concise, approachable API.

There are two main ways to get TensorFlow.js in your browser based projects:

- Using script tags.
- Installation from NPM and using a build tool like Parcel, WebPack, or Rollup.

1.4.1.1 How it works

- **Run existing models**
  - Use official TensorFlow.js models
  - Convert Python models
- **Retrain existing models**
  - Retrain pre-existing ML models using your own data (use transfer learning to customize models)
  - This is used for the App Image Classification for Visually Impaired (Transfer Learning)
- **Develop ML with JavaScript**
  - Build & train models directly in JavaScript using flexible & intuitive APIs.

1.4.2 ml5.js

ml5.js is machine learning *for the web* in your web browser. Through some clever and exciting advancements, the folks building TensorFlow.js figured out that it is possible to use the web browser's built in graphics processing unit (GPU) to do calculations that would otherwise run very slowly using a central processing unit (CPU). ml5 strives to make all these new developments in machine learning on the web more approachable for everyone. In this project I have used a pre-trained model called MobileNet -- a machine learning model trained to recognize the content of certain images -- in ml5.js.

The fastest way to get started exploring the creative possibilities of ml5.js are to either:

- Download a ml5.js project boilerplate. You can download a zip file here: ml5 project boilerplate..
  Or…
- ml5.js has been designed to play very nicely with p5. Open the p5 web editor sketch with ml5,js added.
- You can also copy and paste the cdn link to the ml5 library here

```
<script src="https://unpkg.com/ml5@0.4.3/dist/ml5.min.js"></script>
```

1.4.3 p5.js

p5.js is a JavaScript library for creative coding, with a focus on making coding accessible and inclusive for artists, designers, educators, beginners, and anyone else! p5.js is free and open-source and is accessible to everyone. Using the metaphor of a sketch, p5.js has a full set of drawing functionality. However, you're not limited to your drawing canvas. You can think of your whole browser page as your sketch, including HTML5 objects for text, input, video, webcam, and sound.

1.4.4 Coral

Coral is a complete toolkit to build products with local AI. One of the projects of coral is the Embedded Teachable MachineA machine that can quickly learn to recognize new objects by re-training a vision classification model directly on your device.

1.4.5 Framer

It's the prototyping tool that gives your work a competitive edge. Without sacrificing on speed or quality.

1.4.6 Node.js

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. As an Asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications. Many connections can be handled concurrently. Upon each connection, the callback is fired, but if there is no work to be done, Node.js will sleep.

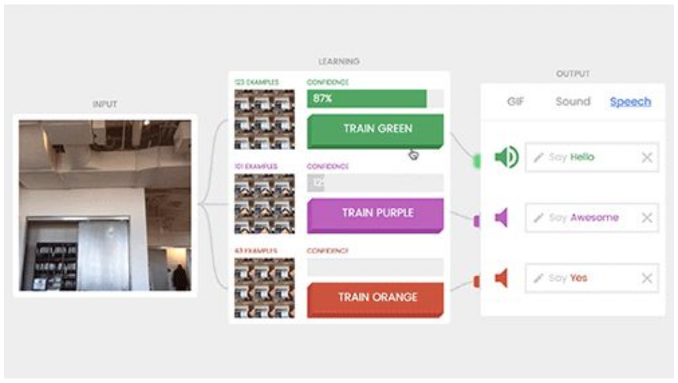
1.4.7 Glitch

Glitch is the friendly community where everyone codes together. Used to Build fast, full-stack web apps in your browser.

Chapter 2

2.1 How to build a Teachable Machine with TensorFlow.js

Teachable Machine lets anyone build their own image classification model with no coding required. All you need is a webcam.



A small library has also been released that implements the algorithm behind Teachable Machine, which doesn't require understanding the implementation details described in this work.

2.2 Approach: "transfer learning"

The approach we're going to take is called *transfer learning*. This technique starts with an already trained model and specializes it for the task at hand.

This lets you train far more quickly and with less data than if you were to train from scratch.

In this case, we'll bootstrap our model from a pre-trained model called MobileNet. Our system will learn to make predictions using our own classes that were never seen by MobileNet. We do this by using the activations produced by this pretrained model, which informally represent high-level semantic features of the image that the model has learned.

