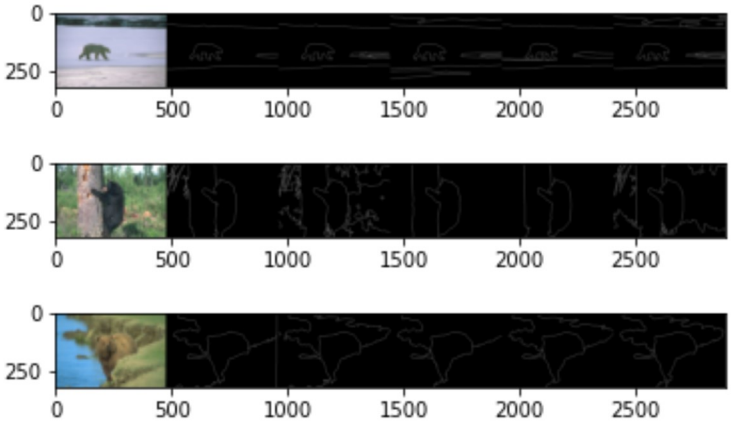
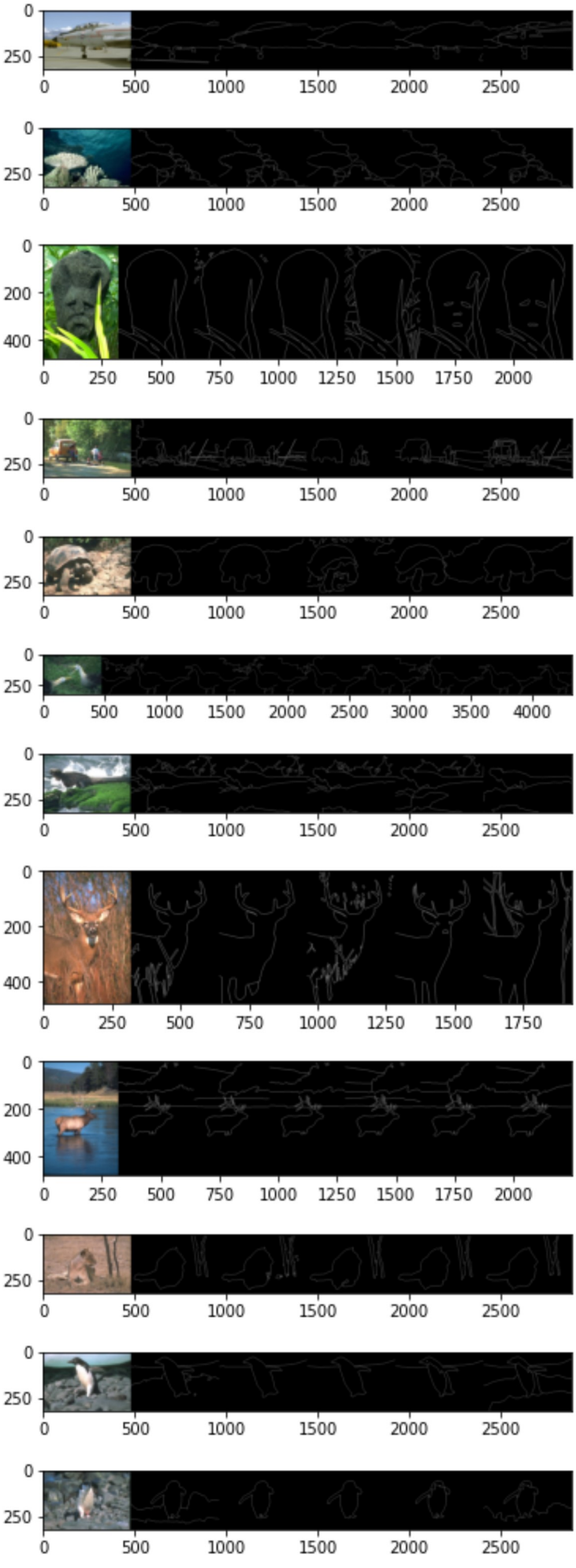
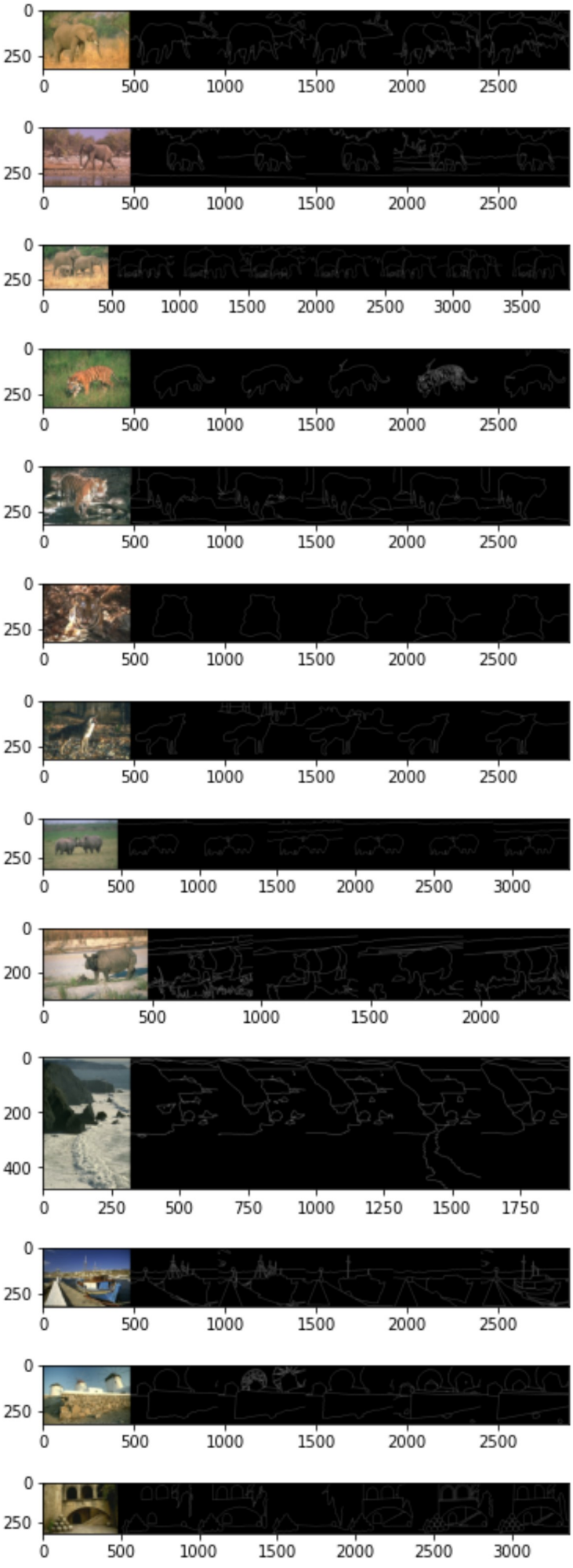
In [1]:
```
import os
import cv2
import imageio
import numpy as np
from PIL import Image
from matplotlib import pyplot as plt
from scipy import io
from sklearn.neighbors import kneighbors_graph
from sklearn.cluster import KMeans
from scipy.spatial import distance_matrix
```

