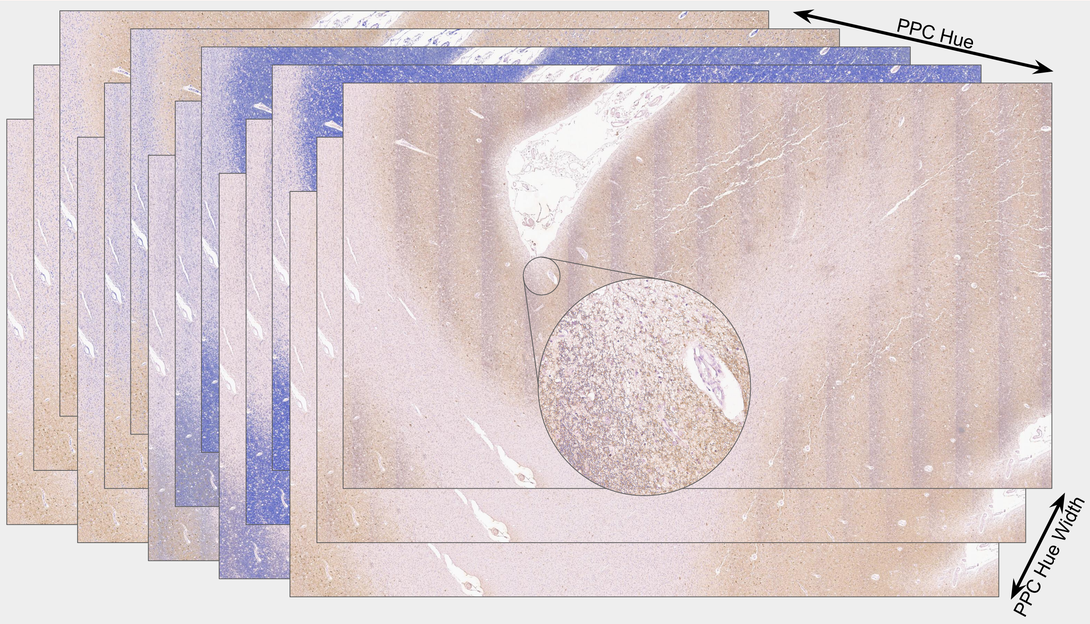
Efficiently Writing and Visualizing Many-Dimension Large Images

Anne Haley: Kitware, Inc.; **David Manthey**: Kitware, Inc.; **Lee Cooper**: Northwestern Feinberg School of Medicine; **David Gutman**: Emory University School of Medicine

Many digital pathology workflows require the **generation of large, multiresolution images** for visualization or analysis. Increasing utilization of highly multiplexed imaging is also producing new multichannel data that is complex to process and store. This data can be massive; a processed 5 gigapixel image with numerous channel and parameter variations can produce terrapixels worth of results. This motivated us to create a simple, yet **flexible, Python API to write WSI and multi-dimension large images**. This open source library can store generated or modified imaging data into existing formats that can be directly and efficiently handled by existing tools. We demonstrate this in exploring optimum parameters for several use cases

Parameter Sweep Example: Positive Pixel Count



The optimum parameters are not always obvious at different scales. Parameter sweeps can be run on full resolution WSI and easily output and visualized. A good viewer can let the user explore the different parameter values, zooming in and out as desired. In this example, a positive pixel count was generated for different hue and hue-width values across a set of slides showing Tau factor on brain tissue. Besides determining the optimal values to understand the biological features of the sample, the parameter sweep also shows the miscalibration of the lighting in the original scan.

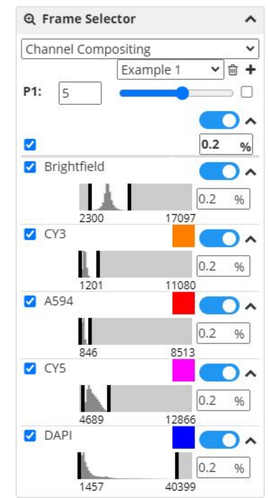
Efficient API for Writing Many-Dimension WSI

Writing large images is efficient and simple. The **open source** large_image library (see QR code) makes this easy in Python:

```python
import large_image
result = large_image.new()
for nparray, x, y in fancy_algorithm():
    result.addTile(nparray, x, y)
result.write('sample.tiff', lossy=False)
```

Multi-dimension data is just as clean, whether the dimensions are channels, z, time, or complex parameter sweeps:

```python
result = large_image.new()
wsi = large_image.open('multi_channel.tiff')
for channel in range(wsi.frames):
    for param1 in parameter_list:
        for tile in wsi.tileIterator(frame=channel, format='numpy'):
            data = my_algorithm(tile['tile'])
            result.addTile(data, tile['x'], tile['y'], c=frame, p1=param1)
# The writer supports a variety of formats, here we write OME NGFF
result.write('sample.zarr.zip', lossy=False)
```



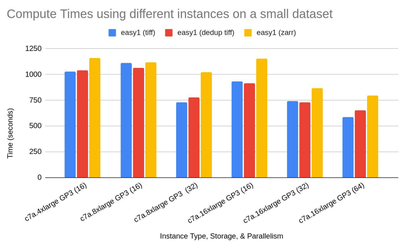
Key features of the new API are

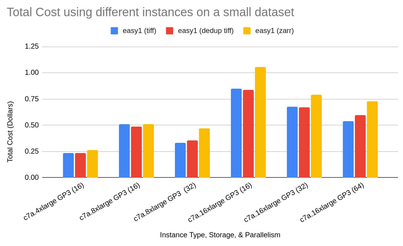
- Reduced code complexity by using well-tested, properly abstracted interfaces
- Increased compatibility via output files written in standard formats, including TIFF and OME NGFF.
- Can be used independently or as part of a HistomicsTK plugin

The HistomicsTK platform allows for efficient viewing of any number of axes of data.

- Composite channels or bands (colors)
- Max-merge on any axis
- Auto-scaling or custom contrast adjustments
- From the API, complex, on-the-fly tile compositing

Comparing Amazon Web Services Costs



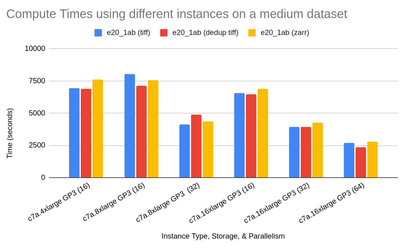


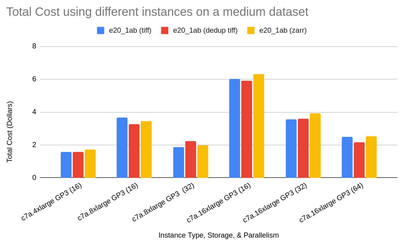
For the small data set, compute times were only marginally improved with greater parallelism, buts costs rose significantly. The processing method only made a mild difference. If we need to run this process on a lot of images, **the cheaper instances are sufficient**.

Frequently, the same algorithms need to be run on a large number of images. Scalability can be achieved by using a cloud computing service, such as AWS EC2 compute instances and S3 storage. Especially when running length parameter sweeps, it is useful to run representative data sets on a variety of instances and with some method variations to determine the fastest or cheapest options.

A study was done with (left) a small image with 6250 parameter combinations and (right) a medium image with 1100 parameter combinations. The algorithm was a PPC sweep where there is a balance between compute time and I/O access time. Three different storage methods were compared; the zarr storage should have the least computation overhead but requires a relatively slow compression step; tiff output is highly parallellizable; deduplicate tiff reduces data volume at the expense of computation.

Data was loaded and saved directly to an S3 bucket.





For the medium data set, compute times are greatly improved with more cores and more parallelism. With the added data volume, the smaller deduplicated tiffs are both **faster and cheaper**. If we need to run this process on a lot of images, our budget and time constraints will determine which instances we prefer.

Running Algorithms and Visualizing Multi-Dimension Data

Scale any algorithm:

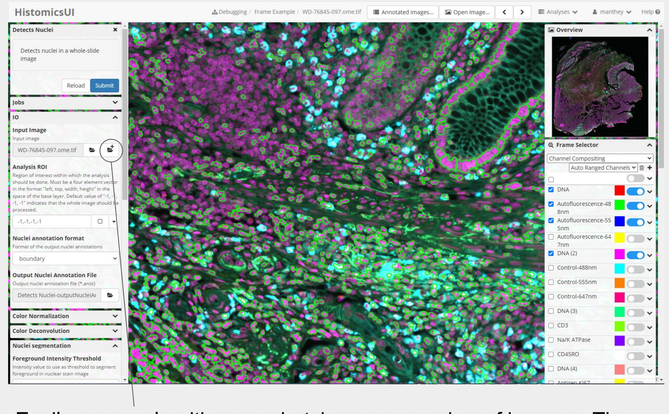
- Use a library of existing algorithms from traditional image processing and the latest trained AI models
- Package your own algorithms to run on images
- Run at different scales:
  - On a Region Of Interest. Depending on the algorithm this can be a rectangle or a complex shape
  - Run on a single image
  - Run on a batch of images

Handle multi-dimensional and multi-channel data.
Run algorithms on:

- Original data (such as H&E or the DAPI channel of an immunofluorescnce image)
- A specific frame of an image, including different channels, different z-slices, time, or arbitrary axes
- A composited view of an image. Use the dynamic image controls to mix channels, adjust brightnesses, and make biological features more visible.
- Compositions from multiple images (fuse H&E with fluorescence data, for instance).

The open source reference software can be run on a local computer or on cloud compute resources.

Algorithms can run on the same machine or scale out to arbitrarily many worker nodes.



Easily run an algorithm as a batch on any number of images. The same parameters can be applied in all cases. For algorithms that take multiple inputs this can be processed as a matrix of values or a parallel set of values.

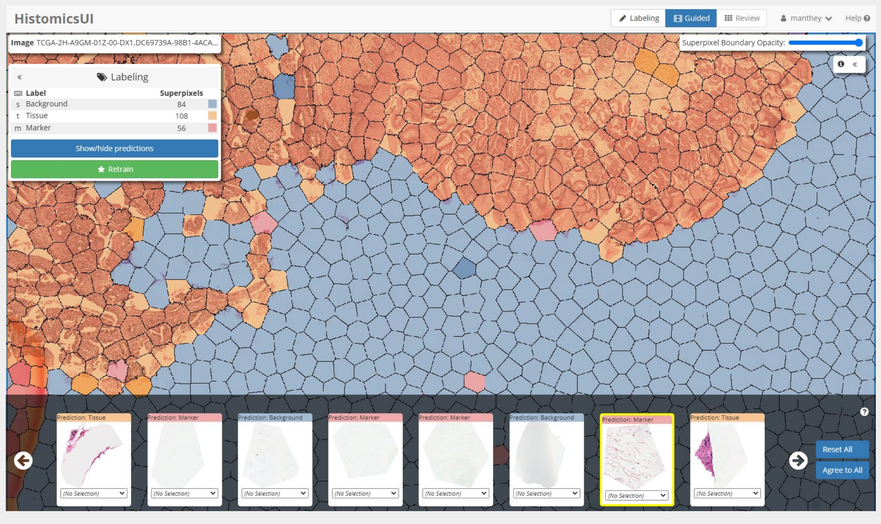
Superpixels and Masks

To improve concordance between annotations, images can be divided into superpixels using standard algorithms like SLIC-0. Typically, these algorithms expect the entire image to fit in memory, which is unrealistic with many WSI. Rather, we can run superpixel algorithms on tiled subsets of the original image with overlapping regions between tiles. This requires masking off edge superpixels, and providing a prefilled mask where adjacent superpixels were already located.

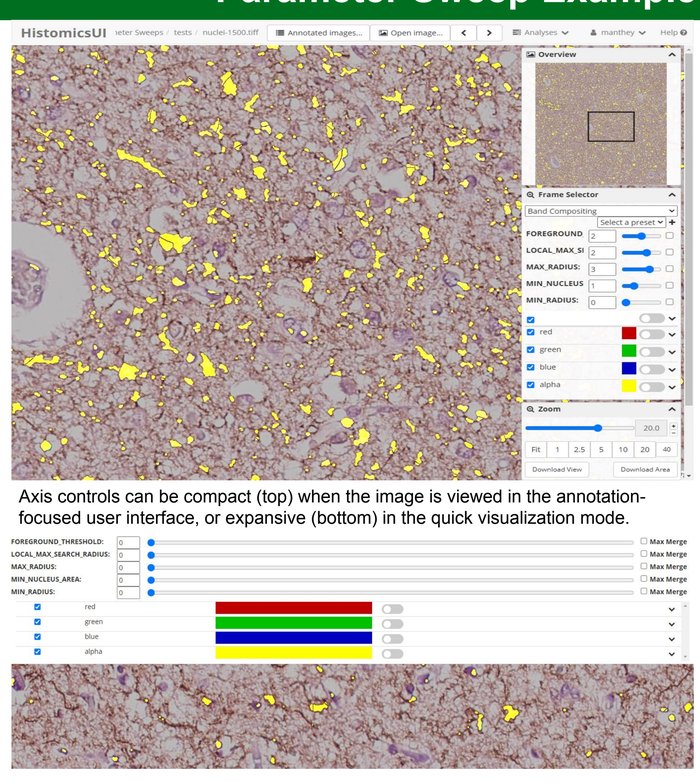
Using our new API, this process becomes vastly easier and more parallelizable. Prior to this, the generation code had to do the bookkeeping for which tiles had been partially generated, maintaining individual tile masks. With the new library, aside from substantially reduced code complexity, **accessing partial results is faster by around a factor of 2**, and parallelization becomes straightforward.

In the example shown to the right, the superpixels are a label map image that is one quarter the scale of the base WSI. The overlaid colors are a simple tissue / background detector, though the same ML training algorithm is being used to detect specific tissue types.

The displayed UI is optimized to label the least certain superpixels. The trained model can then predict labels on new images.



Parameter Sweep Example: Segmentation



Axis controls can be compact (top) when the image is viewed in the annotation-focused user interface, or expansive (bottom) in the quick visualization mode.

Many segmentation algorithms are optimized for specific images and structures, such as nuclei on hematoxylin stained images. When using other image modalities, these segmentation methods may not produce the desired results. A sweep of algorithm parameters can show if the algorithm can successfully perform the desired segmentation, and, if so, what parameters are most effective.

In the shown example, a standard geometric nuclei detection algorithm was applied to a WSI stained to show tau proteins. To determine if this algorithm could successfully highlight neurons with tau protein, the algorithm was run with five different adjustable parameters varying across a range of values.

After writing the results to a single file, the HistomicsTK interface allows exploring each of these parameters by just scrubbing through the selected values and seeing the resultant segmentations.

To better visualize the segmentations, colors can be adjusted or the image itself can be adjusted to be dimmer or brighter.

The examples shown are based on open source software by Kitware, Inc. Some of this work has been funded by National Library of Medicine grant R01LM013523, “Cloud strategies for improving cost, scalability, and accessibility of a machine learning system for pathology images” in collaboration with Northwestern University. Some of the data is also from an Alzheimer’s research project at Emory. The core software is HistomicsTK using the Digital Slide Archive reference deployment. Most of the API shown for efficient image storage is part of the large_image python library.

HistomicsTK

Kitware provides advanced technical computing, state-of-the-art AI, and tailored software solutions to our customers. The Data and Analytics depatment works in digital special collections, including HistomicsTK as a platform for histology visualization and analysis. Kitware partners with academic and commercial customers to optimize workflows, develop new capabilities, and advance scientific research.

kitware