A proposed web standard to

Load and Run ML Models

on the Web

Jonathan Bingham / 2020-07-31

webmachinelearning.github.io/model-loader/#intro

TABLE OF CONTENTS

Model Loader API

Draft Community Group Report, 24 June 2020

This version:

https://webmachinelearning.github.io/model-loader/

Editor:

Jonathan Bingham (Google Inc.)

Explainer:

explainer.md

Abstract

This document describes an API to load a custom pre-trained machine learning model.

Status of this document

This specification was published by the Machine Learning for the Web Community Group. It is not a W3C Standard nor is it on the W3C Standards Track. Please note that under the W3C Community Contributor License Agreement (CLA) there is a limited opt-out and other conditions apply. Learn more about W3C Community and Business Groups.

Draft Spec for the Model Loader API

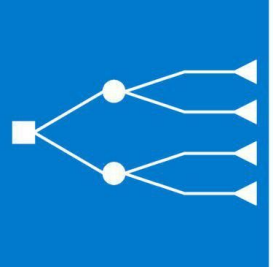

```
const modelUrl = 'url/to/ml/model';
var exampleList = [{
  'Feature1': value1,
  'Feature2': value2
}];
var options = {
  maxResults = 5
};

const modelLoader = navigator.ml.createModelLoader();
const model = await modelLoader.load(modelUrl)
const compiledModel = await model.compile()
compiledModel.predict(exampleList, options)
  .then(inferences => inferences.forEach(result => console.log(result)))
  .catch(e => {
    console.error("Inference failed: " + e);
  });
```

Why use machine learning in a web page as opposed to on the server?

According to the **TensorFlow.js** team:

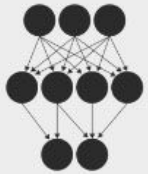
ML in the browser / client side means:

lower latency
high privacy, and
lower serving cost



machinelearn.js

for the web and node



ConvNetJS

stdlib

a standard library for javascript and node.js

MIL WebDNN

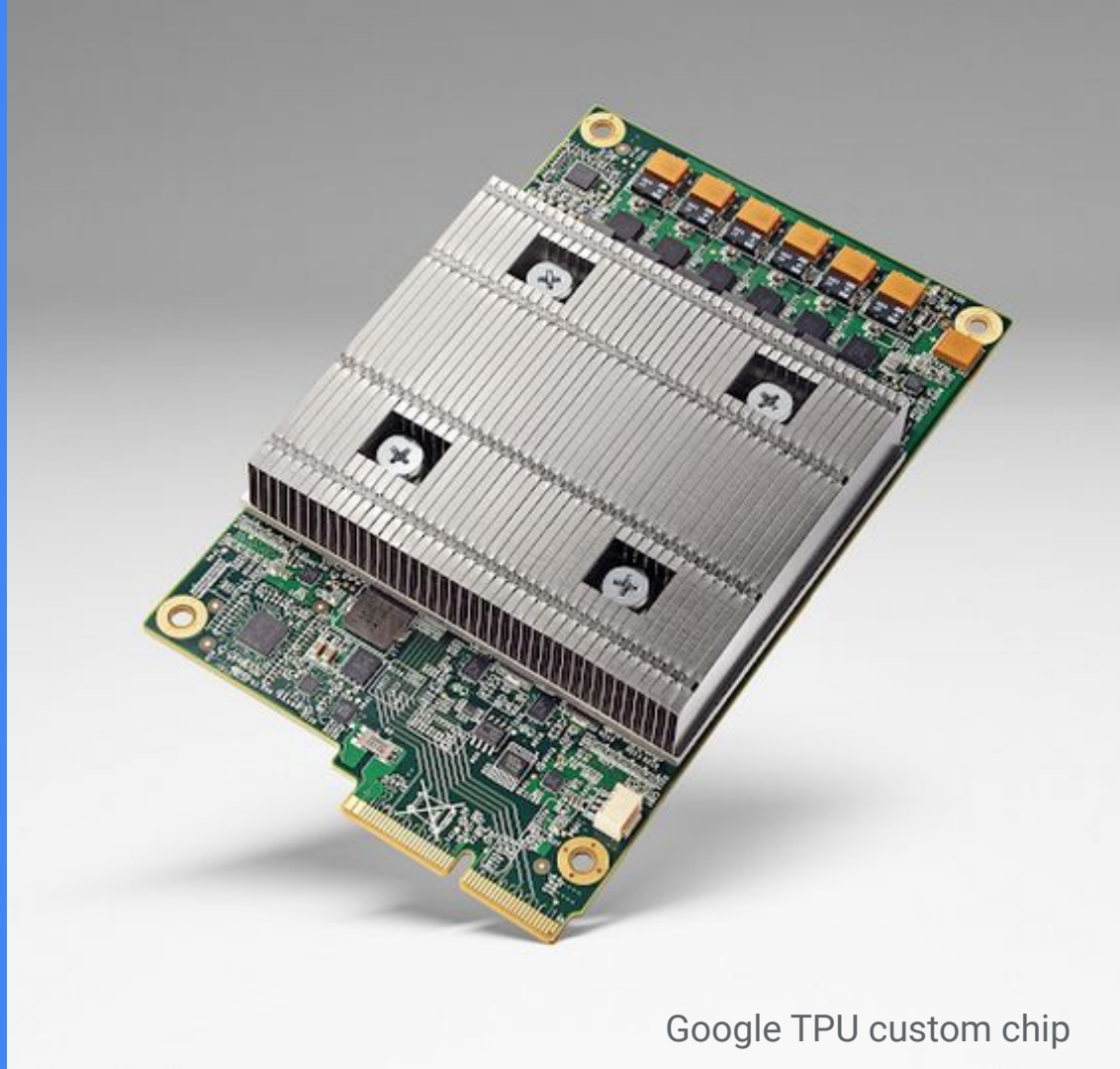
Why create a new web standard?

