UNDERSTANDING ALZHEIMER'S DISEASE: AN IMAGING APPROACH USING SEGMENTATION & CLASSIFICATION OF AMYLOID PLAQUES

Savina Kim

Quantitative Big Imaging
Spring 2019

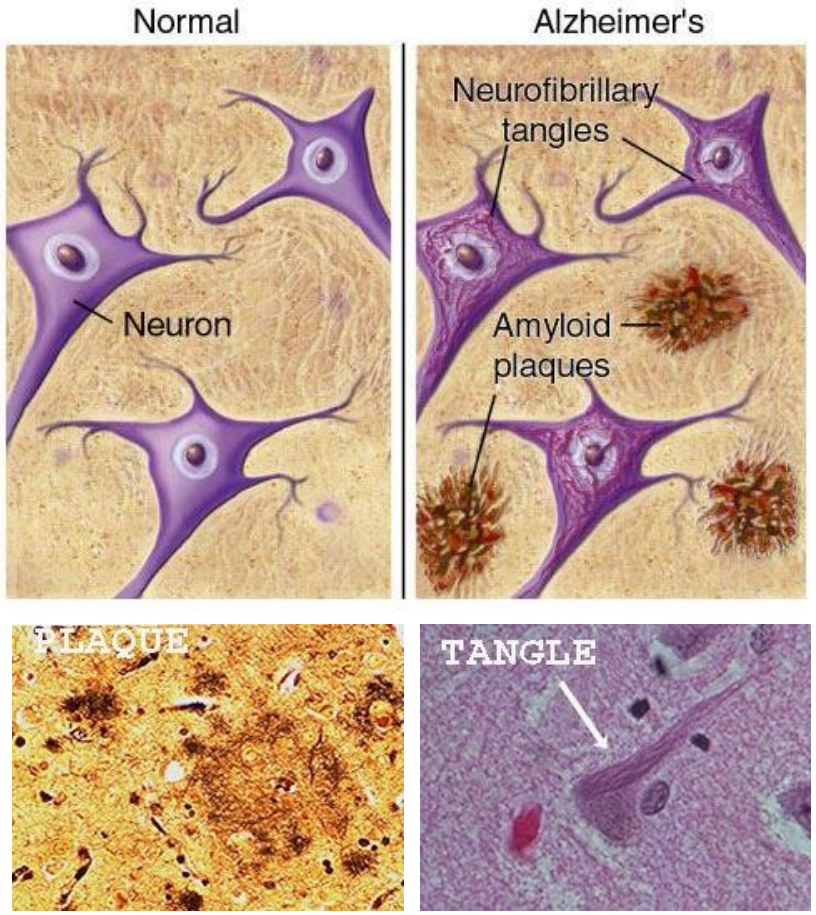
PART 1: WHOLE IMAGE

Segmentation, Shape Analysis

ALZHEIMER'S DISEASE OVERVIEW

AD is pathologically characterized by:

- Presence and abnormal accumulation of -
  - **Amyloid plaques,** containing amyloid-β (Aβ)
  - Neurofibrillary tangles **(NFT),** containing hyperphosphorylated tau
- Significant **loss of neurons** and deficits in neurotransmitter systems
- Wide range of molecular and cellular changes take place in the brain of a person with AD which can be observed in brain tissue post-mortem
- Research is ongoing to determine what changes cause AD and which may be a result of the disease



ALZHEIMER'S DISEASE FACTS

Statistics:

- Worldwide ~44 million people have Alzheimer's or related dementia
- Only 1-in-4 people with AD have been diagnosed
- Alzheimer's and dementia is most common in Western Europe with North America close behind

Alzheimer's Care Costs

- In 2016, 15.9 million family caregivers provided an estimated 18.2 billion hours and $230 billion to people with dementia
- In 2017, Alzheimer's cost the United States $259 billion
- By 2050, it is estimated that costs associated with dementia could be as much as $1.1 trillion
- Global cost of Alzheimer's and dementia is estimated to be $605 billion, which is equivalent to 1% of the entire world's GDP

ORIGINAL IMAGE

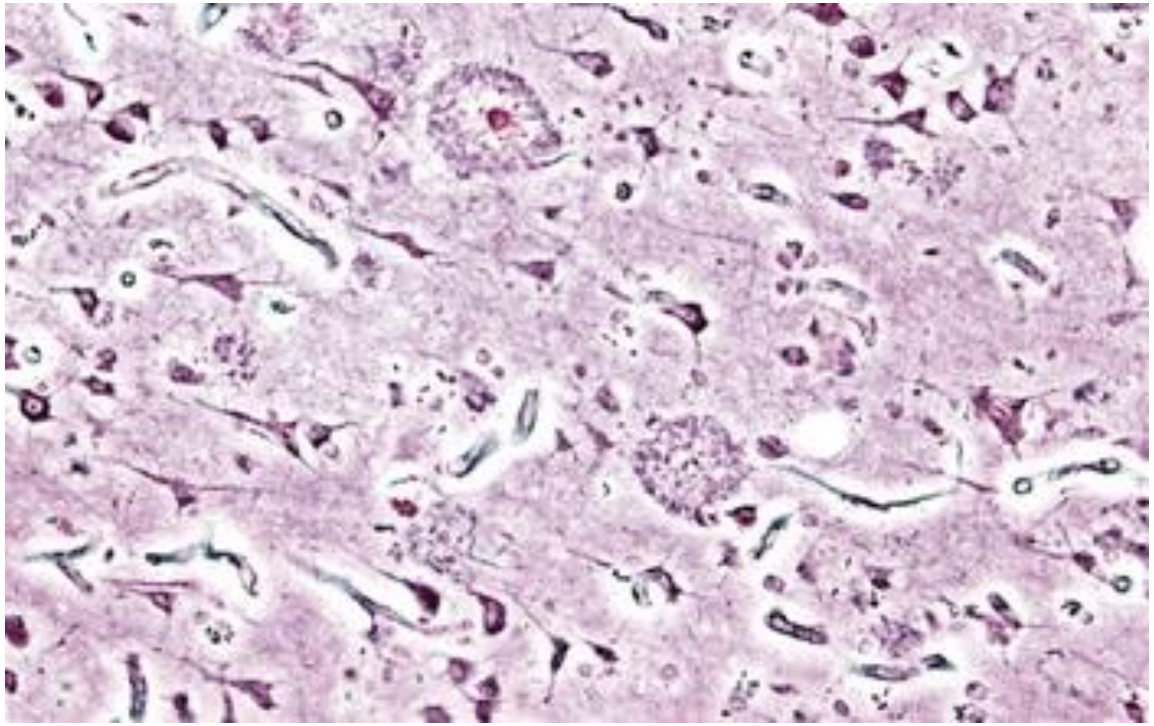
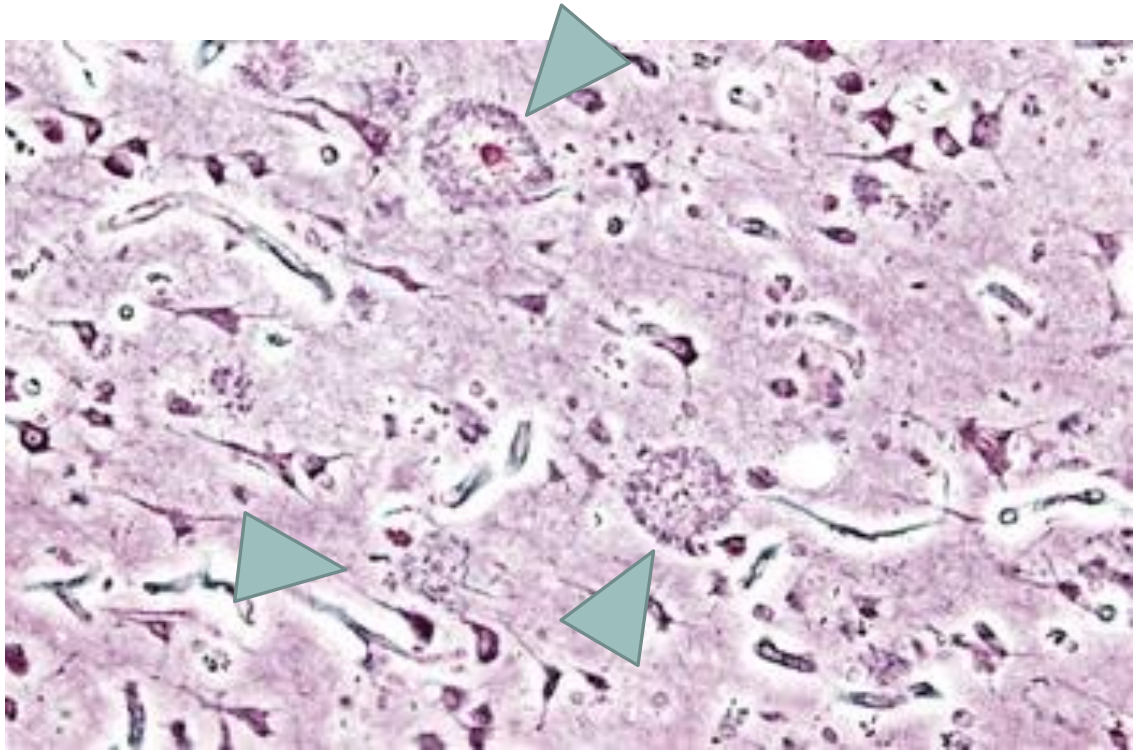


Image Description: **Microscopic image of senile plaques seen in the cerebral cortex of a person with AD**

