C3-Flow: Compute Compression Co-Design Flow for Deep Neural Networks

Matthew Sotoudeh*

Intel Corporation

University of California, Davis

masotoudeh@ucdavis.edu

Sara S. Baghsorkhi

Intel Corporation

sara.s.baghsorkhi@intel.com

ABSTRACT

Existing approaches to neural network compression have failed to holistically address algorithmic (training accuracy) and computational (inference performance) demands of real-world systems, particularly on resource-constrained devices. We present C3-Flow, a new approach adding non-uniformity to low-rank approximations and designed specifically to enable highly-efficient computation on common hardware architectures while retaining more accuracy than competing methods. Evaluation on two state-of-the-art acoustic models (versus existing work, empirical limit study approaches, and hand-tuned models) demonstrates up to 60% lower error. Finally, we show that our co-design approach achieves up to 14X inference speedup across three Haswell- and Broadwell-based platforms.

ACM Reference Format:

Matthew Sotoudeh and Sara S. Baghsorkhi. 2019. C3-Flow: Compute Compression Co-Design Flow for Deep Neural Networks. In *The 56th Annual Design Automation Conference 2019 (DAC '19)*, June 2–6, 2019, Las Vegas, NV, USA. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3316781.3317786>

1 INTRODUCTION AND MOTIVATION

Machine learning algorithms have been increasingly used in consumer-facing products (primarily executed on traditional CPUs [21]), with one particular use case being speech recognition in personal assistants. Although these algorithms would ideally run directly on the client devices which utilize them (such as Amazon's Echo [20] and Google's Home [13]), their reliance on large weight matrices makes executing such expensive algorithms infeasible due to the significant storage, performance, and energy limitations involved.

Recent research has investigated methods of compressing models to sizes which can be efficiently executed directly on the client device. Existing compression methods, however, either cannot effectively compress models to sizes significantly less than $\frac{1}{50}$ their original size [7, 8], introduce significant storage or computational overhead [3, 8, 9, 19], produce unpredictable compression rates [9],

or require significant engineering time to implement and tune important hyperparameters [6, 16].

Denil et al. have shown that weight matrices may be factored into smaller matrices at training time to aid in distributed model training [5, 12, 14, 15, 18]. Unfortunately, previous work hits a lower bound where factorizations of rank smaller than ~ 5 are unable to learn any useful representation of the data. This significantly limits the size and performance gains realizable from existing factorization techniques, and motivates our work in understanding and overcoming this limitation.

2 C3-FLOW ARCHITECTURE

In this section, we describe how C3-Flow can be applied to fully-connected layers, but note that the idea extends naturally to other layer types.

We define a single layer with non-linearity a , weights W , and biases B like so:

$$Y = a(X \cdot W + B) \quad (1)$$

, where W and B must be stored. For clarity, we only show here compression of the W parameter, but do compress B as well in our evaluations.

Traditional rank factorization replaces the W parameter with a function of a smaller set of learned parameters [5, 6, 18]. For example, in Figure 1, a $Q \times R$ weight matrix W is represented as the product of two lower rank factors, $W_{Q \times R} \approx X^f \cdot W^f$, where X^f is a $Q \times r$ matrix and W^f is $r \times R$. During training, the error signal is back-propagated to X^f and W^f , and the learned factors are then used to reconstruct the full $W_{Q \times R}$ matrix at inference time.

Lower ranks are appealing from both a storage and efficiency standpoint, however, as $r \rightarrow 1$, columns of the reconstructed weight matrix begin to resemble scalar multiples of each other. As columns in a weight matrix define the patterns recognized by each node in the layer, this clearly causes all of the recognized patterns to be redundant to each other, severely limiting the ability of the network to learn to recognize distinct patterns in the input.

To mitigate such constraints, C3-Flow first applies rank approximation to an *auxiliary matrix* W_{aux} of size $m \times n$:

$$W_{aux} \approx X^f \cdot W^f \quad (2)$$

where X^f is $m \times r$ and W^f is $r \times n$.

