# Project1: Simple Linear Iterative Clustering

Hanbin Wang
PB21000272

Maze Mo
PB21000211

Zhouyang Liu
PB21000231

## Abstract

*This report presents an implementation of the Simple Linear Iterative Clustering (SLIC) algorithm for superpixel segmentation. The primary goal is to segment images into meaningful superpixels that adhere to color boundaries and spatial proximity. The methodology involves converting images to the LAB color space, initializing cluster centers, iteratively refining clusters, and enforcing connectivity. Experimental results demonstrate the effectiveness of the implemented algorithm on various images, highlighting its advantages and limitations.*

## 1. Introduction

Superpixel segmentation is a crucial preprocessing step in many computer vision tasks, such as object recognition, image segmentation, and texture analysis. By grouping pixels into perceptually meaningful atomic regions, superpixels reduce computational complexity and preserve essential image structures.

The Simple Linear Iterative Clustering (SLIC) algorithm is a popular method for generating superpixels due to its efficiency and simplicity. This report details the implementation of the SLIC algorithm, discusses current research in the field, outlines the methodology, presents experimental results, and provides a critical analysis of the advantages and shortcomings of the approach.

## 2. Current Status of Research and Main Methods

### 2.1. Superpixel Segmentation

Superpixel segmentation aims to partition an image into regions (superpixels) that are coherent in terms of color and texture. Superpixels serve as a middle ground between pixels and objects, capturing local image structures while reducing computational load for higher-level vision tasks.

### 2.2. SLIC Algorithm

Introduced by Achanta et al. [1], the SLIC algorithm is an adaptation of k-means clustering for superpixel segmentation. It operates in the five-dimensional space of color and spatial coordinates, balancing color similarity and spatial proximity. The key advantages of SLIC include:

- **Efficiency**: Reduces computational complexity by limiting cluster updates to local neighborhoods.
- **Simplicity**: Straightforward implementation with intuitive parameters.
- **Flexibility**: Parameters can be adjusted to control the size and compactness of superpixels.

### 2.3. Related Work

Other superpixel algorithms include:

- **Normalized Cuts**[4]: Produces high-quality superpixels but is computationally intensive.
- **Graph-Based Methods**: Such as Felzenszwalb's algorithm [2], which is efficient but may produce irregular superpixels.
- **Turbopixels**[3]: Offers uniform size and compactness but suffers from slow processing and relatively poor boundary adherence.

SLIC strikes a balance between computational efficiency and segmentation quality, making it widely used in practical applications.

## 3. Research Methodology

The implementation follows the standard procedure of the SLIC algorithm with additional steps for enforcing connectivity and visualizing results.

### 3.1. Preprocessing

- **Image Loading**: Read images from the specified data directory.
- **Color Space Conversion**: Convert images from BGR to LAB color space using OpenCV's `cv2.cvtColor` function. The LAB color space is used because it is perceptually uniform, making Euclidean distances in this space correspond to perceptual color differences.

## 3.2. Initialization

- **Parameter Setting**:
  - $n$: Desired number of superpixels.
  - $nc$: Compactness factor for color proximity.
  - $step$: Computed as the square root of the image area divided by $n$, representing the initial spacing between cluster centers.
- **Cluster Centers Initialization**:
  - Place initial cluster centers on a regular grid across the image.
  - Adjust each center to the position with the lowest gradient in a $3 \times 3$ neighborhood to avoid placing centers on edges.
  - Store each center's LAB color values and spatial coordinates.

## 3.3. Clustering

- **Iterative Refinement**:
  - For a maximum of `max_iter` iterations (default is 20):
    - **Local Clustering**:
      - For each cluster center, consider pixels within a $2D$ square neighborhood of size $2 \times step$.
      - Compute the distance between the center and each pixel in the combined five-dimensional space (LAB color and XY spatial coordinates).
      - Update the label of each pixel if the computed distance is less than its current distance.
    - **Center Update**:
      - After all pixels are labeled, recompute the cluster centers by averaging the LAB color values and spatial coordinates of all pixels assigned to each cluster.

## 3.4. Enforcing Connectivity

- **Small Region Merging**:
  - Identify connected components in the label matrix using a flood-fill algorithm.
  - Merge regions smaller than a threshold (quarter of the expected superpixel size) with the nearest larger region to ensure superpixel connectivity.

## 3.5. Post-processing and Visualization

- **Contour Detection**:
  - Identify superpixel boundaries by detecting pixels that have neighboring pixels with different labels.
  - Overlay the contours on the original image for visualization.
- **Superpixel Coloring**:
  - Optionally, assign random colors to each superpixel and blend them with the original image to visualize superpixel regions.

