

Compresso: Efficient Compression of Segmentation Data For Connectomics

Brian Matejek, Daniel Haehn, Fritz Lekschas, Michael Mitzenmacher,
Hanspeter Pfister

Harvard University, Cambridge, MA 02138, USA

`bmatejek,haehn,lekschas,michaelm,pfister@seas.harvard.edu`

Abstract. Recent advances in segmentation methods for connectomics and biomedical imaging produce very large datasets with labels that assign object classes to image pixels. The resulting label volumes are bigger than the raw image data and need compression for efficient storage and transfer. General-purpose compression methods are less effective because the label data consists of large low-frequency regions with structured boundaries unlike natural image data. We present Compresso, a new compression scheme for label data that outperforms existing approaches by using a sliding window to exploit redundancy across border regions in 2D and 3D. We compare our method to existing compression schemes and provide a detailed evaluation on eleven biomedical and image segmentation datasets. Our method provides a factor of 600-2200x compression for label volumes, with running times suitable for practice.

Keywords: compression, encoding, segmentation, connectomics

1 Introduction

Connectomics—reconstructing the wiring diagram of a mammalian brain at nanometer resolution—results in datasets at the scale of petabytes [21,8]. Machine learning methods find cell membranes and create cell body labelings for every neuron [18,12,14] (Fig. 1). These segmentations are stored as label volumes that are typically encoded in 32 bits or 64 bits per voxel to support labeling of millions of different nerve cells (neurons). Storing such data is expensive and transferring the data is slow. To cut costs and delays, we need compression methods to reduce data sizes.

The literature currently lacks efficient compression of label volumes. General-purpose compression schemes [3,24,11,15,23,19,16,22,6,2] are not optimized for this data. In this paper, we exploit the typical characteristics of label volumes such as large invariant regions without natural relationship between label values. These properties render 2D image compression schemes inadequate since they rely on frequency reduction (using e.g., wavelet or discrete cosine transform) and value prediction of pixels based on local context (differential pulse-code modulation) [17,20]. Color space optimization strategies in video codecs [1] also have no effect on label volumes, even though the spatial properties of a segmentation stack (z -axis) are similar to the temporal properties of video data (time-axis). A compression scheme designed specifically for label volumes is part of

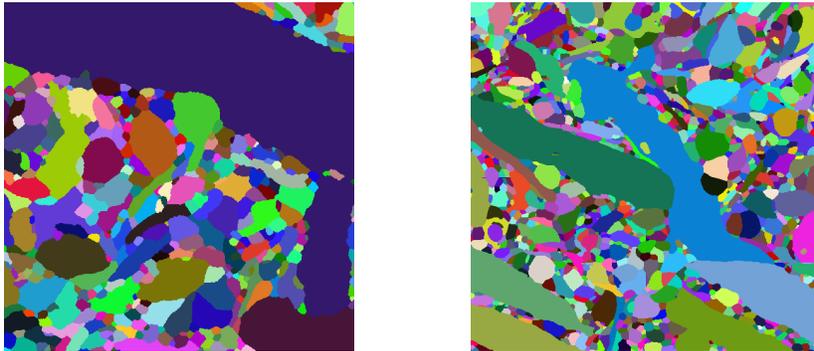


Fig. 1. Examples of connectomics segmentation data with a different color per cell.

the visualization software Neuroglancer [7]. This method exploits segmentation homogeneity by creating small blocks with N labels and reducing local entropy to $\log_2 N$ per pixel. Lookup tables then decode the values $[0, N)$ to the original 64-bit labels. We compare the Neuroglancer scheme with our method.

We explore the lossless compression of gigavoxel neuron segmentation volumes with high bit-encodings. We study and evaluate the performance of existing lossless compression methods, and their combinations, on multiple connectomics, magnetic resonance imaging (MRI) and general segmentation datasets. As our main contribution, we present Compresso—a novel compression method designed for label volumes using windowed feature extraction. Compresso yields compression ratios on label volumes 80% higher than the current best tools (Sec. 3). We release an open-source C++ implementation of our method including a Python interface.

2 The Compresso Scheme

2.1 Encoding

Overview. Segmentation datasets contain two important pieces of information across the image stack: per-segment shape and per-pixel label. Decoupling these two components allows for better compression on each.

