Lawrence Berkeley National Laboratory

Title

qlty: Handling large tensors in scientific imaging deep-learning workflows

Permalink

Author

Zwart, Petrus H

Publication Date

2024-09-01

DOI

Copyright Information

Peer reviewed

qlty: handling large tensors in scientific imaging deep-learning workflows

Petrus H. Zwart[a,b,c]

[a]*Center for Advanced Mathematics in Energy Research Applications, Lawrence Berkeley National Laboratory,*
[b]*Berkeley Synchrotron Infrared Structural Biology program, Lawrence Berkeley National Laboratory,*
[c]*Molecular Biophysics and Integrated Bioimaging Division, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, 94720, California, USA*

Abstract

In scientific imaging, deep learning has become a pivotal tool for image analytics. However, handling large volumetric datasets, which often exceed the memory capacity of standard GPUs, require special attention when subjected to deep learning efforts. This paper introduces `qlty`, a toolkit designed to address these challenges through tensor management techniques. `qlty` offers robust methods for subsampling, cleaning, and stitching of large-scale spatial data, enabling effective training and inference even in resource-limited environments.

Keywords: machine learning, deep learning, segmentation, denoising

1. Introduction

In computer vision applications within the experimental sciences, such as in the analysis of X-ray or Electron Tomography, Focused Ion Beam Scanning Electron Microscopy (FIB-SEM), or Adaptive Optics Lattice Light Sheet Microscopy (AO-LLSM) [1, 2, 3, 4, 5], the use of deep learning networks for im-

Current code version	0.2.0
Permanent link to code/repository used for this code version	`https://github.com/phzwart/qlty/releases/tag/v0.2.0`
Permanent link to reproducible capsule	`https://codeocean.com/capsule/c5aabf63-6d0b-4e4b-abbc-99abe85870c8/`
Legal code license	BSD License
Code versioning system used	git
Software code languages, tools and services used	Python, Jupyter Notebook
Compilation requirements, operating environments and dependencies	Python, pytorch, dask, einops
If available, link to developer documentation/manual	`https://qlty.readthedocs.io/`
Support email for questions	PHZwart@lbl.gov

Table 1: Code metadata

age processing and vision-related tasks has become increasingly important. A prime example of the use of deep learning techniques is in semantic segmentation [6, 7, 8, 9, 10], where pixel-level classifications in images are made so specific objects can be extracted, or properties of a sample can be derived [11, 12].

When these large datasets are inspected, visualized, or analyzed, they are done as a complete object. Manual annotation often results in labeling only part of an image, the full volume is still needed to contextualize the

information during the annotation process as smaller cropped sections can make proper labeling difficult. Similarly, when extracting downstream conclusions after segmentation, the spatial relationships within the full object need to remain intact. While libraries for handling large [13] and very large [14] data volumes are available, these methods are not directly geared towards enabling deep learning, necessitating the development of specialized tools and techniques for handling, processing, and segmenting large datasets within deep learning workflows. During the training and inference of a deep learning workflow, the associated memory requirements for large datasets rapidly outruns the available capacity conventional hardware has to offer, pushing the need for efficient data handling and processing techniques.

Out-of-core processing methods have emerged as a crucial approach to handling datasets that exceed available memory, offering a practical solution to the challenges posed by large volumetric data in scientific computer vision [14]. These techniques typically involve data chunking, where the dataset is divided into manageable portions that can be loaded and processed sequentially, allowing for the analysis of data that far exceeds the capacity of RAM or GPU memory. While effective in many data processing scenarios, existing out-of-core methods often lack the specific functionality required for complex computer vision tasks, particularly those involving deep learning models. A key requirement for a tool to aid in these scenarios is the ability to handle overlapping subtensors, both during the chunking process and when stitching processed results back together. This capability is crucial for maintaining spatial continuity and addressing edge effects in convolutional neural networks. The absence of a light-weight tool that can efficiently man-

age patching, manipulation and stitching, while also integrating seamlessly with PyTorch's tensor operations for computer vision applications, drove the development to fill this gap.

Here we present `qlty`, a lightweight python toolkit that addresses these challenges through a suite of specialized functions. By providing robust methods for tensor patching, manipulation, and stitching, `qlty` enables researchers and developers to work effectively with large-scale spatial data, even in resource-constrained environments. We will illustrate `qlty`'s functionality and demonstrate its effectiveness through a segmentation of experimental data, illustrating how `qlty` can enable the processing of larger datasets that would otherwise be challenging on limited hardware.