## 3.6. Execution Flow

- **Batch Processing**:
  - Iterate over a set of predefined values for $n$ and $nc$.
  - For each combination, process all images in the data directory.
  - Save the resulting images in an output directory named after the parameter combination.

## 4. Experimental Results

### 4.1. Experimental Setup

- **Datasets**: A set of images located in the `data` directory, representing various scenes with different complexities.
- **Parameters**:
  - **Number of Superpixels** ($n$): [100, 400, 900]
  - **Compactness Factor** ($nc$): [10, 20, 40, 80, 160]
  - **Maximum Iterations (`max_iter`)**: 20
- **Environment**: Python environment with OpenCV, NumPy, and tqdm libraries installed.

### 4.2. Results Overview

For each combination of $n$ and $nc$, images were processed, and the superpixel segmentation results were saved. The effects of varying parameters were observed as follows:

#### 4.2.1 Varying Number of Superpixels

- **Low $n$ (100)**:
  - Superpixels are large, capturing broad regions.
  - Suitable for images with large homogeneous areas.
  - Fine details and small objects may be lost.
- **High $n$ (900)**:
  - Superpixels are smaller, capturing fine details.
  - Better adherence to edges and complex structures.
  - Increased computational time.

#### 4.2.2 Varying Compactness Factor

- **Low $nc$ (10, 20)**:
  - Superpixels adhere closely to color boundaries.
  - Shapes are irregular, following image contours.
  - Effective in preserving edges and color variations.
- **High $nc$ (80, 160)**:
  - Superpixels are more regular and compact.
  - Shapes tend toward squares or circles.
  - May cross over subtle color boundaries.

### 4.3. Visual Examples

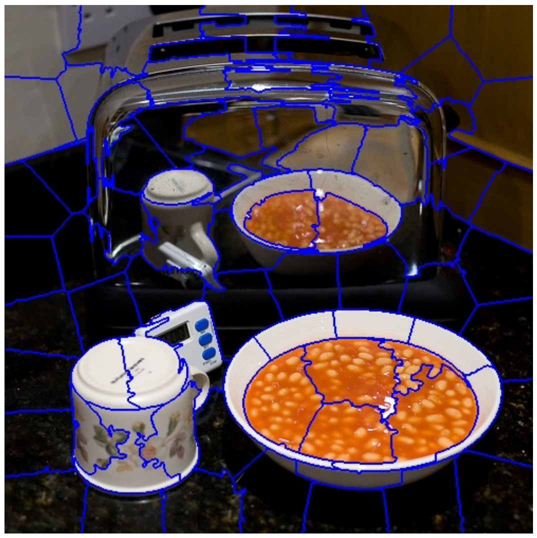

Figure 1. $n = 100$ , $nc = 40$.

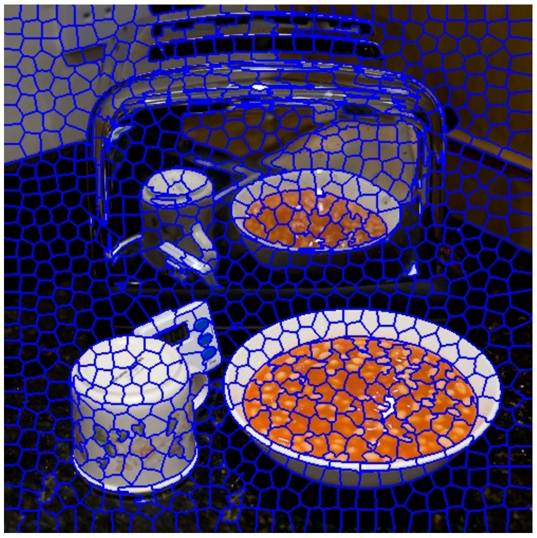

Figure 3. $n = 900$ , $nc = 40$.

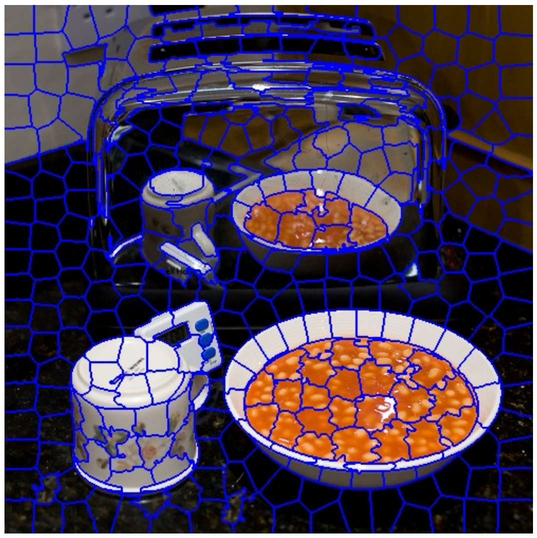

Figure 2. $n = 400$ , $nc = 40$.

