Homework_1_Superpixels

November 1, 2020

1 Assignment 1

by ### Saumil Shah ### Preyas Parikh

A superpixel can be defined as a group of pixels that share common characteristics. Simple Linear Iterative Clustering (SLIC) generates superpixels by clustering pixels based on their color similarity and proximity in the image plane. The purpose of this assignment is to understand and implement SLIC Superpixels.

Some pointers before we start: - Please follow all submission guidlines which are posted on piazza. - Ensure all outputs are displayed while rendering the PDF. - Only modify the code blocks which has a "TODO". - Below you can see some outputs for an image of a cow. These images represent the kind of output that is expected. - Feel free to reach out to any of the TAs for any doubts/issues.

Let's download the dataset first.

[ ]:
```
!wget http://download.microsoft.com/download/A/1/1/
↪A116CD80-5B79-407E-B5CE-3D5C6ED8B0D5/msrc_objcategimagedatabase_v1.zip
```

[ ]:
```
!unzip --qq msrc_objcategimagedatabase_v1.zip
```

We only focus on six images in this assignment.

[1]:
```
im_list = ['MSRC_ObjCategImageDatabase_v1/1_22_s.bmp',
           'MSRC_ObjCategImageDatabase_v1/1_27_s.bmp',
           'MSRC_ObjCategImageDatabase_v1/3_3_s.bmp',
           'MSRC_ObjCategImageDatabase_v1/3_6_s.bmp',
           'MSRC_ObjCategImageDatabase_v1/6_5_s.bmp',
           'MSRC_ObjCategImageDatabase_v1/7_19_s.bmp']
```

[70]:
```
#All important functions to plot, do not modify this block
%matplotlib inline
import cv2
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.patches as mpatches
from skimage import color
from skimage.transform import resize
import math
```

```python
def plot_image(im,title,xticks=[],yticks= [],isCv2 = True):
    """
    im :Image to plot
    title : Title of image
    xticks : List of tick values. Defaults to nothing
    yticks :List of tick values. Defaults to nothing
    cv2 :Is the image cv2 image? cv2 images are BGR instead of RGB. Default True
    """
    plt.figure()
    if isCv2:
        im = im[:,:,::-1]
    plt.imshow(im)
    plt.title(title)
    plt.xticks(xticks)
    plt.yticks(yticks)

def superpixel_plot(im,seg,title = "Superpixels"):
    """
    Given an image (nXmX3) and pixelwise class mat (nXm),
    1. Consider each class as a superpixel
    2. Calculate mean superpixel value for each class
    3. Replace the RGB value of each pixel in a class with the mean value

    Inputs:
    im: Input image
    seg: Segmentation map
    title: Title of the plot

    Output: None
    Creates a plot
    """
    clust = np.unique(seg)
    mapper_dict = {i: im[seg == i].mean(axis = 0)/255. for i in clust}

    seg_img =  np.zeros((seg.shape[0],seg.shape[1],3))
    for i in clust:
        seg_img[seg == i] = mapper_dict[i]

    plot_image(seg_img,title)

    return

def rgb_segment(seg,n = None,plot = True,title=None,legend = True,color = None):
    """
    Given a segmentation map, get the plot of the classes
    """
```

```
    clust = np.unique(seg)
    if n is None:
        n = len(clust)
    if color is None:
        cm = plt.cm.get_cmap('hsv',n+1)
        # mapper_dict = {i:np.array(cm(i/n)) for i in clust}
        mapper_dict = {i:np.random.rand(3,) for i in clust}
    #elif color == 'mean':
        #TODO..get the mean color of cluster center and assign that to␣
↪mapper_dict

    seg_img =  np.zeros((seg.shape[0],seg.shape[1],3))
    for i in clust:
        seg_img[seg == i] = mapper_dict[i][:3]

    if plot:
        plot_image(seg_img,title = title)
    if legend:
        # get the colors of the values, according to the
        # colormap used by imshow
        patches = [ mpatches.Patch(color=mapper_dict[i], label=" : {l}".
↪format(l=i) ) for i in range(n) ]
        # put those patched as legend-handles into the legend
        plt.legend(handles=patches, bbox_to_anchor=(1.05, 1), loc=2,␣
↪borderaxespad=0. )
        plt.grid(True)
        plt.show()

    return seg_img
```