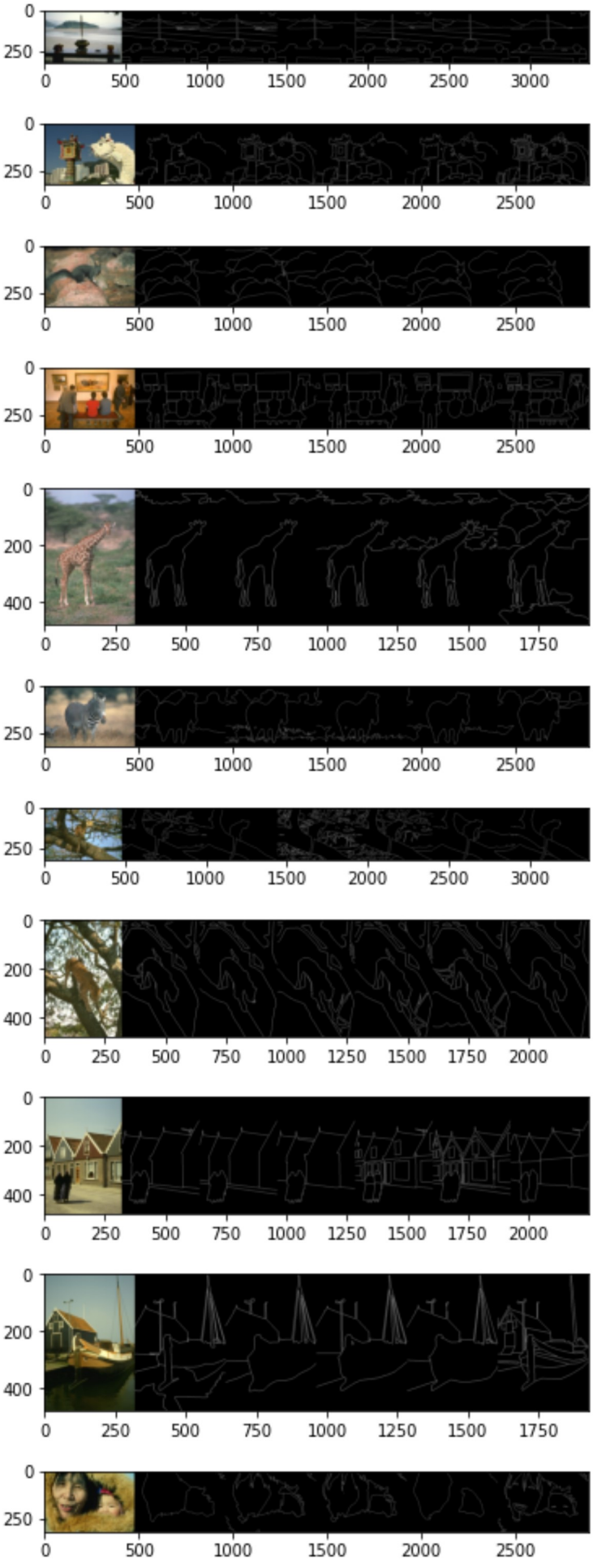
In [2]:
```
gt_path = 'BSR\\BSDS500\\data\\groundTruth\\test'
pics = os.listdir(gt_path)[0:50]

# create ground truth array

groundTruth = []

for filename in pics:

    data = io.loadmat(os.path.join(gt_path, filename))
    c = data['groundTruth'].shape[1]

    groundTruthPerImage = []
    for n in range(c):
        edge_data = data['groundTruth'][0][n][0][0][0]
        groundTruthPerImage.append(edge_data)

    groundTruthPerImage = np.array(groundTruthPerImage)
    groundTruth.append(groundTruthPerImage)

groundTruth = np.array(groundTruth,dtype=object)

#plt.imshow(groundTruth[0][0])
#plt.show()
```

