Moving Deep Learning into Web Browser: How Far Can We Go?

Yun Ma et al (2019)

Present by

Mohamed Masoud

Why DL in Browser?

- Developing AI applications that is portable to multiple platforms
- Edge computing.
- Integrated Graphics Cards and WebGL/WebGPU
- Privacy and timely response

Paper Research Questions

- RQ1: What features do existing frameworks provide to implement various kinds of DL tasks in the browser?

- RQ2: How well do existing frameworks perform over different DL tasks?

- RQ3: How big is the performance gap between running DL in the browser and on the native platform

What features do existing frameworks provide to implement various kinds of DL tasks in the browser

Table 1: Characteristics of JavaScript-based frameworks that support deep learning in browsers.

		TensorFlow.js	ConvNetJS	Keras.js	WebDNN	brain.js	synaptic	Mind
Basic Information								
Github Stars		9453	9364	4348	1464	6366	6315	1333
Main Contributor		Google	Stanford University	Leon Chen	The University of Tokyo	Robert Plummer	Juan Cazala	Steven Miller
Last Commit Date		Oct 30, 2018	Nov 25, 2016	Aug 17, 2018	Oct 25, 2018	Nov 5, 2018	Mar 25, 2018	Jul 7, 2017
Status		Active	Not Active	Not Active	Active	Active	Active	Not Active
Functionality								
Support for Training		Y	Y	N	N	Y	Y	Y
Supported Network Types	**DNN**	Y	Y	Y	Y	Y	Y	Y
	CNN	Y	Y	Y	Y	N	N	N
	RNN	Y	N	Y	Y	Y	Y	N
Supported Layer Types		49	7	NA	NA	7	1	1
Supported Activation Types		16	4	NA	NA	4	5	2
Supported Optimizer Types		7	3	NA	NA	1	NA	NA
Support for GPU Accelaration (WebGL)		Y	N	Y	Y	N	N	N
Developer Support								
Documents		Y	Y	Not finished	Y	Only tutorials	Y	Y
Demos		20	10	9	8	7	7	4
Importing Models from Other Frameworks	**TensorFlow**	Y	N	N	Y	N	N	N
	Keras	Y	N	Y	Y	N	N	N
	Caffe&Pytorch	N	N	N	Y	N	N	N
API to Save/Load Model	**Save**	Y	Y	N	N	Y	Y	Y
	Load	Y	Y	Y	Y	Y	Y	Y
Support for Server Side (Node.js)		Y	Y	Y	Y	Y	Y	Y
Library Size		732KB	33KB	650KB	130KB	819KB	106KB	NA

Experiment Setup

- Basic FC DL Model for MNIST handwritten digit recognition database with different configurations to test:
    - Layers (depth) of the neural network, ranges in [1, 2, 4, 8]
    - Hidden layers (width) with number of neurons in range [64, 128, 256]
- Laptop Hasee T97E CPU Intel i7-8750H, Intel HD Graphics 630 and Nvidia 1070 Max-Q (with 8GB GPU memory).
- Ubuntu 18 , Chrome 71

