Evaluating and Optimizing Machine Learning Techniques for Automatic Nuclei Detection

1. Motivation

Many people die from cancer every year. This is terribly sad. :'(We would like to prevent this. One option is to identify the cancer early; treatment of nascent cancers tends to be more successful. Often, cells suffering from cancer display different biomarkers in their nuclei. These biomarkers can be seen in immunohistochemical (IHC) imaging. If there were a rapid and accurate way to screen images for cancerous cells, many cancer prognoses could be improved. We aim to develop an automated cell nuclei detection technique.

2. Challenges

Imaging conditions can vary tremendously - illumination, contrast, fluorescence and staining will all affect the appearance of the cell. We would also like our technique to be generalizable to many cell types because cancer is insidious and affects many cell types. Depending on the biological sample that was imaged, cells may also be aggregated; where a trained eye could distinguish individual nuclei, an algorithm might falter.

3. Pre-treatment and Initial Segmentation ("Detection")

Colour is usually either normalized or thresholded to remove noise and background [1]. A variety of pre-processing techniques can be used to find objects, from conventional blob detection, to morphological and/or contour resolution, to watershed segmentation [2-4]. We will use sensible discretion to pick a pre-treatment technique.

4. Segmentation and Identification

There are two primary approaches to cell segmentation:

Traditional methods segment nuclei from single or overlapping cells

Several traditional methods, often involving a-priori knowledge of cell shape and size [5]. First, cell clusters are segmented from the background by concavity [6-7]. Next, cell clusters can be separated into individual cells based on the concavity of the intensity distribution [7]. After individual cells have been identified, cell boundaries are often approximated using elliptical curve-fitting techniques [5,7]. Further segmentation can be applied to separate the cell nucleus from the cytoplasm. A gradient vector flow active contour model (GVF-ACM) has been shown to find boundaries between the nucleus and cytoplasm [8].

In this work, we can begin performing segmentation using the scikit-image package for python. This package includes methods for ellipse and boundary fitting, as well as edge detection and active contour modeling.

Machine learning methods identify nuclei via classification algorithms

Machine learning and pattern recognition have been successfully used to identify and segment cells in IHC images [9]. Whether using techniques such as cluster analysis [10], random forests [9], or deep neural networks [11-13], the workflow is similar. First, initial segmentation is performed (often in the pre-treatment step) to find cells and agglomerates. Then, training data is fed through a classifier to extract the most important features. Repeating this process while keeping only the most important feature vectors establishes a model, which is finally used to classify new test data.

We will start with cluster analysis and random forest classifiers (from the scikit-learn package), but likely will also employ traditional GVF or ellipse-fitting algorithms (which we will implement in python) to refine initial segmentation. If necessary, we also intend to look into convolutional neural networks implemented with the TensorFlow package.

References

[1] M. Veta, P.J. Van Diest, R. Kornegoor, A. Huisman, M.A. Viergever, and J.P.W. Pluim. (**2013**) Automatic Nuclei Segmentation in H&E Stained Breast Cancer Histopathology Images. *PLoS One*, 8, 7.

[2] X. Yang, H. Li, and X. Zhou. (**2006**) Nuclei Segmentation using Marker-Controlled Watershed, Tracking using Mean-Shift, and Kalman Filter in Time-Lapse Microscopy. *IEEE Trans. Circuits Syst. I, Reg. Papers.*, 53, 11.

[3] J. Cheng and J.C. Rajapakse. (**2009**) Segmentation of Clustered Nuclei with Shape Markers and Marking Function. *IEEE Trans. Biomed. Eng.*, 56, 3.

[4] S. Ali and A. Madabhushi, “An Integrated Region-, Boundary-, Shape-Based Active Contour for Multiple Object Overlap Resolution in Histological Imagery. *IEEE Trans. Med. Imag.*, 31, 7.

[5] S. Kothari, Q. Chaudry, M.D. Wang. (2009) Automated Cell Counting and Cluster Segmentation Using Concavity Detection and Ellipse Fitting Techniques. *Proc. IEEE Int. Symp. Biomed. Imaging*, 795.

[6] Y. Toyoshima, T. Tokunaga, O. Hirose, M. Kanamori, T. Teramoto, M.S. Jang, S. Kuge, T. Ishihara, R. Yoshida, and Y. Iino. (**2006**) Accurate Automatic Detection of Densely Distributed Cell Nuclei in 3D Space. *PLoS. Comput. Biol.*, 12, 6.

[7] H.S. Wu, J. Gil, and J.Barba. (**1998**) Optimal Segmentation of Cell Images. *IEE P-Vis. Image. Sign.*, 145, 1.

[8] S.F. Yang-Mao, Y.K. Chan, and Y.P. Chu. (**2008**) Edge enhancement nucleus and cytoplast contour detector of cervical smear images. *IEEE Trans. Syst. Man, Cybern. B*, 38, 2.

[9] O. Rujuta and A.J. Vyavahare. (**2017**) Review of Nuclei Detection, Segmentation in Microscopic Images. *J. Bioengineer. Biomed. Sci.*, 7, 2.

[10] S. Wienert, D. Heim, K. Saeger, A. Stenzinger, M. Beil, P. Hufnagl, M. Dietel, C. Denkert, F. Klauschen. (**2012**) Detection and Segmentation of Cell Nuclei in Virtual Microscopy Images: A Minimum-Model Approach. *Sci. Rep.*, 2, 503.

[11] S.K. Sadanandan, P. Ranefall, S. Le Guyader, and C. Wahlby. (**2017**) Automated Training of Deep Convolutional Neural Networks for Cell Segmentation. *Sci. Rep.*, 7, 1.

[12] K. Sirinukunwattana, S.E.A. Raza, Y.W Tsang, I.A. Cree, D.R.J. Snead, and N.M. Rajpoo. (**2016**) Locality Sensitive Deep Learning for Detection and Classification of Nuclei in Routine Colon Cancer Histology Images. *IEEE Trans. Med. Imag.*, 35, 99.

[13] N. Kumar, R. Verma, S. Sharma, S. Bhargava, A. Vahadane, and A. Sethi. (**2017**) A Dataset and a

In [6]:
```
## change the following to where you have stored and/or extracted the files
path = './project/'
# path = '/Users/arrakis/Dropbox/Tool - Classes/ChBE 8803/Project'

n_samples = 3 #/665 total datasets in train1

####################################################################
```

UPDATED 3/28 ideas for future work

Updates (3/29 4 am)

- KMEANS clustering to separate into different cell/conditions type - will fix tomorrow (tired)
- Watershed performance metrics calculated (comparable to the random forest now)
- SVM classifier for detection (slow to train) - took 30 min and wasn't finished so I removed it(same Xtrain, Ytrain as random forest)
- Random forest classifier using HSV instead of RGB values
- Example of an active contour model, but so far it seems too slow to be practical
- Plan to add chan-vese level set segmentation

Detection

- try splitting three channels as HSV instead of RGB
- do dimensional reduction / PCA on all training set images to parse into different cell/conditions type, then threshold each one with best method for group

Segmentation

Package Requirements

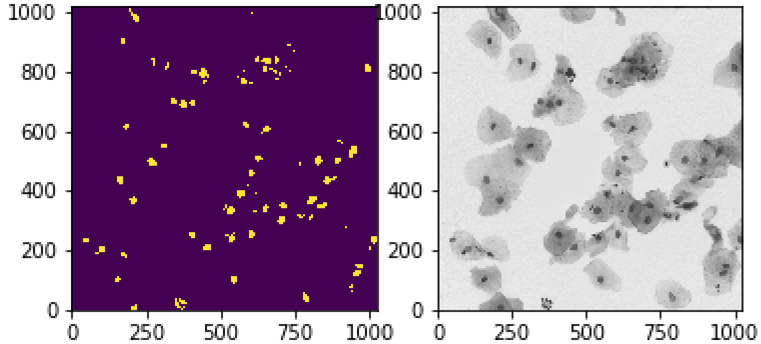
- numpy version 1.13.3
- pandas version 0.20.3
- matplotlib version
- sklearn version 0.19.1
- skimage version 0.13.0
- cv2 (used for image processing)
  - See lines 1-3 in the next block for installation on mac (version 3.2.0.6)
  - For PC, installation worked with pip install opencv-python from the anaconda prompt (version 3.4.0)

In [49]:
```
#import sys
#!{sys.executable} -m pip install opencv-python==3.2.0.6 # for mac
# >> https://stackoverflow.com/questions/47963386/image-not-found-error-afte
r-installing-opencv-python-wheel-on-mac

## load all packages used below
from skimage.color import rgb2gray
from skimage.filters import threshold_otsu
import zipfile, io
import numpy as np
import pandas as pd
import pylab as plt
import sklearn, cv2
import matplotlib.image as mpimg
from scipy import ndimage
from scipy.ndimage import label
from skimage import feature
from skimage.filters import sobel, laplace
from skimage.morphology import watershed
from sklearn import linear_model
from sklearn import svm
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
```

Loading Images and Masks from Training Set

- load_zipped_img takes an image and set of masks from the .zip
- A working example of extracting an individual image and associated masks is shown below

In [8]:
```
## STEP 1: Load an image (by index) and corresponding masks from ZIPPED stage1
_train as np array

def load_zipped_img(path, img_index): # Load an image and all its masks
    z = zipfile.ZipFile(path,'r') # access zip folder
    zlist = z.namelist() # list of files in zip folder

    img_name = zlist[img_index] # get selected image
    img_name = img_name[0:-1] # eliminate "/"

    # get image and return as np array
    img_raw = z.read('{}/images/{}.png'.format(img_name,img_name)) # get raw i
mage
    img = io.BytesIO(img_raw) # convert image
    img = mpimg.imread(img) # numpy array
    img = np.flip(img,0) # flip image

    # get all masks and return as np array
    mask_list = []
    for string in zlist:
        if string.startswith(img_name+'/mask'):
            mask_list.append(string)
    mask_list = mask_list[1:-1] # list of masks

    masks = []
    for m in mask_list:
        mask_raw = z.read(m) # get raw mask
        mask = io.BytesIO(mask_raw) # convert mask
        mask = mpimg.imread(mask) # numpy array
        mask = np.flip(mask,0) # flip mask
        masks.append(mask)

    return img, masks

# WORKING EXAMPLE OF load_zipped_img

(img, masks) = load_zipped_img(path+'/stage1_train.zip',177) # The one example
 that the model performs poorly for

fig, ax = plt.subplots(1,2)

ax[0].imshow(sum(masks), origin='lower')
ax[1].imshow(img, origin='lower')
plt.show()
```



Detection: Separating Object from Background

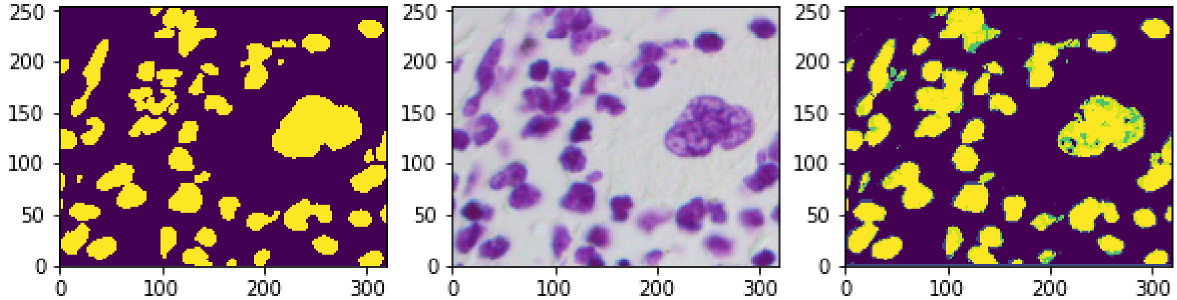
Before determining the positions of nuclei in an image, it is key to separate the image from the background using a process called image segmentation. Image segmentation is typically performed on grayscale images.

- The grayscale function converts images from rgb to grayscale
- The otsu function selects an optimal threshold for equal inter-/intra-class variance
- The function float2int8 converts the data type to int8, which is required for the cv2 package
- The function watershed applies watershed segmentation to a given image to determine objects and background with the following outputs
  - img_guess: masked objects identified by watershed segmentation
  - sure_bg: background pixels identifed with high confidence
  - sure_fg: object pixels identified with high confidence
  - uncertain: the region between sure_bg and sure_fg where pixels are identified with low confidence

In [9]:

```
# convert to grayscale
def grayscale(im):
    return rgb2gray(im)

# Otsu's Method, calculates optimal threshold for equal inter-/intra-class var
iance
def otsu(image_gray):
    threshold_val = threshold_otsu(image_gray) #Select threshold from Otsu's m
ethod
    img_masked = np.where(image_gray > threshold_val, 1, 0)

    if np.sum(img_masked==0) < np.sum(img_masked==1):
        img_masked = np.where(img_masked, 0, 1)
    return img_masked

# Function to convert float32 raw images to int8 single channel
def float2int8(img_float):
    img_int8 = (img_float * 255).round().astype(np.uint8)
    return img_int8
```

In [51]:
```
# Function to watershed segment images
def watershed(img_float32):
    img = img_float32[:, :, :3]

    # convert input image (float32) to 3-channel int8
    ch1 = float2int8(otsu(img[:,:,0:1][:,:,0]))
    ch2 = float2int8(otsu(img[:,:,1:2][:,:,0]))
    ch3 = float2int8(otsu(img[:,:,2:3][:,:,0]))
    img_guess = cv2.merge([ch1,ch2,ch3])

    # greyscale and otsu threshold original image
    img_grey = grayscale(img)
    int8_thresh = float2int8(otsu(img_grey))

    # noise removal
    kernel = np.ones((3,3),np.uint8)
    opening = cv2.morphologyEx(int8_thresh,cv2.MORPH_OPEN,kernel,iterations=2)

    # find sure background area
    sure_bg = cv2.dilate(opening,kernel,iterations=3)

    # find sure foreground area
    dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
    ret, sure_fg = cv2.threshold(dist_transform,0.2*dist_transform.max(),255,0
)
    sure_fg = np.uint8(sure_fg)

    # finding uncertain region
    uncertain = cv2.subtract(sure_bg,sure_fg)

    # marker labelling
    ret, markers = cv2.connectedComponents(sure_fg)
    markers = markers+1 # add one to all labels so sure background is 1 (not
 0)
    markers[uncertain==255] = 0 # mark unknown region as 0

    # apply watershed and mark boundary as -1
    markers = cv2.watershed(img_guess, markers)
    img_guess[markers == -1] = [255,0,0]

    return img_guess, markers, sure_bg, sure_fg, uncertain


#### TESTING
(img, mask) = load_zipped_img(path+'/stage1_train.zip', 510)

img_guess, markers, sure_bg, sure_fg, uncertain = watershed(img)

fig, ax = plt.subplots(1,3, figsize = (10,10))

ax[0].imshow(sum(mask), origin='lower')
ax[1].imshow(img, origin='lower')
ax[2].imshow(grayscale(img_guess), origin='lower')
plt.show()
```


