CSCI 8360 Data Science Practicum
Department of Computer Science
University of Georgia

Project 2: Cilia Segmentation

DUE: Tuesday, March 5 by 11:59:59pm

Out February 12, 2019

1 Overview

You've all done extraordinarily well with "document" classification. Now on to image processing!

Your task is to design an algorithm that learns how to segment *cilia*. Cilia are microscopic hairlike structures that protrude from literally every cell in your body. They beat in regular, rhythmic patterns to perform myriad tasks, from moving nutrients in to moving irritants out to amplifying cell-cell signaling pathways to generating calcium fluid flow in early cell differentiation. Cilia, and their beating patterns, are increasingly being implicated in a wide variety of syndromes that affected multiple organs.

Connecting ciliary motion with clinical phenotypes is an extremely active area of research. We'll try to address a very small slice of it here. Your goal: *find the cilia*.

2 Data

The data are all available on GCP: `gs://uga-dsp/project2`

In that parent folder, you'll find two subfolders: `data` and `masks`.

- `data` contains a bunch of folders (325 of them), named as hashes, each of which contains 100 consecutive frames of a grayscale video of cilia.

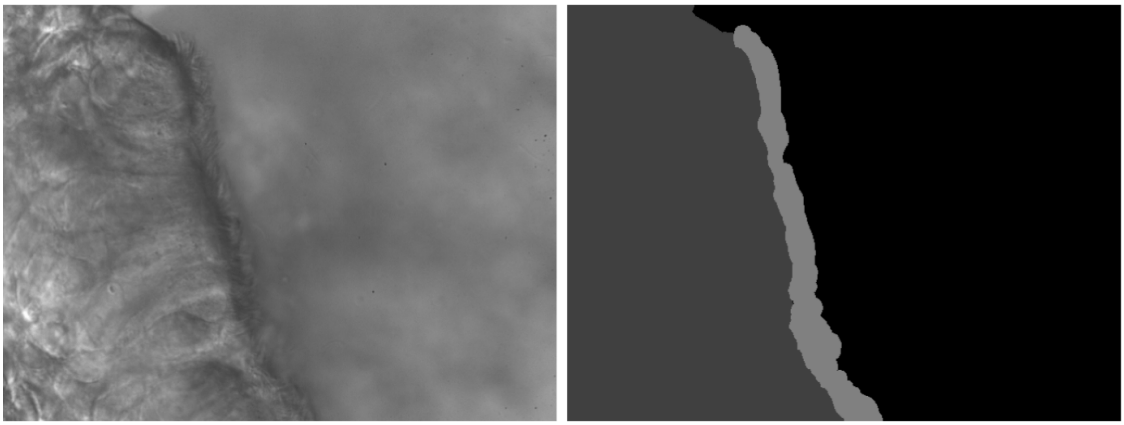

Figure 2.1: LEFT: A single frame from the grayscale video. RIGHT: A 3-label segmentation of the video (0: background, 1: cell, 2: cilia). The "cilia" label is what we're interested in for this project.

- `masks` contains a number of PNG images (211 of them), named as hashes (corresponding to the subfolders of `data`), that identify regions of the corresponding videos where cilia is.

Also within the parent folder are two text files: `train.txt` and `test.txt`. They contain the names, one per line, of the videos in each dataset. Correspondingly, you will only find masks in the `masks` folder for those named in `train.txt`; the others, you'll need to predict (and are on AutoLab)! The training / testing split is 65 / 35, which equates to about 211 videos for training and 114 for testing.

The data itself are grayscale 8-bit images taken with DIC optics of cilia biopsies published in this 2015 study. For each video, you are provided 100 subsequent frames, which is roughly equal to about 0.5 seconds of real-time video (the framerate of each video is 200 fps). Since the videos are grayscale, if you read a single frame in and notice its data structure contains three color channels, you can safely pick one and drop the other two. Same goes for the masks.

Speaking of the masks: each mask is the same spatial dimensions (height, width) as the corresponding video. Each pixel, however, is colored according to what it contains in the video:

- 2 corresponds to **cilia** (what you want to predict!)
- 1 corresponds to **a cell**
- 0 corresponds to **background** (neither a cell nor cilia)

Pay attention, as this is important: you are **ONLY** required to predict **CILIA**. In preliminary testing, we found that prediction could be vastly improved if one incorporated

cell-specific predictions as well, since cilia tends to run along the sides of cells (a strong conditional relationship). Furthermore, the addition of another label helps even out the label space, as otherwise the "background" label would dominate (leading to some of those problems you wrestled with in Project 1 with skewed labels!). These problems aren't entirely solved by the inclusion of a cell label, but it does help. Nevertheless, you are welcome to train your model without it if you prefer. The only label the autograder will be testing for is that of cilia.

To repeat: you **ONLY** need to predict cilia! That said, your label for cilia **MUST BE** the number 2! If we run into problems with this we'll re-evaluate, but it makes writing the autograder a lot easier if both the masks and the predictions use the number 2 as the cilia label.

The 100 accompanying frames with each video are available if you want them. We've had pretty good success predicting regions of cilia with only a single frame and a ground-truth mask, but this obviously ignores motion, which can help differentiate heavily-textured regions that move from those that don't. Again, though, the specific modeling decisions are yours.

Your goal is to develop a video segmentation pipeline that identifies the regions of the videos containing cilia as accurately as possible (see Fig. 2.1). The ground-truth labels for the 114 test videos are stored in AutoLab and will be used to grade your submissions.

The scoring mechanism to evaluate segmentation is simple: **intersection-over-union**. The PyImageSearch blog had a great article on this exact metric for image segmentation (Fig. 2.2). It's pretty simple: take the area of overlap between the predicted region and the actual region (intersection), and divide that by the union of the predicted region and the actual region (union). Perfect overlap gives you 1.0, no overlap gives you 0.0, and you know you're doing well if you get something better than 0.5.

The average score over the 114 testing videos is what will be displayed to you on AutoLab. I know this is imprecise, but there's just no feasible way of giving feedback on 114 different videos. Plus, if you're working on the Kaggle competition, this is *exactly* how the submissions are scored!

3 Theory Guidelines

We're solidly in imaging territory, which means Convolutional Neural Networks (CNNs) are the Big Thing^{TM}. Unfortunately, due to the DIC nature of this data, everything is lit, and arbitrarily so: there isn't really a correspondence between the grayscale values of the cilia in one video and those of another. What you'll have to take advantage of here is *texture*, and that can't be learned through linear methods, at least not easily. Therefore,

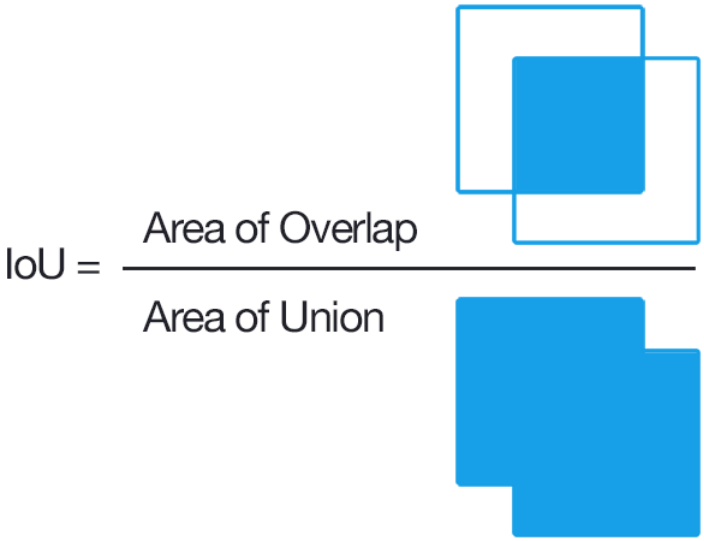

Figure 2.2: Source: https://www.pyimagesearch.com/wp-content/uploads/2016/09/iou_equation.png

convolutions are going to be your best bet.

That said, a good starting point could still take advantage of *relative* fluorescence changes. One of the favorite (because there really aren't any others) metric of researchers who study cilia is *ciliary beat frequency*: simply, the frequency at which cilia beat back and forth. There is a lot of evidence showing that healthy cilia tend to beat in a range around 10-12Hz (so in 100 frames of a 200fps video, you'd expect to see about 5-6 full "oscillations" of healthy cilia). Unfortunately, there are a lot of examples where this metric breaks down as a rudimentary classifier.

But it does give us an indicator that while raw pixel intensity isn't going to mean much here, *relative fluctuations* could still be important. In that 2015 study, the authors found that simply thresholding pixels based on *fluctuation variance* relative to some threshold worked decently well. It also left a ton of artifacts, but at the very least usually had a very low false-positive rate for identifying pixels as either having cilia or not.

Let's say you have a single pixel: over the 100 frames, this pixel's intensity fluctuates (a time series). Let's call the *variance* in that fluctuation σ_i at pixel $\vec{x}_i$. Now I calculate that same variance at every pixel, giving me $\sigma_1, \sigma_2, ..., \sigma_i, ..., \sigma_n$ for n pixels in a video. If I throw out every pixel that has a $\sigma_i < t$ for some threshold t (usually a function of the σ_i's), I'm left with all the pixels that should a fluctuation variance larger than that threshold... and therefore, plausibly, moving.

As stated, this will usually cut down on the pixels that are *definitely not* cilia, but will

still give you a lot of artifacts; a loose bound on the regions of cilia. To tighten the bounds, here are some additional things to try:

- Compute the actual beat frequency at each pixel, instead of just the fluctuation variance. This will entail using the Fast Fourier Transform, and choosing the frequency that is *dominant* at each pixel. This will take more work but will be a lot more robust to noise than simple grayscale variance.
- Use a median filter to smooth the pixel values (either fluctuation variance or beat frequencies) before applying a threshold. This will help eliminate small artifacts that tend to appear all over the mask.
- Consider moving out of the grayscale space entirely and into the *optical flow* space. This essentially replaces the grayscale pixel values with motion vectors at each pixel that estimate direction and magnitude of movement at each pixel. These motion vectors are computed over small neighborhoods, making them much more robust than any grayscale-based statistics such as fluctuation variance or even beat frequency.
- Full convolutional networks for semantic segmentation. U-Net, FCN, and Tiramisu are excellent choices for this.

4 Engineering Guidelines

Code reviews by teammates are now REQUIRED. This means everyone is **REQUIRED** to lock your `master` branch against direct pushes; instead of pushing directly to `master`, everyone will now create their own branches for any modifications, commit them, submit a pull request, and have their code reviewed by at least one other teammate before it can be merged into the `master` branch. You can set this up in two steps: first, protect the master branch; second, enable required reviews for pull requests (go through Step 5, and then Step 9-10).

The gloves are off; anything is fair game. Use whatever framework you want, distributed or not. If you're not sure where to start, I would highly recommend checking out this post at PyImageSearch about semantic segmentation and deep learning; from there, any literature on semantic segmentation is going to be the ticket. Just know that I will be **strictly enforcing Academic Honesty policies**. If I find code that has been substantially copied from others, or ideas used without attribution, there will be problems. I have already found a few close calls in previous projects, but I found later they were cited, just in an unexpected location (MOSS is *extraordinarily* accurate; don't try to fool it. Just cite where you got your work, if you built on something that somebody else started).

Create a GitHub repository for you and your team to work on the project. You can make it private, and you can create a GitHub team within the DSP-UGA org, but the development must take place under the GitHub organization as I will be using

it and the activity you record on it as part of the grading process. Name the repository with the scheme of `team-name` to help differentiate from other teams and future projects. **Seriously, please name it correctly**.

Exercise good software design principles. This means: adopt a consistent coding style, document your code, use the GitHub ticketing system to identify milestones and flag bugs, and provide informative and frequent `git commit` comments. Also, be sure you include an informative `README`, an accurate `CONTRIBUTORS`, a `LICENSE`, and are organizing your code in a clean and logical way.

Shut down your GCP clusters when you are finished testing. GCP clusters incur costs for every hour they are turned on, even if they are not actively running any jobs. That said, it's also expensive to spin them back up if you're running a lot of jobs, so I would encourage you to turn them on, run a bunch of tests, and only when you're out of tests to run, you shut them down until you can get more in the pipeline.

5 Hints and Reminders

These datasets a few GBs; large enough to keep you honest, but not huge. You don't have to worry about distributing your computation, but don't be naive about your implementation either. Start with a small subset of the data and expand your dataset as you grow confident in your methods.

If you run into problems, work with your teammates. Open up GitHub tickets, discuss the problems, find workarounds. If you still run into problems, query the Slack channel. I'll be regularly checking Slack and interjecting where I can. Whatever you do, please don't turn into Lone Riders and try to solve the whole thing yourself. You will be penalized for it!

Who will set this year's top score? Last year's 8360 cohort managed a project-based 45.81168 IoU. Yep, less than 50%! This is a tricky project; can anyone beat that??

6 Submissions

All submissions will go to **AutoLab**, our brand-new autograder and leaderboard system.

One teammate should be designated for submitting on behalf of the team; they should submit to Autolab *using the team name* so I can easily match submissions with teams. If two people from the same team submit, I will pick one at random and grade that submission, so be sure you're communicating clearly.

When submitting, you will submit one file: **the `tar` archive containing all the PNGs of your predicted masks.** Each PNG should be named according to the hash of the

video it predicts segmentations for, and should have a "2" for every pixel predicted to be cilia (any other non-2 value is fine for other pixels, since they'll be ignored anyway). I haven't tested it yet, but you should *probably* ensure the images are in 8-bit unsigned integer format when you save them, just to avoid problems with the autograder.

You don't have to submit masks for every single video when you submit to AutoLab–it skips any videos for which it doesn't find a prediction–but it *does* compute its average based on the total number of testing videos. So you'll essentially get 0s for those videos you don't submit.

Once your model has made its predictions and created the image files of the masks, `tar` them all together with the command:

```
> tar cvf p2.tar *.png
```

That will create a file named `p2.tar` of all the PNG files in the current directory, though you can name the tar archive whatever you want (AutoLab will automatically rename it). The PNG images inside, however, *must* be named correctly to correspond with the videos, otherwise they may not be correctly paired with the right video or even used at all.

Your average IoU accuracy will be tabulated and compared against the true labels (hidden on the server) and should appear on the leaderboard.

If you think you can do better than what shows up–particularly to beat out everyone else–make another submission! Each team will get 50 unpenalized submissions, which should be more than enough to test your models (but not so much to let you spam it). AutoLab is programmed to close submissions *promptly* at 11:59pm on March 5, so give yourself plenty of time!

7 Grading

Given that this is a practicum, and that you're all talented graduate students, the grading for the projects is a little different, operating on a sliding scale with some stationary points that require extra effort to move beyond.

Everyone's starting grade is a **B** (85%). If you follow the guidelines in this handout, implementing the basic algorithm as described, and everything functions properly, and your documentation is reasonably ok, and there were no major issues with your team, then you also finish with a **B**. If you're happy with that, great! If not, read on:

Getting an **A** requires an investment of additional work, both in terms of functionality and algorithm performance, as well as on the software engineering side. You must

demonstrate the extra customizations you made to your code and how those tweaks resulted in an improved score on the final testing set (getting the top score, for instance, would certainly qualify–though I feel I should point out that this sort of thing is relative, in that if you get a new top score that is only 0.000001% better than the previous one, it won't translate to an entire additional letter grade. #SorryNotSorry). If your pipeline is multi-staged, clearly showing that a lot of thought went into the theoretical design–perhaps even using a current method (that is *properly* and *prominently* cited)–it will help boost your grade.

Furthermore, you must show exemplary software engineering techniques: your code style should be clean, logical, and well-documented, and you should provide a clear README with instructions to new users on installing and deploying your document classifier. Your `git commit` comments and GitHub tickets should provide a "paper trail" of documentation to show the route your team took in developing the software. Unit testing, continuous integration tools, consistent and thorough use of branches and merges, and adherence to coding style standards (e.g. Python's PEP8) will also help push your grade above the default.

It should go without saying, but violations of standard coding practices or errors in the theory will drop your grade; you don't get a **B** for simply turning in an empty assignment.

This combination of **algorithmic novelty** and **engineering best-practices** will earn you and your team that well-deserved **A**.

8 Summary

In summary, these are the things I will look over when determining your team grade:

1. **Code**: Organization, structure, style, and clarity. Strategy and theory will also be a big component (what did you implement? was it implemented correctly? does the theory follow logically from the problem you're trying to solve?).
2. **Documentation**: This includes comments in the code, but also in the repository itself (GitHub wiki, `README` instructions). How easy would it be for someone to get up and running with your code, or to submit a bugfix?
3. **Accuracy**: Testing accuracy as submitted on AutoLab, and how far you are from the very top.
4. **Team**: Like the code, I'm looking for structure and organization, as well as a "paper trail" to get a feel for your team's working dynamics. How were project efforts divided among teammates? What do the git logs reveal about who was contributing to the project? How does this align with the `CONTRIBUTORS` content? Were code reviews conducted properly?

5. **Extras**: These include things like continuous integration tools, project websites, theoretical novelty (within the constraints of the project), unit tests with good coverage, among others. Use your imagination!