Next, elements of W_{aux} are redistributed to form $W_{Q \times R}$ such that the artificial symmetry is broken. This reshaping process adds non-uniformity to the approximation function like an activation function add a non-linear decision boundary to the network's output layer. There is a large set of possible transforms we could use. On first glance, we would ideally randomly scatter elements, but

*Work completed during an internship at Intel.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

DAC '19, June 2–6, 2019, Las Vegas, NV, USA

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-6725-7/19/06...\$15.00

<https://doi.org/10.1145/3316781.3317786>

Figure 1: Low-rank factorizations are ideal, yet as $r \rightarrow 1$ columns in the reconstructed matrix become redundant, scaled versions of each other. With a well-designed relayed operation, we can break symmetry and improve computational efficiency.

preliminary tests as a limit study found such an approach extremely slow (due primarily to random memory accesses). Instead, for each layer type intended to be supported, we propose to choose a re-shaping process that enables a highly-efficient fused kernel.

For matrix multiplication, for example, a more cost-effective alternative would be to tile rows of W_{aux} along columns of $W_{Q \times R}$ as shown in Figure 1. This is especially appealing as it constructs (part of) a column of $W_{Q \times R}$ at a time, in an array of consecutive memory location, while multiplying them by different rows of input matrix X in Equation 1. Thus, a fused matrix multiplication kernel can make full use of the generated elements before discarding them.

To our surprise, in Figures 3 and 5, reshaping with this method often achieves better accuracy than the random scattering (“C3-Flow Shuffled”) method. We hypothesize that the random scattering is unable to take advantage of local patterns in the data and weight matrices, implying a trade-off between exploiting such patterns and being unnecessarily limited by them.

Finally, the above “re-layout” function may still result in repetition patterns in matrix $W_{Q \times R}$ in the form of blocks (of columns) scaled slightly differently. To mitigate this problem, we pick the number of columns in W^f to be prime with respect to the number of rows in $W_{Q \times R}$, i.e.

$$\text{LCM}(n, Q) = n \times Q \quad (3)$$

The LCM value determines the repetition frequency of similar, scaled blocks in matrix $W_{Q \times R}$. Here Q , the original weight matrix width, is fixed. If we only wanted to minimize such repetition, we would choose the largest value for n that is prime with respect to Q (for example, $n = Q + 1$, $m = R$ would be one such trivial choice). But parameter n must also satisfy other constraints within the C3-Flow compression framework. First, matrix W_{aux} must have at least as many elements as matrix $W_{Q \times R}$. Therefore:

$$m \times n \geq Q \times R \quad (4)$$

In addition, C3-Flow parameters need to abide by the specified compression rate. Because C3-Flow compresses weights on a per-matrix basis, we can calculate the compressed size of any individual compressed matrix in the network. Therefore, for a compression ratio of α and a weight matrix of shape $Q \times R$ we have:

$$\alpha = \frac{r \times (m + n)}{Q \times R} \quad (5)$$

where the denominator totals the size of $W_{Q \times R}$, and the numerator sums up the size of low rank factors of matrix W_{aux} . Note that we do not include the size of the bias vector in this calculation. In practice, we compress and compute the bias vector’s compression rate separately if required by the user. Again, note that Equation 5 ignores other, much smaller parameters of the network that are not compressed (such as batch-normalization [11] parameters), thus the actual network size will have some overhead. If the exact network architecture is known, one may add those other parameters to Equation 5 to get an exact ratio for the entire network or to further compress the other parameters to reach a desired overall compression rate (which our library handles automatically). To facilitate ease of comparison with prior work we maintain the simple form of Equation 5 in this work.

Now, if we replace m in Equation 5 with its lower bound value derived from Inequality 4, for each rank factorization value ($r = 1, r = 2, \dots$), we have a single variable quadratic inequality as shown below:

$$\alpha \geq \frac{r \times \left(\frac{Q \times R}{n} + n \right)}{Q \times R} \quad (6)$$

for which valid ranges of n can be easily determined if any exist. Within valid ranges, we then pick a minimum value of n that satisfies conditions expressed by Equations 4 and 3. Note that we take the *minimum* n here as, in most cases, $\frac{Q \times R}{n} \gg n$, so taking *larger* n will *decrease* the compressed size. Thus, taking the minimum n satisfying Inequality 6 gives the largest network that is still smaller than the desired compressed size (i.e., satisfies the desired

compression rate but is not significantly over-compressed beyond that).

Since this fine tuning only happens once during initialization of the network the performance overhead is negligible.

Non-existence of a valid range for n in Inequality 6 indicates that the spatial overhead of C3-Flow surpasses any compression benefit it provides for the specific matrix being compressed. This creates an effective lower-bound on the compression rate α supported by C3-Flow. This lower-bound is dependent on the size of the original network weight matrix $W_{Q \times R}$ but is often less than $\frac{1}{1000}$ of the matrix size. Furthermore, as different matrices in a network may have different lower bounds, we may over-compress larger matrices in order to “make up for” matrices that have hit this lower bound, thus achieving a desired overall compressed network size.

2.1 Inference Optimization

A major focus of our work was on designing a compression method that could be efficiently computed (while retaining accuracy). After finding experimentally in Section 3 that our C3-Flow method results in better accuracy than any competing method, we decided to investigate the real-world performance of C3-Flow-compressed networks further by writing a custom inference kernel that computes a C3-Flow-compressed matrix multiplication without explicitly decompressing the entire weight matrix at once. We predict that such a fused operation will significantly lower runtime memory, compute, and energy usage of the inference phase by fitting the entire model in the compute device’s cache and taking advantage of particularly designed features of the C3-Flow compression method to reduce total computation.

Written in C++ using MKL [10] for computations and interfacing with the TensorFlow C++ APIs, our operation takes three parameters: the low-rank factors X^f and W^f , as well as the layer input to be multiplied. We tested a range of values for r , and found that $r = 1$ works either as well or better than other possible values, and as such all future discussion will assume $r = 1$.

As demonstrated in Figure 2, the decompressed weight matrix W has copies of W^f tiled in a column-major fashion throughout the matrix. Each tiled copy of W^f is scaled by the corresponding scalar in X^f . The matrix multiplication is computed such that each cell in the output matrix consists of the sum of the dot products between each of the X^f -scaled copies of W^f in a single column and the respective slices of the input X . We consider two major optimizations for the above C3-Flow-compressed matrix multiplication.

First, instead of scaling each copy of W^f by an element of X^f and then computing the dot product against the input slice (requiring $2 \times n$ multiplications and n additions), one can *first* compute the dot product between the relevant portions of W^f and the input slice, then scale the resulting scalar value(s) by the scalar elements of X^f (requiring $n + 1$ multiplications and n additions).

Second, at certain points in the computation we will end up computing the dot product between the same elements of W^f and the same input slice. This redundancy can be exploited, such that after the first time a particular W^f product is computed for a particular input slice that dot product is stored, and the kernel scales that dot product by all the X^f values using it and sums those partial

products for all of the cells that make use of that same dot product, cutting down on the most expensive part.

Finally, due to the implicit transpose in our re-layout operation, all memory accesses to X^f and W^f in our fused kernel are contiguous. For other layer computations, the re-layout operation can be modified accordingly to produce similar memory-access benefits.

3 ACCURACY RESULTS

We compare the accuracy of models compressed with C3-Flow against previous deep compression techniques:

- HashedNet: Weight values distributed pseudo-randomly into a matrix according to a hashing function [3]
- Pruning: Matrices iteratively pruned and the resultant sparse matrices stored with CSR format [7]
- Same-Size Networks: Manually lowering the number of hidden units to reach a desired size
- Rank Factorization: Factoring weight matrices into smaller, lower-rank factors [5, 18]

For C3-Flow, all networks were compressed with $r = 1$ and n calculated as discussed in Section 2. We also compare against “C3-Flow-Shuffled,” where weight values are randomly distributed throughout the final weight matrix (as opposed to our re-layout operation).

We compared accuracy and inference speedup on two state-of-the-art speech recognition models. All accuracy plots have had models with a very similar compression rate grouped together horizontally around their mean compression rate for readability. Additionally, due to high lower-bounds, rank factorizations tend to have fewer possible configurations.

Finally, we discuss preliminary CNN results on a ResNet CIFAR model, demonstrating its applicability to a variety of layer types.

3.1 TFKaldi/Voxforge

First, we evaluated C3-Flow on an acoustic model from the commonly used, open-source toolkit TFKaldi [17] and Voxforge dataset [1]. The feed-forward network has 36,080,400 parameters and an application (speech recognition) that is often used on low-power devices. In order to encourage replication, we used the standard, open-source testing scripts, which report lowest word error rate (WER) from a series of test batches (although results are near-identical when taking the average instead). It achieves an uncompressed error of 15.17% WER.

As expected, in Figure 3 C3-Flow networks show the best results at practically all tested sizes. In particular, it outperforms “SSN”s with hand-tuned reduction in model parameters, clearly showing that our method makes more efficient use of parameters.

Next, we find that pruned models perform particularly poorly, losing nearly all ability to learn models smaller than $\frac{1}{100}$ of the uncompressed network. This is due to the requirement that vast majorities of the model parameters be set to 0 in order to maintain the compression rate, which is exacerbated by the inefficiencies of CSR.

Although HashedNet models require *significantly* more computation and randomized memory access patterns [3], at almost all configurations C3-Flow outperforms HashedNet.

Rank factorization performs very poorly due to the increase in symmetry effects discussed in Section 2. Solving this symmetry

Figure 2: Exploiting redundancy generated by weight compression to optimize computation of $Y = XW_{QxR}$.

Figure 3: C3-Flow models result in lower WERs at all compression rates tested except for the smallest, $\frac{1}{1000}$, configuration, for which extremely inefficient HashedNet achieves a lower WER. Pruning and rank factorization rapidly degrade in performance as the model becomes smaller. Even with non-scalable manual effort, same-size networks fail to match our low error rates.

issue with the re-layout operation is one of our main contributions, and the improvements over rank factorization highlight our success.

Finally, we see that our method performs very similarly to a “shuffled C3-Flow” approach, indicating that our re-layout operation — while being significantly more efficient to compute — is still able to effectively deal with the symmetry issues.

Figure 4 shows the change in validation loss during training for a single compressed size (about $\frac{1}{250}$ of the uncompressed model). Note that the pruned network (in red) begins training as an uncompressed model, then prunes weights incrementally every 50 iterations until the 550th iteration. As expected, pruning has very little effect on accuracy for the first few pruning iterations when it prunes exclusively unimportant weights. Beyond that, pruning iterations are visible as large ‘jumps’ in the loss, as the network is forced to remove vital weights. This visually demonstrates both

	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 improvement of C3-Flow compared to four other compression methods. TFKaldi: relative WER decrease. DeepSpeech: relative test loss reduction. Computed by taking the mean of the relative improvements for each size for each compression method in Figure 3. C3-Flow consistently beats all other compression methods tested.

TFKaldi - Iteration vs. Validation Loss (1-250)

Figure 4: Validation loss during training for a single compression rate ($\frac{1}{250}$) on TFKaldi. The pruned model is not completely compressed until iteration 550, at which point it trains very poorly. Additionally, instability in the training of rank-factored matrices is removed in the C3-Flow curve through the use of the re-layout operation. HashedNets have similar accuracy, but rely on random memory access patterns and complex hash computations that significantly impede model runtime performance.

how well pruning works for fairly large sizes and the negative effects of pruning past this point (when the validation loss begins to increase significantly after each pruning step).

Rank-factorization, meanwhile, is particularly unstable, primarily due to the symmetry effects described in Section 2, which motivated and is solved by the use of an auxiliary matrix and re-layout operation in C3-Flow.

Same-size networks converge slowly, while HashedNet accuracy is comparable to C3-Flow, but requires both the computation of a complex hash and a random memory access for each individual weight in a weight matrix, having extreme effects on runtime performance. In contrast, we show in Section 4 that C3-Flow can be highly optimized to run significantly faster than uncompressed models.