250
200
150
100
50
0
0
100
200
300

In [43]:
```
# Watershed Testing

for i in range(0,560):
    (img, masks) = load_zipped_img(path+'/stage1_train.zip', i)

    (img_guess, markers, sure_bg, sure_fg, unknown) = watershed(img)
    yp = np.round(grayscale(img_guess)).reshape(-1,1)
    y_real = sum(masks).reshape(-1,1)
    accuracy = sklearn.metrics.accuracy_score(y_real, yp)
    precision = sklearn.metrics.precision_score(y_real, yp)
    f1 = sklearn.metrics.f1_score(y_real, yp)
    print(i, accuracy, precision, f1)
    acc.append([accuracy, precision, f1])

    #output = yp.reshape(shap[0],shap[1])

precision = np.mean(np.array(acc)[:,1])
accuracy = np.mean(np.array(acc)[:,0])
f1 = np.mean(np.array(acc)[:,2])
print('The average precision of the model is {}%'.format(precision*100))
print('The average accuracy of the model is {}%'.format(accuracy*100))
print('The average f1-score of the model is {}%'.format(f1*100))
```

```
0 0.97004699707 0.941236068896 0.791059073976
1 0.99040222168 0.960157516794 0.929476398699
2 0.907202148438 0.934694875871 0.857800224467
3 0.917321777344 0.693524020546 0.802634262902
4 0.980444335938 0.877852348993 0.851087562744
5 0.993896484375 0.940975776186 0.932386747803
6 0.920379638672 0.951196898179 0.812221102634
7 0.960447530864 0.994979882618 0.913171624094
8 0.995407104492 0.922718808194 0.868155935173
9 0.962515432099 0.986386632938 0.924245259481
10 0.943564814815 0.90031102703 0.884764455648
11 0.946896475283 0.923741207605 0.847025767106
12 0.969436645508 0.954275952166 0.90459633246
13 0.986877441406 0.952051926298 0.91360257183
14 0.943410493827 0.952805361592 0.881022679342
15 0.993118286133 0.929990966576 0.901291311009
16 0.982620239258 0.935864978903 0.853843192609
17 0.944467592593 0.998547429582 0.897829389134
18 0.995651245117 0.814143245694 0.863046612206
19 0.997802734375 0.771889400922 0.823095823096
20 0.997329711914 0.856569709127 0.907063197026
21 0.857272325376 0.882874936519 0.718924801393
22 0.952530864198 0.939936651938 0.886548887987
23 0.813229560852 0.110506356741 0.198839021636
24 0.995269775391 0.767552182163 0.839211618257
25 0.868709564209 0.0867509248252 0.159494969229
26 0.982330322266 0.955830950111 0.838583774742
27 0.947402954102 0.929693961952 0.765302648601
28 0.996292114258 0.906022845275 0.934904902223
29 0.996438439434 0.714472537053 0.717757827896
30 0.996459960938 0.763440860215 0.859564164649
31 0.982604980469 0.904942965779 0.833625218914
32 0.96396484375 0.932766514061 0.828511676542
33 0.902526855469 0.966092047487 0.788126036484
34 0.932629394531 0.875267620108 0.881058597875
35 0.987823486328 0.937463242501 0.922987840185
36 0.95600308642 0.93775985107 0.913801965231
37 0.859802246094 0.880932095084 0.782658062563
38 0.904113769531 0.944340105856 0.637726276951
39 0.964123535156 0.950193593805 0.786052267598
40 0.930212402344 0.918021841766 0.870705837121
41 0.986831665039 0.952532123961 0.935907909395
42 0.839823717949 0.868505823078 0.717464092679
43 0.954282407407 0.990402811571 0.925184670749
44 0.993270874023 0.950012817226 0.94384311728
45 0.97871076481 0.974870362984 0.898800845844
46 0.994430541992 0.892811296534 0.884016523673
47 0.929061671088 0.686122188783 0.782858012788
48 0.975814819336 0.937899361022 0.855633482102
49 0.96103515625 0.98400250941 0.797204574333
50 0.994003295898 0.809190031153 0.840955078915
51 0.980529785156 0.668940609952 0.723210412148
52 0.154040527344 0.142570972487 0.116182678451
53 0.960684204102 0.900950407639 0.756405165731
54 0.947862654321 0.937856923549 0.90041120724
55 0.963635253906 0.802779207411 0.839640415568
56 0.925268015031 0.874902581894 0.837312256768
```

```
57 0.967939814815 0.955864667682 0.919857266853
58 0.956018066406 0.924381119159 0.865705020687
59 0.982070922852 0.933272394881 0.63475287535
60 0.995574951172 0.905826558266 0.902159244265
61 0.965329354553 0.970945784095 0.939413838057
62 0.997573852539 0.930870083433 0.907611853574
63 0.787580128205 0.316058892356 0.144315209528
64 0.929627577034 0.900619759388 0.795543103726
65 0.885168075562 0.0842899635606 0.155432419163
66 0.959320987654 0.95775295199 0.911326403606
67 0.995376586914 0.698232323232 0.784953867991
68 0.989410400391 0.727028479312 0.795882352941
69 0.783592224121 0.105670092915 0.190842961061
70 0.969523651636 0.98002002169 0.894962384535
71 0.987701416016 0.937206241775 0.925204157387
72 0.93200617284 0.893618161838 0.883272399724
73 0.989990234375 0.812095032397 0.851449275362
74 0.960632324219 0.880391173521 0.731305977921
75 0.964771220159 0.983054368856 0.922332817583
76 0.982678222656 0.969102792632 0.873360107095
77 0.99040222168 0.95767377839 0.898466505246
78 0.997680664063 0.958574181118 0.929038281979
79 0.996643066406 0.895683453237 0.918819188192
80 0.913757324219 0.764435946463 0.693585462116
81 0.99626159668 0.780973451327 0.852142426071
82 0.993988037109 0.893305439331 0.896587926509
83 0.971887207031 0.833089506583 0.844717146517
84 0.995422363281 0.99767261443 0.895543175487
85 0.948526234568 0.956312010086 0.912746059774
86 0.965539345712 0.984578783627 0.928269896592
87 0.979888916016 0.924585218703 0.866868686869
88 0.99250793457 0.937685459941 0.794302471722
89 0.926257079383 0.385687415441 0.553540137343
90 0.965707397461 0.933019755409 0.77927282013
91 0.961273193359 0.934991245704 0.819652094821
92 0.974668435013 0.931744749596 0.905322510688
93 0.99528503418 0.86406743941 0.841457157517
94 0.956130981445 0.877067669173 0.618750828803
95 0.916528320312 0.895421834329 0.773800860073
96 0.944934082031 0.72025649834 0.736059914575
97 0.994491577148 0.830484330484 0.865948756034
98 0.997055053711 0.870755750274 0.891755468312
99 0.96501159668 0.898277172536 0.860819423369
100 0.970840454102 0.908847663218 0.842079167011
101 0.809753417969 0.963739985133 0.651770751871
102 0.986340332031 0.974313774868 0.873659252569
103 0.989273071289 0.941265060241 0.925632074474
104 0.992233276367 0.879919678715 0.895931302392
105 0.96200617284 0.90534196168 0.896845016131
106 0.948834876543 0.947447611752 0.912631592817
107 0.953449074074 0.951535187986 0.899689074373
108 0.984252929688 0.939160239931 0.864353312303
109 0.931259155273 0.966878255208 0.8406268794
110 0.96692199707 0.829400780852 0.643254451502
111 0.83251953125 0.0799274340274 0.147925319256
112 0.973125 0.98916470536 0.927487352445
113 0.976028442383 0.901449275362 0.780862044916
```

```
114 0.995512820513 0.918578972636 0.921765102611
115 0.995254516602 0.9 0.870362651105
116 0.99446105957 0.827988338192 0.886668748049
117 0.921157836914 0.968920989276 0.631481349404
118 0.942822265625 0.900603845543 0.828763617752
119 0.992537024757 0.912825893938 0.893321221217
120 0.944807098765 0.908367312741 0.900932094234
121 0.974265030946 0.985526860491 0.930026745199
122 0.993698120117 0.813702848345 0.836565096953
123 0.993117263484 0.862539446816 0.881814299948
124 0.9 0.8233156649 0.857743201473
125 0.993408203125 0.87582659809 0.846590909091
126 0.983523983201 0.97181423395 0.921777229736
127 0.983734130859 0.944028206258 0.889350217978
128 0.94091796875 0.879844016573 0.881778212018
129 0.820159912109 0.952877697842 0.692078587104
130 0.972366333008 0.867140319716 0.729418795757
131 0.996292114258 0.918341708543 0.923271234607
132 0.956741333008 0.881519639408 0.658885813981
133 0.993560791016 0.790643274854 0.865003198976
134 0.924652099609 0.931374214371 0.689512072435
135 0.986511230469 0.943423497832 0.911794053083
136 0.989334106445 0.915775034294 0.905220338983
137 0.963927469136 0.911527686508 0.918532717609
138 0.934020996094 0.774792549179 0.831667133825
139 0.912487792969 0.766741429029 0.842290516312
140 0.990097045898 0.84171388102 0.879881547289
141 0.948217592593 0.951875056751 0.90359005301
142 0.981131189213 0.978120609813 0.916742864806
143 0.995849609375 0.927219796215 0.933561309233
144 0.950911803714 0.928180508881 0.844751651578
145 0.951867675781 0.852456066695 0.878185918626
146 0.990249633789 0.947139938713 0.920690083158
147 0.97151184082 0.946584938704 0.698433209498
148 0.932189941406 0.918799970659 0.849335503119
149 0.74963092804 0.0460766257307 0.0879558379567
150 0.85090637207 0.971898016997 0.637103064067
151 0.990325927734 0.787800129786 0.792945787067
152 0.873382568359 0.93172649219 0.684630586804
153 0.983047485352 0.964981830195 0.840212857759
154 0.992523193359 0.870455709165 0.874035989717
155 0.953137207031 0.853546393307 0.855060973308
156 0.963063271605 0.986078418185 0.924536927564
157 0.935211181641 0.951202956482 0.855921275874
158 0.991943359375 0.907549052212 0.911794186435
159 0.992309570313 0.926315789474 0.862745098039
160 0.982019042969 0.903639696256 0.824119402985
161 0.88688659668 0.951050514081 0.696598862195
162 0.996139526367 0.825974025974 0.882924571957
163 0.985885620117 0.883671854392 0.87868852459
164 0.97868347168 0.913701540096 0.831259813987
165 0.912756347656 0.875 0.811310293846
166 0.948638916016 0.986490885417 0.782670454545
167 0.959922839506 0.970884840599 0.932389159355
168 0.995498657227 0.558724832215 0.693028095734
169 0.966545092838 0.980501467058 0.923917961092
170 0.969816534041 0.987973555619 0.940234815244
```

```
171 0.971081542969 0.849731332222 0.885627383769
172 0.966320800781 0.87564344005 0.716545949556
173 0.989974975586 0.848201084278 0.839717004147
174 0.986419677734 0.941744284505 0.925907425907
175 0.546643066406 0.104758522727 0.160112168977
176 0.9388671875 0.811715481172 0.840468909276
177 0.80295085907 0.105991222636 0.191468630528
178 0.989791870117 0.809333333333 0.78398450113
179 0.971766975309 0.98573830728 0.945364411462
180 0.983162024757 0.941638002104 0.915761245196
181 0.939453125 0.813057583131 0.76380952381
182 0.960615605659 0.980424624969 0.893512431269
183 0.971250552608 0.970117919438 0.904797196527
184 0.962783813477 0.969872657625 0.884817001181
185 0.954772949219 0.933865706895 0.84854368932
186 0.907397460937 0.950318985114 0.872341141626
187 0.966887709991 0.976551168413 0.923868574187
188 0.988588638373 0.711119186047 0.791266552108
189 0.948742283951 0.897803279802 0.888407330881
190 0.880712890625 0.786333184457 0.843872823135
191 0.995788574219 0.869728915663 0.893271461717
192 0.991973876953 0.892956441149 0.879908675799
193 0.967453703704 0.976818841251 0.937520367353
194 0.980025972591 0.975710023207 0.924320306529
195 0.955352783203 0.890433661725 0.868600682594
196 0.96240234375 0.945222007722 0.835715809686
197 0.993942260742 0.955371900826 0.897283311772
198 0.993347167969 0.911801242236 0.870699881376
199 0.96212962963 0.951219512195 0.922881116244
200 0.873138427734 0.953259532595 0.736598656698
201 0.97158203125 0.845272206304 0.828039592259
202 0.982788085938 0.951341681574 0.790428061831
203 0.980365826702 0.958402330268 0.931131398887
204 0.875451660156 0.789106794312 0.833128894559
205 0.973038240495 0.97657163414 0.930329858632
206 0.987182617188 0.874867818118 0.855272226051
207 0.950637817383 0.935980228016 0.878906981097
208 0.947985839844 0.827955593066 0.85685490644
209 0.931882736516 0.893517442311 0.823863108634
210 0.775718688965 0.0726754077087 0.135108894724
211 0.959877873563 0.968046077518 0.875220197125
212 0.973278808594 0.922724296005 0.837382066711
213 0.946337890625 0.887932212655 0.855318588731
214 0.970164677277 0.977025835538 0.915810319824
215 0.995834350586 0.769516728625 0.858475894246
216 0.960185185185 0.910519152246 0.900746326075
217 0.959421296296 0.968968192397 0.92509293945
218 0.954567901235 0.942418132942 0.912886521675
219 0.995208740234 0.858546168959 0.893051771117
220 0.849877929687 0.824781897492 0.831080709851
221 0.6100172617 0.156551371421 0.270308905645
222 0.964105327144 0.982301835614 0.918447930595
223 0.994812011719 0.837209302326 0.855687606112
224 0.810587882996 0.0893986469096 0.16378755026
225 0.887234748011 0.937330784135 0.781648921941
226 0.957013888889 0.990146665995 0.933799151545
227 0.979202270508 0.888408304498 0.818870431894
```

```
228 0.990783691406 0.9659632403 0.903821656051
229 0.965502929687 0.893664047151 0.83741801864
230 0.980041503906 0.910294852574 0.84780079125
231 0.95198059082 0.934686672551 0.573750507924
232 0.949591049383 0.952193168719 0.90574639678
233 0.979187011719 0.894157630685 0.890124053488
234 0.993942260742 0.928223844282 0.884894172224
235 0.968533950617 0.990399543741 0.938763251945
236 0.770544433594 0.344323686246 0.473694526109
237 0.888229370117 0.958464034613 0.707549806364
238 0.947924382716 0.930670140397 0.851110767941
239 0.962939453125 0.916446073441 0.898934753662
240 0.994537353516 0.549450549451 0.62630480167
241 0.848358154297 0.947508812042 0.666845457593
242 0.987716674805 0.971919155786 0.93103743682
243 0.904769897461 0.928460823784 0.657708550431
244 0.876655061892 0.926529692371 0.768270842958
245 0.941759259259 0.945089285714 0.905087644292
246 0.997543334961 0.832409972299 0.881878209831
247 0.994003295898 0.929650613787 0.909258831679
248 0.958602905273 0.909386812264 0.864586972798
249 0.934645061728 0.928456238088 0.882033426184
250 0.97592285588 0.948319644487 0.916465355267
251 0.940910493827 0.874906819069 0.865535889872
252 0.961990740741 0.980964989703 0.931212645925
253 0.918933105469 0.947228528711 0.837441558759
254 0.963989257813 0.908847836655 0.760113844277
255 0.915826416016 0.917477398567 0.6801716141
256 0.983001708984 0.946138211382 0.89313123561
257 0.777010917664 0.0837036891757 0.154351371976
258 0.925280761719 0.814342437733 0.869010678594
259 0.991561889648 0.914997292907 0.859394863972
260 0.83104133606 0.0965549985195 0.175941430379
261 0.956799316406 0.799855816887 0.833779531257
262 0.966713638373 0.980182129874 0.930478691174
263 0.994924292661 0.868502475248 0.901654264147
264 0.992919921875 0.915350344269 0.906902086677
265 0.994857788086 0.917971195992 0.896910370144
266 0.986923217773 0.910671351951 0.919943951425
267 0.924682617188 0.942792709867 0.869113279593
268 0.947708129883 0.962394927002 0.792038351842
269 0.934432983398 0.983912483912 0.681018484151
270 0.985641479492 0.956271576525 0.925168986083
271 0.975643788683 0.980570797529 0.928023777057
272 0.976765583554 0.978535881169 0.924543031739
273 0.975296020508 0.894990366089 0.85159042992
274 0.993347167969 0.812460667086 0.855533465871
275 0.954606481481 0.93337446021 0.918804775378
276 0.935760498047 0.907567718514 0.606909430439
277 0.97883605957 0.949043676848 0.905549880831
278 0.899523925781 0.746034198599 0.778742506922
279 0.911936728395 0.908591961803 0.866118455781
280 0.993270874023 0.959258133171 0.934676344245
281 0.954390432099 0.900320502444 0.903934602071
282 0.989453125 0.872399445215 0.853459972863
283 0.766860961914 0.112797374897 0.201265094882
284 0.857223510742 0.959466724008 0.656055872082
```

```
285 0.991348266602 0.764080765143 0.835318036596
286 0.883569335937 0.802006728794 0.849586828991
287 0.983917236328 0.884382702218 0.882208314707
288 0.954284667969 0.846410128269 0.844405667041
289 0.954291003537 0.912271641598 0.872352410127
290 0.985220490716 0.970704399492 0.913554309356
291 0.955727785146 0.972175620934 0.922254300201
292 0.969830246914 0.997104177224 0.933731059964
293 0.984970092773 0.984013117442 0.906961367715
294 0.953472222222 0.930653529995 0.913714154885
295 0.935803222656 0.855088967972 0.872666521392
296 0.986053466797 0.575607064018 0.695333333333
297 0.909350585937 0.758114856429 0.82076655725
298 0.989105224609 0.946382978723 0.903330625508
299 0.981128426172 0.936593069336 0.902002984389
300 0.99284362793 0.848451327434 0.765851223165
301 0.948773148148 0.971975172636 0.915948194007
302 0.950298408488 0.973513853102 0.922409330895
303 0.982534814324 0.987849135729 0.958769543862
304 0.954429012346 0.94974781646 0.912723511157
305 0.930700683594 0.894691364692 0.85372703615
306 0.9154296875 0.844883258818 0.854898839693
307 0.93291015625 0.822364901017 0.848327629981
308 0.943930053711 0.929665457477 0.501072640869
309 0.994583129883 0.817537643933 0.838709677419
310 0.98779296875 0.921668667467 0.884759435321
311 0.939868164062 0.311920741451 0.442129105323
312 0.986785888672 0.933045356371 0.92291258679
313 0.988250732422 0.95678729577 0.942113967824
314 0.759778022766 0.0556815791846 0.1053196137
315 0.993194580078 0.921785421785 0.909862570736
316 0.943148148148 0.919334291058 0.902023882344
317 0.975748784262 0.977273489651 0.92995826384
318 0.959497070312 0.884757872453 0.889642785871
319 0.944020061728 0.992808503944 0.924118022362
320 0.888244628906 0.696721638781 0.792172708906
321 0.90412902832 0.94335477797 0.656797946141
322 0.987915039063 0.872549019608 0.745009658725
323 0.943472222222 0.945032236733 0.90350877193
324 0.949336419753 0.933255052222 0.901956099746
325 0.987966954023 0.92816845098 0.905619487246
326 0.938427734375 0.849665271967 0.889526479478
327 0.979156494141 0.940424410175 0.912637503198
328 0.980377197266 0.939234042553 0.895633825678
329 0.996459960938 0.773100616016 0.866513233602
330 0.994369506836 0.879536290323 0.904379372895
331 0.912506103516 0.948287059588 0.802928237558
332 0.992263793945 0.878703703704 0.882175226586
333 0.959182098765 0.972367544944 0.939244286207
334 0.944381713867 0.929201744599 0.834069285747
335 0.942044753086 0.912697741308 0.896034327635
336 0.978735632184 0.89750982962 0.914367099875
337 0.919104003906 0.92576419214 0.82942524002
338 0.743612289429 0.0586987542906 0.110754615878
339 0.995376586914 0.874637681159 0.888479941112
340 0.929150390625 0.921459974354 0.873952134822
341 0.983427276746 0.935571994229 0.910421458228
```

```
342 0.923715209961 0.890208484597 0.578657272405
343 0.990341186523 0.800621439669 0.830067114094
344 0.971508789063 0.972379706353 0.851451120163
345 0.995758056641 0.917302798982 0.912080961417
346 0.995483398438 0.916005291005 0.90345727332
347 0.977515432099 0.984240272444 0.949098658468
348 0.989883422852 0.918276374443 0.903112669882
349 0.910583496094 0.83064733648 0.87058075231
350 0.997444186561 0.668847691748 0.799131378936
351 0.95724537037 0.942247842693 0.905808557295
352 0.978810234306 0.985621391334 0.94829736599
353 0.770373535156 0.371206721933 0.512706266352
354 0.991821289063 0.84168618267 0.870217917676
355 0.987976074219 0.942563330825 0.905105973025
356 0.967600308642 0.957561638517 0.921547745829
357 0.983444213867 0.882029598309 0.884929472903
358 0.994201660156 0.915413533835 0.83676975945
359 0.993560791016 0.782908339076 0.843355605048
360 0.995834350586 0.837398373984 0.882983283326
361 0.989868164063 0.699811202014 0.770083102493
362 0.990447998047 0.823275862069 0.845964566929
363 0.966260499558 0.94867660646 0.895581608134
364 0.955606211317 0.943840624157 0.827249561861
365 0.810827255249 0.111923363262 0.201035952214
366 0.97361571618 0.961685823755 0.919406159533
367 0.992813110352 0.962366922506 0.942874469375
368 0.996490478516 0.942753293957 0.947488584475
369 0.996411132813 0.789087093389 0.836484983315
370 0.989807128906 0.811784654515 0.851489550912
371 0.996047973633 0.767466110532 0.850375505488
372 0.90125994695 0.945691655736 0.807560581583
373 0.6916015625 0.309520903265 0.445356750823
374 0.993103027344 0.959595959596 0.557729941292
375 0.971466064453 0.903753105793 0.880830996686
376 0.849383354187 0.0985186201221 0.179180807551
377 0.960220336914 0.943224632974 0.847784200385
378 0.953216552734 0.983712784588 0.785594405594
379 0.991760253906 0.837850689491 0.867125984252
380 0.986975097656 0.74618902439 0.78587196468
381 0.997848510742 0.744094488189 0.842809364548
382 0.95930480957 0.907138596142 0.872483863256
383 0.928332519531 0.859955947137 0.869280609178
384 0.926416015625 0.667599095346 0.788208839857
385 0.996307373047 0.816573556797 0.87875751503
386 0.9361328125 0.91045217803 0.746204220228
387 0.908801269531 0.66296793074 0.744310209111
388 0.966393125553 0.983578154426 0.92796437011
389 0.9765625 0.955312417611 0.904179663132
390 0.95321341733 0.887064101693 0.885144713727
391 0.946072530864 0.921769455528 0.893569068177
392 0.99299621582 0.861717612809 0.8376370711
393 0.996337890625 0.845849802372 0.842519685039
394 0.968700265252 0.984143961068 0.928553408346
395 0.907669067383 0.933223289316 0.67272432257
396 0.993850708008 0.872340425532 0.855088097807
397 0.991470336914 0.795415472779 0.832383808096
398 0.935058745289 0.945387881127 0.833801378606
```

```
399 0.980941772461 0.850900900901 0.858148779103
400 0.992904663086 0.859296482412 0.786206896552
401 0.997650146484 0.766666666667 0.80701754386
402 0.967987060547 0.896000932727 0.879894664529
403 0.995376586914 0.876811594203 0.888725670217
404 0.983840942383 0.914296407186 0.821867115223
405 0.890930175781 0.874948384843 0.860449497868
406 0.99333190918 0.951305220884 0.945490831982
407 0.956921296296 0.949560292145 0.919424439666
408 0.931640625 0.870647697492 0.832865755387
409 0.984878540039 0.939367502726 0.896824570536
410 0.986991600354 0.971331424481 0.920184450548
411 0.97041015625 0.932426502852 0.913192952299
412 0.989233398438 0.98642172524 0.848505668155
413 0.949637345679 0.908506800932 0.88107861893
414 0.961898803711 0.890530557421 0.761121209222
415 0.995376586914 0.896914446003 0.894092974484
416 0.759886364822 0.163228401332 0.280134927553
417 0.940354938272 0.994028518012 0.902122163695
418 0.994705200195 0.755842062853 0.843904633378
419 0.942291259766 0.91044221479 0.721543218966
420 0.891143798828 0.971043013776 0.659474940334
421 0.889221191406 0.945248416139 0.862021255569
422 0.99235534668 0.877725118483 0.880856123662
423 0.993728637695 0.832773109244 0.828249059758
424 0.973610190097 0.957452047374 0.890621958063
425 0.954602050781 0.859789760266 0.859432286351
426 0.995193481445 0.824652777778 0.857787810384
427 0.950863647461 0.912923978374 0.789675127036
428 0.921728515625 0.900667779633 0.864765681022
429 0.99186706543 0.921215880893 0.847844704539
430 0.9775390625 0.812975027144 0.765006385696
431 0.996368408203 0.899935856318 0.921813403417
432 0.966448386384 0.981887940729 0.924378016503
433 0.954189814815 0.944564357148 0.898427742896
434 0.964876215738 0.990208431376 0.911209208762
435 0.883251953125 0.979353966107 0.837352471004
436 0.935864257813 0.881543677073 0.89487795118
437 0.987335205078 0.842928216063 0.74078700812
438 0.957871905393 0.985400759928 0.915138393379
439 0.93974609375 0.794606214579 0.86824685031
440 0.967912798408 0.935997530685 0.898766508303
441 0.940383911133 0.9326171875 0.854363141611
442 0.966796875 0.989220038047 0.811373092926
443 0.996307373047 0.84991708126 0.894415357766
444 0.936340332031 0.790954826538 0.852687777181
445 0.9931640625 0.722849695917 0.787878787879
446 0.971524093722 0.97172495186 0.938192678597
447 0.967559814453 0.961059570313 0.881042972247
448 0.995452880859 0.999363462763 0.913321698662
449 0.947692871094 0.707634041289 0.816433191963
450 0.957438271605 0.981671810863 0.917938647386
451 0.92111882716 0.949026108579 0.870468684667
452 0.957885742188 0.809828009828 0.85145957117
453 0.945861816406 0.892703862661 0.778637384577
454 0.840841049383 0.256985605419 0.392984314764
455 0.956176757813 0.841389615241 0.862525848204
```

```
456 0.977767944336 0.914123006834 0.846356638195
457 0.9716796875 0.093296746374 0.170243204578
458 0.971411132813 0.830938384498 0.866992276238
459 0.990585327148 0.96821769838 0.937921320052
460 0.9875 0.86444493247 0.874382184706
461 0.994491577148 0.948275862069 0.927524593455
462 0.955679012346 0.927816665924 0.878726458914
463 0.944787597656 0.888729154687 0.845372807767
464 0.933904320988 0.909814459373 0.892478786966
465 0.990173339844 0.991662324127 0.855280898876
466 0.934938271605 0.742035903289 0.77232962523
467 0.974571728559 0.969020406958 0.934475368634
468 0.924621582031 0.894720821012 0.869519281564
469 0.95637433687 0.953636615044 0.897289280069
470 0.994476318359 0.860248447205 0.88441890166
471 0.989273071289 0.94119612429 0.889064225974
472 0.996231079102 0.780434782609 0.853238265003
473 0.966766975309 0.997706572308 0.933952860714
474 0.994705200195 0.815927873779 0.862246923382
475 0.992691040039 0.873411764706 0.885707468385
476 0.950447530864 0.937975778547 0.910156687185
477 0.993240356445 0.910447761194 0.852579034942
478 0.953464854111 0.908938000029 0.880664909446
479 0.993148803711 0.910390324354 0.880616857219
480 0.969930555556 0.995197764778 0.921255228435
481 0.995193481445 0.933908045977 0.925266903915
482 0.941774691358 0.890048927072 0.859614525971
483 0.977428713528 0.978146787615 0.927418280038
484 0.995513916016 0.882838283828 0.879211175021
485 0.990798950195 0.957842675459 0.930983175003
486 0.985443115234 0.863761955367 0.894957057917
487 0.952868652344 0.705359729599 0.790946992257
488 0.995086669922 0.795112781955 0.840119165839
489 0.986892700195 0.964187831171 0.924734951371
490 0.972152709961 0.819234194123 0.858142246405
491 0.914770507813 0.910040099871 0.775049938785
492 0.972319849691 0.981241772707 0.92592866438
493 0.977229938272 0.982719186785 0.940208692128
494 0.99891688771 0.984681064792 0.952392518824
495 0.933279320988 0.953472939095 0.856768978483
496 0.969623121132 0.980029048656 0.915267822736
497 0.997772216797 0.902847571189 0.880718954248
498 0.985977172852 0.891130924449 0.805172779309
499 0.938732910156 0.851429584121 0.85170630817
500 0.96630859375 0.0584461867427 0.10621761658
501 0.970901489258 0.925568573398 0.738013463388
502 0.983703613281 0.956859035005 0.904489357897
503 0.885894775391 0.954576363389 0.652444692322
504 0.989105224609 0.891280947255 0.698734177215
505 0.984497070313 0.945125628141 0.902495201536
506 0.930529785156 0.813688747306 0.859234707759
507 0.992935180664 0.89790897909 0.863139225539
508 0.972503662109 0.948662925058 0.864470517449
509 0.988052368164 0.764091858038 0.737160120846
510 0.946398925781 0.873414781437 0.895439933325
511 0.9939627542 0.724106463878 0.813359528487
512 0.952523148148 0.995904501873 0.910684995137
```

```
513 0.919519042969 0.85142293951 0.86281445723
514 0.987152099609 0.939368421053 0.913782510752
515 0.909416445623 0.941000023074 0.832659561436
516 0.989791870117 0.923186344239 0.795099540582
517 0.984756469727 0.972948104527 0.913678389355
518 0.808378219604 0.112445661067 0.20197470848
519 0.851806640625 0.959200507614 0.756871776899
520 0.969153404067 0.947451555581 0.923284131827
521 0.929138183594 0.957390146471 0.553289726818
522 0.980941772461 0.981481481481 0.433045846573
523 0.991241455078 0.846225104215 0.864238410596
524 0.997665405273 0.97037037037 0.895420369105
525 0.973388671875 0.939904799683 0.844590982
526 0.894393920898 0.964545742324 0.725389834544
527 0.991119384766 0.9229390681 0.898464759246
528 0.962484567901 0.980613029381 0.923476454294
529 0.991409194192 0.930246700007 0.934681042701
530 0.944043209877 0.925970042584 0.895567523977
531 0.951203703704 0.921760633037 0.898390051094
532 0.996612548828 0.940561724363 0.928433268859
533 0.908983645713 0.293551315132 0.451934022626
534 0.85126953125 0.918309002433 0.712492330926
535 0.917407226562 0.880326530612 0.761205618691
536 0.970620579134 0.978473534119 0.93863299263
537 0.965709876543 0.983576754791 0.93492077439
538 0.993148803711 0.867386276022 0.833642089663
539 0.986709594727 0.914888735949 0.90154854753
540 0.986846923828 0.937446928956 0.884851723217
541 0.985748291016 0.950241122315 0.902748854644
542 0.953680555556 0.945542521994 0.914837776107
543 0.95112654321 0.998953294294 0.917695366304
544 0.99577331543 0.886327503975 0.889509373753
545 0.989761352539 0.807311500381 0.759584378359
546 0.993087768555 0.852567642187 0.872070036713
547 0.9392578125 0.907197862643 0.822703627165
548 0.986679077148 0.749255706252 0.838423098279
549 0.994567871094 0.889105058366 0.885087153002
550 0.995162963867 0.841616964877 0.889044452223
551 0.989959716797 0.96718050721 0.922019435885
552 0.994232177734 0.771599657827 0.826764436297
553 0.996139526367 0.902649006623 0.91507217187
554 0.957484567901 0.950760230354 0.91887036928
555 0.978868258179 0.977553531229 0.951294690051
556 0.917556762695 0.988331495774 0.799271835643
557 0.95861882716 0.916947636414 0.908585746672
558 0.956080246914 0.934598378777 0.899117365567
559 0.942908950617 0.947619047619 0.892285743402
The average precision of the model is 78.3639440129741%
The average accuracy of the model is 95.20080933923725%
The average f1-score of the model is 78.23251191371496%
```

