COMP 204

Introduction to image analysis with scikit-image (part three)

Mathieu Blanchette, based on slides from Christopher J.F. Cameron and Carlos G. Oliver

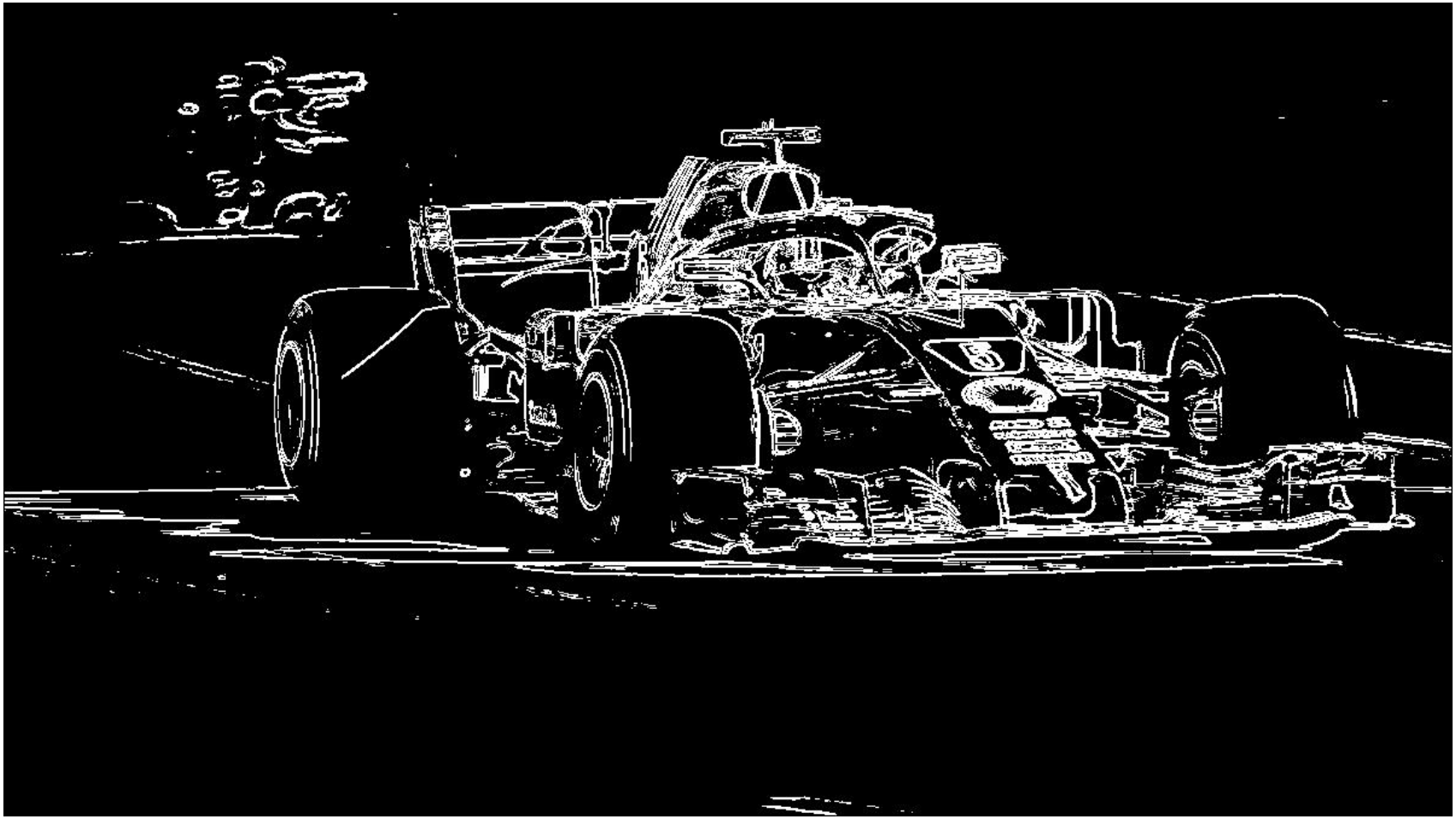
Edge detection

Goal: Identify regions of the image that contain sharp changes in colors/intensities.
Why? Useful for

- delineating objects (image segmentation)
- recognizing them (object recognition)
- etc.