Boundary Encoding. To encode the segment shapes, we consider the boundary pixels between two segments. Removing the per-pixel labels, we produce a boundary map for each slice where a pixel (x, y, z) is 1 if either pixel at $(x+1, y, z)$ or $(x, y+1, z)$ belongs to a different segment. The boundary map is divided into non-overlapping congruent 3D windows. If there are n pixels per window, each window w is assigned an integer $V_w \in [0, 2^n)$ where V_w is defined as:

$$V_w = \sum_{i=0}^{n-1} \mathbb{I}(i)2^i, \quad (1)$$

and $\mathbb{I}(i)$ is 1 if pixel i is on a boundary and 0 otherwise. Figure 2 shows an example segmentation with a window size of $4 \times 4 \times 1$.

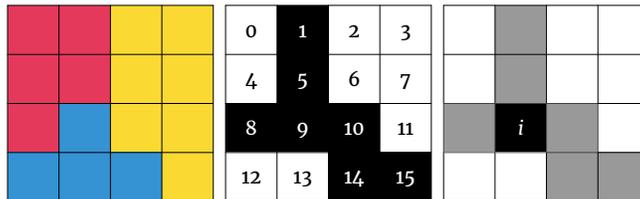


Fig. 2. A $4 \times 4 \times 1$ pixel window where three unique labels meet (left). The boundary map for the same window, where dark pixels represent the boundary (center). This window has an encoded value of 50,978 ($2^1 + 2^5 + 2^8 + 2^9 + 2^{10} + 2^{14} + 2^{15}$). A boundary pixel i that is indeterminate and requires additional decoding information (right).

A priori, each window could take any of 2^n distinct values, and therefore require n bits to encode without further manipulation. However, boundaries in segmentation images are not random, and many of these values never appear. Indeed, we find that a small subset of high-frequency V_w values accounts for most windows, allowing for significant compression. Figure 3 shows the 100 most common windows for a representative connectomics dataset. These 100 frequently occurring windows account for approximately 82% of the over 1.2 million V_w values in this dataset. Nearly all of these windows correspond to simple lines traversing through the window. For contrast, we also provide 5 randomly generated windows that never occur in the dataset.

We define N as the number of distinct V_w representing all of the windows in an image stack. We construct an invertible function $f(V_w) \rightarrow [0, N)$ to transform the window values into a smaller set of integers. For all real-world segmentations $N \ll 2^n$; however, we assume no constraint on N in order to guarantee lossless compression. With this function, each V_w requires $\log_2 N$ bits of information to encode. This is fewer than the initial number of bits so long as $N \leq 2^{n-1}$. We create two arrays that store the per-segment shape encoding: `WindowValues[]` contains the value $f(V_w)$ for every window w and `ValueMapping[]` contains the reverse mapping from $[0, N) \rightarrow [0, 2^n)$ based on the function f . Long sequences of 0s in `WindowValues[]` are reduced using run-length encoding.

Per-Pixel Label Compression. So far we have focused exclusively on transforming the boundary map of an image segmentation. However, the per-pixel labels themselves are equally important. The boundary map divides each image slice into different segments. By design, all pixels in the same segment have the same label so we store only one label per segment for each slice. We use a connected-component labeling algorithm to store one label per segment [9]. The algorithm labels all pixels clustered within a component m from M section labels. We store

the original label for a segment m in slice z in `Labelsz[m]`. We concatenate these arrays for every image slice to create a variable `Labels[]`.

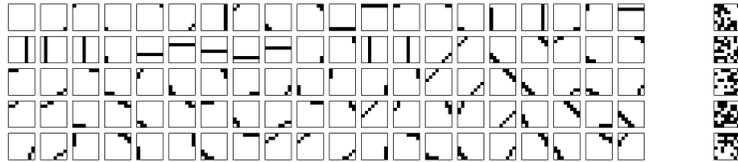


Fig. 3. The 100 most frequent windows accounting for approximately 82% of the over 1.2 million V_w values on a representative connectomics dataset contrasted with 5 randomly generated windows. Each box represents an $8 \times 8 \times 1$ window where black pixels are boundary and white pixels are non-boundary.

Exceptions. Thus far, we have assumed the boundaries described in Section 2.1 provide enough information to reconstruct the entire segmentation. Pixels not on a segment boundary are easily relabeled using the `Labels[]` array. However, more care is needed for pixels on the segment boundaries. Consider Figure 2, which depicts a difficult boundary to decode. If a boundary pixel has a non-boundary neighbor to the left or above, then that pixel merely takes on the value of that neighbor. However, the pixel i requires more care since its relevant neighbors are both boundary pixels. If a non-boundary neighbor pixel shares a label with the undetermined pixel, we add the offset to that neighbor to an array `IndeterminateValues[]`. Otherwise we add that per-pixel label.