Classify Images for Future Segmentation

- Imaging conditions can vary from image to image. It can be difficult to determine a general method for nucleus segmentation under the variety of conditions. This section provides a method for separating images by type using a KMeans clustering algorithm.
- The function img_info determines a

In [35]:
```
# one-indexes a 2d array into 1d, top down then left right, output is np 1d ar
ray
def one_index(arr2d):
    h, w = arr2d.shape[0:2]

    arr1d = []
    for col in range(0, w):
        for row in range(0, h):
            arr1d.append(arr2d[row][col])
    return np.array(arr1d)

# pads all vectors in array to have max_len, returns np array
def pad_normalize(array, max_len):
    for i in range(0, len(array)):
        vec = array[i]
        print(len(vec))
        if len(vec) < max_len:
            array[i] = np.concatenate(( np.array(vec).reshape(1,-1), np.zeros
((1, (max_len-len(vec)))) ), axis=1)
        else:
            array[i] = np.array(vec).reshape(1,-1)
    return np.array(array)
```

In [71]:
```
# Get image information, apply clustering
def get_image_info(samples):
    # complete img ,rgb mode
    full_img_list = []
    # just grey mode
    img_list = []

    # average value of gray pixels per image
    average_list = []

    # max Contour area value per image
    max_cnt_area = []

    # mean Contour area value per image
    average_cnt_area = []

    # how many Contour areas per image
    num_cnt = []

    #  width per image
```

```
    wid_list = []

    #  length per image
    len_list = []

    #  red per image
    r=[]

    #  green
    g=[]

    #  blue
    b=[]
    for i in samples:
        (img, masks) = load_zipped_img(path+'/stage1_train.zip', i)
        full_img_list.append(i)
        r.append(np.average(img[:,:,0]))
        g.append(np.average(img[:,:,1]))
        b.append(np.average(img[:,:,2]))

        img = grayscale(img)
        img = float2int8(img)
        #img = cv2.imread(img,cv2.IMREAD_GRAYSCALE)
        print(i)
        # in some cases, image background is bright and cell darker, there nee
ds a inverse of pixel value
        if np.average(img) > 125:
            img = 255 - img
        img_list.append(img)

        lenth = img.shape[0]
        len_list.append(lenth)
        width = img.shape[1]
        wid_list.append(width)
        average_list.append(np.average(img))

        # use opencv to find contour and get some stactistic data
        img = cv2.GaussianBlur(img, (3, 3), 1)
        ret, thresh = cv2.threshold(img, 0, 255, cv2.THRESH_OTSU)

        _, cnts, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_
SIMPLE)
        cnts = sorted(cnts, key=cv2.contourArea, reverse=True)
        max_cnt_area.append(cv2.contourArea(cnts[0])/lenth/width)

        av = 0
        for i in cnts:
            av = av + cv2.contourArea(i)
        av = av/len(cnts)

        # since different pic has different size, we'd better normalise it
        average_cnt_area.append(av/lenth/width)
        num_cnt.append(len(cnts))

    df = pd.DataFrame({'img':full_img_list,'max_area':max_cnt_area,'average_ar
ea':average_cnt_area,
                       'num_cnt':num_cnt,'average':average_list,'wid':wid_list
```

```
,'len':len_list,
                        'r':r,'g':g,'b':b
                       })
    return df

df = get_image_info(range(0,560))
```

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
```

In [87]:
```
df.to_csv('KMEANS_summary.csv')
df
```

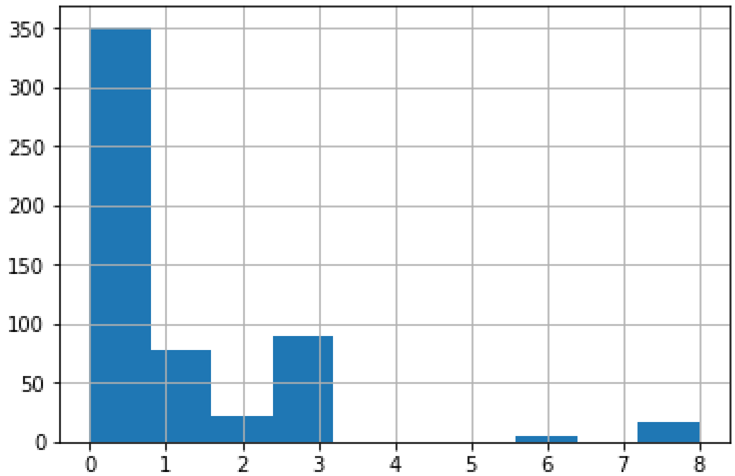
Out[87]:

	average	average_area	b	g	img	len	max_area	num_cnt	
0	6.142227	0.002843	0.024087	0.024087	0	256	0.008263	20	0.024
1	9.018600	0.001648	0.035367	0.035367	1	256	0.004219	36	0.035
2	93.074377	0.006824	0.756961	0.604796	2	256	0.031854	45	0.695
3	81.215198	0.004652	0.802636	0.662910	3	256	0.033124	53	0.702
4	92.987366	0.008269	0.786623	0.604949	4	256	0.021552	8	0.686
5	15.242493	0.002391	0.059774	0.059774	5	256	0.006172	17	0.059
6	16.667801	0.004530	0.065364	0.065364	6	256	0.022232	39	0.065
7	10.765440	0.011759	0.042217	0.042217	7	360	0.026779	18	0.042
8	16.871002	0.001431	0.066161	0.066161	8	256	0.003487	11	0.066
9	11.848765	0.013118	0.046466	0.046466	9	360	0.030363	18	0.046
10	10.967122	0.011311	0.043008	0.043008	10	360	0.028152	22	0.043
11	41.003713	0.001948	0.160799	0.160799	11	260	0.014038	78	0.160
12	17.424622	0.002013	0.068332	0.068332	12	256	0.006447	69	0.068
13	17.616776	0.002971	0.069085	0.069085	13	256	0.006477	24	0.069
14	8.932731	0.011968	0.035030	0.035030	14	360	0.036238	19	0.035
15	19.130478	0.001559	0.075021	0.075021	15	256	0.006889	20	0.075
16	6.385361	0.003363	0.025041	0.025041	16	256	0.008949	15	0.025
17	13.160849	0.009568	0.051611	0.051611	17	360	0.034024	26	0.051
18	4.869278	0.001752	0.019095	0.019095	18	256	0.004295	9	0.019
19	13.656174	0.001205	0.053554	0.053554	19	256	0.001526	5	0.053
20	4.103210	0.001934	0.016091	0.016091	20	256	0.002579	7	0.016
21	44.637525	0.001336	0.175049	0.175049	21	520	0.017916	158	0.175
22	11.913171	0.007606	0.046718	0.046718	22	360	0.023819	27	0.046
23	49.972590	0.000765	0.804029	0.804029	23	1024	0.042792	285	0.804
24	13.539978	0.002114	0.053098	0.053098	24	256	0.003540	7	0.053
25	39.509685	0.000499	0.845060	0.845060	25	1024	0.031569	313	0.845
26	19.444366	0.000888	0.076252	0.076252	26	256	0.003250	48	0.076
27	8.581192	0.003105	0.033652	0.033652	27	256	0.011017	29	0.033
28	15.459991	0.002969	0.060627	0.060627	28	256	0.005096	9	0.060
29	10.707253	0.000910	0.041989	0.041989	29	520	0.001405	7	0.041
...	...	...	...	...	...	...	...	...	...
530	14.649954	0.013698	0.057451	0.057451	530	360	0.051968	19	0.057

	average	average_area	b	g	img	len	max_area	num_cnt	
531	11.667593	0.011300	0.045755	0.045755	531	360	0.042519	21	0.045
532	5.318527	0.001477	0.020857	0.020857	532	256	0.003426	14	0.020
533	4.768578	0.000778	0.018700	0.018700	533	603	0.141384	198	0.018
534	109.992847	0.001993	0.804628	0.525771	534	256	0.052612	111	0.633
535	105.444226	0.002133	0.802869	0.547219	535	256	0.015259	73	0.645
536	10.283626	0.001660	0.040328	0.040328	536	520	0.005555	134	0.040
537	12.737670	0.012654	0.049952	0.049952	537	360	0.024618	20	0.049
538	3.725769	0.002308	0.014611	0.014611	538	256	0.004410	8	0.014
539	18.685562	0.001206	0.073277	0.073277	539	256	0.003036	50	0.073
540	16.692749	0.002115	0.065462	0.065462	540	256	0.007034	24	0.065
541	9.459320	0.001871	0.037095	0.037095	541	256	0.008713	35	0.037
542	11.870787	0.012394	0.046552	0.046552	542	360	0.045467	22	0.046
543	12.611875	0.011758	0.049458	0.049458	543	360	0.030096	24	0.049
544	16.025085	0.002214	0.062843	0.062843	544	256	0.003166	8	0.062
545	18.036499	0.001205	0.070731	0.070731	545	256	0.003441	16	0.070
546	13.798157	0.001833	0.054110	0.054110	546	256	0.005379	14	0.054
547	20.384412	0.006652	0.079939	0.079939	547	256	0.113544	24	0.079
548	18.372604	0.002701	0.072049	0.072049	548	256	0.004974	16	0.072
549	5.101028	0.001887	0.020004	0.020004	549	256	0.003365	11	0.020
550	5.940491	0.002115	0.023296	0.023296	550	256	0.004662	10	0.023
551	10.402206	0.001809	0.040793	0.040793	551	256	0.005920	31	0.040
552	4.996063	0.002335	0.019592	0.019592	552	256	0.003830	7	0.019
553	4.743835	0.002055	0.018603	0.018603	553	256	0.003914	10	0.018
554	10.327932	0.013414	0.040502	0.040502	554	360	0.048264	19	0.040
555	8.276042	0.001775	0.032455	0.032455	555	520	0.004642	113	0.032
556	13.301880	0.002986	0.052164	0.052164	556	256	0.008171	52	0.052
557	11.551404	0.011880	0.045300	0.045300	557	360	0.025058	19	0.045
558	8.408935	0.011111	0.032976	0.032976	558	360	0.023954	19	0.032
559	9.683364	0.011822	0.037974	0.037974	559	360	0.042052	22	0.037

560 rows × 13 columns

In [83]:
```
from sklearn.cluster import KMeans

# train seperately
input_x = np.array(df[['max_area','average_area','num_cnt','average','wid',
'len']])

fkmeans = KMeans(n_clusters = 3).fit(input_x)

df['flabel'] = fkmeans.labels_

input_c = np.array(df[['r','g','b']])

ckmeans = KMeans(n_clusters = 3).fit(input_c)

df['clabel'] = ckmeans.labels_

# and then make an combination
df['cflabel'] = 3 *df['flabel']

df['cflabel'] = df['cflabel'] + df['clabel']

df['cflabel'].hist()
plt.show()
df['cflabel'].values
```




```
Out[83]: array([0, 0, 1, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0,
                8, 0, 8, 0, 0, 0, 3, 0, 0, 1, 0, 2, 0, 0, 1, 0, 1, 1, 0, 3, 0, 0, 3,
                0, 3, 0, 1, 0, 0, 1, 3, 0, 2, 3, 0, 2, 0, 0, 3, 0, 3, 0, 8, 0, 0, 0,
                8, 3, 0, 0, 0, 0, 3, 2, 0, 0, 0, 1, 0, 0, 2, 0, 0, 3, 0, 0, 6, 3, 3,
                3, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 3, 8, 0, 0, 3,
                0, 0, 0, 2, 3, 0, 3, 0, 3, 1, 0, 3, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                2, 1, 0, 0, 3, 0, 3, 2, 0, 0, 0, 8, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
                0, 0, 0, 0, 1, 0, 0, 0, 3, 3, 1, 3, 0, 0, 1, 2, 8, 0, 0, 3, 1, 3, 3,
                0, 0, 1, 3, 3, 0, 1, 0, 0, 0, 3, 0, 1, 0, 0, 0, 0, 1, 1, 3, 1, 3, 0,
                0, 1, 3, 8, 3, 1, 2, 3, 0, 0, 0, 0, 0, 1, 6, 3, 0, 8, 3, 0, 0, 0, 2,
                0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 3, 0, 0,
                1, 0, 3, 0, 8, 1, 0, 8, 1, 3, 3, 0, 0, 0, 1, 0, 0, 0, 3, 3, 0, 0, 0,
                0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 3, 3, 3, 0, 0, 0, 1, 0, 1, 0,
                3, 0, 0, 3, 3, 0, 1, 1, 2, 3, 0, 0, 1, 0, 0, 8, 0, 0, 3, 2, 0, 1, 0,
                0, 0, 0, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 3, 2, 8, 0, 1, 3, 3, 0, 1,
                0, 0, 0, 0, 1, 3, 0, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 8, 3, 0,
                0, 1, 0, 0, 3, 1, 0, 0, 8, 0, 0, 0, 2, 0, 0, 1, 2, 0, 3, 1, 3, 0, 3,
                0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 3, 2, 1, 0,
                0, 0, 6, 0, 0, 0, 0, 1, 0, 0, 3, 2, 0, 3, 1, 0, 0, 0, 3, 0, 3, 1, 1,
                0, 3, 1, 3, 0, 0, 0, 1, 0, 3, 0, 0, 1, 0, 0, 1, 3, 0, 1, 0, 1, 2, 0,
                3, 0, 0, 1, 0, 0, 0, 3, 1, 3, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0,
                3, 0, 0, 0, 1, 0, 0, 0, 2, 3, 0, 3, 0, 3, 0, 0, 1, 1, 0, 0, 0, 0, 0,
                1, 0, 0, 0, 1, 3, 0, 1, 0, 3, 0, 0, 8, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0,
                6, 0, 0, 0, 6, 1, 1, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 3, 0, 0, 0, 0])
```

Segmentation: Separating Individual Objects

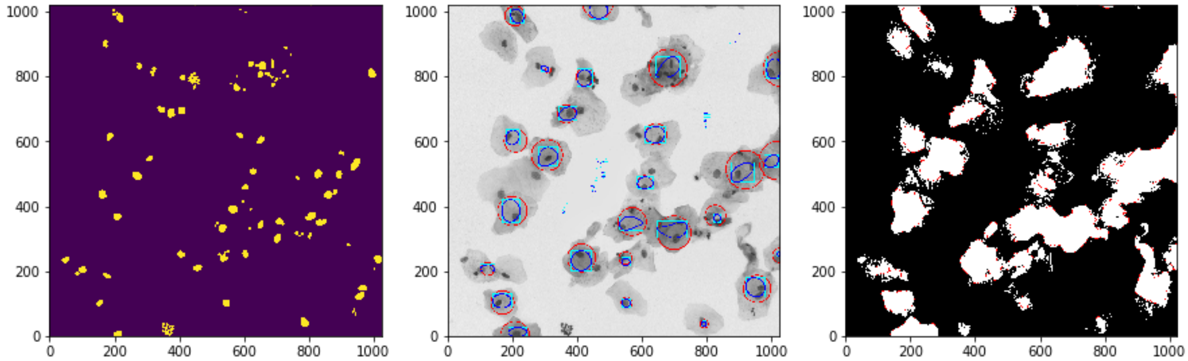
- The function separate_obj separates the objects in an image after a thresholding method has been applied
- The function convert2runlength finds the objects in an image (1 corresponds to object, 0 to background) and finds runs of continuous object pixels
- The function rle generates a dataframe of images in run-length format. This is the output format required by the Kaggle competition

In [38]:
```
## STEP 3: Separate individual objects and encode in run-length format

# separate objects in image into individual masks
def separate_obj(img_masked):
    labels, nlabels = ndimage.label(img_masked)

    label_arrays = []
    for label_num in range(1, nlabels+1):
        label_mask = np.where(labels == label_num, 1, 0)
        label_arrays.append(label_mask)
    return labels, nlabels, label_mask

