DeepThin: A Self-Compressing Library for Deep Neural Networks

Matthew Sotoudeh*
Intel Labs/UC Davis
masotoudeh@ucdavis.edu

Sara S. Baghsorkhi
Intel Labs
sara.s.baghsorkhi@intel.com

1 Introduction and Motivation

As the industry finds more uses for deep learning algorithms in consumer-facing devices, it is becoming clear that running such expensive algorithms on what are usually portable, low-power devices is often infeasible due to the significant storage, performance, and energy limitations involved. This issue has motivated much recent research into machine learning compression methods.

Existing compression methods, however, either struggle to compress models to sizes significantly less than $\frac{1}{50}$ their original size [6, 7], introduce significant storage or computational overhead [3, 6, 8, 15], produce unpredictable compression rates [8], or require significant engineering to implement and tune important hyperparameters [5, 12].

Our work builds on existing research in the area of low rank factorization [4, 10, 11, 14]. We develop a new compression method and library, DeepThin, that:

1. Solves the fundamental scaled-symmetry issue with extremely low-rank matrix factorization of deep learning model parameters
2. Is integrated (along with previous-work techniques for comparison) with the TensorFlow framework
3. Consistently results in up to 60% better accuracy than previous methods
4. Empirically demonstrates inference speed-ups from 2X to 14X

2 DeepThin Architecture

Rank factorization compression algorithms approximate a given weight matrix, $W_{Q\times R}$, with the matrix product of two smaller matrices [4, 10, 11, 14].

Rank-1 factorization of a weight matrix is extremely appealing from both a storage and computational efficiency standpoint, but, as the rank decreases, rows/columns of the reconstructed weight matrix begin to resemble each other - a limitation we have found to significantly impact the learning capacity and accuracy of a network.

To prevent this, DeepThin first applies rank approximation to an auxiliary matrix, W_{aux}, of (semi-arbitrary) size $m \times n$:

$$W_{aux} \approx X^f . W^f \tag{1}$$

where now X^f is a $m \times r$ matrix and W^f is an $r \times n$ matrix.

Finally, rows of W_{aux} are "spread" along columns of $W_{Q\times R}$, as demonstrated in Figure 1, such that the artificial symmetry is broken.

Nevertheless, the above reshape function may still result in patterns in $W_{Q\times R}$, in the form of blocks (of columns) scaled slightly differently. To mitigate this, our library picks an n dimension that is prime relative to R. Our library computes and handles all other constraints necessary to achieve a given compression rate.

The X^f and W^f factors can then be backpropagated to and completely learned during training.

3 DeepThin Library Implementation

We implement both DeepThin and previously proposed compression methods as part of a library module integrated with the open-source TensorFlow [1] framework.

Instead of calling the standard TensorFlow methods to declare a learnable variable, the user calls our library, which creates a separate sub-graph to generate each variable and returns a "tensor" that can be used as a regular variable in TensorFlow. The contents of each sub-graph depend on the compression method and size requested. To the programmer, the process is completely transparent and controllable with a single configuration file.

We also wrote a fused matrix-multiplication operation for DeepThin-compressed networks directly in C++ with MKL [9]. We used two major optimizations unique to the DeepThin method as demonstrated in Figure 1.

4 Results

We compare the accuracy of two state-of-the-art speech recognition models compressed with DeepThin for the same compression rate against previous deep compression techniques, which include:

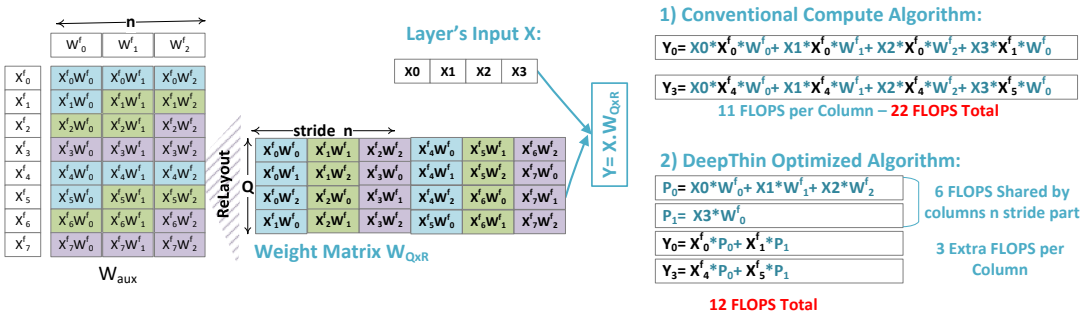
- HashedNet: A small set of possible weight values are distributed into a larger weight matrix according to a (computationally expensive) hashing function [3].
- Pruning: Matrices are iteratively pruned and the resultant sparse matrices stored with CSR format until the overall size is achieved [7].
- Same-Size Networks: Manually lowering the number of hidden units in a network until the desired size is reached
- Rank Factorization: Factoring weight matrices into smaller, lower-rank factors [4, 14].

