A Deeper Look at FFT and Winograd Convolutions

Aleksandar Zlateski[*†] **Zhen Jia**[*‡] **Kai Li**[‡] **Fredo Durand**[†]
[†]**Massachusetts Institute of Technology** [‡]**Princeton University**
[*]**Equally contributed**

Motivation and Previous Work

Since convolutional layers are computationally expensive and dominate the total execution time of modern deep ConvNets [13, 16, 18, 19], many efforts have been made to improve the performance of the convolutional primitives for CPUs [1, 7, 20, 25, 27], GPUs [4, 8, 15, 21] or both [26]. Initially, several approaches using FFT–based convolutions were proposed [15, 21, 25, 26]. Recent work by Lavin et al. on Winograd–based convolutions [14] demonstrated a great speedup, which shifted the focus from FFT–based to Winograd–based implementations, as it became widely accepted that the Winograd–based approach provides greater reduction in the number of operations required by the algorithm, especially for small kernels (e.g. 3×3). A well optimized manycore CPU implementation [3, 12] of the Winograd approach can improve the performances by more than 3X.

The main reduction in operations in the Winograd method, compared to FFT, comes from the fact that it works with real numbers. However, due to its numerical instability, the Winograd method can only use small tile (transform) sizes [7, 14, 22], which result in a larger amount of required data movement to and from memory. In contrast, the FFT–based method does not suffer from such instability, thus larger tile sizes can be used, which can partially reduce the number of required operations and greatly reduce the amount of data movements; these savings can, in certain cases, offset the increase in the number of operations due to complex arithmetic.

These observations raise the question, under what conditions the Winograd-based approach performs better than the FFT–based approach and vice versa, and how to compare the two approaches.

Our Contributions and Novelty

In this paper, we propose a performance model based on the idea of Roofline [23] that, in addition to the number of operations, accounts for the total amount of data movement (to and from memory), as well as the arithmetic intensity (operations per moved byte) to compare the Winograd-based and FFT-based approach. To estimate the run times, alongside the processor's speed, our model also considers memory bandwidth and the cache sizes. We have also compared optimized implementations of both approaches on the same hardware.

Our model is suitable for modern processors, both CPUs and GPUs, which tend to have large compute–to–memory ratios, the ratio between the speed (in FLOPS) and memory bandwidth. Both compute speed and memory bandwidth are improving exponentially. However, the exponent for speed is substantially larger than that for memory bandwidth [24], which results in an increasing compute–to–memory ratio. For instance, the 4.5 TFLOPS Intel Knights Landing processor [11] has a compute–to–memory ratio of 11, and the latest Skylake Xeon processor family has ratios in the range of 20 to 30. Titan Xp GPU from NVIDIA has a ratio of 20, and Tesla V100 has a ratio of 16 for single precision computation, and a ratio of 133 for half–precision computation using its dedicated tensor cores.

Our model suggests that whether the Winograd or FFT approach is faster depends on both the layer and hardware it is executed on. However, on average, the FFT–based approach outperforms the Winograd–based ones on most commonly used networks, with the margin increasing as the system's compute–to–memory ratio increases.

Through a set of empirical experiments, we confirm the predictions of our model on modern CPUs.

Performance Model

The Roofline model [23] estimates an application's upper bound performance as a function of its arithmetic intensity, derived by dividing the number of floating point operations by the total data movement. The runtime (T) is estimated by:

$$T = \frac{\text{FPO}}{\min(\text{Peak FLOPS},\ \text{AI} \times \text{MB})} \tag{1}$$

Where MB indicates the memory bandwidth; AI the arithmetic intensity and FPO is short for the total number of required floating point operations. In our model, data movement is defined as movement between main memory and any of the caches. We focus on 32-bit floating point arithmetic, similar analysis can be performed for 16-bit half-precision floats.

Both FFT–based and Winograd–based approaches perform a change of basis of the inputs and kernels, in which convolution becomes a multiplication. Both approaches contain four stages [12, 14, 15, 21]: ① transforming input images, ② transforming kernels, ③ performing element–wise products of the transformed images and kernels, which is an equivalent problem to a matrix multiplication (with real matrices for Winograd, and complex matrices for FFT), and ④ transforming the results of ③ back to the spatial domain.

The time required for each stage can be estimated using Eqn. 1. The transform stages (①,②,④) have very low arithmetic intensity – lower than 2.5 for Winograd and lower than 3.5 for FFT, and will thus be memory bound on all modern systems. In this case Eqn. 1 can be reduced to $T = \frac{DM}{MB}$, where DM represents data movement in bytes.

The element–wise product stage will generally have a larger AI, which will depend on the number of layer's input and output channels and the amount of available cache. However, the FFT–based approach will typically have the AI twice as large as its Winograd counterpart, due to working in the complex domain. In some cases, on machines with large compute–to–memory ratios, the Winograd method will be memory bound, while the FFT method is compute bound.