# convert path to run-length encoding (RLE) output format
def convert2runlength(x):
    obj = np.where(x.T.flatten()==1)[0] #1 corresponds to object, 0 to backgro
und
    run_lengths = []
    prev = -2
    for b in obj: # find continuous set of object pixels
        if (b>prev+1): run_lengths.extend((b+1, 0))
        run_lengths[-1] += 1
        prev = b
    return " ".join([str(i) for i in run_lengths])

def rle(img_masked, im_id):
    (labels, nlabels, label_mask) = separate_obj(img_masked)
    im_df = pd.DataFrame()
    for label_num in range(1, nlabels+1):
        label_mask = np.where(labels == label_num, 1, 0)
        if label_mask.flatten().sum() > 10:
            rle = convert2runlength(label_mask)
            s = pd.Series({'ImageId': im_id, 'EncodedPixels': rle})
            im_df = im_df.append(s, ignore_index=True)
    return im_df
```

In [39]:
```
# Function to contour segment images
def contouring(img_float32, borderSize, gap, draw='off'):
    img = float2int8(img_float32[:, :, :3])

    # perform a BGR->HSV conversion and use the V channel for processing
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    # threshold, apply morphological closing, then take the distance transfo
rm (dist)
    th, bw = cv2.threshold(float2int8( hsv[:, :, 2] ), 0, 255, cv2.THRESH_BI
NARY | cv2.THRESH_OTSU)
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
    morph = cv2.morphologyEx(bw, cv2.MORPH_CLOSE, kernel)
    dist = cv2.distanceTransform(morph, cv2.DIST_L2, cv2.DIST_MASK_PRECISE)
    distborder = cv2.copyMakeBorder(dist, borderSize, borderSize, borderSize
, borderSize, cv2.BORDER_CONSTANT | cv2.BORDER_ISOLATED, 0)

    # create a template, take its distance transform and use it as the templ
ate (temp)
    kernel2 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2*(borderSize-ga
```

```
p)+1, 2*(borderSize-gap)+1))
    kernel2 = cv2.copyMakeBorder(kernel2, gap, gap, gap, gap, cv2.BORDER_CON
STANT | cv2.BORDER_ISOLATED, 0)
    distTempl = cv2.distanceTransform(kernel2, cv2.DIST_L2, cv2.DIST_MASK_PR
ECISE)

    # template matching (dist*temp)
    nxcor = cv2.matchTemplate(distborder, distTempl, cv2.TM_CCOEFF_NORMED)

    # find local maxima of the resulting image, positions correspond to circ
le centers and values correspond to radii
    mn, mx, _, _ = cv2.minMaxLoc(nxcor)

    # thresholdig template matched image
    th, peaks = cv2.threshold(nxcor, mx*0.5, 255, cv2.THRESH_BINARY)
    peaks8u = cv2.convertScaleAbs(peaks)
    _, contours, _ = cv2.findContours(peaks8u, cv2.RETR_CCOMP, cv2.CHAIN_APP
ROX_SIMPLE)
    # peaks8u = cv2.convertScaleAbs(peaks)    # to use as mask

    # detecting circles as local maxima
#     if draw == 'on':
    for i in range(len(contours)):
        x, y, w, h = cv2.boundingRect(contours[i])
        _, mx, _, mxloc = cv2.minMaxLoc(dist[y:y+h, x:x+w], peaks8u[y:y+h, x
:x+w])
        cv2.circle(img, (int(mxloc[0]+x), int(mxloc[1]+y)), int(mx), (255, 0
, 0), 2)
        cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 255), 2)
        cv2.drawContours(img, contours, i, (0, 0, 255), 2)

    img_contoured = img

    return img_contoured

#### TESTING
(img, mask) = load_zipped_img(path+'/stage1_train.zip', 177)
#177
img_guess, markers, sure_bg, sure_fg, uncertain = watershed(img)

# one way of determining borderSize and gap
num_markers = np.max(markers)
obj_size = []
for i in range(1, num_markers):
    obj_size.append(len(markers[markers==i]))

min_size = min(a for a in obj_size)
max_size = max(a for a in obj_size)
avg_size = np.sum(obj_size)/num_markers

borderSize = int(np.sqrt(5000))
gap = int(borderSize/10)

img_contoured = contouring(img, borderSize, gap, draw='off')

fig, ax = plt.subplots(1,3, figsize = (15,15))
```

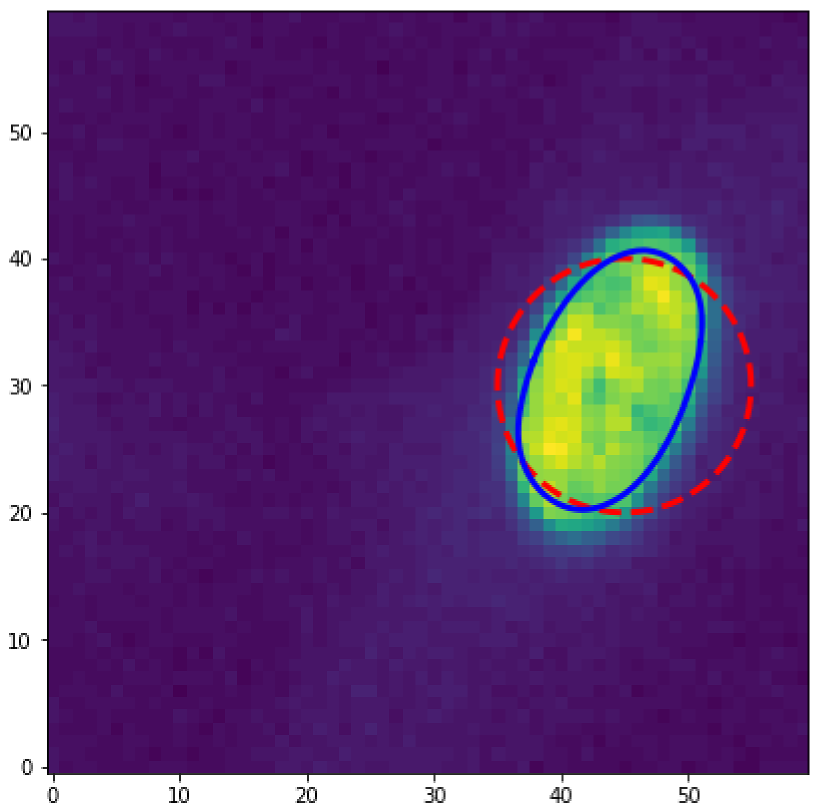
```
ax[0].imshow(sum(mask), origin='lower')
ax[1].imshow(img_contoured, origin='lower')
ax[2].imshow(img_guess, origin='lower')
plt.show()
```



Active Contour Modeling

- This method applies level set methods for nucleus segmentation

In [24]:
```
from skimage.segmentation import active_contour
from skimage.filters import gaussian

(img, mask) = load_zipped_img(path+'/stage1_train.zip', 1)
img = grayscale(img)[100:160,100:160]

s = np.linspace(0, 2*np.pi, 400)
x = 45 + 10*np.cos(s)
y = 30 + 10*np.sin(s)
init = np.array([x, y]).T

snake = active_contour(gaussian(img, 3), init, alpha=0.015, beta=10, gamma=
0.001)
fig, ax = plt.subplots(figsize=(7, 7))
ax.imshow(img, origin = 'lower')
ax.plot(init[:, 0], init[:, 1], '--r', lw=3)
ax.plot(snake[:, 0], snake[:, 1], '-b', lw=3)
#ax.set_xticks([]), ax.set_yticks([])
#ax.axis([0, img.shape[1], img.shape[0], 0])
plt.show()
```



Data Shape Manipulation

- The function one_index takes an image (2d array) and converts it to a 1d array. They are indexed from top to bottom then left to right
- The function pad_normalize helps account for variation in image sizes. It determines the maximum length in a set of one-indexed images and "pads" all other one-indexed images with zeros so that all images have the same length.

In [40]:
```
# one-indexes a 2d array into 1d, top down then left right, output is np 1d ar
ray
def one_index(arr2d):
    h, w = arr2d.shape[0:2]

    arr1d = []
    for col in range(0, w):
        for row in range(0, h):
            arr1d.append(arr2d[row][col])
    return np.array(arr1d)

# pads all vectors in array to have max_len, returns np array
def pad_normalize(array, max_len):
    for i in range(0, len(array)):
        vec = array[i]
        if len(vec) < max_len:
            array[i] = np.concatenate(( np.array(vec).reshape(1,-1), np.zeros
((1, (max_len-len(vec)))) ), axis=1)
        else:
            array[i] = np.array(vec).reshape(1,-1)
    return np.array(array)
```

In [35]:
```
#Separate the cells
```

Training a Model

- X = a vector of the one-indexed images in the training set
- Y = a vector containing the sum of the one-indexed masks for each image (The correct nuclei)
- Currently fits a Random Forest Classifier with a maximum depth of 4. The feature vector contains the following information:
    1. The grayscale pixel intensity (continuous)
    2. The watershed prediction for a pixel (discrete)
    3. The magnitude of the gradient of pixel intensity (calculated using the Sobel operator) 4-6. The intensity for the rgb color channels respectively (continuous)

Model Validation

- Tests the performance of the model
- Uses cross-validation for tuning of hyperparameters?
- Uses confusion matrix to quantify types of errors (fp, fn, tp, tn)

In [53]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier

# Not rigorous but I picked a few images to train on - it went pretty well

n_samples = [0, 19, 50, 100, 149, 150, 175, 177, 200, 250, 300, 350, 400, 51
0, 560, 650]
#n_samples = range(0,560)
x_train = np.zeros(7).reshape(1,7) # predicted segmentation using Otsu's thr
esholding

y_train = np.zeros(1) # "correct" segmentation from sum of masks

max_len = 0
conm = []
acc = []
for i in n_samples:
    (img, masks) = load_zipped_img(path+'/stage1_train.zip', i) # Loads imag
e and associated masks

    print(i)
    (img_guess, markers, sure_bg, sure_fg, uncertain) = watershed(img)

    intensity=grayscale(img)
    #img_guess = otsu(intensity)
    img_guess=grayscale(img_guess)
    #img_guess = np.dot(0,intensity)
    laplac = laplace(grayscale(img))[uncertain==255].reshape(-1,1)
    fil = sobel(grayscale(img))[uncertain==255].reshape(-1,1)
    #fil = sobel(grayscale(img)).reshape(-1,1)
    intensity_raw = intensity[uncertain==255].reshape(-1,1)
    #intensity_raw = intensity.reshape(-1,1)
    watershed_raw = img_guess[uncertain==255].reshape(-1,1)
    #watershed_raw = img_guess.reshape(-1,1)

    img3 = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    ch1,ch2,ch3 = cv2.split(img3)
    #ch1 = ch1.reshape(-1,1)
    #ch2 = ch2.reshape(-1,1)
    #ch3 = ch3.reshape(-1,1)

    ch1 = ch1[uncertain==255].reshape(-1,1)
    ch2 = ch2[uncertain==255].reshape(-1,1)
    ch3 = ch3[uncertain==255].reshape(-1,1)

    feature = np.concatenate((intensity_raw,watershed_raw, fil, laplac, ch1,
 ch2, ch3), axis = 1) # Features include grayscale intensity, watershed, sob
```

```
el, rgb channels


    y_raw = sum(masks)
    y_raw = y_raw[uncertain==255].reshape(-1,1)


    x_train = np.append(x_train,feature).reshape(-1,7)
    y_train = np.append(y_train,y_raw).reshape(-1,1)


Xtrain, Xtest, ytrain, ytest = train_test_split(x_train,y_train)
# Kept a split training set around but not needed

#clf = svm.SVC(C = 1.0)
clf = RandomForestClassifier(max_depth=5, random_state=0)
%time clf.fit(Xtrain, ytrain[:,0]) #599 ms for a training set of this size!


    #y_pred = clf.predict(Xtest)
    #cm = confusion_matrix(ytest, y_pred)
    #print(cm)
    #conm.append(cm)
    #accuracy = sklearn.metrics.accuracy_score(ytest, y_pred)
    #print(accuracy)
    #acc.append(accuracy)

    #(img_guess, markers, sure_bg, sure_fg, unknown, reduced_area) = watersh
ed(img)

    #fig, ax = plt.subplots(1,4, figsize = (10,10))

    #ax[0].imshow(sum(masks), origin='lower')
    #ax[1].imshow(grayscale(img), origin='lower')
    #ax[2].imshow(output, origin='lower')
    #ax[3].imshow(img_guess, origin='lower')
    #plt.show()

#A = np.array(acc)
#Avg_acc = np.mean(A)
#print(Avg_acc)
```

```
0
19
50
100
149
150
175
177
200
250
300
350
400
510
560
650
Wall time: 4.87 s
```

```
Out[53]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=5, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                     oob_score=False, random_state=0, verbose=0, warm_start=False)
```

In [28]:
```
# Random Forest Except using HSV instead of RGB

# Not rigorous but I picked a few images to train on - it went pretty well
from sklearn.ensemble import RandomForestClassifier
n_samples = [0, 19, 50, 100, 149, 150, 175, 177, 200, 250, 300, 350, 400, 51
0, 560, 650]
#n_samples = range(0,560)
x_train = np.zeros(6).reshape(1,6)

y_train = np.zeros(1) # "correct" segmentation from sum of masks

max_len = 0
conm = []
acc = []
for i in n_samples:
    (img, masks) = load_zipped_img(path+'/stage1_train.zip', i) # loads imag
e and associated masks

    print(i)
    (img_guess, markers, sure_bg, sure_fg, uncertain) = watershed(img)

    intensity=grayscale(img)
    #img_guess = otsu(intensity)
    img_guess=grayscale(img_guess)
    #img_guess = np.dot(0,intensity)
    fil = sobel(grayscale(img))[uncertain==255].reshape(-1,1)
    #fil = sobel(grayscale(img)).reshape(-1,1)
    intensity_raw = intensity[uncertain==255].reshape(-1,1)
    #intensity_raw = intensity.reshape(-1,1)
    watershed_raw = img_guess[uncertain==255].reshape(-1,1)
    #watershed_raw = img_guess.reshape(-1,1)
```

```
    img_cv = float2int8(img[:, :, :3])

    # perform a BGR->HSV conversion and use the V channel for processing
    hsv = cv2.cvtColor(img_cv, cv2.COLOR_BGR2HSV)

    #img3 = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    ch1,ch2,ch3 = cv2.split(hsv)
    #ch1 = ch1.reshape(-1,1)
    #ch2 = ch2.reshape(-1,1)
    #ch3 = ch3.reshape(-1,1)

    print(ch1.shape)

    ch1 = ch1[uncertain==255].reshape(-1,1)
    ch2 = ch2[uncertain==255].reshape(-1,1)
    ch3 = ch3[uncertain==255].reshape(-1,1)

    feature = np.concatenate((intensity_raw,watershed_raw, fil, ch1, ch2, ch
3), axis = 1) # Features include grayscale intensity, watershed, sobel, rgb
 channels


    y_raw = sum(masks)
    y_raw = y_raw[uncertain==255].reshape(-1,1)


    x_train = np.append(x_train,feature).reshape(-1,6)
    y_train = np.append(y_train,y_raw).reshape(-1,1)


Xtrain, Xtest, ytrain, ytest = train_test_split(x_train,y_train)
# Kept a split training set around but not needed

#s_v = svm.SVC(C = 1.0)
clf = RandomForestClassifier(max_depth=5, random_state=0)
%time clf.fit(Xtrain, ytrain[:,0]) #599 ms for a training set of this size!


    #y_pred = clf.predict(Xtest)
    #cm = confusion_matrix(ytest, y_pred)
    #print(cm)
    #conm.append(cm)
    #accuracy = sklearn.metrics.accuracy_score(ytest, y_pred)
    #print(accuracy)
    #acc.append(accuracy)

    #(img_guess, markers, sure_bg, sure_fg, unknown, reduced_area) = watersh
ed(img)

    #fig, ax = plt.subplots(1,4, figsize = (10,10))

    #ax[0].imshow(sum(masks), origin='lower')
    #ax[1].imshow(grayscale(img), origin='lower')
    #ax[2].imshow(output, origin='lower')
    #ax[3].imshow(img_guess, origin='lower')
    #plt.show()
```

```
#A = np.array(acc)
#Avg_acc = np.mean(A)
#print(Avg_acc)
```

```
0
(256, 256)
19
(256, 256)
50
(256, 256)
100
(256, 256)
149
(1024, 1024)
150
(256, 256)
175
(256, 320)
177
(1024, 1024)
200
(256, 256)
250
(520, 696)
300
(256, 256)
350
(520, 696)
400
(256, 256)
510
(256, 320)
560
(256, 256)
650
(256, 256)
Wall time: 8.93 s
```