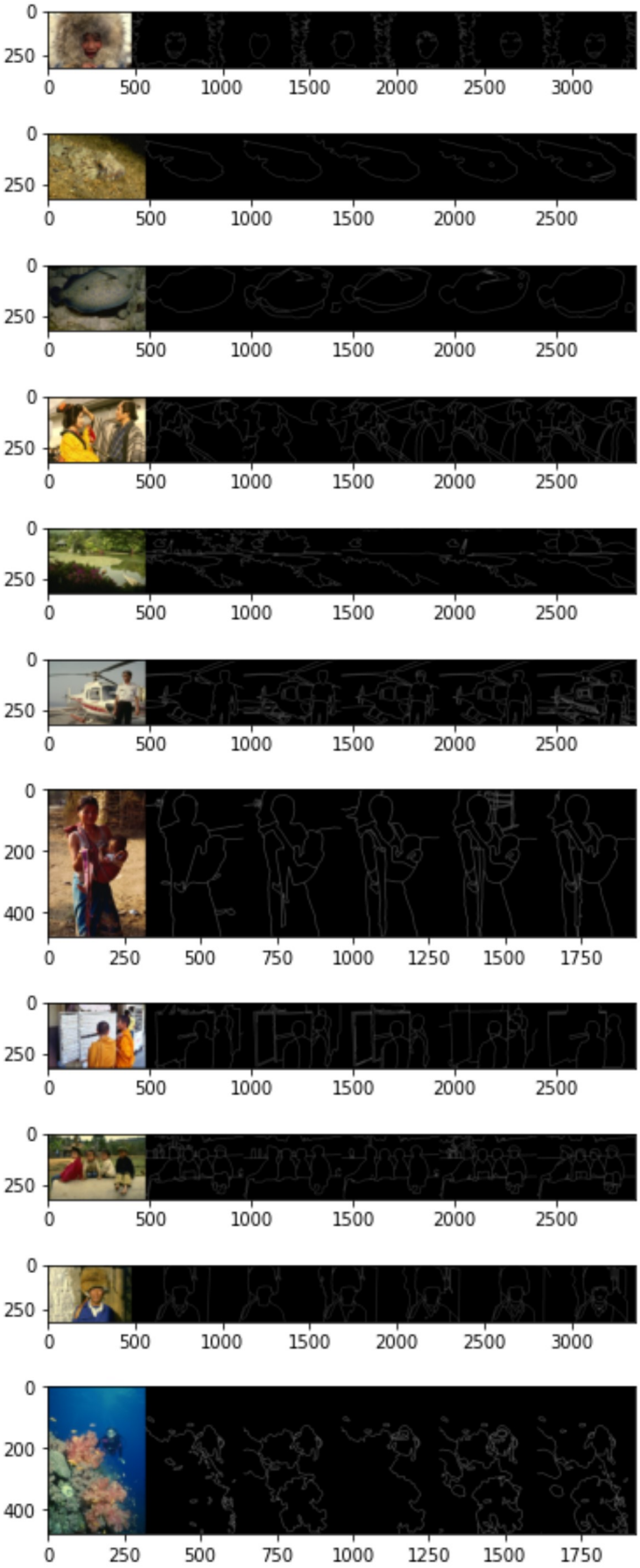
In [3]:
```
##########################################################################################

root = "BSR\\BSDS500"
PATH = os.path.join(root, 'data\\groundTruth\\test')

save_pth = os.path.join(root, 'data\\converted_mat_jpg', 'test')
os.makedirs(save_pth, exist_ok=True)  #to save gt converted images
pics = os.listdir(PATH)[0:50]

# create converted images for ground truth segmentation
i = 0
for filename in pics:
    data = io.loadmat(os.path.join(PATH, filename))
    c = data['groundTruth'].shape[1]
    for n in range(c):
        edge_data = data['groundTruth'][0][n][0][0][1]
        edge_data_255 = edge_data * 255
        new_img_name = filename.split('.')[0] + '(' + str(n) + ')' + '.jpg'
        imageio.imwrite(os.path.join(save_pth, new_img_name), edge_data_255)  #
        pics[i] = filename.split('.')[0] + '.jpg'
    i = i + 1


# Visualize the image and the ground truth segmentation

root = "BSR\\BSDS500\\data\\images\\test"
converted = "BSR\\BSDS500\\data\\converted_mat_jpg\\test"

for filename in pics:
    gt = []
    im = Image.open(os.path.join(root, filename))

    for file in os.listdir(converted):
        if file.startswith(filename.split('.')[0]):
            gt.append(file)

    new_im = Image.new('RGB', (im.width * (len(gt) + 1), im.height))  #HORIZONTA
    new_im.paste(im, (0, 0))

    for i in range(len(gt)):
        im_gt = Image.open(os.path.join(converted, gt[i]))
        new_im.paste(im_gt, (im.width * (i + 1), 0))
    #plt.imshow(new_im)
    #plt.show()

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






0
250
0 500 1000 1500 2000 2500
0
250
0 500 1000 1500 2000 2500
0
250
0 500 1000 1500 2000 2500 3000 3500
0
250
0 500 1000 1500 2000 2500
0
250
0 500 1000 1500 2000 2500
0
250
0 500 1000 1500 2000 2500
0
250
0 500 1000 1500 2000 2500
0
250
0 500 1000 1500 2000 2500 3000
0
200
0 500 1000 1500 2000
0
200
400
0 250 500 750 1000 1250 1500 1750
0
250
0 500 1000 1500 2000 2500
0
250
0 500 1000 1500 2000 2500
0
250
0 500 1000 1500 2000 2500 3000


0
250
0 500 1000 1500 2000 2500 3000
0
250
0 500 1000 1500 2000 2500
0
250
0 500 1000 1500 2000 2500
0
250
0 500 1000 1500 2000 2500
0
200
400
0 250 500 750 1000 1250 1500 1750
0
250
0 500 1000 1500 2000 2500
0
250
0 500 1000 1500 2000 2500 3000
0
200
400
0 250 500 750 1000 1250 1500 1750 2000
0
200
400
0 250 500 750 1000 1250 1500 1750 2000
0
200
400
0 250 500 750 1000 1250 1500 1750
0
250
0 500 1000 1500 2000 2500


0
250
0 500 1000 1500 2000 2500 3000
0
250
0 500 1000 1500 2000 2500
0
250
0 500 1000 1500 2000 2500
0
250
0 500 1000 1500 2000 2500
0
250
0 500 1000 1500 2000 2500
0
250
0 500 1000 1500 2000 2500
0
200
400
0 250 500 750 1000 1250 1500 1750
0
250
0 500 1000 1500 2000 2500
0
250
0 500 1000 1500 2000 2500
0
250
0 500 1000 1500 2000 2500 3000
0
200
400
0 250 500 750 1000 1250 1500 1750

In [4]:
```
# original images

