NYU | TANDON

Cell Image Pre-Processing with Dask Image

Presented by Jake Nemiroff, Su Win, and Regan Bragg

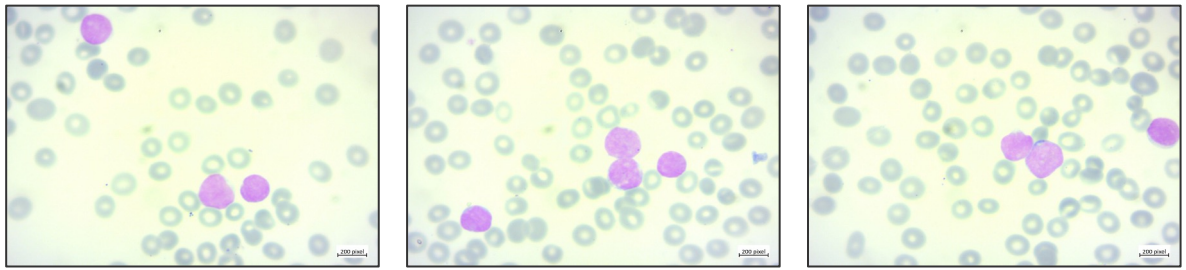
CSGY - 6513 (Fall 2022)

Presentation Agenda

- Problem Statement
- Why Dask?
- Data Pipeline Overview
- Image Segmentation Pipeline
- Performance Comparison between Dask and Scikit Image library
- Lessons Learned

Problem Statement - Why is this “Big Data”?

In many scientific fields, experts rely on collected images to perform analyses and distinguish important behaviors. In the context of biology, snapshots of cancerous cells can help scientists to further understand their mechanisms in contrast to their healthy counterparts. Even though experts have the area knowledge to decipher the meanings of images and recognize patterns, they still must obtain quantifiable data by processing the images. If they are captured at each second or even millisecond, it becomes a hefty task that is not easily handled by a single machine. Instead, it makes sense for them to **utilize big data methods** by sending the task to clusters of multiple nodes or enable multithreading.



Images courtesy of Abolghasemi, Aria, Asadi, Bashash, Ghaderzadeh, Hosseini, in “A Fast and Efficient CNN Model for B-ALL Diagnosis and its Subtypes Classification using Peripheral Blood Smear Images” (2021)

Why Dask?

We could have chosen Spark...

Short Answer:

> Dask is written in Python and we prefer Python

- Dask is written in Python while Spark is in Scala
- Both Spark and Dask offer in memory computing, data locality, and lazy evaluation. And, a lot of other similarities.
- However, Spark does not has native support for multi-dimensional array
- Dask fully **supports the NumPy model for scalable multi-dimensional arrays** since it is very closely coupled with libraries like NumPy, pandas, and Scikit-learn
- It is also **lighter-weight transition from local computing to cluster computing** compare to Spark

