Lazy Random Walks for Superpixel Segmentation

Team: LenaIsLove

Superpixels

Superpixels are groups of perceptually uniform pixels in an image that can segment the image into intensity-based multiple regions.

Natural and perceptually meaningful representation of the image

Reduces number of image primitives for computational efficiency

Ideal criteron for superpixel algorithm:

- **adhere well to object boundaries of image.**
- **maintain the compact constrains in the complicated texture regions.**

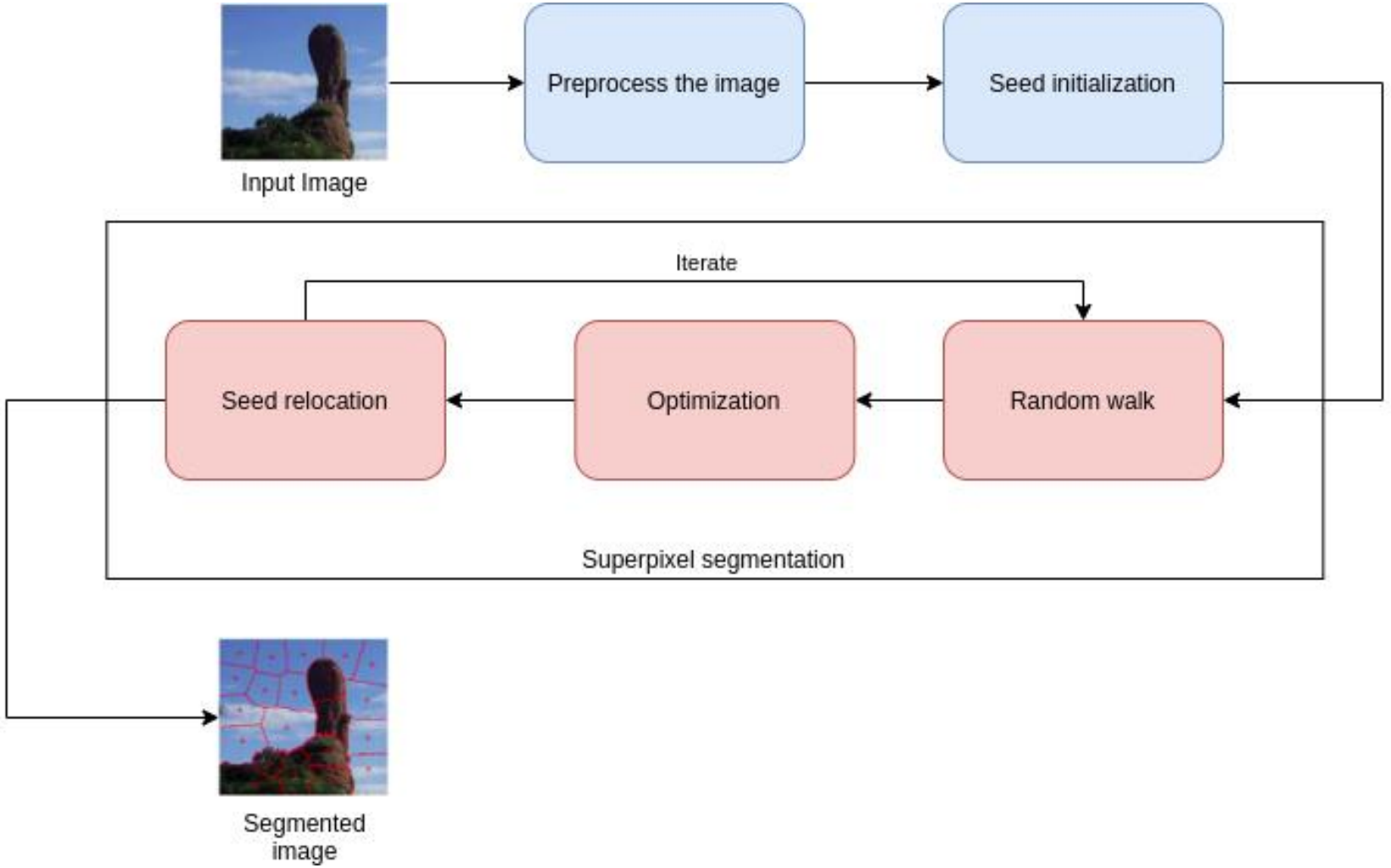
Related work

Turbopixels: a geometric-flow-based algorithm where each superpixel is grown by the means of an evolving contour after selecting initial superpixel centers

Simple Linear Iterative Clustering(SLIC): Adopts segmentation of an image based on a simple k-means clustering approach. It divides the image in k superpixels which results in very irregular boundaries.

Random walk: This algorithm leverages random walks for image segmentation. Assuming that a particular cell is labelled, this algorithm calculates the probability of reaching an unlabeled cell starting from a labelled cell. Backbone of our project.

