Assignment 4: Image Segmentation

Due Date: 22 November 2024, 23:59

Extension granted until: 24 November 2024, 23:59

Introduction

For this assignment you are asked to implement two well-known image segmentation techniques. In the first part, you will need to apply the K-Means clustering algorithm on an image, so as to cluster its pixels into k coherent segments. Regarding the second part, you will implement the Efficient Graph-Based Image Segmentation method, in order to segment an image.

Preliminary step

For all checks that are needed for the following questions, you should use the `assert()` function. All intermediate and final results should be plotted using the `matplotlib` package.

1 K-Means Clustering (50 points)

For the first part of this assignment you will implement the K-Means clustering algorithm, and apply it on the image `data/home.jpg` (load and plot the image). You will need to create a function with the following function signature:

```
def kmeans(data, K, thresh, n_iter, n_attempts):
    """
        Cluster data in K clusters using the K-Means algorithm
    :param data: numpy.array(float), the input data array with N (#data) x D (#feature_dimensions) dimensions
    :param K: int, number of clusters
    :param thresh: float, convergence threshold
    :param n_iter: int, #iterations of the K-Means algorithm
    :param n_attempts: int, #attempts to run the K-Means algorithm
    :return:
        compactness: float, the sum of squared distance from each point to their corresponding centers
        labels: numpy.array(int), the label array with Nx1 dimensions, where it denotes the corresponding cluster of
                                  each data point
        centers : numpy.array(float), a KxD array with the final centroids
    """
```

Your implementation should do the followings checks:

- `data`: should be a $2D$ array
- `K, thresh, n_iter, n_attempts`: should be non-zero positive numbers
- `K`: should be less or equal to the number of data (N)

The algorithm will run `n_attempts` times, for `n_iter` iterations each time. For each attempt, `K` different centroids should be selected randomly from the data and then use the procedure as explained in class (and in the lab) to cluster the data and recalculate the centroids. Each attempt will terminate either if the centroids have converged w.r.t. to `thresh` or if the maximum number of iterations is reached. At the end of each attempt you will calculate the compactness (distortion) of the clustering as the sum of squared Euclidean distances of points to their corresponding cluster centroid:

$$compactness^{(i)} = \sum_{j=1}^{k} \sum_{x_p \in C_j^{(i)}} \left|\left|x_p - c_j^{(i)}\right|\right|^2$$

where $c_j^{(i)}$ is the centroid of cluster $C_j^{(i)}$ at the i^{th} attempt. The function will return the `labels` and `centers` (cluster centroids) of the attempt with the smaller compactness, along with the `compactness` itself.

Your results should be validated using OpenCV's function as follows:

```
K, thresh, n_iter, n_attempts = 4, 1.0, 10, 10
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, n_iter, thresh)
cv2.kmeans(<input_data>, K, None, criteria, n_attempts, cv2.KMEANS_RANDOM_CENTERS)
```

For your implementation you should use the same input parameters. Run your implementation (and OpenCV's) two times on the same image and use different input features at each time:

1. use as input features the (r, g, b) pixel values
2. use as input features the pixel coordinates and values, i.e. (i, j, r, g, b) where i is the row (y coordinate) and j (x coordinate) is the column of the pixel

Plot the clustered images from your and OpenCV's implementation in two 1×2 grids (one grid for each input feature).

2 Efficient Graph-Based Image Segmentation (50 points)

For this part of the assignment you are asked to implement a simplified version of the Efficient Graph-Based Image Segmentation method. First you will load (and plot) the image `data/eiffel_tower.jpg`. Then smooth out the input image using Gaussian blurring with kernel size 3×3 and $\sigma = 0.8$. Next you are going to create a Nearest Neighbor graph (NN-graph) based on the blurred image. Each pixel will represent a node of the graph with coordinates (i, j) (pixel coordinates) and it will be connected with its 10 nearest neighbors in the feature space (i, j, r, g, b). Each edge weight will be determined by the Euclidean distance in that feature space. In order to construct this graph you need to implement the following function:

```
def nn_graph(input_image, k):
"""
        Create a graph based on the k-nearest neighbors of each pixel in the (i,j,r,g,b) feature space.
        Edge weights are calculated as the Euclidean distance of the node's features
        and its corresponding neighbors.
    :param input_image: numpy.array(uint8), input image of HxWx3 dimensions
    :param k: int, nearest neighbors for each node
    :return:
        graph: tuple(V: numpy.array(int), E: <graph connectivity representation>), the NN-graph where
               V is the set of pixel-nodes of (W*H)x2 dimensions and E is a representation of the graph's
               undirected edges along with their corresponding weight
    """
```

This function should check that the `input_image` has 3 dimensions and that `k` is a non-zero positive number. For representing the connectivity of the graph you can use the `sklearn.neighbors.NearestNeighbors()` class from the `scikit-learn` package.

Then you will need to implement the main algorithm of the graph-based segmentation method, as described in Algorithm 1. The *minimum internal difference* $MInt(\mathcal{C}_1, \mathcal{C}_2)$ between the two clusters $\mathcal{C}_1$ and $\mathcal{C}_2$ is calculated as:

$$MInt(\mathcal{C}_1, \mathcal{C}_2) = min\big(Int(\mathcal{C}1) + \tau(\mathcal{C}1), Int(\mathcal{C}_2) + \tau(\mathcal{C}_2)\big)$$

where $Int(\mathcal{C})$ is the *internal difference* of the cluster $\mathcal{C}$:

$$Int(\mathcal{C}) = \max_{e_i \in G=(\mathcal{C},\mathcal{E})} w(e_i)$$

and $\tau(\mathcal{C}) = \frac{k}{|\mathcal{C}|}$. The $G = (\mathcal{C}, \mathcal{E})$ is a sub-graph of the input graph, which represents the connectivity of the nodes of the cluster $\mathcal{C}$.

The above procedure should be implemented by the following function:

```
def segmentation(G, k, min_size):
    """
        Segment the image base on the Efficient Graph-Based Image Segmentation algorithm.
    :param G: tuple(V, E), the input graph
    :param k: int, sets the threshold k/|C|
    :param min_size: int, minimum size of clusters
    :return:
        clusters: numpy.array(int), a |V|x1 array where it denotes the cluster for each node v of the graph
    """
```

Algorithm 1: Efficient Graph-Based Image Segmentation

Input: $G = (\mathcal{V}, \mathcal{E})$
Output: segmentation S

Sort edges into $\pi = (e_1, \cdots, e_m)$ by ascending edge weight ;
Initial segmentation $S^{(0)} \in N^{|\mathcal{V}|}$; `// each vertex` v_i `is in its own segment`
for $q = [1, \cdots, m]$ **do**
  $(v_i, v_j) \longleftarrow e_q$; `// vertices of edge` e_q
  $c_i \longleftarrow S^{(q-1)}(v_i)$; `// cluster/segment id of vertex` v_i `at segmentation` $S^{(q-1)}$
  $c_j \longleftarrow S^{(q-1)}(v_j)$; `// cluster/segment id of vertex` v_j `at segmentation` $S^{(q-1)}$
  if $c_i != c_j$ **then**
    $\mathcal{C}_i \longleftarrow \{v_p : S^{(q-1)}(v_p) == c_i, \forall p, 1 \leq p \leq |\mathcal{V}|\}$; `// cluster` $\mathcal{C}_i$
    $\mathcal{C}_j \longleftarrow \{v_p : S^{(q-1)}(v_p) == c_j, \forall p, 1 \leq p \leq |\mathcal{V}|\}$; `// cluster` $\mathcal{C}_j$
    if $w(e_q) \leq MInt(\mathcal{C}_i, \mathcal{C}_j)$ **then**
      $S^{(q)} \leftarrow [S^{(q-1)}(v_p) = c_i \text{ or } c_j : v_p \in \{\mathcal{C}_i \cup \mathcal{C}_j\}]$; `// merge` $\mathcal{C}_i$ `and` $\mathcal{C}_j$
    end
  else
    $S^{(q)} \longleftarrow S^{(q-1)}$;
  end
end
$S \longleftarrow S^{(m)}$;

You should check that `k` and `min_size` are non-zero positive numbers. As an extra post-processing step after the graph is segmented using Algorithm 1, clusters that are adjacent, i.e. an edge exists that connects them, should be merged if at least one of them has size lower than `min_size`. Run your implementation with the following input parameters:

```
k, min_size = 550, 300
```

The output clusters should be mapped into distinct colors (you can use HSV color space for this) and the input image should be colored based on this mapping. Plot your results.

3 Instructions

- The assignment is due by **Friday November 22nd, at 23:59** (see the course's GitHub repository for more details regarding the late assignment policy).
- The assignment should be submitted **only** through Moodle. Compress all the needed **source code** files, e.g. all *.py files into a .zip file and name it as follows "**Lab_Assignment_4_<UC-ID-Number>.zip**".
- For further questions you can create an issue on the course's GitHub repository.
- All lab assignments should be conducted **independently**.
- The Moss system will be used to detect code similarity.
- Plagiarism and/or cheating will be punished with a grade of zero for the current assignment. A second offense will carry additional penalties.