We also compare against DeepThin-Shuffled – a DeepThin-compressed network where the weight values are randomly distributed throughout the final weight matrix (as opposed to our re-layout operation).

	Rank	Prune	SSNN	Hash
TFKaldi	60.08%	56.96%	23.40%	6.12%
DeepSpeech	28.09%	27.21%	20.45%	12.16%

Table 1. Average relative accuracy improvement of DeepThin compared to four other compression methods. Different compression methods tend to perform better on different datasets, however DeepThin consistently achieves better accuracy results than all other compression methods tested.

*Work completed while at Intel



With **Q** and **n** being relatively prime, after LCM(n, Q) = n*Q entries – every $\mathbf{n}^{th}$ column of W_{QxR} will be a scaled version of the other. We can exploit this redundancy by factoring out the scale operations – multiplication by $\mathbf{X}^{f}$ elements.

Figure 1. Exploiting redundancy generated by our DeepThin compression method to optimize computation of $Y = X.W_{QxR}$.

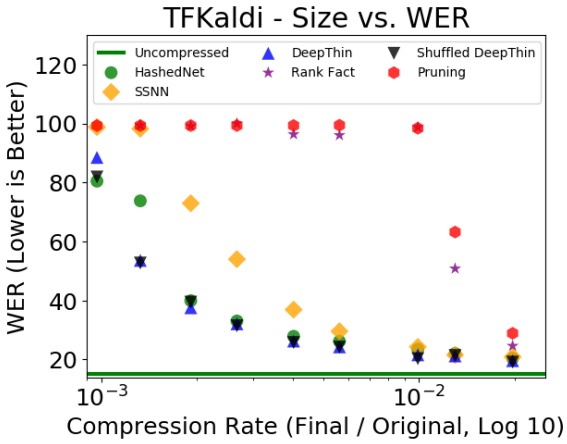


Figure 2. Test word error rate comparison for the TFKaldi model. DeepThin results in lower WERs at all compression rates except for the very smallest configuration (for which HashedNet achieves a lower WER at the cost of significant computational overhead).

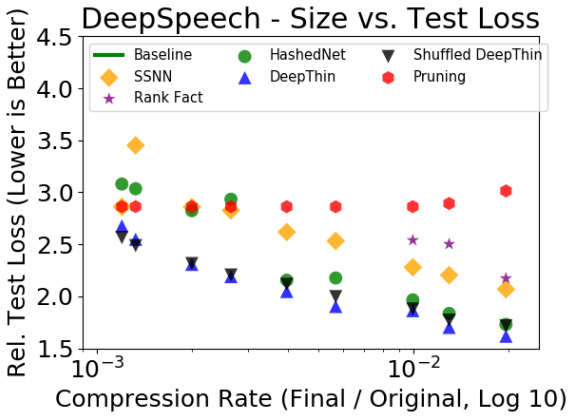


Figure 3. Test loss for the DeepSpeech model. DeepThin significantly outperforms all other compression methods.

We do not report results for CNN models because the unusually large input/output buffers modern CNNs require would prevent inference models from fitting on the target devices even if the weights were completely removed (although there is no fundamental limitation of our method). However, preliminary results show that DeepThin accuracy is on par with or better than competing methods on state-of-the-art ResNet models.

Figure 2 and Figure 3 compare the test error for each compression method on the TFKaldi [13] (a feed-forward network) and DeepSpeech [2] models (a combination of convolutional, feed-forward, and recurrent layers) respectively. DeepThin networks show better results at practically all tested compression rates. The average relative accuracy improvement is summarized in Table 1.

Table 2 demonstrates the impressive performance gains realizable with DeepThin, reaching speeds of up to 14X faster than the uncompressed model. This effect is more pronounced on smaller-cache machines that more closely match a real-world client device, though we also find large performance benefits on even the most capable machine. Performance gains come from a combination of smaller memory footprints that can fit the entire model in the caches, the inhererently simple operations required by DeepThin, and finally DeepThin's capability to re-use partial compututations.