Algorithm flowchart



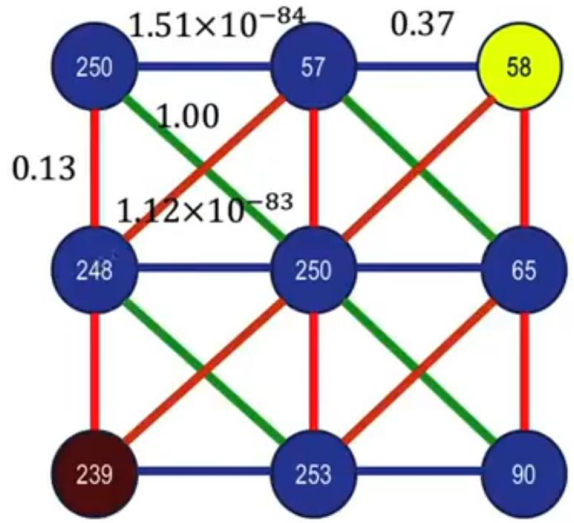
Seed initialization

Evenly spaced seeds need to be distributed throughout the whole image.

- If K = input number of superpixels, N = total number of pixels in image, we create a grid of seeds uniformly placed at distances of root(N/K).
- Since image intensities are not uniform, this strategy may place seeds on or close to a strong edge.
- We perturb the position of each seed by moving it in the direction of the image gradient as a function of the gradient magnitude obtained from the grayscale image, with the maximum perturbation determined by the seed density.

Defining the graph

- To apply the Lazy Random Walks algorithm to our image we first need to convert the image to a graph.
- We create an undirected graph from the image by assuming each pixel to be a node in the graph.
- We then connect the node of pixel with its 8-neighborhood pixels by edges and set the weight of the edges between nodes v_i and v_j with intensities g_i and g_j using the Gaussian weighing function as:



$$w_{ij} = exp(-\frac{||g_i - g_j||^2}{2\sigma^2})$$

$\beta = 1/(2*\sigma^2)$ is a user defined parameter.

Random Walks on a Graph

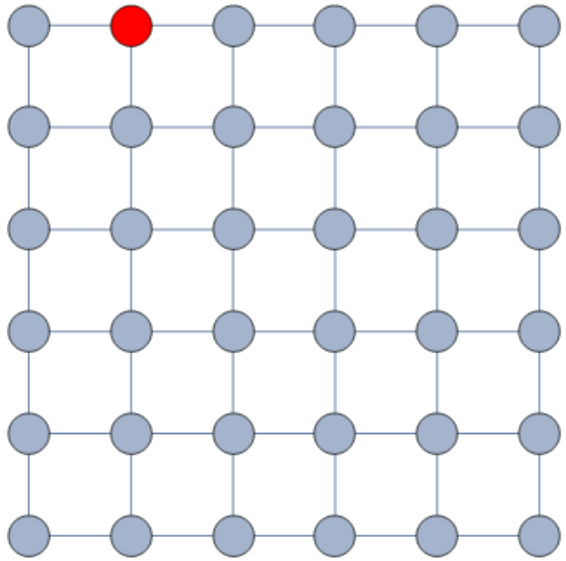
A random walk on a graph is defined as a process which begins at a node and moves to a connected neighbor at every timestep along all possible paths.

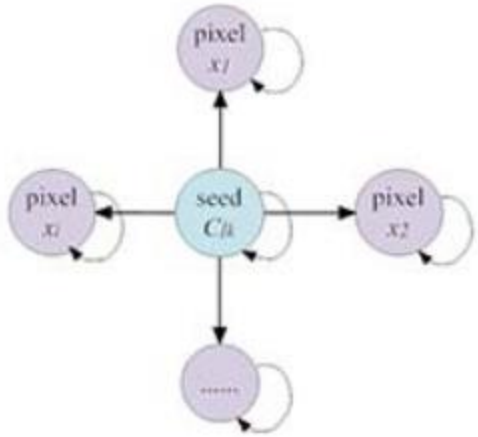
In weighted graphs, the probability of choosing the next node is determined by the weight of the edge.

Lazy random walk is a special type of random walk in which the process is allowed to stay on the same node with some predefined probability.

In this paper the authors use the lazy random walk (LRW) algorithm to determine the labels of each pixel.

α : probability to walk out of a node along an edge
(1-α): probability to stay at the current node





