

CS 470: Assignment 03

Programming Assignments (80%)

Your goal is to detect the bounding boxes for **white blood cells** in the BCCD dataset.

WARNING: You will have to run the script `Prepare_A03.py` to download the data.

A03.py

In this module, you must write a function, **find_WBC(image)**, that:

- Takes a color image
- Finds the white blood cells (which tend to have **blue blobs in the center** and to be **fairly large**)
- Computes the bounding box (ymin, xmin, ymax, xmax) for each white blood cell found
- Returns a list of detected bounding boxes

The way you approach the problem(s) above is largely up to you. It is assumed that this will be done via some manner of color segmentation.

I am NOT assuming that you are going to use machine learning / deep learning (since we have not really covered this in depth). HOWEVER, if you do, you are ONLY allowed to train on the training dataset, NOT the testing dataset!

Unlike some of the other assignments, **you are permitted to use OpenCV, Numpy, Scikit-Learn, scipy, and even PyTorch functionality**, including:

- numpy
 - o where
 - o expand_dims
 - o sum
 - o sqrt
 - o argmin
 - o flatten
 - o reshape
- skimage.segmentation
 - o slic
- cv2
 - o mean
 - o kmeans
 - o connectedComponents
 - o floodFill
 - o cvtColor

- Any of the edge detection approaches
- Any of the Hough approaches
- Any of the morphological approaches

Prepare_A03.py

This script must be run ONCE to download all of the BCCD data (graciously available from: https://github.com/Shenggan/BCCD_Dataset)

General_A03.py

This is the core functionality of both training programs.

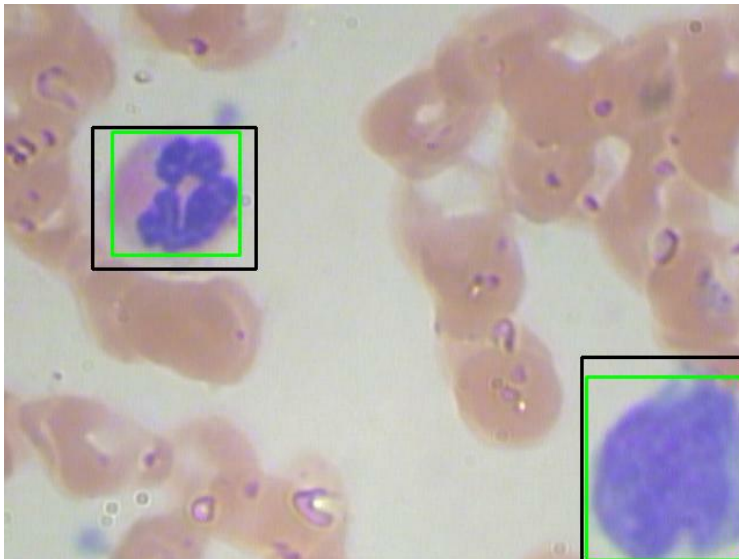
Train_A03.py

The main program (together with dataset loading and evaluation code) has been provided:

Train_A03.py.

This program:

- Loads the BCCD dataset into training and testing sets.
- Creates an "assign03/output_wbc" directory
 - If it already exists, the program will ask you if you want to overwrite it
- Calls your prediction approach on the training and testing images, computing the accuracy of the count as well as the IOU (Intersection-Over-Union) of the predicted bounding boxes.
 - It also saves the output images with ground truth (black) and predicted (green) boxes.
- Prints the metrics as well as saves them to a file, assign03/output_wbc/RESULTS_WBC.txt.



Results (20%)

After running the training programs, copy your result txt file(s) to the MAIN directory of your repository.