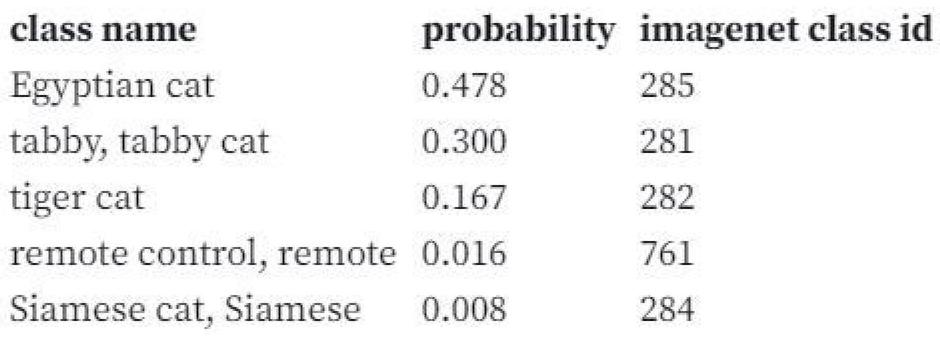
The pretraining is so effective that we don't have to do anything fancy like train another neural network, but instead we just use a nearest neighbors approach. What we do is feed an image through MobileNet and find other examples in the dataset that have similar activations to this image. In practice, this is noisy, so instead we choose the k nearest neighbors and choose the class with the most representation.

By bootstrapping our model with MobileNet and using k nearest neighbors, we can train a realistic classifier in a short amount of time, with very little data, all in the browser. Doing this fully end-to-end, from pixels to prediction, would require too much time and data for an interactive application.

2.3 MobileNet

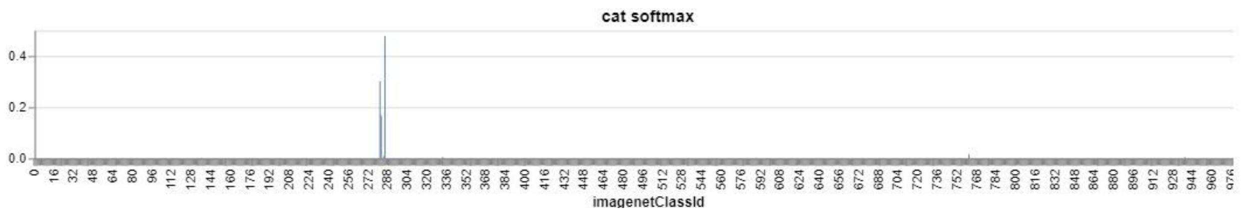
Let's first take a look at an off-the-shelf MobileNet, a model that is trained on ImageNet a dataset of millions of images with labels for 1000 different classes of objects, like dogs, cats, and fruits.

Let's see what a prediction through this image of a cute cat looks like:



class name	probability	imagenet class id
Egyptian cat	0.478	285
tabby, tabby cat	0.300	281
tiger cat	0.167	282
remote control, remote	0.016	761
Siamese cat, Siamese	0.008	284

We can visualize the full probability distribution over the 1000 classes by using a bar chart, where each number in the domain corresponds to a class from ImageNet (like "Egyptian cat" or "remote control, remote") and the height of the bar chart represents the probability of that class:



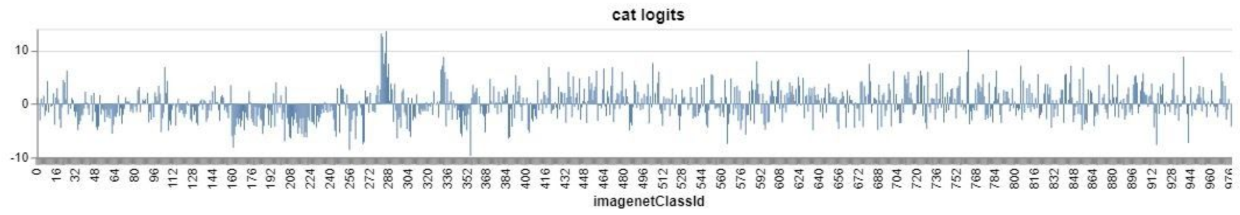
Neural network models, like MobileNet, typically consist of a stack of layers, which are mathematical transformations of tensors with parameters that are automatically tuned by the training process. We call the output of each of these layers "activations".

In this MobileNet model, the last layer is a softmax normalization function. Intuitively, this function "squashes" a vector of unnormalized predictions, generating a probability for each of the 1000 classes (normalized predictions).

The unnormalized predictions vector is usually called logits. This is the activation we'll use for transfer learning.

The logits are represented as a vector with 1000 elements. In TensorFlow.js, this is represented as a tensor with shape [1000], where each value contains a number representing the prediction for that class: [l1 l2 l3 … l1000]

For the cat image above, they look like this:



Notice that the values are not normalized, and the peaks around index 285 (Egyptian cat) lines up with the softmax probabilities in the chart above.

We can consider the logits produced by running our image through MobileNet as a "semantic fingerprint" of the image.

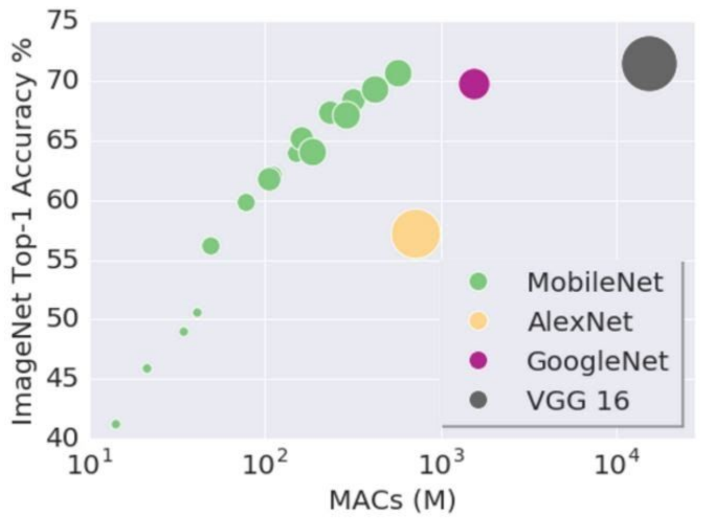
Just by crudely visualizing the logits with a bar chart, you can already see that semantically similar images have a similar logits activation.

- **MobileNetV1**
- **MobileNetV2**
- **MobileNetV3**

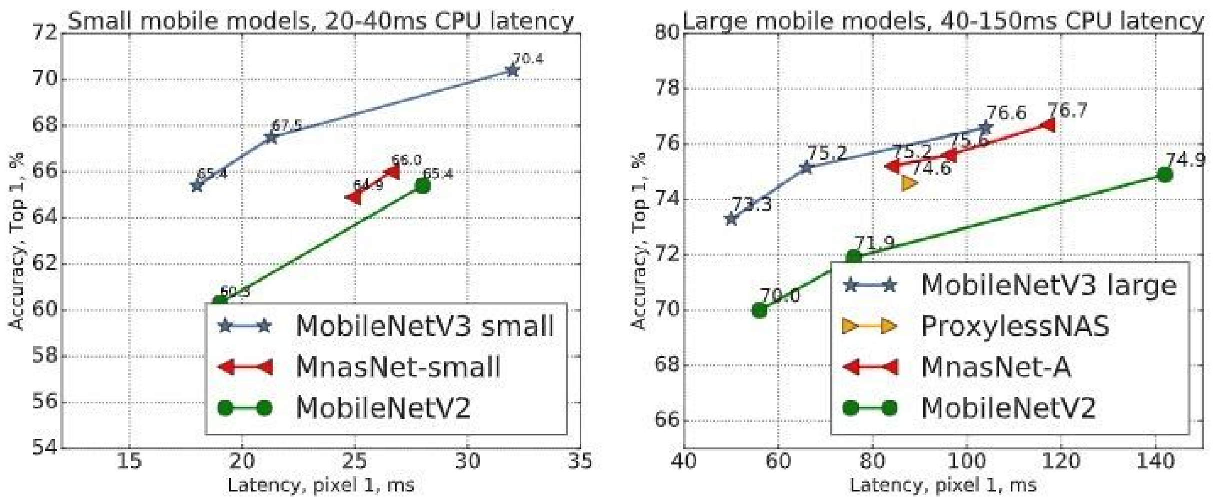
2.3.1 MobileNetV1

MobileNets are small, low-latency, low-power models parameterized to meet the resource constraints of a variety of use cases. They can be built upon for classification, detection, embeddings and segmentation similar to how other popular large scale models, such as Inception, are used. MobileNets can be run efficiently on mobile devices with TensorFlow Lite.

MobileNets trade off between latency, size and accuracy while comparing favorably with popular models from the literature.