images = []

images_path = "BSR\\BSDS500\\data\\images\\test\\"

for filename in pics:
    img = cv2.imread(os.path.join(images_path, filename))
    im_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    images.append(im_rgb)

images = np.array(images, dtype=object)
plt.imshow(images[0])
plt.show()

flattened_images = np.empty((0, 3))

for i in range(50):
    flattened_image = images[i].reshape(images[i].shape[0] * images[i].shape[1],
    flattened_images = np.concatenate((flattened_images, flattened_image))
```

In [11]:
```
from sklearn.cluster import KMeans


# k = 3 5 7 9 11
def image_segmentation(k):
    segmented_images = []
    flat_segmented_images = []

    k_means = KMeans(n_clusters=k)
    k_means.fit(flattened_images)

    # build segmented images from clustering labels

    labels = k_means.labels_
    labels.reshape(labels.shape[0], 1)

    index = 0
    for i in range(50):
        size = images[i].shape[0] * images[i].shape[1]
        flat_segmentation = labels[index:size + index]

        segmented_images.append(flat_segmentation.reshape(images[i].shape[0], im

        flat_segmented_images.append(flat_segmentation)
        index += size

    flat_segmented_images = np.array(flat_segmented_images, dtype=object)

    return segmented_images, flat_segmented_images
```

In [12]:
```
def recreate_groundTruth(image_index, gt_index, flat_segmented_images, k):

    import sklearn.metrics.cluster as sk
    from scipy.optimize import linear_sum_assignment

    flattened_gt = np.array(groundTruth[image_index][gt_index]).reshape(
        images[image_index].shape[0] * images[image_index].shape[1], 1)


    # add dummy classes to match k clusters in case ground truth clusters < k
    max_gt = flattened_gt.max()
    if max_gt < k:
        j = 1
        for i in range(max_gt,k+1):
            flattened_gt[i][0] = max_gt + j
            j+=1

     # add dummy classes to match k clusters in case k_means classes < k
    max_seg = flat_segmented_images.max()
    if max_seg < k:
        j = 1
        for i in range(max_seg,k+1):
            flat_segmented_images[i] = max_seg + j
            j+=1

    contingency_matrix = sk.contingency_matrix(flattened_gt, flat_segmented_imag
    gt_clusters = contingency_matrix.shape[0]

    # max matching

    idx = linear_sum_assignment(-contingency_matrix.T)[1]
    for i in range(k):
        contingency_matrix[[idx[i], i]] = contingency_matrix[[i, idx[i]]]
        replaced_index = np.argwhere(flattened_gt == (idx[i]+1))
        # give the major k clusters new values
        flattened_gt[replaced_index] = gt_clusters + i + 1

    # give the minor clusters (which will be merged into the major clusters) new
    values_gt = np.unique(flattened_gt)
    for i in range(gt_clusters - k):
        replaced_index = np.argwhere(flattened_gt == (values_gt[i]))
        flattened_gt[replaced_index] = gt_clusters + k + i + 1


    # merge the minor clusters into the major clusters
    contingency_matrix = sk.contingency_matrix(flattened_gt, flat_segmented_imgs
    for i in range(k,gt_clusters):
        replaced_index = np.argwhere(flattened_gt == (gt_clusters + i + 1))
        flattened_gt[replaced_index] = gt_clusters + np.argmax(contingency_matri
        contingency_matrix[np.argmax(contingency_matrix[i])] += contingency_matr
    contingency_matrix = contingency_matrix[:k,:]


    new_gt = flattened_gt.reshape((images[image_index].shape[0], images[image_in

    return new_gt,contingency_matrix
```

In [13]:
```
def evaluate_segmentation(image_index, gt_index, flat_segmented_images, k):

    new_gt,contingency_matrix = recreate_groundTruth(image_index,gt_index,flat_s

    # calculate F-score

    precision = np.zeros((k, 1))
    recall = np.zeros((k, 1))
    f_score = np.zeros((k, 1))

    for i in range(k):
        TP = np.amax(contingency_matrix[:, i])
        precision[i] = TP / np.sum(contingency_matrix[:, i])
        recall[i] = TP / np.sum(contingency_matrix[i])

        f_score[i] = (2 * precision[i] * recall[i]) / (precision[i] + recall[i])

    f_score_avg = np.mean(f_score)

    ##############################################################################

    # calculate conditional Entropy

    pixels_count = images[image_index].shape[0] * images[image_index].shape[1]
    entropy = 0
    for i in range(k):
        sum_cluster = np.sum(contingency_matrix[:, i])

        H_cluster = 0
        for j in range(k):
            n_ij = contingency_matrix[i][j]
            if n_ij != 0:
                H_cluster += (n_ij / pixels_count) * np.log10(n_ij / sum_cluster

        entropy -= H_cluster

    ##############################################################################

    return f_score_avg, entropy , new_gt
```

In [8]:
```
k_clusters = [3,5,7,9,11]

fig, ax = plt.subplots(1, 5, figsize=(35, 10))
seg_fig, seg_axs = plt.subplots(50, 5, figsize=(70, 500))

for k in k_clusters:

    plot = k_clusters.index(k)