Out[28]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

In [55]:
```
# Testing the classifier on the remaining images
acc = []
for i in range(0,560):
    (img, masks) = load_zipped_img(path+'/stage1_train.zip', i)

    (img_guess, markers, sure_bg, sure_fg, unknown) = watershed(img)
    intensity = grayscale(img)
    shap = intensity.shape
    intensity_raw = intensity.reshape(-1,1)
    #watershed_raw = img_guess[unknown==255].reshape(-1,1)
    watershed_raw = grayscale(img_guess).reshape(-1,1)
    fil = sobel(grayscale(img)).reshape(-1,1)
    laplac = laplace(grayscale(img)).reshape(-1,1)
    #img3 = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    #img3 = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    ch1,ch2,ch3 = cv2.split(hsv)


    #ch1,ch2,ch3 = cv2.split(img3)
    ch1 = ch1.reshape(-1,1)
    ch2 = ch2.reshape(-1,1)
    ch3 = ch3.reshape(-1,1)
    #print(ch1.shape)
    #print(intensity_raw.shape)
    feature = np.concatenate((intensity_raw,watershed_raw, fil, laplac, ch1,
 ch2, ch3), axis = 1)

    yp = clf.predict(feature)
    y_real = sum(masks).reshape(-1,1)
    accuracy = sklearn.metrics.accuracy_score(y_real, yp)
    precision = sklearn.metrics.precision_score(y_real, yp)
    f1 = sklearn.metrics.f1_score(y_real, yp)
    print(i, accuracy, precision, f1)
    acc.append([accuracy, precision, f1])

    #output = yp.reshape(shap[0],shap[1])

precision = np.mean(np.array(acc)[:,1])
accuracy = np.mean(np.array(acc)[:,0])
f1 = np.mean(np.array(acc)[:,2])
print('The average precision of the model is {}%'.format(precision*100))
print('The average accuracy of the model is {}%'.format(accuracy*100))
print('The average f1-score of the model is {}%'.format(f1*100))
```

```
0 0.975708007813 0.894091187896 0.846005029986
1 0.988327026367 0.898442367601 0.91876393756
2 0.714721679688 0.89941083489 0.348807400802
3 0.835083007813 0.676328502415 0.212244897959
4 0.934167480469 0.728658536585 0.0814171350707
5 0.979019165039 0.692124105012 0.808362369338
6 0.93537902832 0.905896242795 0.860337037892
7 0.97013117284 0.979405737705 0.936774193548
8 0.97004699707 0.229555236729 0.245870149827
9 0.97112654321 0.971032041729 0.943552766548
10 0.952330246914 0.891814006544 0.905817427892
11 0.911483041454 0.823177758571 0.741084165478
12 0.961273193359 0.831437032419 0.893700787402
13 0.974761962891 0.786783854167 0.853912736266
14 0.954444444444 0.947524069267 0.907134767837
15 0.959701538086 0.432483474976 0.409568522245
16 0.985092163086 0.8735395189 0.886434964547
17 0.961404320988 0.989761290135 0.93171331058
18 0.991958618164 0.655839668279 0.782680412371
19 0.980895996094 0.153968253968 0.236585365854
20 0.994262695313 0.702702702703 0.824626865672
21 0.844473916888 0.826363929306 0.703184982071
22 0.963163580247 0.928815142046 0.915498442368
23 0.945902824402 0.297616653258 0.454970838898
24 0.980590820313 0.406043437205 0.57486631016
25 0.976623535156 0.328990820129 0.469781527147
26 0.9609375 0.685367702805 0.679358717435
27 0.953964233398 0.875250166778 0.813038359051
28 0.980194091797 0.5842994369 0.731040198923
29 0.989680039788 0.356158992523 0.492182188987
30 0.980484008789 0.359129994942 0.526120785476
31 0.96662902832 0.678074109039 0.724316147737
32 0.88740234375 0.936708860759 0.074267362505
33 0.925079345703 0.933016595289 0.85028662032
34 0.766613769531 0.86209439528 0.328474588177
35 0.973449707031 0.763193302437 0.854393305439
36 0.963047839506 0.927003019554 0.929605620967
37 0.642309570313 0.589333333333 0.0148601398601
38 0.919174194336 0.922604422604 0.718469306404
39 0.903967285156 0.979591836735 0.046539813356
40 0.786584472656 0.873429200674 0.435431265541
41 0.972396850586 0.822108626198 0.876729131175
42 0.799718169761 0.791876689313 0.642161073428
43 0.967662037037 0.978982914168 0.948819714973
44 0.978118896484 0.749801587302 0.840524911032
45 0.987792882405 0.960583574393 0.945317721613
46 0.991485595703 0.760485093482 0.843609865471
47 0.912350795756 0.623476064002 0.752109902476
48 0.933013916016 0.677920324105 0.578452083733
49 0.888146972656 0.954751131222 0.0440271257173
50 0.99267578125 0.722747321991 0.826964671954
51 0.976058959961 0.584280012281 0.708093023256
52 0.4291015625 0.387334315169 0.0219991635299
53 0.962051391602 0.874840829739 0.774036452181
54 0.956913580247 0.931550646708 0.919866827392
55 0.913537597656 0.803687265278 0.399287592231
56 0.91962035809 0.817164754333 0.836117198371
```

```
57 0.974845679012 0.940385260256 0.939335293461
58 0.867932128906 0.954150197628 0.400974475389
59 0.986190795898 0.898560209424 0.752122706108
60 0.993850708008 0.8059785674 0.876418276602
61 0.960734416446 0.909725721054 0.935419516385
62 0.996688842773 0.848167539267 0.8818726184
63 0.792561892131 0.353397661714 0.155690508322
64 0.883928175571 0.788875799394 0.641639860379
65 0.972061157227 0.26646452707 0.414501558878
66 0.968310185185 0.948757918477 0.933022391102
67 0.980072021484 0.317435082141 0.478434504792
68 0.97477722168 0.480572597137 0.630449362844
69 0.966395378113 0.429607314254 0.581970033099
70 0.978395778073 0.953760527819 0.929762942069
71 0.983703613281 0.867280334728 0.906578026592
72 0.944737654321 0.886381187256 0.90856165258
73 0.972946166992 0.546119235096 0.68658299452
74 0.931533813477 0.639653035936 0.534977717898
75 0.96844882847 0.933527202784 0.934559729505
76 0.926232910156 1.0 0.0360504067634
77 0.991027832031 0.927508705286 0.908808933002
78 0.997512817383 0.894383906119 0.92903787549
79 0.981369018555 0.520582120582 0.672214765101
80 0.844799804688 0.379381443299 0.0281302553126
81 0.978500366211 0.317371937639 0.447234209494
82 0.990463256836 0.761396303901 0.855758135241
83 0.925146484375 0.806652806653 0.336076223473
84 0.995483398438 0.931589537223 0.903708523097
85 0.956350308642 0.941732324013 0.928217036558
86 0.970358090186 0.938764038871 0.941828435094
87 0.965957641602 0.759869550292 0.798737032025
88 0.97151184082 0.345694531741 0.370744860128
89 0.921158098397 0.370774640517 0.539537504473
90 0.967016601563 0.907372972973 0.795202182893
91 0.964874267578 0.911602209945 0.843507817811
92 0.966042219275 0.834677995145 0.883852798306
93 0.994995117188 0.815789473684 0.841085271318
94 0.933013916016 0.583192138258 0.439479060266
95 0.794274902344 0.847682119205 0.043584359571
96 0.898010253906 0.504178272981 0.0415280486406
97 0.991729736328 0.711614730878 0.822527832351
98 0.994094848633 0.701417848207 0.812953117448
99 0.953186035156 0.780674065777 0.833278991414
100 0.95198059082 0.747853124075 0.762472639444
101 0.839706420898 0.942031899209 0.727983634998
102 0.9421875 0.987012987013 0.0603174603175
103 0.986938476563 0.874739039666 0.915028786976
104 0.990646362305 0.819285714286 0.882138050375
105 0.968834876543 0.892284216292 0.918648915386
106 0.96111882716 0.93714955653 0.935873452195
107 0.962554012346 0.943873334876 0.92167906654
108 0.971267700195 0.732733361239 0.788069780529
109 0.944732666016 0.909416208336 0.883424525266
110 0.966537475586 0.802009175763 0.64904138527
111 0.977265357971 0.389194571084 0.554802323193
112 0.980694444444 0.972409262605 0.949829556848
113 0.94465637207 0.574598839194 0.481338481338
```

```
114 0.994725353669 0.867526581837 0.912499427052
115 0.995574951172 0.849153789551 0.888375673595
116 0.993087768555 0.771653543307 0.866489832007
117 0.929733276367 0.945250780819 0.690835850957
118 0.862622070313 0.902243589744 0.411893812709
119 0.984819849691 0.727178296468 0.812465865647
120 0.953441358025 0.896539114536 0.919204092017
121 0.974452917772 0.929318782697 0.934615656601
122 0.99055480957 0.689483509645 0.781657848325
123 0.988444960212 0.719721444468 0.827418289865
124 0.663500976562 0.734693877551 0.00260510890803
125 0.976043701172 0.475226195605 0.584216101695
126 0.96617484527 0.902068407673 0.836104640266
127 0.984069824219 0.905037984806 0.896613190731
128 0.750891113281 0.520089285714 0.0223254922627
129 0.849945068359 0.932180218735 0.761310679612
130 0.935424804688 0.428972712681 0.335635792779
131 0.994750976563 0.83689107827 0.898883009994
132 0.93212890625 0.602713704206 0.499662542182
133 0.988052368164 0.645651173493 0.781833379772
134 0.941070556641 0.916159567275 0.778198943258
135 0.984008789063 0.888324873096 0.900171461231
136 0.965270996094 0.662494412159 0.722574353974
137 0.967060185185 0.895151020408 0.927762830601
138 0.840905761719 0.679875073199 0.348317415871
139 0.75068359375 0.865168539326 0.00748372047818
140 0.974533081055 0.612401185771 0.748151501434
141 0.95700617284 0.936444018923 0.922699148192
142 0.972850353669 0.925508583529 0.88197564081
143 0.99089050293 0.776579352851 0.871030460143
144 0.949530282935 0.86050656507 0.852118719539
145 0.857470703125 0.877819548872 0.444423296536
146 0.96696472168 0.717238346526 0.750834388307
147 0.943603515625 0.543505154639 0.416298168035
148 0.931976318359 0.855141059974 0.860338345865
149 0.978895187378 0.335533753966 0.453984702689
150 0.879989624023 0.965523514964 0.729660055683
151 0.987457275391 0.680433660299 0.762290341238
152 0.891448974609 0.90783055199 0.748888104483
153 0.957931518555 0.66133038918 0.634786064379
154 0.975479125977 0.555818673884 0.671571632945
155 0.842102050781 0.714932126697 0.06828495282
156 0.973094135802 0.977371847544 0.946632179862
157 0.945098876953 0.894921410232 0.888191423244
158 0.977401733398 0.681268151016 0.791731120799
159 0.963500976563 0.386363636364 0.379346133887
160 0.945788574219 0.814207650273 0.062882464655
161 0.902481079102 0.922583643123 0.756486949895
162 0.994110107422 0.725618631732 0.83781512605
163 0.973037719727 0.708073283672 0.799227360527
164 0.958969116211 0.697573656846 0.705444188849
165 0.758374023438 0.822441430333 0.0631389625142
166 0.964904785156 0.948319755601 0.866279069767
167 0.968125 0.957195760439 0.947767704738
168 0.993728637695 0.469086021505 0.629395852119
169 0.972670755968 0.940021564748 0.941320946126
170 0.971391467728 0.938395871169 0.946426723513
```

```
171 0.878491210938 0.44262295082 0.0264084507042
172 0.965881347656 0.838874240514 0.723609394314
173 0.989608764648 0.772246177969 0.852629301017
174 0.984405517578 0.884702678166 0.919628814092
175 0.873718261719 0.797101449275 0.020823473734
176 0.846838378906 0.798939359004 0.355804281974
177 0.975440979004 0.489325701872 0.645299027575
178 0.989868164063 0.741584671155 0.81179138322
179 0.981936728395 0.975881044374 0.966082786398
180 0.975613395225 0.879747005065 0.881495206638
181 0.869848632813 0.945488721805 0.0862187178608
182 0.951389809903 0.939849743737 0.870341300198
183 0.941440097259 0.910952233388 0.795234966765
184 0.96809387207 0.901576480614 0.910068384156
185 0.960861206055 0.88682303585 0.878820805972
186 0.699560546875 0.913902921151 0.40343222804
187 0.965743810787 0.919096157959 0.925816449864
188 0.983651083112 0.601102551417 0.741920006979
189 0.956682098765 0.889522292994 0.908679810007
190 0.648742675781 0.8125 0.0203588329418
191 0.993896484375 0.759371221282 0.862637362637
192 0.990600585938 0.832642916321 0.867126833477
193 0.973804012346 0.958326108317 0.951301728466
194 0.984933134394 0.938778706659 0.946197941847
195 0.944900512695 0.795079485238 0.853312751351
196 0.878491210938 0.93470790378 0.0985328744793
197 0.991073608398 0.80032800328 0.869681443529
198 0.963790893555 0.320610687023 0.315151515152
199 0.972685185185 0.93765160363 0.946494966899
200 0.897338867188 0.916233124794 0.805290270302
201 0.917956542969 0.783536585366 0.0710435383552
202 0.953271484375 0.972972972973 0.053412462908
203 0.972126436782 0.907377562049 0.904650283554
204 0.647827148438 0.766666666667 0.0015919158361
205 0.9739693855 0.922410033436 0.936777328153
206 0.959823608398 0.550361881785 0.580932675473
207 0.953536987305 0.869938160782 0.89575844716
208 0.8259765625 0.642105263158 0.0330982094411
209 0.922353006189 0.824770843625 0.810523618809
210 0.97549533844 0.406259861721 0.524316418905
211 0.93558244916 0.910213767087 0.793812791849
212 0.914221191406 0.920103092784 0.0922361452009
213 0.841735839844 0.90280590107 0.324985682303
214 0.95708443855 0.932468642633 0.87957636186
215 0.994140625 0.693004115226 0.814313346228
216 0.963765432099 0.889743867635 0.912962894317
217 0.97074845679 0.960425412818 0.947654750563
218 0.965864197531 0.935012580055 0.936646140627
219 0.992538452148 0.748917748918 0.849861836045
220 0.560314941406 0.862068965517 0.00689293887342
221 0.607962545762 0.156714847334 0.270847124723
222 0.971979995579 0.942882218295 0.940044814151
223 0.992324829102 0.710577547047 0.813219457854
224 0.972257614136 0.401136535511 0.560706735125
225 0.885911251105 0.891357139553 0.788742050519
226 0.968703703704 0.969690630593 0.953642535488
227 0.963012695313 0.680521777581 0.717482517483
```

```
228 0.990905761719 0.920649509804 0.909778988798
229 0.889404296875 0.945 0.0400508582327
230 0.981781005859 0.849732400165 0.873650793651
231 0.92399597168 0.529746281715 0.327164662974
232 0.960262345679 0.940076174355 0.928264987742
233 0.970123291016 0.777331085462 0.859601319375
234 0.993942260742 0.894618834081 0.88938422959
235 0.975046296296 0.976503731231 0.952719298246
236 0.858532714844 0.453879941435 0.272459036977
237 0.91145324707 0.943808255659 0.785606088595
238 0.956604938272 0.916769149684 0.881275068609
239 0.81376953125 0.947368421053 0.00702941942203
240 0.97819519043 0.173829377806 0.274987316083
241 0.871871948242 0.927521501173 0.73859851197
242 0.988784790039 0.939214719895 0.939600624538
243 0.919448852539 0.917588068912 0.730015854345
244 0.875444849691 0.881651200289 0.776239569942
245 0.95524691358 0.934749731681 0.929647509764
246 0.975997924805 0.160844250364 0.21935483871
247 0.990707397461 0.803787878788 0.874510611993
248 0.958221435547 0.842815559672 0.87467960454
249 0.948418209877 0.924957517952 0.909882584489
250 0.969722590628 0.903729743956 0.897688228264
251 0.949691358025 0.872569037657 0.888843426077
252 0.971134259259 0.958955324971 0.949807467833
253 0.722998046875 0.835526315789 0.0110694674453
254 0.968383789063 0.867848509267 0.806137724551
255 0.918286132813 0.892511491135 0.700156774916
256 0.981597900391 0.860449513586 0.894819466248
257 0.974416732788 0.442459102441 0.601325644989
258 0.770678710937 0.718369123622 0.3451617401
259 0.961166381836 0.373692900385 0.347937483987
260 0.993950843811 0.795638686976 0.84420209761
261 0.876782226563 0.670068027211 0.0375667429443
262 0.970811229001 0.932533539732 0.942605672063
263 0.986289787798 0.650475263828 0.777926960258
264 0.990158081055 0.81976744186 0.881412024269
265 0.980422973633 0.572493477451 0.705396096441
266 0.968017578125 0.720462682457 0.833121019108
267 0.753198242187 0.917927261023 0.360715866692
268 0.960083007813 0.922150691464 0.855357735265
269 0.949127197266 0.969795778274 0.772205520634
270 0.972106933594 0.807814149947 0.87005970998
271 0.976649535809 0.925386949924 0.935200165623
272 0.979373894783 0.933120696638 0.936764618682
273 0.969131469727 0.788209606987 0.833237160992
274 0.990188598633 0.691520467836 0.815282964665
275 0.962638888889 0.921039453918 0.93521194605
276 0.902389526367 0.608993448481 0.390006674931
277 0.950775146484 0.790855649902 0.789617842702
278 0.783325195313 0.529510961214 0.0341712917619
279 0.926597222222 0.900508865208 0.892434333269
280 0.977005004883 0.72983776158 0.804666234608
281 0.960810185185 0.887095308762 0.920218029877
282 0.964038085937 0.8 0.0563741191544
283 0.749267578125 0.10150313384 0.182162054549
284 0.881195068359 0.95113251293 0.73260526135
```

```
285 0.975006103516 0.486864551348 0.63535173642
286 0.636804199219 0.706896551724 0.00820027334244
287 0.960983276367 0.68278346606 0.737985449329
288 0.855871582031 0.780701754386 0.0568735522007
289 0.947648651636 0.844123717169 0.863076956431
290 0.990450928382 0.938998082454 0.947744042579
291 0.95482979664 0.91644107235 0.925279265773
292 0.97824845679 0.989912506433 0.953421126551
293 0.982559204102 0.894418524279 0.900565463245
294 0.960077160494 0.917247625844 0.928150863745
295 0.753186035156 0.594594594595 0.00217144549178
296 0.981628417969 0.496418979409 0.648158971362
297 0.770153808594 0.71186440678 0.0302827419272
298 0.986312866211 0.867013372957 0.886441321686
299 0.972281167109 0.870623313929 0.859397337071
300 0.97575378418 0.356725146199 0.434318262727
301 0.962507716049 0.965965612139 0.940290254003
302 0.949693302387 0.916028025396 0.926211270745
303 0.975969827586 0.923700789302 0.946238154405
304 0.963587962963 0.930152358877 0.933093249777
305 0.752453613281 0.666666666667 0.00294999754167
306 0.717443847656 0.918714555766 0.0403001782827
307 0.81044921875 0.6970703125 0.314921027089
308 0.942367553711 0.851455629604 0.50698342253
309 0.993530273438 0.728169014085 0.829855537721
310 0.971389770508 0.695691405369 0.766878030586
311 0.966320800781 0.220895522388 0.0509115927073
312 0.973114013672 0.785922897196 0.859332588217
313 0.974716186523 0.819538500125 0.88748557072
314 0.986886024475 0.542312804608 0.587490625469
315 0.974395751953 0.622340425532 0.715110356537
316 0.952939814815 0.905810613315 0.921802679659
317 0.976182581786 0.919804034073 0.935508005387
318 0.857885742188 0.833602258975 0.415209965843
319 0.956882716049 0.97581420628 0.943527033855
320 0.767419433594 0.441253263708 0.0174307668506
321 0.922439575195 0.929961089494 0.744585699211
322 0.972137451172 0.489938757655 0.550909985243
323 0.959783950617 0.938222561752 0.933941698352
324 0.95824845679 0.919846998936 0.922008100434
325 0.982968611848 0.840135524574 0.874994930034
326 0.775842285156 0.791516400481 0.334215583191
327 0.965072631836 0.794779938588 0.871541612885
328 0.968444824219 0.800054817048 0.849534342258
329 0.993408203125 0.639966273187 0.778461538462
330 0.990295410156 0.751114714228 0.853523721787
331 0.928848266602 0.9070079957 0.852721013234
332 0.989776611328 0.789390962672 0.857081911263
333 0.967986111111 0.959956783037 0.953613952708
334 0.951446533203 0.870004974299 0.868349193215
335 0.947037037037 0.896034099333 0.907932505298
336 0.96658377542 0.794977690918 0.877006000203
337 0.780212402344 0.89874884152 0.301129526841
338 0.982109069824 0.470590328435 0.58504755585
339 0.981063842773 0.520219039596 0.665588790084
340 0.769995117188 0.8682766191 0.401803289098
341 0.974469496021 0.853761504039 0.867830067229
```

```
342 0.92639465332 0.871062441752 0.60785302008
343 0.988647460938 0.726488352028 0.81906614786
344 0.895654296875 0.989690721649 0.06313020605
345 0.994552612305 0.834872351983 0.895948703002
346 0.993515014648 0.807965084561 0.874520224387
347 0.981304012346 0.959987834139 0.959096510627
348 0.97607421875 0.700968013468 0.809477521264
349 0.672583007813 0.774647887324 0.012153800825
350 0.990006078691 0.338877718881 0.506211604096
351 0.968209876543 0.935853039148 0.932316993035
352 0.976760057471 0.925776883523 0.946592417152
353 0.868994140625 0.711740041929 0.275158719438
354 0.977355957031 0.576154992548 0.72261682243
355 0.967727661133 0.731399747793 0.76694214876
356 0.975308641975 0.948484848485 0.94198484354
357 0.966735839844 0.719923170945 0.790907347017
358 0.977264404297 0.443216080402 0.542102028273
359 0.979187011719 0.476001586672 0.637619553666
360 0.995742797852 0.811178247734 0.885043263288
361 0.975112915039 0.442361111111 0.609715242881
362 0.987930297852 0.742529676627 0.821000226296
363 0.951848474801 0.899850226465 0.851255964015
364 0.957125884173 0.876488796868 0.846410436607
365 0.984228134155 0.618700552726 0.73414994856
366 0.961803713528 0.915171006674 0.88449197861
367 0.990203857422 0.898479427549 0.926019820235
368 0.980438232422 0.64020006252 0.761621420602
369 0.990051269531 0.758620689655 0.0974529346622
370 0.984466552734 0.688530104272 0.800860719875
371 0.980575561523 0.376051459673 0.544217687075
372 0.904014699381 0.898391382166 0.82308965447
373 0.855151367187 0.681637519873 0.224241631799
374 0.975021362305 0.104852686308 0.128791910591
375 0.962890625 0.790017567428 0.862769439115
376 0.994248390198 0.80897171958 0.831982170219
377 0.961196899414 0.857840590253 0.866530205217
378 0.965118408203 0.952353616533 0.85306594678
379 0.975814819336 0.556490384615 0.700321421819
380 0.976049804688 0.724293785311 0.395191122072
381 0.980178833008 0.178217821782 0.279534109817
382 0.951461791992 0.802157140608 0.864977291056
383 0.781164550781 0.822158749248 0.3789364282
384 0.862854003906 0.754491017964 0.100840336134
385 0.993850708008 0.700696055684 0.818058690745
386 0.936645507813 0.884174880238 0.755735968938
387 0.845642089844 0.620448179272 0.0654792698248
388 0.972853116711 0.937186040363 0.945408477938
389 0.958404541016 0.828771849126 0.840956826138
390 0.942600574713 0.808936467303 0.870011388239
391 0.955455246914 0.914855875831 0.914680105819
392 0.975662231445 0.466822429907 0.556081269134
393 0.97883605957 0.319396051103 0.442299959791
394 0.976856763926 0.947921371016 0.950111974079
395 0.920257568359 0.907395974812 0.737703272435
396 0.992202758789 0.777709736681 0.832513929859
397 0.989517211914 0.708352561144 0.817141336172
398 0.878297495012 0.830493069757 0.666362807657
```

```
399 0.95002746582 0.615858843537 0.638879700077
400 0.974578857422 0.37156448203 0.457682291667
401 0.980270385742 0.130153597413 0.199380804954
402 0.955795288086 0.785211267606 0.850662405279
403 0.993621826172 0.772861356932 0.862409479921
404 0.965637207031 0.643843336151 0.669891527411
405 0.890612792969 0.844609404682 0.865774928476
406 0.979385375977 0.756209400076 0.854213877199
407 0.965007716049 0.939006892671 0.936385697653
408 0.792272949219 0.818431911967 0.0653594771242
409 0.98469543457 0.897172236504 0.900466408653
410 0.985820070734 0.90490625 0.918599118104
411 0.830810546875 0.989949748744 0.0537957400328
412 0.960974121094 0.984 0.0714493174557
413 0.957461419753 0.903997727757 0.902315856619
414 0.94091796875 0.70079787234 0.655822222222
415 0.974655151367 0.446343779677 0.530923467947
416 0.757319273653 0.162502702898 0.279326018445
417 0.954212962963 0.98658156993 0.927073860145
418 0.990875244141 0.621827411168 0.766223612197
419 0.953460693359 0.877898444379 0.796856267484
420 0.913513183594 0.964830412019 0.749447440545
421 0.563427734375 0.660377358491 0.0019534520288
422 0.988372802734 0.775072224515 0.831341301461
423 0.975402832031 0.3870789619 0.465162574652
424 0.977641467728 0.926112288136 0.912062595088
425 0.87060546875 0.860032804811 0.372484016102
426 0.992431640625 0.697836706211 0.801282051282
427 0.949990844727 0.870593069443 0.795031832793
428 0.704418945313 0.869932432432 0.0408017746791
429 0.993057250977 0.855797819623 0.883601944231
430 0.963146972656 0.666077217801 0.599549011805
431 0.981536865234 0.55335661622 0.702117183653
432 0.971562776304 0.940893852111 0.939352512051
433 0.960779320988 0.930239601561 0.915801156223
434 0.974300950486 0.955430821991 0.938786255372
435 0.589453125 0.82 0.0024322240019
436 0.700109863281 0.846153846154 0.0044575920898
437 0.974319458008 0.522310756972 0.609059233449
438 0.968893678161 0.955237952388 0.940615472259
439 0.792700195313 0.532110091743 0.0134774021146
440 0.962276193634 0.853292769122 0.89053166668
441 0.945205688477 0.859563846558 0.879346840036
442 0.976501464844 0.942809083263 0.879215686275
443 0.9931640625 0.715249662618 0.82554517134
444 0.831225585937 0.680888114933 0.404308487721
445 0.991790771484 0.648838845883 0.774139378673
446 0.963168656057 0.899867294004 0.924870933562
447 0.970260620117 0.886743371686 0.900940279543
448 0.996185302734 0.911869587366 0.934725848564
449 0.880554199219 0.739583333333 0.0282053828583
450 0.970038580247 0.971872227152 0.944210571687
451 0.934182098765 0.942658175417 0.895152170707
452 0.865197753906 0.292307692308 0.00342929338507
453 0.949017333984 0.877311503206 0.798567570175
454 0.833009259259 0.253531813419 0.393373696603
455 0.84462890625 0.513513513514 0.00297665674448
```

```
456 0.962265014648 0.740278796772 0.765481270744
457 0.9966796875 0.135135135135 0.0354609929078
458 0.919372558594 0.772087451333 0.4383980954
459 0.976776123047 0.787641705253 0.866350544433
460 0.982070623342 0.765653623937 0.833994218322
461 0.994201660156 0.901371894698 0.92750858451
462 0.963333333333 0.91401025641 0.903630095315
463 0.814575195313 0.70985915493 0.0321141837645
464 0.943788580247 0.903557235631 0.911007683757
465 0.988815307617 0.871180842279 0.852008883505
466 0.938981481481 0.740318017754 0.793330545683
467 0.971181476569 0.90449168279 0.930345002605
468 0.755456542969 0.875130981488 0.333455331892
469 0.934449602122 0.913120084803 0.843107689866
470 0.992279052734 0.752234359484 0.856900452489
471 0.987533569336 0.827246827247 0.886606523248
472 0.994232177734 0.668711656442 0.801470588235
473 0.97787037037 0.986455498823 0.95744554573
474 0.992263793945 0.711622125544 0.818734358241
475 0.976608276367 0.585389298302 0.704453441296
476 0.959645061728 0.925379971974 0.929236347892
477 0.970565795898 0.418832761157 0.469617816882
478 0.945932802829 0.837917847205 0.870810996382
479 0.976104736328 0.567723342939 0.668079694786
480 0.978834876543 0.982708350207 0.946506230864
481 0.993637084961 0.875659050967 0.905291846468
482 0.951242283951 0.879327398615 0.887367876941
483 0.97858642794 0.924342159531 0.935313167735
484 0.99528503418 0.812457221081 0.884830413716
485 0.990203857422 0.907992368031 0.930278019114
486 0.972030639648 0.714285714286 0.822263162998
487 0.900646972656 0.341772151899 0.00659099231051
488 0.993774414063 0.708782742681 0.818505338078
489 0.981994628906 0.870024875622 0.904623343033
490 0.96369934082 0.730390951561 0.833181403829
491 0.811108398438 0.869141239383 0.263142857143
492 0.979313107869 0.941899194273 0.947932096834
493 0.98550154321 0.968732814832 0.963302931469
494 0.998903072502 0.961676646707 0.952889521775
495 0.944212962963 0.949732065687 0.88370970855
496 0.972897325376 0.931175265521 0.928914623629
497 0.977981567383 0.203056768559 0.278860569715
498 0.960067749023 0.493686354379 0.480856972823
499 0.793774414063 0.528535980149 0.0245958429561
500 0.995642089844 0.190082644628 0.114143920596
501 0.939666748047 0.570523507836 0.463937093275
502 0.952590942383 0.76380484115 0.722217255253
503 0.90837097168 0.950220750552 0.741420143823
504 0.969039916992 0.302684563758 0.307744796998
505 0.984237670898 0.862114248194 0.910461991852
506 0.76748046875 0.561403508772 0.0165220983065
507 0.992721557617 0.84632034632 0.867683772538
508 0.955612182617 0.809523809524 0.795587098588
509 0.970840454102 0.4186866461 0.498556809236
510 0.796362304688 0.802783109405 0.375673652695
511 0.98721540672 0.527100998175 0.679770226313
512 0.96612654321 0.991833328366 0.938120207488
```

```
513 0.710485839844 0.430769230769 0.00235561351113
514 0.953674316406 0.699757869249 0.695547533093
515 0.89932581786 0.890003214401 0.820125984874
516 0.990341186523 0.845546786922 0.825763831544
517 0.974258422852 0.82382892057 0.870360408822
518 0.940093994141 0.28285083157 0.435199338237
519 0.881393432617 0.927310596199 0.820861468968
520 0.967028625111 0.890575348611 0.922945791496
521 0.894607543945 0.584642233857 0.326606220142
522 0.979217529297 0.917647058824 0.364145658263
523 0.968170166016 0.497063903282 0.579774375504
524 0.996994018555 0.833898305085 0.882247459653
525 0.939590454102 0.714589371981 0.651343020696
526 0.918731689453 0.948353164339 0.805079783341
527 0.971572875977 0.648418156809 0.716740155086
528 0.97388117284 0.96717967736 0.948694241933
529 0.962735535358 0.895477699576 0.631329330811
530 0.952183641975 0.909995577178 0.913974762969
531 0.961041666667 0.912897770945 0.921639532538
532 0.996810913086 0.909254807692 0.935394126739
533 0.903479197305 0.283200946589 0.440977701932
534 0.689599609375 0.877218934911 0.0445630119486
535 0.807629394531 0.845010615711 0.0480821504077
536 0.974021883289 0.929026774331 0.948980367046
537 0.978279320988 0.970341035357 0.960313544149
538 0.993621826172 0.79854368932 0.862950819672
539 0.9716796875 0.736435501258 0.815396856972
540 0.974990844727 0.740240863787 0.813091572585
541 0.981842041016 0.841610500177 0.888576779026
542 0.965601851852 0.937087633805 0.938861155302
543 0.964359567901 0.986428909579 0.941994951715
544 0.980529785156 0.493899683687 0.631426920855
545 0.973007202148 0.423702149974 0.477400295421
546 0.976608276367 0.533198517021 0.673621460507
547 0.936413574219 0.878502577897 0.818647077255
548 0.970611572266 0.555474095797 0.702410383189
549 0.994216918945 0.828220858896 0.886831890116
550 0.992645263672 0.742441209406 0.846202935546
551 0.990676879883 0.931308093403 0.930780559647
552 0.991882324219 0.661301140174 0.787539936102
553 0.993133544922 0.774959525094 0.864539434076
554 0.96475308642 0.937917778352 0.934699945679
555 0.9752127542 0.914383205136 0.94630619415
556 0.937835693359 0.941780568895 0.862215909091
557 0.965470679012 0.902324985589 0.926430696894
558 0.963194444444 0.923262881613 0.918094714791
559 0.953302469136 0.941174757564 0.914386759089
The average precision of the model is 76.34429243351293%
The average accuracy of the model is 93.71913791606703%
The average f1-score of the model is 67.14543699222496%
```

