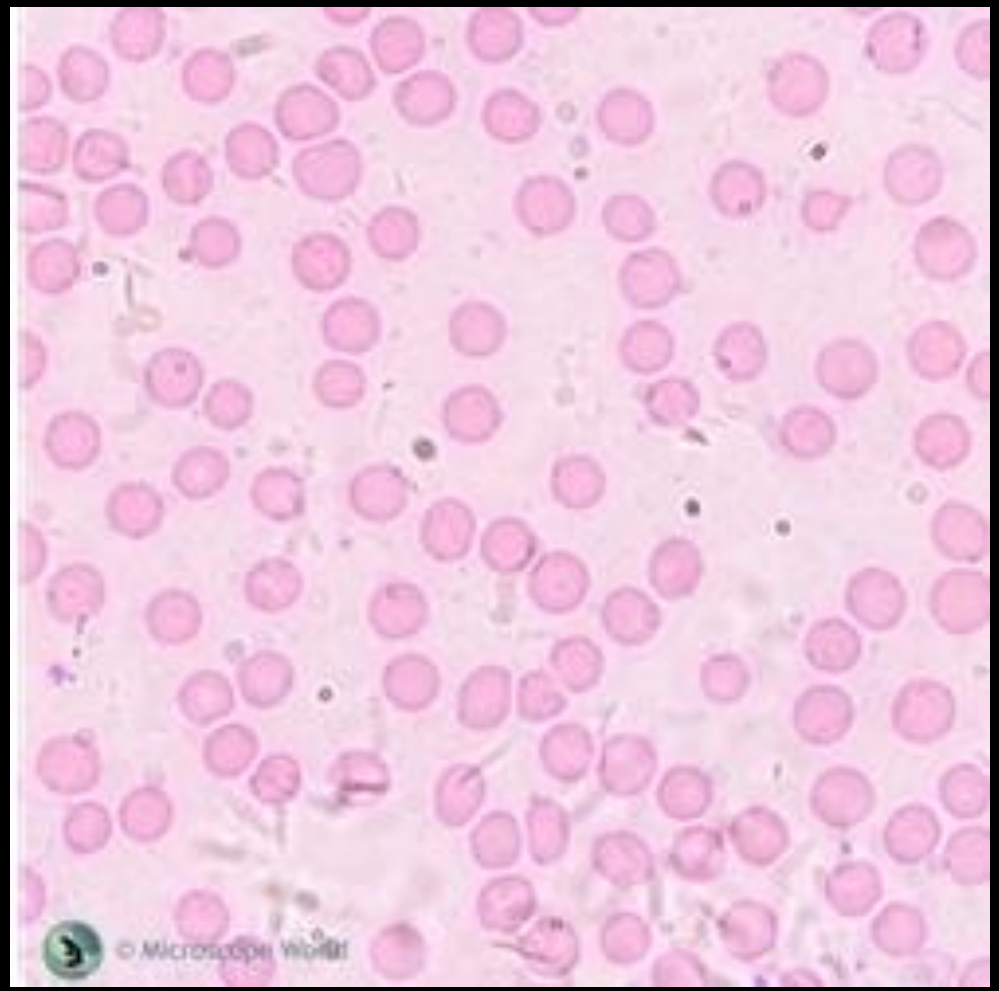
BLOB ANALYSIS

ACTIVITY 10

Krishna Lyn Delima
2014-64503

Original Image



Blood cells under the microscope

- https://www.microscopeworld.com/p-3468-microscope-resolution-explained-using-blood-cells.aspx

Blob Analysis Process:

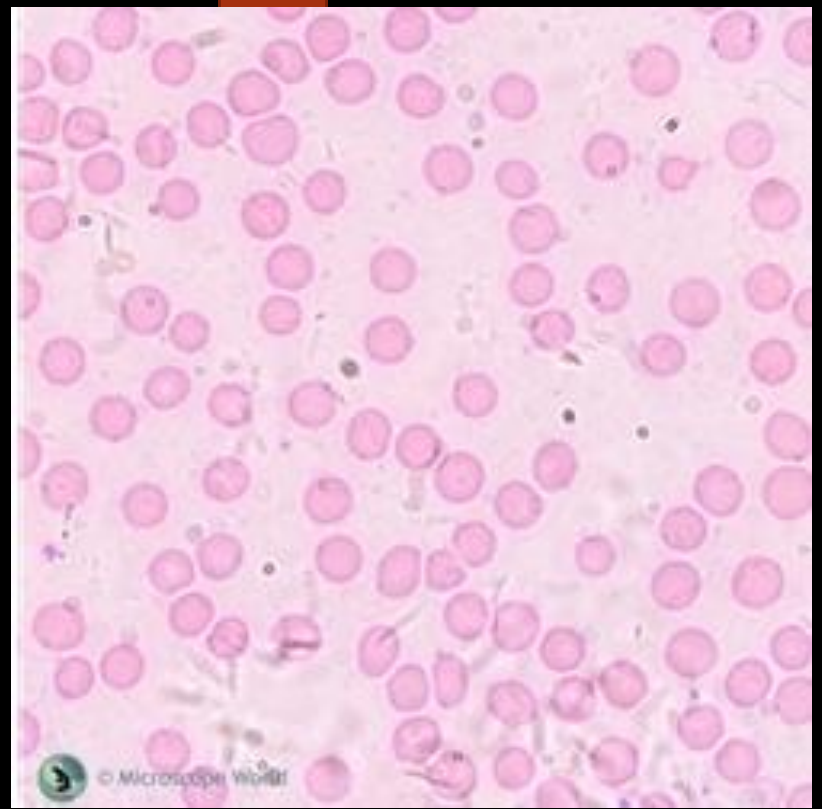
- Thresholding
- Morphological Cleaning
- Blob Detection
- Extraction of values of blobs (centroids, area, and etc.)

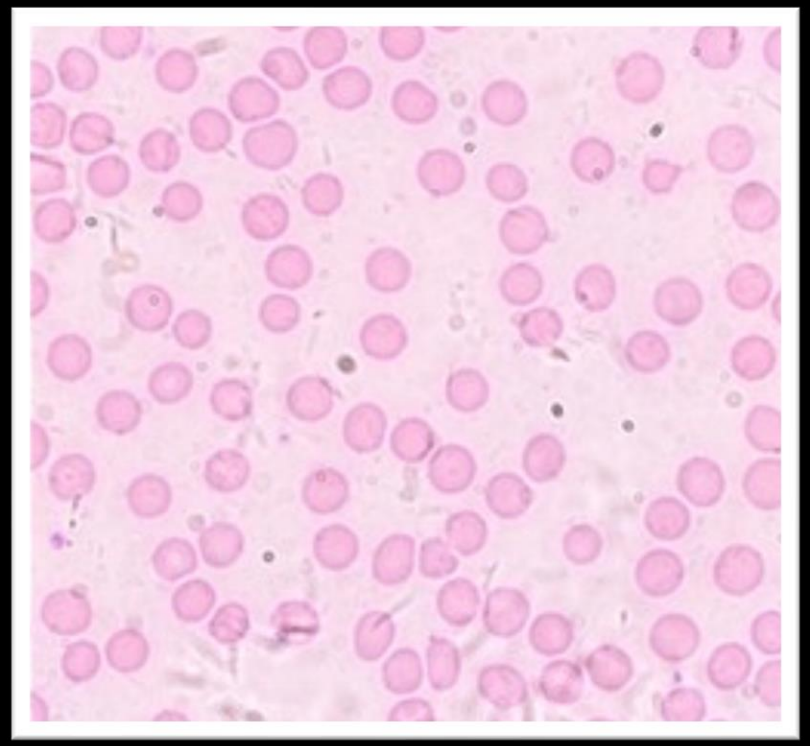
Done using:

- Jupyter Notebook (Python)
  - Packages:

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
```

Step 1 Cropping the image



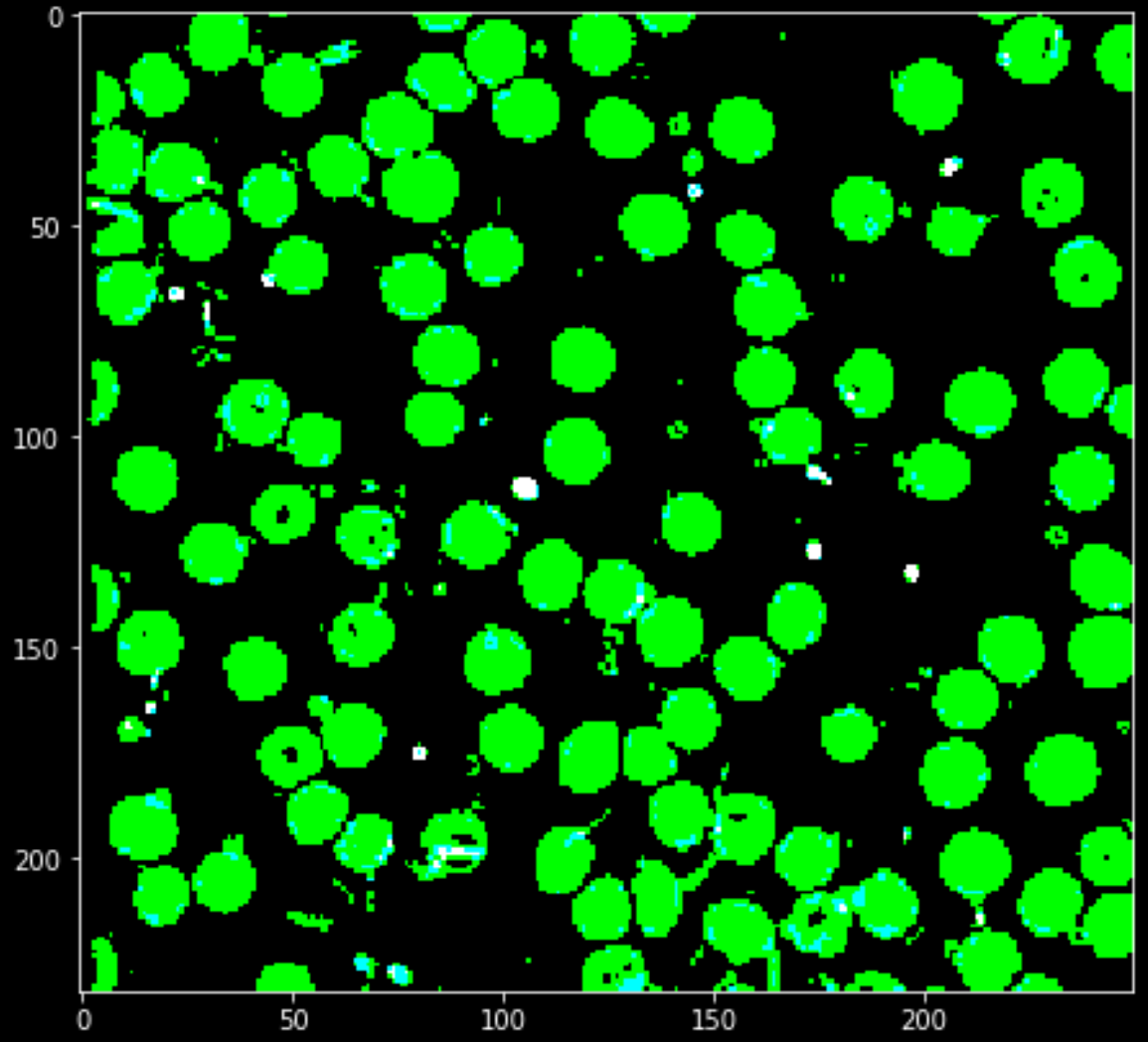


- The image was cropped to get rid of the logo

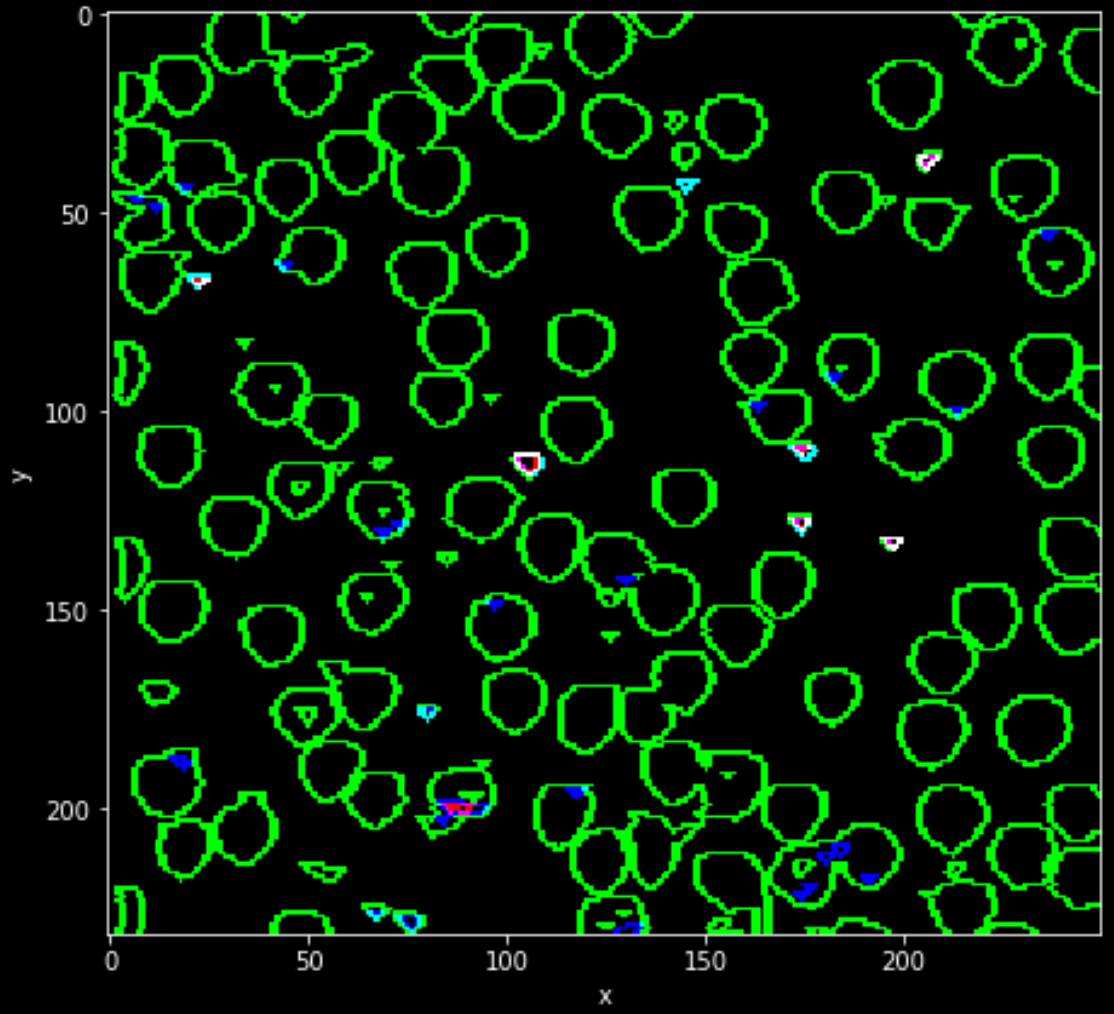
Step 2 Thresholding

```
# Binarization
thresh, BW = cv2.threshold(Blob, 200, 255, cv2.THRESH_BINARY_INV)
```

- The threshold was set at 200
- The threshold value is adjustable in order to binarize the original image and to help in morphological cleaning
- The colormap was set to ‘gray’ but it returned with green blobs (I don’t know why)
- Nevertheless, the image is read in binary despite the green hue.



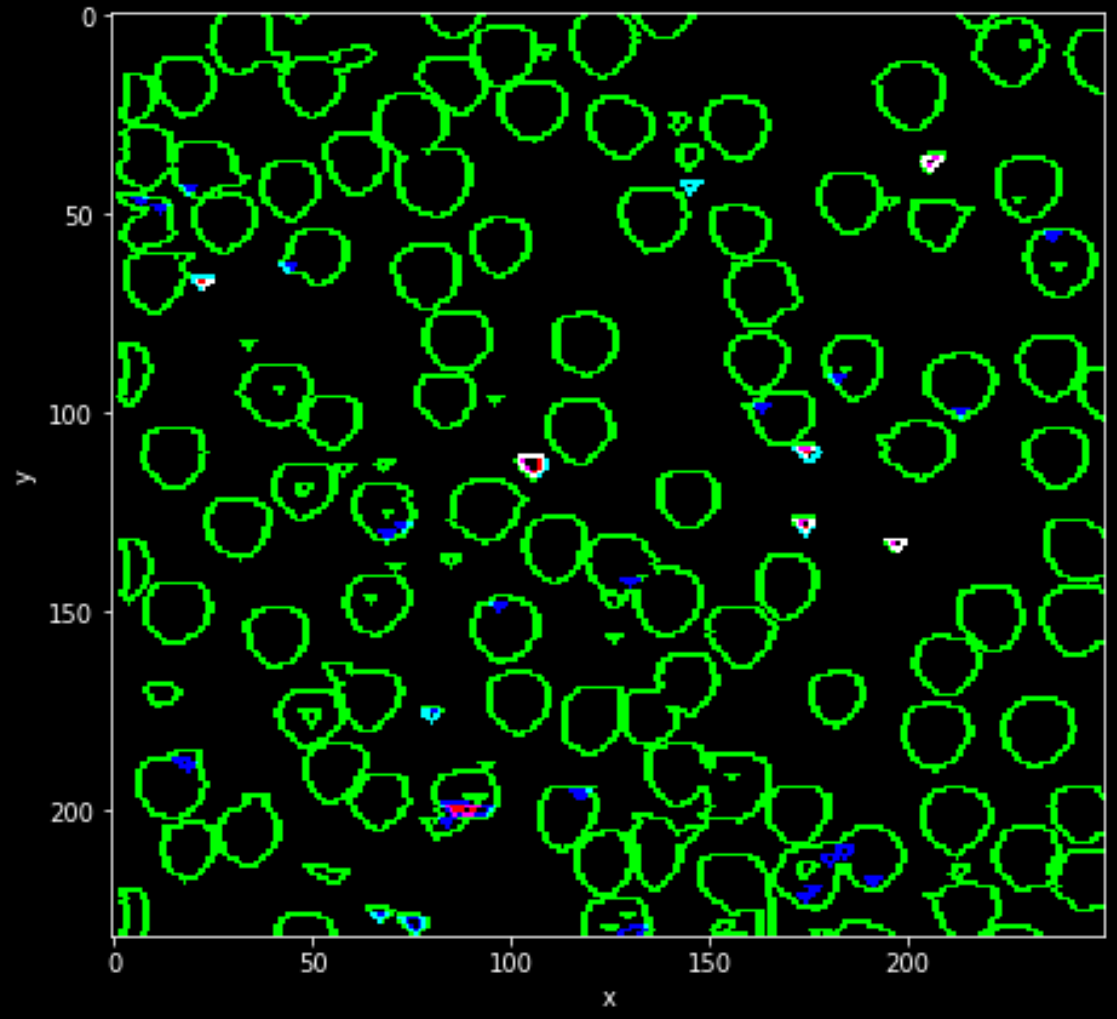
Step 3 Morphological Cleaning