    F_score_entropy_list = np.zeros((50, 2))
    segmented_imgs, flat_segmented_imgs = image_segmentation(k=k)

    for img_index in range(50):
        seg_axs[img_index][plot].imshow(segmented_imgs[img_index])

        seg_f_score_avg = 0
        seg_entropy_avg = 0
        gt_count = len(groundTruth[img_index])

        for gt_img_index in range(gt_count):
            F_score, Entropy,_ = evaluate_segmentation(img_index, gt_img_index,
            seg_f_score_avg += F_score
            seg_entropy_avg += Entropy

        seg_f_score_avg /= gt_count
        seg_entropy_avg /= gt_count

        F_score_entropy_list[img_index][0] = seg_f_score_avg
        F_score_entropy_list[img_index][1] = seg_entropy_avg

    print("for k = {} : F-score per dataset: {}   Entropy per dataset: {}".forma
        F_score_entropy_list[:, 0]) / 50, np.sum(F_score_entropy_list[:, 1]) / 5

    ax[plot].set_axis_off()
    table = ax[plot].table(
        cellText=F_score_entropy_list,
        rowLabels=[r + 1 for r in range(50)],
        colLabels=["Average F-score", "Average Conditional Entropy"],
        cellLoc='center',
        loc='upper left')
    table.scale(1, 5)
    ax[plot].set_title('for k = {}'.format(k))

plt.show()
```

```
for k = 3 : F-score per dataset: 0.7025027323162596   Entropy per dataset: 0.267
7956151581025
for k = 5 : F-score per dataset: 0.6807246951194934   Entropy per dataset: 0.327
3995441475374
for k = 7 : F-score per dataset: 0.6987563859007224   Entropy per dataset: 0.325
0597296048128
for k = 9 : F-score per dataset: 0.7475784559070928   Entropy per dataset: 0.360
0018064443664
for k = 11 : F-score per dataset: 0.7810211292750207   Entropy per dataset: 0.38
2556285253411
```

for k = 3

	Average F-score	Average Conditional Entropy
1	0.8356387318691432	0.12122214738696293
2	0.6556247183340151	0.32459860794591
3	0.6993046317140634	0.23062962173666588
4	0.6073934326247493	0.2591285779560174
5	0.6861047329962295	0.21944093695640804
6	0.6360935204745983	0.31749782745334004
7	0.7234303034168905	0.34191184583673045
8	0.5553440453883841	0.3838837781153931
9	0.6317082775160671	0.18508624510114025
10	0.6791962845398595	0.27329696029906833
11	0.7900469547805326	0.18139107819005768
12	0.7647904212861425	0.23515201782684467
13	0.7284354144377144	0.28382406295888646
14	0.8936282672678502	0.22100586388406745
15	0.8854196763241665	0.24771542846825692
16	0.4975899803952929	0.17876419819707544
17	0.765775698915306	0.30804218279594175
18	0.6845755823208414	0.26909523605446006
19	1.031579249313984	0.19673724427268508
20	0.5759617914976458	0.32747147145388834
21	0.6838017770433235	0.34918997804888025
22	0.7441884694744633	0.07475340893320467
23	0.7985653396963285	0.15867362429147763
24	0.6822932316693877	0.2719074229424535
25	0.8440926637959514	0.18650575568832115
26	0.6623613203210663	0.3228802307036637
27	0.5589585240299282	0.34546938376614356
28	0.7248646117671211	0.28435626194033395
29	0.7163683854364669	0.2539886382926115
30	0.7665144506536422	0.2727190573277046
31	0.7058147333605798	0.32226333668777013
32	0.7135833918123631	0.29450669814293325
33	0.6978080931036467	0.2202490723122026
34	0.6177932935804554	0.31376086561049726
35	0.6225532264652358	0.33192031853591364
36	0.639389941264341	0.34058646007387664
37	0.7111009948884822	0.26878720571315745
38	0.7082598705686394	0.25726230855328575
39	0.6122088917974369	0.2788817659127073
40	0.8518178190038744	0.28148236952247835
41	0.7584556590725394	0.23563920933001828
42	0.5772278707941253	0.25450277444869907
43	0.5654772419959138	0.40116166642262563
44	0.5887739612201324	0.28949982119292006
45	0.6581810631909687	0.30057924464122554
46	0.6155594927695107	0.32692037146068432
47	0.7011466762042757	0.30452180873513174
48	0.7684413508870479	0.2744284355317486
49	0.7599137706263156	0.23467764435140567
50	0.741978913107268	0.23181276927191713

for k = 5

	Average F-score	Average Conditional Entropy
1	0.6569756452006299	0.2764358615718778
2	0.5886921680021898	0.4732754823720107
3	0.4694532506064046	0.006534349503038705
4	0.5561032210209056	0.3099796444615653
5	0.6126128043347274	0.33253989968307124
6	0.587199832861264	0.4518399409008517
7	0.8293950302453247	0.4522801880886882
8	0.5761806969031043	0.4799277810627215
9	0.5721368744340748	0.23459749896759635
10	0.7358966805741055	0.3912094445038753
11	0.7545870225730698	0.34855377754469663
12	0.5181361841513619	0.3052315763905465
13	0.8046150554964339	0.23322416154610942
14	0.9327844724906343	0.288487627104914 3
15	1.1085193818489	0.31102181191134687
16	0.6832951241625629	0.3258597025434943
17	0.5674949437922481	0.3076715560118234
18	0.7464931338476141	0.3639379549276173
19	1.1667284130138131	0.24834992362577477
20	0.5770575853519964	0.5069061978146729
21	0.8772421583948091	0.2950034435157546
22	0.8463404044427382	0.3291625047371852
23	0.7313404099636367	-0.13072426125785222
24	0.5501932728160834	0.45741651006437717
25	0.8647401455274515	0.29131107093260453
26	0.5507827048642907	0.41503821177946787
27	0.5128493398969791	0.4928657464761755
28	0.6003481598557949	0.38454606973553414
29	0.6475148292888231	0.30371602402528647
30	0.6486242215190909	0.31271120148482423
31	0.5444661007017774	0.3756206779525499
32	0.6881751422866639	0.40604555099347717
33	0.7578808916557345	0.34170123599873636
34	0.6298456757368106	0.3873795595732645
35	0.543448582271112	0.4096514674891872 23
36	0.5613701353446116	0.503713871884693
37	0.4974735054140746	0.39456386505282887
38	0.5719714441555244	0.18395260216423626
39	0.6168622947291376	0.11576433989664943
40	1.026223822032039	0.33281122129073 87
41	0.9765060618920958	-0.4908927521281169
42	0.8760661528333186	0.36866823338534693
43	0.5412173425000787	0.5119979341197489
44	0.5819531442053 27	0.3775428964619807
45	0.6030399896128685	0.3092544789546899
46	0.4694385278453724	0.43239545954514 68
47	0.5684281383636318	0.4444351549671636
48	0.6817956855271851	0.3842088481122191
49	0.6903413945586744	0.3718882745301029
50	0.7433977071290114	0.32349418725950563

for k = 7

	Average F-score	Average Conditional Entropy
1	0.6430721322442132	0.20383747836318006
2	0.5855733972944336	0.5211616048282517
3	0.816396723555826	0.15512170895330796
4	0.4730565484490454	0.2448737451361008 8
5	0.673151849253349	0.30082804719803746
6	0.5752699793917009	0.46523320659156586
7	0.8381606550840963	0.5032482946681198
8	0.6470544148384116	0.4743655319032139
9	0.6514406014737664	0.20862346172206758
10	0.767895240067951 7	0.3931337632288244
11	0.9868284806829106	0.3321956331896986
12	0.7233629282223563	0.3587675758834747
13	0.8604345084733586	0.3368173103261956 6
14	0.9856718325294231	0.3023213307891798
15	1.3519438827995696	0.3442368475463314
16	0.5420925643705135	0.2966489415553174
17	0.6978004094641763	0.3701973363415500 5
18	0.5683448509285278	0.2991425541317181
19	1.0762507885146464	0.2430117209517886 6
20	0.5391598637050582	0.4926221386884831 5
21	1.043507576225152	0.4629606704387950 6
22	1.001469376875628	0.2882051307021203 4
23	0.9332536717 23806	0.0659368890794789 1
24	0.6149019758130454	0.501385378945707
25	0.7696041325350386	0.3181441237603656
26	0.5604511549761173	0.3893508791583954
27	0.6413612842040168	0.3963461338683323
28	0.4291211827487091	0.170750904843195
29	0.6887967028699494	0.2770568459719313
30	0.604240157262425	0.3188608881949237
31	0.3982744705040479 6	0.3170291842805693
32	0.5013631291053419	0.0071917291822497 46
33	0.7931913243632881	0.351754489843595 9
34	0.7090281974286265	0.3773005184741779
35	0.524896515061085	0.4782184966604798 3
36	0.4341864317498803	0.4617872976130814 6
37	0.5045758308456413	0.2497389515129987
38	0.5488291124799876	0.0539432371909221 6
39	0.7179859517675301	0.1411170360649128 3
40	0.8829640574660677	0.3222263296017905
41	1.106370173429342	0.2331590018461239 6
42	1.0494756790540 71	0.3876950247698379 7
43	0.3937752442914502	0.3727427255088599
44	0.6044980529200935	0.3720224344074863
45	0.5025733595110737	0.2006777710395895 6
46	0.4584732908354324	0.3737186197686846
47	0.5660125898841593	0.4704540429342163 3
48	0.533328270409093	0.3427077110366676
49	0.7408053969159779	0.4042374594882705
50	0.8775436649881163	0.3198251006794863 3

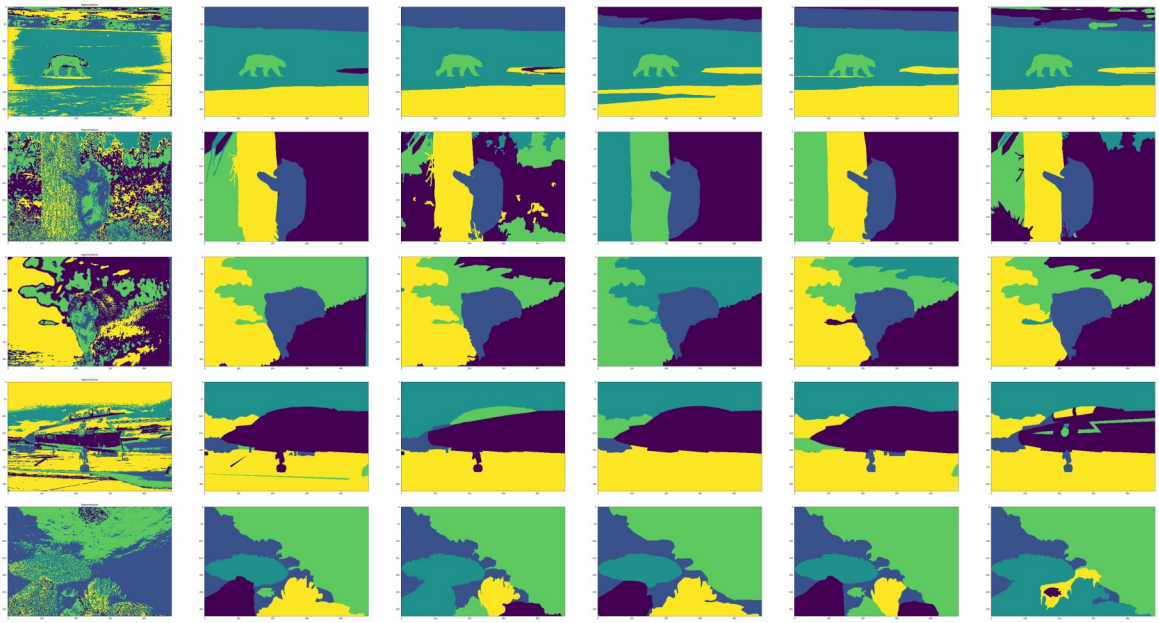
for k = 9

	Average F-score	Average Conditional Entropy
1	0.7704572996095849	0.3469924181402785 4
2	0.6669089136684168	0.4090343543824020 4
3	0.9697748003783952	0.1255168520113637 8
4	0.6009610579802562	0.2275148808720002 8
5	0.6755003510234376	0.3214989426956213
6	0.6111680033775247	0.527088666263 8
7	0.8114116437830962	0.2443254496808285 6
8	0.8258798835152087	0.4900844562343756
9	0.7019520737255298	0.2791162972713544
10	0.7106341066144551	0.3116468127585364
11	1.081186366343838	0.4180126143342993
12	0.8048193177589965	0.4199613018806509 6
13	1.0054509102409614	0.4204695666084013
14	1.0575745036303585	-0.4790566393986798
15	1.3981044445059 19	0.2953422229367698 7
16	0.6445926840344505	0.4098114387404233 6
17	0.7342765365461736	0.4704217747882115
18	0.6602092616043649	0.4181635769578231 4
19	1.1299514049465096	0.3287264828329211 8
20	0.5560395750104606	0.4531944130405081 7
21	1.0902589783863301	0.5333546084380988
22	1.1010946477999197	0.4262071865167388 3
23	1.1209712120767616	0.1834188046356172 4
24	0.5683067362856622	0.5473192284969891
25	0.7242993461884469	0.2839517291653355 6
26	0.5251658497516759	0.4690762667837705 7
27	0.763603488304383	0.4289446428186599
28	0.5447693070870223	0.4319969033753847
29	0.4604869887328724	0.2874164334728793
30	0.6205514292609442	0.3865255252511896
31	0.4042564930361854	0.3773812049636214
32	0.4809610732509134	0.0685581866569437
33	0.8341492710209563	0.2799495605199597
34	0.8410421402212611	0.3912504092930852 4
35	0.5491047501460525	0.5524209736862512
36	0.4211678199627536	0.5823521801140381
37	0.5460865067160211	0.3899830783903968
38	0.5464707874746474	0.2682566958494408
39	0.7750312609134309	0.3336931736750175
40	0.9650106501987584	0.3427934689786482
41	1.2842505373291711	0.3485356628872055
42	1.0780877461764713	0.4292545781824753
43	0.3901958935906435	0.4253930218774225
44	0.5467639499311863	0.4670706147290256 7
45	0.4878777134290887	0.2500369869716094
46	0.5417036510419296	0.4630476294216933
47	0.5393870492574223	0.4728426780699112
48	0.5617958219230703	0.3936755846352994
49	0.7384557501882417	0.4311618006554760 5
50	0.900779071479527	0.3160556143688 73

for k = 11

	Average F-score	Average Conditional Entropy
1	0.7377160972668363	0.1772252622374970 7
2	0.7509542761666452	0.5372721099362747
3	1.0796597294560075	0.3441483921998829
4	0.7076337683860123	0.3249889731706737
5	0.7537037208242909	0.1344219760014812 3
6	0.6705089586612444	0.5586098952869102
7	0.8158571558031769	0.4545367999845342
8	0.8759306352540293	0.5275010806226037
9	0.8416158145015683	-0.4112611530043973
10	0.7248228499775757	0.4022422964104565
11	1.1301154260932553	0.4534539638494688
12	0.8994777597618633	0.3960048849219562 4
13	1.032830554229373	0.3766022263139665
14	0.9701436718039087	0.2035340256616812 7
15	1.4089063076746004	0.3859481446575254
16	0.6684999742058237	0.3726014622815319
17	0.7561514824383027	0.5364732999974124
18	0.7194490831205181	0.4354568671735384
19	1.1732330970692 51	0.3278842462092787
20	0.6964961571492476	0.5560242940651703
21	1.1293428571550959	0.5574321603103769
22	1.1096626668863	0.4605870023970878 3
23	1.0375681409532833	0.2728030258804951 5
24	0.5429426107813723	0.5567765550052451
25	0.7872205374311423	0.3262577707831027
26	0.5578548124867055	0.4180875394861 1066
27	0.8390431039074955	0.4512241247835736
28	0.5804976274488949	0.4790725265709093
29	0.4324774359945453	0.2220003449283011
30	0.5768424368307943	0.4262012671364 1354
31	0.4588503931280582	0.4208539979266687
32	0.3575691875195466	0.2825233083919203
33	0.8968681648184175	0.3734060183063954 7
34	0.8256079732246151	0.3111213866837040 4
35	0.5373016063755457	0.4282731350669865
36	0.4708214161902937	0.5970920661896334
37	0.6755981385882001	0.4677104324952781
38	0.611044956906005	0.1978779265335673 7
39	0.8224750719992977	0.4869220042810714
40	0.9732772394664199	0.2076671348105232
41	1.2864286159303338	0.4060231352455773
42	1.1432360072735 37	0.4131199605041457
43	0.4201527921209154	0.4248922689388243
44	0.6319034907752958	0.5180654504004026
45	0.4703715825693614	0.0868169194190498
46	0.5792715056208941	0.5461594970608072
47	0.5549955177784582	0.4630012472410 2776
48	0.5551161682630175	0.4545163612639921 4
49	0.8238714923052565	0.4460150738094896
50	0.9491323913618647	0.3316459779006497 6

In [14]:
```
segmented_imgs, flat_segmented_imgs = image_segmentation(k=5)
for img_index in range(5):
    fig, axs = plt.subplots(1, 6, figsize=(100, 20))
    axs[0].imshow(segmented_imgs[img_index] + 1)
    axs[0].set_title("Segmentation")

    for gt_img_index in range(len(groundTruth[img_index])):
        _, _,_new_gt = evaluate_segmentation(img_index, gt_img_index, flat_segme
        axs[gt_img_index + 1].imshow(_new_gt)
```



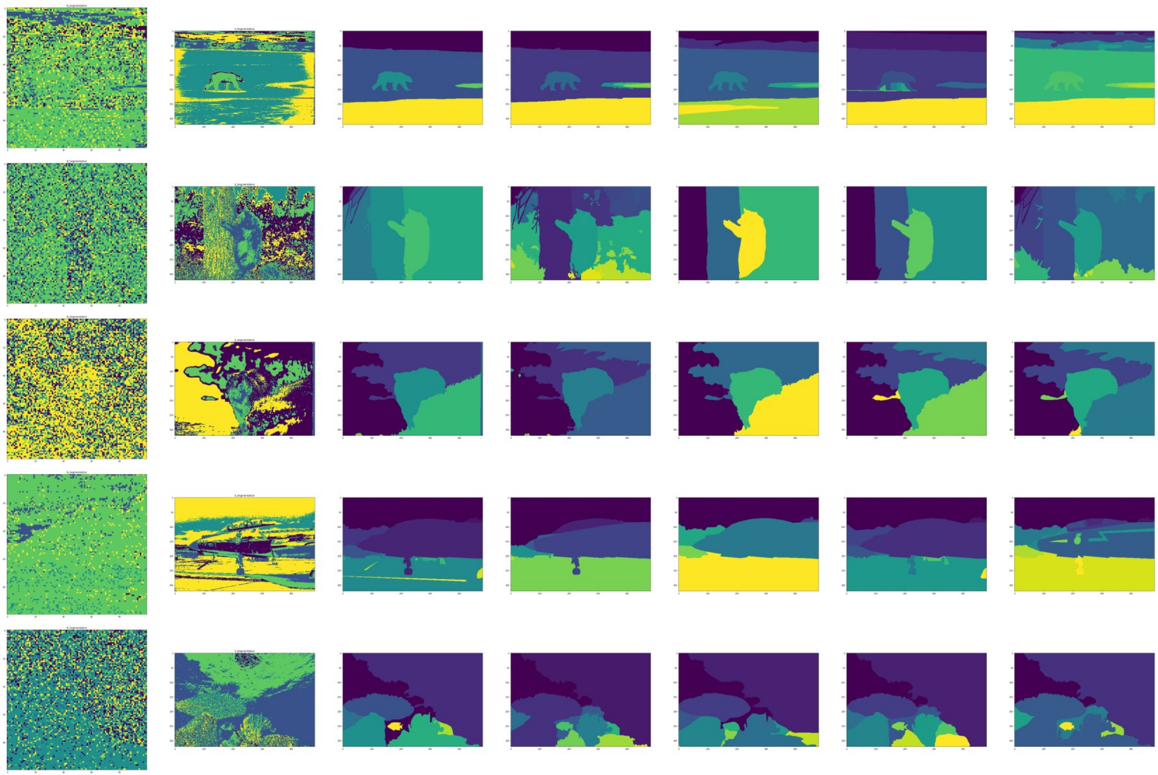
In [6]:
```
# resize and get distance matrix
dist_mat=[]
for i in range(5):
    resized_image = cv2.resize(images[i], (100, 100))
    flattened_image = resized_image.reshape(resized_image.shape[0] *resized_imag
    dist_mat.append(distance_matrix(flattened_image,flattened_image,p=2))
dist_mat=np.array(dist_mat)
```

In [7]:
```
def Ncut(dist_mat, k, knn):

