Project Report

Akhil Agnihotri | agnihotri.akhil@gmail.com

1 Problem Statement

We are given a problem where the microbiologists have a dataset $\mathcal{D}$ of images from the microscope and images from the sensor, and would like to store two sets of images in a dataset $\mathcal{D}'$:

- **All** images of the microscope which show the parasite body.
- **Subset** of the sensor images which show presence of dye for cancerous parasite *only*.

The image resolution for both equipments is $10^5 \times 10^5$, and a parasite is deemed cancerous if the total area covered by the dye is $p_{area} = 10\%$ more than the total area covered by the parasite. It is expected that fewer than 0.1% of the parasites will have cancer.

2 Steps

2.1 Storing the images

Since the image size of each parasite is $10^5 \times 10^5$ bits, which is ~ 1.25 GB per image, it is quite inefficient to just use a simple 2D binary matrix to store the images. Instead, for both, the microscope and sensor, we only store the pixels coordinates which contain information (parasite in case of microscope and dye in case of sensor images), normalized by the side of the original image (10^5), so that the images can later be re-scaled later to any desired size. This is a sparse and efficient data structure scheme.

- For images generated by the microscope (black and white) and letting the number of pixels that have parasite present (black pixels) be N, use a $N \times 2$ matrix to store the images, where each row r in the matrix is a list $[r_x, r_y]$:

  $$r_x = x/10^5 \quad ; \quad r_y = y/10^5,$$

  where x, y are from the pixel (x, y) where a parasite is present (i.e. pixel is black), assuming a (x,y) coordinate system for the image.

- For images generated by the dye sensor, we use the similar storage scheme as for the microscope, except that we take (x, y) to be coordinates of a pixel where dye is detected.

Worst case storage for this kind of data storage scheme is still ~ 1.25 GB per image.

2.2 Generate synthetic $\mathcal{D}$

2.2.1 Microscope images

For this, we initialize a 2D matrix of size $(10^5, 10^5)$ with all entries to be 0. Then we start a breadth first search (BFS) from a randomly selected point (x_{start}, y_{start}), and set all visited points to be 1. Once the number of visited points exceeds the minimum threshold (25% of the image area in our case), we randomly stop the search with a probability of $p_{stop} = 0.5$. In the end, return a $(N, 2)$ matrix containing N visited points and their coordinates.

2.2.2 Sensor images

Generating microscope images was a straightforward task. To generate realistic dye permeation inside a parasite, a lot of design choices need to made:

1. Do we randomly do BFS again from the (x_{start}, y_{start}) used in the generation of the microscope images? This would just require us to call the function again.
2. Do we draw random lines inside the blob generated from our microscope generation scheme, and do BFS starting from these lines? These lines represent the blood vessels of the parasite

While the second option above is more realistic, due to lack of time and for sake of simplicity, we adopt the first approach. To make the first approach a little more realistic, we add randomness to the BFS itself, as described below.

- With a probability of `cancer_probability = 0.001` (given), we first let `cancer=1`, else `cancer=0`.
- During the BFS loop if:
  - `cancer=1`: stop randomly when

    `area >= min_parasite_area + (1+random.random())*extra_sensor_area`,

    where `area` is the area covered by the dye, `min_parasite_area` is the minimum area that the parasite occupies in the image (25% in our case), `extra_sensor_area` is the additional area that the dye needs to cover to declare a parasite cancerous (calculated based on the given 10% area threshold for having cancer). This formulation ensures randomness in the cancerous parasites.
  - `cancer=0`: stop randomly when

    `area >= min_parasite_area + random.random()*extra_sensor_area`.

    This ensures that the dye area is greater than the area occupied by the parasite but less than the threshold for the parasite to become cancerous. This also induces randomness in dye areas of non-cancerous parasites.

3 Detecting cancer

Once the images are generated, it straightforward to find if a parasite is cancerous or not. Let A be the set of points that have parasite present (generated from the microscope) and let B be the set of points that have dye present (generated from the sensor). If $|B| \geq (1 + p_{area}) \cdot |A|$, the parasite is cancerous, otherwise not (here $|S|$ denotes the cardinality of a set S and $p_{area} = 10\%$ is given).

4 Optimization

Since the working size of images is $(10^5, 10^5)$, an internal optimization that we have implemented is as follows. Instead of initializing the grid of all zeros every time for each parasite and doing BFS by setting visited cells as 1, we initialize the grid only once. Then, when generating images for i^{th} parasite, mark visited cells as i, and keep adding cells to the BFS queue if their value in the grid is *not* i.