Edge detection

Edge detection



Edge detection

What's an edge in an image?

Horizontal edge at row i:

- $image[i-1, j]$ is very different from $image[i+1, j]$

Vertical edge at column j:

- $image[i, j-1]$ is very different from $image[i, j+1]$

Idea: To determine if an RGB pixel (i, j) belongs to an edge:
For each color $\in \{R, G, B\}$:

- $L_x[color] = image[i, j-1, color] - image[i, j+1, color]$
- $L_y[color] = image[i-1, j, color] - image[i+1, j, color]$
- gradient[color] $= \sqrt{L_x[color]^2 + L_y[color]^2}$

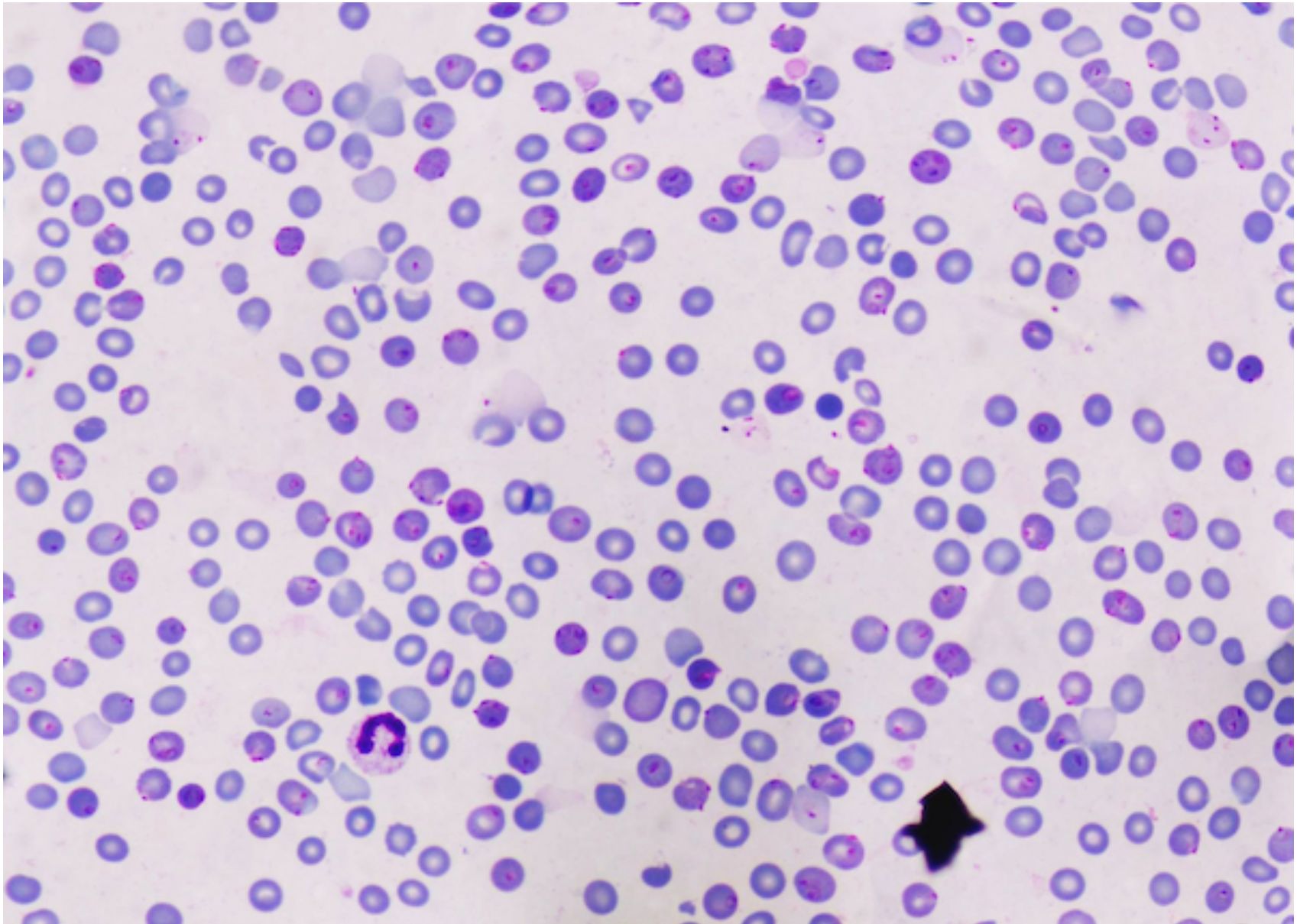
edginess $= \sqrt{gradient[R]^2 + gradient[G]^2 + gradient[B]^2}$
if edginess $>$ some_threshold, then pixel (i, j) belongs to an edge