```python
#ThresholdingandMorphing for 2 lines:
def morph(BW1,struct1):
    cl_morph1 = cv2.morphologyEx(BW1, cv2.MORPH_OPEN, struct1)
    op_morph1 = cv2.morphologyEx(cl_morph1, cv2.MORPH_OPEN, struct1)
    return op_morph1

def open_morph(BW1,struct1):
    op_morph1 = cv2.morphologyEx(BW1, cv2.MORPH_OPEN, struct1)
    return op_morph1

def close_morph(BW1,struct1):
    cl_morph1 = cv2.morphologyEx(BW1, cv2.MORPH_OPEN, struct1)
    return cl_morph1

def grad_morph(BW1,struct1):
    grad_morph1 = cv2.morphologyEx(BW1, cv2.MORPH_GRADIENT, struct1)
    return grad_morph1

def dil_morph(BW1,struct1,iterations):
    dil_morph1 = cv2.dilate(BW1,struct1,iterations = 1)
    return dil_morph1

def err_morph(BW1,struct1,iterations):
    err_morph1 = cv2.erode(BW1,struct1,iterations = 1)
    return err_morph1
```

- I have tried different Morphological operations in cleaning (separating the blobs) the image.

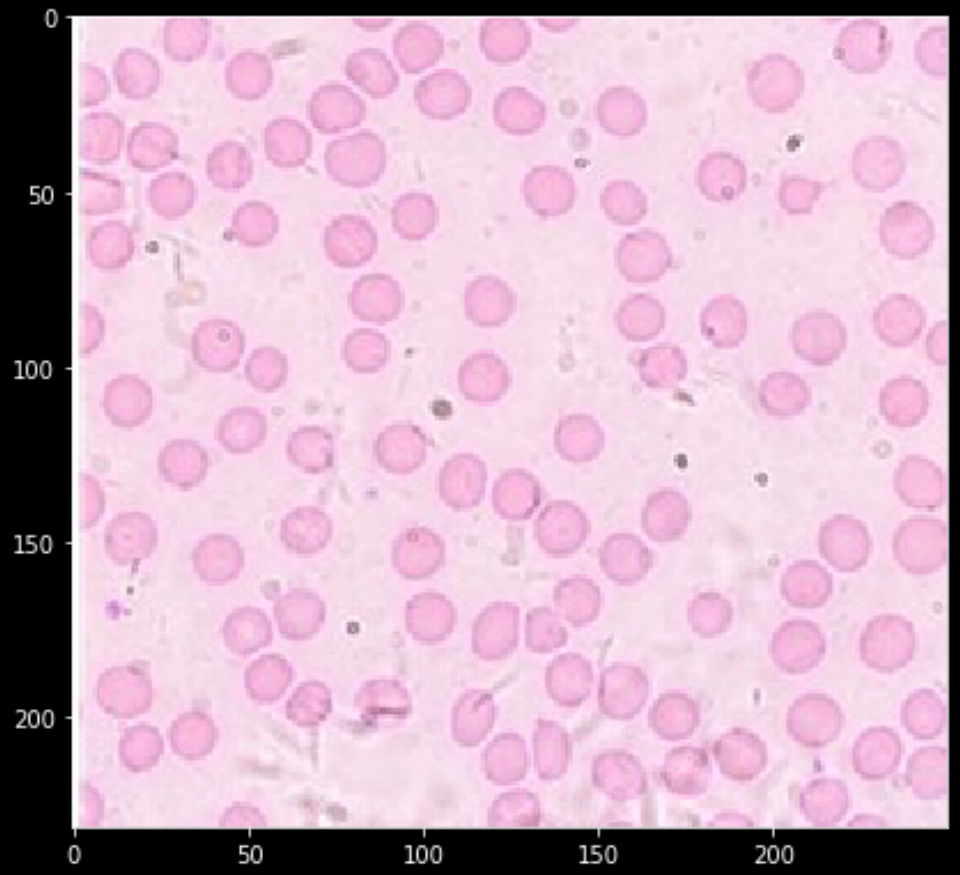
Step 3 Morphological Cleaning

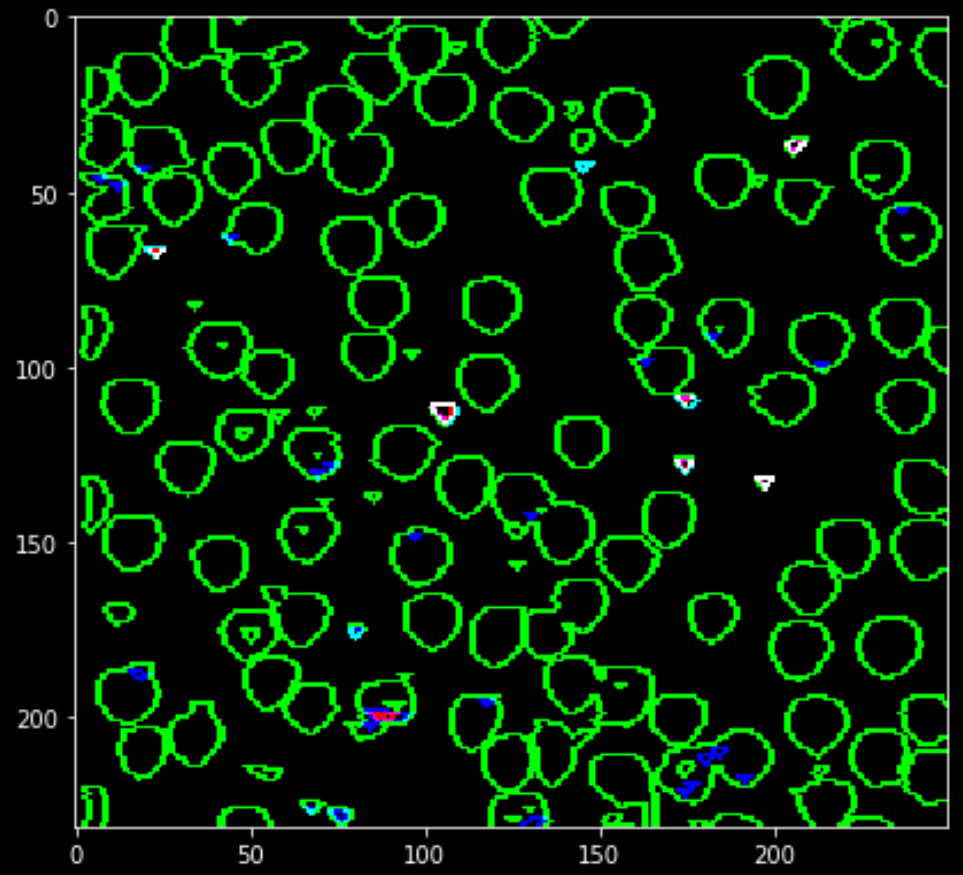