Let's see what the six images are:

[5]:
```
for i in im_list:
    plot_image(cv2.imread(i),i.split("/")[-1])
```

1_22_s.bmp

1_27_s.bmp

3_3_s.bmp

3_6_s.bmp

6_5_s.bmp

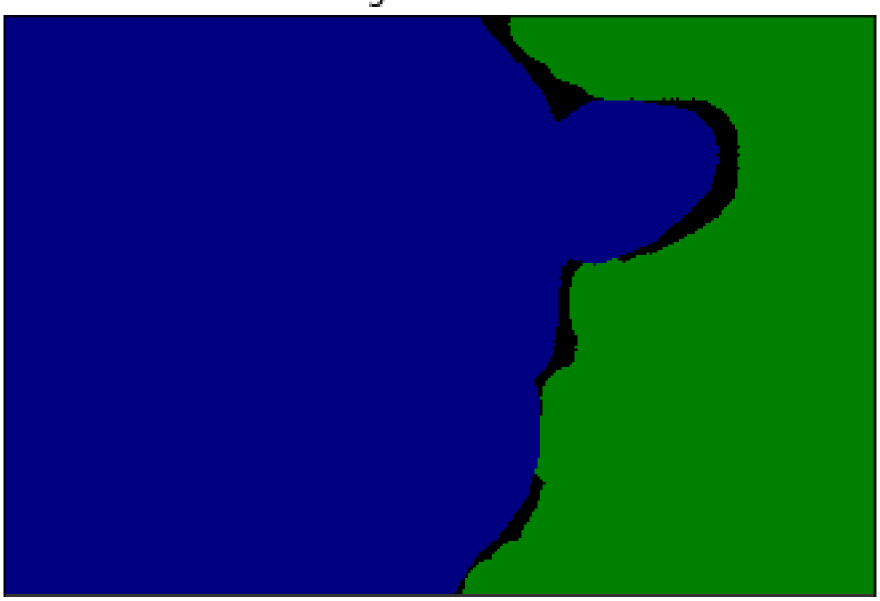
7_19_s.bmp

Get image and visualize it. Its a scenery with 3 elements. You can see the segmentation ground truth in the GT bitmap.

[28]:
```
im = cv2.imread(im_list[0])
seg = cv2.imread(im_list[0].replace("_s","_s_GT"))

plot_image(im,"Image")
plot_image(seg,"Segmentation")
```

Image

Segmentation



1.0.1 Question 1: K-means on RGB

We know k-means clustering algorithm. It is an unsupervised algorithm which minimizes **ANSWER:** __________?.

Complete the pixel clustering function. It should take input an image (dim = (n × m × 3)) and number of clusters needed. Does K means clustering work on image pixels? Let the number of clusters be K = 5, 10, 50

[16]:
```
from sklearn.cluster import KMeans
import numpy as  np
from sklearn.utils import shuffle
from sklearn.metrics import pairwise_distances_argmin

def createCentroids(k, range_):
    centroids = {
        i: np.array([np.random.randint(l) for l in range_])
        for i in range(k)
    }
    return centroids

def calculateDistance(data, centroid, lambda_1=None, lambda_2=None):
    if lambda_1 and lambda_2:
        dis_1 = np.sqrt(np.sum((data[:3] - centroid[:3])**2), axis=1)
        dis_2 = np.sqrt(np.sum((data[3:] - centroid[3:])**2), axis=1)
        dis = lambda_1*dis_1 + lambda_2*dis_2
    else:
        dis = np.sqrt(np.sum((data - centroid)**2, axis=1))
    return dis

def assignment(data, centroids, lambda_1=None, lambda_2=None):
    distances = []
    for i in centroids.keys():
        dis = calculateDistance(data, centroids[i], lambda_1=None,␣
↪lambda_2=None)
#         m = np.sqrt(np.sum((data - centroids[i])**2, axis=1))
#         distances.append(m)
        distances.append(dis)
    distances = np.array(distances)
    nearest_clusters = np.argmin(distances.T, axis=1)
    return nearest_clusters

def update(k, centroids, nearest_clusters, data):
    for i in centroids.keys():
        temp =  data[nearest_clusters==i]
```