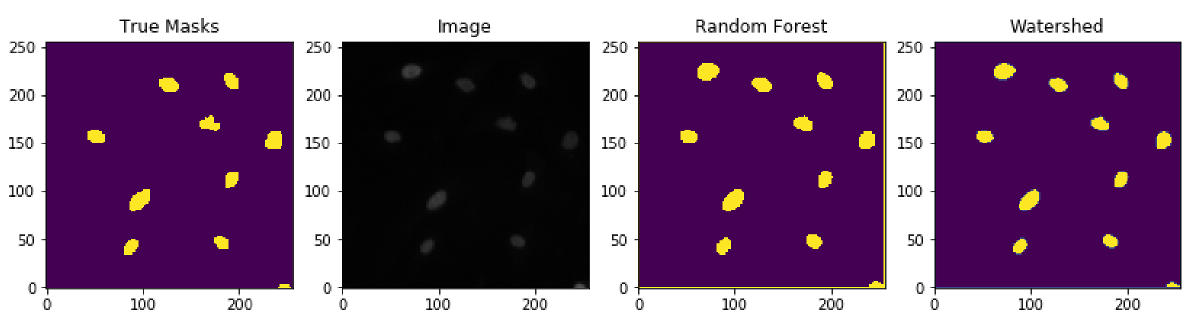
Working Example of the Code

- Demonstrates the methods involved using a smaller set of 30 training images
- Evaluates performance based on a confusion matrix and accuracy (compared to the true masks)

In [53]:
```
precision = np.mean(np.array(acc)[:,1])
accuracy = np.mean(np.array(acc)[:,0])
f1 = np.mean(np.array(acc)[:,2])
print('The average precision of the model is {}%'.format(precision*100))
print('The average accuracy of the model is {}%'.format(accuracy*100))
print('The average f1-score of the model is {}%'.format(f1*100))
```

```
The average precision of the model is 81.38706474131074%
The average accuracy of the model is 94.68844817046038%
The average f1-score of the model is 74.83144803204694%
```

In [42]:
```
for i in range(173,174):
    (img, masks) = load_zipped_img(path+'/stage1_train.zip', i)

    (img_guess, markers, sure_bg, sure_fg, uncertain) = watershed(img)

    intensity = grayscale(img)
    shap = intensity.shape
    intensity_raw = intensity.reshape(-1,1)
    #watershed_raw = img_guess[unknown==255].reshape(-1,1)
    watershed_raw = grayscale(img_guess).reshape(-1,1)
    fil = sobel(grayscale(img)).reshape(-1,1)

    img3 = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    ch1,ch2,ch3 = cv2.split(img3)
    ch1 = ch1.reshape(-1,1)
    ch2 = ch2.reshape(-1,1)
    ch3 = ch3.reshape(-1,1)
    feature = np.concatenate((intensity_raw,watershed_raw, fil, ch1, ch2, ch3
), axis = 1)

    yp = clf.predict(feature)
    y_real = sum(masks).reshape(-1,1)

    labels, nlabels, label_mask = separate_obj(grayscale(yp.reshape(shap[0], s
hap[1])))

    #print(labels, nlabels, label_mask)
    #print(len(masks))

fig, ax = plt.subplots(1,4, figsize = (14,14))

ax[0].imshow(sum(masks), origin='lower')
ax[0].set_title('True Masks')
ax[1].imshow(img, origin='lower')
ax[1].set_title('Image')
ax[2].imshow(grayscale(yp.reshape(shap[0], shap[1])), origin='lower')
ax[2].set_title('Random Forest')
ax[3].imshow(grayscale(img_guess), origin='lower')
ax[3].set_title('Watershed')
plt.show()
```



In [46]:
```
# Compare mask by mask

def mask_by_mask_score(img, masks):
    mask_scores = []
    mask_num = len(masks)
    for i in range(0,mask_num):
        individual_mask = img[masks[i] == 1]
        score = np.sum(individual_mask)/individual_mask.size
        mask_scores.append(score)



    return mask_scores

print(mask_by_mask_score(grayscale(yp.reshape(shap[0], shap[1])),masks))
```

```
[0.89323843416370108, 0.97409326424870468, 0.9543568464730291, 0.919540229885
05746, 1.0, 0.98469387755102045, 0.96385542168674698, 0.95580110497237569, 0.
99521531100478466, 0.9285714285714286, 0.94545454545454544]
```

In [40]:
```
import numpy as np
from scipy.ndimage.morphology import distance_transform_edt as dtx
import scipy.ndimage.filters as filters

def ConvertMask(Mask):
    # convert binary mask to signed distance function
    Phi0 = dtx(1-Mask) - dtx(Mask) + Mask - 1/2
    return Phi0

def Kappa(Phi):
    dPhi = np.gradient(Phi)  # calculate gradient of level set image
    xdPhi = np.gradient(dPhi[1])
    ydPhi = np.gradient(dPhi[0])
    K = (xdPhi[1]*(dPhi[0]**2) - 2*xdPhi[0]*dPhi[0]*dPhi[1] +
         ydPhi[0]*(dPhi[1]**2)) / ((dPhi[0]**2 + dPhi[1]**2 + 1e-10)**(3/2))
    K *= (xdPhi[1]**2 + ydPhi[0]**2)**(1/2)
    return K

def Impulse(X, Epsilon):
    # Smooth dirac delta function.

    # calculate smoothed impulse everywhere
    Xout = (1 + np.cos(np.pi * X / Epsilon)) / (2 * Epsilon)

    # zero out values |x| > Epsilon
    Xout[np.absolute(X) > Epsilon] = 0

    return Xout

def ChanVese(I, Mask, Sigma, dt=1.0, Mu=0.2, Lambda1=1, Lambda2=1, It=100):

    # smoothed gradient of input image
    I = filters.gaussian_filter(I, Sigma, mode='constant', cval=0)
#    dsI = np.gradient(sI)
#    I = 1/(1 + dsI[0]**2 + dsI[1]**2)
```

```
    # generate signed distance map
    Phi = ConvertMask(Mask)

    # evolve level set function
    for i in range(0, It):

        # calculate interior and exterior averages
        C1 = np.sum(I[Phi > 0]) / (np.sum(Phi > 0) + 1e-10)
        C2 = np.sum(I[Phi <= 0]) / (np.sum(Phi <= 0) + 1e-10)
        Force = Lambda2 * (I - C2)**2 - Lambda1 * (I - C1)**2

        # curvature of image
        Curvature = Kappa(Phi)

        # evolve
        Phi += dt * Force / np.max(np.abs(Force)) + Mu*Curvature

    return Phi



print(ChanVese(grayscale(img),sum(masks),3))
```

```
C:\Users\John\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: RuntimeWa
rning: overflow encountered in square

C:\Users\John\Anaconda3\lib\site-packages\ipykernel_launcher.py:15: RuntimeWa
rning: overflow encountered in square
  from ipykernel import kernelapp as app
C:\Users\John\Anaconda3\lib\site-packages\ipykernel_launcher.py:15: RuntimeWa
rning: overflow encountered in power
  from ipykernel import kernelapp as app
C:\Users\John\Anaconda3\lib\site-packages\ipykernel_launcher.py:15: RuntimeWa
rning: invalid value encountered in true_divide
  from ipykernel import kernelapp as app
C:\Users\John\Anaconda3\lib\site-packages\ipykernel_launcher.py:16: RuntimeWa
rning: overflow encountered in square
  app.launch_new_instance()
C:\Users\John\Anaconda3\lib\site-packages\ipykernel_launcher.py:44: RuntimeWa
rning: invalid value encountered in greater
C:\Users\John\Anaconda3\lib\site-packages\ipykernel_launcher.py:45: RuntimeWa
rning: invalid value encountered in less_equal
C:\Users\John\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: RuntimeWa
rning: overflow encountered in multiply

C:\Users\John\Anaconda3\lib\site-packages\ipykernel_launcher.py:14: RuntimeWa
rning: invalid value encountered in subtract

C:\Users\John\Anaconda3\lib\site-packages\ipykernel_launcher.py:15: RuntimeWa
rning: invalid value encountered in add
  from ipykernel import kernelapp as app
C:\Users\John\Anaconda3\lib\site-packages\ipykernel_launcher.py:16: RuntimeWa
rning: overflow encountered in multiply
  app.launch_new_instance()
C:\Users\John\Anaconda3\lib\site-packages\numpy\lib\function_base.py:1768: Ru
ntimeWarning: invalid value encountered in subtract
  out[slice1] = (f[slice4] - f[slice2]) / (2. * dx[i])
C:\Users\John\Anaconda3\lib\site-packages\ipykernel_launcher.py:15: RuntimeWa
rning: overflow encountered in multiply
  from ipykernel import kernelapp as app
```

```
[[ nan  nan  nan ...,  nan  nan  nan]
 [ nan  nan  nan ...,  nan  nan  nan]
 [ nan  nan  nan ...,  nan  nan  nan]
 ...,
 [ nan  nan  nan ...,  nan  nan  nan]
 [ nan  nan  nan ...,  nan  nan  nan]
 [ nan  nan  nan ...,  nan  nan  nan]]
```

```
C:\Users\John\Anaconda3\lib\site-packages\ipykernel_launcher.py:52: RuntimeWa
rning: invalid value encountered in true_divide
```

In [94]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier

# Not rigorous but I picked a few images to train on - it went pretty well

n_samples = [0, 19, 50, 100, 149, 150, 175, 177, 200, 250, 300, 350, 400, 51
0, 559]
#n_samples = range(0,560)
```

```
x_train = np.zeros(8).reshape(1,8) # predicted segmentation using Otsu's thr
esholding

y_train = np.zeros(1) # "correct" segmentation from sum of masks

max_len = 0
conm = []
acc = []
for i in n_samples:
    (img, masks) = load_zipped_img(path+'/stage1_train.zip', i) # Loads imag
e and associated masks

    print(i)
    (img_guess, markers, sure_bg, sure_fg, uncertain) = watershed(img)

    intensity=grayscale(img)
    #img_guess = otsu(intensity)
    img_guess=grayscale(img_guess)
    #img_guess = np.dot(0,intensity)
    laplac = laplace(grayscale(img))[uncertain==255].reshape(-1,1)
    fil = sobel(grayscale(img))[uncertain==255].reshape(-1,1)
    #fil = sobel(grayscale(img)).reshape(-1,1)
    intensity_raw = intensity[uncertain==255].reshape(-1,1)
    #intensity_raw = intensity.reshape(-1,1)
    watershed_raw = img_guess[uncertain==255].reshape(-1,1)
    #watershed_raw = img_guess.reshape(-1,1)
    #print(np.ones(len(watershed_raw)).shape)
    img_type = df['cflabel'][i]*np.ones(len(watershed_raw)).reshape(-1,1)
    img3 = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    ch1,ch2,ch3 = cv2.split(img3)
    #ch1 = ch1.reshape(-1,1)
    #ch2 = ch2.reshape(-1,1)
    #ch3 = ch3.reshape(-1,1)

    ch1 = ch1[uncertain==255].reshape(-1,1)
    ch2 = ch2[uncertain==255].reshape(-1,1)
    ch3 = ch3[uncertain==255].reshape(-1,1)

    feature = np.concatenate((intensity_raw,watershed_raw, fil, laplac, ch1,
 ch2, ch3, img_type), axis = 1) # Features include grayscale intensity, wate
rshed, sobel, rgb channels


    y_raw = sum(masks)
    y_raw = y_raw[uncertain==255].reshape(-1,1)


    x_train = np.append(x_train,feature).reshape(-1,8)
    y_train = np.append(y_train,y_raw).reshape(-1,1)


Xtrain, Xtest, ytrain, ytest = train_test_split(x_train,y_train)
# Kept a split training set around but not needed

#clf = svm.SVC(C = 1.0)
clf = RandomForestClassifier(max_depth=6, random_state=0)
%time clf.fit(Xtrain, ytrain[:,0]) #599 ms for a training set of this size!
```

```
    #y_pred = clf.predict(Xtest)
    #cm = confusion_matrix(ytest, y_pred)
    #print(cm)
    #conm.append(cm)
    #accuracy = sklearn.metrics.accuracy_score(ytest, y_pred)
    #print(accuracy)
    #acc.append(accuracy)

    #(img_guess, markers, sure_bg, sure_fg, unknown, reduced_area) = watersh
ed(img)

    #fig, ax = plt.subplots(1,4, figsize = (10,10))

    #ax[0].imshow(sum(masks), origin='lower')
    #ax[1].imshow(grayscale(img), origin='lower')
    #ax[2].imshow(output, origin='lower')
    #ax[3].imshow(img_guess, origin='lower')
    #plt.show()

#A = np.array(acc)
#Avg_acc = np.mean(A)
#print(Avg_acc)
```

```
0
19
50
100
149
150
175
177
200
250
300
350
400
510
559
Wall time: 12.7 s
```

Out[94]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=6, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

In [95]:
```
# Testing the classifier on the remaining images
acc = []
for i in range(0,560):
    (img, masks) = load_zipped_img(path+'/stage1_train.zip', i)

    (img_guess, markers, sure_bg, sure_fg, unknown) = watershed(img)
    intensity = grayscale(img)
    shap = intensity.shape
    intensity_raw = intensity.reshape(-1,1)
    #watershed_raw = img_guess[unknown==255].reshape(-1,1)
    watershed_raw = grayscale(img_guess).reshape(-1,1)
    fil = sobel(grayscale(img)).reshape(-1,1)
    laplac = laplace(grayscale(img)).reshape(-1,1)
    img3 = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    #hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    img_type = df['cflabel'][i]*np.ones(len(watershed_raw)).reshape(-1,1)
    img3 = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    #ch1,ch2,ch3 = cv2.split(hsv)


    ch1,ch2,ch3 = cv2.split(img3)
    ch1 = ch1.reshape(-1,1)
    ch2 = ch2.reshape(-1,1)
    ch3 = ch3.reshape(-1,1)
    #print(ch1.shape)
    #print(intensity_raw.shape)
    feature = np.concatenate((intensity_raw,watershed_raw, fil, laplac, ch1, c
h2, ch3, img_type), axis = 1)

    yp = clf.predict(feature)
    y_real = sum(masks).reshape(-1,1)
    accuracy = sklearn.metrics.accuracy_score(y_real, yp)
    precision = sklearn.metrics.precision_score(y_real, yp)
    f1 = sklearn.metrics.f1_score(y_real, yp)
    print(i, accuracy, precision, f1)
    acc.append([accuracy, precision, f1])

    #output = yp.reshape(shap[0],shap[1])

precision = np.mean(np.array(acc)[:,1])
accuracy = np.mean(np.array(acc)[:,0])
f1 = np.mean(np.array(acc)[:,2])
print('The average precision of the model is {}%'.format(precision*100))
print('The average accuracy of the model is {}%'.format(accuracy*100))
print('The average f1-score of the model is {}%'.format(f1*100))
```

```
0 0.969528198242 0.768273716952 0.83185989728
1 0.974746704102 0.740964840556 0.845572454978
2 0.892297363281 0.947755211117 0.828282828283
3 0.949694824219 0.824115876599 0.86447199658
4 0.980419921875 0.872095220102 0.851974898487
5 0.459716796875 0.0773522433976 0.143534420202
6 0.871292114258 0.689623740315 0.763533402484
7 0.976666666667 0.942311809868 0.953216374269
8 0.440200805664 0.0274388619532 0.053165406354
9 0.969714506173 0.913987645848 0.944333347516
10 0.953032407407 0.855305989583 0.911970150549
11 0.841542895145 0.546974185358 0.684526436579
12 0.894958496094 0.618857142857 0.758828475336
13 0.498123168945 0.131054131054 0.23034983035
14 0.966597222222 0.933867373366 0.934754103302
15 0.459533691406 0.0551354842182 0.103699579938
16 0.967849731445 0.683611924866 0.790160342595
17 0.970231481481 0.945651341485 0.950504195212
18 0.990325927734 0.608860759494 0.752150117279
19 0.431427001953 0.00837110103972 0.016574294009
20 0.992889404297 0.655325443787 0.791778373548
21 0.868520667551 0.815531584522 0.769230023133
22 0.953240740741 0.850072825312 0.900528544696
23 0.968495368958 0.422118094959 0.582675374878
24 0.435562133789 0.0227290747152 0.0444318152463
25 0.982411384583 0.392988929889 0.509664211842
26 0.468139648438 0.0884088542362 0.159650899272
27 0.94352722168 0.748766488773 0.800732245733
28 0.462295532227 0.0484602917342 0.0924099209313
29 0.996084770115 0.652748037116 0.72078817734
30 0.443344116211 0.0190191805881 0.0373136297665
31 0.465560913086 0.0865250098984 0.157668165749
32 0.955004882812 0.936489093337 0.773975962718
33 0.921493530273 0.827044834509 0.861986641273
34 0.92880859375 0.923701698253 0.865504358655
35 0.482131958008 0.13321737351 0.234832600609
36 0.968927469136 0.922659276645 0.941876073496
37 0.868005371094 0.865649877828 0.802477029027
38 0.947235107422 0.913841080078 0.836825217063
39 0.962072753906 0.949818181818 0.770785687938
40 0.922717285156 0.923751315064 0.853479599158
41 0.534957885742 0.181533899447 0.306474001593
42 0.839378868258 0.779763615462 0.749188872014
43 0.971195987654 0.935302104549 0.956791480988
44 0.505966186523 0.107053001189 0.193016126218
45 0.988315097259 0.960608639071 0.947788189686
46 0.9892578125 0.70141101502 0.814051769678
47 0.912201591512 0.620338596736 0.755109590308
48 0.507583618164 0.136539524599 0.23439538801
49 0.9619140625 0.984181141439 0.802681507716
50 0.989318847656 0.630843632456 0.770341207349
51 0.96598815918 0.486700215672 0.64568431092
52 0.473059082031 0.60203133709 0.310530434921
53 0.962713623047 0.878815228093 0.778104681995
54 0.965864197531 0.918872688768 0.938649285813
55 0.9650390625 0.856514503091 0.834201690402
56 0.922891799293 0.819219408366 0.843903994272
```

```
57 0.977145061728 0.920021056326 0.946528504892
58 0.943725585938 0.958935361217 0.81400790769
59 0.988159179688 0.857294429708 0.806387225549
60 0.990707397461 0.710387745333 0.829745596869
61 0.969031830239 0.932865387051 0.948462330783
62 0.992965698242 0.657206870799 0.792435839712
63 0.804686118479 0.478396734798 0.297991935964
64 0.91103968078 0.715336231616 0.793993839836
65 0.980066299438 0.333876719307 0.484715511291
66 0.970223765432 0.902087811773 0.940746541373
67 0.432922363281 0.0158183407965 0.0311277960269
68 0.468643188477 0.0403016407933 0.0773164463051
69 0.978558540344 0.558409710181 0.667258654117
70 0.979337975243 0.955233198678 0.932945965818
71 0.941558837891 0.589839867477 0.736080485116
72 0.942399691358 0.84689222152 0.909470160928
73 0.450805664063 0.0538427108291 0.102040816327
74 0.523788452148 0.141937830503 0.245339136743
75 0.970167440318 0.936159031189 0.938190186683
76 0.965344238281 0.994631673129 0.701440740351
77 0.981674194336 0.760479041916 0.835411813074
78 0.994567871094 0.764124293785 0.85873015873
79 0.46305847168 0.0341906202723 0.0659854014599
80 0.914453125 0.754763424195 0.702622422134
81 0.412033081055 0.0163355182302 0.0320530533296
82 0.98779296875 0.707169811321 0.824098504837
83 0.970373535156 0.890362793671 0.821136413885
84 0.992034912109 0.785983827493 0.848167539267
85 0.963317901235 0.918399353658 0.942061131965
86 0.971916445623 0.942644876918 0.944834623275
87 0.475769042969 0.122577632501 0.215723873442
88 0.437911987305 0.0291079064085 0.0561633657024
89 0.925212772615 0.382684782609 0.551069441292
90 0.967440795898 0.907926409904 0.798367130951
91 0.966305541992 0.916864260414 0.85015539541
92 0.966230106101 0.835416592265 0.884471415608
93 0.99348449707 0.722610722611 0.813292522956
94 0.546401977539 0.128898315211 0.22499152697
95 0.917517089844 0.880379978078 0.780993744531
96 0.946643066406 0.717702304248 0.750442477876
97 0.990417480469 0.675704412547 0.801892744479
98 0.993286132813 0.665130568356 0.797421731123
99 0.899337768555 0.572334840309 0.704104059206
100 0.517288208008 0.158256756022 0.269011253091
101 0.887939453125 0.887163217955 0.836129953588
102 0.984106445313 0.99203734212 0.847326454034
103 0.971054077148 0.719836710009 0.833873368946
104 0.97932434082 0.643444871092 0.775921944766
105 0.968935185185 0.863193715382 0.922148741153
106 0.961211419753 0.891753048084 0.939536450127
107 0.971558641975 0.93773820681 0.942153170119
108 0.472152709961 0.0970019035533 0.175001788653
109 0.898880004883 0.731852439508 0.816665283426
110 0.966516113281 0.800851802193 0.649232736573
111 0.988081932068 0.555848179706 0.696682119366
112 0.973479938272 0.894895002325 0.935717358371
113 0.502807617188 0.0994156975111 0.178457969845
```

```
114 0.994893899204 0.86331366008 0.916129617863
115 0.991973876953 0.710076605775 0.820844686649
116 0.985610961914 0.612278533169 0.759131545338
117 0.941390991211 0.857434068034 0.777989711577
118 0.926428222656 0.926182350073 0.759141589737
119 0.989508731211 0.807819346141 0.863245092743
120 0.952885802469 0.862115959354 0.921940119148
121 0.978638925729 0.949885678952 0.944716575016
122 0.98828125 0.627635960044 0.746534653465
123 0.994382736516 0.84648902296 0.908575797095
124 0.886010742187 0.769769710586 0.847954930311
125 0.450393676758 0.0371967221895 0.0716034744955
126 0.980763704686 0.909729783233 0.913504783203
127 0.96272277832 0.69789646427 0.792948554962
128 0.944665527344 0.863162705667 0.892849544971
129 0.888549804688 0.87796670209 0.844463373083
130 0.500198364258 0.097869377524 0.176658371666
131 0.990295410156 0.712925170068 0.831746031746
132 0.51220703125 0.125598052657 0.219264397011
133 0.986022949219 0.607388316151 0.755341880342
134 0.964141845703 0.895083994641 0.880831643002
135 0.968002319336 0.726951399116 0.824797393266
136 0.525817871094 0.100531728409 0.181305653617
137 0.967584876543 0.902490796325 0.928338706651
138 0.9357421875 0.818193855932 0.824416277518
139 0.903381347656 0.733561185824 0.833041533951
140 0.468688964844 0.0678034263653 0.1268368524
141 0.951435185185 0.871685327048 0.918574866103
142 0.983325044209 0.929850267869 0.930841250014
143 0.985107421875 0.675873544093 0.80627233029
144 0.950801282051 0.863648977327 0.855964148776
145 0.959191894531 0.904830569575 0.891997544664
146 0.500885009766 0.104581314879 0.187652113446
147 0.446350097656 0.0780178818132 0.141613437426
148 0.86962890625 0.674154095084 0.76830458835
149 0.986149787903 0.460158013544 0.528948136616
150 0.922485351563 0.945097771093 0.844771741123
151 0.984481811523 0.612903225806 0.729016786571
152 0.91047668457 0.837360711156 0.820135503847
153 0.47282409668 0.0877244944667 0.158675270912
154 0.444900512695 0.0484082446111 0.0921364577874
155 0.952087402344 0.857417519034 0.850322236205
156 0.979807098765 0.971566425927 0.960735772907
157 0.877990722656 0.692831322473 0.786841544039
158 0.461059570313 0.0766552347043 0.142218768214
159 0.446243286133 0.0432179258589 0.0821467411922
160 0.97783203125 0.911906193626 0.769601623953
161 0.919509887695 0.849609642515 0.824675108851
162 0.99055480957 0.619076923077 0.764728240213
163 0.482513427734 0.098161803997 0.178081527798
164 0.484588623047 0.104195406963 0.185719107083
165 0.909887695313 0.87137525122 0.804357044419
166 0.955902099609 0.795026513074 0.857509121388
167 0.96899691358 0.915156278898 0.951700925592
168 0.991821289063 0.403607666291 0.571884984026
169 0.973358753316 0.937829009212 0.943037077298
170 0.972477343059 0.93616502388 0.948712007703
```

```
171 0.970776367188 0.827715355805 0.888130841121
172 0.966192626953 0.84258301735 0.725859935659
173 0.988433837891 0.739941477688 0.842214820983
174 0.923248291016 0.549718400886 0.703034596765
175 0.893444824219 0.576818675353 0.593603054146
176 0.935290527344 0.868695195983 0.813888986413
177 0.985399246216 0.636751640439 0.741262759413
178 0.989364624023 0.721238938053 0.808041861746
179 0.983788580247 0.953960862576 0.970381335025
180 0.981393678161 0.880758739771 0.91268832819
181 0.937231445313 0.767732713954 0.769726824899
182 0.966622458002 0.939658290294 0.914919990985
183 0.96846540672 0.919547836663 0.900091915788
184 0.921981811523 0.705976967123 0.811042536679
185 0.916351318359 0.684225451967 0.778808908974
186 0.905151367188 0.951682229139 0.86861239812
187 0.969473916888 0.930336110313 0.933644051509
188 0.989099801061 0.696895624542 0.812597976343
189 0.960632716049 0.863501397299 0.92081082759
190 0.879418945312 0.766622989211 0.847735610568
191 0.991302490234 0.68838992333 0.815175097276
192 0.982009887695 0.671941971645 0.77564224548
193 0.976141975309 0.930656050784 0.957227832342
194 0.986077033599 0.942623432027 0.950294445486
195 0.880004882813 0.599921033335 0.730093355299
196 0.952014160156 0.94530077752 0.779045584846
197 0.989044189453 0.7501878287 0.847623089983
198 0.446594238281 0.0394287239722 0.0753148743052
199 0.965478395062 0.883661016949 0.935759002929
200 0.897247314453 0.798770394892 0.833785851804
201 0.971142578125 0.848871141146 0.823818750932
202 0.981628417969 0.959500378501 0.771102661597
203 0.979984526967 0.907034595497 0.933746730322
204 0.867065429688 0.747548749587 0.832908828674
205 0.979263373121 0.944961816584 0.949060279235
206 0.462799072266 0.0713924858412 0.132344242902
207 0.892013549805 0.679429467402 0.791503402763
208 0.948901367187 0.80255648038 0.865764494613
209 0.926514146773 0.817221760366 0.825510753041
210 0.983442306519 0.537633958473 0.596682772719
211 0.962397767462 0.923230107164 0.889376610497
212 0.972692871094 0.929915966387 0.831842441555
213 0.941381835938 0.939678714859 0.829716312057
214 0.97237234748 0.93343570172 0.926056572379
215 0.98991394043 0.563408190225 0.720743557245
216 0.959598765432 0.841432142046 0.908269096006
217 0.977538580247 0.948322539548 0.960857346475
218 0.96450617284 0.891931806405 0.937372362151
219 0.984268188477 0.577586206897 0.731859557867
220 0.851586914062 0.806941598361 0.838247033154
221 0.609576592926 0.157017783626 0.271215974495
222 0.974121352785 0.95233299765 0.944328867438
223 0.989532470703 0.629300776915 0.767772511848
224 0.984539985657 0.560393986522 0.680904670984
225 0.904310344828 0.887758827518 0.830933109421
226 0.958726851852 0.905023190326 0.942339409488
227 0.469573974609 0.093192718408 0.168293616614
```

```
228 0.97998046875 0.743756003842 0.825206501465
229 0.930798339844 0.934079299441 0.575323994307
230 0.961074829102 0.655398203137 0.771395286316
231 0.502517700195 0.124179610588 0.21787213626
232 0.952484567901 0.8747858473 0.919631437446
233 0.930648803711 0.58140655106 0.726452001204
234 0.991485595703 0.780729401171 0.861400894188
235 0.981959876543 0.956570311409 0.966995115616
236 0.823571777344 0.410940447319 0.513416153251
237 0.938659667969 0.903769705278 0.867728349566
238 0.959891975309 0.870079340861 0.897713408635
239 0.964001464844 0.894068877551 0.904831058186
240 0.421752929688 0.00823887489505 0.0163015263213
241 0.907119750977 0.883942028986 0.833611240194
242 0.968307495117 0.758087382395 0.849938588252
243 0.942916870117 0.881336624264 0.832309830113
244 0.895106653404 0.876460583126 0.82104320362
245 0.951226851852 0.880725790717 0.927821866971
246 0.432113647461 0.0153442518683 0.0301758957655
247 0.985397338867 0.704827136334 0.818715665846
248 0.918121337891 0.674560629708 0.787468314322
249 0.958263888889 0.910116355876 0.929805209131
250 0.977561339523 0.910038506313 0.926001184564
251 0.956782407407 0.865915261729 0.907352576296
252 0.961134259259 0.890118047075 0.936549726019
253 0.929760742188 0.943057722309 0.863078241005
254 0.951477050781 0.671013713364 0.758395380641
255 0.91907043457 0.895739462727 0.703164351515
256 0.967163085938 0.727903181942 0.832633379997
257 0.985030174255 0.591875724263 0.706559736788
258 0.931237792969 0.842074074074 0.875982475067
259 0.454238891602 0.0486451716652 0.092138995355
260 0.994714736938 0.881235955056 0.849867259035
261 0.956823730469 0.800379027164 0.83374853114
262 0.971957891247 0.933399059121 0.944948007355
263 0.992092175066 0.770481661511 0.859000886787
264 0.984268188477 0.717086015214 0.826226192483
265 0.442993164063 0.0419734904271 0.0804111245466
266 0.468276977539 0.131126257709 0.231615620383
267 0.915673828125 0.958974104574 0.848409041036
268 0.928176879883 0.714077378399 0.779583235776
269 0.97151184082 0.949494949495 0.887060673885
270 0.496170043945 0.160910136776 0.276010261583
271 0.978815760389 0.932674744277 0.941119550275
272 0.979274425287 0.92882223114 0.936755394046
273 0.947906494141 0.637974380462 0.75739056282
274 0.988067626953 0.646446355817 0.785046728972
275 0.962469135802 0.888875356247 0.937517663078
276 0.683670043945 0.239784205693 0.37679242447
277 0.573196411133 0.203090249327 0.331652768154
278 0.881433105469 0.675264894911 0.762012103986
279 0.934382716049 0.867021709082 0.909350616126
280 0.529907226563 0.0987498532692 0.179284991209
281 0.961072530864 0.867382380802 0.922952396952
282 0.98779296875 0.882218844985 0.822820694543
283 0.494781494141 0.0561290138107 0.106004968139
284 0.907608032227 0.886187341112 0.817867348473
```

```
285 0.441802978516 0.0394034247836 0.0756986204457
286 0.879809570312 0.776755783349 0.850637135922
287 0.557113647461 0.128856624319 0.226927686776
288 0.953735351563 0.848747997639 0.841581675305
289 0.950889699381 0.842202791923 0.873483856272
290 0.991196949602 0.943814154511 0.951784254971
291 0.963229442971 0.94089319019 0.938420248947
292 0.985663580247 0.985014788038 0.969938193703
293 0.955368041992 0.668116467319 0.790847336432
294 0.960941358025 0.887928404353 0.932425577359
295 0.932543945312 0.812234494477 0.873737604533
296 0.979568481445 0.468423253869 0.625873148924
297 0.90458984375 0.730129998354 0.819541928334
298 0.972473144531 0.698360655738 0.800574839708
299 0.980360300619 0.881442350653 0.904184190661
300 0.427215576172 0.0242826183451 0.0472105183004
301 0.970285493827 0.927738296672 0.955270340902
302 0.954282714412 0.927604635509 0.932550650198
303 0.974795534925 0.916268028201 0.943946711893
304 0.951820987654 0.857974244676 0.91733523976
305 0.93857421875 0.866241121136 0.877656212011
306 0.918835449219 0.821937592642 0.866756177231
307 0.923034667969 0.843992900422 0.813929467316
308 0.944134521484 0.889372111987 0.517475881702
309 0.991790771484 0.671153846154 0.795592705167
310 0.475799560547 0.0885659225952 0.161524943864
311 0.939916992188 0.316520924422 0.451648841355
312 0.519622802734 0.149478374266 0.259002965683
313 0.51708984375 0.173662270928 0.295081967213
314 0.990597724915 0.732139797841 0.634215115201
315 0.466766357422 0.0617400702513 0.115649357222
316 0.943163580247 0.846576017562 0.911199517782
317 0.979901635721 0.936304019838 0.945156523313
318 0.956079101562 0.922887796156 0.873184830114
319 0.936728395062 0.888794259414 0.922858379273
320 0.874291992187 0.658628979384 0.778356506392
321 0.947860717773 0.910616141163 0.846805648958
322 0.447891235352 0.0405704975829 0.0774585043727
323 0.952908950617 0.875249031854 0.927606372252
324 0.946990740741 0.84573964202 0.908059206124
325 0.989323607427 0.893753485778 0.920712438954
326 0.939526367188 0.888047293553 0.885878829763
327 0.926559448242 0.628127969591 0.767229288582
328 0.512161254883 0.160548994574 0.274095770043
329 0.991104125977 0.567186340015 0.723827569872
330 0.985092163086 0.659219858156 0.79190628328
331 0.907608032227 0.769165118731 0.835000136251
332 0.986206054688 0.707121364092 0.823918971562
333 0.973294753086 0.960628699373 0.96159863304
334 0.910308837891 0.692094227923 0.792472814574
335 0.946612654321 0.862089870747 0.911316474192
336 0.967357427056 0.795661891327 0.880277265449
337 0.902014160156 0.939481268012 0.781370012257
338 0.987717628479 0.610796645702 0.644098709481
339 0.452194213867 0.033599611378 0.0648589513167
340 0.932385253906 0.925947894329 0.879918485919
341 0.979956896552 0.867417982824 0.898249452954
```

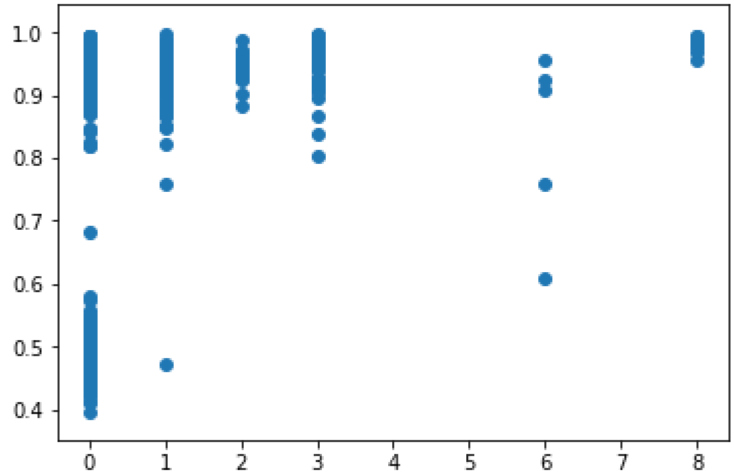
```
342 0.927059936523 0.874327583009 0.612053433752
343 0.984756469727 0.649943417578 0.775253093363
344 0.968151855469 0.976569678407 0.830176397839
345 0.989456176758 0.701659937192 0.819062581828
346 0.990997314453 0.72607879925 0.839934888768
347 0.980910493827 0.935281302063 0.959347990404
348 0.455459594727 0.0869687876689 0.159692952507
349 0.905151367188 0.797276759406 0.868705643799
350 0.995161914235 0.514285714286 0.679245283019
351 0.968402777778 0.887077895476 0.936732329085
352 0.978111184792 0.928786994374 0.949699032332
353 0.850024414062 0.490469289934 0.588904503781
354 0.447616577148 0.0514790726948 0.0978393600319
355 0.494338989258 0.110913116259 0.198670051989
356 0.980038580247 0.924995780591 0.954924816615
357 0.495239257813 0.119690188406 0.212605922117
358 0.434997558594 0.0293467389588 0.0568037087982
359 0.443649291992 0.0325209727089 0.0629642003547
360 0.990417480469 0.638161721931 0.777462792346
361 0.457260131836 0.0348553124491 0.0673361827097
362 0.981735229492 0.632168306753 0.756855575868
363 0.968178050398 0.906281850096 0.906787960213
364 0.958979885057 0.886767868961 0.852425447316
365 0.989488601685 0.75463689542 0.793347832608
366 0.974276083112 0.917867710222 0.925345607339
367 0.973068237305 0.709168808912 0.82422069515
368 0.483657836914 0.0592248148354 0.111674060851
369 0.997253417969 0.921195652174 0.857685009488
370 0.947311401367 0.381150506512 0.549628277031
371 0.437454223633 0.0201536784451 0.0394966521637
372 0.910947170645 0.89433340421 0.839229418572
373 0.757287597656 0.36964580657 0.503235477826
374 0.431594848633 0.0172094744634 0.033771690919
375 0.927963256836 0.636251844565 0.766783579509
376 0.993918418884 0.875475285171 0.801741022851
377 0.906890869141 0.622101555621 0.735362997658
378 0.960861206055 0.829703403743 0.855305466238
379 0.447555541992 0.0498475128825 0.0948071105333
380 0.988061523437 0.804587560653 0.788586251621
381 0.435180664063 0.00974434479989 0.0192878338279
382 0.886993408203 0.599149571913 0.737933474876
383 0.91826171875 0.899243554778 0.839216251261
384 0.958422851562 0.838667323158 0.857166820431
385 0.991668701172 0.628065395095 0.771548117155
386 0.93981628418 0.890869318642 0.769875258177
387 0.907678222656 0.65310642266 0.747689741451
388 0.976605327144 0.949368779641 0.95270440111
389 0.552108764648 0.214770960085 0.34911413176
390 0.943741710875 0.811096771193 0.872649032718
391 0.961412037037 0.911874767904 0.927358559082
392 0.448165893555 0.0321328445115 0.0618433681808
393 0.428359985352 0.0187273776426 0.0367180067368
394 0.981263815208 0.964616406109 0.959295031485
395 0.938674926758 0.854818181818 0.823921139102
396 0.98893737793 0.680187207488 0.782999102065
397 0.984329223633 0.610435464888 0.751872432955
398 0.843626690313 0.594670454161 0.713705913389
```

```
399 0.580810546875 0.12734251524 0.22325265777
400 0.439483642578 0.0274336753682 0.0531986184855
401 0.427154541016 0.00833465629465 0.0165042439484
402 0.921493530273 0.641465256798 0.76752067236
403 0.988632202148 0.646138807429 0.78017114193
404 0.460906982422 0.0732902057876 0.134832010971
405 0.8244140625 0.727249974972 0.8016
406 0.475921630859 0.104692105057 0.189455798367
407 0.970293209877 0.916544770661 0.947992651429
408 0.931225585938 0.869540803594 0.831861048108
409 0.96272277832 0.684203002144 0.796670828132
410 0.986557802829 0.906524642092 0.923079355543
411 0.935607910156 0.988674725035 0.774890112235
412 0.989428710937 0.987669053302 0.851508916324
413 0.960601851852 0.902407671195 0.910505836576
414 0.543228149414 0.156033665585 0.2662810363
415 0.447906494141 0.0328322891177 0.0632249378625
416 0.759251436737 0.163316993464 0.280428022539
417 0.972307098765 0.962270325868 0.95822712618
418 0.989486694336 0.587672052663 0.740294006785
419 0.956146240234 0.804342857143 0.830442477876
420 0.944549560547 0.952296506473 0.855299832763
421 0.907824707031 0.931018688164 0.889813071838
422 0.982620239258 0.663923182442 0.772700059868
423 0.397094726563 0.0262188033342 0.0509223674097
424 0.978824049514 0.933880072899 0.916424941659
425 0.960925292969 0.924515528476 0.872220669834
426 0.989959716797 0.625698324022 0.753926701571
427 0.952142333984 0.880197206245 0.803862220777
428 0.928564453125 0.871218703104 0.882899107536
429 0.988571166992 0.734979423868 0.826660495256
430 0.941638183594 0.458599556657 0.59534490055
431 0.456985473633 0.0390625 0.075109805858
432 0.972540893015 0.938910851716 0.941708507344
433 0.966990740741 0.908775301448 0.932002415996
434 0.97554984527 0.956306666667 0.941895663022
435 0.908117675781 0.961311730029 0.87858306583
436 0.937927246094 0.862161059386 0.90139807256
437 0.45036315918 0.0391554909248 0.0747000950448
438 0.969545755968 0.952622745331 0.94210465495
439 0.928942871094 0.753091452643 0.851022445167
440 0.963367595049 0.853616138821 0.894122344673
441 0.819076538086 0.55606187291 0.691697652045
442 0.971939086914 0.831243329776 0.871407593875
443 0.991333007813 0.660669975186 0.789473684211
444 0.939013671875 0.817117571277 0.853798431464
445 0.990173339844 0.600507292327 0.746256895193
446 0.970186781609 0.923855529967 0.93843573124
447 0.939407348633 0.721364870105 0.824097452935
448 0.994537353516 0.854391733208 0.91041041041
449 0.940466308594 0.672556598564 0.799835830084
450 0.974297839506 0.924047656372 0.955025990684
451 0.958773148148 0.926662253772 0.938259050833
452 0.952197265625 0.756095801472 0.842591848219
453 0.950115966797 0.875690319892 0.804422216373
454 0.820100308642 0.249246889325 0.395028412777
455 0.952197265625 0.789615511009 0.859862582307
```

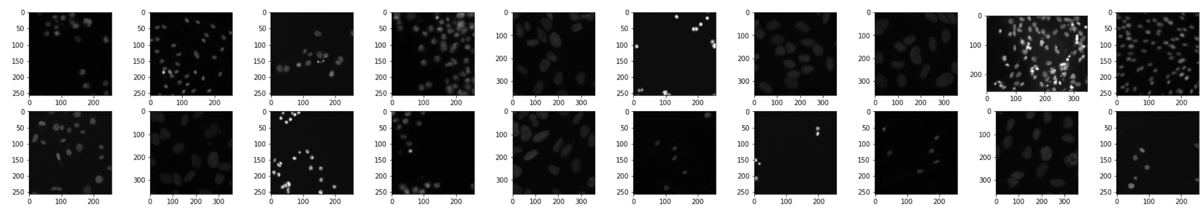
```
456 0.487426757813 0.12219953867 0.215323522541
457 0.985388183594 0.169902912621 0.290456431535
458 0.973388671875 0.862906976744 0.871930442956
459 0.484954833984 0.129991986972 0.229571806811
460 0.988389699381 0.836085022244 0.889508282935
461 0.985229492188 0.732253313697 0.837037037037
462 0.965941358025 0.874429806955 0.915663571401
463 0.942578125 0.885130730789 0.838605640568
464 0.948132716049 0.903120611268 0.918623795458
465 0.987579345703 0.80671668453 0.847279549719
466 0.94412037037 0.738743645606 0.818012765744
467 0.972405503979 0.907034980278 0.933331553194
468 0.918518066406 0.91585791011 0.853473822851
469 0.960734416446 0.917497203927 0.912208953933
470 0.989074707031 0.681042228212 0.808964781217
471 0.978744506836 0.712388417157 0.824492881441
472 0.991653442383 0.582442748092 0.736131210806
473 0.983356481481 0.954154567791 0.969406425076
474 0.990264892578 0.653459821429 0.785906040268
475 0.460174560547 0.0521858364252 0.0988792664289
476 0.962538580247 0.900182897803 0.936712160911
477 0.435821533203 0.0349424199401 0.0670670165523
478 0.947480106101 0.835156064895 0.875725717872
479 0.444625854492 0.047537754432 0.0904615538396
480 0.982106481481 0.930487545242 0.957655436867
481 0.985061645508 0.688940092166 0.810454985479
482 0.956157407407 0.846327329679 0.904855994642
483 0.981940760389 0.942652028447 0.944923823648
484 0.992141723633 0.70564281559 0.82488949337
485 0.976043701172 0.747752332485 0.848834970152
486 0.939758300781 0.52658321973 0.682789651294
487 0.951208496094 0.681669755538 0.794445872975
488 0.992752075195 0.67172812726 0.796399485641
489 0.949554443359 0.64584717608 0.779158316633
490 0.926940917969 0.562940678763 0.71438797423
491 0.884008789063 0.912607876203 0.659450935417
492 0.981719717065 0.951204232215 0.953820166683
493 0.98300154321 0.925160731055 0.958769253804
494 0.999107537577 0.9537999538 0.962358699452
495 0.95287037037 0.949872360176 0.90367449929
496 0.976928603006 0.949766286019 0.938946814267
497 0.436401367188 0.0143907984486 0.0283068504683
498 0.440399169922 0.0603817384576 0.113212109488
499 0.936291503906 0.828706305496 0.849644205007
500 0.971728515625 0.0616379310345 0.109915449654
501 0.500015258789 0.106546926105 0.190038314176
502 0.545455932617 0.153156030428 0.261533503557
503 0.933380126953 0.916362090948 0.831610614008
504 0.412567138672 0.0306983105798 0.0592346415131
505 0.95539855957 0.655295566502 0.784518982676
506 0.924780273438 0.777644075809 0.854505100113
507 0.990600585938 0.755029056772 0.845768652979
508 0.540176391602 0.184344757231 0.307352839773
509 0.444061279297 0.0333796940195 0.0641150783457
510 0.948620605469 0.903549267808 0.896363233447
511 0.991843501326 0.638465447154 0.772992925254
512 0.979768518519 0.979295895018 0.96440691771
```

```
513 0.914538574219 0.819457723049 0.859578394207
514 0.522705078125 0.128094024356 0.224321777513
515 0.921902630416 0.881091562885 0.868819819276
516 0.990493774414 0.792060491493 0.843270440252
517 0.500030517578 0.153880150786 0.26540220608
518 0.957629203796 0.357216745771 0.513868677032
519 0.848068237305 0.763042111879 0.805652606718
520 0.968584217507 0.893999800658 0.926589274415
521 0.53092956543 0.175212129643 0.290652329418
522 0.976104736328 0.556895252449 0.485545335085
523 0.452285766602 0.0473208072373 0.089675635921
524 0.995040893555 0.718602455146 0.824038982133
525 0.518981933594 0.14261943708 0.242903117345
526 0.916000366211 0.819506095655 0.826455660288
527 0.480407714844 0.0754176220111 0.139623022892
528 0.971188271605 0.909439649267 0.946743874262
529 0.95736186544 0.915877226222 0.540881253776
530 0.930331790123 0.819185382566 0.884084577562
531 0.964020061728 0.879298343311 0.931247511906
532 0.99201965332 0.758382642998 0.854681856071
533 0.907652513116 0.291828832805 0.450991334542
534 0.849133300781 0.90353640416 0.710988471342
535 0.917309570313 0.870385105029 0.763576713667
536 0.977251878868 0.941118828407 0.955017565714
537 0.968804012346 0.903157894737 0.946277422698
538 0.992828369141 0.765073947668 0.851265822785
539 0.496520996094 0.116683698166 0.208235350578
540 0.470809936523 0.0969781228687 0.175733808675
541 0.964813232422 0.694594594595 0.808979456594
542 0.963726851852 0.892683652226 0.938738809179
543 0.970362654321 0.943307197521 0.954386214923
544 0.428329467773 0.0288546599365 0.0558929516418
545 0.430221557617 0.0315577731916 0.0607691727243
546 0.464218139648 0.0459207268968 0.0877133726519
547 0.906286621094 0.728541604533 0.760878367855
548 0.502395629883 0.0667545644759 0.12498323003
549 0.989532470703 0.700831024931 0.815690488984
550 0.987808227539 0.627972027972 0.771256799313
551 0.971588134766 0.714261744966 0.820547417116
552 0.987686157227 0.557285873192 0.712913553895
553 0.989608764648 0.684456928839 0.811095700416
554 0.967615740741 0.915389578964 0.941942980454
555 0.982590075155 0.944083939818 0.961649655206
556 0.909561157227 0.769292261778 0.829669224358
557 0.955069444444 0.840795734679 0.909702730783
558 0.966334876543 0.905042098761 0.927335409623
559 0.964992283951 0.925048097478 0.938502202643
The average precision of the model is 64.76129213178552%
The average accuracy of the model is 85.67608686011847%
The average f1-score of the model is 68.71381375457058%
```

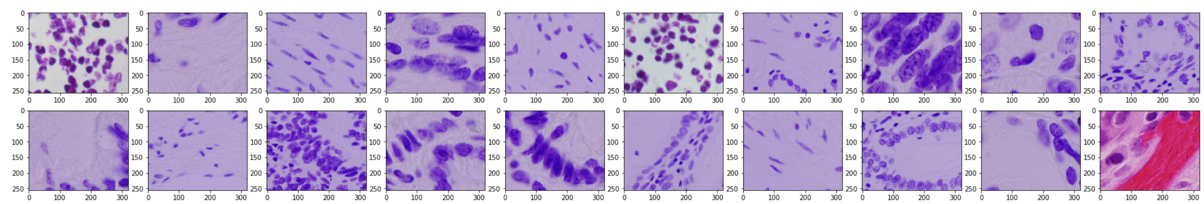
In [114]:
```
fig, ax = plt.subplots()
zz = df['cflabel'].values
acc_vec = np.array(acc)

ax.scatter(zz, acc_vec[:,0])
plt.show()

FDIV = 3
CDIV = 3
for t in range(FDIV*CDIV):
    print('for type=>'+str(t))
    fig,ax= plt.subplots(2,10,figsize=(32,5))

    n=0
    for i in range(2):
        for j in range(10):
            if n < len(df[df['cflabel']==t].index):
                sn = df[df['cflabel']==t].index[n]
                (img, masks) = load_zipped_img(path+'/stage1_train.zip', sn)
                ax[i,j].imshow(img)
                n = n+1
    plt.show()
```



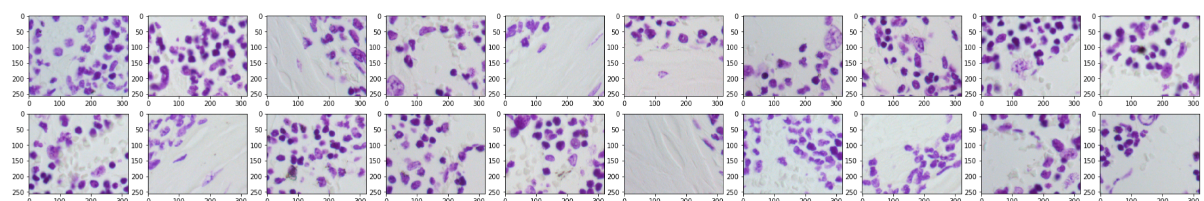
for type=>0



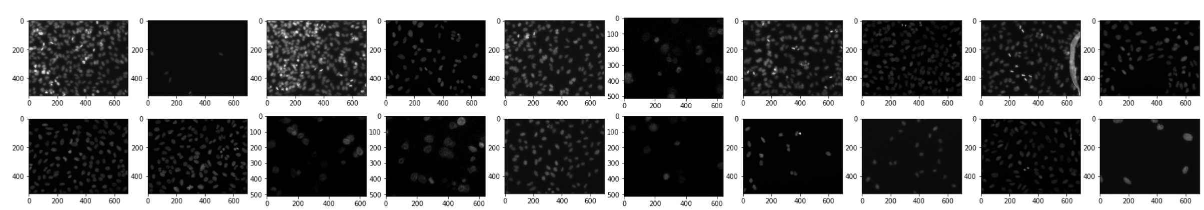
for type=>1



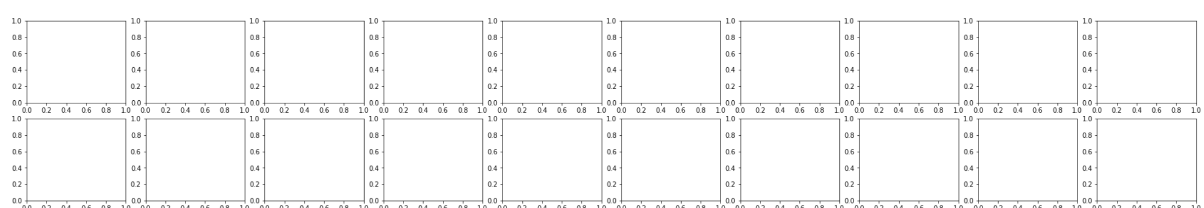
for type=>2



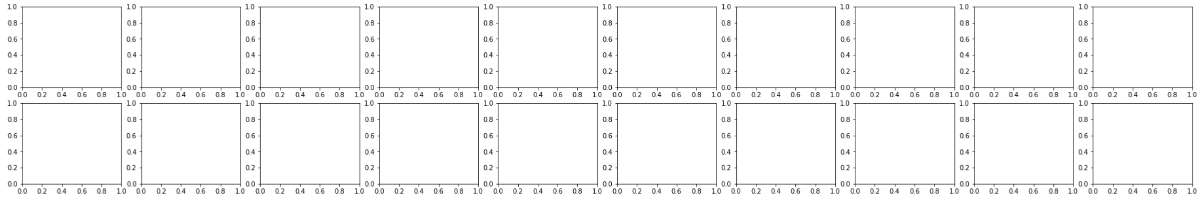
for type=>3



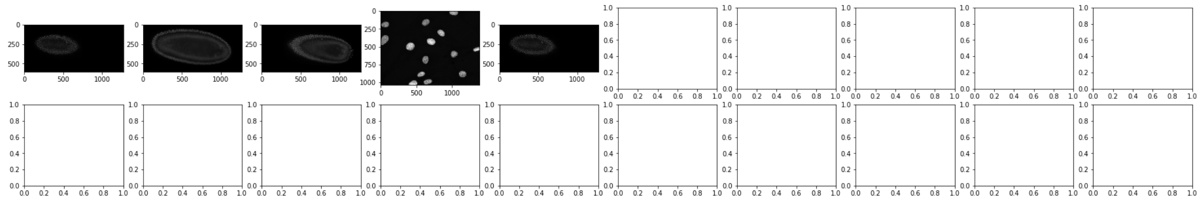
for type=>4



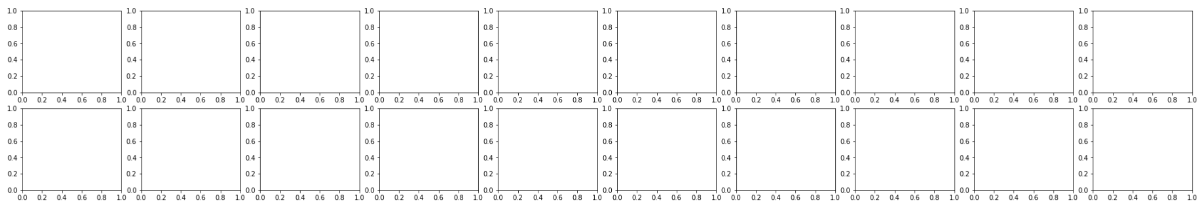
for type=>5



for type=>6



for type=>7



for type=>8