Lazy Random Walk

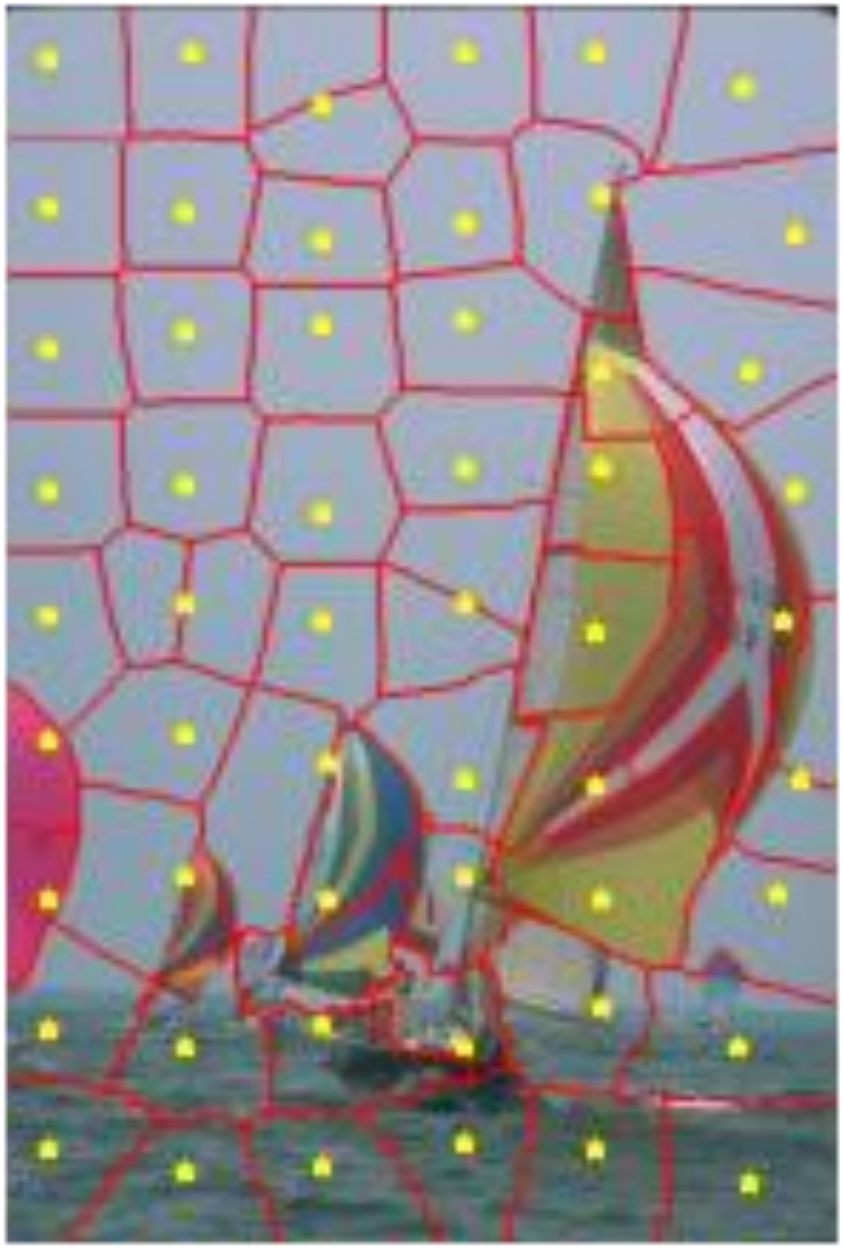
We use the LRW algorithm for superpixel initialization given seed points. For each pixel in the image, find the commute time to each of the seeds defined.

Commute Time (CT)

The expected time taken for a process which starts at a pixel to go to the seed and back. Commute time is inversely related to the probability of a pixel being assigned to a seed.

We then assign the pixel to the seed label which has the lowest commute time.

$$R(x_i) = \underset{l_k}{argmin}\, \mathcal{CT}(c_{l_k}, x_i)$$



Finding the Commute Time

- To find the commute time for each pixel to each seed we need to first define the Laplacian of graph.
- The Laplacian of a graph is defined as $$L_{ij} = \begin{cases} d_i & \text{if } i = j, \\ -\alpha w_{ij} & \text{if } i \sim j, \\ 0 & \text{otherwise,} \end{cases}$$
- d_i is defined to be the degree of a node. It is the sum of all the weights of edges connected to a node.
- Hence the Laplacian can also be written as $L = \mathcal{D} - \alpha \mathcal{W}$
- Then the commute time between to vertices i and j can be defined as:

$$CT_{ij} = \begin{cases} L_{ii}^{-1} + L_{jj}^{-1} - L_{ij}^{-1} - L_{ji}^{-1} & \text{if } i \neq j, \\ 1/\pi_i & \text{if } i = j, \end{cases}$$

$$\pi_i = d_i / \sum_{i=1}^{N=|V|} d_i$$

Why does LRW perform better than RW?

The RW algorithm computes the first arrival probability that a random walk starts at one pixel first reaches one of the seeds with each label, and then that pixel is denoted as the same label with maximum probability of the corresponding seed. A random walk starting from a pixel must first arrive at the position of the prelabeled seed, and thus it only considers the local relationship between the current pixel and its corresponding seed.

The first arrival probability ignores the global relationship between current pixel and other seeds.

In order to respect the global relationship as described above, the authors add a self-loop to the graph to make the RW process lazy.

Since a vertex with a heavy self-loop is more likely to absorb its neighboring pixels than the one with light self-loop, which makes the vertex to absorb and capture both the weak boundary and texture information with self-loops.

Energy optimization

- For improving the performance of superpixels, we minimize the following optimization function:

$$E = \sum_{l} (Area(S_l) - Area(\bar{S}))^2 + \sum_{l} \widetilde{W}_x \mathcal{CT}(c_l^n, x)^2$$

Data item:
Makes texture information of image to be distributed uniformly in the superpixels by **splitting large superpixels** into small ones.

Smoothness item:
makes the boundaries of superpixels to be more consistent with the object boundaries in the image by performing **optimal center relocation**.

W_x is a penalty function for the superpixel label inconsistency inversely related to commute time (CT)

As the above function is non-convex, we iteratively optimize it. First the optimal centre locations are found by taking the derivative of the smoothness term wrt c and equating it to 0.

Energy optimization(contd.)

$$Area(S_l) = \sum_{i \in S_l} LBP_i, \quad Area(\bar{S}) = \frac{\sum_{i \in S_l} LBP_i}{N_{sp}}$$

Area(S_l) is the area of superpixel

Area(S) is the average area of all superpixels and and Nsp is the user defined number of superpixels.

