

Tensorfolding: Improving Convolutional Neural Network Performance with Fused Microkernels

Michael Anderson, Evangelos Georganas, Sasikanth Avancha, and Alexander Heinecke
Intel Corporation, Parallel Computing Lab, Intel Labs

Abstract—Convolution layers are prevalent in many classes of deep neural networks, including Convolutional Neural Networks (CNNs) which provide state-of-the-art results for tasks like image recognition, neural machine translation and speech recognition. In the recent past, several techniques to improve generalization capabilities of neural networks have been developed; the most prominent and successful is batch normalization. In deep neural network training, the batch normalization layer consists of a memory-bandwidth bound kernel. On the latest Intel Skylake based Xeon processors, a significant portion of execution time is spent in this kernel. By leveraging the CPU’s large caches and its latency-optimized execution model, we are able to reduce this kernel’s time to a bare minimum while allowing to improve forward pass layer runtimes by 21% compared to an unfused implementation and by 2% compared to a fused implementation.

I. INTRODUCTION

In the last few years, deep learning has evolved into one of the most important computational concepts. Several academic groups and companies have released open source frameworks which abstract many implementation details from the data scientist: Tensorflow [1] and Caffe [2], are two of the most popular ones according to GitHub stars. Small convolutions and small matrix multiplications (GEMM) arranged as building blocks in each layer are the predominant kernels when training deep neural networks. Methods to optimize these standalone kernels on CPUs have been published as prior work [3].

In order to achieve best inference results on unknown or test data, the neural network needs capabilities to generalize the provided training information. Researchers have developed several techniques to enhance generalization capabilities and the most widely used technique is called batch normalization [4]. While batch normalization (BN) helps improve the accuracy of inference tasks significantly, it is challenging from a computational point of view. It is a memory-bandwidth-bound operation that scales activation tensors. Consequently, a sub-optimally integrated or implemented BN routine can significantly slow down the execution of the training process. Therefore, this work has two main contributions:

- We describe and evaluate how BN and other memory-bandwidth-bound operations can be fused to minimize performance overhead
- We describe and evaluate how memory-bandwidth-bound operations, convolutions and GEMMs can be overlapped by following a DAG-based approach on sub-tensors

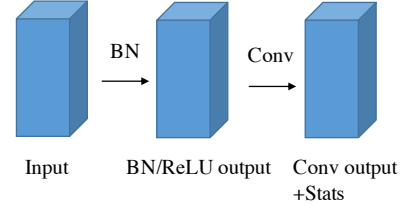


Fig. 1: Memory passes for Fused Batch Normalization.

All of these techniques are highlighted by using the well-known ResNet-50 topology as an example. We also discuss their generalization to other neural network topologies. The proposed tensorfolding technique benefits only the training phase of neural networks. In case of inference, layers such as BN can be captured within the pre-computed weights. Hence, such layers can be eliminated entirely during the inference phase.

II. FUSED BATCH NORMALIZATION AND TENSORFOLDING

The ResNet topology, a large number of non-convolution layers, e.g. BN, ReLU and split (elementwise) layers, in the main part of the topology. Following a straightforward implementation for BN, ReLU etc. would mean that every compute-bound convolution is followed by 4-5 passes over the output data: a) calculating the expected value of activations $E(X) = \frac{1}{N} \sum_{i=1}^N x_i$ b) calculating the variance of activations $Var(X) = E[(X - E(X))^2]$ c) norming of the activations using $E(X)$ and $Var(X)$ d) applying ReLU e) applying split layer (if needed).

A. Fused Batch Normalization

As discussed, but not deeply analyzed in [3], [5] these 4-5 passes can be reduced to 1 pass over the data. This is done by a fused norming and ReLU and an optional elementwise operation. The two activation statistics (mean and variance) can be merged into the the convolution kernel by exploiting the following formula for decoupling variance calculation from the mean value calculation: $Var(X) = E[(X - E(X))^2] = E(X^2) - E(X)^2$. We call the 1 pass BN *Fused Batch Normalization* as shown in Fig. 1. Note that a variance calculation following this scheme might suffer from numerical instabilities when assuming full spectrum floating point values. However, the ranges of activation values in deep learning tasks are very narrow; therefore this is not an issue.

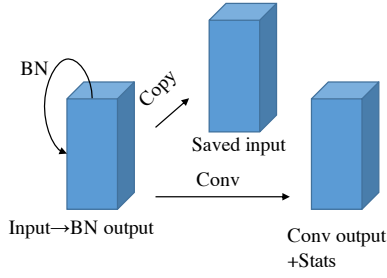


Fig. 2: Memory passes for Folded Batch Normalization.

Count	Conv	ReLU	BN	Eltwise	C	K	H/W	R/S	stride
1	x				3	64	224	7	2
1	x				64	256	56	1	1
3	x	x	x		64	256	56	1	1
1	x				64	64	56	1	1
3	x	x	x		64	64	56	3	1
2	x	x	x	x	256	64	56	1	1
1	x				256	512	56	1	2
1	x	x	x	x	256	128	56	1	2
4	x	x	x		128	128	28	3	1
4	x	x	x		128	512	28	1	1
3	x	x	x	x	512	128	28	1	1
1	x				512	1024	28	1	2
1	x	x	x	x	512	256	28	1	2
6	x	x	x		256	256	14	3	1
6	x	x	x		256	1024	14	1	1
5	x	x	x	x	1024	256	14	1	1
1	x				1024	2048	14	1	2
1	x	x	x	x	1024	512	14	1	2
3	x	x	x		512	512	7	3	1
3	x	x	x		512	2048	7	1	1
2	x	x	x	x	2048	512	7	1	1

TABLE I: Description of ResNet-50 fused layers

B. Tensorfolding

Tensorfolding’s objective is to further hide the cost of the remaining pass of BN behind the computation of the convolution. Note that we need to store the batch-normed activations for the backward pass as input for gradient calculations. Instead of storing the batch-normed inputs in a separate buffer, we use streaming stores to save the input prior to batch norm to another buffer, then write the batch-normed input into the input buffer. This is depicted in Fig. 2. While the total amount of memory transferred is not reduced compared to Fused Batch Normalization, we test whether this particular memory traffic pattern leads to better overlap of computation and communication. Additionally, the BN operation is broken down into small sub-tensors which are executed in a DAG-fashion during the convolution or GEMM operation. For example, some DAG nodes which execute small GEMMs for convolution, and others execute fused kernels. Such fused operations are combined in a fine grained manner using a JIT compiler, where possible, to maximize the overlap of computation and communication.

III. PERFORMANCE EVALUATION

We evaluate performance of (1) naive four-pass batch normalization, (2) fused batch normalization, and (3) tensorfolding for forward propagation of ResNet-50 training on a single socket on an Intel® Scalable Xeon® 8180 processor with turbo mode disabled. We benchmark each layer in isolation and compute a weighted average performance to estimate the performance of the full network. The fusability of each layer in ResNet-50 depends on its exact location in the network. We summarize our fused network layer benchmark in Table I.

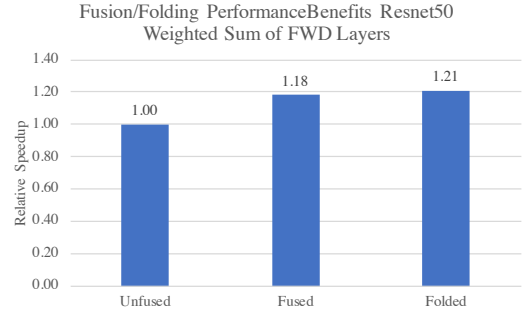


Fig. 3: Performance of Fused and various foldings on a Intel Scalable Xeon Platinum 8180 CPU.

Figure 3 depicts the measured performance of this benchmark with minibatch 224 for the different variants discussed. Applying the fused batch norm attains an 18% increase in performance over the baseline which uses an unfused batch normalization. Tensorfolding only provides a small improvement over the fused batch normalization.

IV. RELATED WORK AND GENERALIZATION

Optimizing BN layers has been already proposed and used in [3], [5]. However, although being used, it has not been that precisely evaluated as in this work. Also work on GLOW [6], aims to provide simple, potentially automatic, loop fusion. Other recent work, called batch normalization fission [7], aims at similar goals as the described tensorfolding in this work. However, it was carried out as an inner module optimization of DenseNet, whereas tensorfolding works across split layers and Residual modules. TVM [8] and Tensor Comprehensions [9] aim at generally solving the problem presented here. Often performance of such fully-automated tools is sub-optimal [10], hence they are complementary to our work which can be seen as rooflining these general tensor compilers. Additionally, these fusion techniques are not limited to CNNs with BN, and can be transferred to RNN/LSTM topologies. Memory bandwidth bound activation functions (e.g. sigmoid, tanh) can be folded into the GEMM operation when the destination matrix is still hot in some cache level close to the execution units.

V. CONCLUSION AND FUTURE WORK

In this poster, we presented various techniques to limit performance impacts of BN on low bandwidth CPU platforms. While fused batch normalization increases performance of our forward pass benchmark by 18%, the tensorfolding optimization is only slightly better at 21% above the baseline. This gain is possible as the large caches of the CPU can be efficiently used for latency hiding and bandwidth savings when fusing multiple kernels. We expect to find other benefits by moving from layer-focused implementation of neural network algorithms to a DAG-based formulation of sub-tensors with aggressive fusion, as was done in this work, perhaps by testing other network topologies or hardware configurations.

REFERENCES

- [1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. [Online]. Available: <http://tensorflow.org/>
- [2] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional Architecture for Fast Feature Embedding,” *arXiv preprint arXiv:1408.5093*, 2014.
- [3] E. Georganas, S. Avancha, K. Banerjee, D. Kalamkar, G. Henry, H. Pabst, and A. Heinecke, “Anatomy of high-performance deep learning convolutions on simd architectures,” in *Accepted to Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’18. IEEE Press, 2018.
- [4] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: <http://arxiv.org/abs/1502.03167>
- [5] D. Das, N. Mellempudi, D. Mudigere, D. D. Kalamkar, S. Avancha, K. Banerjee, S. Sridharan, K. Vaidyanathan, B. Kaul, E. Georganas, A. Heinecke, P. Dubey, J. Corbal, N. Shustrov, R. Dubtsov, E. Fomenko, and V. O. Pirogov, “Mixed precision training of convolutional neural networks using integer operations,” *CoRR*, vol. abs/1802.00930, 2018. [Online]. Available: <http://arxiv.org/abs/1802.00930>
- [6] N. Rotem, J. Fix, S. Abdulrasool, S. Deng, R. Dzhabarov, J. Hegeman, R. Levenstein, B. Maher, N. Satish, J. Olesen, J. Park, A. Rakhov, and M. Smelyanskiy, “Glow: Graph lowering compiler techniques for neural networks,” *CoRR*, vol. abs/1805.00907, 2018. [Online]. Available: <http://arxiv.org/abs/1805.00907>
- [7] D. Jung, W. Jung, B. Kim, S. Lee, W. Rhee, and J. H. Ahn, “Restructuring batch normalization to accelerate cnn training,” 2018.
- [8] T. Chen, T. Moreau, Z. Jiang, H. Shen, E. Q. Yan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, “TVM: end-to-end optimization stack for deep learning,” *CoRR*, vol. abs/1802.04799, 2018. [Online]. Available: <http://arxiv.org/abs/1802.04799>
- [9] N. Vasilache, O. Zinenko, T. Theodoridis, P. Goyal, Z. DeVito, W. S. Moses, S. Verdoolaege, A. Adams, and A. Cohen, “Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions,” *CoRR*, vol. abs/1802.04730, 2018. [Online]. Available: <http://arxiv.org/abs/1802.04730>
- [10] <http://vertex.ai>, “Tensor Compilers: Comparing PlaidML, Tensor Comprehensions, and TVM,” 2018. [Online]. Available: <http://vertex.ai/blog/compiler-comparison>

Optimization Notice: Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to <http://www.intel.com/performance>.

Intel, Xeon, and Intel Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries.