MLCV Fall 2023 - Homework 2

Guoyu Lu - guoyu.lu@uga.edu

Due: See elc

Instructions

Your homework submission must cite any references used (including articles, books, code, websites, and personal communications). All solutions must be written in your own words, and you must program the algorithms yourself. Submit your solutions as a PDF to the Folder on elc.

Your homework solution must be prepared in LaTeXand output to PDF format. I suggest using `http://overleaf.com` or BaKoMa TeXto create your document.

Your programs must be written in Python. The relevant code to the problem should be in the PDF you turn in. If a problem involves programming, then the code should be shown as part of the solution to that problem. One easy way to do this in LaTeXis to use the verbatim environment, i.e., \begin{verbatim} YOUR CODE \end{verbatim}

If told to implement an algorithm, don't just call a toolbox.

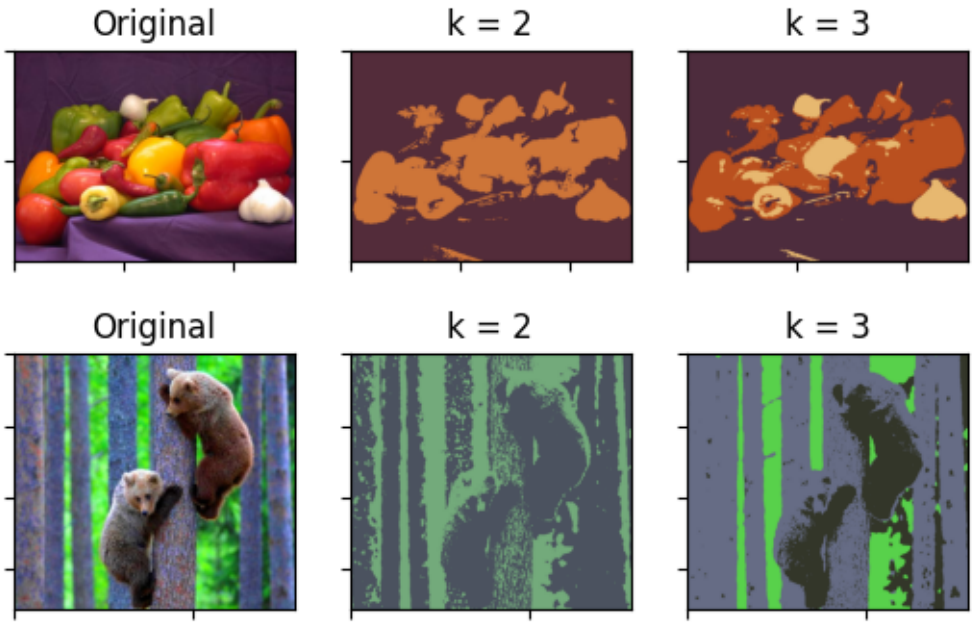
Part 1 - Segmentation with k-Means (10 points)

Write a function `seg = kMeansSegment(img, k, colorSpace)` that takes as input a $n \times m \times 3$ images `img`, the $k > 1$ for use in k-Means, and the `colorSpace` (RGB, grayscale, etc). The function will output a $n \times m$ integer image containing the assignment of each pixel to each cluster.

Now, put all of your pieces of code together to segment the images `peppers.jpg` and `bears.jpg`. For each image, you must visualize two different interesting parameter settings, with one setting chosen to optimize the segmentation into semantically reasonable segments. Make sure to label the images with the parameters chosen, and display each as a 1×3 labeled figure with the segmentation.

You may use smaller images when debugging your system to make it faster, and if necessary you can use them throughout. Just explain what you are doing.

Solution:




```
#https://github.com/majdjamal/image_segmentation/tree/main
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
from scipy.spatial import distance_matrix
import cv2
```

```
def kMeansSegment(img, k, colorSpace):
  image = cv2.imread(img)
  if colorSpace == 'RGB':
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) #Convert to RGB
  elif colorSpace == 'grayscale':
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  X = image.reshape((-1,3))   #Reshape to (Npts, Ndim = 3)
  X = np.float32(X)
  Npts, Ndim = X.shape
  max_iter = 60
  X_max, X_min = np.max(X), np.min(X)
  centroids = np.random.uniform(low = X_min, high=X_max, size = (k,Ndim))
  for i in range(max_iter):
    diff = cdist(X, centroids, metric="euclidean")
    clusters = np.argmin(diff, axis=1)
    for i in range(k):
      centroids[i] = np.mean(X[np.where(clusters == i)] , axis=0)
  centers = centroids
  clusters = clusters
  segmented_image = centers[clusters]
  segmented_image = segmented_image.reshape((image.shape))
  return segmented_image

k=2
plt.figure(figsize=(8., 8.))
f, axarr = plt.subplots(1,3)
img = cv2.imread('peppers.jpg')
image = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
axarr[0].imshow((image).astype(np.uint8))
axarr[0].set_title('Original')
axarr[0].tick_params(labelleft=False, labelbottom=False, labelright=False, labeltop=False)
seg = kMeansSegment('peppers.jpg', k, 'RGB')
axarr[1].imshow((seg).astype(np.uint8))
axarr[1].set_title('k = ' + str(k))
axarr[1].tick_params(labelleft=False, labelbottom=False, labelright=False, labeltop=False)
k=3
seg = kMeansSegment('peppers.jpg', k, 'RGB')
axarr[2].imshow((seg).astype(np.uint8))
axarr[2].set_title('k = ' + str(k))
axarr[2].tick_params(labelleft=False, labelbottom=False, labelright=False, labeltop=False)
plt.show()
```

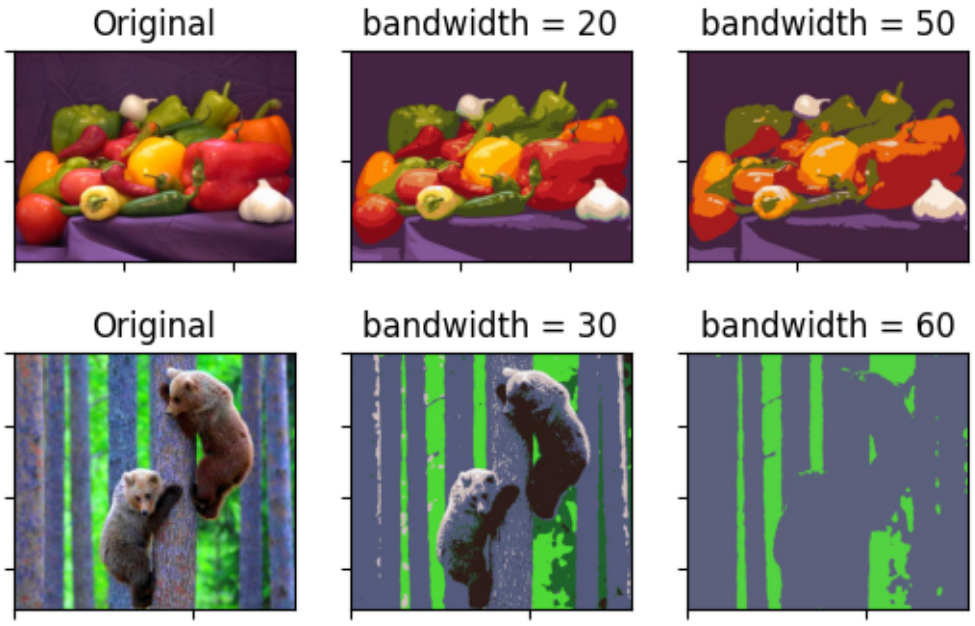
Part 2 - Segmentation with Mean Shift (10 points)

Write a function `seg = meanShiftSegment(img, bandwidth, colorSpace)` that takes as input a $n \times m \times 3$ images `img`, a floating point value for `bandwidth` that indicates the window size to use with Mean Shift.

Now, put all of your pieces of code together to segment the images `peppers.jpg` and `bears.jpg`. For each image, you must visualize three different interesting parameter settings, with one setting chosen to optimize the segmentation into semantically reasonable segments. Make sure to label the images with the parameters chosen and the number of modes found, and display each as a 1×3 labeled figure with the segmentation.

You may use smaller images when debugging your system to make it faster, and if necessary you can use them throughout. Just explain what you are doing.

Solution:




```
#https://github.com/leixiaoning/TF_Mean_Shift_Image_Clustering/tree/master
import numpy as np
from sklearn.neighbors import NearestNeighbors
import tensorflow as tf
from PIL import Image
import six
from collections import defaultdict
def meanShiftSegment(X, bandwidth, max_iter):
```

```
(m,n) = X.shape
print(m,n)
data = tf.constant(X, name="data_points")
b = tf.constant(bandwidth,dtype=tf.float32, name="bandwidth")
m = tf.constant(max_iter, name="maximum_iteration")
old_mean = tf.compat.v1.placeholder(tf.float32, [n], name="old_mean")
neighbors = tf.compat.v1.placeholder(tf.float32, [None,n], name="neighbors")
new_mean = tf.reduce_mean(neighbors,0)
euclid_dist = tf.sqrt(tf.reduce_sum(tf.pow(tf.subtract(old_mean, new_mean), 2)), name="
center_intensity_dict = {}
nbrs = NearestNeighbors(radius=bandwidth).fit(X)
bin_sizes = defaultdict(int)
data_point = tf.compat.v1.placeholder(tf.float32, [n],"data_point")
binned_point = tf.math.floordiv(data_point,b)
sess = tf.compat.v1.Session()
init = tf.compat.v1.initialize_all_variables()
sess.run(init)
for point in X:
    feed={data_point:point}
    bp = sess.run(binned_point,feed_dict=feed)
    bin_sizes[tuple(bp)] +=1

bin_seeds = np.array([point for point, freq in six.iteritems(bin_sizes) if freq >= 1],
bin_seeds = bin_seeds*bandwidth
print(len(bin_seeds))
j=0
for x in bin_seeds:
    print("Seed ",j,": ",x)
    i = 0
    o_mean=x
    while True:
        i_nbrs = nbrs.radius_neighbors([o_mean], bandwidth, return_distance=False)[0]
        points_within = X[i_nbrs]
        feed = {neighbors: points_within}
        n_mean = sess.run(new_mean, feed_dict=feed)
        feed = {new_mean: n_mean, old_mean: o_mean}
        dist = sess.run(euclid_dist, feed_dict=feed)
        if dist < 1e-3*bandwidth or i==max_iter:
            center_intensity_dict[tuple(n_mean)] = len(i_nbrs)
            break
        else:
```

```
                o_mean = n_mean
            print("\t",i,dist,len(i_nbrs))
            i+=1
        j+=1
    print(center_intensity_dict)
    sorted_by_intensity = sorted(center_intensity_dict.items(),key=lambda tup: tup[1], reve
    sorted_centers = np.array([tup[0] for tup in sorted_by_intensity])
    unique = np.ones(len(sorted_centers), dtype=np.bool)
    nbrs = NearestNeighbors(radius=bandwidth).fit(sorted_centers)
    for i, center in enumerate(sorted_centers):
        if unique[i]:
            neighbor_idxs = nbrs.radius_neighbors([center],return_distance=False)[0]
            unique[neighbor_idxs] = 0
            unique[i] = 1  # leave the current point as unique
    cluster_centers = sorted_centers[unique]
    nbrs = NearestNeighbors(n_neighbors=1).fit(cluster_centers)
    labels = np.zeros(154401, dtype=np.int)
    distances, idxs = nbrs.kneighbors(X)
    labels = idxs.flatten()
    return cluster_centers, labels

im_name = "bears"
im = Image.open(im_name+".jpg")
X = np.array(im)
r,c,_ = X.shape
X = X.reshape(r*c,3)
cluster_centers,labels = meanShiftSegment(X, 60, 300)
number_of_clusters = len(np.unique(labels))
labels = np.reshape(labels,[r,c])
# print labels.shape
# print labels
# print cluster_centers
segmented = np.zeros((r,c,3),np.uint8)
for i in range(r):
    for j in range(c):
            segmented[i][j] = cluster_centers[labels[i][j]][0:3]
Image.fromarray(segmented).save("segmented_"+im_name+"_60bands.jpg")
```

Part 3 - Using Pre-Trained Deep CNNs (6 points)

For this problem you will use a pre-trained neural network and you will use it as a feature extractor for the images. Run the model on `peppers.png` and output the labels of the top-3 predicted categories.

You may use the toolbox of your choice, but I suggest using Keras, TensorFlow, or PyTorch. Keras is probably easiest. TensorFlow is the most widely used toolbox in industry. PyTorch is widely used by academics and it is what I recommend if you are a PhD student doing deep learning research because it makes implementing new algorithms much easier than the other toolboxes. You can get information about how to do this in Keras here: `https://keras.io/applications/` **Preprocessing steps:** Resize your image to be $n \times n \times 3$ ($n = 224$ generally), then cast the image as a single precision image with values from 0 to 255. Finally, subtract the average image in the CNN model from your image. Now it is ready to be given to the CNN.

Solution:
https://keras.io/api/applications/ Predicted: 'bell_pepper': 0.9926563, 'cucumber': 0.0027988425, 'acorn_squash': 0.00042056912

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np

model = ResNet50(weights='imagenet')

img_path = 'peppers.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

Appendix