For getting the superpixel area, we leverage LBP(local binary patterns) which is an efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number. We follow the below criteria to split the superpixels:

If user-defined threshold Th is small, over-segmentation occurs, and we get poor results.

$$\frac{Area(S_l)}{Area(\bar{S})} \geq Th$$

If Th is large, more focus is put on the smoothness term minimization and centre relocation, and to achieve the desired no. Nsp of superpixels the algorithm takes longer to converge.

Splitting superpixels

The larger superpixel is split into smaller superpixels along the direction of the largest variation in Commute Time(CT) and shape. For this we define a covariance matrix:

$$\sum_{\{x|x\in S_l, x\neq c_l\}} \frac{CT(c_l,x)^2}{||x-c_l||^2}(x-c_l)(x-c_l)^T$$

The eigenvectors corresponding to the largest eigenvalue of the above matrix is found. By Principal Component **Analysis**, the split direction is determined by:

$$(x-c_l)\cdot s = 0$$

LRW is repeated after each iteration to refine the obtained superpixels.

Convergence: The optimization converges when the changes of energy values between two successive iterations is less than a predefined value V or the user defined number of superpixels have been obtained.

Evolution of superpixels with iterations

Experiments

We use the **Berkeley Segmentation dataset (BSDS300)** - a widely used public dataset for image segmentation and boundary detection consisting of 300 images and their corresponding ground truth human segmentations.

https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/

Output images link:

https://iiitaphyd-my.sharepoint.com/:f:/g/personal/dipanwita_g_research_iiit_ac_in/EhbiJ1CZBhhDrqRKd8xSZVMBuYEK957HeD6tmW15Urz8tw?e=wkeVB4

Test parameters

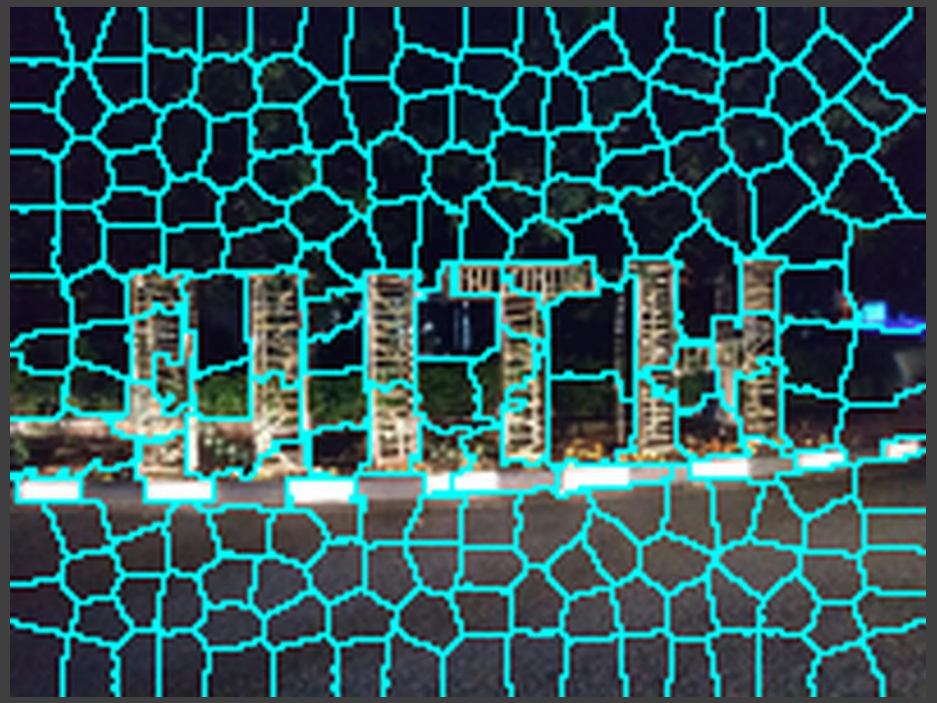
Number of superpixels = 200	Threshold = 1.35	$\beta = 30$	$\alpha = 0.992$	Max iterations = 10

Qualitative results - 1

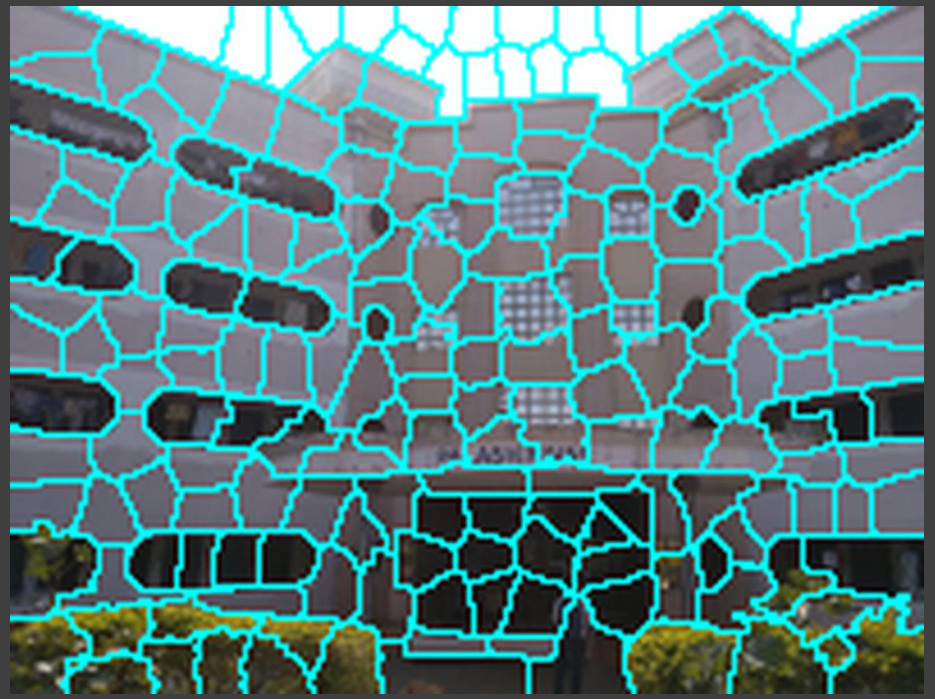
Qualitative results - 2

Qualitative results - 3

Qualitative results - 4



Qualitative results – Own image1

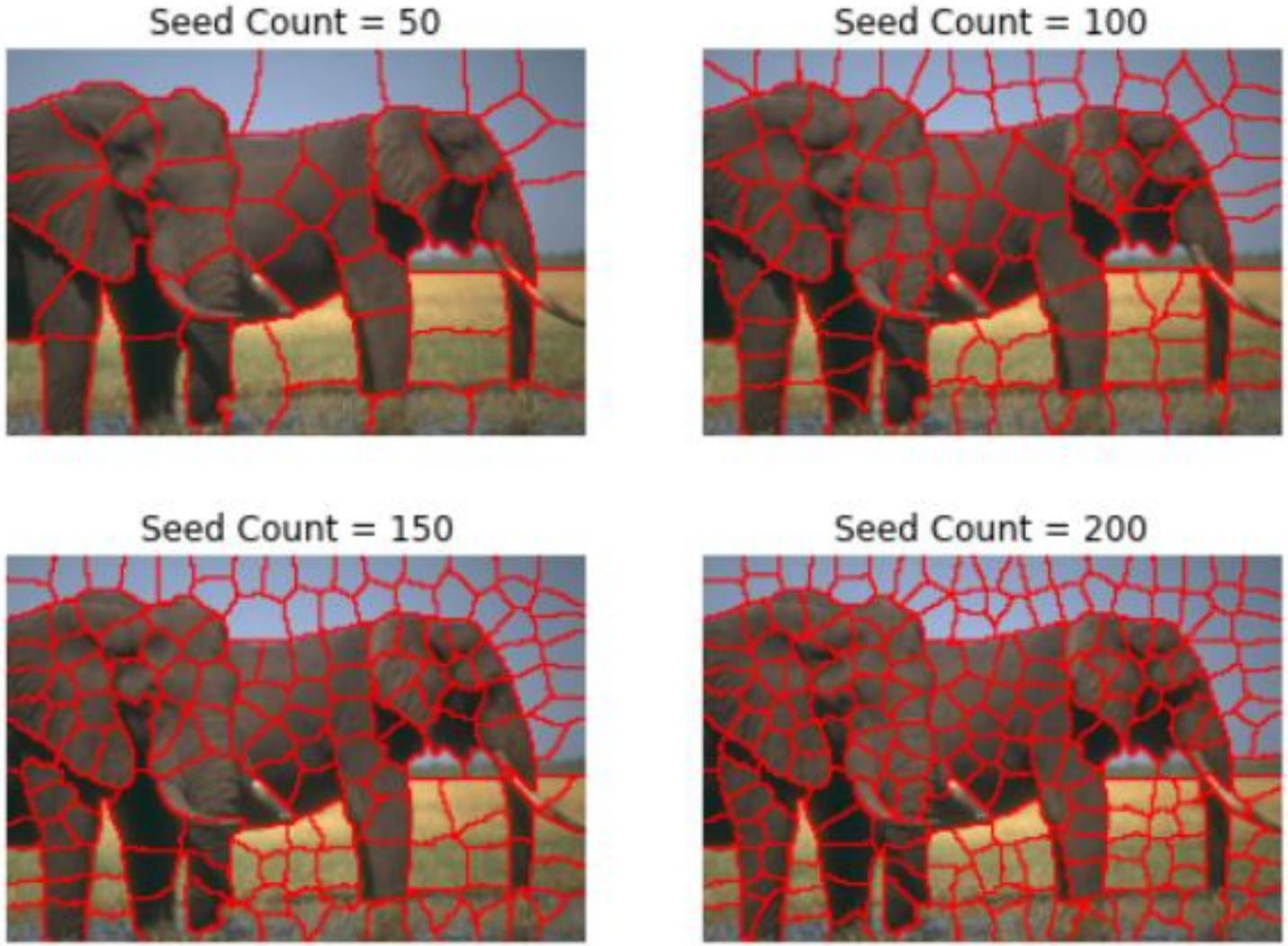


Qualitative results – Own image2

Qualitative results – Own image3

Parameter Analysis

- **Role of seed count (Nsp)**



Parameter Analysis

- **Role of Threshold(Th):** With increase of threshold, the number of iterations to converge to Nsp will increase, but segmentation should be finer.
- Lower value of threshold will lead to over segmentation.

