Measuring SAVOIAS Complexity

In this notebook, I will calculate number of regions and feature congestion for images in the Scenes, Objects, and Interiors categories of the SAVOIAS dataset as proxies for image complexity. I will then compute the Pearson correlation coefficients between number of regions and feature congestion, and the human-annotated visual complexity scores provided by SAVOIAS for each image.

Citation: Saraee, E., Jalal, M., & Betke, M. (2018). SAVOIAS: A Diverse, Multi-Category Visual Complexity Dataset (arXiv:1810.01771). arXiv. http://arxiv.org/abs/1810.01771

1. Preliminaries

In [11]:
```
import collections
import cv2
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import os
import pickle
import pymeanshift as pms
import scipy.stats
import time
import urllib
```

In [4]:
```
cv2.getVersionString()
```

Out[4]:
```
'4.5.5'
```

In [5]:
```
## Download the SAVOIAS dataset
# wget https://github.com/esaraee/Savoias-Dataset
```

2. Using Mean-Shift for Image Segmentation

Mean-shift and its application to image segmentation is described in:

Comaniciu, D., & Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(5), 603–619. https://doi.org/10.1109/34.1000236

Mean-shift segmentation works by identifying regions of points that are close in terms of (x,y) coordinates and in terms of colorspace. We draw circular windows (kernels) over the image and iteratively move the center of the window toward the center of gravity of the points following the mean-shift vector. Finally, merge windows with close centers to get the finalized regions. These regions can then be filled with a single color and different colored regions can be counted. The process is ultimately controlled by the choice of bandwidth (how large/small to make the windows) and the merging criteria (e.g., minimum # of pixels per region).

In [8]:
```
# Load and display single image from SAVOIAS scenes
img = Image.open("SAVOIAS/Images/Scenes/0.jpg")

def show_image(img, grayscale=False):
    if grayscale:
        plt.imshow(img, cmap='gray')
    else:
        plt.imshow(img)
    plt.axis('off')
    plt.show()

show_image(img)
print(img.size, img.mode)
```

```
(768, 512) RGB
```

We will use PyMeanShift: https://github.com/fjean/pymeanshift

2.1 Testing segment() on SAVOIAS Images

First, let's try out the segment() function on the image loaded above with several different values for min_density, spatial, and range radius:

In [9]:
```
# what does min_density control?
(segmented_image, labels_image, number_regions) = pms.segment(img, spatial_radius=2
                                                              range_radius=40, min_
```

In [10]:
```
print("Segmented image")
show_image(segmented_image)
print("Labels")
show_image(labels_image)
print("Number of regions:", number_regions)
```

```
Segmented image
```

```
Labels
```

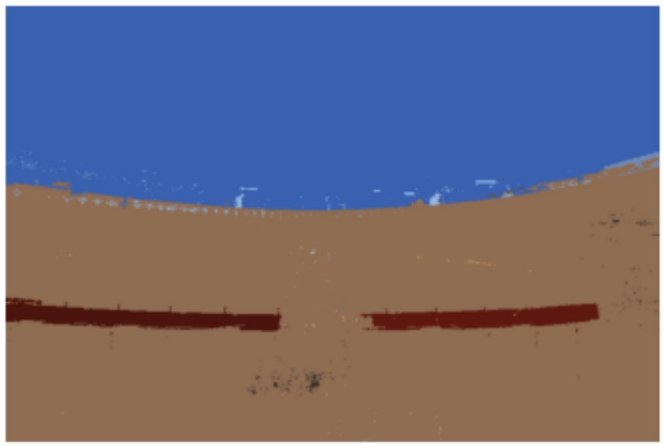
```
Number of regions: 7
```

In [14]:
```
start = time.perf_counter()
(segmented_image, labels_image, number_regions) = pms.segment(img, spatial_radius=2
                                                              range_radius=40, min_
stop = time.perf_counter()
print("Time to segment img: {0:.4f}".format(stop - start))
```

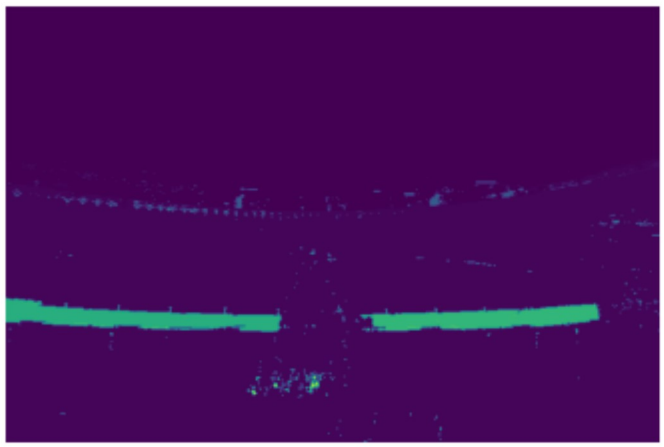
```
Time to segment img: 43.9228
```

In [15]:
```
print("Segmented image")
show_image(segmented_image)
print("Labels")
show_image(labels_image)
print("Number of regions:", number_regions)
```

```
Segmented image
```



```
Labels
```



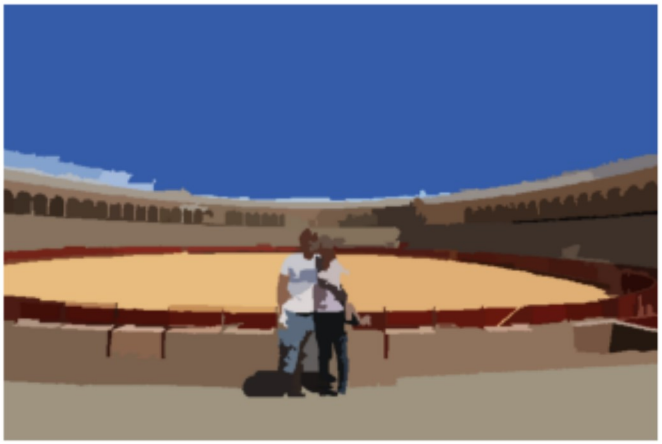
```
Number of regions: 652
```

In [30]:
```
start = time.perf_counter()
(segmented_image, labels_image, number_regions) = pms.segment(img, spatial_radius=2
                                                              range_radius=10, min_
stop = time.perf_counter()
print("Time to segment img: {0:.4f} s".format(stop - start))
```

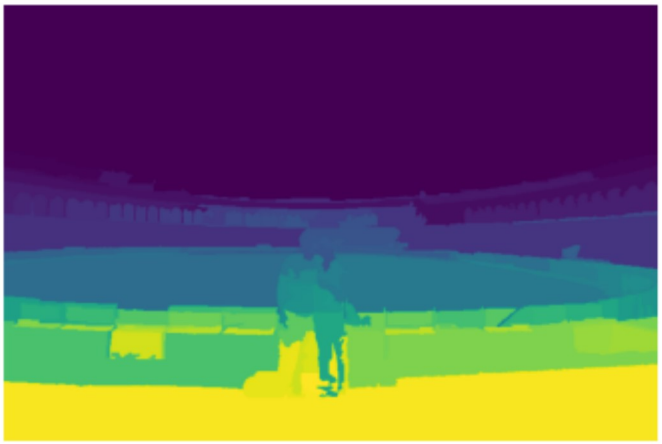
```
Time to segment img: 0.2794 s
```

In [31]:
```
print("Segmented image")
show_image(segmented_image)
print("Labels")
show_image(labels_image)
print("Number of regions:", number_regions)
```

```
Segmented image
```



```
Labels
```



```
Number of regions: 105
```

From the above it looks like min_density controls the number of pixels per region.
Investigating this further (see below), this seems to be true.

In [32]:
```
# investigating further
labels = list(labels_image.flatten())
label_counts = collections.Counter(labels)
print(label_counts)
```

```
Counter({0: 160421, 103: 53661, 74: 18333, 37: 18319, 84: 16427, 42: 14981, 16: 135
30, 15: 10384, 9: 5559, 4: 4807, 5: 4721, 62: 2727, 8: 2491, 7: 2188, 27: 2044, 10
0: 2009, 1: 1861, 97: 1683, 89: 1526, 47: 1434, 2: 1422, 88: 1415, 28: 1203, 33: 12
00, 73: 1142, 79: 1089, 78: 1072, 14: 1056, 76: 1054, 81: 1038, 20: 1030, 17: 1017,
44: 995, 71: 981, 30: 965, 11: 960, 38: 937, 59: 858, 18: 833, 80: 831, 65: 829, 6:
824, 53: 818, 82: 814, 66: 784, 46: 771, 69: 733, 41: 729, 39: 719, 29: 702, 25: 68
5, 22: 683, 68: 671, 98: 671, 96: 652, 51: 643, 34: 636, 26: 622, 43: 613, 54: 611,
3: 600, 104: 588, 13: 587, 45: 583, 77: 549, 36: 526, 50: 512, 57: 492, 24: 485, 6
1: 485, 21: 480, 35: 477, 70: 476, 49: 455, 32: 439, 72: 439, 93: 439, 19: 432, 87:
411, 12: 408, 40: 407, 55: 400, 99: 397, 101: 396, 86: 392, 56: 387, 23: 380, 67: 3
73, 52: 371, 85: 371, 60: 364, 83: 358, 95: 358, 102: 357, 94: 356, 91: 348, 48: 33
6, 90: 332, 10: 331, 92: 330, 31: 328, 64: 325, 58: 323, 63: 314, 75: 305})
```

2.2 Segment Images from Scenes, Objects, and Interiors Categories

segment() takes one image at a time, so we will need to loop through the dataset category images one-by-one.

In [37]:
```
def segment_images(img_dir : str, sp=2, sr=10, min_density=300) -> list:
    """
    Given a directory of images img_dir, segment each image and return a list of tu
    (img_name, segmented_image, label_image, number_regions) via pymeanshift.segmen

    Parameters:
    img_dir : str - The name of the directory containing images to segment.
    sp : int - The size of the spatial window for mean-shift filtering (default: 5)
    sr : int - The size of the colorspace window for mean-shift filtering (default:
    min_density : int - The minimum number of pixels per region (helps determine wh
    running mean-shift filtering).

    Returns:
    A list of (img_name, segmented_image, label_image, number_regions) tuples gener
    """
    start = time.perf_counter()
    print("Segmenting images from {} . . . ".format(img_dir), end="")

    segmented_imgs = []
    img_paths = os.listdir(img_dir)

    for i, img_path in enumerate(img_paths):
        if img_path[0] != '.': # avoid picking up hidden files
            img = Image.open(os.path.join(img_dir, img_path))
            (segmented_image, labels_image, number_regions) = pms.segment(img, sp,
            segmented_imgs.append((img_path, segmented_image, labels_image, number_

        if i == (len(img_paths)//2):
            print("Halfway there . . . ", end="")

    stop = time.perf_counter()
    print("Done. Total time to segment: {0:.4f} s".format(stop - start))

    return segmented_imgs
```

2.2.1 Segmenting Object Images

In [38]:
```
segmented_objects = segment_images(img_dir='SAVOIAS/Images/Objects')
```

```
Segmenting images from SAVOIAS/Images/Objects . . . Halfway there . . . Done. Total
time to segment: 102.9841 s
```

Since segmenting the images takes a while, it would be good to save the resulting segmented_objects Python list to a file so we can load it in even if we close the notebook.

In [39]:
```
# save the segmented_objects list
# https://wiki.python.org/moin/UsingPickle
pickle.dump(segmented_objects, open('segmented_objects.p', 'wb'), protocol=3)
```

In [40]:
```
# load segmented objects list from file
segmented_obj_from_pickle = pickle.load(open('segmented_objects.p', 'rb'))
```

In [41]:
```
# Take a look at an example image from the list
img_name, segmented, labels, regions = segmented_obj_from_pickle[0]
print("Image name:", img_name)
print("Number of regions: ", regions)
print("Original:")
original = PIL.Image.open(os.path.join('./SAVOIAS/Images/Objects', img_name))
show_image(original)
print("Segmented:")
show_image(segmented)
```

```
Image name: 0.jpg
Number of regions:  50
Original:
```

```
Segmented:
```

In [46]:
```
def get_sorted_regions(segmented_imgs : list) -> np.ndarray:
    """
    Sort list of segmented images and their metadata by filenames
    (files are numbered 0 - # of images in category).

    Parameters:
    segmented_imgs: a list in the format generated by segment_images() (see above).

    Returns:
    A numpy array region_counts of dimension n_images x 2, where the first column
    contains image numbers as provided in the SAVOIAS dataset, and the second colum
    provides number of regions as determined by mean-shift segmentation (see
    segment_images()).
    """

    sorted_segmented = sorted(segmented_imgs, key=lambda item: int(item[0][:-4]))
    region_counts = [[int(img_id[:-4]), regions] for img_id, _, _, regions in sorte

    return np.array(region_counts)
```

In [47]:
```
# get sorted segmented Object images
sorted_segmented_obj = get_sorted_regions(segmented_obj_from_pickle)
```

Now that we have the count of image regions for each Object image, we can correlate this measure of visual complexity with the SAVOIAS human-annotated complexity scores.

In [48]:
```
def get_groundtruth(filename : str) -> np.ndarray:
    """
    Return list of (image filename, complexity score) tuples sorted by filenames
    (0.jpg - [# of images in category].jpg).

    Parameters
    ----------
    filename : str
        Name of a .npy file from which to read groundtruth visual complexity
        scores.
        Dictionary read from .npy comes in format
        {image_index : [image_filename_as_png, image score]}
        Note that latest GitHub release of dataset stores files as JPEG, not
        PNG, so it is necessary to substitute the correct file extension in the
        image filename.

    Returns
    -------
    list
        ndarray of [image filename, complexity score] rows sorted by filenames.

    """
    scores = np.load(filename, allow_pickle=True).item()
    return np.array([[item[0], int(item[1][1])]
                     for item in sorted(scores.items(), key=lambda item: item[0])])
```

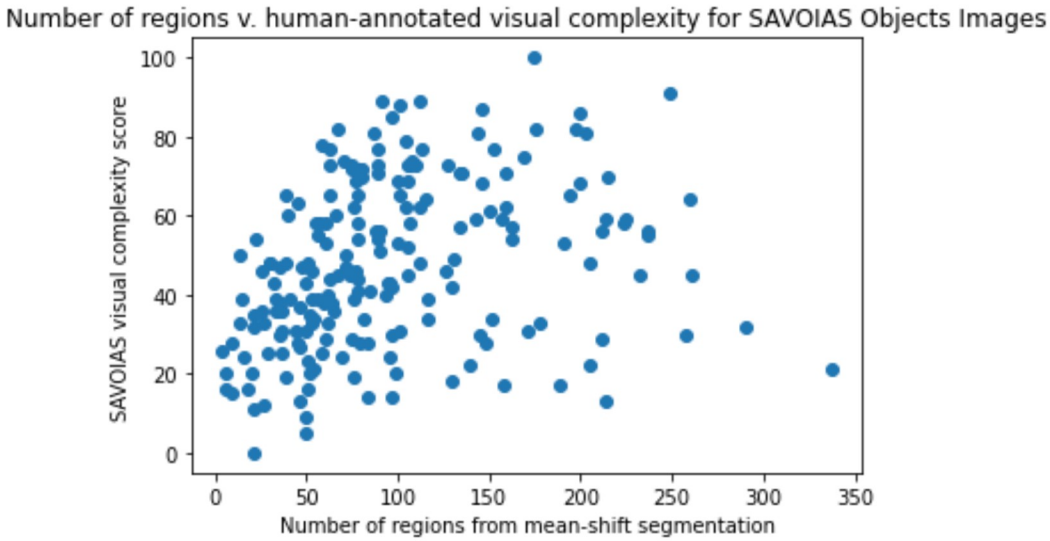
In [50]:
```
# get groundtruth complexity scores for Objects
obj_gt = get_groundtruth('SAVOIAS/ground_truth/npy/global_ranking_objects.npy')
# print(obj_gt)
```

In [65]:
```
region_counts = get_sorted_regions(segmented_objects)

r_obj_regions, p_obj_regions = scipy.stats.pearsonr(region_counts[:, 1], obj_gt[:,
print("Pearson's correlation r=", r_obj_regions)
print("p-value=", p_obj_regions)
```

```
Pearson's correlation r= 0.2972297768348763
p-value= 1.9190111702605754e-05
```

In [66]:
```
plt.scatter(region_counts[:, 1], obj_gt[:, 1])
plt.xlabel('Number of regions from mean-shift segmentation')
plt.ylabel('SAVOIAS visual complexity score')
plt.title('Number of regions v. human-annotated visual complexity for SAVOIAS Objec
plt.show()
```



2.2.2 Segmenting Scenes Images

In [55]:
```
segmented_scenes = segment_images(img_dir='SAVOIAS/Images/Scenes')
```

```
Segmenting images from SAVOIAS/Images/Scenes . . . Halfway there . . . Done. Total
time to segment: 157.8719 s
```

In [56]:
```
# save the segmented_scenes list
pickle.dump(segmented_scenes, open('segmented_scenes.p', 'wb'), protocol=3)
```

In [59]:
```
# Take a look at an example image
img_name, segmented, labels, regions = segmented_scenes[117]
print("Image name:", img_name)
print("Number of regions: ", regions)
print("Original:")
original = PIL.Image.open(os.path.join('./SAVOIAS/Images/Scenes', img_name))
show_image(original)
print("Segmented:")
show_image(segmented)
```

```
Image name: 24.jpg
Number of regions:  97
Original:
```

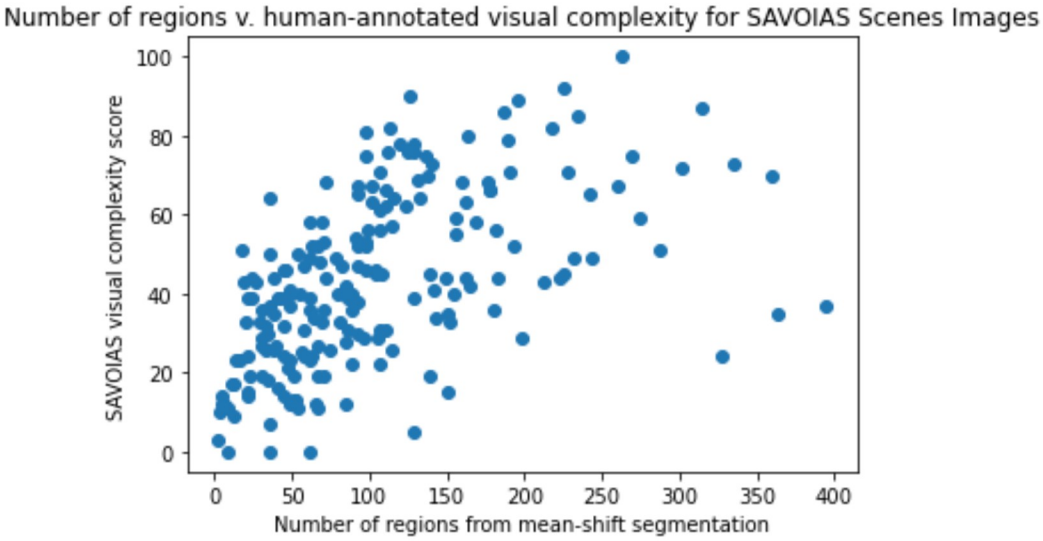
```
Segmented:
```

In [67]:
```
# correlate with visual complexity scores
scenes_regions = get_sorted_regions(segmented_scenes)
scenes_gt = get_groundtruth('/scratch/el55/SAVOIAS/ground_truth/npy/global_ranking_

r_scene_regions, p_scene_regions = scipy.stats.pearsonr(scenes_regions[:, 1], scene
print("Pearson's correlation r =", r_scene_regions)
print("p-value =", p_scene_regions)
```

```
Pearson's correlation r = 0.5440593532115849
p-value = 8.356987850272256e-17
```

In [68]:
```
plt.scatter(scenes_regions[:, 1], scenes_gt[:, 1])
plt.xlabel('Number of regions from mean-shift segmentation')
plt.ylabel('SAVOIAS visual complexity score')
plt.title('Number of regions v. human-annotated visual complexity for SAVOIAS Scene
plt.show()
```



2.2.3 Segmenting Interior Design Images

In [71]:
```
segmented_interiors = segment_images(img_dir='SAVOIAS/Images/Interior Design')
```

```
Segmenting images from SAVOIAS/Images/Interior Design . . . Halfway there . . . Don
e. Total time to segment: 87.9520 s
```

In [72]:
```
# save the segmented_interiors list
pickle.dump(segmented_interiors, open('segmented_interiors.p', 'wb'), protocol=3)
```

In [73]:
```
# Take a look at an example image
img_name, segmented, labels, regions = segmented_interiors[0]
print("Image name:", img_name)
print("Number of regions: ", regions)
print("Original:")
original = PIL.Image.open(os.path.join('./SAVOIAS/Images/Interior Design', img_name
show_image(original)
print("Segmented:")
show_image(segmented)
```

```
Image name: 0.jpg
Number of regions:  178
Original:
```

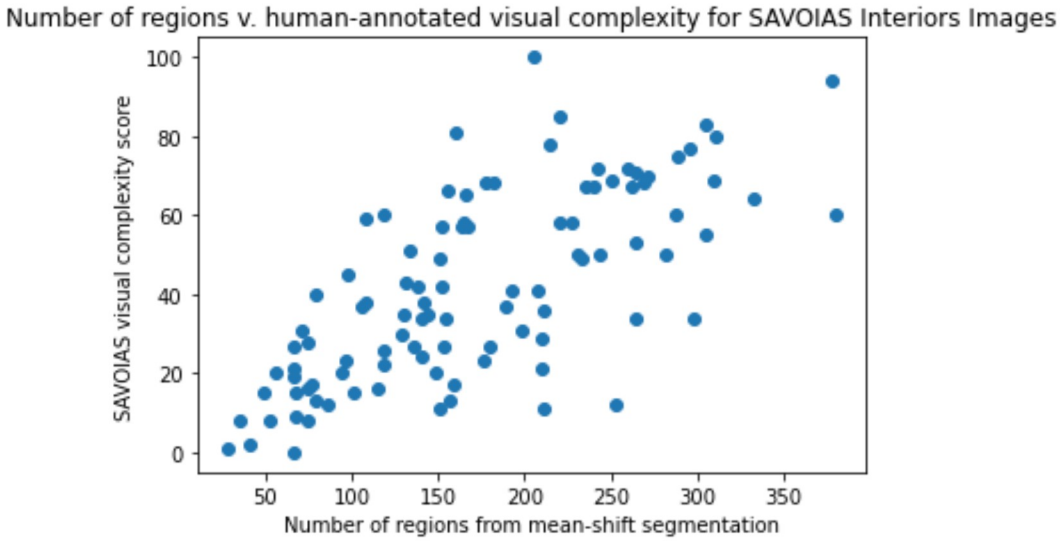
```
Segmented:
```

In [76]:
```
# correlate with visual complexity scores
interiors_regions = get_sorted_regions(segmented_interiors)
interiors_gt = get_groundtruth('/scratch/el55/SAVOIAS/ground_truth/npy/global_ranki

r_interiors_regions, p_interiors_regions = scipy.stats.pearsonr(interiors_regions[:
print("Pearson's correlation r =", r_interiors_regions)
print("p-value =", p_interiors_regions)

# plot # regions v. human-annotated scores
plt.scatter(interiors_regions[:, 1], interiors_gt[:, 1])
plt.xlabel('Number of regions from mean-shift segmentation')
plt.ylabel('SAVOIAS visual complexity score')
plt.title('Number of regions v. human-annotated visual complexity for SAVOIAS Inter
plt.show()
```

```
Pearson's correlation r = 0.699123306333484
p-value = 6.001949618571707e-16
```



In [ ]: