Efficiently Combining SVD, Pruning, Clustering and Retraining for Enhanced Neural Network Compression

Koen Goetschalckx
koen.goetschalckx@esat.kuleuven.be
MICAS, Department of Electrical Engineering, KU Leuven, Belgium

Bert Moons
bert.moons@esat.kuleuven.be
MICAS, Department of Electrical Engineering, KU Leuven, Belgium

Patrick Wambacq
patrick.wambacq@esat.kuleuven.be
PSI, Department of Electrical Engineering, KU Leuven, Belgium

Marian Verhelst
marian.verhelst@esat.kuleuven.be
MICAS, Department of Electrical Engineering, KU Leuven, Belgium

ACM Reference Format:
Koen Goetschalckx, Bert Moons, Patrick Wambacq, and Marian Verhelst. 2018. Efficiently Combining SVD, Pruning, Clustering and Retraining for Enhanced Neural Network Compression. In *EMDL'18: 2nd International Workshop on Embedded and Mobile Deep Learning , June 15, 2018, Munich, Germany.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3212725.3212733

1 Introduction

Electronic devices are rapidly becoming more and more ubiquitous, simplifying or automating everyday tasks, and even solving problems beyond human capabilities. A large amount of these problems are tackled with artificial neural networks (NN) as they often deliver far better accuracies than classical solutions. Current examples of their applications include the development of self-driving cars [1], face recognition [13] and language modeling [12].

Recently a lot of attention has been devoted to neural network acceleration in order to increase inference efficiency [16]. This is important for high performance compute, yet even more so in embedded systems, where processing power and particularly energy are scarce. To this end, energy expensive operations, in particular data transfers, should be avoided as much as possible. Table 1 lists approximate energy costs of basic operations in a 45nm CMOS chip implementation. These indicate the enormous importance of avoiding accesses to DRAM memory, as this is up to two orders of magnitude more expensive than any other operation [5, 7].

EMDL'18, June 15, 2018, Munich, Germany

ACM ISBN 978-1-4503-5844-6/18/06…$15.00
https://doi.org/10.1145/3212725.3212733

However, the size of most common NN models can be up to tens of megabytes [8]. They hence significantly exceed embedded SRAM capacity and must be stored in DRAM. This results in many energy expensive off-chip DRAM accesses, easily summing up to unaffordable energy costs.

Various techniques have been proposed to compress these DNN models, leading to smaller model sizes and, consequently, significantly lower energy usage. The two popular methods in the state-of-the-art are (1) based on singular value decomposition (SVD) [17] and (2) based on pruning and compressed sparse matrix formats, a technique also referred to as Deep Compression (DC) [6].

To the best of our knowledge, this paper presents the first method jointly exploiting SVD, retraining, pruning and clustering to achieve superior compression in Neural Networks. This paper proposes a more effective combination of SVD, retraining, pruning and clustering steps and compares the resulting compressing approach with the baseline state-of-the-art methods. We hereby show our novel technique outperforms prior art with up 5× increased compression capabilities without loss in inference accuracy.

The main contributions of this paper are hence:

- We **extend the SVD step** with a sequence of iterative *train-compress-retrain* cycles.
- We **combine this looped SVD with further pruning, retraining and clustering** for superior compression.
- We implemented all concatenation strategies in a common algorithmic framework to **compare the different strategies on an equal basis** on a set of benchmarks.

This paper is further organized as follows. Section 2 summarizes the SotA compression techniques Deep Compression and the SVD method. Section 3 presents two important enhancements of these methods: 1) a looped SVD approach; and 2) a concatenation of SVD with pruning and clustering steps. The section moreover discussed the implementation of this approach in a Greedy search algorithm to search an effective combination of lossy compression steps. Next, section 4 is an in depth objective comparison of the compression

Table 1. Energy per operation in a 45nm CMOS process, highlighting dominance of DRAM accesses. [7]

Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit 32KB SRAM	5	50
32 bit DRAM read	640	6400

performance of all discussed stand-alone and concatenated compression methods on a series of benchmarks. Finally, section 5 concludes this paper.