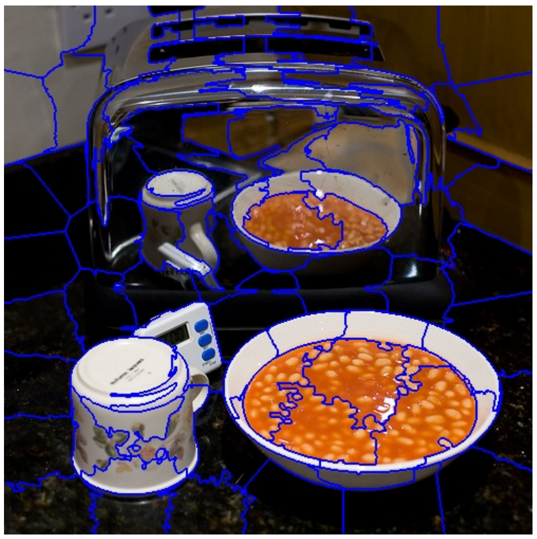

Figure 4. $n = 100$ , $nc = 20$.

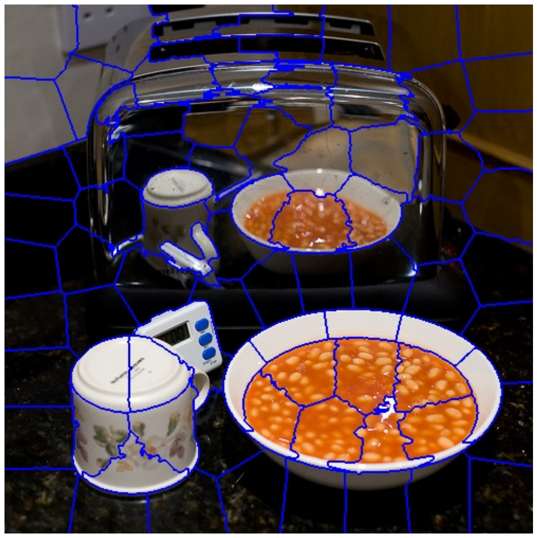

Figure 5. $n = 100$ , $nc = 80$.

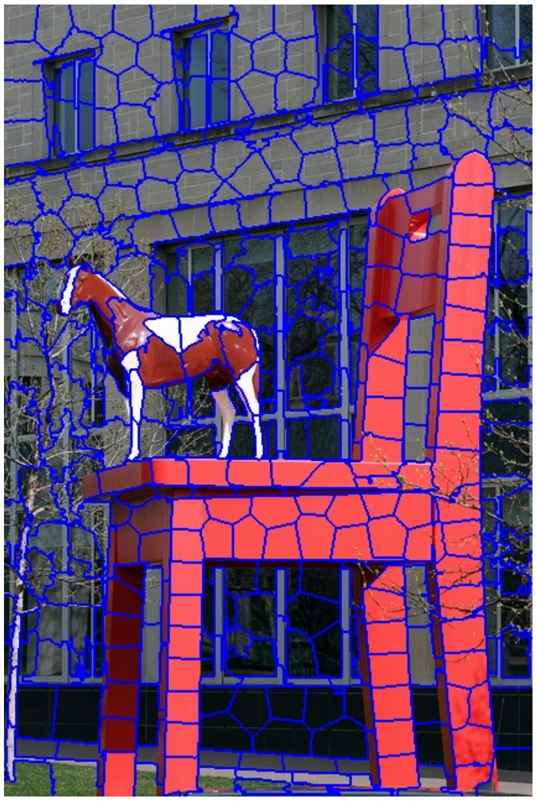

Figure 7. $n = 400$ , $nc = 40$.

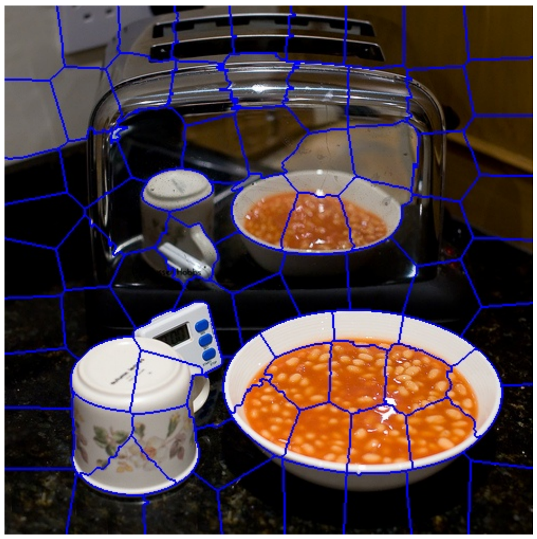

Figure 6. $n = 100$ , $nc = 160$.