Source: https://www.nsf.gov/discoveries/disc_summ.jsp?cntn_id=134049

AMYLOID PLAQUES



Main characteristics of a brain with AD

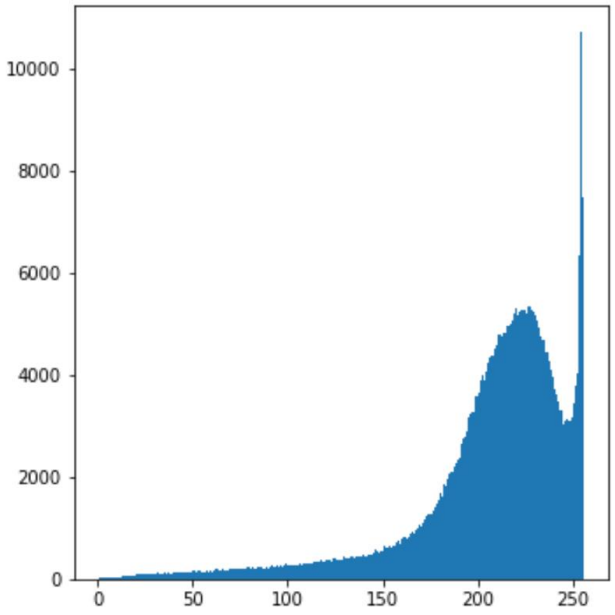
Amyloid Plaques

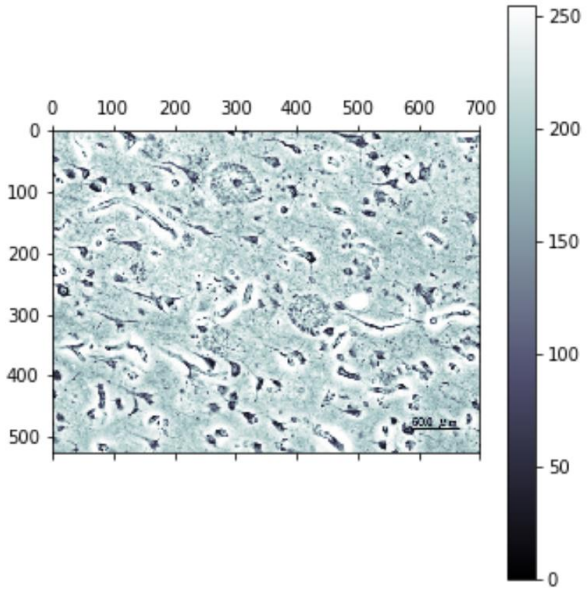
- Beta-amyloid protein, formed from breakdown of a larger protein (amyloid precursor protein) come in different forms which collect between neurons
- Abnormal levels of this naturally occurring protein clump together to form plaques that collect between neurons and disrupt cell function

Neurofibrillary Tangles

- Abnormal accumulations of protein called tau collect inside neurons
- Healthy neurons, are partially supported internally by structures called microtubules, which help guide nutrients and molecules from the cell body to the axon and dendrites
- In healthy neurons, tau normally binds to and stabilizes microtubules
- Abnormal chemical changes cause tau to detach from microtubules and stick to other tau molecules, forming threads that join to form tangles inside neurons, blocking transport system and harming synaptic communication between neurons

INITIAL QUALITATIVE METRIC: THRESHOLDING






```
nonsegmented_img = rgb2gray(imread('../input/nonsegmented-images/Alzheimer_presenile_onset.jpg'))
fig, (ax_hist, ax_img) = plt.subplots(1, 2, figsize = (12,6))
ax_hist.hist(img.ravel(),256,[0,256])
ax_obj = ax_img.matshow(img, cmap = 'bone')
plt.colorbar(ax_obj)
```

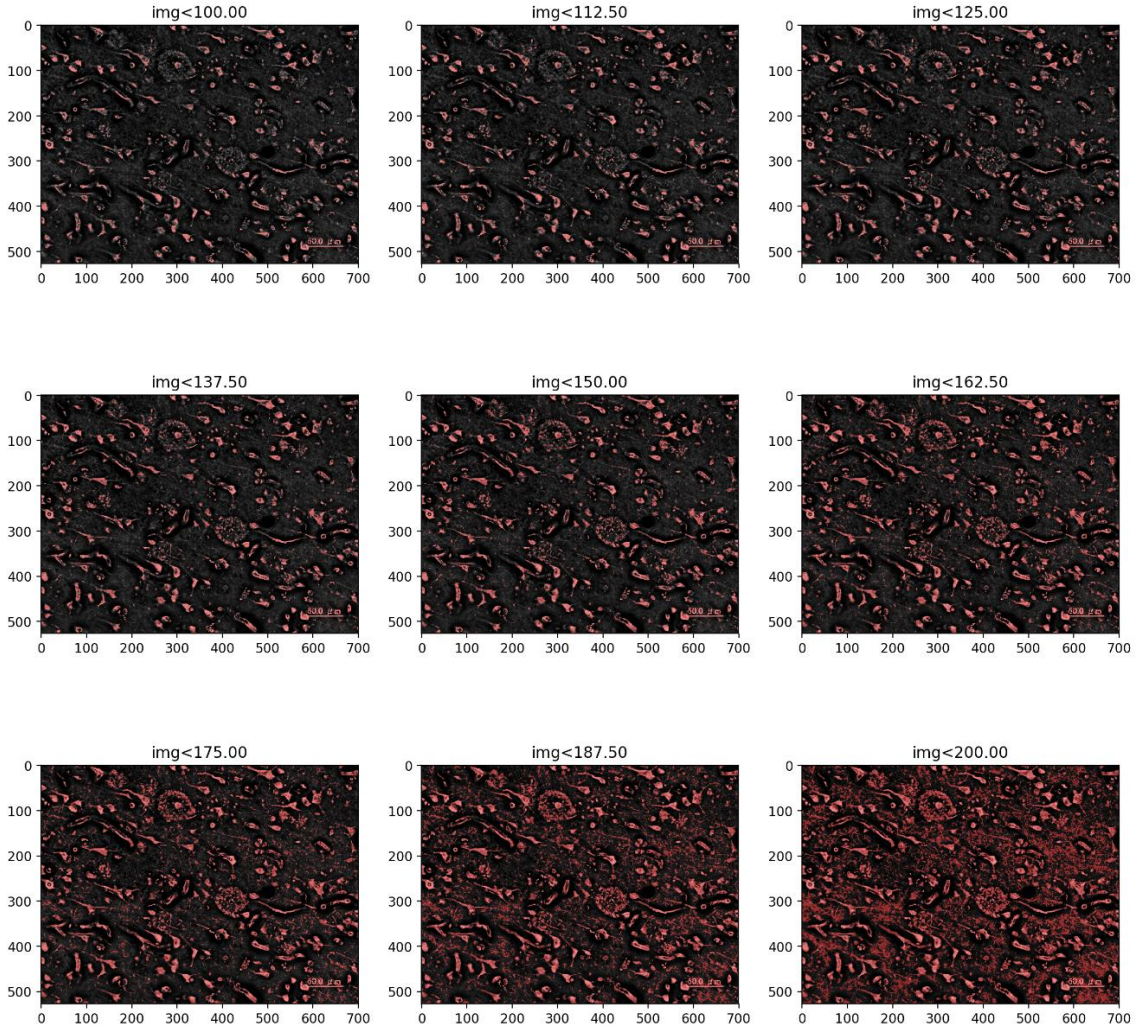
```
<matplotlib.colorbar.Colorbar at 0x7f87be58a7f0>
```

Question:

- Where to apply threshold?
- Looking at histogram of intensity pixels

Results:

- Cells absorb light causing darker regions to appear in image
- Majority of pixels are in 170-230 intensity range – seems to be relatively good threshold
- No visible “peaks” or obvious pixel value segregation with objects in image




```
from skimage.color import label2rgb
fig, m_axs = plt.subplots(3,3, figsize = (15, 15), dpi = 200)
for c_thresh, ax1 in zip(np.linspace(100, 200, 9), m_axs.flatten()):
    thresh_img = img < c_thresh

    ax1.imshow(label2rgb(thresh_img, image = 1-img, bg_label = 0, alpha = 0.4)

    ax1.set_title('img<%2.2f' % c_thresh)
```

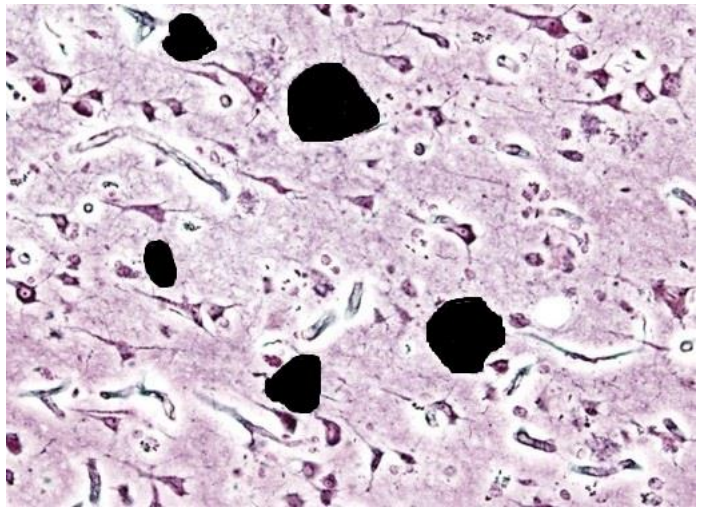
Question / Purpose:

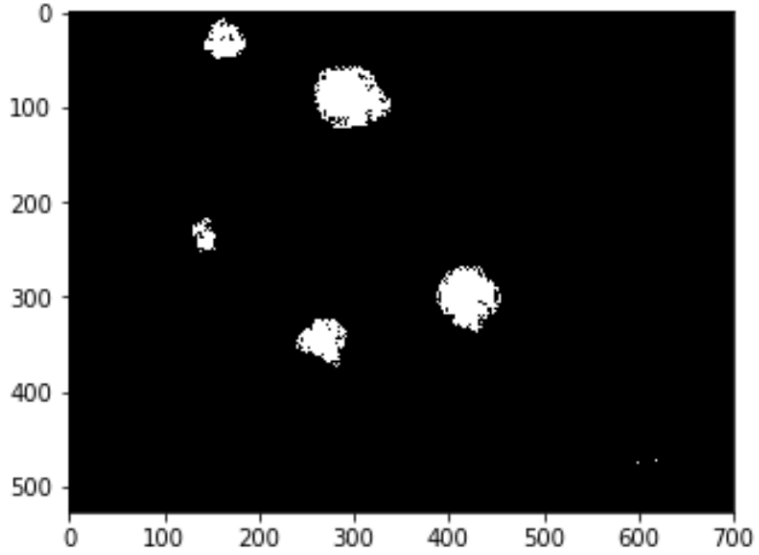
- Qualitative assessment
- Take a number of various thresholds

Results:

- Lot of noise and artifacts
- Particularly gets the whole range of cells since they all have similar absorption capacity regardless of type / shape
- Cleanest image around 125-150 range