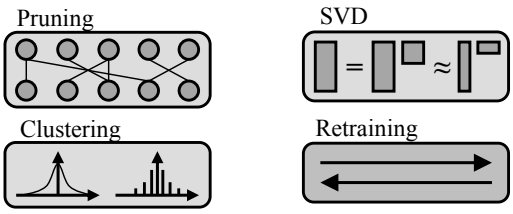

Figure 1. Building blocks: three base compression technique blocks [6, 17] and a block used to indicate retraining (darker shade)

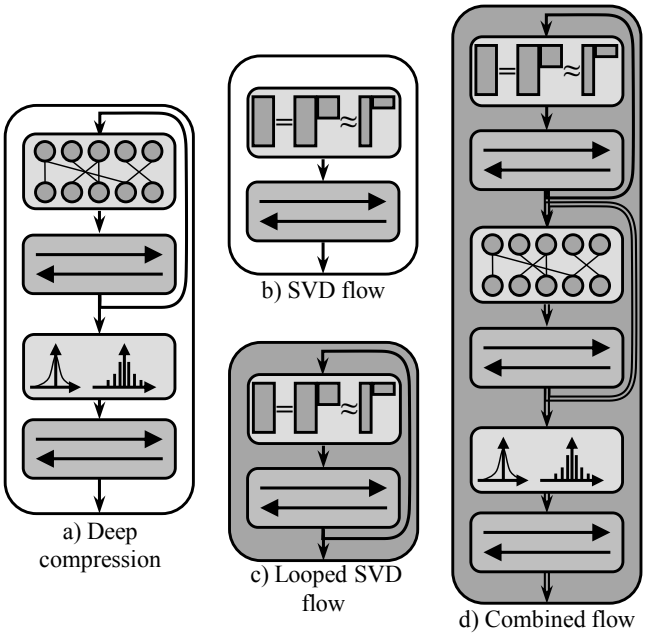

Figure 2. Compression flows built from the building blocks of Figure 1: a) Deep compression [6]; b) SVD [17]; c) looped SVD method, introduced in section 3.1; d) extension of looped SVD with additional pruning and clustering steps, proposed in section 3.2.

2 SotA Deep Compression and SVD building blocks

Both the Deep Compression (DC) [6] and the SVD method [17] can be decomposed into four basic algorithmic building blocks, depicted in Figure 1. Three of those building blocks - pruning, clustering and SVD - represent a lossy compression on a matrix:

- *Pruning* removes redundant network connections with small weights. As such, only a small subset of remaining parameters needs to be stored. The resulting pruned sparse matrix can be represented more efficiently in the differential Compressed Sparse Column (CSC) format, which encodes subcolumns of consecutive zeros with runlengths [6].
- *Clustering* the values in a (sparsified) weight matrix can be done through k-means clustering into n clustersets, such that only a few different weights are possible. Each occurrence can then be described by a small index of $log_2(n)$ bits pointing to the uncompressed value in a small look up table. After an initial clustering step, the values of these clusters centers are retrained to again increase the accuracy of the network. The cluster center values are stored as a high precision floating point or fixed point number [6].
- *SVD* [17] applies singular value decomposition to a weight matrix. This factors it into two new weight matrices of which the columns of the first and rows of the second are sorted by decreasing singular value. These values can intuitively be considered as an importance metric. The rows and columns corresponding to the smallest, least important, singular values are then removed, resulting in two smaller weight matrices. This decreases the total number of weights to be stored, while sequential multiplication of a vector with these smaller weight matrices still closely approximates multiplication with the large original weight matrix.

The fourth algorithmic building block represents retraining of the compressed neural network. This can be applied after any of the previously described lossy compression blocks in order to regain possible lost inference accuracy. Such retraining block can have stop conditions on a minimal improvement of training loss over a certain number of epochs, on a maximal number of epochs and/or on reaching a target error rate (see section 3.3).

Note that in this paper we do not consider the Huffman encoding step from [6], as it has relatively low compression factors and is also dropped in recent works [5]. This lossless Huffman compression technique can however always be added as a final compression step behind any compression flow (SVD or DC or other) without any accuracy loss.

Combinations of these building blocks lead to the state-of-the-art compression flows of Deep Compression and SVD-based compression, as shown in Figures 2(a) and 2(b) respectively. Deep Compression of [6] first uses an iterative loop of pruning and retraining. Each iteration, more connections are removed and the weights of the remaining connections are retrained. After this loop, clustering is applied and the code book is retrained. These compression approaches have resulted in significant model compression at acceptable accuracy losses. Yet, they do not fully exploit the power of combining the different lossy compression building blocks, as is further explored and proven in this paper.