```
# Binarization
thresh, BW = cv2.threshold(Blob, 200, 255, cv2.THRESH_BINARY_INV)
# Structure element
struct1 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(3,2))
# Opening to gradient
Op1 = open_morph(BW,struct1)
OG = grad_morph(Op1,struct1)
```

- I decided to use OPENING ops and use the resulting image for the MORPHOLOGICAL GRADIENT ops.
- The structure element used was an ellipse with 3x2 size.
- This resulted to the image on the left.

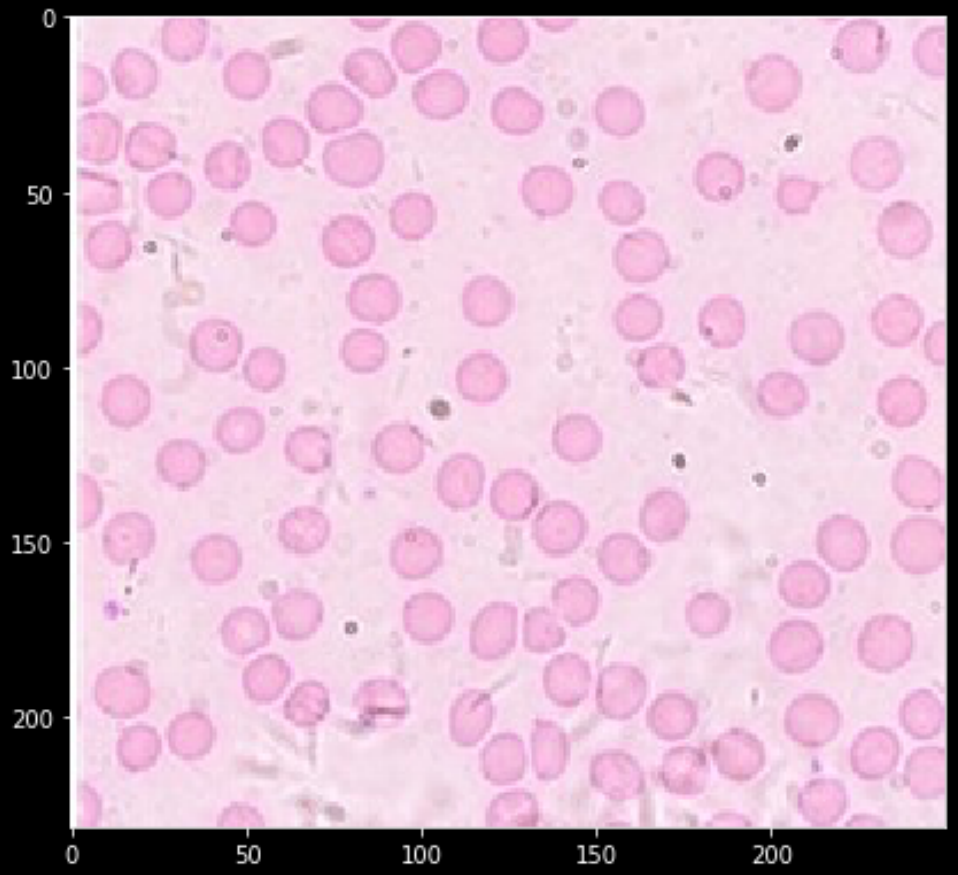
Step 3 Result of Morphological Cleaning

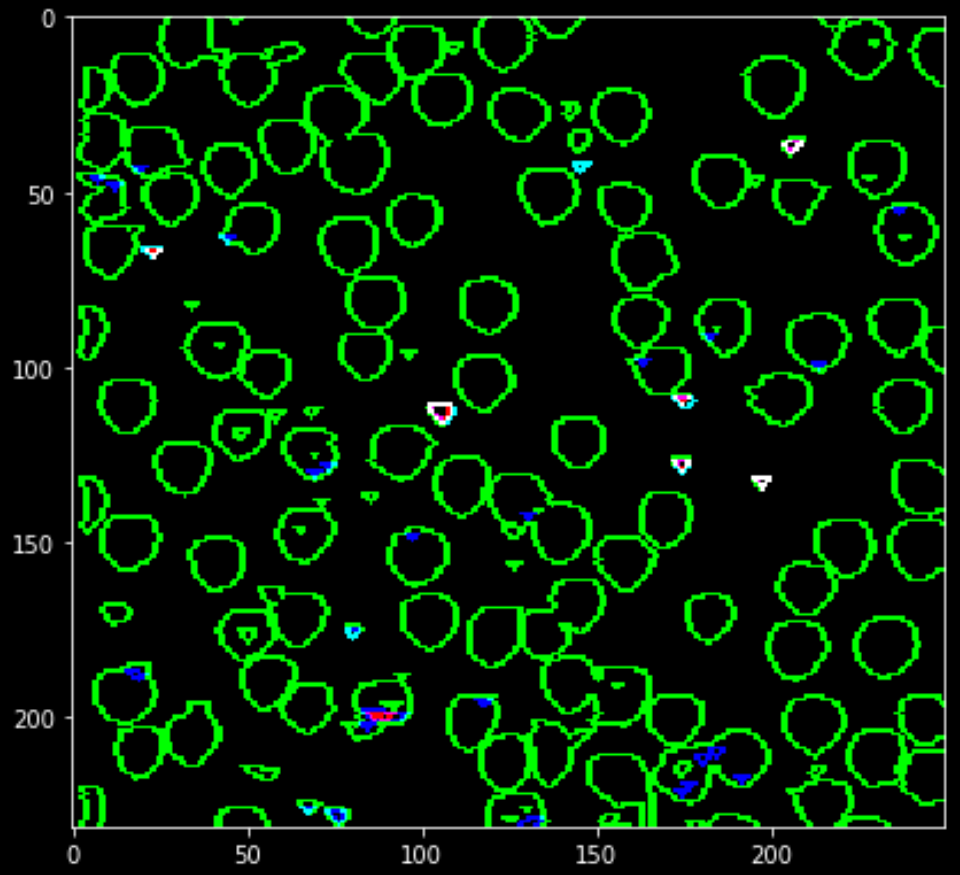




- OPEN ops got rid of unwanted blobs in the image. However, there were some that got integrated into the blobs on the right.

Step 3 Result of Morphological Cleaning





- I used GRADIENT ops to see the boundaries of the blobs better.

Step 4 Blob Detection

```
# Setup SimpleBlobDetector parameters.
params = cv2.SimpleBlobDetector_Params()
# Change thresholds
params.minThreshold = 0;
params.maxThreshold = 255;
# Filter by Area.
params.filterByArea = True
params.minArea = 3
# Filter by Circularity
params.filterByCircularity = True
params.minCircularity = 0.1
# Filter by Convexity
params.filterByConvexity = True
params.minConvexity = 0.9
# Filter by Inertia
params.filterByInertia = True
params.minInertiaRatio = 0.05
detector = cv2.SimpleBlobDetector_create(params)
# Detect blobs.
im = OG
keypoints = detector.detect(im)
print(len(keypoints))
# Draw detected blobs as red circles.
# cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS ensures the size of the circle corresponds to the size of blob
im_with_keypoints = cv2.drawKeypoints(im, keypoints, np.array([]), (0,0,255), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
# Show keypoints
cv2.imwrite("Keypoints.jpg", im_with_keypoints)
```

- SimpleBlobDetector_Params() was used for Blob Detection

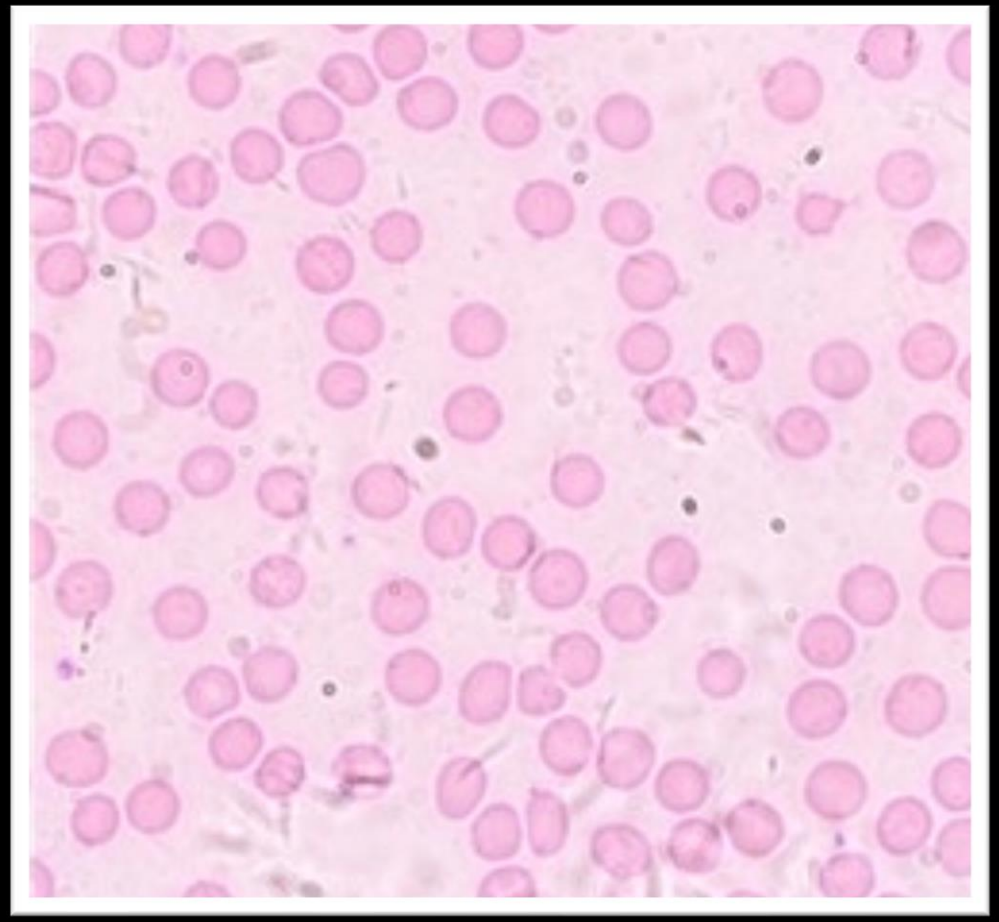
- Taken from: https://www.microscopeworld.com/p-3468-microscope-resolution-explained-using-blood-cells.aspx

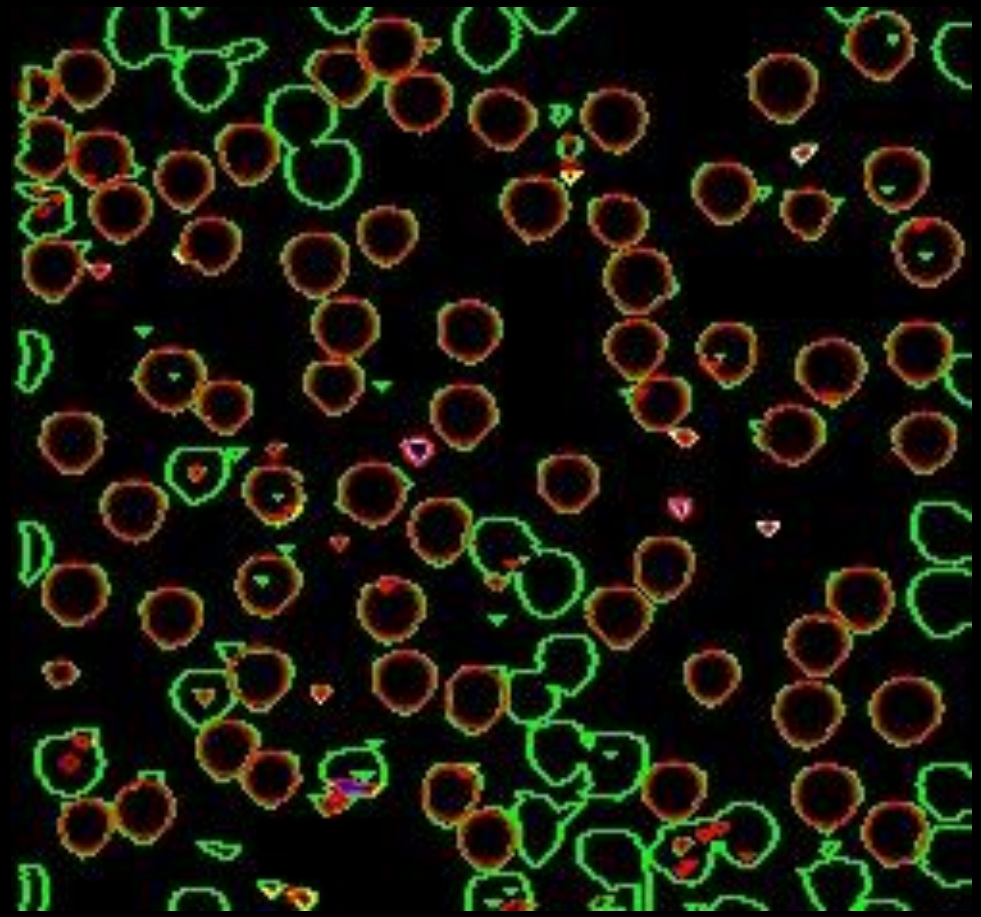
Step 4 Blob Detection