3.1.1 Larger Sizes. Although C3-Flow was designed to enable scenarios where accuracy degradation is an acceptable tradeoff for extremely small sizes, we have also compared C3-Flow to existing work at sizes ranging from $\frac{1}{4}$ to $\frac{1}{50}$ of the original size, and found that C3-Flow continued to have the best accuracy at all tested sizes (in many cases retaining all of the uncompressed network accuracy). This, along with the significant efficiency benefits of C3-Flow, demonstrates that it is a strong method for almost any compression problem.

We note that this is traditionally the only scenario existing state-of-the-art compression methods are designed for, which shows in the extremely poor performance at smaller sizes displayed in Figure 3. By contrast, C3-Flow is able to both beat existing methods at the larger sizes they were designed for, as well as retaining significantly more accuracy at the smaller sizes that C3-Flow was designed to enable.

3.2 DeepSpeech

We next tested on a state-of-the-art speech recognition system called DeepSpeech [2]. This RNN model, open-sourced by Baidu, combines convolutional, GRU [4], and feed-forward layers to produce a character prediction from a raw frequency spectrum.

We found the open-source model had a tendency to overfit the limited training data available. To overcome this, we ported it to TensorFlow, added learning rate decay, and removed batch normalization [11] on the recurrent layers.

Again, we relied on the testing script included with the open-source model, which reports average test CTC loss, to encourage reproducibility. Our uncompressed, base-line model achieves a character error rate (CER) of 36.9% and sum CTC loss of 1310 (higher than the published error rates as we use the publicly-available training set and model).

Figure 5 compares the best test loss after twenty training epochs for each compression method compared on the DeepSpeech model. Here, C3-Flow networks outperform all other compression methods at all tested configurations. Average relative improvements are summarized in Table 1.

We believe that pruning performs particularly poorly for this model because it contains a number of smaller matrices, this forcing many layers to be pruned almost completely. This issue compounds itself as the input goes through the series of heterogeneous layers, each layer receiving significantly less information.

Same-size models perform competitively when the model size is relatively large, but lose almost all learning capacity as the sizes decreases.

Figure 5: Test loss on the DeepSpeech model. C3-Flow models significantly outperform all other compression methods, while same-size networks perform well only for less-compressed sizes. Pruning, rank factorization, and HashedNets all quickly devolve into ineffective models, only ever predicting the few most common letters (or complete silence) instead of learning useful models.

HashedNets are more consistent, and do not lose learning ability as much as same-size or pruned networks, however even at larger sizes they perform worse than C3-Flow networks. Coupled with the serious performance and optimization issues encountered with HashedNets, they are clearly not a good choice for this model.

Rank-factorization models, meanwhile, perform poorly and hit lower-bounds at approximately $\frac{1}{100}$ the original size - making them very difficult to compare to other methods and impossible to use for the majority of compression ratios targeted.

Furthermore, as with TFKaldi, we find that our proposed method performs slightly better than a shuffled C3-Flow approach. Again, this demonstrates the effectiveness of our re-layout operation in removing symmetry issues while remaining highly efficient to compute. We also expect the non-shuffled C3-Flow to do particularly well here, as it can still take advantage of local patterns in the input data that are particularly strong for the raw frequency spectrum inputs of DeepSpeech compared to the pre-processed features in the TFKaldi model.

3.3 Extending to CNNs

Although we focused on fully-connected and recurrent networks, we note that our method can apply to any linear algebra computation involving large learned matrices. We performed an initial investigation into the applicability of C3-Flow to a convolutional ResNet model by compressing the filter tensors, and found that C3-Flow consistently beat or was competitive with other methods. However, we did not extensively optimize the approach, and CNNs require unusually large intermediate buffers (over 89% of runtime memory usage in our ResNet model), making such parameter savings unimportant to overall runtime memory usage.

~Size	TFKaldi			DeepSpeech		
	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-up from C3-Flow, “X faster” than the speed of the uncompressed model. Size is compressed size divided by uncompressed size. Largest gains come from platforms with smaller cache sizes, but execution time is reduced across all configurations tested, making it ideal for environments where latency and power usage are important. Once the model is small enough to fit in a sufficiently-fast cache, the most important factor becomes the reuse frequency of partial dot products, which is controlled by n and worsens for smaller models.