Metadata. We construct a data structure containing the two per-segment shape and two per-pixel label arrays. The last component of the data structure is the `Header`, which contains the dimensions of the original data, the window size, and the size of the arrays. Compresso could be improved by further compressing the individual components of the encoding (e.g., Huffman encoding the V_w values). We achieve strong overall compression by using a second-stage general compression scheme such as LZMA (Sec. 3).

2.2 Decoding

The first step in decoding the data is to reconstruct the boundary map. We iterate over every pixel, determine the corresponding window w , and retrieve the encoded window value $f(V_w)$ from the `WindowValues[]` array. These values range from 0 to $N - 1$ and correspond to an index in `ValueMapping[]` that contains the original V_w value. After decoding V_w , the value of pixel i in window w equals $V_w \wedge 2^i$.

After reproducing the boundary map, we execute the same deterministic connected-components algorithm per slice as when encoding. Each component

in the boundary map receives a label between 0 and $M - 1$. Using the `Labels []` array, we can easily translate these component labels into the original per-pixel labels for every slice. To determine the per-pixel labels for every boundary pixel, we iterate over the entire dataset in raster order. Any boundary pixel (x, y, z) with a non-boundary neighbor at $(x - 1, y, z)$ or $(x, y - 1, z)$ shares the same per-pixel label. If both relevant neighbors are boundaries we consider the next unused value in the `IndeterminateValues []` array and update this pixel’s label.

2.3 Complexity

In what follows, P is the number of input pixels; N is the number of distinct window values; X , Y and Z are the size of the x , y , and z dimensions of the input data; and α is the inverse Ackermann function [5].

Encoding. Extracting the boundaries from the segmentation, generating the V_w values, and populating the `IndeterminateValues []` array are all linear work in P . The N unique window values are sorted to create the `ValueMapping` variable. Generating the `Labels []` array requires running a connected-component labeling algorithm over each z slice; we use a union-find data structure with union by rank and path compression optimizations. The overall complexity of the compression scheme is therefore $O(P(1 + \alpha(XY)) + N \log N)$.

Decoding. Decoding the window values, reconstructing the boundary map, and applying the correct per-pixel labels for all boundary pixels using the array `IndeterminateValues []` are all linear work in P . Reconstructing the per-pixel labels requires running the connected-component labeling algorithm over every image slice. The overall complexity of the decompression scheme is therefore $O(P(1 + \alpha(XY)))$.

3 Evaluation and Results

We consider the following compression schemes: Compresso, Neuroglancer, Brotli, BZip2, Zlib, LZ78, LZF, LZMA, LZO, LZW, Zopfli, Zstandard, PNG, JPEG2000, and X.264. In addition to these stand-alone compression schemes we consider all pairs with a first stage encoding using either Compresso or Neuroglancer and a second stage using one of the general-purpose algorithms. Both Compresso and Neuroglancer leave some redundancies that a general-purpose compressor can easily reduce; such multi-stage schemes are common in image compression. Table 1 presents six connectomics, three MRI, and two image segmentation datasets used for evaluation. Compresso works for any arbitrary 2-D and 3-D window dimensions. We achieve the results in this section using an 8x8x1 window.

The combination of Compresso and LZMA provides superior compression on all connectomics datasets (Tab. 1). Figure 4 shows the compression ratios for every compressor on segmentation data. For example, Compresso achieves a compression ratio of over 950x on *L.Cylinder* reducing the 10 gigabyte volume to

Table 1. For evaluation, we use the following publicly available datasets. Segmentations were obtained using a combination of U-net [18] and watershed, semi-automatic, or manually. Compresso paired with LZMA yields the best compression ratio on all datasets indicated by an asterisk (*). Neuroglancer paired with LZMA achieved the best compression ratio only for the SPL Brain Atlas (724x).