Training Time

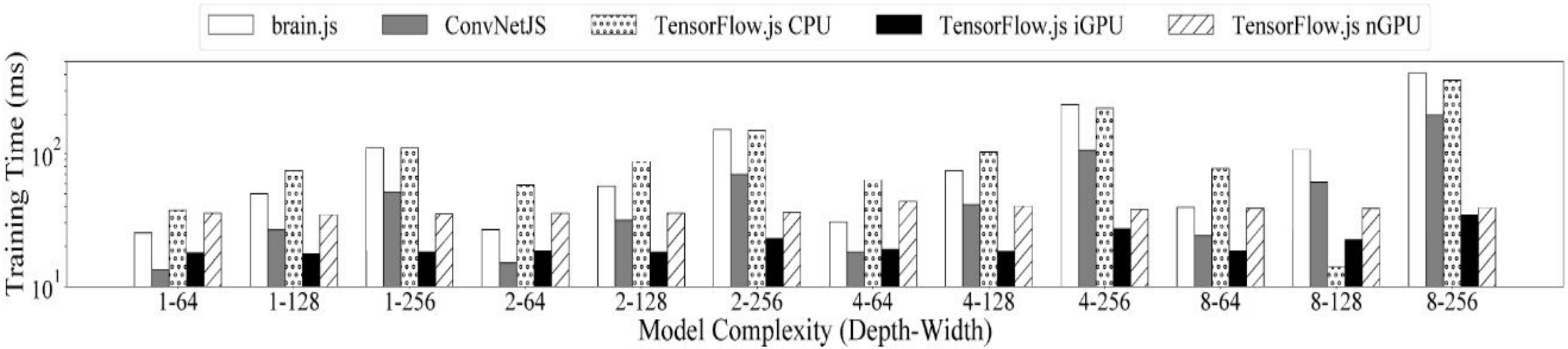


Figure 1: Average training time (ms) on one batch under different model complexities. The y-axis is on log scale.

Model Loading Time

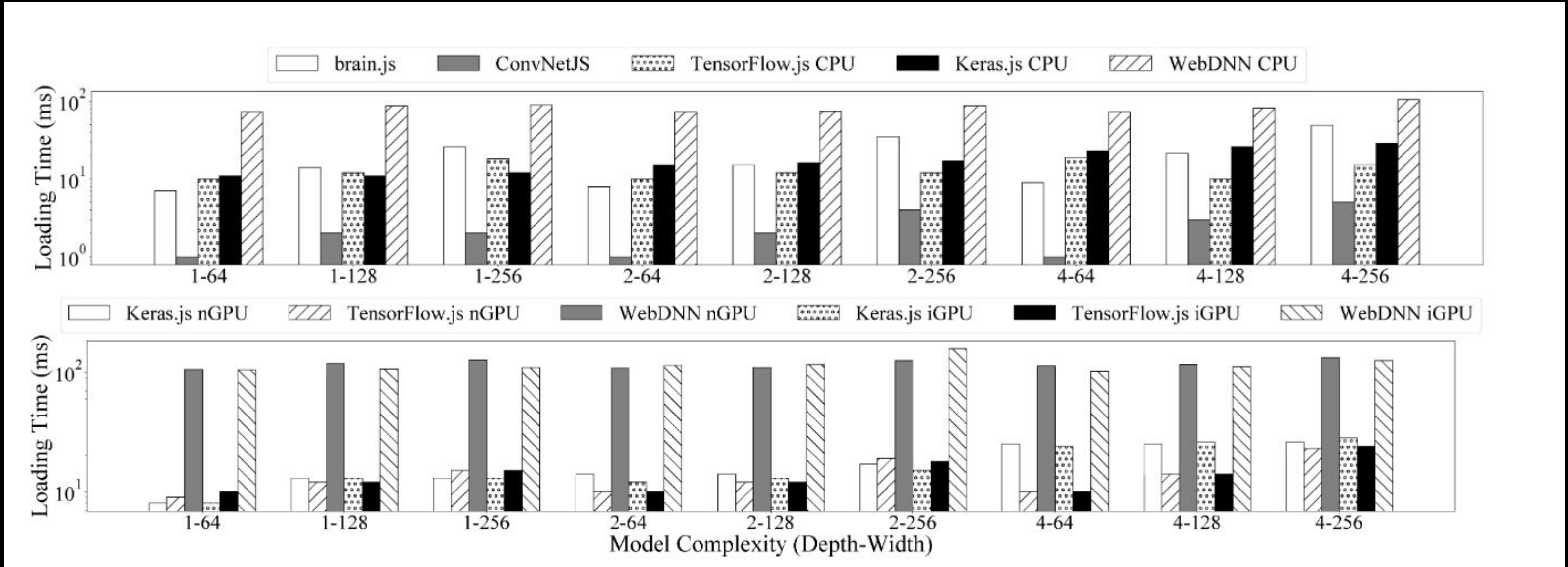


Table 3: Size of model files (MB).