Edge detection

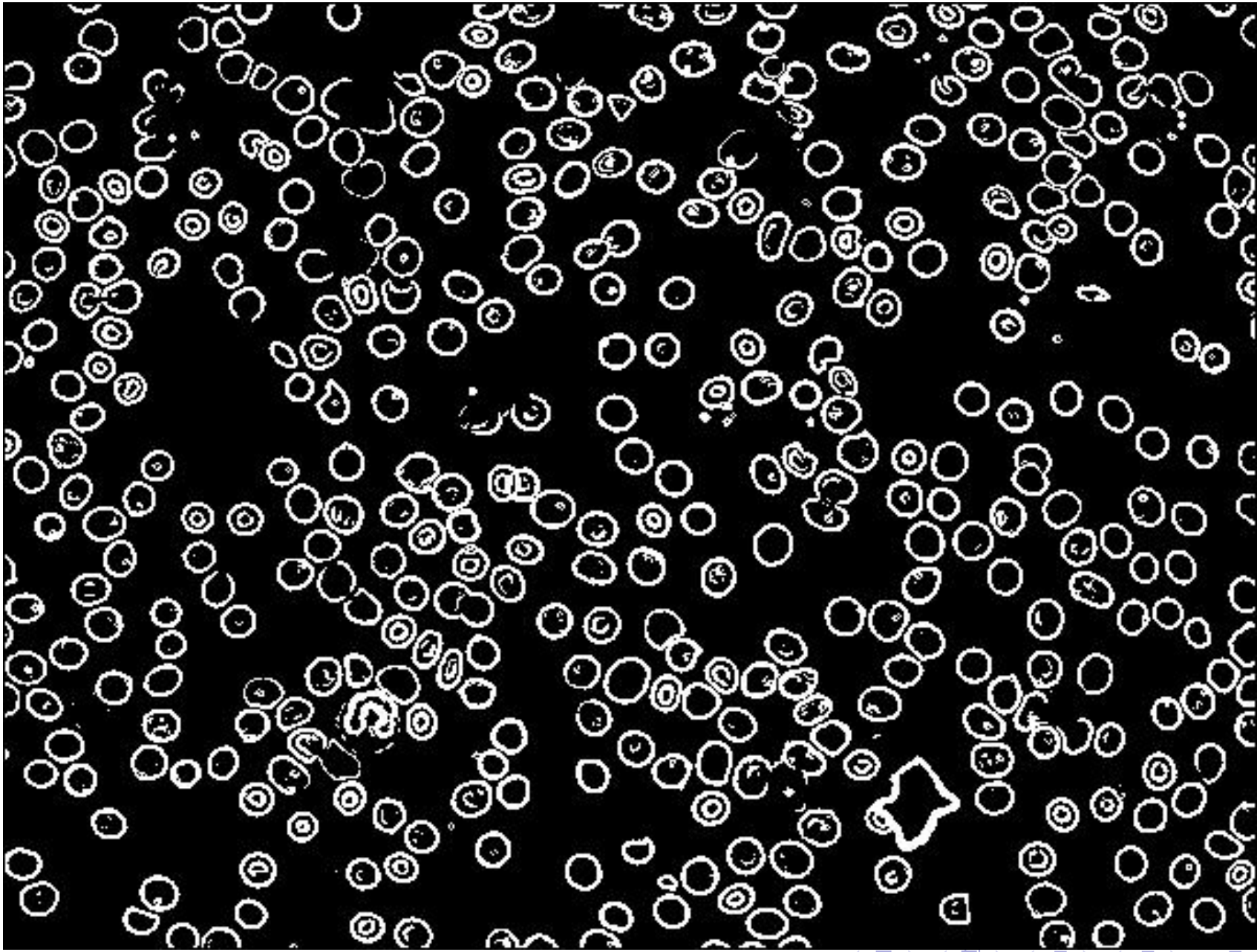
```
def detect_edges(im, min_gradient=50):
    """
    Args:
       im: The image on which to detect edges
       min_gradient: The minimum gradient value
                     for a pixel to be called an edge
    Returns: An new image with edge pixels set to white,
             and everything else set to black
    """
    n_row, n_col, colors = image.shape

    # create a empty empty of the same size as the original
    edge_image = np.zeros( (n_row,n_col,3), dtype=np.uint8)

    for i in range(1,n_row-1):  # avoid the first/last row
        for j in range(1,n_col-1):  # and first/last col
            grad=[0,0,0]
            for c in range(3):  # for each color
                Lx = float(im[i-1,j,c])-float(im[i+1,j,c])
                Ly = float(im[i,j-1,c])-float(im[i,j+1,c])
                grad[c] = math.sqrt(Lx**2+Ly**2)
            norm = math.sqrt(grad[0]**2 + grad[1]**2 \
                             + grad[2]**2)
            if (norm > min_gradient):
                edge_image[i,j] = (255,255,255)
    return edge_image
```