2. Purpose and Functionality

`qlty` is primarily aimed at providing subsampling and out-of-core tools for segmentation, denoising and related techniques in scientific imaging. A high level overview of `qlty` within these types of workflows is depicted in Figure 1

2.1. Subsampling

Assume we have a PyTorch tensor $\mathcal{T}$ with shape (N, C, Y, X). This data structure contains N items of C channel data of a two-dimensional spatial map with lengths Y and X. When subsampling $\mathcal{T}$, the aim is to obtain tensor $\mathcal{S}$ of size (M, C, Y_w, X_w), where (Y_w, X_w) is referred to as the *window size*. The channel count C remains untouched, but the resulting batch size M will typically be not equal to N. Within `qlty`, subsampling is done in a deterministic fashion by translating a window with a predefined step size
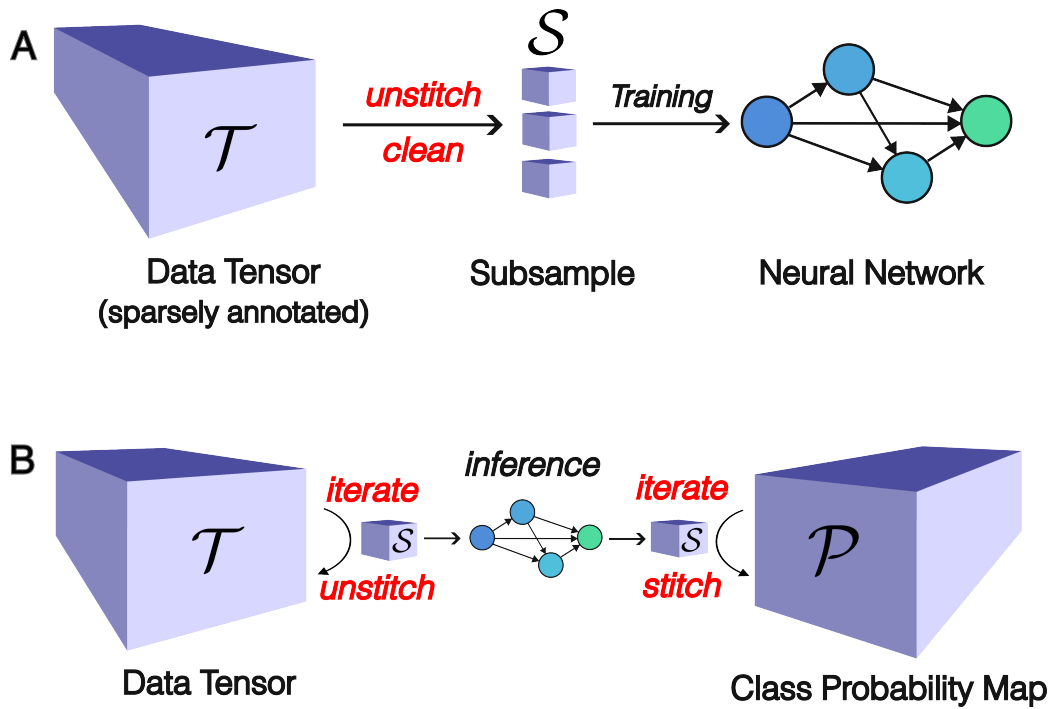
4

Figure 1: High level overview of `qlty`'s functionality, with specific methods depicted in red. A - For generating training data with a smaller spatial footprint, `qlty` subsamples a sparsely annotated tensor $\mathcal{T}$ into smaller chunks $\mathcal{S}$ while discarding parts of $\mathcal{T}$ for which no training data is available. B - In inference scenarios, `qlty` iterates over data tensor $\mathcal{T}$ yielding manageable chunks of data that can be subjected to a neural network of choice. The resulting output is iteratively placed back into a tensor of the right size.

across the input tensor, while deploying an step-back correction at the edges of tensor $\mathcal{T}$ to ensure that the last subsampled window has edges aligned with the input tensor. This process is outlined for a multi-channel 2D tensor in Algorithm 1 and extends analogously to 3D scenarios. At present, 2D + time or 3D + time scenarios are not supported, but will be added in the future.