Data Pipeline Overview

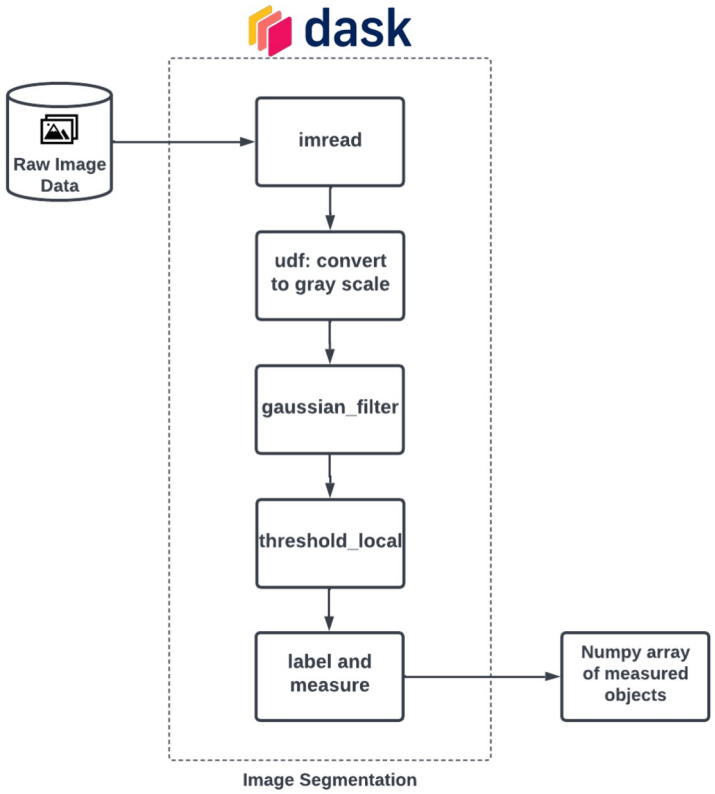
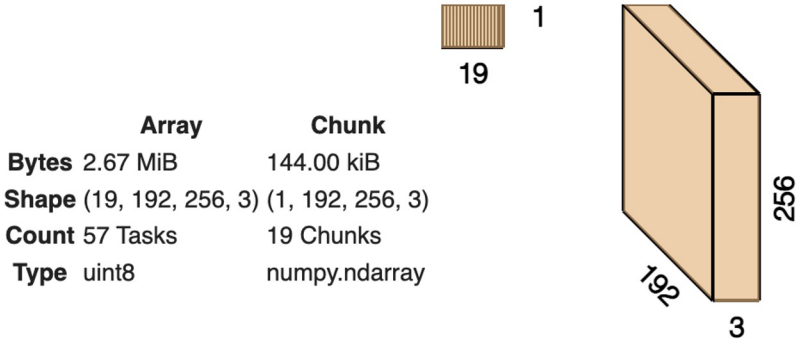


Image Segmentation Pipeline

Step 1: Bulk read of images

- We utilized the `imread` method of the Dask Image library to perform a bulk read on a series of jpg cell images.
- Each individual .jpg file in the input becomes its own chunk in the Dask array.



	Array	Chunk
Bytes	2.67 MiB	144.00 kiB
Shape	(19, 192, 256, 3)	(1, 192, 256, 3)
Count	57 Tasks	19 Chunks
Type	uint8	numpy.ndarray

Chunk 0 in the dask array:

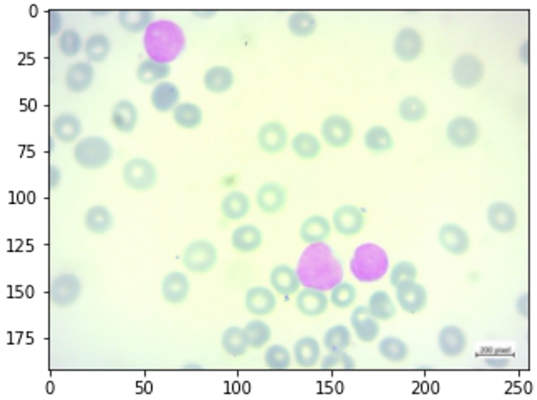
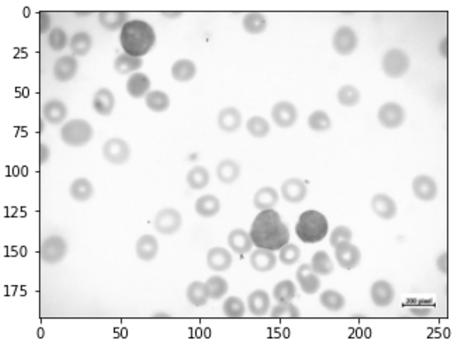


Image Segmentation Pipeline

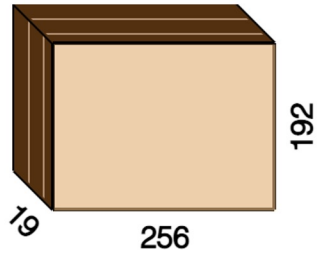
Step 2: Convert to Grayscale

- We want to reduce the number of color channels in the image. Decreasing the data load by a factor of 3 will enable us to do complex filtering operations in a shorter period of time.
- This also increases our ability to visualize our results.

Chunk 0 in the dask array:



	Array	Chunk
Bytes	7.12 MiB	384.00 kiB
Shape	(19, 192, 256)	(1, 192, 256)
Count	209 Tasks	19 Chunks
Type	float64	numpy.ndarray



We eliminated a dimension in our dask array!

Image Segmentation Pipeline

Step 3: Filtering

- We denoised the cell images with a small amount of blur by using a **Gaussian Filter.**
- This will cause a loss of sharpness in the image, but for image segmentation with thresholding techniques, it will improve the results.

