Physics-Based Tracking of Moving and Growing Cells in a Colony

Meta Novitia
46208774
University of California, Irvine
November 28, 2018

Abstract

The purpose of this project is to count the number of cells for every frame in a given GIF. The general idea of the algorithm used is eliminating cell images one by one using breadth-first search and selecting acceptable cells. The GIF contains the animation of moving and reproducing cells, where the left side is the original animation, and the right side is a grayscale version with red highlights on the cell boundaries. For the ease of the project, I chose the grayscale (right side) version. The results of the program, which is the number of cells calculated in each frame, matches the number of cells in the GIF counted by eye.

Methods

I used the Python language to code my algorithm, importing the imgpy library to help me with image processing. The imgpy library supports the usage of GIFs, and is able to retrieve information on each pixel in an image. GIF images can be processed by the library into a list of images which represents each frame. I loop through each of these frames and process them separately. No other non-standard libraries were used.

Formatting the GIF

First, I format the GIF to simplify the process. This includes cropping off the left side of the image (using a crop function from imgpy), and converting some pixels to the colors I want. I convert the pixels by extracting the colors existing in the GIF, and converting all dark red pixels into light red (since there are 2 types of red-highlight colors). I also convert any other color that is not red or gray into black. This simplifies the images to have cell shapes (gray), cell borders (red) and background (gray and black). The problem now is the background, since there are gray marks on the image outside of cells due to lighting. This will be handled in the "Detecting a Cell" section.

Detecting a Cell

It is actually easy to differentiate gray pixels that belong to a cell and those that are not, but the method I am using is very dependent on the fact that cells are surrounded by red borders. What I did was to search the frame pixel by pixel and checking if it is a gray pixel that is touching a red pixel (up, right, left or down). This signifies that it is part of a cell, since the gray background pixels are not touching the red borders.

Processing and Erasing Cell from Frame

After the program finds a gray cell pixel, it will perform a breadth first search on it to count the number of pixels in the cell (area) and to erase the cell from the frame. The breadth first search algorithm keeps a queue that initially has the gray cell pixel we found. It will then search the pixels next to it (up, down, left, right) and process each pixel. If the pixel is gray, then add it to the queue, increase the area count, and change the pixel to black. If the pixel is red, then just change it to black. If the pixel is black, then do not do anything to it. After the bfs finishes, we will have an area count of the cell, which will be used to determine whether or not the cell will be qualified as an actual cell. The red boundaries are not precise, and there are small gray cell pixels outside the boundaries, and also some overlapping cells which creates an intersection between the cells, making it look like 3 cells. To eliminate the mistake of counting these, we will only accept 'cells' that have an area of greater than 40 (arbitrary, and worked fine).

Results

For each frame, I output a line consisting of the count of cells of the frame into an output text file. Below is a chart that shows the results of when the program is run on the given GIF:

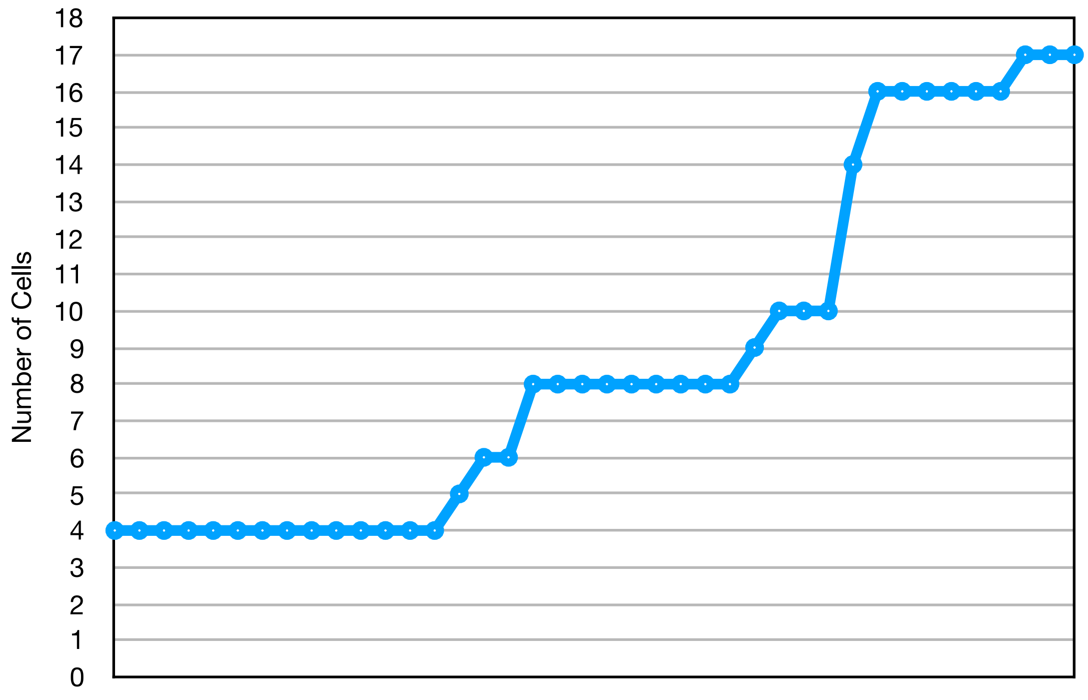

Figure 1: Number of Cells in Each Frame

As seen from the chart, the cells seem to reproduce geometrically, starting from a constant value of 4 and then multiplying to 8 cells, then resting again, then multiplying to 16. The ratio of the geometric sequence is 2, which makes sense since cells reproduce by splitting into two. The results (counted automatically) matches the number of cells in each frame in the GIF (counted manually).

However, the time it took for the program to finish was very long (about 50 seconds in total). The formatting itself took up about 35 seconds, and the processing of data took about 15 seconds. This is expected as the number of pixels is about 150000 per frame, and there are 40 frames, making about 6 million pixels to process. The complexity of the algorithm (breadth first search) is O(V+E), where V is the number of pixels, and E is 4 times the number of pixels, as the edges are the surroundings of the pixels, which makes a final Big-O of V.

In this project, I chose to use BFS instead of DFS as I encountered some complications when implementing the DFS solution. In the DFS solution, I tried to find cycles in the graph using the red borders, which proves unsuccessful because of some discontinuation in the graphs (pixels of the red borders not precise), and that there are overlaps of the cells. The BFS solution has a simpler approach, and an easier implementation than the DFS.

URL List

Python imgpy library	: https://pypi.org/project/imgpy/
Code for project	: https://github.com/MetaNovitia/cell_tracking