2.3.2 MobileNetV2 & MobileNetV3



2.4 *K*-nearest neighbors

Let's say we want to distinguish between images of different kinds of objects we hold up to the webcam. Our process will be to collect a number of images for each class, and compare new images to this dataset and find the most similar class.

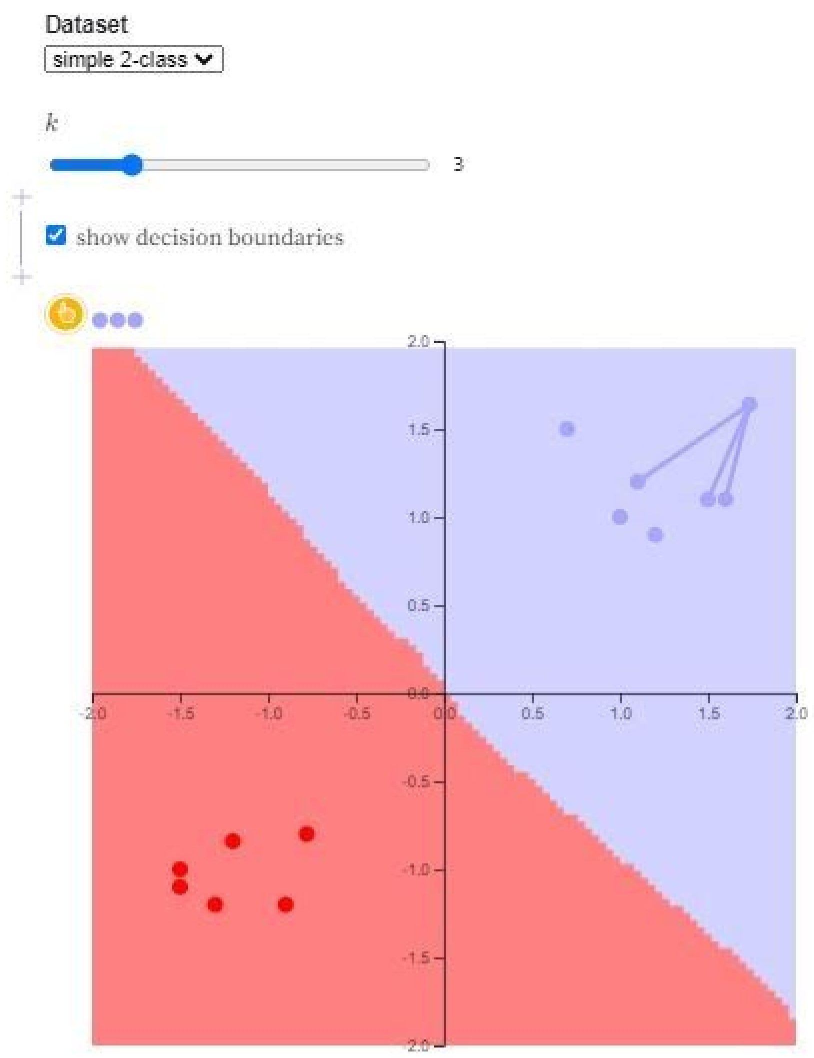
The particular algorithm we're going to take to find similar images from our collected dataset is called k-nearest neighbors. We'll use the semantic information represented in the logits from MobileNet to do our comparison.

In k-nearest neighbors, we look for the most similar k examples to the input we're making a prediction on, and choose the class with the highest representation in that set.

2.5 Let's illustrate this with a small example

Imagine we have a set of 2D points that are labeled "blue" or "red" and we want to determine whether a new point should be labeled "blue" or "red".

In the visualization below try hovering your mouse over the chart. The location of your pointer represents a point that we want to classify. For each of the existing data points, we compute the euclidean distance between the mouse and the point. We then sort the points by their distance to the mouse, and choose a class (color) that has the most representation in the top k sorted distances.


Dataset
simple 2-class

k 3

show decision boundaries

2.6 Cosine similarity

In the Teachable Machine application we use a slightly different way to measure the similarity of two points called *cosine similarity* instead of euclidean distance. Cosine similarity is convenient because we can use a dot product between the two vectors that we want to compute similarity for! Dot products are advantageous because they are much easier to accelerate than euclidean distance, especially in high dimensions (including in TensorFlow.js). Moreover, this metric has a natural interpretation in terms of the angle between two vectors (θ).

