Universiteit
Leiden
The Netherlands

On the optimization of imaging pipelines

Schoonhoven, R.A.

Citation
Schoonhoven, R. A. (2024, June 11). *On the optimization of imaging pipelines*. Retrieved from https://hdl.handle.net/1887/3762676

Version:	Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/3762676

Note: To cite this publication please use the final published version (if applicable).

7 Conclusion and outlook

The main goal of the research presented in this thesis was to develop approaches for optimizing computationally demanding imaging pipelines from several different angles. A subordinate goal was to try and support the research questions with practical problems from real-world applications and test the techniques on real data as much as possible. In this chapter, we summarize the contributions of this thesis, discuss its limitations, and suggest directions for future research.

7.1 Contributions and limitations

The contributions of Chapter 2 can be categorized as an attempt to practically realize the computationally challenging task of doing real-time reconstruction and segmentation, and how machine learning can help in this regard. Conceptually, design choices were made in the RECAST3D framework to perform quasi-3D reconstruction, and to use an omnidirectional CNN. On a software level, RECAST3D is created as a modular system with separate servers running visualization and reconstruction separately, and communication is done via data packets. This allows the system to be extended by different plugins that can asynchronously compute processing steps, e.g., filtering of the projection data can be done while earlier reconstructed slices are being annotated. On the hardware level, this modular system also has benefits since processing steps can be split among different GPUs, or even different machines to avoid blocking the pipeline. Practically, such a computational pipeline potentially represents an important step for online and real-time analysis of tomographic experiments. A limitation of the approach is related to the generalizability of the neural network to new data. For accurate segmentation, the training data needs to be representative of the data that will be segmented during the scanning procedure, meaning similar CT data needs to be available ahead of the experiment. Nevertheless, using such a pipeline, one can perform real-time

and online segmentation of quasi-3D volumes, enabling immediate feedback and analysis during experiments.

The main contribution of Chapter 3 is that the study outlines what benefits arise from connecting traditional CT approaches with modern auto-differentiation frameworks. While the convergence of a gradient descent-based optimization method is not guaranteed, the study shows that in practice the method still produces strong results. A large benefit comes from the engineering perspective, where the user can now trial different learned CT workflows and turn optimization on or off for selected parameters at will, without having to design complicated specialized algorithms. Connecting existing building blocks to an auto-differentiation framework means that users can experiment in a plug-and-play nature to solve common image processing tasks. Despite these benefits, several limitations of the approach emerged in the study. First, the current setup requires a large effort to implement existing algorithms in PyTorch, which inhibits adoption of the approach in practice. Second, the learning rate needs to be picked manually which is challenging when multiple parameters of different magnitude are involved.

The main contribution of Chapter 4 is a method that significantly prunes CNNs which does not require hyperparameters to be set. Furthermore, the graph representation allows for efficient pruning of redundant operations, which further reduces the size of the CNN. A drawback of the method thus far is that PyTorch implements pruning by masking the model parameters with binary tensors, thereby natively not producing any computational speedup. For the MS-D network, we implemented a pruned model that loads a sparse MS-D network to measure actual acceleration, but this is a time-consuming process if it has to be done for any CNN architecture.

Chapter 5 contributes practically usable methods to the GPU auto-tuning community that are implemented in the Kernel Tuner package. It further provides a ranking of black-box optimization methods for tuning GPU kernels, and the learnings from the study contributed to the methodology paper [250] that was set up by the GPU auto-tuning community to streamline research into optimization algorithms in the future. Secondly, the study contributes a difficulty quantification (and visualization) method for discrete search spaces based on graph theory. The application of this method need not be limited to GPU auto-tuning as it generalizes to any discrete search space. Computational limitations of the approach could come into play when the graph becomes too large to store in memory. Second, the entire search space needs to be traversed before the method can be applied. This means that it is currently only useful for post-optimization analysis, and not for steering optimization of unknown GPU kernels.

Finally, Chapter 6 provides a practical method to improve the energy efficiency of GPUs. With growing environmental concerns, methods that focus not only on maximizing compute performance but also take into consideration the environmental footprint of computations are of large importance. The simplicity of the method, i.e. a handful of executions of a small kernel that is shipped with the script, means that it can easily be applied on many different systems. Steering the GPU towards energy efficiency proved potentially highly effective as for the Tesla A100 GPU we

were able to increase its energy efficiency by 113.8% while losing only 12.0% in speed. This highlights that such a trade-off may be worthwhile on many systems if it is not crucial to extract maximal performance. A drawback of the current implementation is that it only works for NVidia server-grade GPUs. While the method was also designed to work for older models (those that do not support voltage measurements), it does not work for models where setting the core clock frequency is not allowed. It appears that this functionality has been disabled on consumer-grade GPUs, something we hope to see changed with future releases.

7.2 Outlook

Several promising research directions are unfolding to optimize computational imaging pipelines that we want to discuss in turn. A promising direction to further develop the idea of optimizing legacy pipelines with modern auto-differentiation tools is to develop infrastructure that approximates the gradients instead of computing them analytically. With this approach, the traditional building blocks are registered in the auto-differentiation framework by e.g. wrapping them in a Python class. Instead of computing gradients by traversing the computational graph, which requires expensive re-implementation of the existing algorithms, we use a method to approximate the gradients such as a finite differences method. In fact, instead of wrapping the legacy code and calling it from the auto-differentiation framework, a client-server model can be designed that separates the environments. In this manner, the auto-differentiation framework can work with external C or Fortran code or even an executable. Users would need to set up a lightweight server that can run on the legacy side and run the legacy code for a set of parameters (by for example passing them as CLI arguments, or writing them to a .txt file). Hopefully, such infrastructure would allow us to extend the learnings from Chapter 3 to a larger range of legacy pipelines, many of which cannot easily be reimplemented in PyTorch.

An alternative to approximating the gradient during pipeline optimization would be to approximate the forward model. If the forward model is prohibitively expensive to run repeatedly, it may be possible to approximate the forward model by a surrogate and optimize the parameters using the surrogate model instead. For example, in [118], a surrogate to TV reconstruction is introduced which could be used to tune the regularization parameter λ. Potentially, approaches could be developed that are suitable to approximate many computational building blocks. For example, neural networks are generic surrogate models that, if given the dimensions of the input and output space of a computational block, could be used to replace that block during pipeline optimization.

Another research direction is to investigate methods that can adaptively adjust gradient-based optimization methods for traditional algorithms. When parameters have significantly different magnitudes, it is challenging to find a combination of step sizes that makes the optimization method converge. Furthermore, approaches similar to batch normalization in deep learning could be used in between traditional

building blocks to normalize input and output data to avoid vanishing gradient problems. Next, in principle for N parameters, a single finite differences approximation requires $2N$ executions ($f(x+h)$ and $f(x-h)$). If an approximation of the gradient is used such as finite differences, approaches could be applied that limit the amount of executions necessary to run a finite differences method.

To improve the usefulness of neural network pruning, more development could be done on methods that create sparse models from the current PyTorch implementation with masked parameters. Potentially, the computational graph itself could be used to construct the pruning graph that LEAN uses, and the masked tensors could be used to remove branches of the computational graph altogether.

A major issue in auto-tuning research is the lack of proper algorithm comparisons. Most studies demonstrate their optimization algorithms on a specific dataset and only compare them to selected methods using different quality metrics. As mentioned, a proposed methodology by the auto-tuning community to improve reproducibility and transferability of results is currently in preparation [250]. Another interesting direction of research could be if methods can be developed that approximate the FFG on the fly, and use it as a heuristic to guide optimization.

As mentioned in Chapter 6, a major limitation to improving the energy efficiency of GPUs is that consumer-grade GPUs do not support core clock frequency tuning. In my view, vendors must expose functionality that can help with improving the energy efficiency of as many models as possible. Furthermore, we want to extend the power consumption model to work for other vendors. The model in principle is generically applicable, but code needs to be developed that can query the relevant measurements for non-NVidia GPUs. Another dimension of the GPU auto-tuning problem that was not explored in this thesis is the incorporation of memory energy efficiency. In addition to core clock frequency, GPUs have several memory clock frequencies which influence its performance and energy consumption. Thus far, there are usually only a few available memory clock frequencies, but if future GPUs allow for a larger range of memory clock frequencies, these would need to be incorporated into the power consumption model for further gains.