Depth	Width	brain.js	ConvNetJS	synaptic	TensorFlow.js
1	64	1.4	1.3	3.4	0.2
	128	2.7	2.7	6.7	0.4
	256	5.5	5.4	13.3	0.8
2	64	1.5	1.5	3.7	0.2
	128	3.2	3.1	7.8	0.5
	256	7.2	7.1	17.7	1.1
4	64	1.7	1.7	4.2	0.3
	128	4.0	4.0	10.1	0.6
	256	10.7	10.5	26.5	1.6

Inference Time

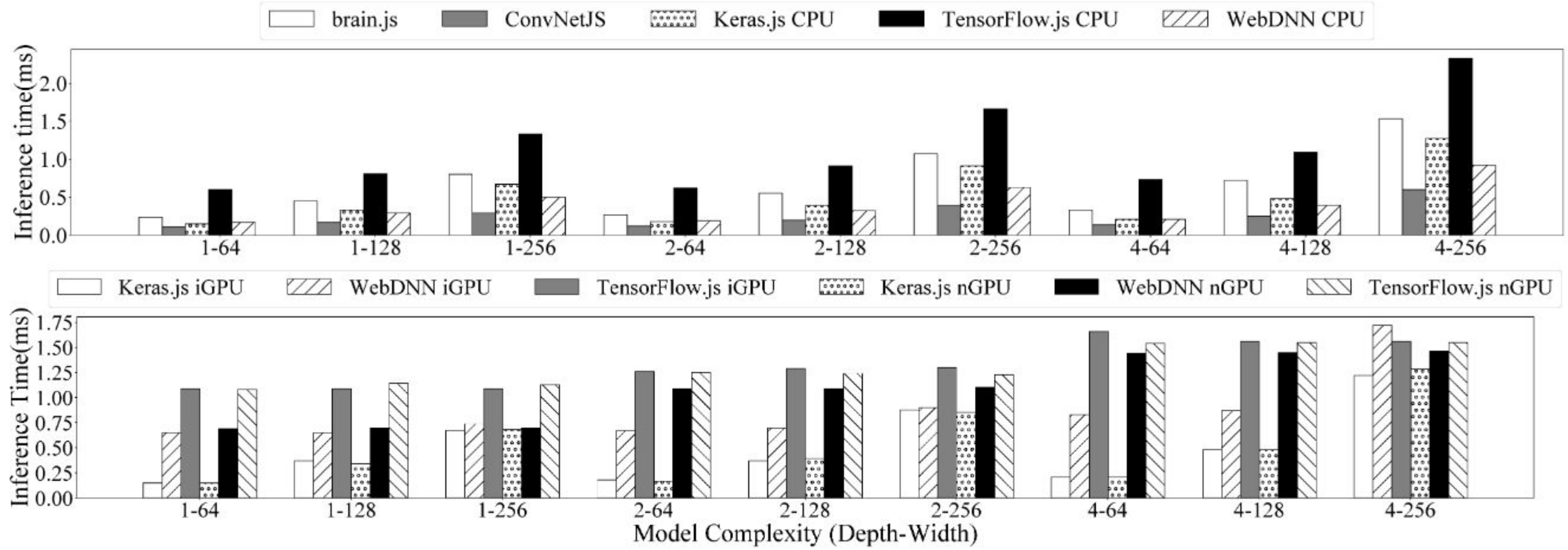
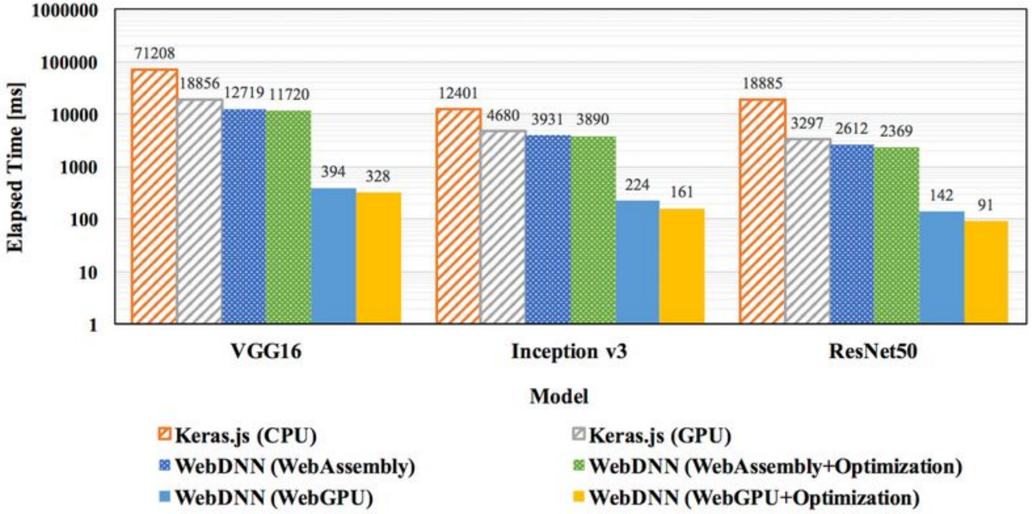


Figure 4: Average inference time (ms) on one sample under different model complexities.