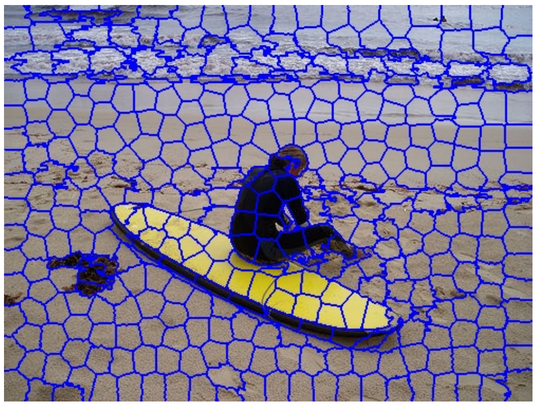

Figure 8. $n = 400$ , $nc = 40$.

### 4.4. Performance Analysis

- **Computational Efficiency**:
  - Processing time increases with higher $n$ due to more superpixels.
  - The algorithm remains efficient due to localized clustering.
- **Convergence**:
  - The algorithm typically converges within 10 iterations.
  - Labels stabilize after a few iterations, indicating effective clustering.

### 4.5. Observations

- **Edge Adherence**:
  - Lower $nc$ values improve adherence to edges but may result in irregular superpixel shapes.
- **Superpixel Regularity**:
  - Higher $nc$ values produce more regular superpixels but may overlook subtle color changes.
- **Parameter Sensitivity**:
  - The choice of $n$ and $nc$ significantly impacts segmentation quality.
  - Optimal parameters depend on image content and desired output.

## 5. Summary

### 5.1. Advantages

The implementation of the SLIC algorithm in this project offers several advantages, particularly due to specific design choices that enhance both performance and segmentation quality.

- **Efficient Computations with NumPy**:
  - The extensive use of NumPy for array and matrix operations significantly improves computational efficiency.
  - Vectorized computations and broadcasting capabilities of NumPy reduce the reliance on explicit loops, resulting in faster execution times.
  - NumPy's optimized numerical routines facilitate handling large images and datasets with high performance.
- **Improved Cluster Initialization through Local Gradient Minimization**:
  - Initial cluster centers are adjusted to positions of local minimum gradient within a $3 \times 3$ neighborhood.
  - This strategy prevents cluster centers from being placed on edges or noisy regions, which can lead to unstable clustering.
  - By starting at low-gradient positions, the algorithm ensures that superpixels are more homogeneous and better adhere to underlying image structures.
- **Enforcement of Superpixel Connectivity**:
  - The algorithm incorporates a post-processing step to enforce connectivity by merging small, isolated regions.
  - A flood-fill approach identifies connected components, and regions smaller than a predefined threshold are merged with adjacent larger regions.
  - This enforcement eliminates fragmented superpixels, resulting in more coherent and visually pleasing segmentation outcomes.

These advantages contribute to a robust and effective implementation of the SLIC algorithm, enhancing its applicability to a wide range of images and ensuring high-quality superpixel segmentation.

### 5.2. Shortcomings

- **Parameter Selection**:
  - Requires careful tuning of parameters for optimal results.
  - No one-size-fits-all; parameters may need adjustment for different images.
- **Irregular Superpixels**:
  - At low $nc$ values, superpixels may become highly irregular.
  - May not be suitable when regular superpixel shapes are desired.
- **Over-Segmentation**:
  - High $n$ values can lead to over-segmentation, resulting in unnecessarily small superpixels.

### 5.3. Future Work

- **Adaptive Parameter Selection**:
  - Develop methods to automatically select optimal parameters based on image content.
- **Integration with Higher-Level Tasks**:
  - Apply superpixel segmentation as a preprocessing step for object recognition or semantic segmentation.
- **Algorithm Optimization**:
  - Implement parallel processing to further improve computational efficiency.

## References

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels. 2010. 1

[2] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59:167–181, 2004. 1

[3] Alex Levinshtein, Adrian Stere, Kiriakos N Kutulakos, David J Fleet, Sven J Dickinson, and Kaleem Siddiqi. Turbopixels: Fast superpixels using geometric flows. *IEEE transactions on pattern analysis and machine intelligence*, 31(12): 2290–2297, 2009. 1

[4] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000. 1