Relative Performances We are interested in the relative performances of the two approaches. We define the speedup

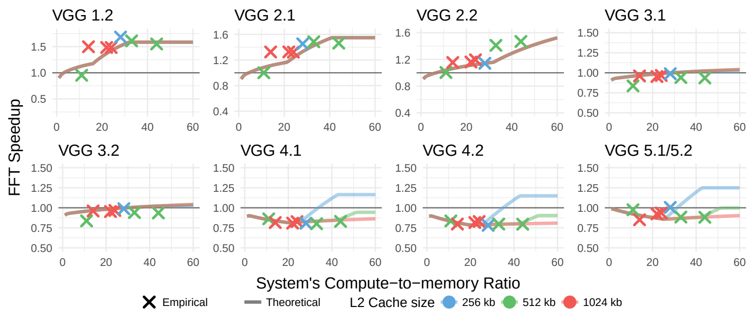

Figure 1: Estimated and empirically obtained speedup of FFT–based convolutions versus Winograd–based ones on CPUs with various compute–to–memory ratios and cache sizes.

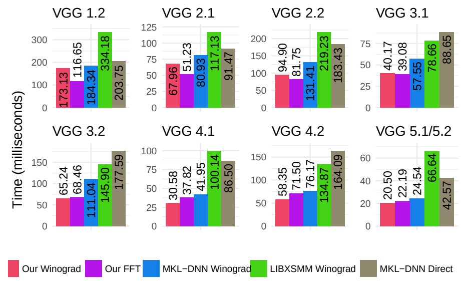

Figure 2: Absolute performances of our Winograd– and FFT– based implementations compared to the latest version of MKL-DNN and LIBXSMM.

$\alpha = \sum_S T_s^W / \sum_S T_s^F$ to be the ratio of the time required for processing a layer using the Winograd approach and the time required using the FFT approach, each of which is estimated to be equal to the sum of times required for all four stages. When α is greater than one, the FFT method is expected to run faster.

The speed of both methods will depend on the tile sizes used. For Winograd, due to its numerical instability, we allow tile sizes up to 6×6, as larger tile sizes can increase numerical errors by two or more orders of magnitude, when compared to direct convolution [12, 14], resulting in an unstable computation. This limit is also enforced by the implementations, such as MKL-DNN [2], LIBXSMM [5, 10], and cuDNN [8]. For the FFT method we allow for an arbitrary tile size, however, using tiles larger than 64^2 is never optimal. Using the methodology from [14], we estimate that the FFT method has lower error than direct convolution for all tile sizes up to 64^2.

Implementation and Empirical Results

To empirically confirm the estimates of our model on modern systems, we used the fastest available implementation of the Winograd–based approach [3, 12] which uses "wincnn" [6] to generate transform codelets and just–in–time (JIT) compiled primitives for fast matrix multiplication on AVX512 capable machines. We extended the implementation to support AVX2, and implemented an FFT–based approach by replacing the "wincnn" primitives, with ones generated by "genfft" from the fftw package [9], as well as implemented JIT primitives for complex matrix multiplication.

In Fig. 1 we show the empirically measured values for the speedup of the FFT–based approach over the Winograd–based one, together with the theoretical estimates of our model for forward propagation of the layers of VGG [18] network. The color represents the amount of L2 cache available (note that the lines sometimes overlap). A total of 7 system configurations were used, two of which were AVX2–based and four AVX512. The slowest CPU was capable of approximately 1 TFLOPS (Intel i7-6950X desktop CPU), while the fastest one was capable of 4.5 TFLOPS (Intel Xeon Phi 7210). The oldest generation of CPUs was Intel E7-8890v3 with 1.44 TFLOPS and only 256KB of L2 cache, whereas the other CPUs had either 512 or 1024 KB of cache per core. The position on the x axis was determined by dividing processors' speed (in FLOPS), by theoretically achievable memory bandwidth.

As our model suggests, and our empirical results confirm, for some of the layers of VGG, the FFT approach is faster and for some the Winograd one is faster. The model was also confirmed on AlexNet [13] and OverFeat [17] layers (not presented) where FFT was an optimal choice for all layers.

For credibility, we compare absolute performances of our implementations against two state–of–the–art implementations of Winograd–based convolutions (latest MKL-DNN and LIBXSMM, as January 2018) and the fastest implementation of direct based convolution (MKL-DNN). Fig. 2 shows the performances of our two implementations against other implementations on a 2 TFLOPS Intel i9-7900X processors based system (compute–to–memory ratio of 22). Our Winograd implementation outperformed on 5 out of 7 CPUs that supported AVX512 (MKL-DNN and LIBXSMM do not provide an AVX2 implementation). The relative performances of our FFT implementation was consistent with our model.

Conclusion and Discussion

While on popular neural networks, the FFT approach outperforms on average. The choice of the algorithm should, perhaps, be decided on a per layer basis.

Generally, the relative performance of FFT over Winograd increases in the following scenarios: (1) as the compute–to–memory ratio increases, (2) when the cache size is limited, (3) when the image sizes are large, (4) when the number of input/output channels is relatively small.

Another important finding is that no specific sizes of FFT are required, in contrast to the popular belief that power–of–two sizes should be used. The arithmetic intensity of FFT transforms are much lower than the compute–to–memory ratios of modern processors, even for large prime numbers, and are thus memory bound. Using non–typical FFT sizes, such as large prime numbers can, in certain cases, greatly improve the performances, as it can minimize the amount of required padding, and thus unnecessary computation, when dividing images into tiles.

Another FFT–based approach, using Gauss' complex number multiplication, as proposed in [14], was analyzed, implemented and empirically confirmed. This approach reduced the amount of operations with the expense of more data movement, and for certain layers and on certain systems it can outperform both the Winograd and regular FFT–based approach.

References

[1] 2016. FALCON Library: Fast Image Convolution in Neural Networks on Intel Architecture. "https://colfaxresearch.com/falcon-library/". (2016).

[2] 2016. Intel(R) Math Kernel Library for Deep Neural Networks. "https://github.com/01org/mkl-dnn". (2016).

[3] 2018. N-Dimensional Winograd–based convolution framework. https://bitbucket.org/poozh/ond-winograd. (2018).

[4] Accessed: 2018-01-01. Intel® Nervana reference deep learning framework. https://github.com/NervanaSystems/neon. (Accessed: 2018-01-01).

[5] Accessed: 2018-01-01. LIBXSMM. https://github.com/hfp/libxsmm. (Accessed: 2018-01-01).

[6] Accessed: 2018-01-01. Wincnn. "https://github.com/andravin/wincnn". (Accessed: 2018-01-01).

[7] David Budden, Alexander Matveev, Shibani Santurkar, Shraman Ray Chaudhuri, and Nir Shavit. 2016. Deep Tensor Convolution on Multicores. *arXiv preprint arXiv:1611.06565* (2016).

[8] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* (2014).

[9] Matteo Frigo and Steven G Johnson. 1998. FFTW: An adaptive software architecture for the FFT. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, Vol. 3. IEEE, 1381–1384.

[10] Alexander Heinecke, Greg Henry, Maxwell Hutchinson, and Hans Pabst. 2016. LIBXSMM: accelerating small matrix multiplications by runtime code generation. In *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 981–991.

[11] James Jeffers, James Reinders, and Avinash Sodani. 2016. *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition*. Morgan Kaufmann.

[12] Zhen Jia, Aleksandar Zlateski, Kai Li, and Fredo Durand. 2018. Optimizing N-Dimensional, Winograd-Based Convolution for Manycore CPUs. In *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[14] Andrew Lavin and Scott Gray. 2016. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4013–4021.

[15] Michael Mathieu, Mikael Henaff, and Yann LeCun. 2013. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851* (2013).

[16] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. 2014. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*. 2924–2932.

[17] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. 2013. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229* (2013).

[18] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[19] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[20] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. 2011. Improving the speed of neural networks on CPUs. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, Vol. 1. 4.

[21] Nicolas Vasilache, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. 2014. Fast convolutional nets with fbfft: A GPU performance evaluation. *arXiv preprint arXiv:1412.7580* (2014).

[22] Kevin Vincent, Kevin Stephano, Michael Frumkin, Boris Ginsburg, and Julien Demouth. 2017. On Improving the Numerical Stability of Winograd Convolutions. (2017).

[23] Samuel Williams, David Patterson, Leonid Oliker, John Shalf, and Katherine Yelick. 2008. The roofline model: A pedagogical tool for auto-tuning kernels on multicore architectures. In *Hot Chips*, Vol. 20. 24–26.

[24] Wm A Wulf and Sally A McKee. 1995. Hitting the memory wall: implications of the obvious. *ACM SIGARCH computer architecture news* 23, 1 (1995), 20–24.

[25] Aleksandar Zlateski, Kisuk Lee, and H Sebastian Seung. 2016. ZNN–A Fast and Scalable Algorithm for Training 3D Convolutional Networks on Multi-core and Many-Core Shared Memory Machines. In *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, 801–811.

[26] Aleksandar Zlateski, Kisuk Lee, and H Sebastian Seung. 2016. ZNNi: maximizing the inference throughput of 3D convolutional networks on CPUs and GPUs. In *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 854–865.

[27] Aleksandar Zlateski and H Sebastian Seung. 2017. Compile-time optimized and statically scheduled ND convnet primitives for multi-core and many-core (Xeon Phi) CPUs. In *Proceedings of the International Conference on Supercomputing*. ACM, 8.

Acknowledgments

We thank Sebastian Seung for helpful discussions, and Nir Shavit for providing an access to a Skylake Xeon Gold Server. We are grateful to Toyota Research Institute for supporting the Toyota - CSAIL Joint Research Center at MIT, and Intel Corporation for supporting the Intel Parallel Computing Center at Princeton University. Zhen Jia was partially supported by IARPA (D16PC00005).