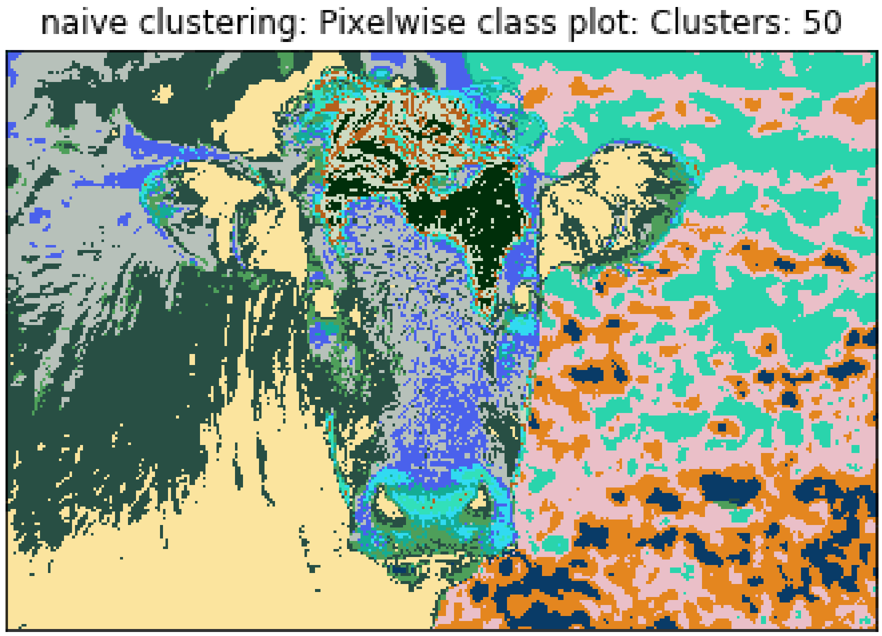
```
        if temp.shape[0] > 0:
            centroids[i] = temp.mean(axis=0) # mean along axis 0
    return centroids

def getRGBdata(img):
    h,w,_ = img.shape
    data = img.reshape([h*w, 3])
    return data

def getRGBXYdata(img):
    h,w,d = img.shape
    x,y = np.meshgrid(np.arange(0,w), np.arange(0,h))
    data = np.hstack([np.reshape(img, (w * h, d)), np.reshape(y, (w*h, 1)), np.
↪reshape(x, (w*h, 1))])
    return data

def cluster_pixels(im, k):
    h,w,_ = im.shape
    data = getRGBdata(im)
    centroids = createCentroids(k, [255, 255, 255])
    for i in range(10):
        near = assignment(data, centroids)
        centroids = update(k, centroids, near, data)
    near = assignment(data, centroids)
    segmap = near.reshape([h,w])
    return segmap

for k in [5,10,50]:
    clusters = cluster_pixels(im,k)
    _ = rgb_segment(clusters,n = k, title =  "naive clustering: Pixelwise class␣
↪plot: Clusters: " + str(k),legend = False)
    superpixel_plot(im,clusters,title =  "naive clustering: Superpixel plot:␣
↪Clusters: "+ str(k))
```