HAND-LABELING PLAQUES






```
# Creating Labels for AD plaques (hand-colored in black - thresholding to see labeled plaques only)

img_marked = cv2.imread('../input/colored/Alzheimer_presenile_onset_colored.jpg',0)
img_marked_edited = 1-(img_marked >0.5)
plt.imshow(img_marked_edited,cmap='bone')
```

Where datasets and ground truths become important:

- Rather than going through "Image" and tweaking threshold value until it looks good..
- Took image, manually went through with paintbrush and marked image based on our “expert” understanding given certain texture and pattern

Procedure:

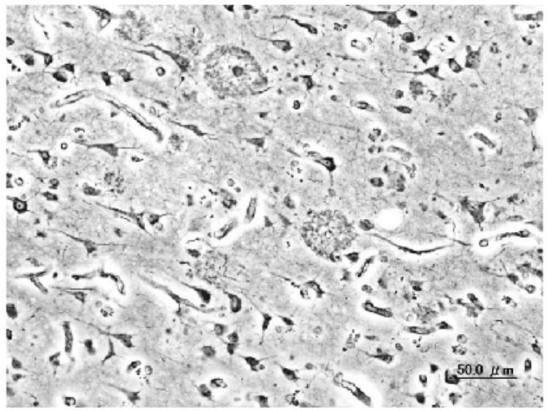
- Hand drawn / painted over plaques to indicate location
- Applied threshold to create a “labeled image”

Goal:

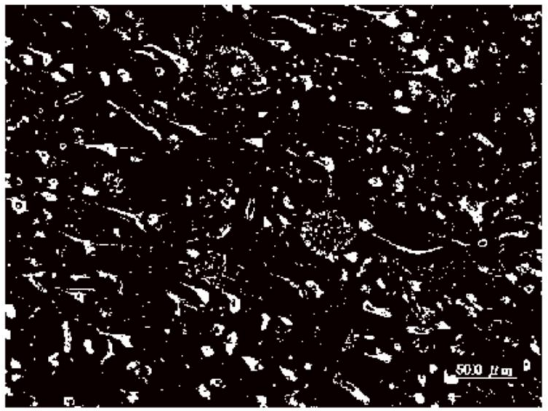
- Identify what class each pixel belongs to
- Want to be able to take a pixel and decide if its part of plaque or outside it

PIXEL CLASS IDENTIFICATION

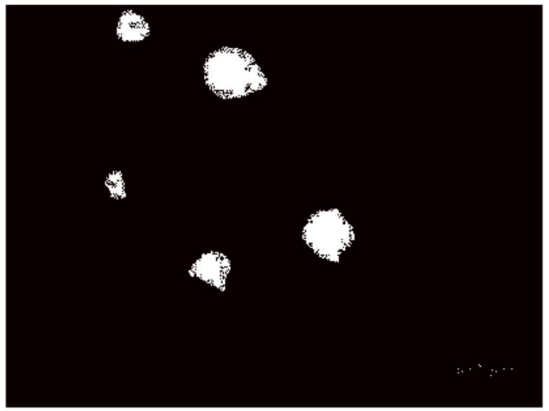
Image



Threshold



Plaque Labels

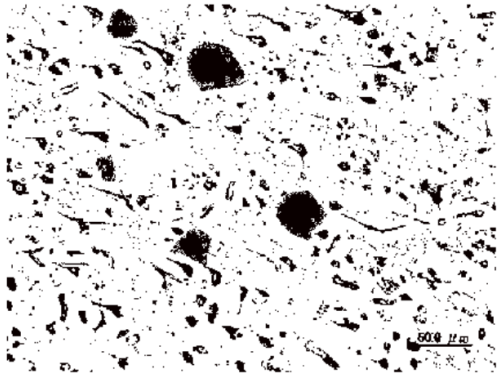


```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(12, 4), dpi=150)
ax1.imshow(img, cmap='gray')
thresh_img = img < 150
ax2.imshow(thresh_img, cmap='hot')
ax3.imshow(img_marked_edited, cmap='hot')
ax1.axis('off')
ax2.axis('off')
ax3.axis('off')
ax1.set_title('Image')
ax2.set_title('Threshold')
ax3.set_title('Plaque Labels')
```

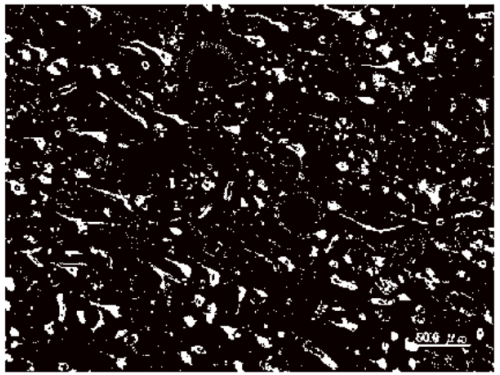
Goal:

- Want to identify which class each pixel belongs to
- Classify pixels in plaque as ***Foreground***
- Classify pixels outside plaque as ***Background***

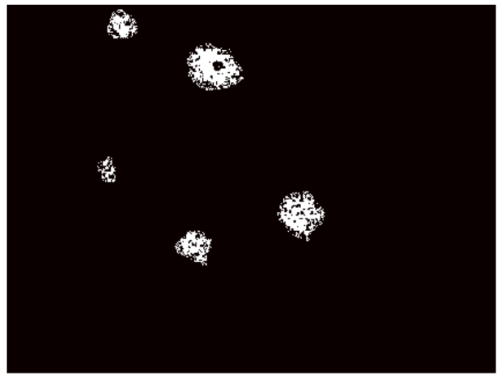
True Negative



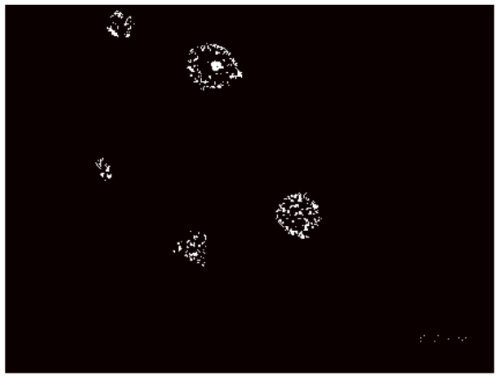
False Positive



False Negative



True Positive



```
def tp_func(real_img_idx, pred_img_idx):
    if real_img_idx == 1 and pred_img_idx == 1:
        return 'True Positive'
    if real_img_idx == 0 and pred_img_idx == 0:
        return 'True Negative'
    if real_img_idx == 0 and pred_img_idx == 1:
        return 'False Positive'
    if real_img_idx == 1 and pred_img_idx == 0:
        return 'False Negative'


out_results = {}
fig, m_ax = plt.subplots(2, 2, figsize=(8, 8), dpi=150)
for real_img_idx, n_ax in zip([0, 1], m_ax):
    for pred_img_idx, c_ax in zip([0, 1], n_ax):
        match_img = (thresh_img == pred_img_idx) & (img_marked_edited == real_img_idx)
        tp_title = tp_func(real_img_idx, pred_img_idx)
        c_ax.matshow(match_img, cmap='hot')
        out_results[tp_title] = np.sum(match_img)
        c_ax.set_title(tp_title)
        c_ax.axis('off')
print(out_results)
```

```
{'True Negative': 321903, 'False Positive': 37377, 'False Negative': 7204, 'True Positive': 2416}
```

```
# Precision & Recall

print('Recall: %2.2f' % (out_results['True Positive'] /
                         (out_results['True Positive']+out_results['False Negative'])))
print('Precision: %2.2f' % (out_results['True Positive'] /
                            (out_results['True Positive']+out_results['False Positive
```

```
Recall: 0.25
Precision: 0.06
```

How to quantify this?

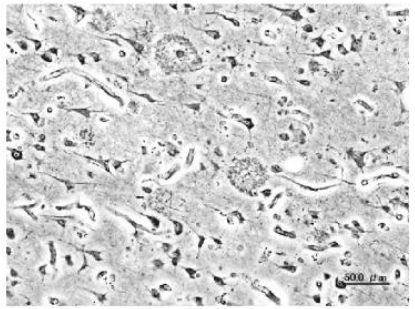
- **True Positive =** values in plaques classified as *Foreground*
- **True Negative =** values outside plaques classified as *Background*
- **False Positive =** values outside plaques classified as *Foreground*
- **False Negative =** values in plaques classified as *Background*

Result:

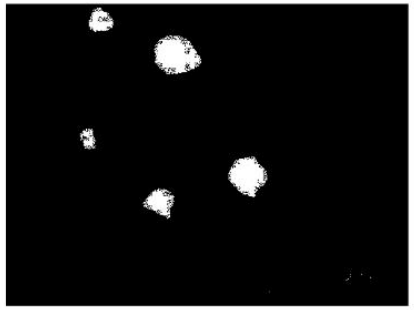
- Recall (0.25) and Precision (0.06) not so great..

FILTERING THE IMAGE

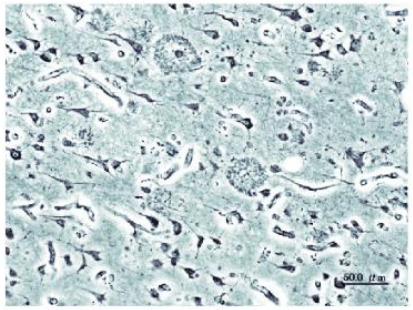
Original



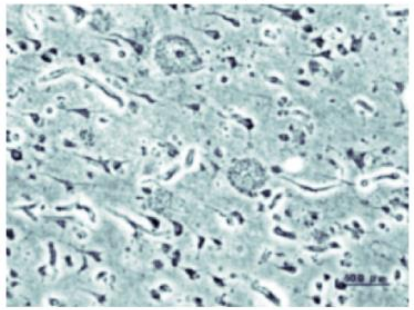
Labeled Plaques



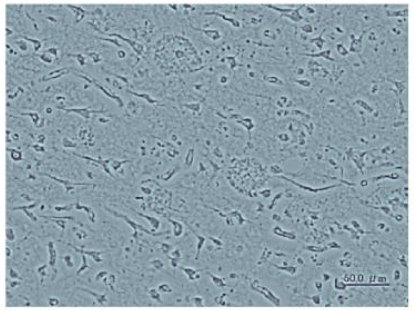
no_filter



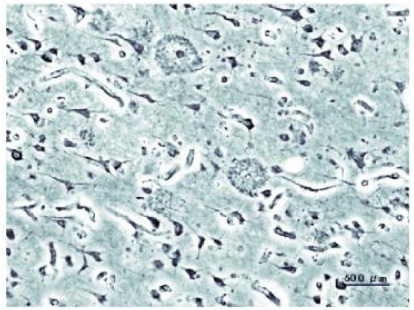
gaussian_filter



diff_of_gaussian_filter



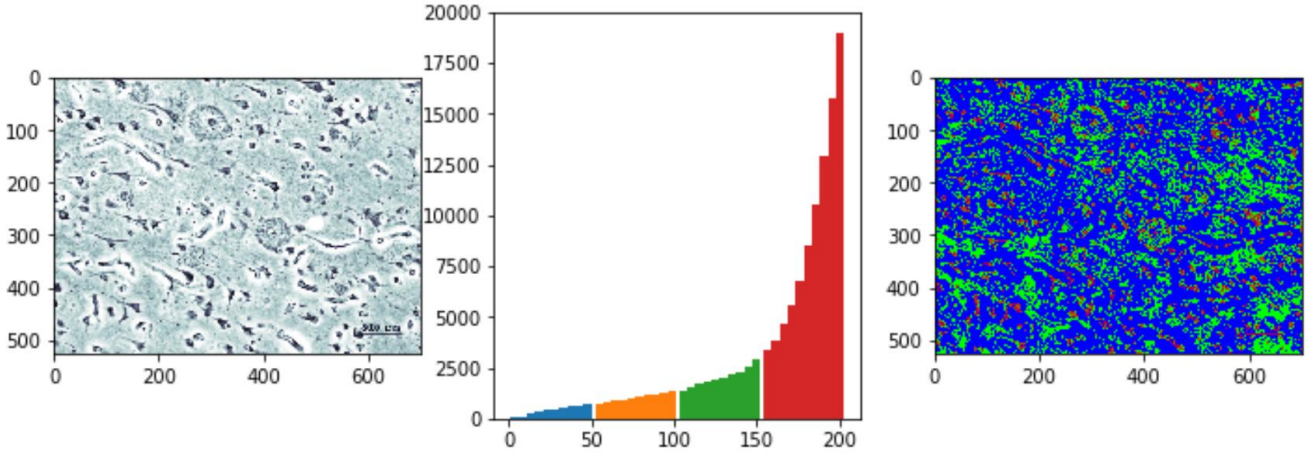
median_filter



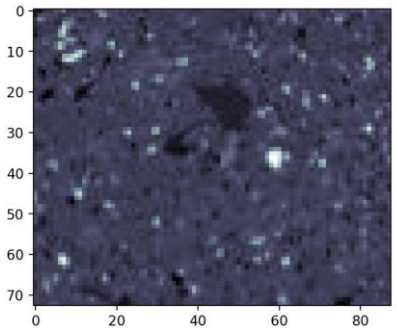
- Plaques are not unique enough?
- Neither entirely connected nor have strong absorption compared to other types of cells
- Seems more an issue of shape

```python
from skimage.filters import gaussian, median

def no_filter(x):
    return x

def gaussian_filter(x):
    return gaussian(x, sigma=2)

def diff_of_gaussian_filter(x):
    return gaussian(x, sigma=1)-gaussian(x, sigma=3)

def median_filter(x):
    return median(x, np.ones((3, 3)))

fig, m_axs = plt.subplots(1, 6, figsize=(15, 3), dpi=300)
m_axs[0].imshow(img, cmap='gray')
m_axs[0].set_title('Original')
m_axs[0].axis('off')
m_axs[1].imshow(img_marked_edited, cmap='gray')
m_axs[1].axis('off')
m_axs[1].set_title('Labeled Plaques')

for c_filt, c_ax in zip([no_filter, gaussian_filter, diff_of_gaussian_filter, median_filter], m_axs[2:]):
    c_ax.imshow(c_filt(img), cmap='bone')
    c_ax.set_title(c_filt.__name__)
    c_ax.axis('off')
```

APPLYING VARIOUS THRESHOLDS TO SEGMENT




```
np.random.seed(100)
fig, (ax1, ax2, ax3) = plt.subplots(1, 3,
                                    figsize = (12, 4), dpi = 72)
ax1.imshow(img, cmap = 'bone')
thresh_vals = np.linspace(img.min(), img.max(), 6)[:-1]
out_img = np.zeros_like(img)
for i, (t_start, t_end) in enumerate(zip(thresh_vals, thresh_vals[1:])):
    thresh_reg = (img>t_start) & (img<t_end)
    ax2.hist(img.ravel()[thresh_reg.ravel()])
    out_img[thresh_reg] = i
ax3.imshow(out_img, cmap = 'brg');
```

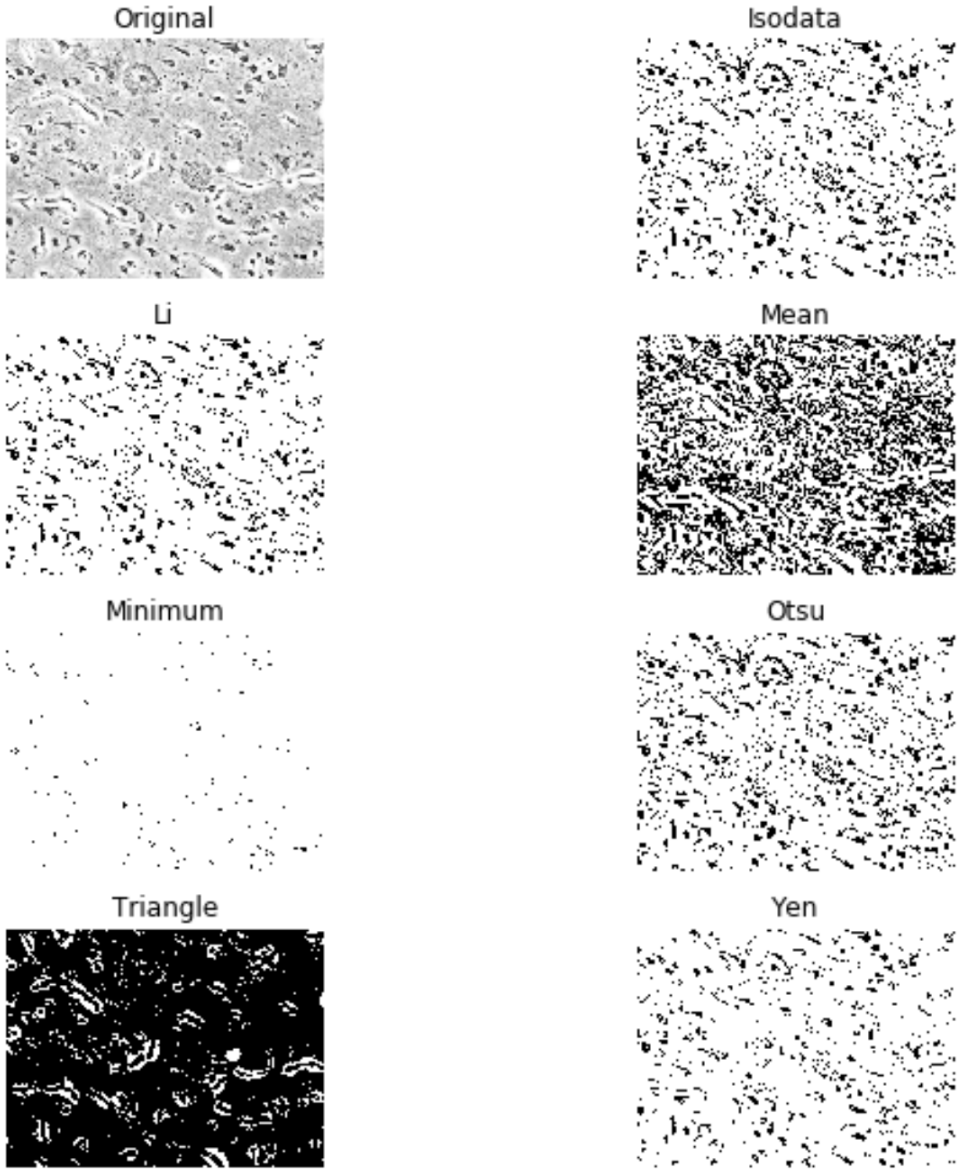


Procedure:

- Chose 4 arbitrary ranges for different phases and performed visual inspection

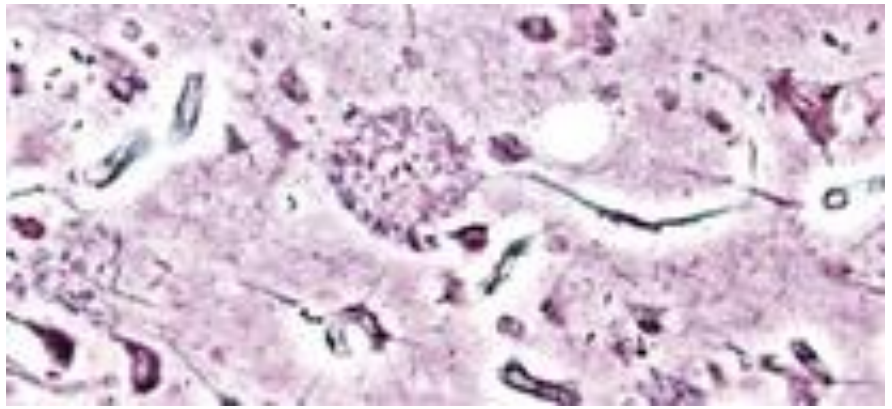
Result:

- Thresholding - does not work!
- No separate peaks in histogram, not sufficient to label structure
- No major differences in brightness (regardless of cell type)
- None of the cutoff points are any good..
- There are no distinct classes / phases *(unlike "segmenting shale" example from Lecture 4 with distinctive clay, rock and air)*



Automatic Threshold Selection:

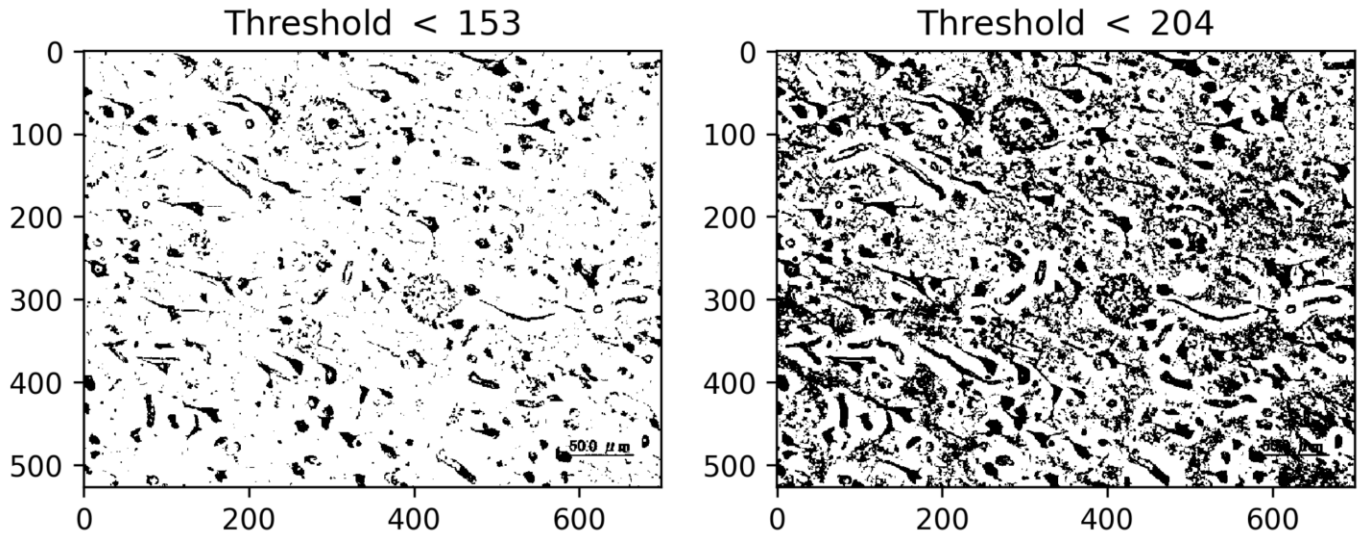
- Again, issue lies in being a distinction in tiny structures (dependent on shape)



```
from skimage.filters import try_all_threshold
fig, ax = try_all_threshold(img, figsize=(10, 8), verbose=True)
```

```
skimage.filters.thresholding.threshold_isodata
skimage.filters.thresholding.threshold_li
skimage.filters.thresholding.threshold_mean
skimage.filters.thresholding.threshold_minimum
skimage.filters.thresholding.threshold_otsu
skimage.filters.thresholding.threshold_triangle
skimage.filters.thresholding.threshold_yen
```

HYSTERESIS THRESHOLDING

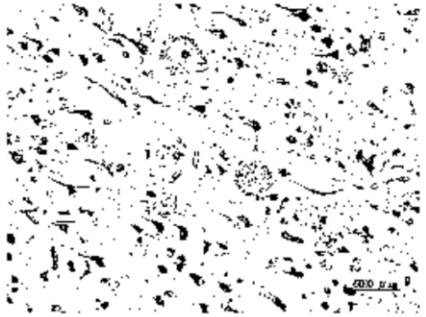



```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8,4), dpi=300)
ax1.imshow(img<thresh_vals[3], cmap = 'bone_r')
ax1.set_title('Threshold $<$ %d' % (thresh_vals[3]))
ax2.imshow(img<thresh_vals[4], cmap = 'bone_r')
ax2.set_title('Threshold $<$ %d' % (thresh_vals[4]))
```

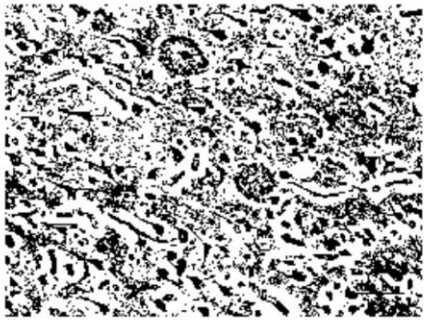
Goldilocks Thresholding:

- Where single thresholds don't work – have larger structures clearly defined and smaller structures which are difficult to differentiate
- *Goldilocks situation:* one threshold too low and one too high, gets cells but also lot of artifacts

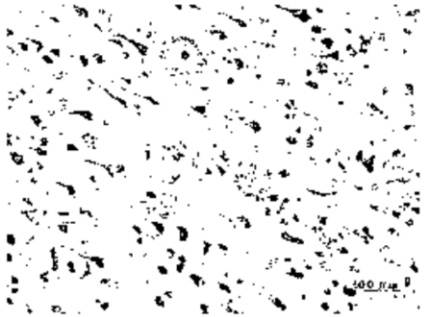
1) Strict Threshold



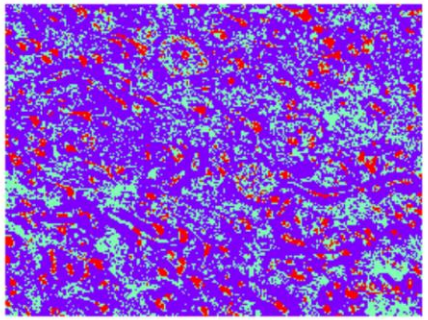
3) Looser Threshold



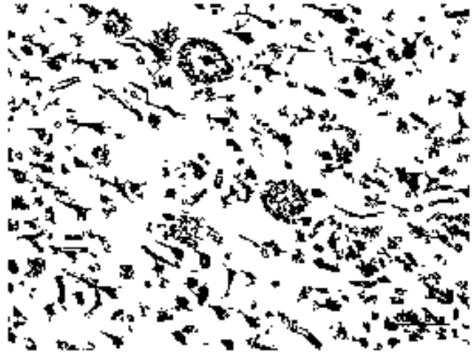
2) Remove Small Objects



4) Both Thresholds



5) Connected Thresholds



Reducing Pixels

Goal: Happy medium between two - use hysteresis thresholding to combine images together:

1) Take more strict threshold as initial starting point (things we are very confident are cells)
2) Remove objects that are too small using opening operation
3) Take 2nd threshold at higher value
4) Combine both images together by only keeping pixels which are connected ("between pixels") and ignore others

** Assumes noise structures here are only meaningful if they are **connected** already to one of these cells

Results:

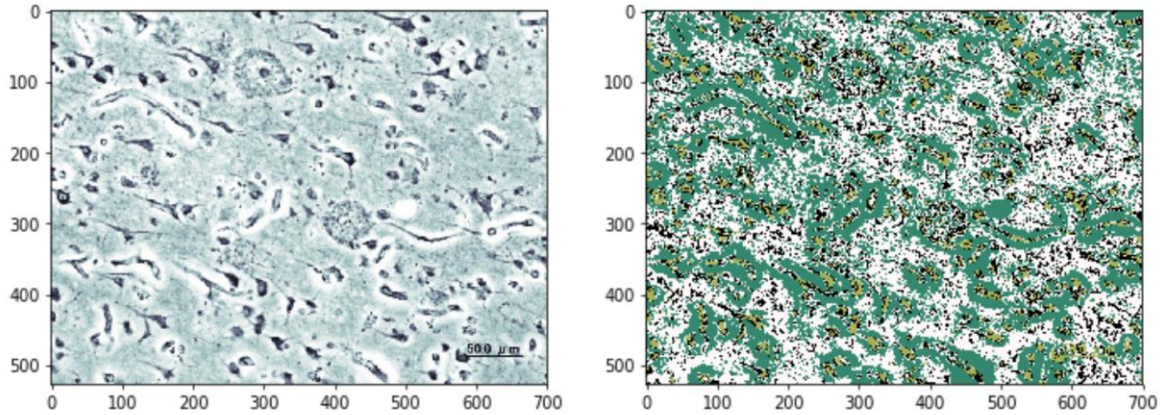
- Works decently well, can see the amyloid shapes more clearly

```
from skimage.morphology import dilation, opening, disk
from collections import OrderedDict

step_list = OrderedDict()
step_list['Strict Threshold'] = img<thresh_vals[3]
step_list['Remove Small Objects'] = opening(step_list['Strict Threshold'], disk(1))
step_list['Looser Threshold'] = img<thresh_vals[4]
step_list['Both Thresholds'] = 1.0*step_list['Looser Threshold'] + 1.0*step_list['Remove Small Objects']
# the tricky part keeping the between images
step_list['Connected Thresholds'] = step_list['Remove Small Objects']
for i in range(10):
    step_list['Connected Thresholds'] = dilation(step_list['Connected Thresholds'] ,
                                                 disk(1.8)) & step_list['Looser Threshold']

fig, ax_steps = plt.subplots(len(step_list), 1,
                             figsize = (6, 3*(len(step_list))),
                             dpi = 150)

for i, (c_ax, (c_title, c_img)) in enumerate(zip(ax_steps.flatten(), step_list.items()),1):
    c_ax.imshow(c_img, cmap = 'bone_r' if c_img.max()<=1 else 'rainbow')
    c_ax.set_title('%d) %s' % (i, c_title))
    c_ax.axis('off')
```

K-MEANS

		x	y	intensity	group
group					
0	78717	1.585	0.560	157	0
	309961	2.805	2.210	164	0
	94803	1.515	0.675	127	0
1	88541	1.705	0.630	228	1
	66119	1.595	0.470	238	1
	5756	0.780	0.040	242	1
2	196363	1.815	1.400	103	2
	160444	0.720	1.145	91	2
	268750	3.250	1.915	99	2
3	39842	3.210	0.280	221	3
	145364	2.320	1.035	195	3
	127958	2.790	0.910	190	3

```
#Applying K-means after rescaling compnonents
# Since distance is calculated and values of position are much greater than values of intensity,
# need rescaling to fit on same axis

from sklearn.cluster import KMeans
km = KMeans(n_clusters=4, random_state=100)
scale_img_df = img_df.copy()
scale_img_df.x = scale_img_df.x/200
scale_img_df.y = scale_img_df.y/200
scale_img_df['group'] = km.fit_predict(scale_img_df[['x','y','intensity']].values)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,8), dpi=70)
ax1.imshow(img, cmap='bone')
ax2.imshow(scale_img_df['group'].values.reshape(img.shape), cmap='gist_earth')
scale_img_df.groupby(['group']).apply(lambda x: x.sample(3))
```



- Model for clustering data together, not a means of making prediction but taking large set of data and grouping in natural way
- Input: number of pixels
- Output: determining what refers to foreground vs. background

Procedure:

- For pixels: Euclidean distance using magnitude of intensity differences
- k=3 (# of clusters)

Problems:

- Focuses on big differences, not small differences inside tissue
- Required normalization of distance function to account for intensity distance > x- and y-position difference

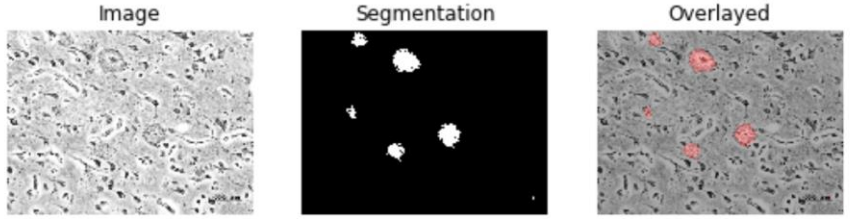
Result:

- Some of plaques seem to have fallen in black color group

SINGLE OBJECT ANALYSIS

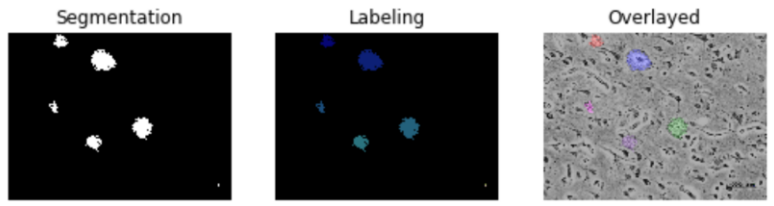
```
# show the slice and threshold
fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize = (9, 4))
ax1.imshow(em_slice, cmap = 'gray')
ax1.axis('off')
ax1.set_title('Image')
ax2.imshow(em_thresh, cmap = 'gray')
ax2.axis('off')
ax2.set_title('Segmentation')
# here we mark the threshold on the original image

ax3.imshow(label2rgb(em_thresh,em_slice, bg_label=0))
ax3.axis('off')
ax3.set_title('Overlayed')
```

```
Text(0.5, 1.0, 'Overlayed')
```




```
# make connected component labels
em_label = label(em_thresh)

# show the segmentation, labels and overlay
fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize = (9, 4))
ax1.imshow(em_thresh, cmap = 'gray')
ax1.axis('off')
ax1.set_title('Segmentation')
ax2.imshow(em_label, cmap = plt.cm.gist_earth)
ax2.axis('off')
ax2.set_title('Labeling')
# here we mark the threshold on the original image

ax3.imshow(label2rgb(em_label,em_slice, bg_label=0))
ax3.axis('off')
ax3.set_title('Overlayed')
```

```
Text(0.5, 1.0, 'Overlayed')
```



Procedure:

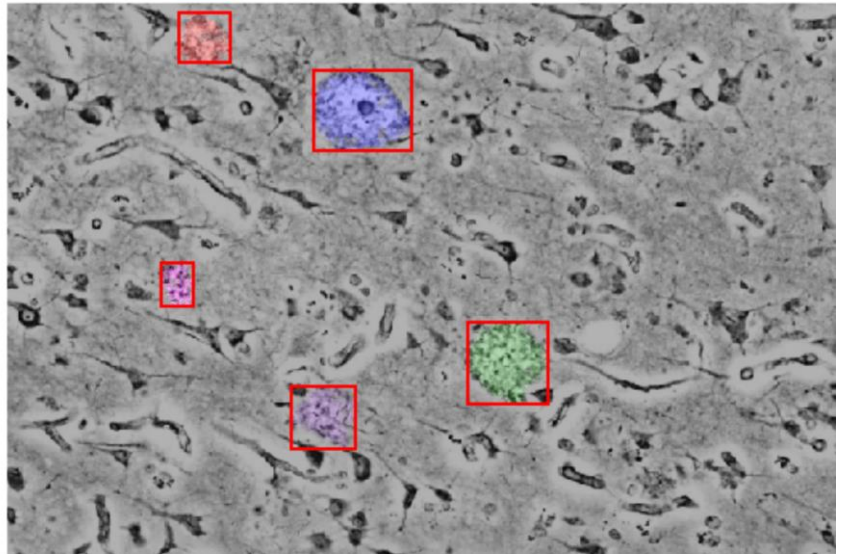
- Started with “ground truth” file as the threshold
- Performed component labeling and shape analysis

SHAPE ANALYSIS

```
shape_analysis_list = regionprops(em_label)
first_region = shape_analysis_list[0]
print('List of region properties for',len(shape_analysis_list), 'regions')
print('Features Calculated:',', '.join([f for f in dir(first_region) if not f.startswith('_')]))
```

```
List of region properties for 24 regions
Features Calculated: area, bbox, bbox_area, centroid, convex_area, convex_image, coords, eccentricity, equivalent_diameter, euler_number, exte
nt, filled_area, filled_image, image, inertia_tensor, inertia_tensor_eigvals, intensity_image, label, local_centroid, major_axis_length, max_i
ntensity, mean_intensity, min_intensity, minor_axis_length, moments, moments_central, moments_hu, moments_normalized, orientation, perimeter,
slice, solidity, weighted_centroid, weighted_local_centroid, weighted_moments, weighted_moments_central, weighted_moments_hu, weighted_moments
_normalized
```

```
fig, ax = plt.subplots(figsize=(10, 6))
ax.imshow(label2rgb(em_label,em_slice, bg_label=0))

for region in shape_analysis_list:
    # draw rectangle using the bounding box
    minr, minc, maxr, maxc = region.bbox
    rect = mpatches.Rectangle((minc, minr), maxc - minc, maxr - minr,
                              fill=False, edgecolor='red', linewidth=2)
    ax.add_patch(rect)

ax.set_axis_off()
plt.tight_layout()
```



- Performed shape analysis on the image and calculated basic shape parameters for each
- Bounding box: tells us what is the extent of our object in x,y,z
- Allows us to get very simple measurements for what we're looking at

ANISOTROPY

Anisotropy: Variation in magnitude according to direction of measurement

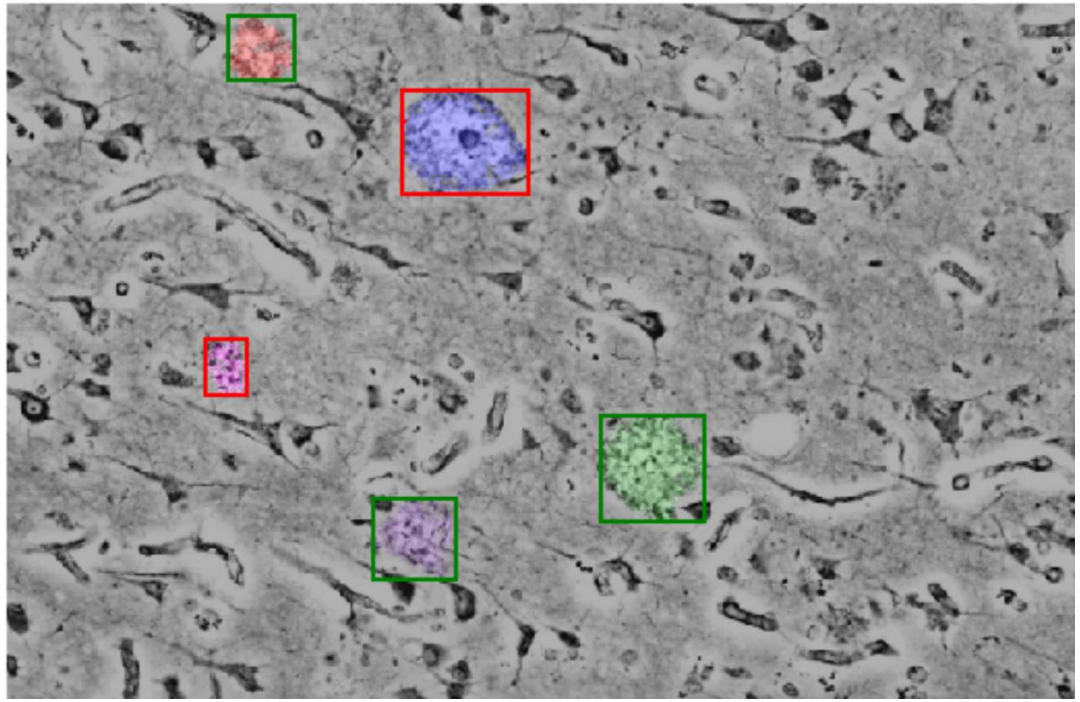
- For characterizing shape, allows us to define metrics w.r.t. one another
- Take longest side and compare to shortest side (i.e. is it tall, skinny, short, round?)

Procedure:

- For plaques, drew bounding box around limits in order to have some idea of extents of object
- Provides representation for image shape

Result:

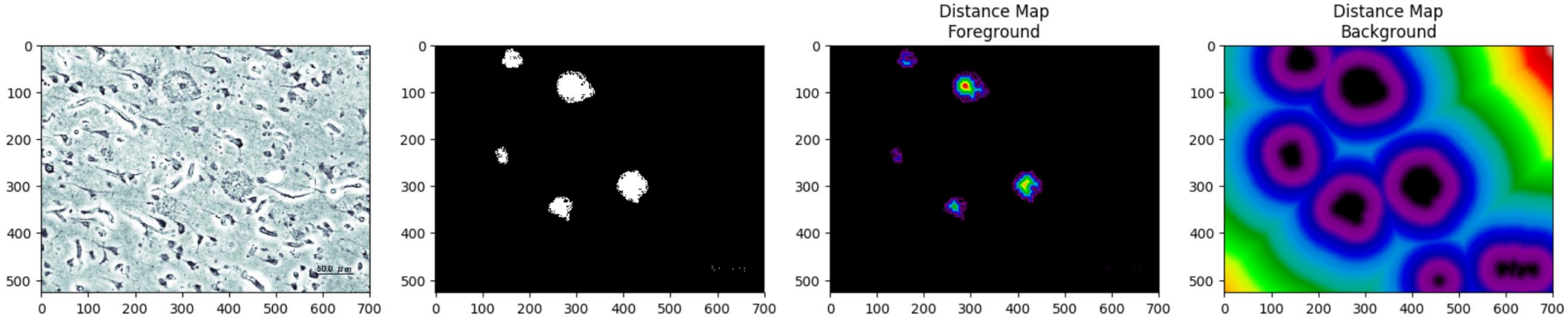
- Low anisotropy values → Plaques can be characterized as very circular and some nearly perfectly round



```
fig, ax = plt.subplots(figsize=(10, 6))
ax.imshow(label2rgb(em_label,em_slice, bg_label=0))

for region in shape_analysis_list:
    x1=region.major_axis_length
    x2=region.minor_axis_length
    if (x1+x2) > 0:
        anisotropy = (x1-x2)/(x1+x2)
    # for anisotropic shapes use red for the others use green
    print('Label:',region.label,'Anisotropy %2.2f' % anisotropy)
    if anisotropy>0.1:
        minr, minc, maxr, maxc = region.bbox
        rect = mpatches.Rectangle((minc, minr), maxc - minc, maxr - minr,
                                  fill=False, edgecolor='red', linewidth=2)
        ax.add_patch(rect)
    else:
        minr, minc, maxr, maxc = region.bbox
        rect = mpatches.Rectangle((minc, minr), maxc - minc, maxr - minr,
                                  fill=False, edgecolor='green', linewidth=2)
        ax.add_patch(rect)

ax.set_axis_off()
plt.tight_layout()
```

```
Label: 1 Anisotropy 0.03
Label: 2 Anisotropy 0.12
Label: 3 Anisotropy 0.12
Label: 4 Anisotropy 0.24
Label: 5 Anisotropy 0.03
```

