M&M's sorting with pictures

November 1, 2021

Jules de Schutelaere

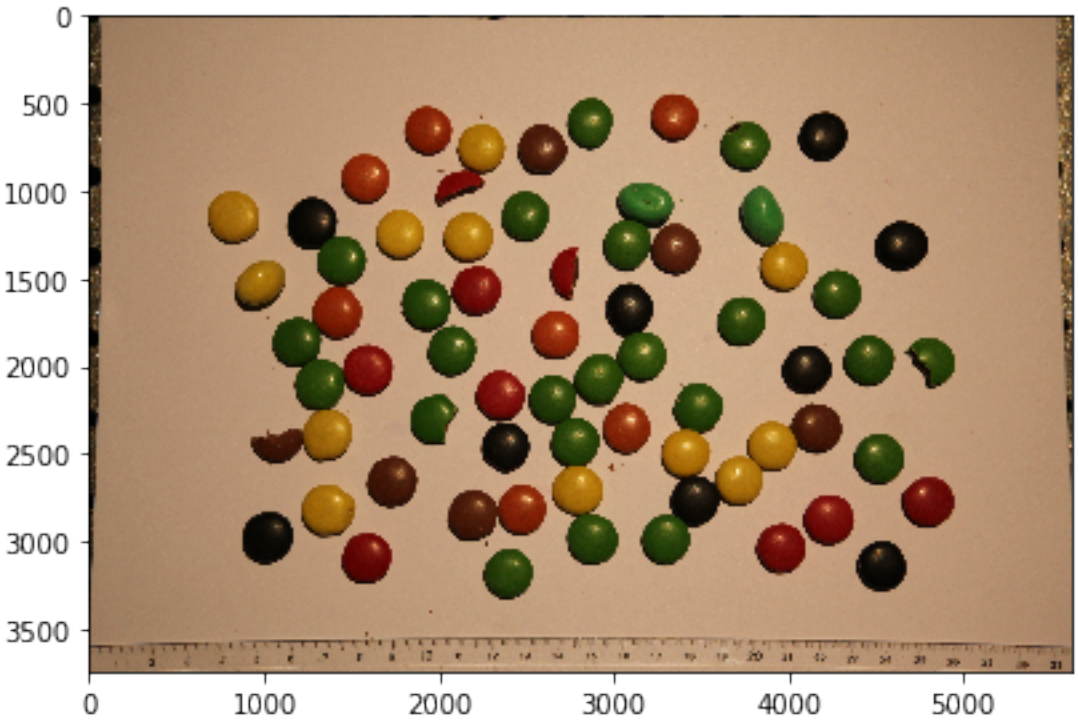
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from skimage import io
from skimage.filters import threshold_otsu
from skimage.color import rgb2gray
from skimage.morphology import binary_dilation
import cv2
```

0.1 Import M&M's image

[3]:
```
target = io.imread("IMG_2754_nonstop_alltogether.jpg")
```

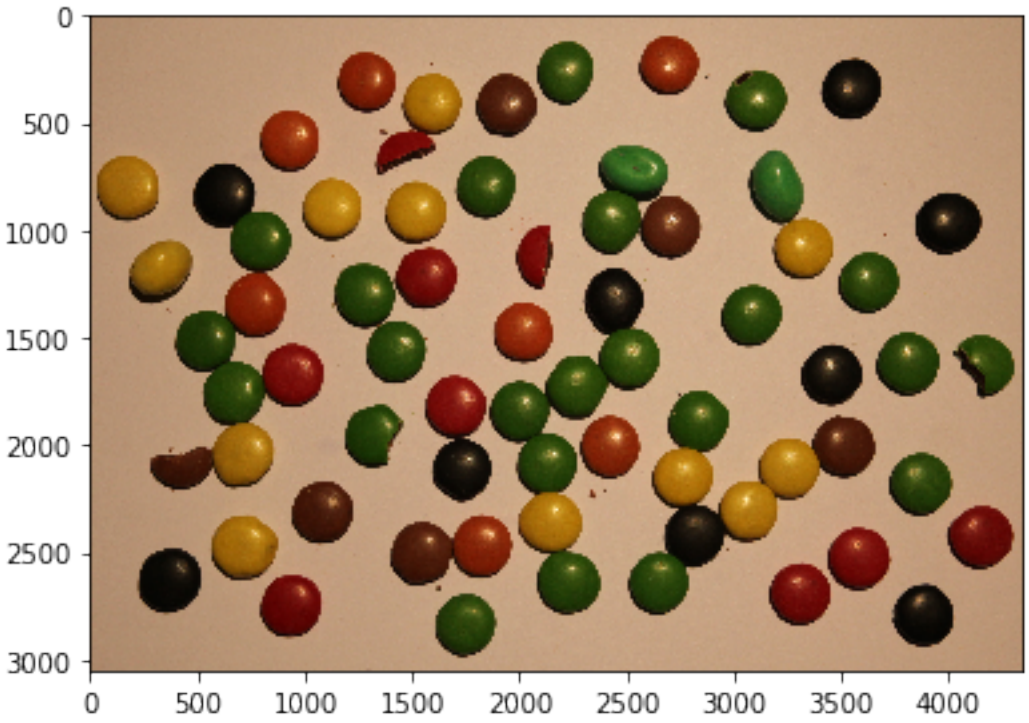
[4]:
```
io.imshow(target)
```

[4]:
```
<matplotlib.image.AxesImage at 0x18f83c86790>
```



[7]:
```
cropped_trgt = target[350:3400,650:5000]