BONUS POINTS

For the entire assignment, bonus points will be awarded to the submission that has the following with respect to the **white blood cell detection**:

- (+%5) Best **testing** accuracy in class (provided it is over 90%)
- (+%5) Best **testing** IOU in class (provided it is over 0.4)

Example Results

The example results were generated using the following approach:

- **(Step 1) Get superpixel groups (to honor color and edge boundaries).**
 - o Let's assume the image with superpixel indices is given by *segments*.
 - o Make sure to set *start_label* parameter to zero!
 - o The number of groups found can then be extracted using:
 - `cnt = len(np.unique(segments))`
- **(Step 2) Compute mean color per superpixel**
 - o Create a numpy array to hold the mean color per superpixel:
 - `group_means = np.zeros((cnt, 3), dtype="float32")`
 - o Loop through each superpixel index (i.e., for *specific_group* in `range(cnt)`):
 - Create mask images using `np.where`:
 - `mask_image = np.where(segments == specific_group, 255, 0).astype("uint8")`
 - Add the channel dimension back in to the mask using `np.expand_dims()`
 - Use `cv2.mean` (with mask) to compute mean value per group
 - Note that `cv2.mean()` returns a 4-channel value, so you will want to slice the result:
 - o `group_means[specific_group]`
`= cv2.mean(image, mask=mask_image)[0:3]`
- **(Step 3) Use K-means on GROUP mean colors to group them into 4 color groups.**
 - o Let's assume `cv2.kmeans()` returned the following:
 - `ret` = not used
 - `bestLabels` = list of which superpixel mean colors map to which kmeans group
 - E.g., if superpixel group 3's mean color fits into kmeans group 1, then:
 - o `bestLabels[3] → 1`
 - `centers` = kmeans group center values
 - E.g., if kmeans group 1 has a color of (2,56,120), then:
 - o `centers[1] → (2, 56, 120)`

- **(Step 4) Find the k-means group with mean closest to:**
 - o White blood cells → blue (255,0,0) (REMEMBER: BGR for OpenCV)
 - o E.g., find the color in *centers* closest to target color
 - o Can use Euclidean distance color segmentation
- **(Step 5) Set that k-means group to white and the rest to black.**
 - o I.e., we only want to keep the group closest to target color
 - o E.g., if group 1 was the closest to target color, then this step should give you:
 - `centers[0] → (0,0,0)`
 - `centers[1] → (255,255,255)`
 - `centers[2] → (0,0,0)`
 - `centers[3] → (0,0,0)`
- **(Step 6) Determine the new colors for each superpixel group**
 - o Convert *centers* to unsigned 8-bit
 - o Get the new superpixel group colors:
 - `colors_per_clump = centers[bestLabels.flatten()]`
- **(Step 7) Recolor the superpixels with their new group colors (which are now just white and black)**
 - o You can use numpy broadcasting and indexing to create a new image *cell_mask*:
 - `cell_mask = colors_per_clump[segments]`
 - o **Convert *cell_mask* to grayscale!**
- **(Step 8) Use `cv2.connectedComponents` to get disjoint blobs from *cell_mask*.**
 - o Slide 14 of region slide deck
 - o Remember this function returns:
 - `retval` = number of connection components / blobs found
 - `labels` = image with labels marked
- **(Step 9) For each blob group (except 0, which is the background):**
 - o Get the COORDINATES of the pixels that belong to that group using `np.where`:
 - `coords = np.where(labels == i)`
 - o As long as coordinates are found, get the min and max x and y coordinates to get the bounding box (`ymin`, `xmin`, `ymax`, `xmax`).
 - o Add bounding box to list of bounding boxes.

You absolutely do NOT have to use this exact approach. You may opt for a subset of this idea (e.g., just use k-means or superpixels), or you may use a completely different approach.

I will recommend though that you take a good look at `np.where`, which can be very useful:

<https://numpy.org/doc/stable/reference/generated/numpy.where.html>

Submission

You must include the following in your repo:

- A03.py
- Result txt file(s) **(COPIED TO MAIN PROJECT FOLDER)**

Grading

Your OVERALL assignment grade is weighted as follows:

- 80% - Programming assignments
- 20% - Results