4 PERFORMANCE RESULTS

We evaluated the inference performance of C3-Flow across three Haswell- and Broadwell-based platforms. All implementations leverage MKL. We did not test the CNN model, as writing an optimized convolution kernel from scratch would require a large team multiple weeks of effort, and was not the main focus of our work.

CONF1 is a Broadwell machine with 4 cores/8 threads at 3.30 GHz, 1 memory channel, and 6 MB of L3 cache. CONF2 is a Broadwell-E with 10 cores/20 threads at 3.00 GHz, 8 memory channels, and 25 MB L3. CONF3 is a Haswell with 36 cores/72 threads running at 2.30 GHz, 16 memory channels, and 45 MB L3. All machines had 32 KB of L1 and 256 KB of L2 cache, and all tests were run on the CPU only to match a typical inference environment.

We ran 1,000 one-batch iterations through the uncompressed TFKaldi and 100 one-batch iterations through the uncompressed DeepSpeech networks. We then ran the same tests on C3-Flow-compressed versions of each model using the fused kernel and report the speedup for each configuration in Table 2.

As expected, we find impressive performance gains, sometimes reaching speeds of up to 14X faster than the uncompressed model. This effect is primarily visible on the machine equipped with smaller caches that more closely represents a real-world lower-end client situation, though we also find large performance benefits on even the most capable, cluster machine.

Although most of the gains come from a smaller, cache-able, memory footprint, this alone is not enough – models like Hashed-Net, for example, have similarly small memory usage but require the computation of an expensive hash function and a random, non-contiguous memory access to compute the matrix multiplication. C3-Flow, by contrast, requires no such hashing function and uses completely contiguous memory accesses. Furthermore, as discussed

in Section 2.1, C3-Flow models can be further optimized for inference performance by re-using partial products, which can cut the amount of computation by over 2X in some models.

5 ACKNOWLEDGEMENTS

We thank James P. Held for supporting this work through an Intel Research Velocity Award.

REFERENCES

- [1] VoxForge english corpus, August 2017.
- [2] Dario Amodei et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 173–182, New York, New York, USA, 20–22 Jun 2016. PMLR.
- [3] Wenlin Chen et al. Compressing neural networks with the hashing trick. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML’15, pages 2285–2294. JMLR.org, 2015.
- [4] Kyunghyun Cho et al. On the properties of neural machine translation: Encoder–decoder approaches. *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014.
- [5] Misha Denil et al. Predicting parameters in deep learning. In C. J. C. Burges et al., editors, *Advances in Neural Information Processing Systems 26*, pages 2148–2156. Curran Associates, Inc., 2013.
- [6] David Ha et al. Hypernetworks. 2016.
- [7] Song Han et al. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1135–1143, 2015.
- [8] Song Han et al. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *International Conference on Learning Representations (ICLR)*, 2016.
- [9] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.
- [10] Intel. Intel math kernel library, 2017.
- [11] Sergey Ioffe et al. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *international conference on machine learning*, pages 448–456, 2015.
- [12] Max Jaderberg et al. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference 2014*, 2014.
- [13] Bo Li et al. Acoustic modeling for google home. 2017.
- [14] Zhiyun Lu et al. Learning compact recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5960–5964, 2016.
- [15] Alexander Novikov et al. Tensorizing neural networks. *neural information processing systems*, 28:442–450, 2015.
- [16] Russell Reed. Pruning algorithms-a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.
- [17] Vincent Renkens. Train a tensorflow neural net with kaldi alignments as targets, May 2017.
- [18] Tara N. Sainath et al. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6655–6659, 2013.
- [19] R.A. Snay. Reducing the profile of sparse symmetric matrices. *Bull. Geodesique*, 1976.
- [20] Anthony Weber et al. *Amazon Echo: The Best User Guide to Master Amazon Echo Fast*. CreateSpace Independent Publishing Platform, USA, 2016.
- [21] Carole-Jean Wu et al. Machine learning at facebook: Understanding inference at the edge. *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2019.