io.imshow(cropped_trgt)
```

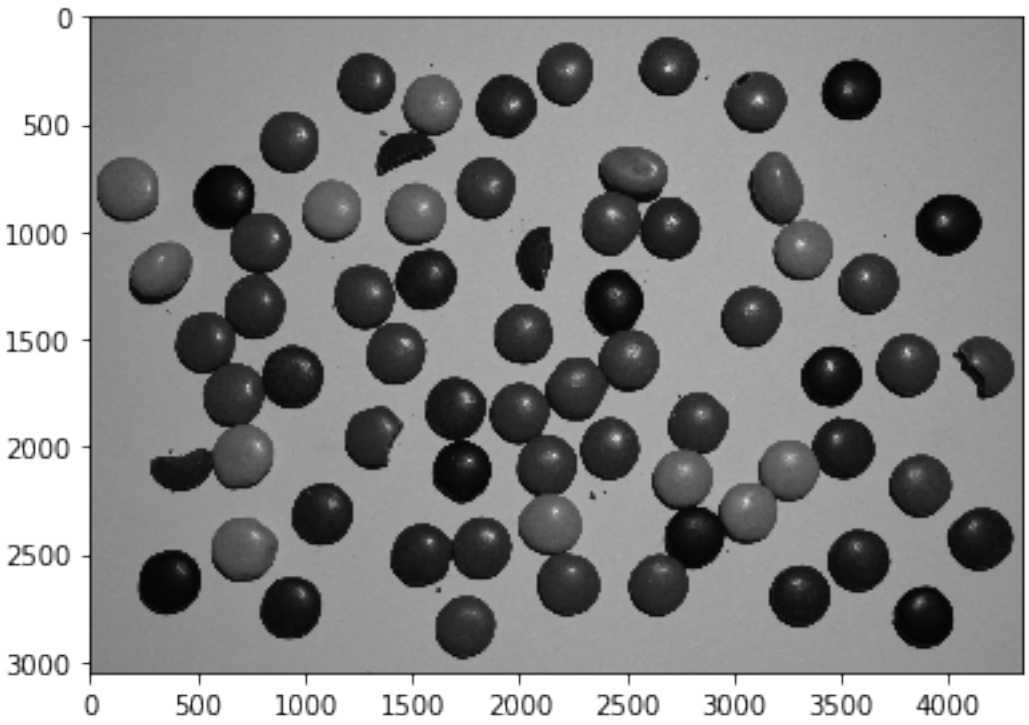
[7]: <matplotlib.image.AxesImage at 0x18f884b1520>



0.2 Change image to grayscale and increase contrast

[8]:
```
cropped_gray = cv2.cvtColor(cropped_trgt, cv2.COLOR_BGR2GRAY)
io.imshow(cropped_gray)
```

[8]: <matplotlib.image.AxesImage at 0x18f8b7f0910>

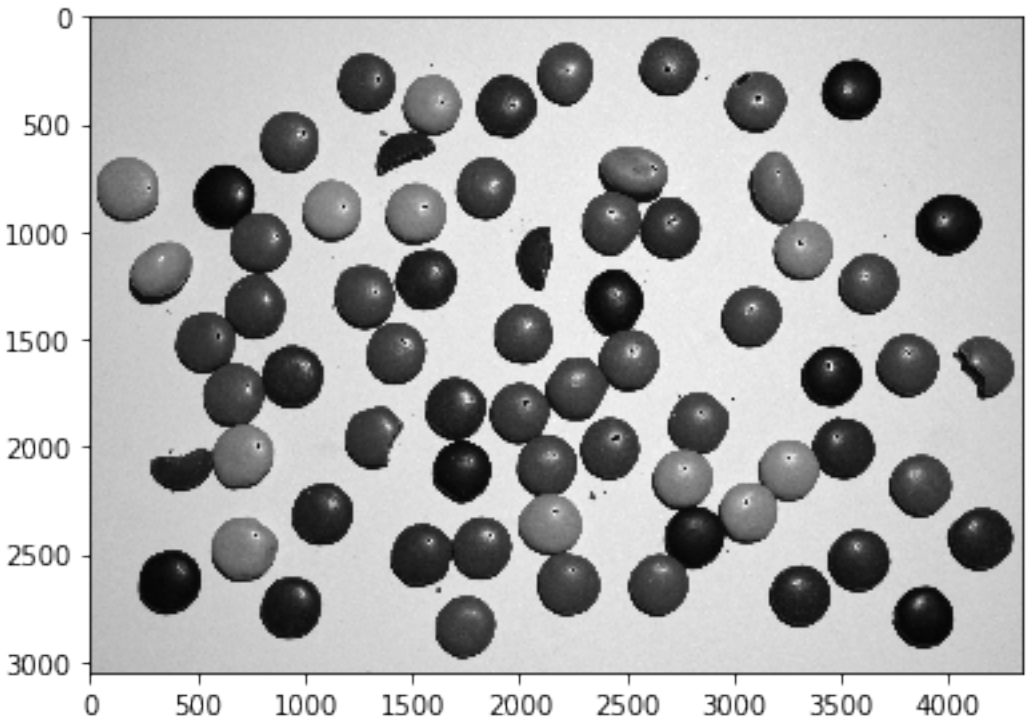


[9]:
```
def change_contrast(im, val):
    image = im.copy()
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            image[i,j] = image[i,j]* val
    return image
```

[10]:
```
cropped_gray = change_contrast(cropped_gray,1.4)
```

[11]:
```
io.imshow(cropped_gray)
```

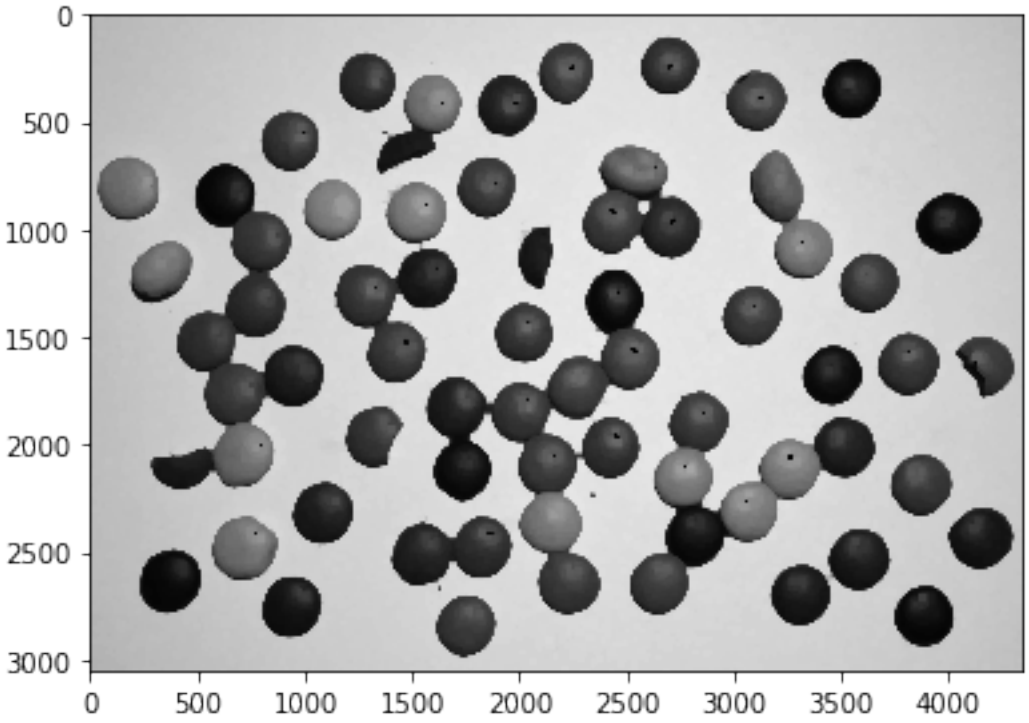
[11]:
```
<matplotlib.image.AxesImage at 0x18f8ab214f0>
```



0.3 Tuned opening of the picture to improve the Threshold

[12]:
```
kernel = np.ones((8, 8), np.uint8)
eroded_gray = cv2.erode(cropped_gray, kernel, iterations=3)
opened_gray = cv2.dilate(eroded_gray,kernel,iterations =4)
io.imshow(opened_gray)
```

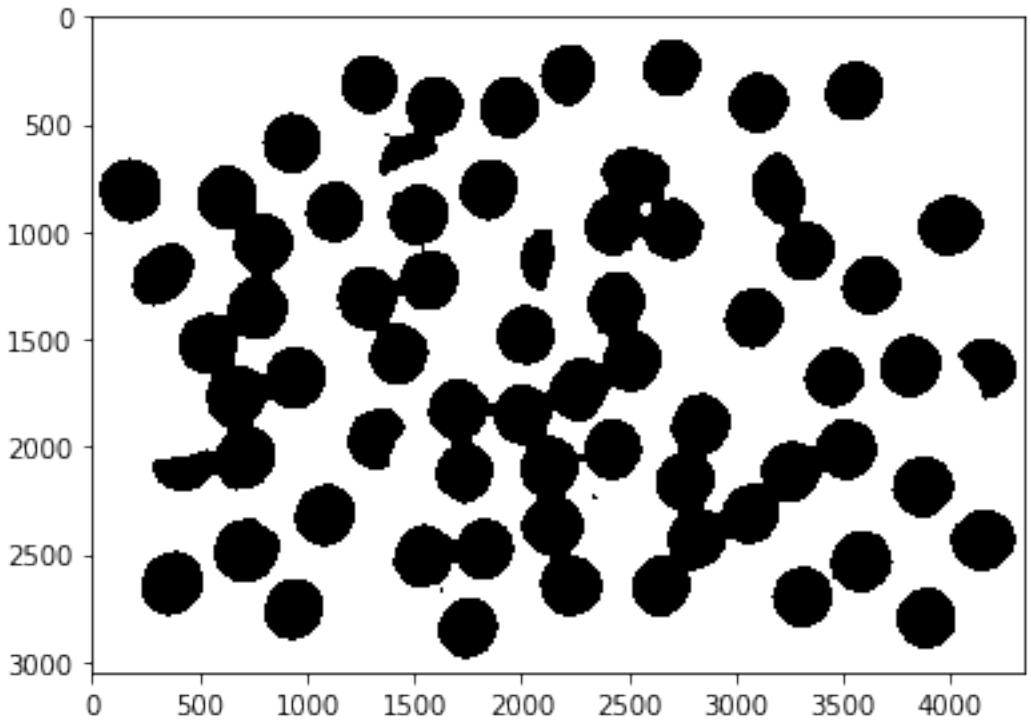
[12]:
```
<matplotlib.image.AxesImage at 0x18f8ab93820>
```



0.4 Threshold

[13]:
```
target_trsh = threshold_otsu(opened_gray)
binary = opened_gray < 169
inv_binary =  opened_gray > 169
io.imshow(inv_binary)
```

[13]: <matplotlib.image.AxesImage at 0x18f8ac05580>



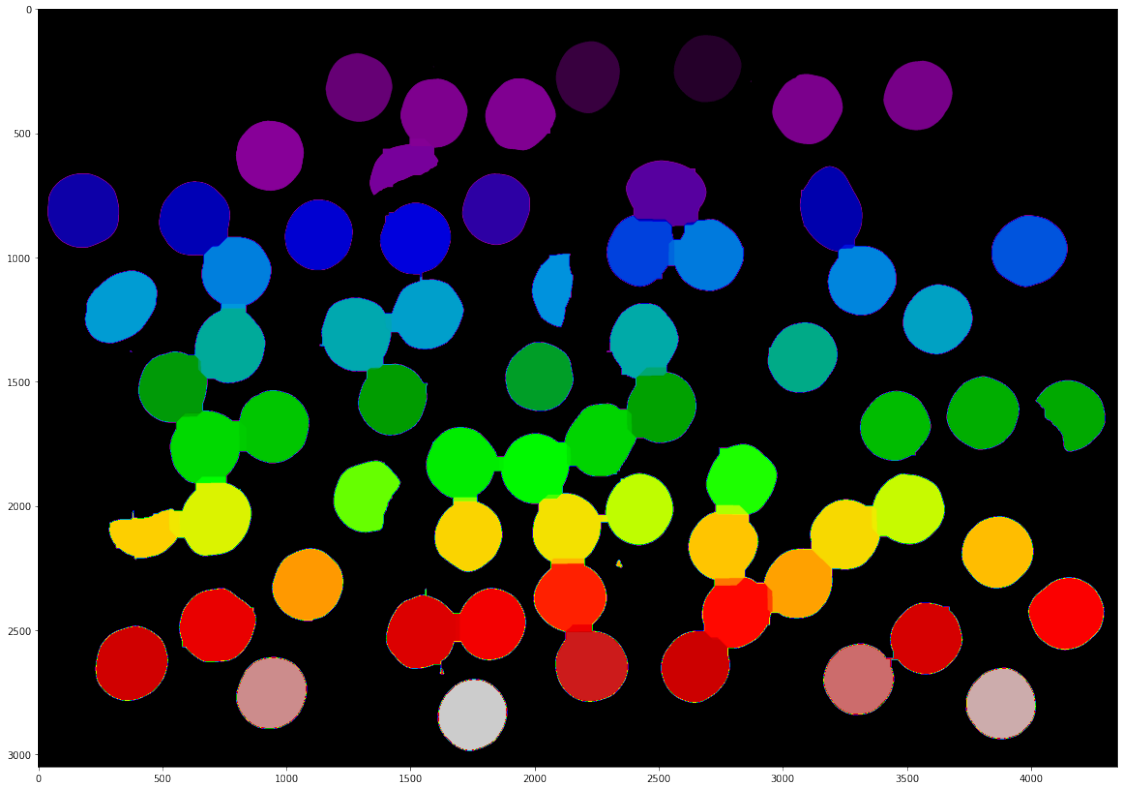
0.5 Lets separate objects using watershed

[18]:
```
from scipy import ndimage as ndi
from skimage.segmentation import watershed
from skimage.feature import peak_local_max

distance = ndi.distance_transform_edt(binary)
coords = peak_local_max(distance, footprint=np.ones((3, 3)), labels=binary,␣
 ↪min_distance=100)
mask = np.zeros(distance.shape, dtype=bool)
mask[tuple(coords.T)] = True
markers, _ = ndi.label(mask)
labels = watershed(-distance, markers, mask=binary)

plt.figure(figsize=[20,20])
plt.imshow(labels, cmap=plt.cm.nipy_spectral)
```

[18]:
```
<matplotlib.image.AxesImage at 0x18f8b5a19d0>
```



0.6 Now we differentiate broken from regular candies

The technique used here is to sort candies using their circularity. The candies are in the outputed labels list of watershed. We discard the small areas (noise). Then we calculate the circularity of each candy. Finally we set the best circularity level we can to sort the broken candies from the regular ones.

[15]:
```
from skimage.measure import find_contours, regionprops
```

[16]:
```
def circularity(area,perimeter):
    return 4*np.pi*(area/(perimeter**2))
```

[26]:
```
from skimage import measure
properties = measure.regionprops(labels)
centroids = []
for prop in properties:
    if prop.area > 50000 and circularity(prop.area, prop.perimeter) > 0.81:
        centroids+= [np.array(prop.centroid)]
        print(prop.perimeter, prop.area, prop.centroid, circularity(prop.area,␣
 ↪prop.perimeter))
```

```
891.619407771256 56864 (240.13836522228476, 2697.592800365785) 0.898852113342009
900.4478348960021 57419 (273.35847019279333, 2216.611696476776)
```

0.8899146828383768
897.9625535217635 57330 (317.0178440607012, 1293.2859236002093)
0.8934604921956587
917.5189027078675 59610 (348.73433987585975, 3546.812682435833)
0.8898134072049242
925.3178925810909 60522 (403.2446548362579, 3101.7538581011863)
0.8882622695515124
906.6488450227787 56353 (416.7248061327702, 1597.8879562756197)
0.8614869038464283
939.0752518939717 61728 (425.47693105235874, 1942.4513348885432)
0.8796122705562301
916.5483399593904 59994 (590.4814814814815, 935.1526152615262)
0.8974431198978342
919.8620484583752 60356 (805.201769500961, 1847.653539001922) 0.8963650435146003
980.0630585851518 68585 (811.1983961507618, 183.09544361011882)
0.8972861569040279
980.0041840821061 61924 (806.762886764421, 3196.7896292229184) 0.810238743547223
957.6610383315985 65395 (844.8412111017661, 631.9826133496445)
0.8960469593176581
914.7909806465098 59803 (909.963379763557, 1131.9352875273817)
0.8980263639586593
977.8204178980325 62852 (924.6198848087571, 1521.2685356074587)
0.8260584208780763
937.050865276332 57537 (972.904583137807, 2425.562368562838) 0.8234376983338039
967.3595231414334 66253 (973.2675350550164, 3994.6246358655458)
0.8896918034608956
962.3645737753169 62612 (991.5823324602312, 2701.8975436018654)
0.8495485158758158
923.5777772109134 60524 (1057.0098969004032, 799.4139680126892)
0.891642041479508
925.0924958366747 60007 (1091.7800923225623, 3320.2134750945725)
0.8811329926318086
956.732106143464 62571 (1198.3836281983668, 333.9762349970434)
0.8590179981035077
955.2346314604056 62443 (1228.6363723716029, 1568.917845074708)
0.8599506037996194
915.1757569573599 59744 (1249.3305603910017, 3625.9779894215317)
0.8963861673046024
978.1219330881976 65178 (1310.4960569517323, 1282.8233759857621)
0.8561007922518786
977.4772721475249 65755 (1356.7025625427725, 774.0081210554331)
0.8648191738870853
923.8620484583753 60473 (1402.4849436938798, 3081.3341821970134)
0.8903425397330024
909.1341263970173 58854 (1481.6446800557312, 2021.5193699663573)
0.8948081444696651
948.1219330881977 60736 (1520.7022853003161, 543.6278813224446)
0.8490390471206999

952.7493500861672 61652 (1570.765052228638, 1428.9707389865698)
0.8534925043965838
958.222438151586 61616 (1604.74618605557, 2512.670929628668) 0.8432778494251137
957.4600282048218 64379 (1626.399447024651, 3808.8635735255284)
0.8824960823578045
924.589970519733 60107 (1678.4388839902174, 3454.8484535910957)
0.8835610450958179
949.3767670841366 62169 (1679.3665975003619, 954.6621949846386)
0.8667753391730887
983.938166904124 63826 (1758.9998276564409, 675.0442766270799)
0.8284606354055714
965.9625535217635 63162 (1827.4658497197681, 1707.7982489471517)
0.8506387360401622
942.222438151586 59037 (1846.919118518895, 2005.3585209275539)
0.8356554689724874
939.5189027078675 58009 (1888.8978606767914, 2835.8058059956215)
0.8258367366704694
926.6904755831213 60219 (2013.6197545625134, 2424.162905395307)
0.8811990180076851
955.9625535217635 63047 (2012.9131917458403, 3509.6282773169223)
0.8669469615893981
1002.0803025278549 67278 (2051.4669282677846, 713.9952138886412)
0.841933685044728
950.6660889654818 59592 (2088.1525708148747, 2130.39919787891)
0.8285940687305693
932.5483399593904 59618 (2116.1118621892715, 3250.5045288335737)
0.8614787318236311
913.619407771256 58857 (2118.685559916408, 1735.57965917393) 0.8860890153260673
944.4234482783626 59807 (2159.8505358904476, 2762.383634022773)
0.8426133966291837
939.4600282048218 63351 (2185.580937948888, 3867.6590424776246)
0.9020003901787117
922.4062043356594 58294 (2311.8823034960715, 3067.9805811918895)
0.8609725191758695
948.2468247692254 63695 (2317.335442342413, 1088.0263285972212)
0.890168888783413
950.4062043356595 61459 (2362.9846076245954, 2145.0348362322848)
0.8550210917297669
952.0214280248092 58838 (2430.4281926646045, 2816.531561235936)
0.815782376290046
971.0163773909259 66702 (2433.16860064166, 4146.2535905969835)
0.8889874096808907
923.2346314604056 60723 (2476.238476359864, 1828.9336659914695)
0.8952388274467953
1006.0041840821061 70192 (2481.975581262822, 721.2049236380215)
0.871561237959989
975.7493500861672 64418 (2508.9779564717937, 1547.015896178087)
0.8502380950693053

```
962.891485709898 64081 (2533.1729061656342, 3580.2031335341208)
0.868529254744021
980.0041840821061 67674 (2634.014215208204, 377.33547595827054)
0.8854740767846839
925.7615433949869 60568 (2646.8307191916524, 2650.1533483027342)
0.8880855938619013
970.0214280248092 65730 (2642.893579796136, 2229.3352959075005)
0.8778308516602233
947.8620484583752 62243 (2696.9309801905433, 3308.1436788072556)
0.8705828569346087
944.7909806465098 63470 (2752.654104301245, 942.1836458169214)
0.8935253637842971
930.8326112068523 61747 (2796.2924190648937, 3881.8055937940303)
0.8955349626007223
923.3178925810909 61007 (2840.268936351566, 1751.6604815840806)
0.8992636318085244
```

0.7 Graphic representation of the sorting

Regular candies are surrounded

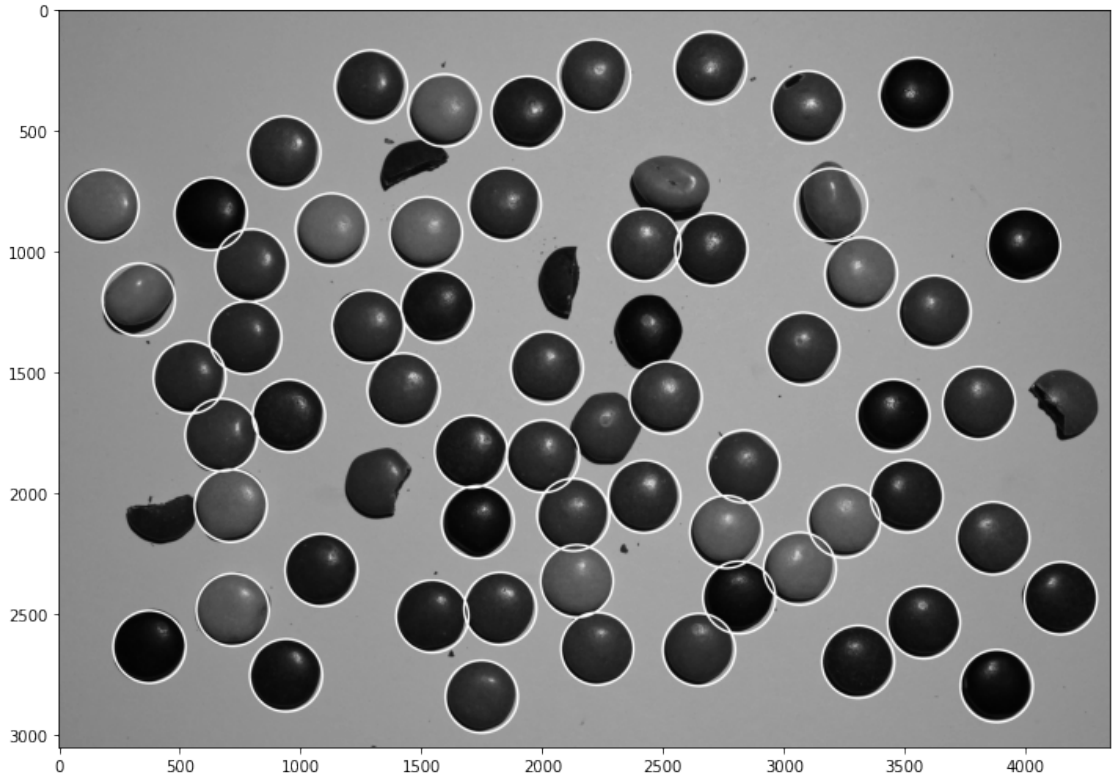
[27]:
```
from skimage.draw import circle_perimeter
from skimage import morphology

fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(12, 12))

image = cv2.cvtColor(cropped_trgt, cv2.COLOR_BGR2GRAY)

for centroid in centroids:
    circy, circx = circle_perimeter(int(centroid[0]), int(centroid[1]), 150,
                                    shape=image.shape)
    color = (255, 255, 255)
    cv2.circle(image, np.flip(centroid.astype(int)), 150, color, thickness = 7)

ax.imshow(image, cmap=plt.cm.gray)
plt.show()
```



I'm quite happy with the result I have. I managed to exclude all the broken M&M's, but couldn't differentiate circular candies from oval ones.