If you are interested in the linear algebra:
$\mathbf{v} \cdot \mathbf{w} = // \mathbf{v} // \ // \mathbf{w} // \cos\theta$
Notice that if $// \mathbf{v} // = // \mathbf{w} // = 1$ then $\mathbf{v} \cdot \mathbf{w} = \cos\theta$.
This means that if we normalize all of our logits vectors to unit length, then we can compute similarities by computing the dot product.

2.7 Let's make it!

Our custom Teachable Machine application will have two phases, data collection and prediction. In this example, we'll use 3 classes, "a", "b", and "c", but this algorithm will work for an arbitrary amount of classes.

2.7.1 Data collection

For each of the classes, we'll create a matrix of shape [N, 1000] where N is the number of samples collected for that class. When we collect a new image for a class, we'll feed it through MobileNet to get a logits activation, normalize it to unit length, and concatenate it to the end of our matrix, creating a new row.
To normalize the vector to unit length, we divide each component of the vector by the length of the vector: $u = v/||v||$
The following matrix represents the dataset for class A with Na examples collected.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,1000} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,1000} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{N_a,1} & a_{N_a,2} & a_{N_a,3} & \cdots & a_{N_a,1000} \end{bmatrix}$$

With our example logits in a matrix of this shape, we can use a matrix multiply to compute the dot product between **all** of the points in this class with a new input example.

We separate each class into their own matrix and stack them into a single matrix for convenience and efficient computation:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,1000} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,1000} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{1,1} & b_{1,2} & b_{1,3} & \cdots & b_{1,1000} \\ b_{2,1} & b_{2,2} & b_{2,3} & \cdots & b_{2,1000} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{1,1} & c_{1,2} & c_{1,3} & \cdots & c_{1,1000} \\ c_{2,1} & c_{2,2} & c_{2,3} & \cdots & c_{2,1000} \\ \vdots & \vdots & \vdots & \ddots & \vdots \end{bmatrix}$$

2.7.2 Prediction

When making a prediction, we feed our input through MobileNet to get the logits vector, then normalize it. This gives a vector, *x*, of shape [1000]. We'll show it as a column vector:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{1000} \end{bmatrix}$$

Now we can use a matrix multiply to compute the dot product between each row vector from our dataset with the input vector, *X*.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,1000} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,1000} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{1,1} & b_{1,2} & b_{1,3} & \cdots & b_{1,1000} \\ b_{2,1} & b_{2,2} & b_{2,3} & \cdots & b_{2,1000} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{1,1} & c_{1,2} & c_{1,3} & \cdots & c_{1,1000} \\ c_{2,1} & c_{2,2} & c_{2,3} & \cdots & c_{2,1000} \\ \vdots & \vdots & \vdots & \ddots & \vdots \end{bmatrix}$$

On the right, we've used the notation an, 1-1000 $\cdot x$ to mean the dot product of the nth row of the dataset and x.

Notice that each row of the result is the dot product of each row of the dataset matrix (a single dataset example) and the x we're making a prediction over. This means we get a cosine similarity between all dataset examples and our new vector!

Now, all we have to do is sort the resulting vector of similarities, truncate it to k top values, and select the class that appears most in the nearest neighbors.