	TFKaldi			DeepSpeech		
~Size	CONF1	CONF2	CONF3	CONF1	CONF2	CONF3
0.0195	3.80X	2.51X	2.46X	7.66X	2.24X	2.08X
0.0129	5.64X	3.16X	3.51X	10.66X	3.09X	2.77X
0.0099	6.47X	3.88X	4.09X	12.12X	3.50X	3.09X
0.0057	8.12X	4.21X	4.84X	13.69X	3.89X	3.67X
0.0040	8.53X	5.69X	5.32X	13.72X	3.86X	3.70X
0.0027	8.22X	5.22X	5.42X	13.04X	3.65X	3.46X
0.0020	7.43X	4.56X	4.58X	11.68X	3.36X	3.22X
0.0014	4.44X	4.12X	2.44X	8.72X	2.57X	2.42X

Table 2. Execution speed-ups ("X faster" than uncompressed) of DeepThin models. CONF1: 1 memory channel/6 MB L3, CONF2: 8 channels/25 MB L3, CONF3: 16 channels/45 MB L3. Most of our performance gains come from fitting the model into L3 cache. The trend starts to reverse for extremely small models due to a reduction in the amount of reuse possible (demonstrated in Figure 1).

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, Jie Chen, Jingdong Chen, Zhijie Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Ke Ding, Niandong Du, Erich Elsen, Jesse Engel, Weiwei Fang, Linxi Fan, Christopher Fougner, Liang Gao, Caixia Gong, Awni Hannun, Tony Han, Lappi Johannes, Bing Jiang, Cai Ju, Billy Jun, Patrick LeGresley, Libby Lin, Junjie Liu, Yang Liu, Weigao Li, Xiangang Li, Dongpeng Ma, Sharan Narang, Andrew Ng, Sherjil Ozair, Yiping Peng, Ryan Prenger, Sheng Qian, Zongfeng Quan, Jonathan Raiman, Vinay Rao, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Kavya Srinet, Anuroop Sriram, Haiyuan Tang, Liliang Tang, Chong Wang, Jidong Wang, Kaifu Wang, Yi Wang, Zhijian Wang, Zhiqian Wang, Shuang Wu, Likai Wei, Bo Xiao, Wen Xie, Yan Xie, Dani Yogatama, Bin Yuan, Jun Zhan, and Zhenyao Zhu. 2016. Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.), Vol. 48. PMLR, New York, New York, USA, 173–182. http://proceedings.mlr.press/v48/amodei16.html

[3] Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. 2015. Compressing Neural Networks with the Hashing Trick. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37 (ICML'15)*. JMLR.org, 2285–2294. http://dl.acm.org/citation.cfm?id=3045118.3045361

[4] Misha Denil, Babak Shakibi, Laurent Dinh, Marc' Aurelio Ranzato, and Nando de Freitas. 2013. Predicting Parameters in Deep Learning. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2148–2156. http://papers.nips.cc/paper/5025-predicting-parameters-in-deep-learning.pdf

[5] David Ha, Andrew Dai, and Quoc Le. 2016. HyperNetworks.

[6] Song Han, Huizi Mao, and William J Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *International Conference on Learning Representations (ICLR)* (2016).

[7] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems (NIPS)*. 1135–1143.

[8] David A. Huffman. 1952. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE* 40, 9 (1952), 1098–1101.

[9] Intel. 2017. Intel Math Kernel Library. (2017). https://software.intel.com/en-us/mkl

[10] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up Convolutional Neural Networks with Low Rank Expansions.. In *British Machine Vision Conference 2014*.

[11] Zhiyun Lu, Vikas Sindhwani, and Tara N. Sainath. 2016. Learning compact recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 5960–5964.

[12] Russell Reed. 1993. Pruning algorithms-a survey. *IEEE Transactions on Neural Networks* 4, 5 (1993), 740–747.

[13] Vincent Renkens. May 2017. Train a tensorflow neural net with kaldi alignments as targets. (May 2017). https://github.com/vrenkens/tfkaldi

[14] Tara N. Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. 2013. Low-rank matrix factorization for Deep Neural Network training with high-dimensional output targets. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (2013), 6655–6659.

[15] R.A. Snay. 1976. Reducing the profile of sparse symmetric matrices. *Bull. Geodesique* (1976).