Inference Time

Benchmark

We measured execution time for VGG16 [2] , Inception-v3 [10] , and ResNet50 [3]. Below figure shows the result compared with Keras.js. Computation time per image is shown in vertical axis as logarithmic scale. All tests were run on Mac Book Pro early 2015, Intel Core i5 2.7 GHz CPU, 16 GB Memory, and Intel Iris Graphics 6100 GPU. The web browser is Safari Technology Preview 30.



Source: https://mil-tokyo.github.io/webdnn/

TF vs TFJS

Table 4: Selected Keras pre-trained models.

Model Name	Pre-trained Model Size	Trainable Parameters	Computation (FLOPs)
MobileNetV2	14MB	3.5M	7.2M
DenseNet121	33MB	8.0M	16.3M
Xception	88MB	22.9M	46.0M
InceptionV3	92MB	23.8M	47.8M
ResNet50	99MB	25.6M	51.4M

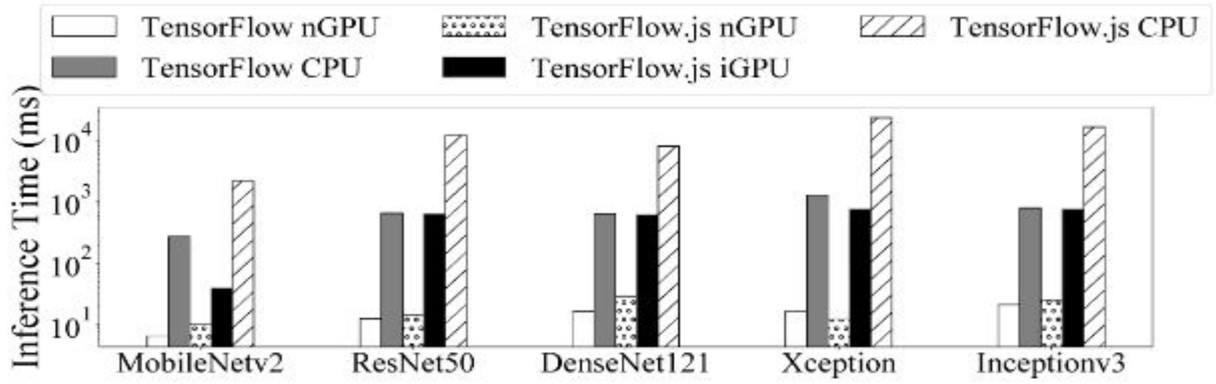
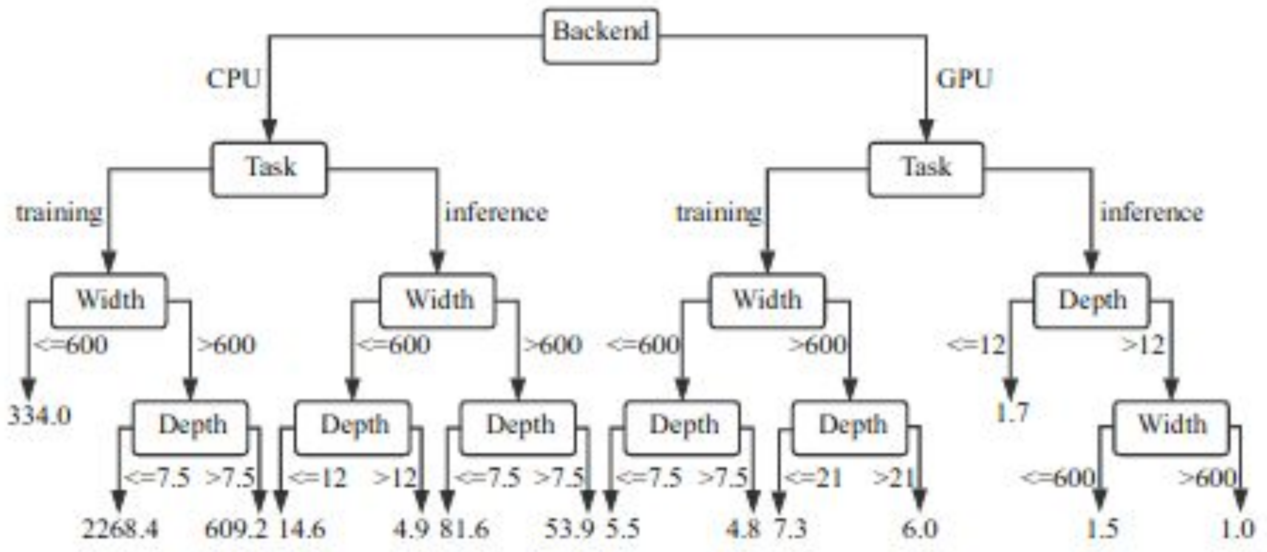