  #lab. mat
  A = kneighbors_graph(dist_mat , knn , mode='connectivity', include_self=False)
  D = np.diag(np.sum(A, axis=1))
  L = D - A

  #eigen
  eigen_values, eigen_vectors = np.linalg.eig(np.dot(np.linalg.inv(D),L))

  idx = eigen_values.argsort()[::1]
  eigen_values = eigen_values[idx]
  eigen_vectors = eigen_vectors[:,idx]

  Y = np.zeros(eigen_vectors.shape)
  for i in range(eigen_vectors.shape[0]):
    norm=np.linalg.norm(eigen_vectors[i])
    Y[i]=eigen_vectors[i]/norm

  km = KMeans(n_clusters=k).fit(Y)
  return km.labels_
```

In [8]:
```
n_segmented=[]
for i in range(5):
    n_segmented.append(Ncut(dist_mat[i],k=5,knn=5))
n_segmented=np.array(n_segmented)
```

```
C:\Users\smart\sheet5\venv\lib\site-packages\ipykernel_launcher.py:17: ComplexWa
rning: Casting complex values to real discards the imaginary part
  app.launch_new_instance()
C:\Users\smart\sheet5\venv\lib\site-packages\ipykernel_launcher.py:17: ComplexWa
rning: Casting complex values to real discards the imaginary part
  app.launch_new_instance()
C:\Users\smart\sheet5\venv\lib\site-packages\ipykernel_launcher.py:17: ComplexWa
rning: Casting complex values to real discards the imaginary part
  app.launch_new_instance()
C:\Users\smart\sheet5\venv\lib\site-packages\ipykernel_launcher.py:17: ComplexWa
rning: Casting complex values to real discards the imaginary part
  app.launch_new_instance()
C:\Users\smart\sheet5\venv\lib\site-packages\ipykernel_launcher.py:17: ComplexWa
rning: Casting complex values to real discards the imaginary part
  app.launch_new_instance()
```

In [15]:
```
for img_index in range(5):
    fig, axs = plt.subplots(1, 7, figsize=(100, 20))
    axs[0].imshow(n_segmented[img_index].reshape((100,100)))
    axs[0].set_title("N_Segmentation")
    axs[1].imshow(segmented_imgs[img_index])
    axs[1].set_title("k_Segmentation")
    for gt_img_index in range(len(groundTruth[i])):
        axs[gt_img_index + 2].imshow(groundTruth[img_index][gt_img_index])
```



In [18]:
```
images=None
segmented_imgs=None
groundTruth=None
groundTruthPerImage=None
```

In [ ]: