**Superpixels**

Rewritten by Han

SLIC:

SLIC was first proposed by Achanta et. al. [10], and their motivation was to find an algorithm to compute nearly equal-sized superpixels efficiently for an image. Intuitively, for all pixels, the algorithm performs local clustering of a 5-D space data defined by *L, a, b*, values of the CIEAB color standard and the *x, y* coordinates. Suppose an image has *N* total pixels, and a user sets *K* as the number of total superpixels to have in the image. Then, there will be around pixels for each superpixel. If our goal is to make each superpixel’s shape similar to a perfect square, then the length of its side will be . Thus, this value will be a good estimate for the interval distance between each pair of initial superpixel centers on the image. After that, we can use to denote each superpixel center, with *k* as the index from 1 to *K*. Here, Achanta et. al. [10] made an assumption that the search area for the pixels associated with a superpixel center will be within a area around the center, since the area of each superpixel is about .

Then according to [10], the *L, a, b* values are used to calculate the color distances from neighboring temporal superpixel centers, while the *x, y* coordinates are used to estimate the Euclidean distances from the center. The following formula from [10] was intended to combine both color and geometric distances for each pixel from a nearby superpixel center:

Here, and are the Euclidean distances for *L, a, b* values and *x, y* coordinates, respectively. is a linear combination of and . Specifically, , and the *m* value is chosen by the user. Since will be fixed for fixed *N* and *K* values, the *m* value will determine how much weight we put on the distance for *x, y* coordinates.

Then, a minor adjustment will be done for the locations of initial superpixel centers to prevent them from sitting on any edges [10]. We ignored this minor detail here, but if you are interested in this, please do not hesitate to check the original paper for the detail. Finally, the formal algorithm proposed by [10] will be to first initialize the superpixel centers on the grid of an image by the interval distance of . Then, each pixel in the image will be assigned to its nearest superpixel center. After this step, a new superpixel center will be calculated by averaging all the of each pixel belonging to that center. Then, it will repeat the above two steps, until the results converge.

The advantage of using SLIC to compute superpixels is that its efficiency is outstanding. According to [10], its efficiency is , where *N* is the total number of pixels in an image. However, since the SLIC algorithm is a special case of the K-means algorithm in the unsupervised learning field, it may inherit some disadvantages from the K-means algorithm family.

NCuts:

Ncuts was first proposed and introduced by Shi et. al. [11], whose intention is to use graph partition method to consider global features in image segmentation problem. Instead of focusing on the local features like the SLIC superpixels, Ncuts attempts to get the global sense of an image as how human beings do for an image. In this algorithm, Shi et. al [11] model an image as a weighted undirected graph . Each pixel in is modeled as a node in a graph, and the edges have the weights that are the similarities between the pixel and its neighboring pixels. Then, segmenting a part of image out of it is similar to have a graph cut. Suppose we want to cut the whole graph into two sub-graphs *A* and *B*, and then let denote the edge weight between the node *u* and *v* in the graph. Now, the graph cut cost can be defined as the following:

According to Shi et. al. [11], by considering the dissimilarities among different groups and the similarities within each group in the modeling, Ncuts avoids the tendency to just group individual pixel out from others. Then, the disassociation measurement between different sub-graphs can be written as following, and here we just use two different sub-graphs as an example [11]:

Here, is defined as the total edge weights from the nodes of group A to all the rest nodes, and similar definition applies to . As we discussed above, instead of just use as a measurement, by putting the and in the denominators, we cost is normalized. This the reason why the algorithm is called Ncuts. Then, Shi et. al. [11] proved that minimization of the disassociation among different sub-graphs is equivalent to maximization of the association within each sub-graph. Therefore, we just need to use the formula for the next step.

Then, to segment one sub-graph out of the whole graph for in image, there is an elegant mathematical way to optimize it [11]. We save the details here and provide the core mathematical part here. If you are interested in the optimization detail, please refer to the original paper. Let be the weights of all edges incidental to the node *i*; let ***D*** be an diagonal matrix, where *N* is the total number of the nodes, and of each node is on the diagonal of the matrix. Moreover, let ***W*** be an edge weight matrix with the size of , and each element as the edge weight between the node *i* and the node *j*. We also need to define a *N* dimensional vector ***x*** to indicate which nodes are in the segmented-out sub-graph and which nodes are among the other sub-graph. Now, after defining these variables, the problem will be reduced as the following mathematical eigenvector calculation [11]:

Here, is a corresponding eigenvalue.

Therefore, with the above definitions and formula. The Ncuts algorithm can be summarized as following [11]. First, it sets up the graph model for an image and calculates the weight for each edge for each pair of neighboring pixel nodes. Then, solve the above eigenvector calculation and use the corresponding eigenvector as the solution for the sub-graph to be segmented out from the original graph. Finally, recursively repeat the above steps until no more necessary sub-graphs left from the original whole graph. According to both [10] and [11], the efficiency of the Ncuts algorithm is , which is generally slower than the SLIC superpixels. But it considers the global features in the calculation and uses normalizations to prevent extreme cases.

TurboPixels:

TurboPixels was first introduced by Levinstein et. al. [14] to compute compact superpixels with high efficiency by using geometric flows. They found that oversegmentation will be better than undersegmentation, since merging superpixels is easier than splitting them. This becomes one of the motivations for TurboPixels. Formally speaking, there are five principles behind TurboPixels; they are uniform size and coverage, connectivity, compactness, smooth and edge-preserving flow, and no superpixel overlap [14]. Since this superpixel algorithm is very mathematical heavy, we will not list all mathematical details in this introduction but show the intuitions of the design. If you want to check the mathematical details involved, please check the original paper. According to [14], the intuition of the design is to first put a user-given number of small circle seeds on the original image and distribute them on the grid of the image with roughly the same distances among the neighboring seeds. Then, the algorithm will try to expand the boundaries of all seeds repeatedly until they cannot evolve. After that, with the final seeds’ circles, the algorithm can infer the corresponding lattices that contain them, respectively. As a result, these lattices are the superpixels that we want. Compared with Ncuts, the efficiency of TurboPixels is higher, and it is roughly , where *N* is the total number of pixels in an image. Different from Ncuts that uses graph-cut technique, TurboPixels utilizes the geometric flows to “grow superpixels from the seeds”. Moreover, TurboPixels does not consider that much global features as Ncuts.