Once this is done, it is extremely straightforward to check for cancer, since we only need to compare the number of points in the image from the microscope and the number of points in the image from the sensor.

The entire code is given in Appendix.

Appendix

```
import random
from collections import deque

class GenerateImages(object):

    def __init__(self, num_images, grid_height = 100000, grid_width = 100000,
                 cancer_probability = 0.001, cancer_area_threshold = 0.1,
                 parasite_occupied_area_percentage = 0.25):

        self.num_images = num_images
        self.grid_height = grid_height
        self.grid_width  = grid_width
        self.parasite_occupied_area_percentage = parasite_occupied_area_percentage
        self.cancer_area_threshold   = cancer_area_threshold
        self.cancer_probability = cancer_probability

        self.grid = [[0 for _ in range(self.grid_height)] for _ in range(self.
                                                    grid_width)]

        self.microscope_images = []
        self.sensor_images = []
        self.cancerous_parasites_list = []

    def bfs_helper(self, grid, start_x, start_y, parasite_num=1, microscope=True):
        """
        do bfs in grid starting from (start_x, start_y) and use parasite_num to label
                                                    visited cells.
        microscope is True if generating a microscope image, False if generating a
                                                    sensor image.
        returns a list of all points visited, stopped using the scheme explained below
                                                     for randomness.
        """

        queue = deque([(start_x, start_y)])
        area = 0

        min_parasite_area = self.parasite_occupied_area_percentage*self.grid_height*
                                                    self.grid_width
        extra_sensor_area = self.cancer_area_threshold*self.grid_height*self.
                                                    grid_width

        cancer = 0
        prob_stop = random.random()
        if(prob_stop >= (1 - self.cancer_probability)):
            cancer = 1

        image = []

        while queue:
            x, y = queue.popleft()
            if(grid[x][y] != parasite_num):
                grid[x][y] = parasite_num  # Mark the cell as visited (1)
                image.append([x,y]) # add to image
                area += 1
```

```
            if(microscope):
                # randomly stop generating parasite image if area exceeds the
                                                        threshold
                if(area >= min_parasite_area):
                    prob_stop_micro = random.random()
                    if(prob_stop_micro >= 0.5):
                        break
            else: # if sensor image

                if(area >= min_parasite_area + (cancer+random.random())*
                                                        extra_sensor_area):
                        break

            # Generate neighbors (up, down, left, right)
            neighbours = [(x + 1, y), (x - 1, y), (x, y + 1), (x, y - 1)]
            if(not microscope):
                random.shuffle(neighbours)

            # Add unvisited neighbors to the queue
            for nx, ny in neighbours:
                if 0 <= nx < self.grid_height and 0 <= ny < self.grid_width and
                                                        grid[nx][ny] !=
                                                        parasite_num:
                     queue.append((nx, ny))

    return image

def generate_images(self):
"""
generates images and stores them
"""

    for i in range(1, self.num_images+1):

        start_x = random.randint(0, self.grid_height - 1) # x is along the rows
        start_y = random.randint(0, self.grid_width  - 1) # y is along the columns
        grid = self.grid

        microscope_image = self.bfs_helper(grid, start_x, start_y, i, microscope=
                                                    True)
        sensor_image = self.bfs_helper(grid, start_x, start_y, -i, microscope=
                                                    False) #  for sensor images use
                                                     -i to signify visited


        self.microscope_images.append(microscope_image)
        self.sensor_images.append(sensor_image)

def detect_cancer(self):
"""
returns the indices of cancerous parasites. calculated based on area covered by
                                            dye and parasite
"""

    for i in range(self.num_images):
        area_parasite = len(self.microscope_images[i])
        area_dye      = len(self.sensor_images[i])
```

```
            if(area_dye >= (1+self.cancer_area_threshold)*area_parasite):
                self.cancerous_parasites_list.append(i)
```

How to use :

```
num_images = 10000
grid_height = 100000
grid_width = 100000
cancer_probability = 0.001 # less than 0.1% have cancer
cancer_area_threshold = 0.1, # if area of dye exceeds 10% then cancer
parasite_occupied_area_percentage = 0.25 # minimum fraction of area occupied by
                                         parasite in image

obj = GenerateImages(num_images, grid_height, grid_width, cancer_probability,
                                         cancer_area_threshold,
                                         parasite_occupied_area_percentage)
obj.generate_images()
obj.detect_cancer()
```