Chunk 0 in the dask array:

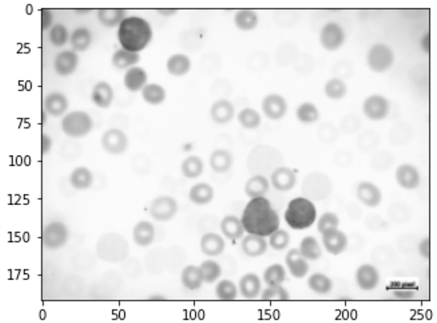
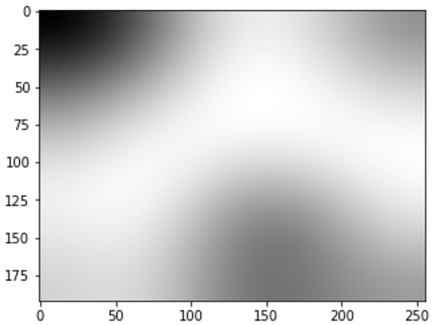


Image Segmentation Pipeline

Step 4: Thresholding

- Thresholding an image allows us to separate the objects in the image from the background.
- Rather than absolute thresholding, which calculates a 'one size fits all' value for the background, we instead opt to use local thresholding. **Local thresholding calculates a threshold value independently for each pixel of each image of the dask array.**

Local threshold for Chunk 0 in the dask array:



Chunk 0 in the dask array:

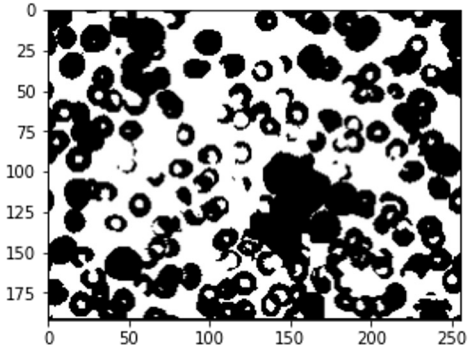


Image Segmentation Pipeline

Step 5: Measuring Objects

- The final step is to measure the objects within each individual image. This is a two-part process that involves:
  - Labeling each continuous object in the image.
  - Measuring the pixel area of each labeled object.

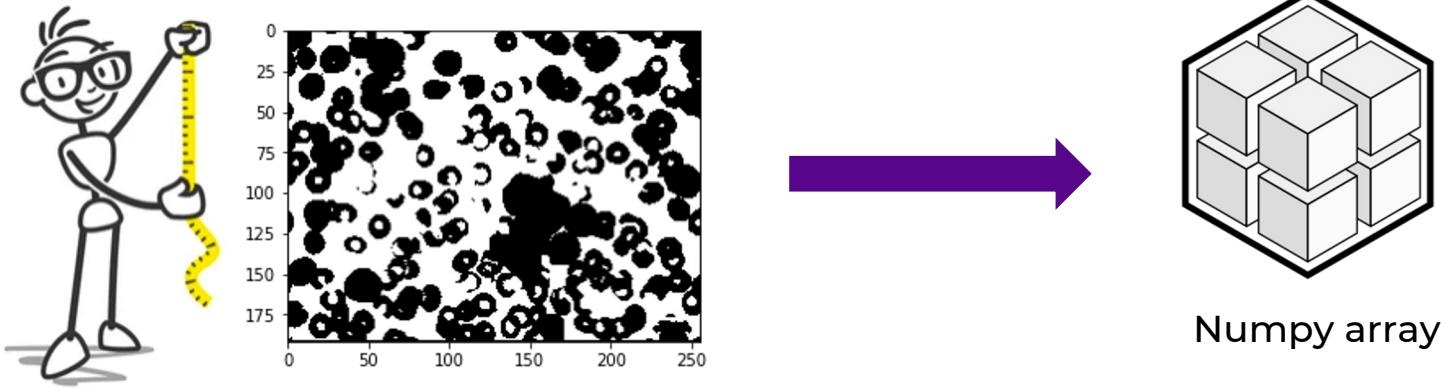
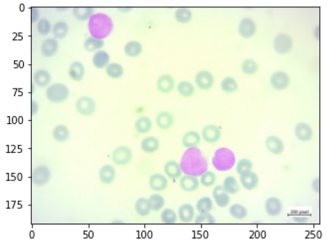
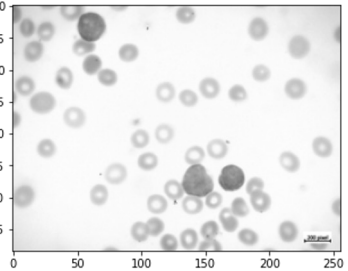
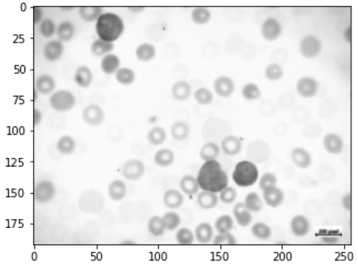
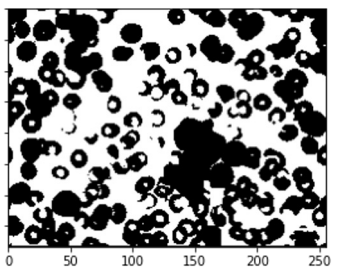
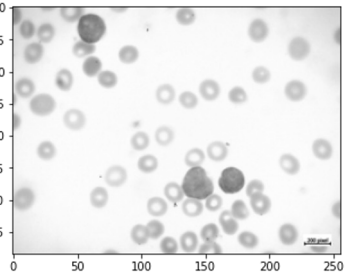
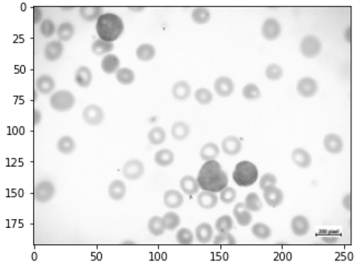
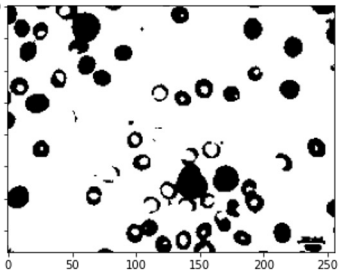