Don’t these awesome JavaScript libraries already address the need?

Speed matters for ML

New hardware enables new applications

TPU >> GPU >> CPU



Google TPU custom chip

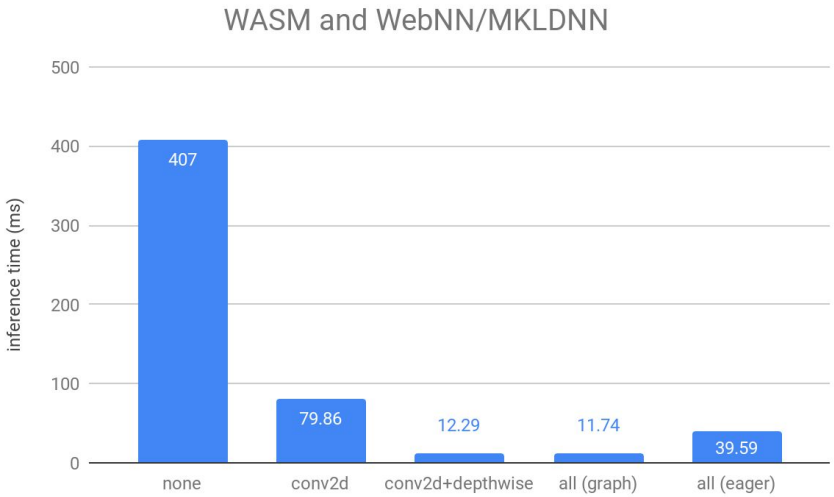
The web provides APIs for acceleration today. They help!

Sample TensorFlow.js performance data:

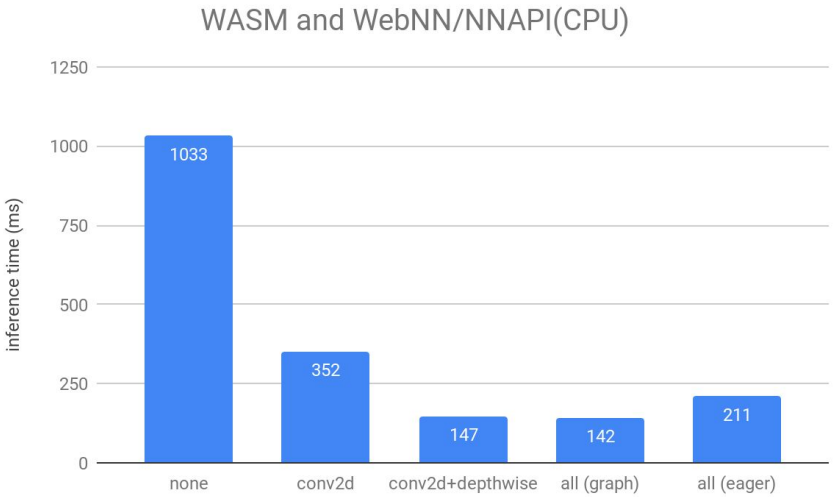
	WebGL	WASM	WASM+SIMD	Plain JS
iPhone XS	18.1	140		426.4
Pixel 3	77.3	266.2		2345.2
Desktop Linux	17.1	91.5	61.9	1049
Desktop Windows	41.6	123.1	37.2	1117
MacBook Pro 2018	19.6	98.4	30.2	893.5

Inference times for **MobileNet** in ms.