```
# Setup SimpleBlobDetector parameters.
params = cv2.SimpleBlobDetector_Params()
# Change thresholds
params.minThreshold = 0;
params.maxThreshold = 255;
# Filter by Area.
params.filterByArea = True
params.minArea = 3
# Filter by Circularity
params.filterByCircularity = True
params.minCircularity = 0.1
# Filter by Convexity
params.filterByConvexity = True
params.minConvexity = 0.9
# Filter by Inertia
params.filterByInertia = True
params.minInertiaRatio = 0.05
detector = cv2.SimpleBlobDetector_create(params)
# Detect blobs.
im = OG
keypoints = detector.detect(im)
print(len(keypoints))
# Draw detected blobs as red circles.
# cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS ensures the size of the circle corresponds to the size of blob
im_with_keypoints = cv2.drawKeypoints(im, keypoints, np.array([]), (0,0,255), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
# Show keypoints
cv2.imwrite("Keypoints.jpg", im_with_keypoints)
```

- Parameters set were minimum area, circularity, convexity, and inertia.
- Altering these values enabled me to detect 90 blobs

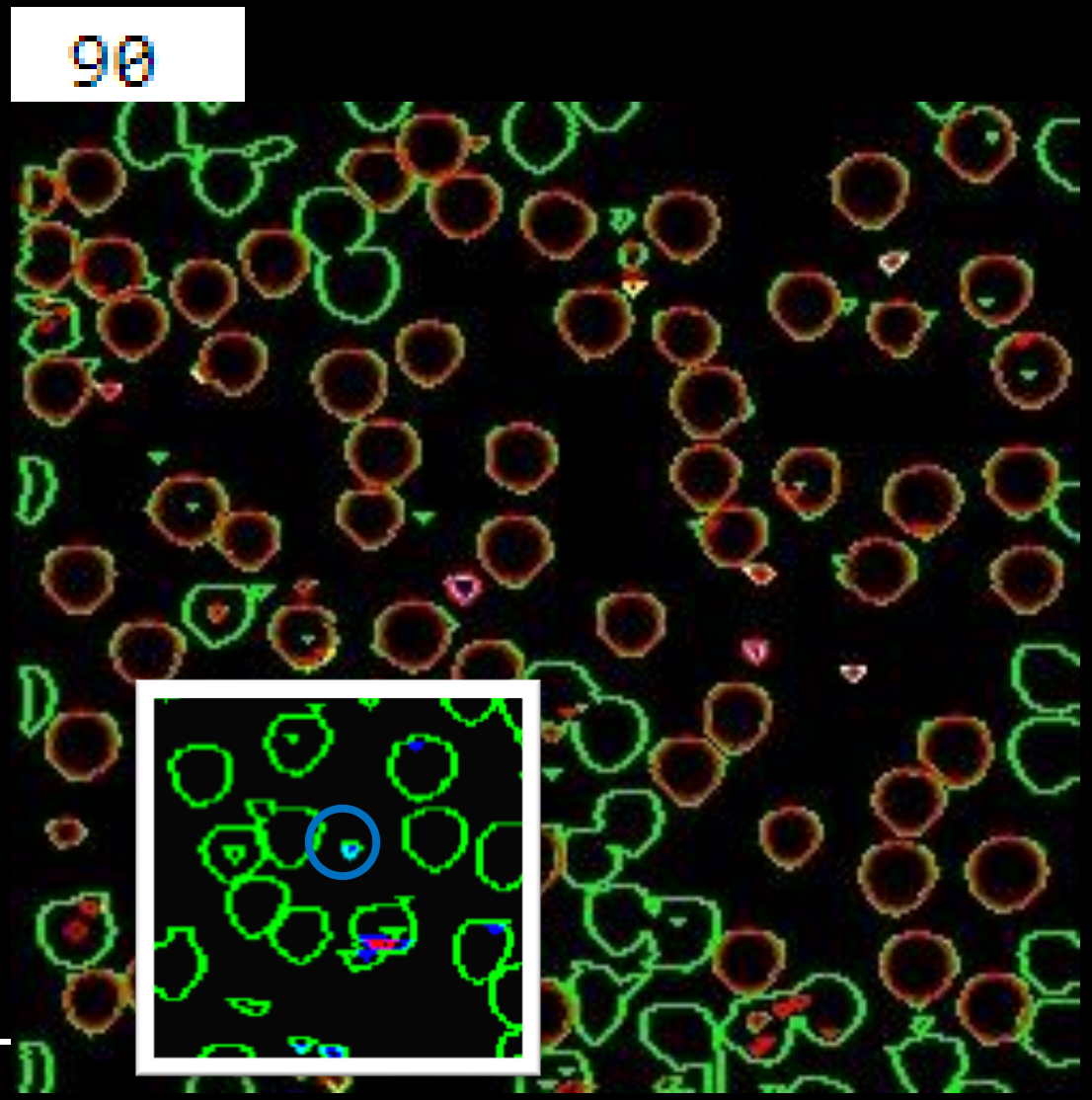
Step 3 Result of Morphological Cleaning





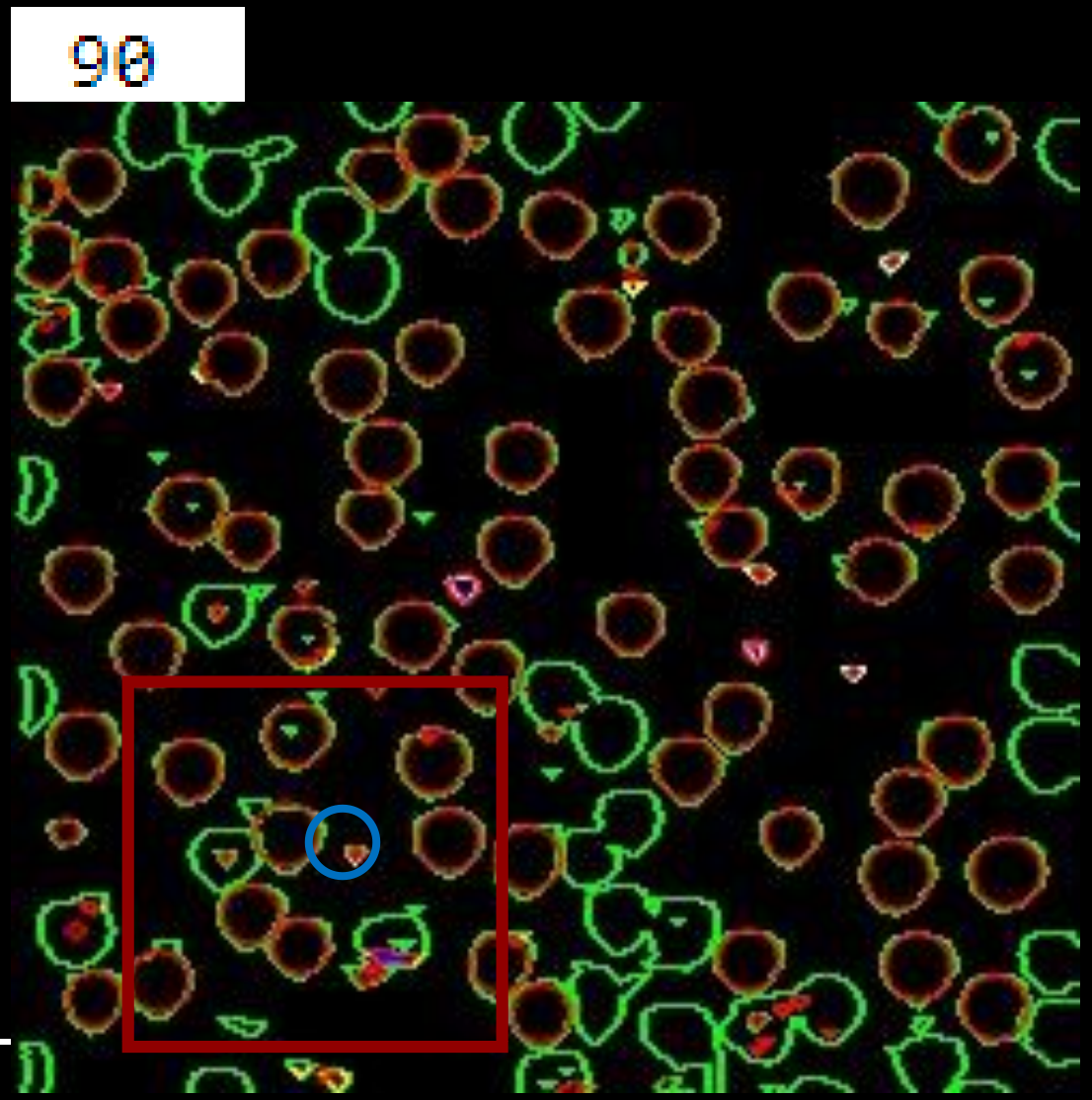
- 90 Blobs were detected out of 120 blobs (as counted manually)

Step 4 Blob Detection



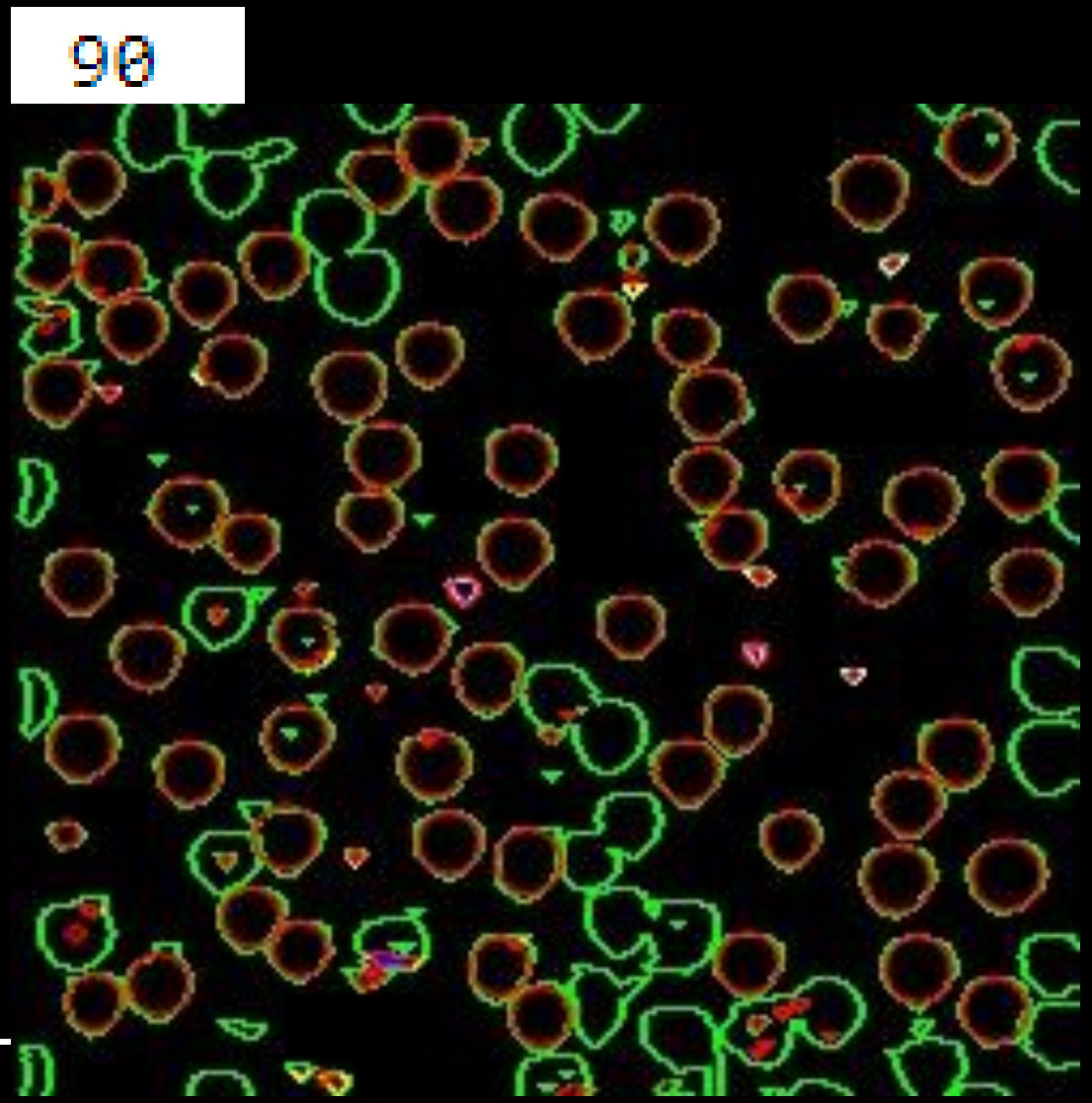
- 90 blobs were detected by the algorithm used
- Note that some of the small blobs with red aren't included – the color was due to processing by the algorithm where (blue turned to red)
- In white frame (original)

Step 4 Blob Detection



- 90 blobs were detected by the algorithm used
- Note that some of the small blobs with red aren't included – the color was due to processing by the algorithm where (blue turned to red)
- In white frame (original)
- In red frame (resulting blob detections)
- In blue circle : An example of a small blob in red that's not included

Step 4 Extracting values

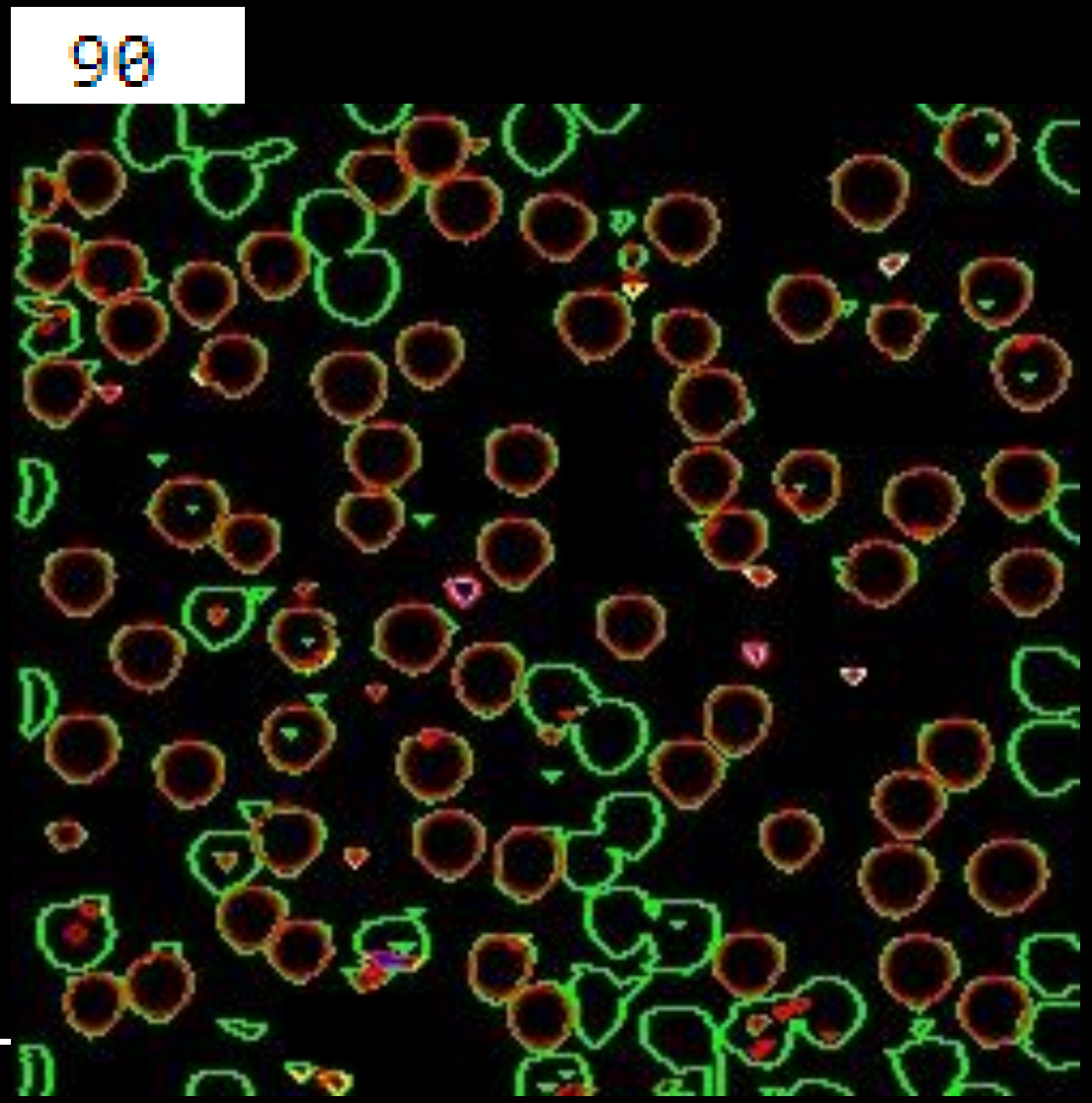