Algorithm 1 Subsampling Tensor $\mathcal{T}$

Require: $\mathcal{T}$ of shape (N, C, Y, X), window size (Y_w, X_w), step size $(\text{step}_y, \text{step}_x)$

Ensure: Tensor with subsampled chunks $\mathcal{S}$

  Initialize an empty list *subsamples*

  for $n = 0$ to $N - 1$ **do**

    for $y = 0$ to $Y - Y_w$ with step step_y **do**

      for $x = 0$ to $X - X_w$ with step step_x **do**

        $y_{\text{start}} = min(y, Y - \text{step}_y)$

        $y_{\text{stop}} = y_{\text{start}} + \text{step}_y$

        $x_{\text{start}} = min(x, X - \text{step}_x)$

        $x_{\text{stop}} = x_{\text{start}} + \text{step}_x$

        $\text{slice}_y = slice(y_{\text{start}}, y_{\text{stop}}, 1)$

        $\text{slice}_x = slice(x_{\text{start}}, x_{\text{stop}}, 1)$

        $\text{chunk} = \mathcal{T}[n : n + 1, :, \text{slice}_y, \text{slice}_x]$

        Append chunk to *subsamples*

      end for

    end for

  end for

  $\mathcal{S} = \text{concatenate}(\textit{subsamples}, \dim = 0)$

  return $\mathcal{S}$

2.2. Augmentation & Cleaning

When tensors are subsampled as described, a specific pixel or voxel from the original tensor can appear in multiple subsamples. Due to the finite size of these subsampled tensors and the receptive field of the neural network, the equivalence of identical observations across subsamples is disrupted. This issue is most evident for pixels at the edges of the subsampled tensors, which, at best, only have half the number of unique neighbors within their vicinity during convolutional calculations. When working with semantic segmentation data, a border region in the subsampled tensor can be defined, which allows us to mark data as *unobserved* and exclude these labels from the training dataset, only retaining internal voxels or pixels.

While the above procedure introduces a sparsification of the training data by excluding edge pixels or edge voxels, it is far more common that the sparsity of labeled data originates from selected human labeling. In such cases, a large amount of subsampled tensors can contain no training data due to absence of labels. `qlty` provides tools to remove or exclude such tensor pairs, enriching the information content of the data, while ensuring it runs on the available hardware.

2.3. Stitching

Even though the memory requirements for inference are less than those for training, subsampling can still be necessary. In `qlty`, the same subsampling algorithm to generate data for inference is used as in the training scenario, without the cleaning step. In addition, accumulator arrays are created, a mean array and a normalization array, where the inference results of subsampled tensors are stored. In the mean array, inference results are

7

multiplied by a weight term that accounts for the relative significance of predictions for border pixels and added to the accumulator. The normalization array accumulates the sum of these weights. The final prediction is obtained by calculating the weighted mean, achieved by dividing the mean array by the normalization array. Both the mean and accumulator arrays are *zarr* arrays, caching the results on disc. The final normalization is performed using *dask*, making use of multiprocessing options when available.

3. Example Usage

The use of `qlty` in a segmentation setting is illustrated by a segmentation of a 3D cryo-electron tomographic reconstruction of the cell derived matrix (CDM) produced by human telomerase immortalized foreskin fibroblasts (TIFFS) [15], publicly available via EMPIAR [16]. For illustrative purposes, a minimal manual annotation was conducted using napari [17], where 52k pixels - the equivalent of 5 data cubes of length 64 - were labeled in a few slices across the XY,YZ and XZ projections. `qlty` was run using typical settings, with a target window of $(64, 64, 64)$. Due to the low number of labeled pixels, a step size of 32 in all directions was used, resulting in duplicating the number of annotated pixels by a factor of 7.46, but reducing the total data volume to analyze by a factor of 9 by a subsequent elimination of subtensors without annotated data. Additional details are found in Table 3.

The generated training data was used to train an ensemble of eight Sparse Mixed Scale neural Networks (SMSNets) with random architecture as available from the *DLSIA* library [18]. Using Adam as an optimizer and a cross-entropy target with a learning rate of 0.01 and batch size of 16, fifty epochs

8

Training Scenario Settings	
Full Tensor shape	$(1, 1, 236, 720, 510)$
Annotated *Non Filament* voxels	51505
Annotated *Filament* voxels	1809
Window shape	$(64, 64, 64)$
Step Size	32 in Z,Y,X
Border Size	0 in Z,Y,X
Total subsample	$(2310, 1, 64, 64, 64)$
Clean subsample	$(256, 1, 64, 64, 64)$
Data duplication rate	7.47

Table 2: `qlty` training settings. Sparse labeling of a tensor, approximately 0.06% of voxels were annotated , resulted in 256 cube-like tensors of length 64. The translating window approach results in annotated data duplication - identical voxels appearing in multiple subsamples - at a rate of 7.47.

on an A100 GPU yielded an average macro F1 score - taking into account the class imbalance - of 89% and 88% for the training and validation data respectively, across all 8 models.

These trained networks were used to run inference across the full data volume. For inference, a larger window size is technically feasible because of lower memory requirements. Using a window size of 128, a step size of 108 and a border width of 10 were used. By setting the border weight to zero, the resulting stitching operation results in patching together non-overlapping tensor of size $(108, 108, 108)$. In this manner, 231 subtensors were generated and run through an inference pass. Further `qlty` setting are found in Table 3.

Inference Scenario Settings	
Input tensor shape	$(1, 1, 236, 720, 510)$
Window shape	$(128, 128, 128)$
Step size	108 in Z,Y,X
Border size	10 in Z,Y,X
Border weight	0 in Z,Y,X
Subsampled tensor shape	$(231, 1, 128, 128, 128)$
Output tensor shape	$(1, 2, 236,720,510)$

Table 3: `qlty` inference settings. Using the ensemble of trained network, inference was performed on subsampled tensors of with spatial footprint of $(128, 128, 128)$, with a step size of 108 and a border size of 10. After stitching the subtensors, a tensor of shape (1,2,236,720,510) was obtained.

The results of the inference are shown in Figure 2. The high quality of the depicted segmentation originates in large part from the data augmentation by the windowed subsampling of the tensor and the associated data duplication that follows from this procedure.

4. Impact

The `qlty` toolkit offers a light-weight solution for handling large-scale (volumetric) data for computer vision tasks in scientific imaging. By addressing the challenges associated with processing large datasets that exceed the memory capacity of typical GPUs, `qlty` enables efficient and effective image analysis in resource-constrained environments. This capability is crucial for a variety of scientific imaging applications in biology and material sciences.

`qlty` has been put to practical use in a number of applications, such as the segmentation of contaminant peaks in X-ray powder diffraction [19], the segmentation of fibers from X-ray tomography scans of reinforced concrete [18] and near-real time segmentation of X-ray tomography datasets at large scale facilities [20]. In a denoising setting, `qlty` has allowed for routine training and inference of X-ray tomographic data exceeding 10^9 pixels on a RTX-3090 NVIDIA consumer hardware GPU [21]. Training and inference on this type of data without chunking would require at least an order of magnitude more memory than the 24GB that GPU possesses, making `qlty` a indispensable tool in these workflows.

The practical impact of `qlty` as observed in the aforementioned applications [18, 19, 20], lies in its integration within standard workflows, allowing users or automated processes to adjust window and step sizes to easily manage GPU memory limitations. It is commonly embedded as a preprocessing step in segmentation or denoising pipelines, simplifying the process of running these workflows across different hardware architectures, such as RTX 3090s or A100 GPUs. While memory constraints are often addressed by adjusting batch sizes, breaking down tensors into smaller subtensors to fit within these limitations is typically more complex — `qlty` effectively streamlines this process.

The seamless integration of `qlty` within the PyTorch framework, streamlines machine learning workflows for scientific imaging. `qlty`'s subsampling and stitching functions can be easily incorporated within existing data analytics pipelines [18], circumventing the need to write *ad-hoc* functionality that addresses these issues. `qlty` thus reduces a barrier to utilize advanced

image processing techniques in the broader scientific community.

5. Future Directions

By working closely with experimental scientists, a number of future development directions have been identified.

Supporting Fragmented Workflows: In many applications, the locations where data is generated, computation is performed, and where results are visualized are often different and distributed across various network locations. A promising future direction for `qlty` is to integrate it with technologies that allow seamless data access across these disparate networked environments. By combining `qlty` with distributed computing frameworks and remote data access protocols, we can facilitate efficient data handling and computation even in workflows fragmented across many locations, ensuring smooth operations regardless of where the data resides.

Supporting Image Pyramids: Image pyramids are powerful data structures for managing large datasets across different scales. Neural networks that leverage these pyramids can effectively decouple receptive field requirements across scale spaces, potentially leading to more efficient models. A key future direction is to extend `qlty` to independently partition large tensors within these scale spaces. By enabling the processing of image data across various scales, and combining the resulting outputs only at the end, `qlty` can help create leaner and more effective neural networks for large-scale imaging tasks.