Running on native hardware can be even faster



Device configuration: XPS 13 Laptop, CPU: Intel i5-8250U, Ubuntu Linux 16.04, Chromium 70.0.3503



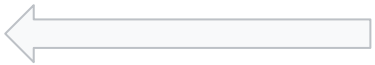
Device: Pixel 3, Android 9, updated on 12/2018, Chromium 70.0.3503

- Offload heavy ops gets significant speedup
  - Conv2D (90% computation): 5X faster on PC, 3X faster on smartphone
  - Conv2D+DepthwiseConv2D (99% computation): 33X faster on PC, 7X faster on smartphone
- Create bigger graph by connecting more ops gets better performance:
  - Per op graph vs. one graph: 3.5X slower on PC, 1.5X slower on smartphone

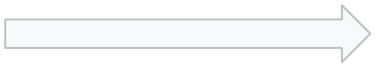
Source: Ningxin Hu
March 14 2019

How could the web platform accelerate ML?

1. **Operations** APIs for the most compute-intensive, like Conv2D and MatMul
2. **Graph** API similar to the Android Neural Networks API
3. **Model loader** API to load a model by URL and execute it with a native engine like CoreML, TFlite, WinML, etc
4. **Application-specific** APIs, like Shape Detection for barcodes, faces, QR codes



Lower level



Higher level

Source: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis non erat sem

Operation-level APIs for ML

Approach	Benefits	Challenges
1. Operations	✓ Small, simple API surface ✓ A few operations could provide a large performance gain	x No fusing, graph optimizations, or shared memory x Too low-level for web developers to use directly

Graph APIs open up more performance gains

Approach	Benefits	Challenges
2. Graph	✓ Allows fusing, graph optimizations, and shared memory	x 100+ operations to standardize x Large attack surface to secure x ML frameworks need to be able to convert to the standard x Large JavaScript API surface for browsers to implement x Too low-level for web developers to use directly

TABLE OF CONTENTS

Web Neural Network API

Draft Community Group Report, 1 July 2020

This version:
https://webmachinelearning.github.io/webnn/

Issue Tracking:
GitHub

Editors:
Ningxin Hu (Intel Corporation)
Chai Chaoweeraprasit (Microsoft Corporation)

Explainer:
explainer.md

Abstract

This document describes a dedicated low-level API for neural network inference hardware acceleration.

Status of this document

This specification was published by the Machine Learning for the Web Community Group. It is not a W3C Standard nor is it on the W3C Standards Track. Please note that under the W3C Community Contributor License Agreement (CLA) there is a limited opt-out and other conditions apply. Learn more about W3C Community and Business Groups.

Application-specific ML APIs are easiest for developers

Approach	Benefits	Challenges
4. Application-specific	✓ Small, simple API surface ✓ Easy for developers to use directly	x Customized models are impossible/hard x Long delay before models are added to the web platform

Examples of Shape Detection APIs:

- Barcodes
- QR codes
- Faces
- Text in an image
- Features of an image

```
const faceDetector = new FaceDetector(
   maxDetectedFaces: 5,
   fastMode: false
});
try {
   const faces = await faceDetector.detect(image);
   faces.forEach(face => drawMustache(face));
} catch (e) {
   console.error('Face detection failed:', e);
}
```

The model loader API balances flexibility and performance

Approach	Benefits	Challenges
3. Model loader	✓ Small, simple JavaScript API surface ✓ Easy for developers to use directly ✓ Allows fusing, graph optimizations, shared memory ✓ Existing ML model formats provide several full specs ✓ Unblocks experimentation and ML evolution	x 100+ operations to parse and validate x Large attack surface to secure x CoreML, PyTorch, TFlite, WinML are only partly convertible. x What format(s) to support? There are many...

Summarizing the options for ML APIs on the web

- Building ML-specific APIs into the web can increase **performance**
- There are multiple approaches, with tradeoffs
- The Model Loader API is complementary to graph, operations, and application-specific APIs
- We don’t know yet which level(s) of API we should propose to a working group.
  - Let’s get feedback from developers

Caveat: it's early days and there are big challenges

- ML is evolving rapidly.
  - New computational operations are being invented and published regularly.
  - Eg, TensorFlow has seen around 20% growth in operations every year.
- Hardware is evolving too, with tensor processing units and more
- Backward compatibility guarantees are essential for web standards, and not yet common for ML libraries.
- ML frameworks each have their own operation sets, and overlap between them is only partial, and conversion is not always possible.
  - The ONNX project (onnx.ai) is trying to define a common subset.

The current plan

- ✓ Incubate in the Web ML Community Group
- ➢ Now: Chrome and Chrome OS are working on an experimental build with TFlite integration
  - Coordinate with WebNN API efforts
- Next: shim the Model Loader API on top
  - Goal: alternate model formats and execution engines are possible
  - Run benchmarking to measure the performance gains
- Make a custom build available to developers
- Gather feedback

Thank You

binghamj@google.com

github.com/webmachinelearning/model-loader