DISTANCE MAPPING




```
from skimage.morphology import binary_opening, binary_closing, disk
from scipy.ndimage import distance_transform_edt
import numpy as np
import matplotlib.pyplot as plt
from skimage.io import imread
%matplotlib inline

bw_img = img
thresh_img = img_marked_edited
fg_dmap = distance_transform_edt(thresh_img)
bg_dmap = distance_transform_edt(1-thresh_img)
fig, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, figsize=(20, 6), dpi=100)
ax1.imshow(bw_img, cmap='bone')
ax2.imshow(thresh_img, cmap='bone')
ax3.set_title('Segmentation')
ax3.imshow(fg_dmap, cmap='nipy_spectral')
ax3.set_title('Distance Map\nForeground')
ax4.imshow(bg_dmap, cmap='nipy_spectral')
ax4.set_title('Distance Map\nBackground')
```

Purpose:

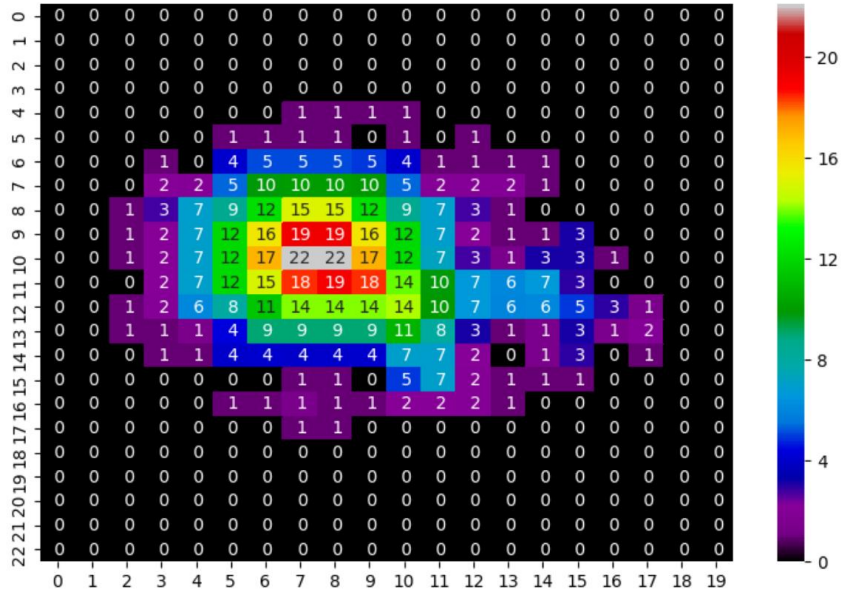
- Have circular plaques and distance map lets us take each point in map as distance that point is from given feature of interest (surface, ROI, center)

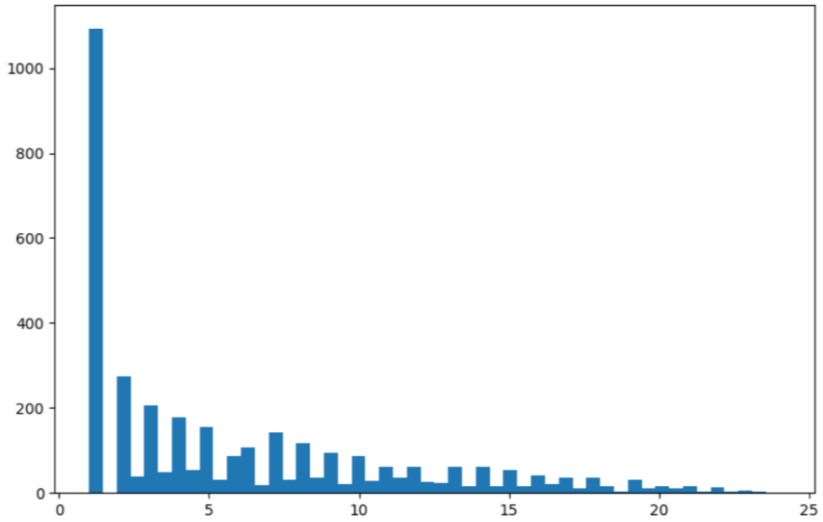
Procedure:

- Took image and computed distance transform, where each pixel value represents distance away from boundary / background
- **Foreground Background:** Information about the objects size and interior *(more on next slide)*
- **Background Foreground:** Information about the distance / space between objects
  - How far apart are the plaques?
  - Pretty evenly spaced about, however, few lie closer together forming cluster in center of image (not seen on borders)

PLAQUE SIZE

- Distance map used also to obtain visualization for what object generally looks like
- Looking at the shape tensor - see it's very round
- Can utilize this information to understand general size of plaque
- 22 is about as far away from edge that we get
  - (i.e. can calculate radius of plaque = ~ largest distance shown)






```
import seaborn as sns

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6), dpi=100)
fd_dmap_select = fg_dmap[38:152, 250:350]
sns.heatmap(fd_dmap_select[::5, ::5], annot=True,fmt="2.0f", cmap='nipy_spectral', ax=ax1, cbar=True)
ax2.hist(fd_dmap_select[fd_dmap_select > 0].ravel(), 50)
```

PART 2: SEGMENTED IMAGES

Classification Analysis

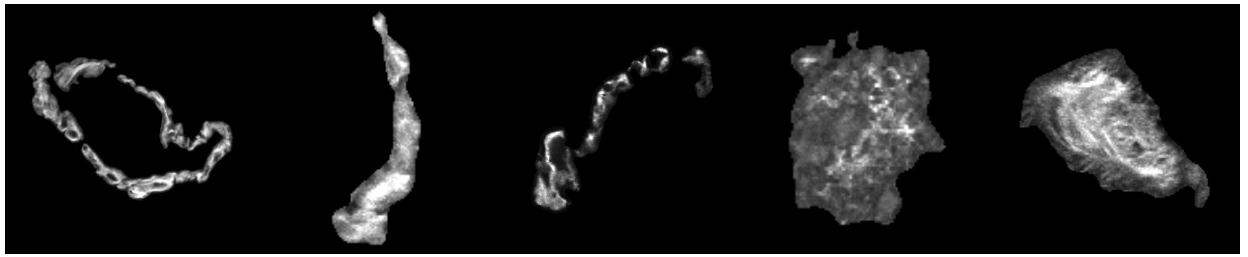
CLASSIFICATION STEPS

Dataset:

- 30 images (3 sets of 10 images per shape group)
- 3 form categories: Line, Circle, Blob (parenchyma vs. cerebrovasculature)

Steps:

1) Create a compiled .csv file data frame to hold all pixels (reshaped as 28x28)
2) Split images into train vs. test data (15 per dataset, equal amounts of each shape)
3) Label connected components
4) Perform shape analysis to quantify image with few parameters (i.e. anisotropy, mean orientation, total area, perimeter)
5) Build a classifier using these shape features
6) Use MAE (mean absolute error) to find best match



```
images[0:5]
r = random.sample(images, 3)

# Matplotlib black magic
plt.figure(figsize=(16,16))
plt.subplot(131)
plt.imshow(cv2.imread(r[0]))

plt.subplot(132)
plt.imshow(cv2.imread(r[1]))

plt.subplot(133)
plt.imshow(cv2.imread(r[2]));
```

```
print('Input Data:', train_data.shape)
print('Number ID:', numb_id.shape)
print('Number Vector:', numb_vec.shape)
```

```
Input Data: (15, 785)
Number ID: (15,)
Number Vector: (15, 784)
```

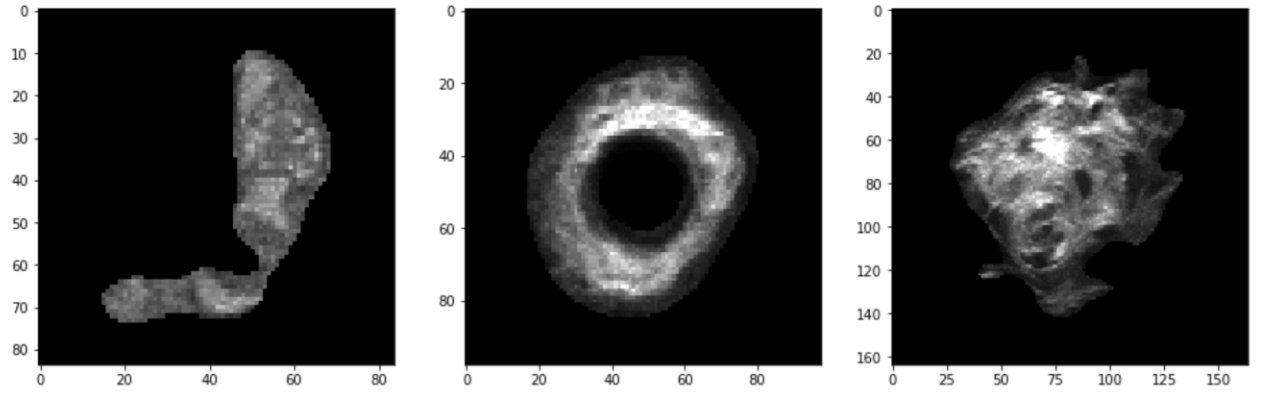
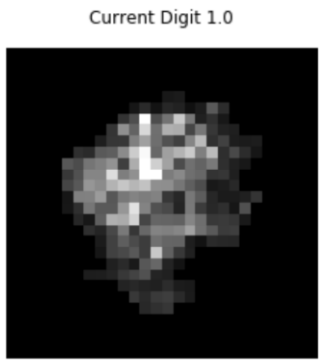