Tracking Positional Embeddings: Aligning large images across modalities poses significant challenges, and the use of positional information is crucial

in such tasks. Currently, tracking positional embeddings across subtensors can be achieved by creating a dummy tensor with positional data. However, a more efficient approach would be to enhance `qlty` by attaching origin and slicing parameters to each chunk. These parameters could then be used to maintain positional context and enable tensor-index to location parameters in a physical model throughout the processing pipeline, improving the accuracy and effectiveness of alignment operations.

6. Acknowledgements

sibility for the content of the publication.

References

[1] Mark A Le Gros, Gerry McDermott, and Carolyn A Larabell. X-ray tomography of whole cells. *Curr. Opin. Struct. Biol.*, 15(5):593–600, October 2005.

[2] Vladan Lucić, Friedrich Förster, and Wolfgang Baumeister. Structural studies by electron tomography: from cells to molecules. *Annu. Rev. Biochem.*, 74:833–865, 2005.

[3] C Shan Xu, Kenneth J Hayworth, Zhiyuan Lu, Patricia Grob, Ahmed M Hassan, José G García-Cerdán, Krishna K Niyogi, Eva Nogales, Richard J Weinberg, and Harald F Hess. Enhanced FIB-SEM systems for large-volume 3D imaging. *Elife*, 6, May 2017.

[4] Güneş Parlakgül, Ana Paula Arruda, Song Pang, Erika Cagampan, Nina Min, Ekin Güney, Grace Yankun Lee, Karen Inouye, Harald F Hess, C Shan Xu, and Gökhan S Hotamışlıgil. Regulation of liver subcellular architecture controls metabolic homeostasis. *Nature*, 603(7902):736–742, March 2022.

[5] Tsung-Li Liu, Srigokul Upadhyayula, Daniel E Milkie, Ved Singh, Kai Wang, Ian A Swinburne, Kishore R Mosaliganti, Zach M Collins, Tom W Hiscock, Jamien Shea, Abraham Q Kohrman, Taylor N Medwig, Daphne Dambournet, Ryan Forster, Brian Cunniff, Yuan Ruan, Hanako Yashiro, Steffen Scholpp, Elliot M Meyerowitz, Dirk Hockemeyer, David G Drubin, Benjamin L Martin, David Q Matus, Minoru Koyama, Sean G

Megason, Tom Kirchhausen, and Eric Betzig. Observing the cell in its native state: Imaging subcellular dynamics in multicellular organisms. *Science*, 360(6386), April 2018.

[6] Gabriela Csurka, Riccardo Volpi, and Boris Chidlovskii. 2022.

[7] Daniël M. Pelt and James A. Sethian. A mixed-scale dense convolutional neural network for image analysis. *Proceedings of the National Academy of Sciences*, 115(2):254–259, 2018.

[8] Uwe Schmidt, Martin Weigert, Coleman Broaddus, and Gene Myers. Cell detection with star-convex polygons. *International Conference on Medical Image Computing and Computer-Assisted Intervention (MIC-CAI)*, 11071:265–273, 2018.

[9] C. Stringer, T. Wang, M. Michaelos, and M. Pachitariu. Cellpose: a generalist algorithm for cellular segmentation. *Nature Methods*, 18:100–106, 2021.

[10] Nicholas Lubbers, Turab Lookman, and Kipton Barros. Inferring low-dimensional microstructure representations using convolutional neural networks. *Physical Review E*, 96(5):052111, 2017.

[11] Joseph C Ferguson, Francesco Panerai, Arnaud Borner, and Nagi N Mansour. Puma: The porous microstructure analysis software. *SoftwareX*, 7:81–87, 2018.

[12] Joseph C Ferguson, Federico Semeraro, John M Thornton, Francesco Panerai, Arnaud Borner, and Nagi N Mansour. Update 3.0 to "puma:

The porous microstructure analysis software",(pii: S2352711018300281). *SoftwareX*, 15:100775, 2021.

[13] Jason Moore, C. Allan, Jean-Marie Burel, Melissa Linkert, Simon Loynton, David Macdonald, William J. Moore, Catarina Neves, Andrew Patterson, Michael Porter, Agata Tarkowska, Benoit Loranger, Julien Avondo, Inigo Lagerstedt, Luke Lianas, Stephen Leo, Sophie Hands, Ronald T. Hay, Alok Patwardhan, and Christopher Best. Ome-ngff: an interoperable bioimaging data storage format. *Nature Methods*, 18:153–155, 2021.

[14] Xiongtao Ruan, Matthew Mueller, Gaoxiang Liu, Frederik Görlitz, Tian-Ming Fu, Daniel E. Milkie, Joshua L. Lillvis, Alexander Kuhn, Chu Yi Aaron Herr, Wilmene Hercule, Marc Nienhaus, Alison N. Killilea, Eric Betzig, and Srigokul Upadhyayula. Image processing tools for petabyte-scale light sheet microscopy data. *bioRxiv*, 2024.

[15] Bettina Zens, Florian Fäßler, Jesse M Hansen, Robert Hauschild, Julia Datler, Victor-Valentin Hodirnau, Vanessa Zheden, Jonna Alanko, Michael Sixt, and Florian K M Schur. Lift-out cryo-FIBSEM and cryo-ET reveal the ultrastructural landscape of extracellular matrix. *J. Cell Biol.*, 223(6), June 2024.

[16] Andrii Iudin, Paul K Korir, Sriram Somasundharam, Simone Weyand, Cesare Cattavitello, Neli Fonseca, Osman Salih, Gerard J Kleywegt, and Ardan Patwardhan. EMPIAR: the electron microscopy public image archive. *Nucleic Acids Res.*, 51(D1):D1503–D1511, January 2023.

[17] napari contributors. napari: a multi-dimensional image viewer for python, 2019.

[18] Eric J. Roberts, Tanny Chavez, Alexander Hexemer, and Petrus H. Zwart. DLSIA: Deep Learning for Scientific Image Analysis. *Journal of Applied Crystallography*, 57(2):392–402, Apr 2024.

[19] Howard Yanxon, Eric Roberts, Hannah Parraga, James Weng, Wenqian Xu, Uta Ruett, Alexander Hexemer, Petrus Zwart, and Nickolas Schwarz. Image segmentation using u-net architecture for powder x-ray diffraction images, 2023.

[20] Advanced Light Source News. Als computing group brings machine learning models to beamtimes around the world, July 2024. Accessed: 2024-07-05.

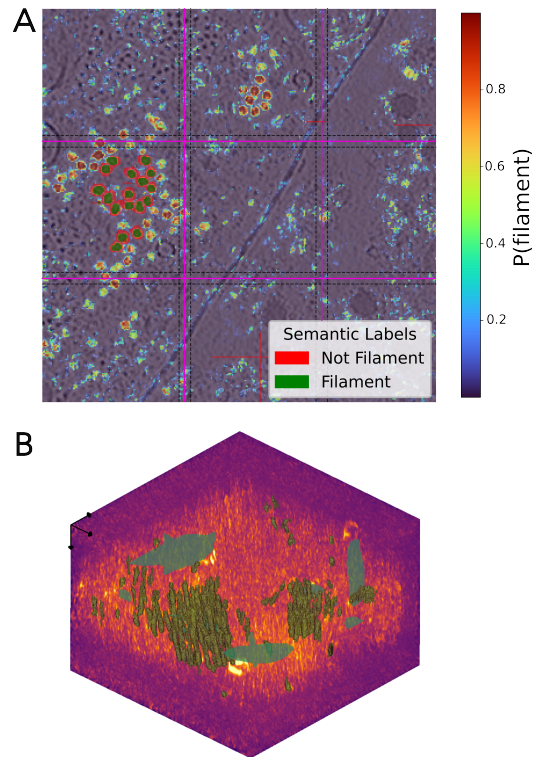[21] Odeta Qafoku, Tamas Varga, and Petrus H. Zwart. Covered affairs: denoising with confidence. In preparation.

Figure 2: Segmentation results. A - A 2D section of the resulting segmentation of the tomographic data (grey) is overlaid with the class probability map for the intermediary filaments (multi-color heatmap) and part of the hand annotated voxels (green, red). Boundaries of the subsampled tensor are shown as black doted lines, while the purple solid line indicates the location of the seam where two adjacent tensors meet when taking into account the border weighting procedure. B - A 3D overview of the tomographic density (magma heatmap), hand-labeled voxels (light-green 2D patches) and resulting class labels in 3D for the filaments from the probability map after a simple connected component sizing filter (bright green).