Analysis of microscopy images

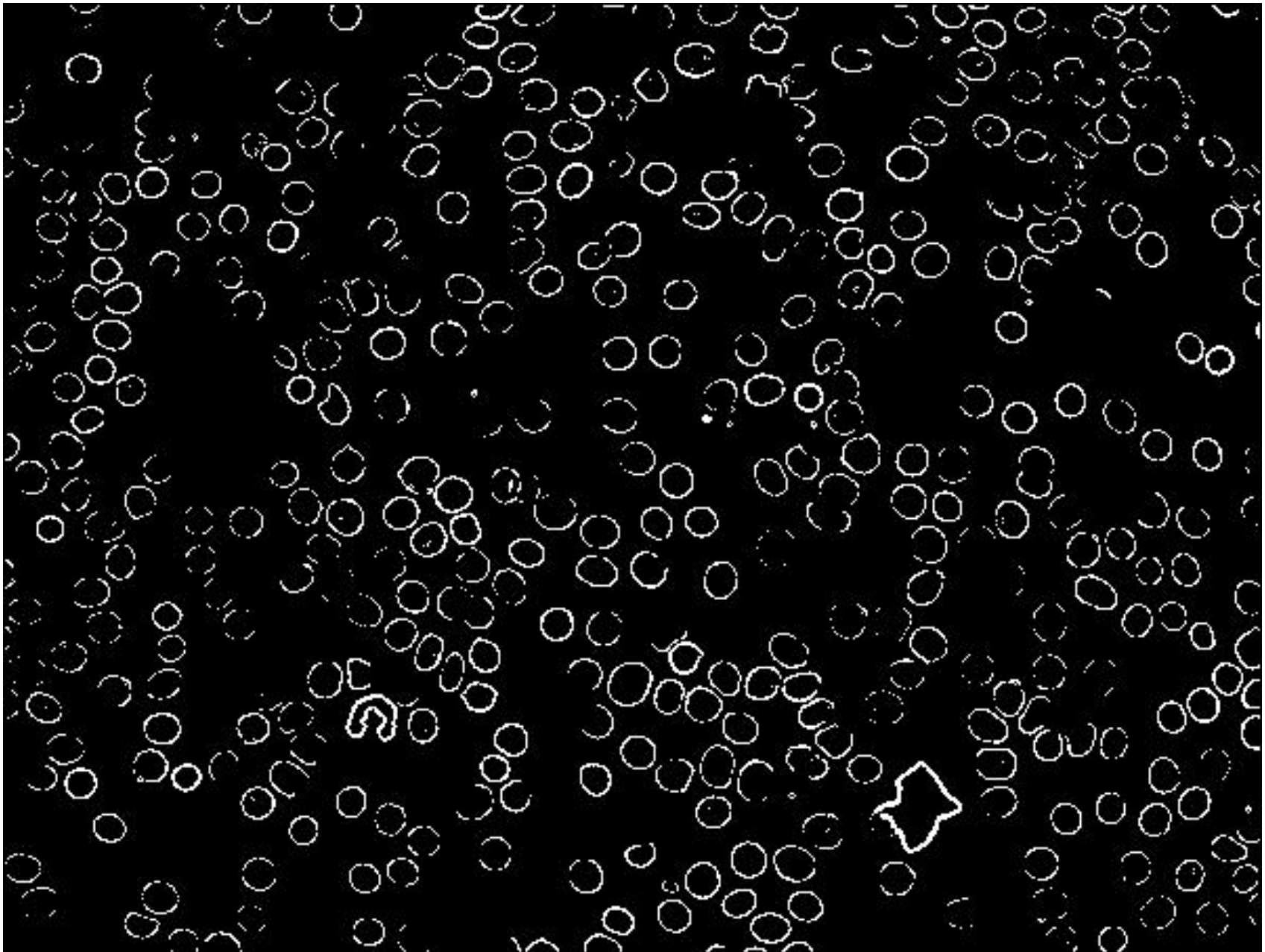
Cells (purple "circles") are infected by Plasmodium falciparum (small red dots).