3 Efficiently combining pruning, clustering SVD and retraining

In an effort to achieve enhanced compression, the different algorithmic building blocks of Deep Compression and the SVD-method can be combined in several alternative ways. As a first step, the SVD-method of [17] can be extended with an iterative *train-compress-retrain* approach, similar to what is done in the DC work [6]. Next, this iterative SVD step is concatenated with pruning and clustering steps. Finally, we use these in an efficient implementation of a greedy compression algorithm to limit the compression search space.

3.1 Looped approach for SVD

In contrast to [6], the SVD method from [17] is not looped, but only does a single pass of compression and retraining. In this approach, retraining is only executed at the end, where it must regain a relatively large accuracy loss all at once, instead of repeatedly fine tuning smaller accuracy losses. Therefore, in order to increase the compression abilities of SVD at an equally small accuracy loss, this paper extends the SVD method with the looped approach from [6], as shown in Figure 2(c). Here, SVD compression will be alternated with retraining to regain lost accuracy. Each iteration a predefined number of least significant singular values are removed and, in case accuracy consequently drops below a set target, the network is retrained to regain lost accuracy. The amount of singular values to be removed in each iteration can be fixed or (dynamically) scheduled, similar to the number of weights to prune in each iteration of Deep Compression. To the best of our knowledge, this paper is the first to show that baseline SVD compression rates can be improved upon in an iterative approach with retraining in the loop. The benefits of this approach will be assessed further in section 4.

3.2 Extending looped SVD with pruning and clustering

To further increase compression of a network already compressed by the looped SVD approach, pruning and clustering compression blocks from [6] can be applied afterwards to each of the two matrices resulting from the SVD decomposition. Note that the reversed order, SVD after pruning, is futile as the SVD would destroy the introduced zeros. Also applying SVD again on the two resulting matrices of a first SVD decomposition will not result in improved compression, as these resulting matrices are already (near) orthogonal. We therefore do not further examine these concatenation approaches, but focus on the first proposed approach of applying pruning and clustering after looped SVD. This flow is shown in Figure 2d.

Algorithm 1 Unified compression framework using Greedy compression

Input:
Uncompressed network
List of target matrices (LTM)
Sequence of compression building blocks (SCBB)
with_retraining ▷ flag enabling retraining
ER_{target} ▷ allowed error rate of compressed network

1: Train uncompressed network
2: Validate error rate
3: $ER_{original}$ = error rate of uncompressed network
4: **For each** block **in** SCBB **do**
5: Sort LTM by their model size
6: **For each** matrix **in** sorted LTM **do**
7: Further compress matrix with current block
8: Validate error rate
9: $ER_{current}$ = error rate of new model
10: **if** $ER_{current} \leq ER_{target}$ **then**
11: **Go to** 7 ▷ successful compression
12: **else if** with_retraining **then**
13: Retrain compressed network
14: Validate error rate
15: $ER_{current}$ = error rate of new model
16: **if** $ER_{current} \leq ER_{target}$ **then**
17: **Go to** 7 ▷ successful compression
18: **end if**
19: **end if**
20: Undo last compression step ▷ Compressed too much
21: **end for** ▷ Inner loop goes to next matrix
22: **end for** ▷ Outer loop goes to next compression method

3.3 Unified compression framework

The compression strategies presented in section 3.1 and 3.2 are represented in a common algorithmic framework for fair comparison. Each algorithm is presented by a different concatenation of the lossy compression steps depicted in Figure 1. Algorithm 1 shows the pseudo-code of the developed unified algorithmic framework. The algorithm takes the following inputs:

- *Uncompressed network*: the uncompressed network to be compressed
- *List of target matrices*: the list of weight matrices in the uncompressed network to which the compression techniques should be applied. These are the weight matrices of fully connected layers and, for LSTM layers,

the stacked matrices of all input weight matrices and all hidden weight matrices.

- *Sequence of compression building blocks*: the sequence of compression building blocks that should be applied to the list of target matrices. Each item corresponds to a compression building block in Figure 1. For example, for original Deep Compression this list consists of 'pruning' and 'clustering'. This way, the unified algorithm can implement all presented alternative compression strategies in one unified flow.
- ER_{target}: the maximum allowed error rate of the compressed network. Each compression block will halt further matrix compression whenever the compression step causes the error rate of the compressed model to exceed the target error rate ER_{target}. The algorithm will then always restore the error rate to a value below the ER_{target}, in a manner dependent on *with_retraining* (see below). Note that as such, the error rate of the compressed is always kept below ER_{target}. Thus the final compressed network has error rate $\leq ER_{target}$.
- *with_retraining*: the boolean flag used to indicate that the algorithm should enter a retraining block whenever the error rate rises above the maximum error rate ER_{target}. If the retraining succeeds in restoring the error rate, the algorithm continues with a next iteration to compress the current target matrix. Otherwise, when retraining fails to restore the error rate, the error rate is again restored by undoing the last compression step. When *with_retraining* is false, the algorithm jumps to and executes this undo action immediately. Performing an undo action indicates that the current target matrix is maximally compressed given ER_{target}. Therefore, the algorithm goes on to the next target matrix after an undo action. This flag is used in this paper to compare the resulting compression rates both with and without retraining.