Dataset	Size	Segmentation	Compresso + LZMA	
			Speed (Com./Dec.)	Compression Ratio
<i>AC3 Subvolume</i> ¹ mouse cortex, EM	1024 × 1024 × 150 vx (6 × 6 × 30 nm ³ /vx)	U-net	100 / 209 MB/s	814x *
<i>AC4 Subvolume</i> mouse cortex, EM	1024 × 1024 × 100 vx (6 × 6 × 30 nm ³ /vx)	U-net	105 / 218 MB/s	701x *
<i>L. Cylinder</i> ² [10] mouse cortex, EM	2048 × 2048 × 300 vx (3 × 3 × 30 nm ³ /vx)	U-net	103 / 180 MB/s	952x *
<i>CREMI A, B, C</i> ³ drosophila brain, EM	1250 × 1250 × 125 vx (4 × 4 × 40 nm ³ /vx)	U-net	110 / 218, 118 / 243, 110 / 219 MB/s	857x *, 1239x * 960x *
<i>SPL Brain Atlas</i> ⁴ T1/T2-weighted MRIs	256 × 256 × 256 vx (1 × 1 × 1 mm ³ /vx)	Semi-autom.	85 / 254 MB/s	636x
<i>SPL Knee Atlas</i> ⁵ MRI	512 × 512 × 119 vx (0.277 × 0.277 × 1 mm ³ /vx)	Semi-autom.	136 / 244 MB/s	1553x *
<i>SPL Abdominal Atlas</i> ⁶ CT	256 × 256 × 113 vx (0.9375 × 0.9375 × 1.5 mm ³ /vx)	Semi-autom.	91 / 254 MB/s	480x *
<i>BSD500</i> ⁷ Segmentation Challenge	321 × 481, 2696 images	Manual	110 / 187 MB/s	1188x *
<i>PASCAL VOC</i> ⁸ 2012 Challenge	Varying, 2913 images	Manual	146 / 222 MB/s	2217x *

10.5 megabytes. LZMA performs very well by itself and paired with any encoding strategy. X.264 performs surprisingly poorly on these datasets, in part because of our requirement of lossless compression. It performs better when information loss is tolerated, however, even then it does not surpass the more specialized encoding schemes. These observations also hold for JPEG2000 and PNG. Compresso with LZMA outperforms all other existing methods on connectomics datasets by 80%.

The fundamental principles guiding Compresso are valid for a diverse set of segmentation datasets (Fig. 4, right). We evaluate the performance of our compression scheme on three MRI and two image segmentation datasets to demonstrate additional potential use cases. Compresso followed by LZMA compresses the MRI datasets reasonably well, particularly on the SPL Knee Atlas which contains highly redundant boundary segments. The Berkeley Segmentation and PASCAL Visual Object Class datasets are two very common benchmarks in image segmentation [13,4]. Currently these datasets use GZIP and PNG compression but Compresso with LZMA can improve on them by a factor of over 10x and 5x respectively.

¹ AC3+AC4 Subvolumes: <http://openconnecto.me/catmaid/?dataview=13>

² L. Cylinder: <https://software.rc.fas.harvard.edu/lichtman/vast/>

³ CREMI A+B+C: <http://www.cremi.org>

⁴ SPL Brain Atlas: <http://www.spl.harvard.edu/publications/item/view/2037>

⁵ SPL Knee Atlas: <http://www.spl.harvard.edu/publications/item/view/1953>

⁶ SPL Abdominal Atlas: <http://www.spl.harvard.edu/publications/item/view/1918>

⁷ BSD500: <https://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html>

⁸ VOC2012: <http://host.robots.ox.ac.uk/pascal/VOC/voc2012/>

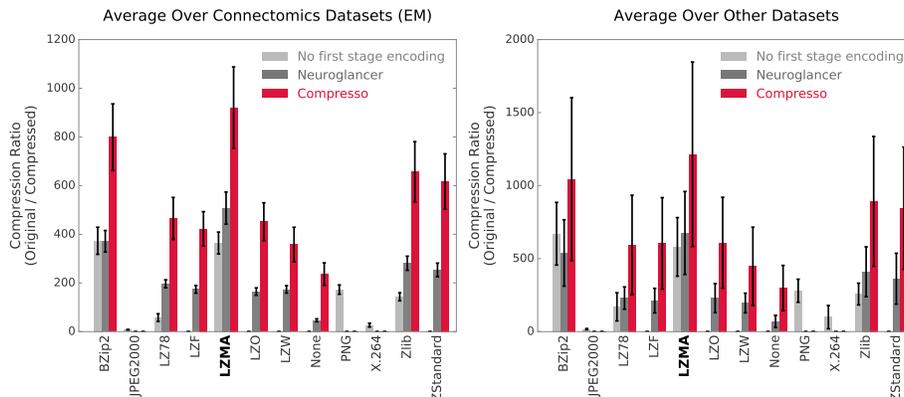


Fig. 4. Compression ratios of general-purpose compression methods combined with Compresso and Neuroglancer. Compresso paired with LZMA yields the best compression ratios for all connectomics datasets and in average (four out of five) for the others.

In terms of speed, Compresso is on par with Neuroglancer across all datasets and achieves throughput of 112.16 megabytes per second ($SD = 18.62$ MB/s) for compression and 222.85 megabytes per second ($SD = 32.14$ MB/s) for decompression. All experiments ran on a single core of a Intel Xeon 2.3GHz CPU.