Edge detection (threshold = 60)



Edge detection (threshold = 120)



Edge detection

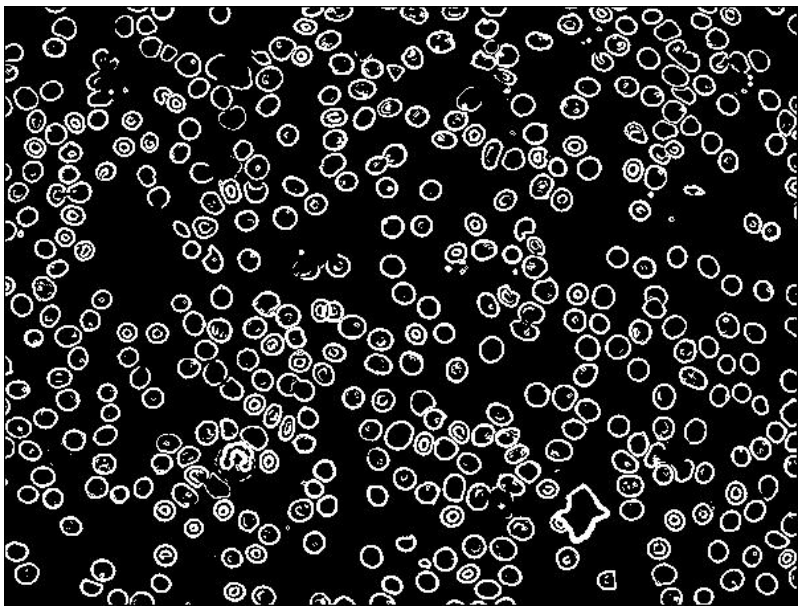
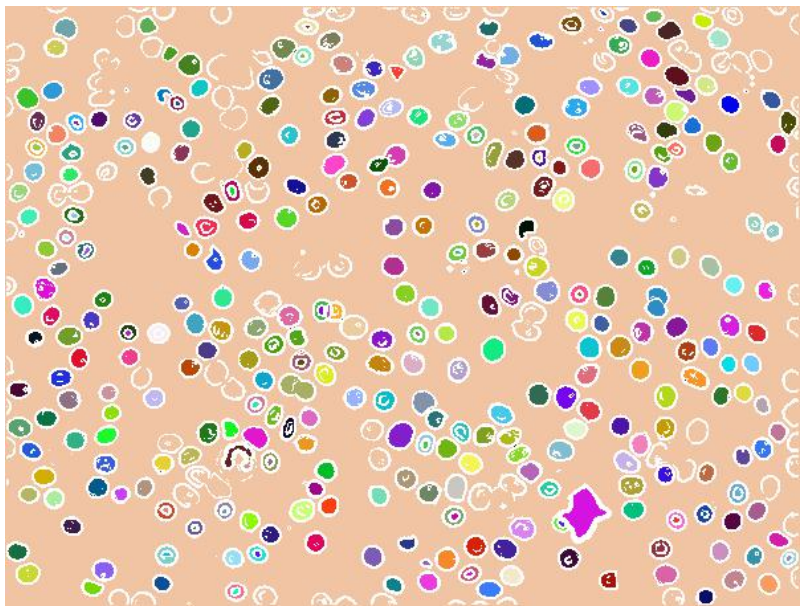
Skimage has many edge detection algorithms:
`http://scikit-image.org/docs/0.5/auto_examples/plot_canny.html`

Counting/annotating cells

What if we want to automatically identify/count cells in the image?