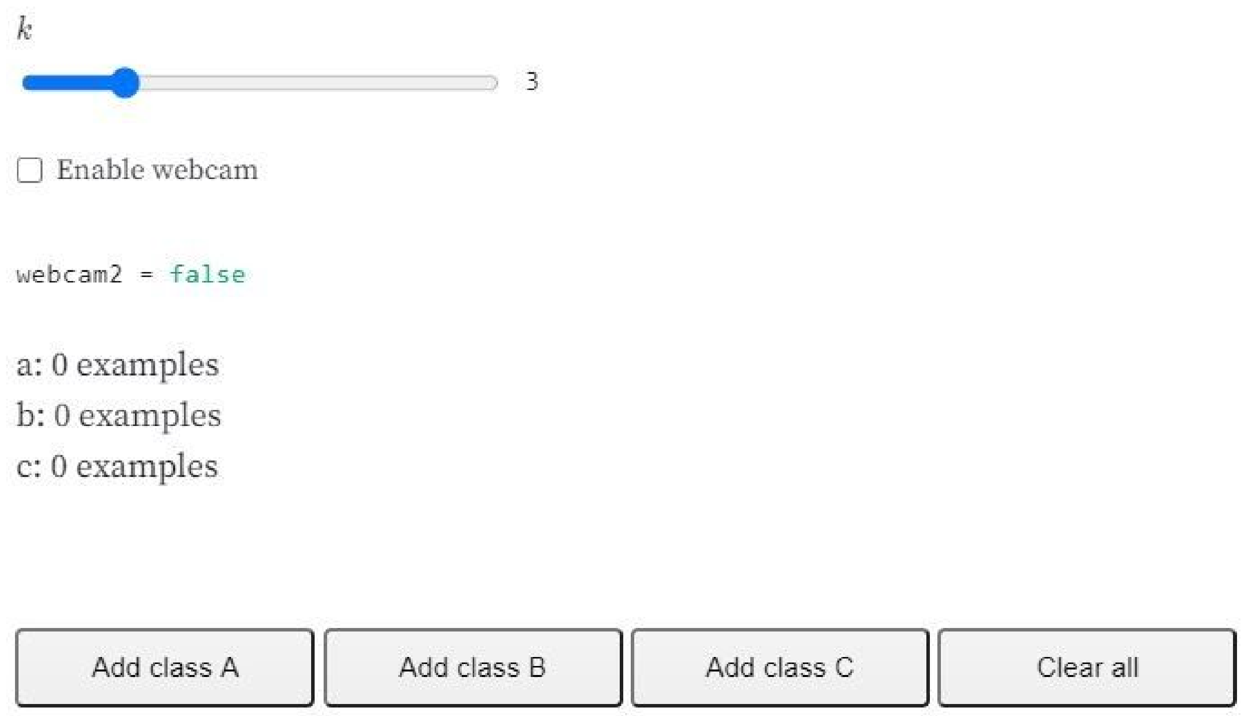
This process of computing all of these dot products at once is known as "vectorization", allowing us to take advantage of GPU parallelism.

2.7.3 Live webcam demo

Putting it all together, we have a full Teachable Machine application!

With your webcam enabled, use the buttons below to add examples for each class. Each time you click a button, it will add the current webcam image as an example for that class.

After you have added your first example, we'll start making predictions on the live webcam stream!



k 3

☐ Enable webcam

```
webcam2 = false
```

a: 0 examples
b: 0 examples
c: 0 examples

Chapter 3

3.1 Tweaking your model

3.1.1 Why isn’t it working like I want it to?

This tool can help anyone understand how machine learning works. Along the way, you might discover situations where your model isn’t working the way you want. Those are great opportunities to play around, learn, and try different approaches to improving your model. Here are some examples:

- **Changing backgrounds/environments.** Try training an image-based model to recognize a few objects. Then, see if it still works when those objects are against a different background, or a different lighting condition or time of day.

- **Framing your examples.** PoseNet (the technology Teachable Machine uses to track poses) doesn't only track how your pose appears, like if your arms are up or down. It also tracks where you appear in the frame. So if you're standing still on the left side of the frame, it appears to PoseNet that you're in a different pose than if you're standing still on the right side of the frame. To see how we dealt with this when training our head tilting demo, read our tutorial.

- **Changing microphones/spaces.** Try training a sound-based model, but then try testing it using a different microphone, changing your proximity to the mic, or change the room you're in and see if it still works.

- **Capturing audio samples.** Teachable Machine is built to recognize only 1-second samples, not longer ones. You can upload audio files created with the tool, but for now, you can't upload external .mp3s.

- **Understanding bias.** Bias is a critical concept to understand when creating machine learning models, and this tool can help give you a starting glimpse at what it's all about. First, watch this video to get a sense of how bias can affect machine learning models. Then, try training a model with some sounds using your voice, and testing it with someone whose voice is a bit different from yours. Does it still work as well? If not, what can you do to improve it?

- **Confusing examples.** It's sometimes fun to deliberately try confusing the computer and see what works. For example, train a model holding your right hand up, and see if it still recognizes that class if you hold your other hand up instead. Or, if you train it to recognize a certain object, what happens if you try tricking the computer with a photo or a drawing of that object? And that's just scratching the surface. Understanding how machine learning works is a really deep (and exciting!) field, and these are just starting points.

3.1.2 Can I use my model outside the Teachable Machine?

Yes. You can export your model as a TensorFlow.js model and host it on Teachable Machine for free, so you can call it into any website or app. You can also convert it to TensorFlow and TensorFlow Lite and download it for local use.

.