```
# Detect blobs.
im = OG
keypoints = detector.detect(im)
```

- Keypoints contain the coordinates of these blobs.
- These coordinates were extracted by using .pt .

```
dia = []
for keypoint in keypoints:
    x = keypoint.pt[0]
    y = keypoint.pt[1]
    s = keypoint.size
    print(x,y)
    ar = (np.pi)*((s/2)**2)
#    print(ar)
    dia.append(ar)
```

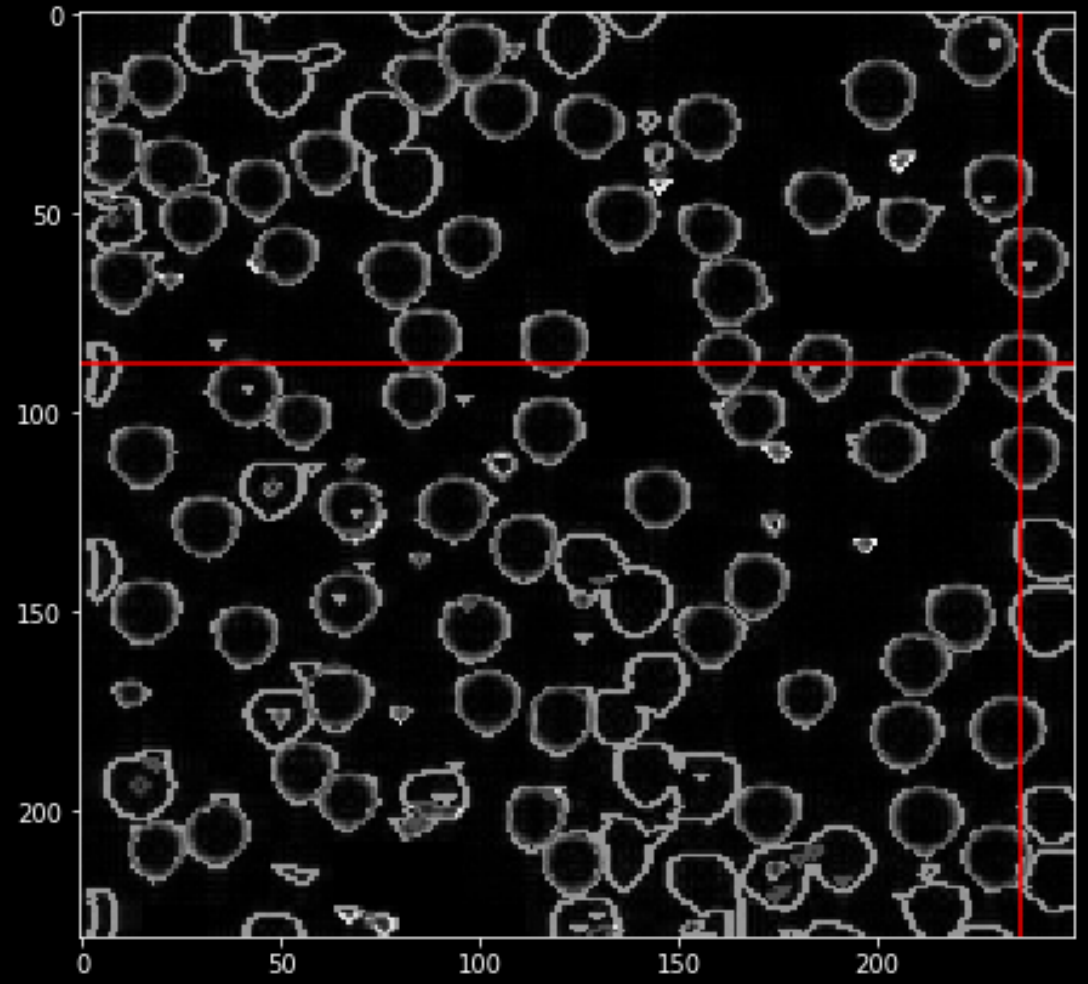
Step 4 Extracting values




```
dia = []
for keypoint in keypoints:
    x = keypoint.pt[0]
    y = keypoint.pt[1]
    s = keypoint.size
    print(x,y)
    ar = (np.pi)*((s/2)**2)
#    print(ar)
    dia.append(ar)
```

- x and y was extracted using .pt
- The size –which pertains the diameter of each blob - was taken
- Using these values, the area or average cell size was calculated

Step 4 Extracting values



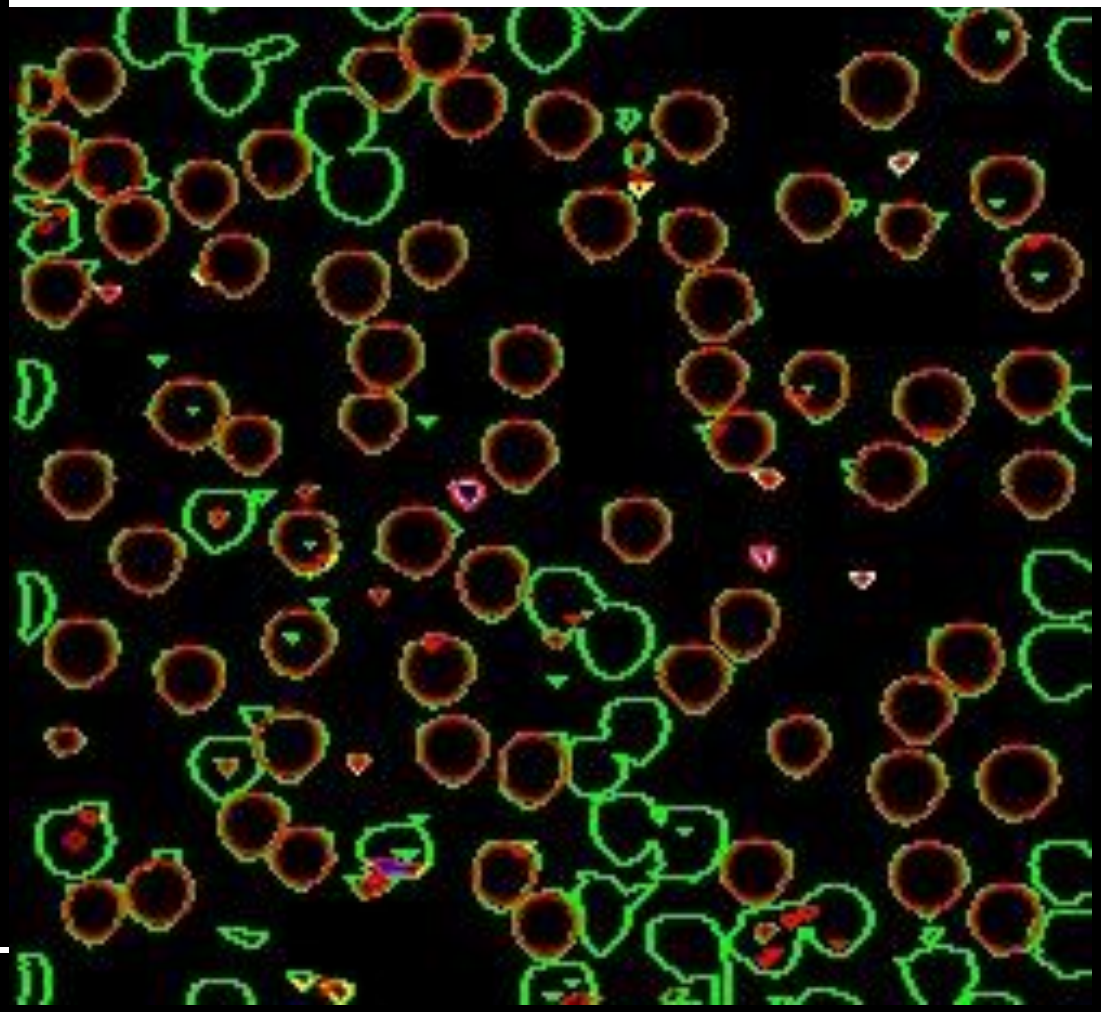

```
dia = []
for keypoint in keypoints:
    x = keypoint.pt[0]
    y = keypoint.pt[1]
    s = keypoint.size
    print(x,y)
    ar = (np.pi)*((s/2)**2)
#    print(ar)
    dia.append(ar)
```

- Keypoints x and y actually yield the centroid coordinates of each blob detected as shown at these coordinates:

```
x:  235.79809570312 y:  87.94000244140625
```

Step 4 Extracting values

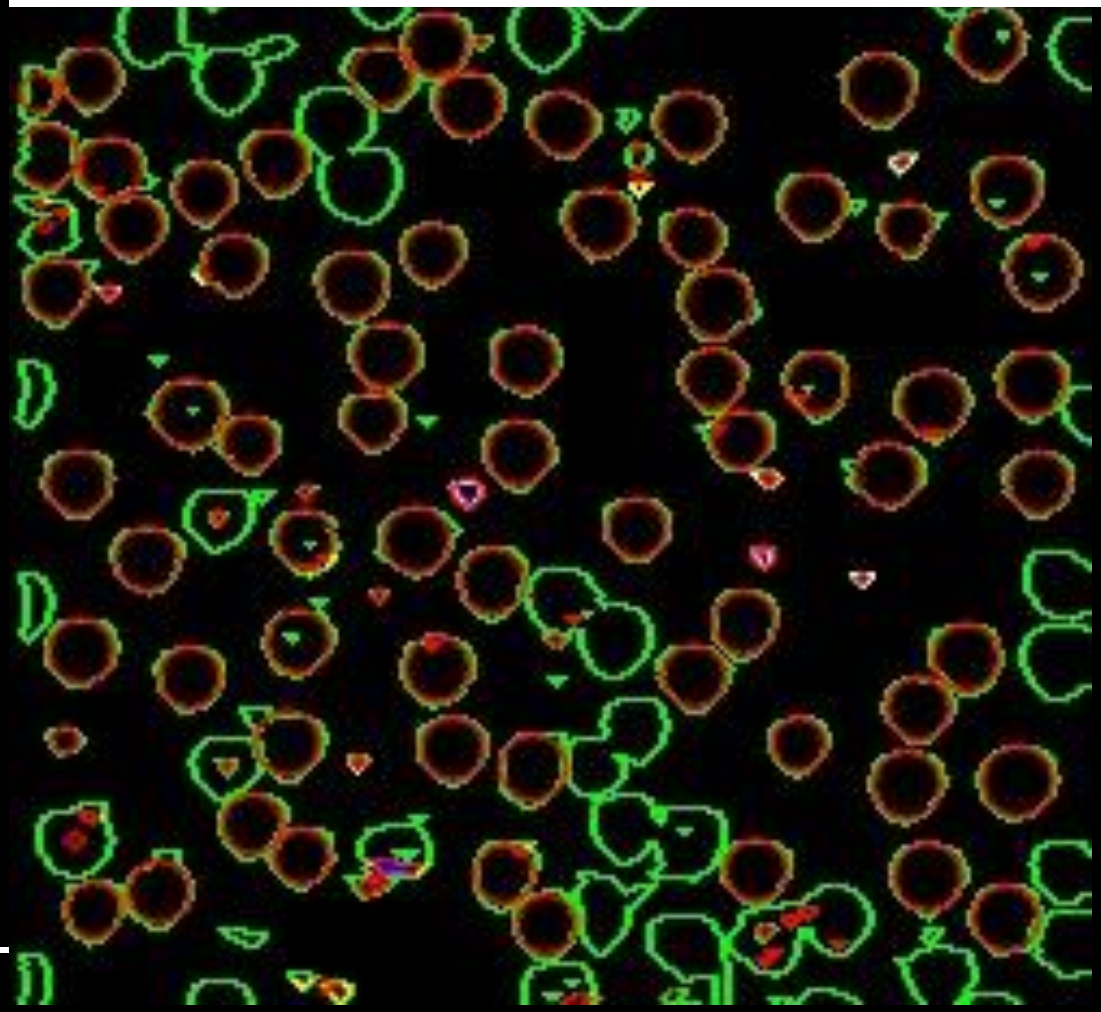
```
Area of the Image:  58000
Best Estimate of Area:  124.26412530349536  +/-  70.04307188443055
```



- The average area was 124.26 sq. px.
  - With a standard deviation of 70.04

Step 4 Extracting values

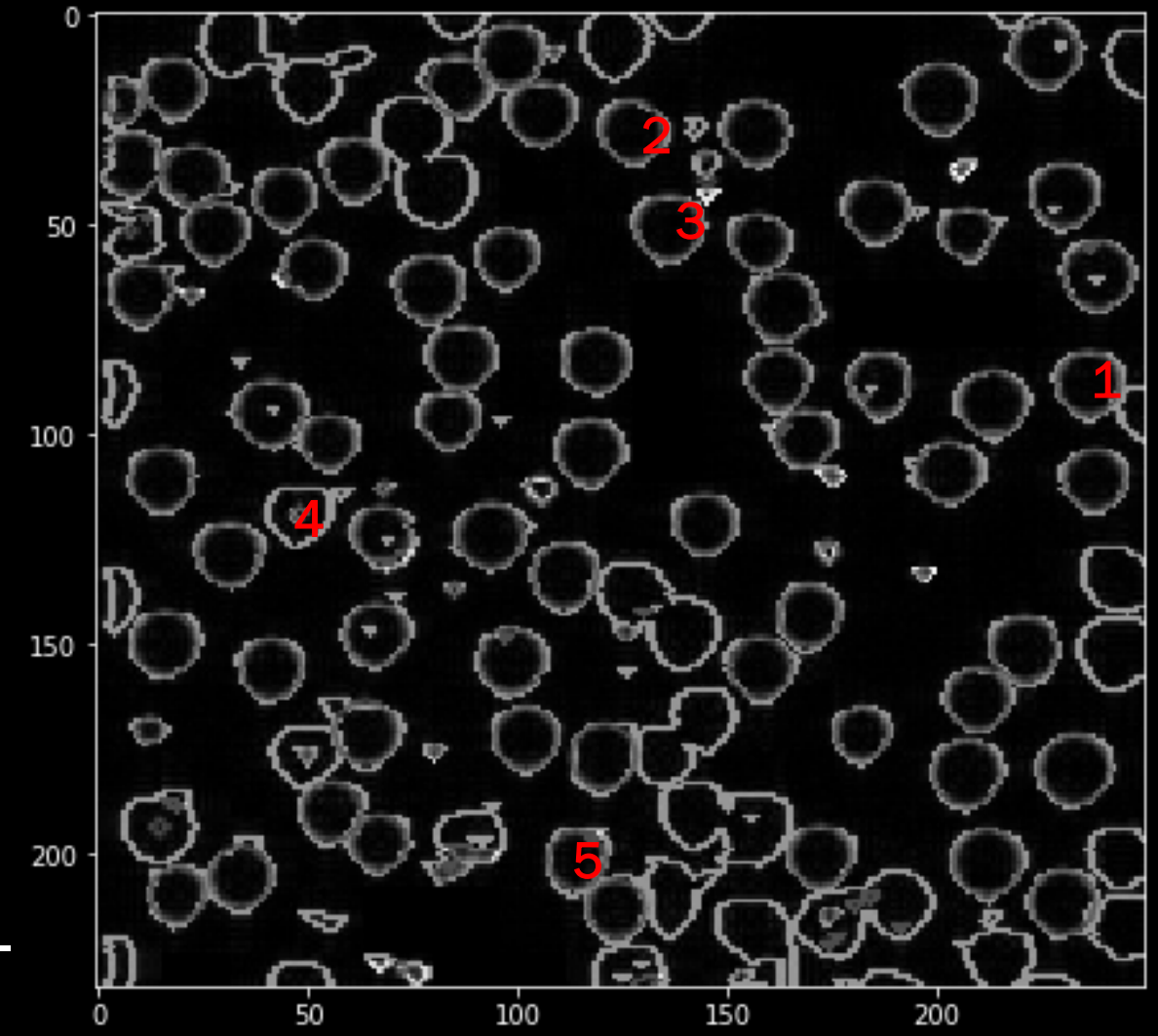
```
Area of the Image:  58000
Best Estimate of Area:  124.26412530349536  +/-  70.04307188443055
```



- Apparently, the blobs present should be more than 90.
- However, comparing the obtained average area of the blobs to the overall area of the image, this would yield a ratio of 0.002.

Step 5 Getting other values

- Since there are 90 Blobs, I'll choose 4 blobs and compare their values



	X	Y
1	235.798095703125	87.94000244140625
2	127.30815887451172	28.08793067932129
3	135.87271118164062	50.81135559082031
4	113.91008758544922	201.30072021484375

Step 5 Getting other values

- Since there are 90 Blobs, I'll choose 4 blobs and compare their values for Area and Perimeter

	X	Y	Diameter	Area	Perimeter
1	235.7980957	87.94000244	14.861644	173.4696798	46.68923095
2	127.3081589	28.08793068	14.671052	169.048917	46.09046912
3	135.8727112	50.81135559	15.403109	186.3402338	48.39029281
4	113.9100876	201.3007202	13.651566	146.3709188	42.88765805

Step 5 Getting other values

	X	Y	Diameter	Area	Perimeter
1	235.7980957	87.94000244	14.861644	173.4696798	46.68923095
2	127.3081589	28.08793068	14.671052	169.048917	46.09046912
3	135.8727112	50.81135559	15.403109	186.3402338	48.39029281
4	113.9100876	201.3007202	13.651566	146.3709188	42.88765805

Conclusion

- Blobs were detected successfully- however, not all blobs from the original image was detected.
- This method was fairly successful in taking the area and perimeter of each blob.
- 90 blobs out of 120 blobs were detected
  - This was due to inaccurate morphological cleaning
  - There were blobs that failed to separate

* *Pointssss*

- TC : 5
- QP : 5
- IN : 1.....??

- This was so much fun ☺! Thank you!