IMAGE PRE-PROCESSING

View Data → Label Connected Components → Shape Analysis

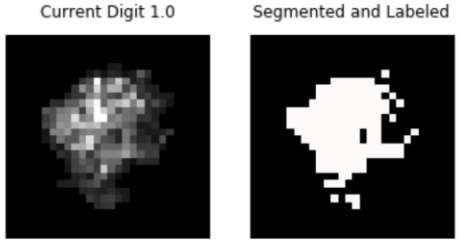
```
%matplotlib inline
fig, ax1 = plt.subplots(1,1)
ax1.matshow(numb_image[0], cmap = 'gray')
ax1.set_title('Current Digit {}'.format(numb_id[0]))
ax1.axis('off')
```

```
(-0.5, 27.5, 27.5, -0.5)
```




```
from skimage.measure import label # connected component labeling
def seg_and_label(in_image):
    norm_image = (in_image - in_image.mean())/in_image.std()
    return (norm_image>0.5).astype(np.uint8)

fig, (ax1,ax2) = plt.subplots(1,2)
ax1.matshow(numb_image[0], cmap = 'gray')
ax1.set_title('Current Digit {}'.format(numb_id[0]))
ax1.axis('off')
ax2.matshow(seg_and_label(numb_image[0]),cmap='gist_earth')
ax2.set_title('Segmented and Labeled')
ax2.axis('off')
```

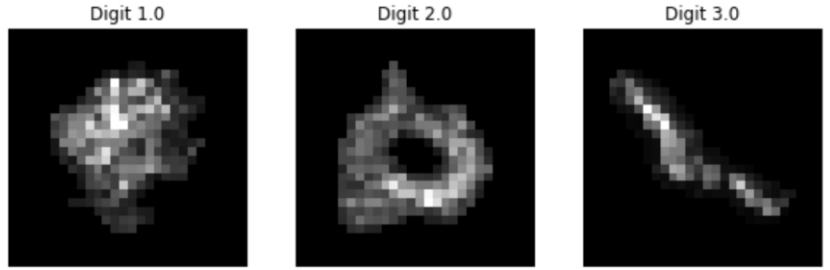
```
(-0.5, 27.5, 27.5, -0.5)
```




```
{'total_area': 151,
 'total_perimeter': 60.83452377915607,
 'mean_anisotropy': 0.08432950318878585,
 'mean_orientation': 1.3293836392660507}
```

```
fig, c_axs = plt.subplots(1, len(digit_examples), figsize = (10, 3))
for c_ax, (c_digit, c_img) in zip(c_axs, digit_examples.items()):
    c_ax.imshow(c_img, cmap = 'gray', interpolation = 'none')
    c_ax.set_title('Digit {}'.format(c_digit))
    c_ax.axis('off')
    print('Digit {}'.format(c_digit))
    print(full_analysis(c_img))
```

```
Digit 1.0
[1.51000000e+02 6.08345238e+01 8.43295032e-02 1.32938364e+00]
Digit 2.0
[1.86000000e+02 9.83969696e+01 6.65136796e-02 2.77951191e-01]
Digit 3.0
[84.         57.49137803  3.27685484 -0.71116957]
```



CLASSIFICATION

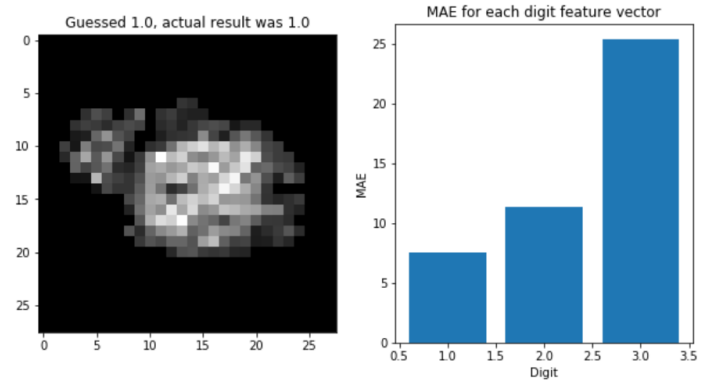
To increase performance:

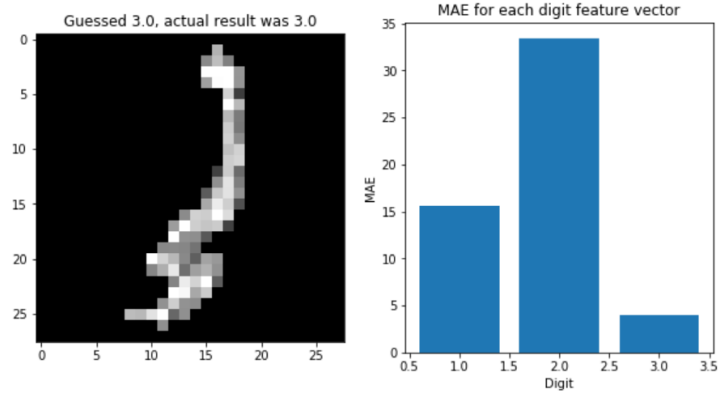
- Need more images - 5 not enough for training/testing!
- Additional parameters that can be used for shape analysis?
  - **Directional Analysis:**
    - Looking at the orientation of different components using Fourier analysis (Analyze Directionality)
  - **Tubeness / Surfaceness:**
    - Characterize binary images and the shape at each point similar to curvature but with a different underlying model
    - Tubeness: Enhances filamentous structures of a specified thickness (trace neurons)

```
def mae(fvec1,fvec2):
    return np.mean(np.abs(fvec1-fvec2))

def classify_image(example_dict, in_image):
    score_dict = {c_digit: mae(full_analysis(in_image), c_fvec) for c_digit, c_fvec in example_dict.items()}
    best_digit, best_score = sorted(score_dict.items(), key = lambda x: x[1])[0] # sort by score and take the first item
    return best_digit, score_dict
```

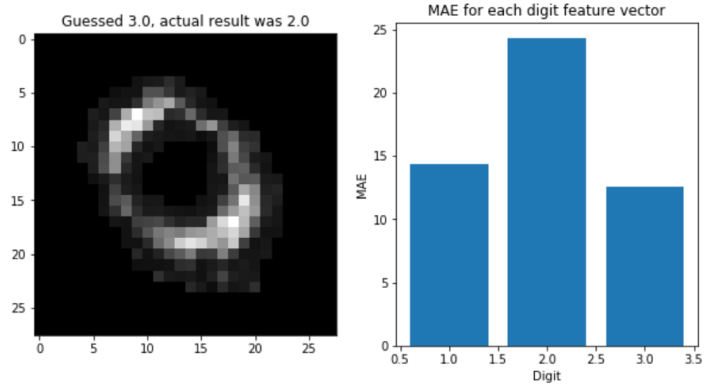
```
rand_digit = np.random.choice(range(len(numb_image))) # just picks a random digit

guess_digit, guess_dict = classify_image(digit_features, numb_image[rand_digit])
print('Guessed {}, actual result was {}'.format(guess_digit, numb_id[rand_digit]))
print('Score for other numbers:', guess_dict)
# show the results
fig, (ax_img, ax_score) = plt.subplots(1,2, figsize = (10, 5))
ax_img.imshow(numb_image[rand_digit], cmap = 'gray', interpolation = 'none')
ax_img.set_title('Guessed {}, actual result was {}'.format(guess_digit, numb_id[rand_digit]))
ax_score.bar(list(guess_dict.keys()), list(guess_dict.values()))
ax_score.set_xlabel('Digit')
ax_score.set_ylabel('MAE')
ax_score.set_title('MAE for each digit feature vector')
```

```
Guessed 1.0, actual result was 1.0
Score for other numbers: {1.0: 7.580634552315553, 2.0: 11.356240138581647, 3.0: 25.42002324034109}

Text(0.5,1,'MAE for each digit feature vector')
```





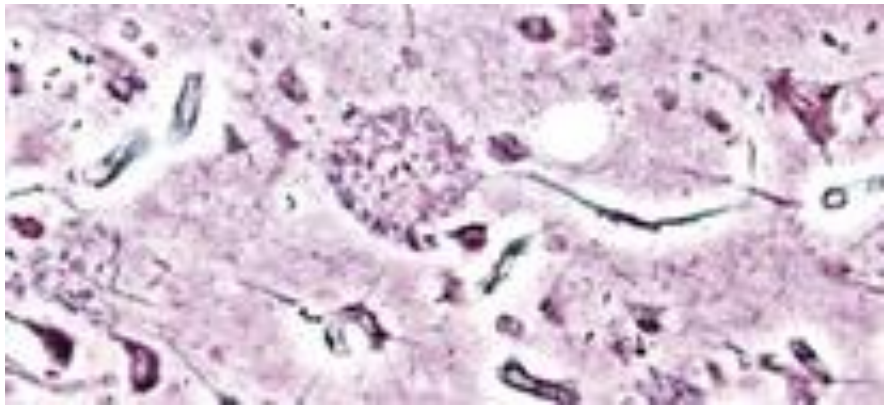

```
Guessed 3.0, actual result was 2.0
Score for other numbers: {1.0: 14.362645528755147, 2.0: 24.312872317479993, 3.0: 12.580286959033245}

Text(0.5,1,'MAE for each digit feature vector')
```



FUTURE STEPS

Advanced Investigation of Surface Structure

- Curvature: Visual indication of cell differences (amyloid is the most circular item in the image) independent of its size
- Interface - Perimeter: Help differentiate connectivity of cell networks
- Further Texture Analysis: what feature detections are best correlated with amyloid object (i.e. co-occurrence matrix)
  - Contrast
    - Dissimilarity
    - Homogeneity
  - Orderliness
    - How regular ("orderly") the pixel value differences are
  - Entropy / Energy
    - Higher entropy value indicates greater complex variability (i.e. yields more useful information vs. "fog" image consisting of identical pixels)
    - ASM (Angular second moment) - orderly
  - Correlation
    - Measures linear dependency of grey levels on those of neighboring pixels



THANK YOU