Idea:

1. Find edges in the image
2. Identify closed (encircled) shapes within the edge image


to


Each closed shape is assigned a different color.
Number of closed shapes (= approximation to cell count) is calculated.

Seed filling algorithm

How to take an edge image and fill-in each closed shape?
Seed filling (aka flood filling) algorithm:

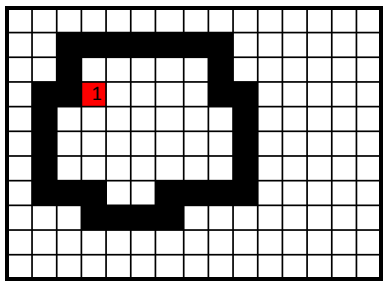
- Start from a black pixel.
- Color it and expand to its neighboring pixel, unless neighbor is an edge (white).
- Keep expanding until no more expansion is possible
- Repeat from a new starting point, until no black pixels are left

Seed filling algorithm

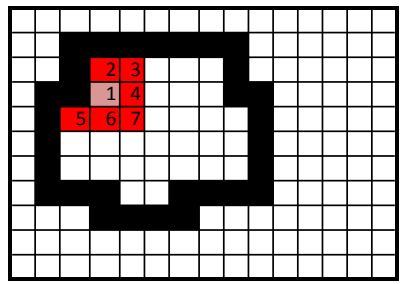
Oops: I've swapped black and white!...

Black = edge, white = background

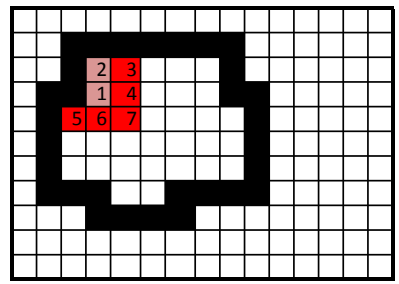
Seed = pixel at position (4,4)



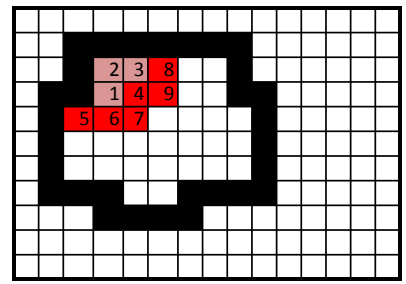
Front: [1]



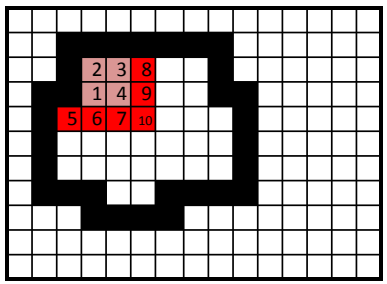
Front: [2, 3, 4, 5, 6, 7]



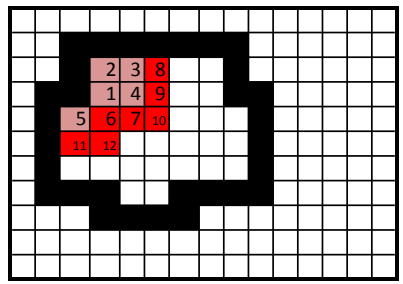
Front: [3, 4, 5, 6, 7]



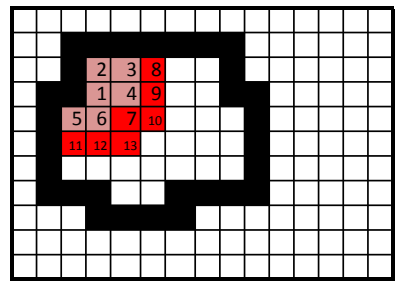
Front: [4, 5, 6, 7,8, 9]



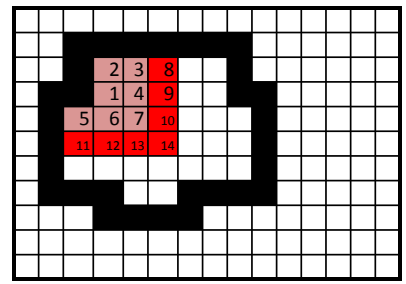
Front: [5, 6, 7, 8, 9, 10]



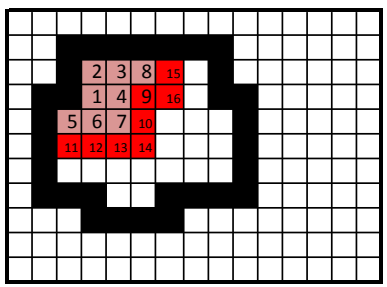
Front: [6, 7, 8, 9, 10, 11, 12]



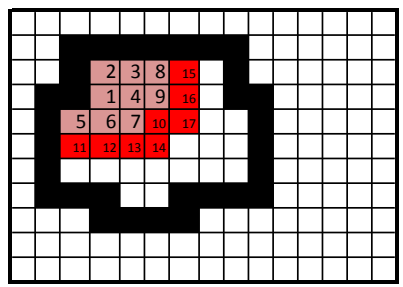
Front: [7, 8, 9, 10, 11, 12, 13]



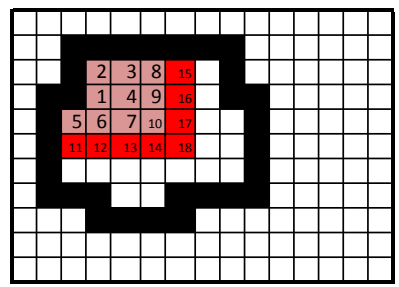
Front: [8, 9, 10, 11, 12, 14]



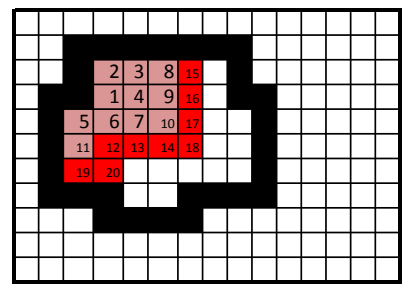
Front: [9, 10, 11, 12, 14, 15, 16]



Front: [10, 11, 12, 14, 15, 16, 17]



Front: [11, 12, 14, 15, 16, 17, 18]



Front: [12, 14, 15, 16, 17, 18, 19, 20]

Seed filling implementation

```
def seedfill(im, seed_row, seed_col, fill_color,bckg):
    """
    im: The image on which to perform the seedfill algorithm
    seed_row and seed_col: position of the seed pixel
    fill_color: Color for the fill
    bckg: Color of the background, to be filled
    Returns: Number of pixels filled
    Behavior: Modifies image by performing seedfill
    """
    size=0  # keep track of patch size
    n_row, n_col, foo = im.shape
    front=[(seed_row,seed_col)]  # initial front
    while len(front)>0:
        r, c = front.pop(0)  # remove 1st element of front
        # This is how to test equality of two np.arrays
        if np.array_equal(im[r, c],bckg):
            im[r, c]=fill_color  # color the pixel
            size+=1
            # look at all neighbors
            for i in range(max(0,r-1), min(n_row,r+2)):
                for j in range(max(0,c-1),min(n_col,c+2)):
                    # if background, add to front
                    if np.array_equal(im[i,j], bckg) and\
                        (i,j) not in front:
                            front.append((i,j))
    return size
```

Seeding from all possible starting pixel...

```
filename="malaria2"
fig=plt.figure()  # ignore this
image = io.imread(filename+".jpg")

edge_image = detect_edges(image, 60)
io.imsave(filename+"_edge.jpg",edge_image)

min_cell_size=100  # based on prior knowledge
max_cell_size=300  # based on prior knowledge
n_cells=0

for i in range(edge_image.shape[0]):
    for j in range(edge_image.shape[1]):
        # if pixel is black, seedfill from here
        if np.array_equal(edge_image[i,j,:],(0,0,0)):
            rand_color = (random.randrange(255),
                          random.randrange(255),
                          random.randrange(255))
            size=seedfill_with_animation(edge_image, i ,j,
                                         rand_color,(0,0,0))
            if size>= min_cell_size and size<max_cell_size:
                n_cells+=1
print("Number of cells:",n_cells) # Number of cells: 208
```

Seed filling execution

See live execution of program

Issues

Several things would need to be improved to get a more accurate cell count:

- Some cells are not surrounded by a closed edge because of lack of contrast; those end up not being counted.
- In some cells, the nucleus is enclosed by an edge. Those cells often end up not being counted, because both the cytoplasmic and nuclear portions are too small to be called a cell.
- Some cells may not be red blood cells, and should not be counted
- etc...