It is important to note that the order in which the different target matrices are compressed matters, as the compression of one matrix influences the compressibility of the other layers in the network. This order is thus part of the search space. To ensure good compression while limiting the search space, a Greedy search is implemented in algorithm 1. More specifically, compression is first applied to the target matrix with the largest model size, as compressing the largest matrix first has the largest impact on the total model size without significantly harming end-to-end accuracy. In a next step, the target matrix with the second largest model size is compressed, and so forth. This greedy approach limits the tested amount of combinations of the compression ratios of the target matrices, while not considerably harming overall compression rate. As such, algorithm 1 compresses matrix by matrix, until all target matrices (layers) are compressed. Moreover, the method ensures the error rate of the final compressed model is never more than the predefined ER_{target}.

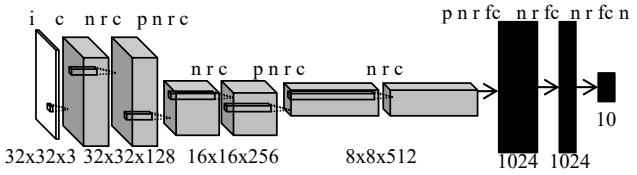

Figure 3. Architecture of the Cifar10-CNN network. (i: input, c: convolutional layer, n: batch normalization layer, r: ReLU, p: maximum pooling layer, fc: fully connected layer.)

4 Experiments and analysis

This section compares the different state-of-the-art techniques with the newly introduced compression flows using the unified algorithmic framework algorithm 1, implemented in Theano [15] and Lasagne [3] and applied to the same benchmark networks. The stepsize for line 7 in this algorithm is fixed to 1% of the singular values for SVD and 1% of the weights for pruning (both relative to the original amounts before compression). In case of clustering, the amount of clusters is halved each time, so that each step can remove a cluster-index bit.

All techniques are tested on multiple networks. These are described next, followed by a description of the used comparison metric. The results are discussed in subsection 4.3. Finally, this section ends with a discussion about the implementation consequences of the compression methods.

4.1 Benchmarks

The different compression strategies are compared across three benchmark networks of diverse nature:

- A CNN, shown in Figure 3, trained and tested on CIFAR-10 [9] with fully connected layers of sizes 8192×1024, 1024×1024 and 1024×10.
- An LSTM network with 39 inputs and a single LSTM layer of 100 units followed by a single fully connected layer with 61 outputs, trained and tested for speech recognition on the TIMIT dataset [4].
- AlexNet [10], which has fully connected layers of sizes 9182×4096, 4096×4096 and 4096×1000, trained [14] and tested on ImageNet [2].

The matrices targeted for compression are the weight matrices of the fully connected layers and, for the LSTM, the stacked matrix of all input weight matrices and stacked matrix of all hidden weight matrices.

Table 2 gives an overview of the tested networks and sequences of compression blocks using the framework of algorithm 1.

Note that AlexNet is not tested with retraining enabled, as this was not feasible in terms of computation time on our systems.

4.2 Details of comparison metric

We compare different flows based on the compression factor they achieve, expressed as the ratio of the uncompressed_size and the compressed_size. Here, the uncompressed_size is the multiplication of height × width ×wordsize of each weight matrix, supposing fp32 (hence wordsize=32 bit). For the compressed_size, the calculation is adapted to the compression methods as follows.

- **SVD:** the sizes of the two resulting weight matrices are added. With only SVD compression applied, compressed_size = $(h_1 * w_1 + h_2 * w_2) * 32bits$. On top of this, the SVD-compressed matrices can be pruned and clustered (see below).
- **Pruning:** a script reads the sparse matrix and derives the sizes of the three CSC arrays (incremental indices, weight values, and column-pointers indicating the starting indices of each column in the previous two arrays) according to the Deep Compression format [Han et al., 2015]. For this, the weight values are still assumed to be 32 bit, while the incremental indices are given the word width which results in the smallest total size. This is achieved by optimally trading off large word widths with filler zeros. The column-pointers have the minimal word width necessary to represent the largest column-pointer.
- **Clustering:** the 32 bit values of the CSC format are replaced by codebook-indices which have the minimal width necessary. Thus, when the codebook contains 16 values (cluster centroids), the 32 bit weight values are replaced by 4 bit indices. The size of the codebook itself is considered negligible and not added to the total model size.

4.3 Results

Table 3 show the results of applying the algorithmic framework of algorithm 1 to realize the alternative compression flows of Table 2, both with and without retraining. Some important observations can be drawn from these results.

Comparing Tables 3a and 3b shows the importance of retraining for each of the benchmarked compression approaches. The retraining allows for an additional compression factor of up to 30x at equal accuracy. On top of that, Table 3 clearly shows that looped SVD is not inferior to pruning. The superiority of pruning to plain SVD claimed in literature is hence mainly due to its incorporated repeated retraining.

When bringing the concatenation of SVD, pruning and clustering into the comparison, it clearly outperforms both Deep Compression and (looped) SVD for all tested benchmark networks, with up to 5× compression gains. This shows that Deep Compression and SVD exploit different kinds of network sparsity. The combined flow creates a synergy, generally leading to higher compression rates.

One can finally observe that for all methods different matrices inside the same networks generally show widely varying compression factors, with the largest compression for larger matrices. This can be due to two reasons. First, large matrices might be relatively more over-dimensioned. Second, the greedy approach of algorithm 1 compresses the largest matrices first, possible leaving little compression opportunities for the smaller ones.

Table 2. Overview of tested compression flows and networks. ✓✓: tested with and without retraining; ✓: tested without retraining only.

Compression flow	C10-CNN	LSTM	AlexNet
Pruning (P)	✓✓	✓✓	✓
Pruning → clustering (P+C)	✓✓	✓✓	✓
SVD (S)	✓✓	✓✓	✓
Looped SVD (LS)	✓✓	✓✓	✓
Looped SVD → pruning (LS+P)	✓✓	✓✓	✓
Looped SVD → pruning → clustering (LS+P+C)	✓✓	✓✓	✓

Table 3. Results of algorithm 1 applied to the networks and flows from Table 2. Numbers represent compression factors. Higher is better. U: uncompressed; P: pruned; C: clustered; S: SVD as in [17]; LS: looped SVD. Note that P+C is the deep compression [6] flow.

(a) With retraining

Matrix / Network	U	P	P+C	S	LS	LS+P	LS+P+C
C10-CNN FC1 (8192x1024)	1	27	114	38	65	224	1114
C10-CNN FC2 (1024x1024)	1	39	154	37	37	40	269
C10-CNN FC3 (1024x10)	1	12	58	1	1	1	3
C10-CNN	**1**	**27**	**117**	**37**	**57**	**132**	**609**
@ Error rate [%]	10.3	10.4	10.4	10.4	10.4	10.4	10.4
LSTM W_in (39x400)	1	1	5	1	2	4	4
LSTM W_hid (100x400)	1	8	24	3	4	31	31
LSTM FC1 (100x61)	1	3	3	1	2	2	8
LSTM	**1**	**3**	**9**	**2**	**3**	**8**	**11**
@ Error rate [%]	22.4	23.0	23.0	23.0	23.0	23.0	23.0

(b) Without retraining

Matrix / Network	U	P	P+C	S = LS	LS+P	LS+P+C
C10-CNN FC1 (8192x1024)	1	2	12	6	6	28
C10-CNN FC2 (1024x1024)	1	1	5	1	1	5
C10-CNN FC3 (1024x10)	1	1	9	1	1	7
C10-CNN	**1**	**2**	**11**	**4**	**4**	**18**
@ Error rate [%]	10.3	10.4	10.4	10.4	10.4	10.4
LSTM W_in (39x400)	1	1	1	1	1	1
LSTM W_hid (100x400)	1	2	2	2	2	2
LSTM FC1 (100x61)	1	1	1	1	1	1
LSTM	**1**	**2**	**2**	**1**	**2**	**2**
@ Error rate [%]	22.4	23.0	23.0	23.0	23.0	23.0
AlexNet FC1 (9182x4096)	1	4	24	13	13	47
AlexNet FC2 (4096x4096)	1	2	16	5	5	26
AlexNet FC3 (4096x1000)	1	1	1	1	1	3
AlexNet	**1**	**3**	**10**	**6**	**6**	**22**
@ Error rate [%]	44.3	47.0	46.5	47.0	47.0	47.0

4.4 Implementation consequences

The different compression methods from Figure 1 also have different impacts on both software and hardware implementations. A model compressed with the SVD method is the easiest to implement as it only requires replacing one matrix vector multiplication by two matrix vector multiplications. Existing matrix-vector multiplication code and accelerators can thus be used for SVD compressed models as well. In high abstraction level software frameworks, SVD compressed fully connected layers can also easily be implemented by creating two new fully connected layers, each with one of the two new weight matrices and the first one without nonlinearity. That way, no special network layers need to be defined and SVD compressed models are supported in software without any lower level code adjustments.

Pruning and clustering techniques are however not as straightforward to implement. The Compressed Sparse Column (CSC) format used after pruning involves extra bookkeeping of indices and irregular memory accesses. Thus, efficient evaluation with pruned models requires specialized hardware, such as the Efficient Inference Engine [5]. Moreover, using clustered weights requires an additional lookup table read for each weight usage, which is often not (directly) supported and hence can cause performance issues. However, the small amount of different weights enables another potential optimization: for each input activation, only a small number of different multiplication results are possible, related to the number of cluster centers. As there are usually more multiplications for a single input activation than there are clusters, pre-calculated multiplication outputs can be cached and reused. This can cause a decrease in multiplier activity or in the required amount of multipliers units. This is exploited in for example [11]. Commercial standard hardware IP, however, can generally not fully exploit pruned and clustered models.

In summary, SVD compressed models can run on default efficient matrix-vector multiplication software and hardware implementations, while pruning and clustering require low level modifications to software and, preferably, also hardware.

5 Conclusion

In energy constrained applications, neural networks require advanced compression, as their large models have to be fetched from expensive off-chip DRAM. Therefore, this paper introduces novel compression approaches based on alternative concatenations of Singular Value Decompositions (SVD), pruning, clustering and retraining. We unify different alternative approaches in a common algorithmic framework, and use this to benchmark their individual compression rates on a set of reference networks. The newly proposed concatenation approach improves compression by up to 5× without any extra loss in inference accuracy, allowing 5× larger models to fit in an energy friendly on-chip SRAM for efficient embedded execution.

References

[1] Bojarski, M., Yeres, P., Choromanska, A., Choromanski, K., Firner, B., Jackel, L., and Muller, U. Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv preprint arXiv:1704.07911* (2017).

[2] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09* (2009).

[3] Dieleman, S., Schlüter, J., Raffel, C., Olson, E., Sønderby, S. K., Nouri, D., et al. Lasagne: First release., Aug. 2015.

[4] Garofolo, J. S., Lamel, L. F., Fisher, W. M., Fiscus, J. G., Pallett, D. S., and Dahlgren, N. L. Darpa timit acoustic phonetic continuous speech corpus cdrom, 1993.

[5] Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., and Dally, W. J. Eie: efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture* (2016), IEEE Press, pp. 243–254.

[6] Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR, abs/1510.00149 2* (2015).

[7] Horowitz, M. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)* (Feb 2014), pp. 10–14.

[8] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. In-datacenter performance analysis of a tensor processing unit. *International Symposium on Computer Architecture (ISCA)* (2017).

[9] Krizhevsky, A., and Hinton, G. Learning multiple layers of features from tiny images. 32–35.

[10] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[11] Shin, D., Lee, J., Lee, J., and Yoo, H. J. 14.2 dnpu: An 8.1tops/w reconfigurable cnn-rnn processor for general-purpose deep neural networks. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)* (Feb 2017), pp. 240–241.

[12] Sundermeyer, M., Schlüter, R., and Ney, H. Lstm neural networks for language modeling. In *Interspeech* (2012).

[13] Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2014), pp. 1701–1708.

[14] Taylor, G., and Ding, W. Theano-based large-scale visual recognition with multiple gpus, 2015.

[15] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints abs/1605.02688* (May 2016).

[16] Verhelst, M., and Moons, B. Embedded deep neural network processing: Algorithmic and processor techniques bring deep learning to iot and edge devices. *IEEE Solid-State Circuits Magazine 9*, 4 (Fall 2017), 55–65.

[17] Xue, J., Li, J., and Gong, Y. Restructuring of deep neural network acoustic models with singular value decomposition. In *Interspeech* (2013), pp. 2365–2369.