4 Conclusions

We have introduced Compresso, an efficient compression tool for segmentation data that outperforms existing solutions on connectomics, MRI, and other segmentation data. In the future we plan to improve random access to lower memory requirements for online viewers and enhance compression of the metadata. Also, we will integrate Compresso into our analysis pipeline and various end-user applications. To encourage testing of our tool, replication of our experiments, and adoption in the community, we release Compresso and our results as free and open research at github.com/VCG/compresso.

M. Mitzenmacher is supported in part by NSF grants CNS-1228598, CCF-1320231, CCF-1535795, and CCF-1563710. H. Pfister is supported in part by NSF grants IIS-1447344 and IIS-1607800, by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/Interior Business Center (DoI/IBC) contract number D16PC00002, and by the King Abdullah University of Science and Technology (KAUST) under Award No. OSR-2015-CCF-2533-01.

References

1. Aimar, L., Merritt, L., Petit, E., et al.: x264-a free h264/avc encoder (2005)
2. Collet, Turner: Smaller and faster data compression with zstandard, url: <https://code.facebook.com/posts/1658392934479273/smaller-and-faster-data-compression-with-zstandard/> (2016), accessed: 23-October-2016

3. Deutsch, P., Gailly, J.L.: Zlib compressed data format specification version 3.3. Tech. rep. (1996)
4. Everingham, M., Van Gool, L., Williams, C.K.I., et al.: The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. <http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html>
5. Fredman, M., Saks, M.: The cell probe complexity of dynamic data structures. In: Proceedings of the twenty-first annual ACM symposium on Theory of computing, pp. 345–354. ACM (1989)
6. Google: Brotli compression format, url: <https://github.com/google/brotli> (2016), accessed: 11-October-2016
7. Google: Neuroglancer compression, url: https://github.com/google/neuroglancer/blob/master/src/neuroglancer/sliceview/compressed_segmentation/readme.md (2016), accessed: 21-October-2016
8. Haehn, D., Knowles-Barley, S., Roberts, M., et al.: Design and evaluation of interactive proofreading tools for connectomics. *IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE SciVis 2014)* 20(12), 2466–2475 (2014)
9. He, L., Chao, Y., Suzuki, K., Wu, K.: Fast connected-component labeling. *Pattern Recognition* 42(9), 1977–1987 (2009)
10. Kasthuri, N., Hayworth, K.J., Berger, D.R., et al.: Saturated reconstruction of a volume of neocortex. *Cell* 162(3), 648–661 (2015)
11. Lehmann, M.: Liblzf, url: <http://oldhome.schmorp.de/marc/liblzf.html> (2016), accessed: 13-October-2016
12. Liu, T., Jones, C., Seyedhosseini, M., Tasdizen, T.: A modular hierarchical approach to 3d electron microscopy image segmentation. *Journal of neuroscience methods* 226, 88–102 (2014)
13. Martin, D., Fowlkes, C., Tal, D., et al.: A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In: Proc. 8th Int'l Conf. CV. vol. 2, pp. 416–423 (July 2001)
14. Nunez-Iglesias, J., Kennedy, R., Plaza, S.M., et al.: Graph-based active learning of agglomeration (gala): a python library to segment 2d and 3d neuroimages. *Frontiers in neuroinformatics* 8, 34 (2014)
15. Oberhumer, M.: Lzo real-time data compression library. User manual for LZO version 0.28, url: <http://www.infosys.tuwien.ac.at/Staff/lux/marco/lzo.html> (February 1997) (2005)
16. Pavlov, I.: Lzma sdk (software development kit) (2007)
17. Roelofs, G., Koman, R.: PNG: the definitive guide. O'Reilly, Inc. (1999)
18. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, pp. 234–241. Springer (2015)
19. Seward, J.: bzip2 (1998)
20. Skodras, A., Christopoulos, C., Ebrahimi, T.: The jpeg 2000 still image compression standard. *IEEE Signal processing magazine* 18(5), 36–58 (2001)
21. Suissa-Peleg, A., Haehn, D., Knowles-Barley, S., et al.: Automatic neural reconstruction from petavoxel of electron microscopy data. *Microscopy and Microanalysis* 22(S3), 536–537 (007 2016)
22. Vandevenne, Alakuijala: Zopfli compression algorithm, url: <https://github.com/google/zopfli> (2016), accessed: 11-October-2016
23. Welch, T.A.: A technique for high-performance data compression. *Computer* 17(6), 8–19 (1984)
24. Ziv, J., Lempel, A.: Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory* 24(5), 530–536 (1978)