Figure 5: Inference time on pre-trained Keras models. The y-axis is on log scale.

Decision Tree Analysis - TFJS / TF Execution-t Ratio



(b) CNN

Takeaway

Table 6: Major findings and implications of DL in browsers.

No.	Name	Finding	Implication	Stakeholder
1	Specific DL Tasks Support	Frameworks supporting DL in browsers are emerging and being actively maintained. Most of them are not for general purpose and support only a specific subset of DL tasks.	It is better for developers to use general-purpose DL frameworks like TensorFlow.js to implement their DL-powered Web applications.	Application Developer
2	Model Complexity	The width of DL models dominates the performance variation of both training and inference tasks considering the complexity of DL models.	Developers should pay attention to the width of their models, and balance the width and required performance if possible.	Application Developer
3	Model Loading	For inference tasks, loading and warming up the DL model accounts for much longer time than running the inference task itself. The warmup time on the integrated graphics card is generally shorter than that on the standalone graphics card.	Developers should pre-load and warm up the model before using it for inference.	Application Developer
4	Benefits from GPU	For popular pre-trained models like MobileNet and Inception, TensorFlow.js has comparable performance with native TensorFlow when running inference on the standalone graphics card.	It is possible to develop Web applications rather than native applications for these tasks.	Application Developer
5	Benefits from Integared Graphics Card	TensorFlow.js running on the integrated graphics card works better than native TensorFlow running on CPU backend.	For devices without standalone GPUs, developers can use the browser for DL tasks, leveraging integrated graphics card for acceleration.	Application Developer
6	Model File Encoding and Size	Model file encoded in JSON is much bigger (7x) in size than that encoded in binary, and significantly increases the model loading time.	It is better to encode DL models in binary files.	DL-Framework Vendor
7	Framework Call Stack	The call stack of TensorFlow.js is much deeper than that of ConvNetJS, pulling down the performance.	Framework vendors could leverage compiler optimization techniques to reduce the call stack when the DL models are used in the production environment.	DL-Framework Vendor
8	System Resource Utilization	The capability of multi-core CPU cannot be utilized when running DL tasks on the CPU backend in browsers since the JavaScript program is single-threaded. GPU memory usage is limited in 1GB, failing to load and run larger models.	JavaScript engine should take into account the support of multi-process or scheduling among multi cores for better performance of DL tasks in browsers. The GPU memory should be configurable for DL tasks.	Browser Vendor

References

[1] **brain.js**. https://github.com/BrainJS.
[2] **Caffe**. http://caffe.berkeleyvision.org/.
[3] **CNTK**. https://www.microsoft.com/en-us/cognitive-toolkit/.
[4] **ConvNetJS**. https://cs.stanford.edu/people/karpathy/convnetjs/.
[5] **Keras.js**. https://github.com/transcranial/keras-js.
[6] MIL **WebDNN** Benchmark. https://mil-tokyo.github.io/webdnn/#benchmar.
[7] **Mind**. https://github.com/stevenmiller888/mind.
[8] The **MNIST** database of handwritten digits. http://yann.lecun.com/exdb/mnist/.
[9] **MorphCast**. https://www.morphcast.com/.
[10] Sklearn Decision Tree Regressor.
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html.
[11] **synaptic.js**. https://github.com/cazala/synaptic
[12] **TensorFlow** Playgournd. http://playground.tensorflow.org.
[13] **TensorFlow**.js. https://js.tensorflow.org/.
[14] **WebDNN**. https://github.com/mil-tokyo/webdnn.
[15] **WebGL**. https://www.khronos.org/webgl/.
[16] **WebGPU**. https://www.w3.org/community/gpu/

Thank you