3D UNet Optimization Using TVM

First Author[1[0000−1111−2222−3333]], Second Author[2,3[1111−2222−3333−4444]], and Third Author[3[2222−−3333−4444−5555]]

[1] Princeton University, Princeton NJ 08544, USA
[2] Springer Heidelberg, Tiergartenstr. 17, 69121 Heidelberg, Germany
lncs@springer.com
http://www.springer.com/gp/computer-science/lncs
[3] ABC Institute, Rupert-Karls-University Heidelberg, Heidelberg, Germany
{abc,lncs}@uni-heidelberg.de

Abstract. The field of connectomics aims to map the interconnections between biological neurons within nervous systems at the scale of single synapses to gain insights into the structure and functional organization of biological neural networks. A critical task for the success of the connectomics enterprise is the segmentation of neurites from high precision electron microscopy (EM) images. This task requires models to be both accurate and computationally efficient in order to process large volumes of very high precision microscopy images. In recent years, segmentation models have become very accurate. In this paper, we analyse the computational efficiency of one such successful 3D U-Net and identify several computational bottlenecks. We propose different optimizations to increase the computational efficiency of this model and achieve a x

Keywords: Neurite segmentation · 3D CNN · Computational efficiency

1 Introduction

Precise reconstruction of neural connectivity is of great importance to understand the functional organization of biological nervous systems. 3D electron microscopy (EM) can capture large volume of neuronal tissues within nano-scale precision, which allows for the identification of even the smallest neuronal objects like vesicles. With advances in imaging technologies, EM systems can now produce terabytes of images within hours. Manually annotating each of the neurons within such large EM volumes is simply impractical as it would requires lifetimes of manual labeling from highly skilled experts to segment. Hence, high-precision segmentation models are needed to automate the reconstruction of neuronal circuits.

Current state-of-the art models for 3D neurite segmentation proceeds in three steps, as illustrated in Figure 1. First, a CNN is trained to detect boundaries between neurons in the raw EM images. In a second step, over-segmentation maps are computed from the boundary map. This is typically done by non-parametric algorithms like Watershed. Finally, an agglomeration algorithm is used to process the over-segmented results typically produced by the Watershed algorithm.

In this pipeline, the first step is the computational bottleneck as both the training of the segmentation model and the inference using trained model are computationally expensive. The benefits of optimizing the computational efficiency of the 3D U-Net is two-folds:

First, computationally efficient models would significantly ease the inference step: AS the resolution needed to accurately detect synaptic connections is on the order of a few nanometers, only a cubic millimeter of brain tissue at this resolution represents on the order of X Bytes. Processing such a large volume is considerable computational challenge. Hence optimized computations will be needed in order to enable the processing of large brain volumes.

However, the most impactful benefit concerns the training step: As training an unoptimized model takes up to days on a single GPU machine, the iterative process needed to fine-tune models and experiment with different architectures and loss functions is very slow. Optimizing the computational efficiency of our model would enable us to more efficiently explore the space of solution. Getting faster iteration cycles would allow us to more efficiently come up with better solutions.

In this paper, we thus analyse the computational efficiency of a successful 3D U-Net segmentation model. We identify several factors responsible for slowing down the computations. We propose several tweaks to speed up the computations and obtain a relative speed up of X

In the next section, we start by presenting related research. We then analyze the baseline model in Section XXX, and present our optimization in Section XXX. Section XXX discusses the results of our experiments.

2 Analysis of baseline model

2.1 Background knowledge

The first stage of the optimization convolutional neural networks is the analysis of our model and evaluate whether the model is efficient or not. Through analysis results, we could find out where the model needs to be optimized.

We could optimize the model from the following factors: number of computation operation (FLOP), computational efficiency (FLOPs), memory access cost (MAC) and arithmetic intensity (ratio of FLOP to memory accesses cost). Since our model computed in GPU computation-unit and do convolution operation, FLOP is number of addition and multiply for floating point numbers. The computational efficiency is how many operation have been done per second. FLOP and FLOPs are two important factors to evaluate whether the model is efficient or not.

In addition to FLOP and FLOPs, the factor of arithmetic intensity will also effect the computational efficiency. The speed of data transfer between GPU memory and computation unit is much slower than that of computing on computation unit alone. Training convolution neural networks usually requires a lot of operations and a large number of memory access. High arithmetic intensity that can efficiently utilize hardware. Therefore, high arithmetic intensity is also one of the method that make the model efficient.

2.2 Baseline model

The architecture of base model we used for analysis is shown in the following figure.

2.3 Model analysis

The following figure shows the profile result of training a 3D UNet. From the figure (a) and figure (c), we can observe that 3D UNet first few layers and last few layers spend the most time on training due to their high FLOP. However, those layer are used to extract low features. It is inefficient to spend a lot of time extracting low features.

According to figure (b), current maximum computational efficiency is close to that of 1.6TFLOPs, but our experimental equipment: Nvidia TITAN X, can reach 11TFLOPs in theory. If each layer of the model can maximize the computational efficiency, we can speed up at least 10 times.

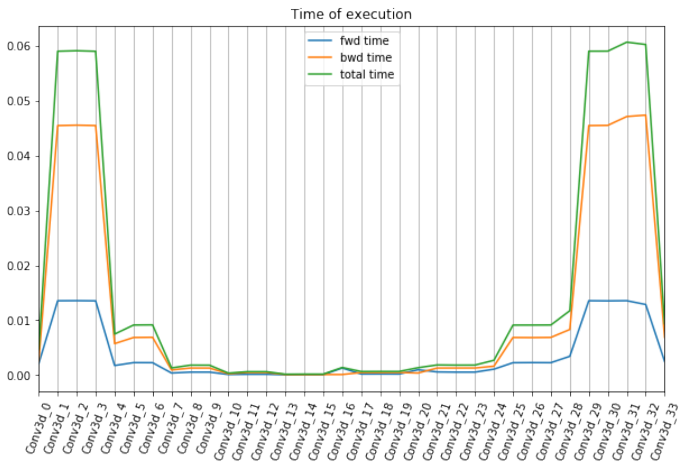
3 Computational Efficiency Optimization

Since computation time can be obtained by FLOP divided by computational efficiency, we optimize our model in two ways. One, reduce FLOP appropriately. Another, make schedule of computation and data transfer more efficient.

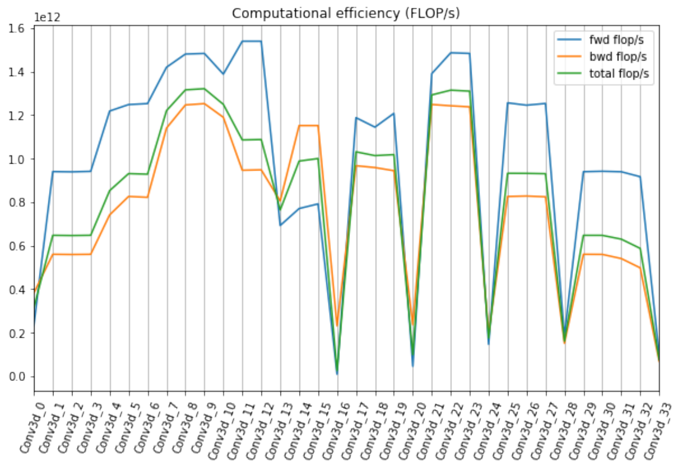
We reduce FLOP of each layer by reducing channel dimension because linear relationship between FLOP and channel dimension.

For optimize computation schedule, usually, cuDNN[?] is the first choice of most researchers. cuDNN is a handwritten low-level deep learning implement acceleration library, provided by NVIDIA for its GPUs. However it is closed source. MKL-DNN, a deep learning acceleration library was proposed by Intel for their CPU running deep learning applications. Both cuDNN and MKL-DNN are optimized for specialized hardware. It is time-consuming for manual hardware optimization for different device or deep learning operation without using excited deep learning acceleration libraries.

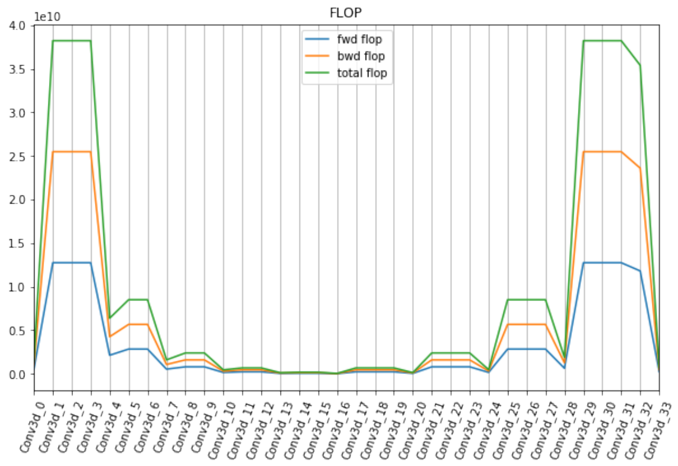
Halide[?], a programming language which main innvation is computation/schedule separation that allow us design different schedule for different devices. TVM[?] extend the idea of Halide, which not only separates computation from schedule, but also uses machine learning to search efficient schedule implements which can towards peak hardware performance in a large optimization search space (memory tiling, parallelization, loop transformations, vectorization, tensorization, etc). In this paper, we choose TVM as a low-level efficient code generator to optimize our model.



(a) Time of execution



(b) Computational efficiency (FLOP/s)



(c) Number of operation

Fig. 1: 3D Unet profile results

References

1. Author, F.: Article title. Journal **2**(5), 99–110 (2016)
2. Author, F., Author, S.: Title of a proceedings paper. In: Editor, F., Editor, S. (eds.) CONFERENCE 2016, LNCS, vol. 9999, pp. 1–13. Springer, Heidelberg (2016). https://doi.org/10.10007/1234567890
3. Author, F., Author, S., Author, T.: Book title. 2nd edn. Publisher, Location (1999)
4. Author, A.-B.: Contribution title. In: 9th International Proceedings on Proceedings, pp. 1–2. Publisher, Location (2010)
5. LNCS Homepage, http://www.springer.com/lncs. Last accessed 4 Oct 2017