Image Segmentation Comparison

Using the following test image:



	Greyscale	Gaussian Filter	Local Thresholding
Dask-image			
Scikit Image			

Performance Comparison

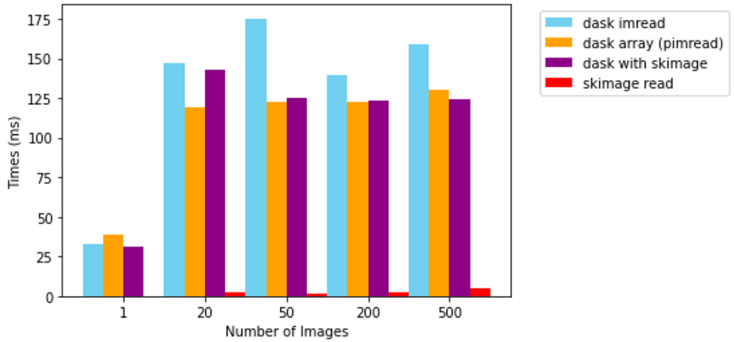
We ran tests for 1, 20, 50, 200, and 500 images

- Compare various packages (dask, skimage, pims).
- Measure execution time for image reading and processing.

Some things to note:

- dask.array.image.imread allows for an optional custom imread function, although dask overhead still associated.
- dask_image reads each individual tile using pims.open on the input file pattern, rather than reading in the relevant file in a more targeted manner.

Performance Comparison of Reading Images



Process	1 image	20 images	50 images	200 images	500 images
Dask imread	32.516 ms	146.798 ms	175.199 ms	139.793 ms	158.926 ms
Dask.array.image (imread = pimread)	38.889 ms	118.978 ms	122.171 ms	122.517ms	130.457 ms
Dask.array.image (imread=skimage.imread)	30.997 ms	143.175 ms	124.876 ms	123.422 ms	123.790 ms
Skimage imread	0.255 ms	2.546 ms	1.583 ms	2.077 ms	4.897 ms

Lessons Learned

What did we find out about Dask?

- Dask is open-source and still quite new - so it isn't perfect!
- Dask essentially scales the Scikit learn library - hence on a single compute source the performance was approximately the same.
- The API documentation was easy to follow, but a lot of the methods require fine-tuning of parameters

Questions?