theshold = 0.5

theshold = 1.0

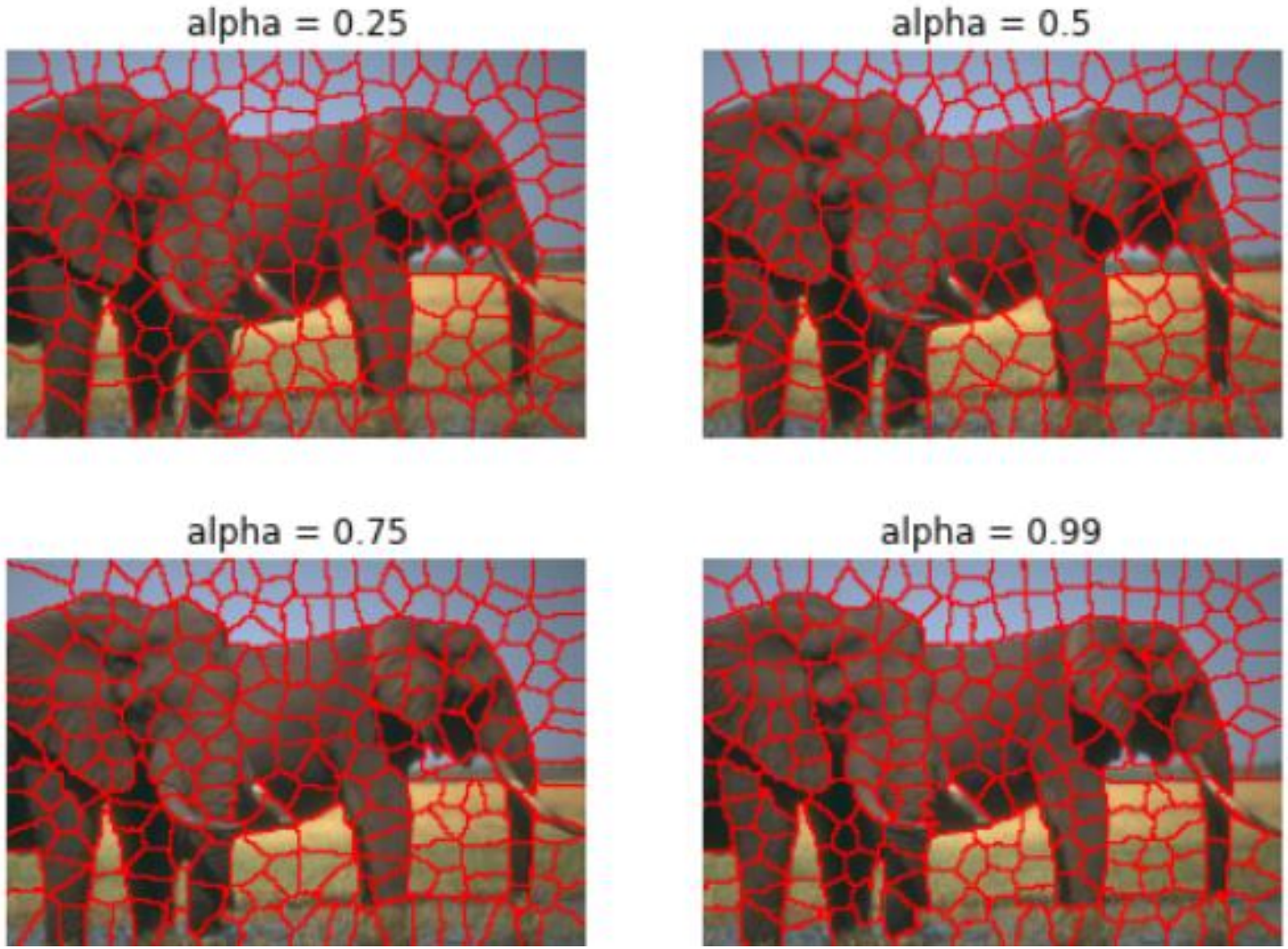
theshold = 1.5

theshold = 2.0

Parameter Analysis

- **Role of α:**

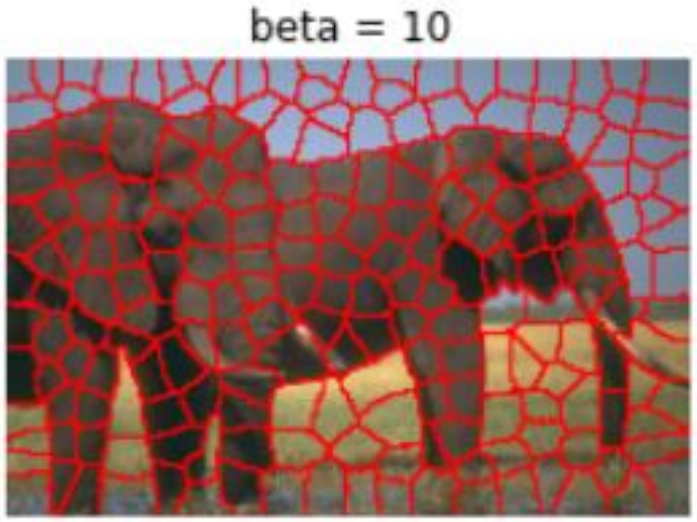
As α tends to 1, the algorithm becomes similar to Random Walk, as (1-α) indicates the probability of staying at the same node in the Lazy Random walk.

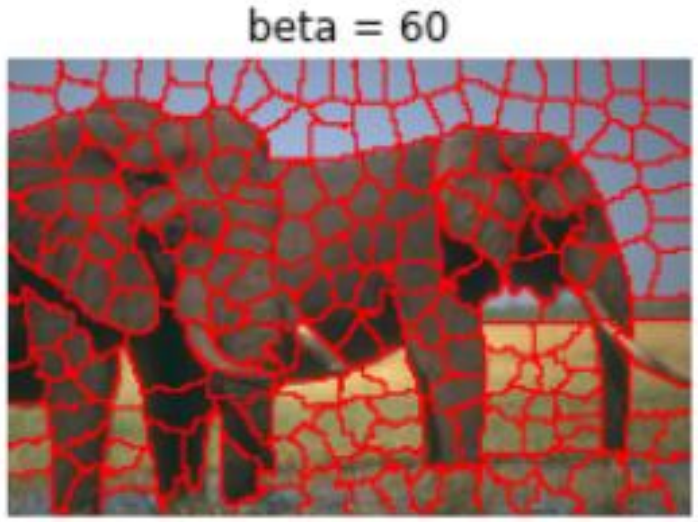


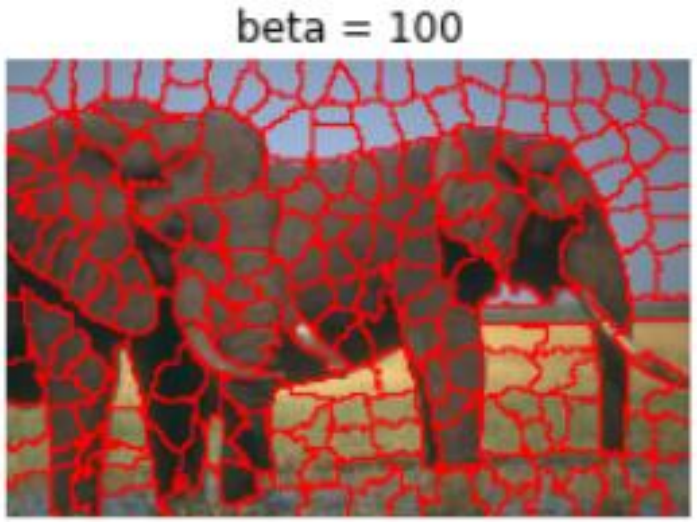
Parameter Analysis

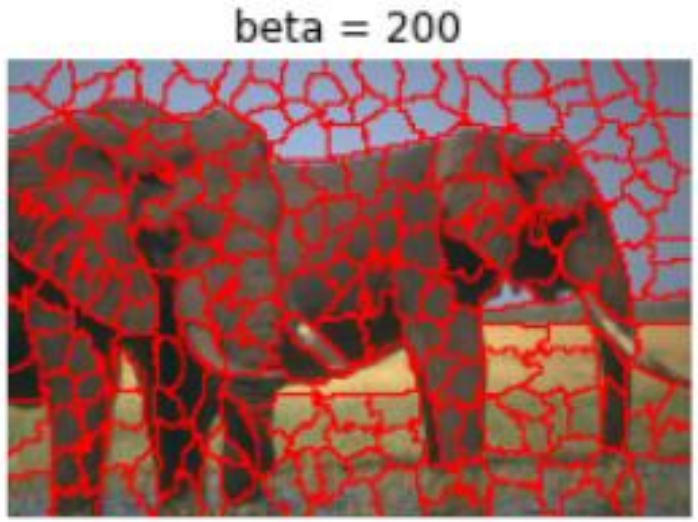
- **Role of β**:

With higher β values the edge weights between high intensity difference pixels will be very small. This would lead to precision errors (rough boundaries), and we won't get good results.



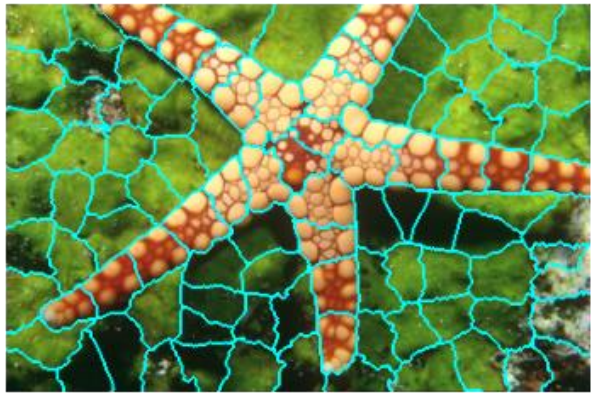




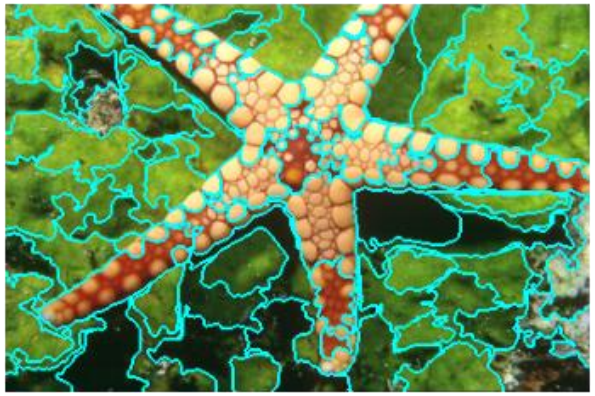


Qualitative comparison against other methods

Lazy Random Walk

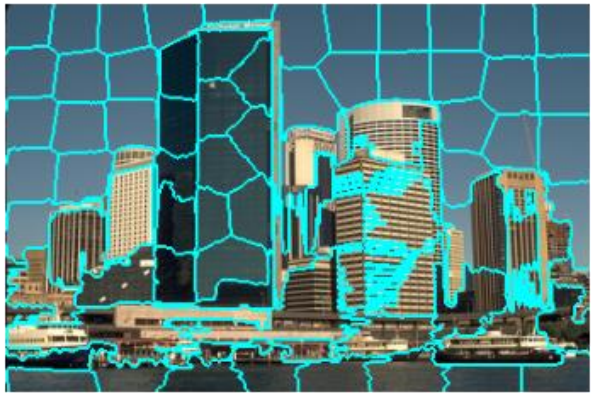


SLIC

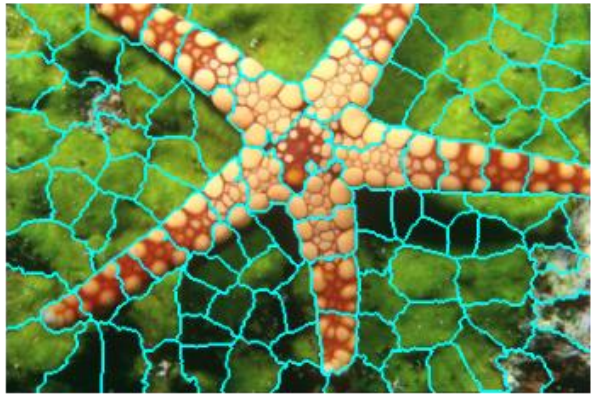


Lazy Random Walk

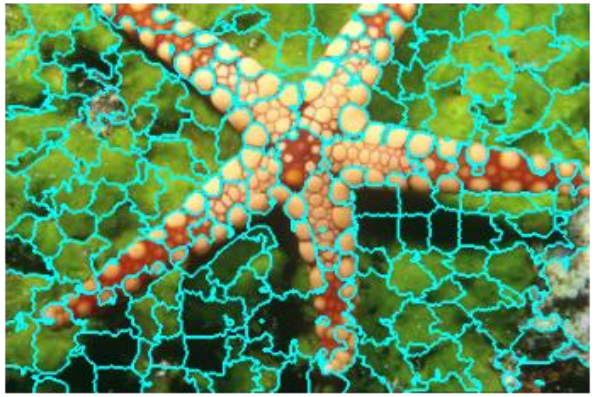
SLIC



Random Walk

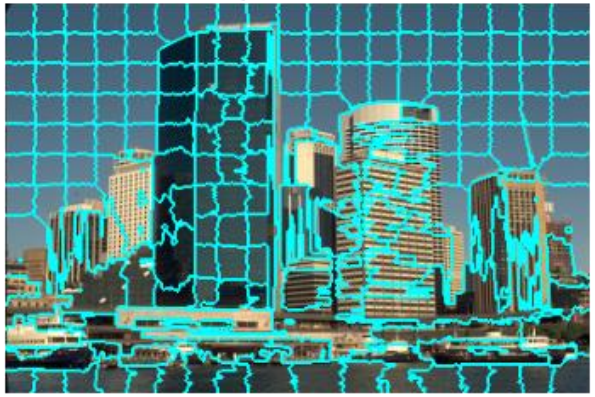


Compact watershed



Random Walk

Compact watershed



Quantitative results

Undersegmentation Error (UE)

It occurs when two superpixels belonging to different objects (semantically) are labelled as the same superpixel. We use the ground-truth segmentation to evaluate it.

$$UE = \frac{\sum_i \sum_{s_j || s_j \cap g_i| > 0} |s_j - g_i|}{\sum_i g_i}$$

Achievable Segmentation Accuracy (ASA)

It is the overlap of the ground-truth segmentation with the achieved segmentation.

$$ASA = \frac{\sum_j \underset{i}{argmax} |s_j \cap g_i|}{\sum_i |g_i|}$$

Quantitative comparisons

The following results were obtained on averaging the UnderSegmentation Error (UE) and Achievable Segmentation Accuracy (ASA) over 300 images from the BSDS300 dataset against the corresponding human labelled ground truth segmentation.

Algorithm	**UE (Lower is better)**	**ASA (Higher is better)**
LRW (Lazy Random Walks)	**0.16**	**0.82**
SLIC (Simple Linear Iterative Clustering)	**0.19**	**0.80**
RW (Random Walk)	**0.17**	**0.80**
QUICKSHIFT	**0.10**	**0.89**
WATERSHED	**0.09**	**0.89**

Building the GUI + Features

- We used the tkinter library in Python to build a GUI.
- Features provided:

The GUI supports upload of an image in any of the standard formats – JPG/PNG.

Initial image is displayed upon upload.

There are input boxes for all the parameters that can be tweaked for easy experimentation.

The output result is displayed after the algorithm is run and the image is finally saved to the disk.

The progress of the algorithm is displayed in STDOUT.

Screenshot of GUI


Live Demo

Thank You