naive clustering: Pixelwise class plot: Clusters: 5

naive clustering: Superpixel plot: Clusters: 5

naive clustering: Pixelwise class plot: Clusters: 10

naive clustering: Superpixel plot: Clusters: 10

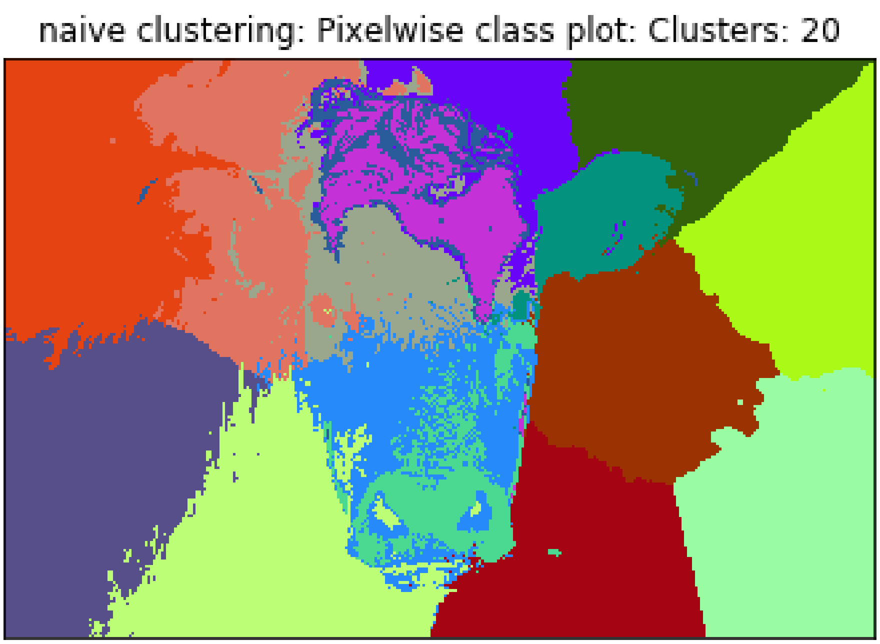

naive clustering: Pixelwise class plot: Clusters: 50


naive clustering: Superpixel plot: Clusters: 50

1.0.2 Question 2: Now that you have a function handy, we have a slightly complex task

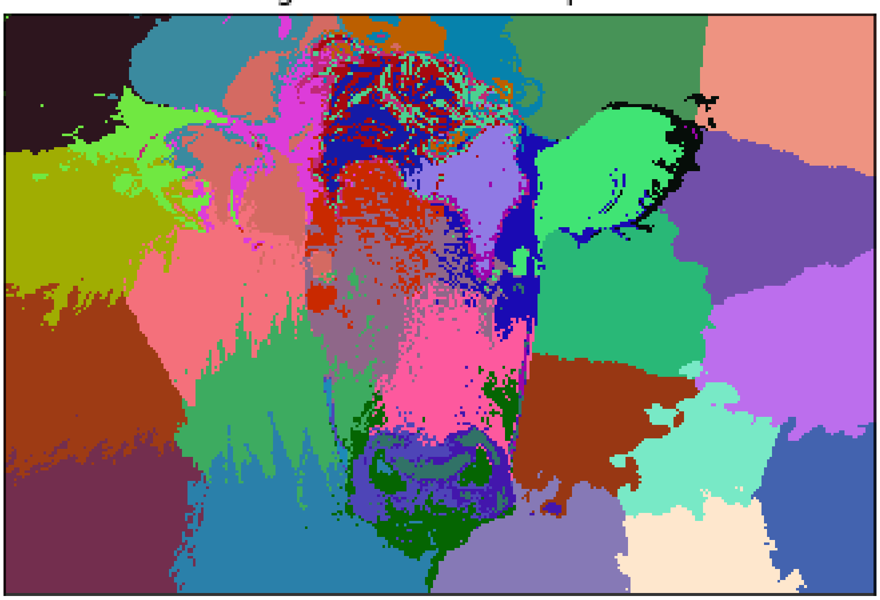
Instead of making clustering run on RGB space, run the clustering on RGBXY space. What advantages does that give us? (try with clusters = 5, 10, 25, 50, 150)

[17]:
```
#TODO: clustering r,b,g,x,y values
#try k = 20,80,200,400,800

def cluster_rgbxy(im,k):
    h,w,_ = im.shape
    data = getRGBXYdata(im)
    centroids = createCentroids(k, [255, 255, 255, h, w])
    for i in range(10):
        near = assignment(data, centroids)
        centroids = update(k, centroids, near, data)
    near = assignment(data, centroids)
    segmap = near.reshape([h,w])
    return segmap

for k in [20,80,200,400,800]:
    clusters = cluster_rgbxy(im,k)
    _ = rgb_segment(clusters,n = k, title =  "naive clustering: Pixelwise class␣
↪plot: Clusters: " + str(k),legend = False)
    superpixel_plot(im,clusters,title =  "naive clustering: Superpixel plot:␣
↪Clusters: "+ str(k))
```



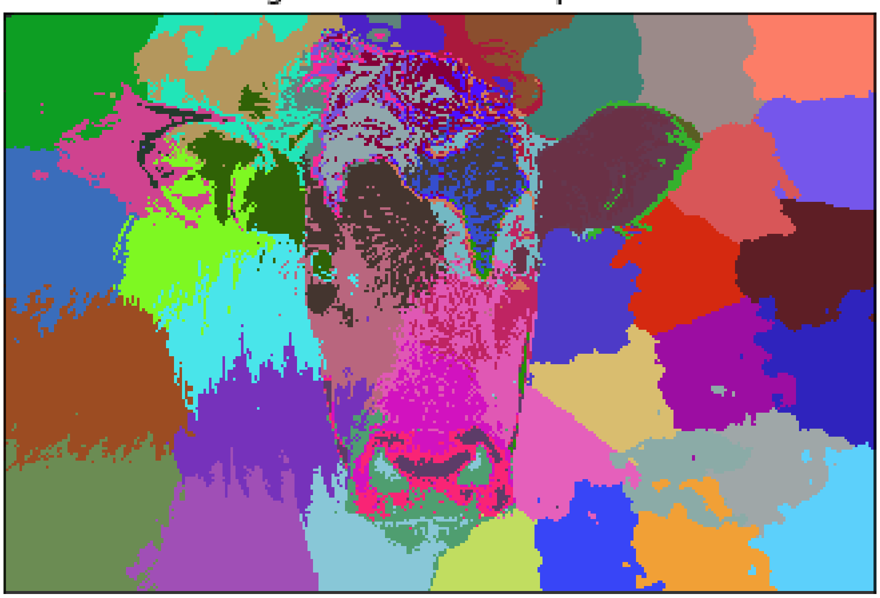
naive clustering: Superpixel plot: Clusters: 20

naive clustering: Pixelwise class plot: Clusters: 80



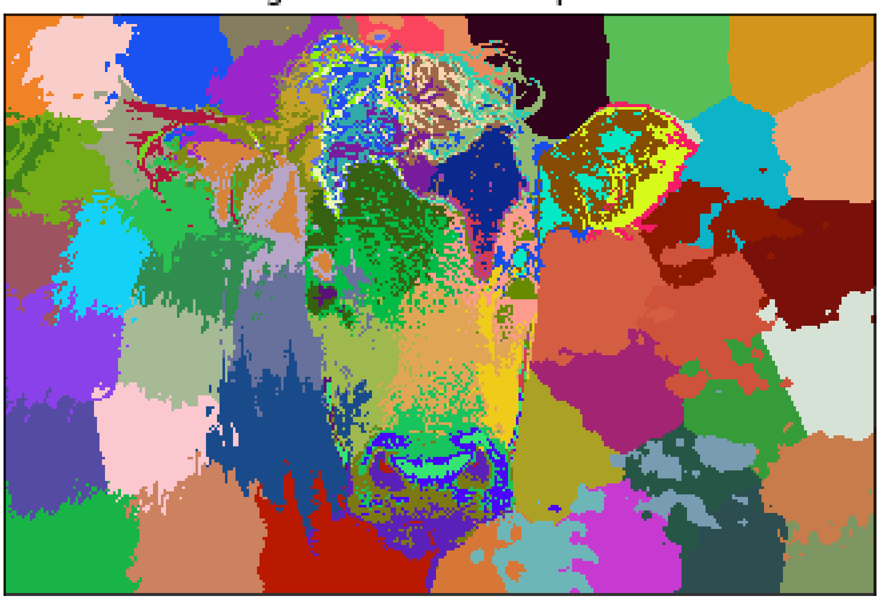
naive clustering: Superpixel plot: Clusters: 80

naive clustering: Pixelwise class plot: Clusters: 200



naive clustering: Superpixel plot: Clusters: 200

naive clustering: Pixelwise class plot: Clusters: 400



naive clustering: Superpixel plot: Clusters: 400

naive clustering: Pixelwise class plot: Clusters: 800

1.0.3 Modified k-means with weighted distances.

Let $cluster\_center_i$ represent i^{th} cluster center, $cluster\_center_i^{rgb}$ denote the RGB value and $cluster\_center_i^{xy}$ be the corresponding coordinate of the center pixel, respectively.

Let x_{rgb} be the the RGB value of a pixel, and let x_{xy} be the corresponding pixel's coordinate.

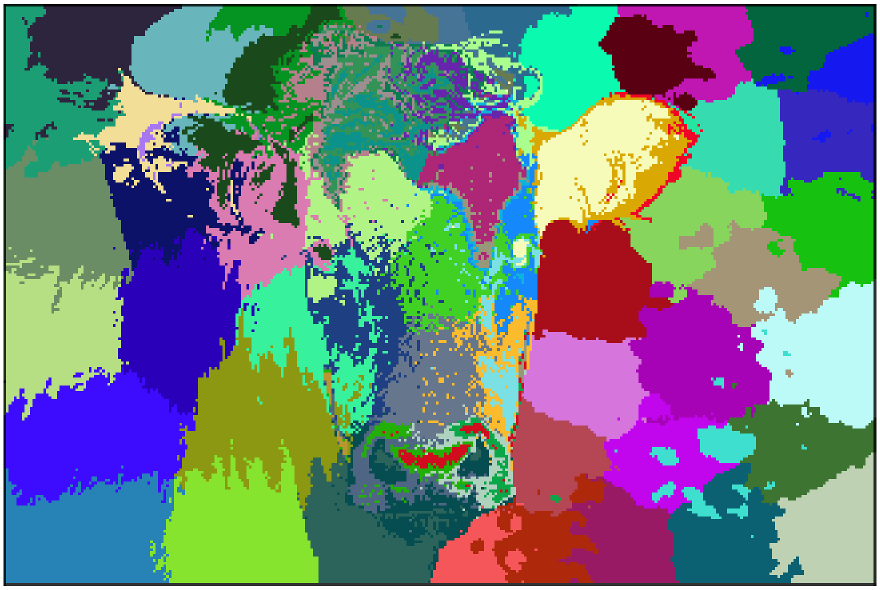
$distance(x_{rgb}, x_{xy}) = \lambda_1 * euclidean(x_{rgb}, cluster\_center_i^{rgb}) + \lambda_2 * euclidean(x_{xy}, cluster\_center_i^{xy})$

Find good values for hyperparmeters λ_1 and λ_2 (try on 250 clusters)

[24]:
```
#TODO: clustering r,b,g,x,y values with lambdas and display outputs
def cluster_rgbxy(im,k, lambda_1, lambda_2):
    h,w,_ = im.shape
    data = getRGBXYdata(im)
    centroids = createCentroids(k, [255, 255, 255, h, w])
    for i in range(10):
        near = assignment(data, centroids, lambda_1, lambda_2)
        centroids = update(k, centroids, near, data)
    near = assignment(data, centroids, lambda_1, lambda_2)
    segmap = near.reshape([h,w])
    return segmap

for k in [250]:
    clusters = cluster_rgbxy(im, k, 1.2, 3.2)
```

```
_ = rgb_segment(clusters,n = k, title =  "naive clustering: Pixelwise class␣
↪plot: Clusters: " + str(k),legend = False)
superpixel_plot(im,clusters,title =  "naive clustering: Superpixel plot:␣
↪Clusters: "+ str(k))
```

naive clustering: Pixelwise class plot: Clusters: 250



naive clustering: Superpixel plot: Clusters: 250

1.0.4 Question 3: SLIC

It doesn't look like we have a very favourable outcome with superpixels simply being implemented as K-means. Can we do better? Have a look at the SLIC paper here. Incorporate S and m and redefine your distance metric as per the paper.

[84]:
```python
class SuperPixels(object):
        def __init__(self, h, w, lab):
                self.update(h, w, lab[0], lab[1], lab[2])
                self.pixels = []

        def update(self, h, w, l, a, b):
                self.h = h
                self.w = w
                self.l = l
                self.a = a
                self.b = b

def create_clusters(S, img, clusters):
        H,W,_ = img.shape
        h = int(S/2)
        w = int(S/2)
        while h < H:
                while w < W:
                        clusters.append(SuperPixels(h, w, img[h,w]))
                        w += S
                w = int(S/2)
                h += S
        return clusters

def gradient(h, w, img):
        H,W,_ = img.shape
        if w + 1 >= W:
                w = img_w - 2
        if h + 1 >= H:
                h = img_h - 2
        diagonal_pixel = img[w+1, h+1]
        gradient = (diagonal_pixel - img[w,h]).sum()
        return gradient

def reassign(clusters,img):
        for c in clusters:
                cluster_gradient = gradient(c.h, c.w, img)
                for dh in range(-1, 2):
                        for dw in range(-1, 2):
                                H = c.h + dh
                                W = c.w + dw
```

```
                                new_gradient = gradient(H, W, img)
                                if new_gradient < cluster_gradient:
                                        c.update(H, W,img[H,W][0],␣
↪img[H,W][1],img[H,W][2])
                                        cluster_gradient = new_gradient

def assign(clusters, S, img, cluster_map, distances):
        height, width, _ = img.shape
        for c in clusters:
                for h in range(c.h - 2 * S, c.h + 2 * S):
                        if h < 0 or h >= height: continue
                        for w in range(c.w - 2 * S, c.w + 2 * S):
                                if w < 0 or w >= width: continue
                                D = find_distance(img[h,w], np.array([h,␣
↪w],dtype=np.float), np.array([c.h, c.w]), np.array([c.l, c.a, c.b]), m, S)
                                if D < distances[h,w]:
                                        if (h, w) not in cluster_map:
                                                cluster_map[(h, w)] = c
                                                c.pixels.append((h, w))
                                        else:
                                                cluster_map[(h, w)].pixels.
↪remove((h, w))
                                                cluster_map[(h, w)] = c
                                                c.pixels.append((h, w))
                                        distances[h, w] = D

def find_distance(lab, pixel, cluster_center, cluster_lab, m, s):
        dc = np.sqrt(np.sum((lab - cluster_lab)**2))
        ds = np.sqrt(np.sum((pixel - cluster_center)**2))
        d = np.sqrt((dc/m)**2 + (ds/s)**2)
        return d

def update_center(clusters, img):
        for c in clusters:
                new_center = list(np.mean(c.pixels, axis=0).astype(np.int))
                h, w = new_center[0], new_center[1]
                c.update(h, w, img[h,w][0], img[h,w][1], img[h,w][2])

def cluster_color(img, clusters):
        image = np.copy(img)
        for c in clusters:
                for p in c.pixels:
                        image[p[0],p[1]][0] = c.l
                        image[p[0],p[1]][1] = c.a
                        image[p[0],p[1]][2] = c.b
        return lab2rgb(image)
```

```
def slic(S, img, clusters, cluster_map, distances):
        img_h, img_w,_ = img.shape
        clusters = create_clusters(S, img, clusters)
        reassign(clusters, img)
        for i in range(10):
                assign(clusters, S, img, cluster_map, distances)
                update_center(clusters, img)
        seg_image = cluster_color(img, clusters)
        return clusters, seg_image

def rgbTOlab(img):
        return color.rgb2lab(img)

def lab2rgb(lab):
        return color.lab2rgb(lab)

def get_cluster_image(img, clusters):
    cluster_image = np.ones(img.shape[:2])
    for i in range(len(clusters)):
        for pixel in clusters[i].pixels:
            cluster_image[pixel[0], pixel[1]] = i
    return cluster_image
```

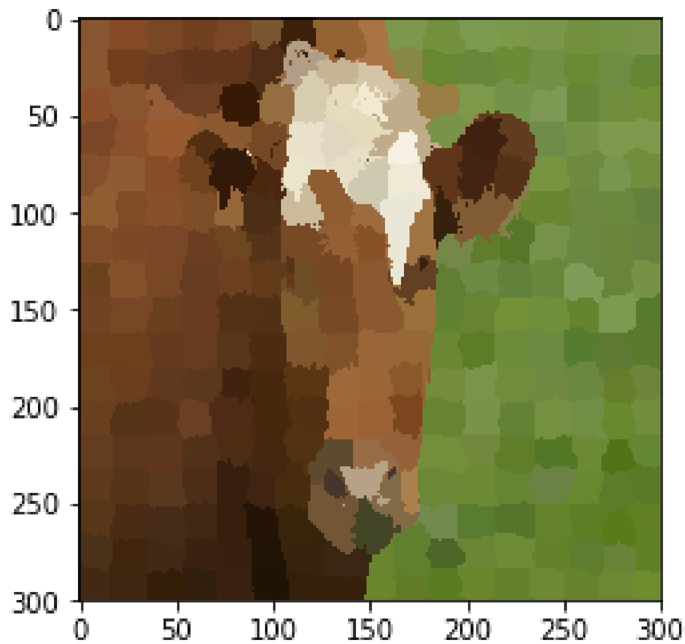
[74]:
```
#TODO
###########Algorithm###########
#Compute grid steps: S
#you can explore different values of m
#initialize cluster centers [l,a,b,x,y] using
#Perturb for minimum G
#while not converged
##for every pixel:
####  compare distance D_s with each cluster center within 2S X 2S.
####  Assign to nearest cluster
##calculate new cluster center


def SLIC(img, k):
    """
    Input arguments:
    im: image input
    k: number of cluster segments


    Compute
    S: As described in the paper
    m: As described in the paper (use the same value as in the paper)
    follow the algorithm..
```

```
    returns:
    segmap: 2D matrix where each value corresponds to the image pixel's cluster␣
↪number
    """

    img_h = img.shape[0]
    img_w = img.shape[1]
    N = img_h * img_w
    S = int(math.sqrt(N /k))
    m = 20

    clusters = []
    cluster_map = {}
    distances = np.full((img_h, img_w), np.inf)

    clusters = create_clusters(S, img, clusters)
    reassign(clusters, img)
    for i in range(10):
        assign(clusters, S, img, cluster_map, distances)
        update_center(clusters, img)
    seg_image = cluster_color(img, clusters)
    return clusters, seg_image
```

[95]:
```
k = 250
rgb = cv2.imread(im_list[0])
rgb = rgb[:,:,::-1]
img = resize(rgb, (300,300),anti_aliasing=True)
img = color.rgb2lab(img)

clusters, final = SLIC(img, k)
plt.imshow(final)
```

```
/usr/local/lib/python3.6/dist-packages/skimage/transform/_warps.py:105:
UserWarning: The default mode, 'constant', will be changed to 'reflect' in
skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "
```

[95]: <matplotlib.image.AxesImage at 0x7fede2d3c2e8>



[87]:
```
cluster_image = get_cluster_image(img, clusters)
```

1.1 Bonus Question:

Enforce connectivity: There are many superpixels which are very small and disconnected from each other. Try to merge them with larger superpixels

O(N) algorithm: 1. Set a minimum size of superpixel 2. If the area of a region is smaller than a threshold, we assign it to the nearest cluster

[ ]:
```
#TODO
```

1.2 Your File

Link to your colab/ipynb file: **Insert google drive/colab link here** https://colab.research.google.com/drive/106RdqX1sWHmXBKziesKL96_TzvXaA9jc#scrollTo=frzcU2Brd5Gh

[ ]:
```
```