MEAN SHIFT SEGMENTATION

In The Name of God

Rudmehr Aghakhani

STUDENT NO

4003663002

Mean Shift Segmentation Implementation with Python

Professor: Dr. Mahavsh

University of Isfahan

AGENDA

MEAN SHIFT SEGMENTATION

INTRODUCTION

In this project, I implemented a custom Mean Shift segmentation algorithm to group similar regions in an image based on spatial and color similarities. Unlike scikit-learn's pre-optimized implementation, I built the algorithm from scratch to better understand its functionality and explore potential optimizations. I used spatial and domain (color) kernels to ensure clustering was sensitive to both pixel proximity and color similarity, which helped preserve discontinuities and maintain sharp boundaries between distinct regions in the image.

To improve efficiency, I introduced sampling to process only a fraction of the pixels, significantly reducing the computational burden while retaining meaningful results. I used KDTree for fast nearest-neighbor searches, optimizing the computational performance for high-dimensional data. Parallel processing was also incorporated to accelerate pixel computations, making the algorithm scalable for larger images. Each design choice—sampling, KDTree, and parallelization—was implemented to address the high computational cost of the Mean Shift algorithm while preserving its segmentation quality. Ultimately, this approach provided an efficient and adaptable solution for image segmentation tasks.

IMPLEMENTATION

I implemented the Mean Shift segmentation algorithm to cluster pixels based on spatial and color similarities. The goal was to create an efficient and effective segmentation process while maintaining accuracy. Here is what I did and why:

1. **Feature Representation**:
   I represented each pixel as a 5D feature vector, combining its spatial coordinates (x, y) and color values (e.g., RGB). This allowed the algorithm to account for both pixel proximity and color similarity during clustering, ensuring effective segmentation.
2. **Spatial and Domain Kernels**:
   I used Gaussian kernels for both spatial and color domains to weigh the influence of neighboring pixels during clustering. This helped prioritize nearby pixels with similar colors, preserving edges and discontinuities in the image.

3. **Subsampling**:
   I processed only a fraction of the pixels (e.g., 0.1%) instead of the entire image. This significantly reduced computational overhead while still capturing the essential structure of the image for clustering.
4. **Mode Seeking with KDTree**:
   I built a KDTree to efficiently find neighboring pixels during mode-seeking iterations. This accelerated nearest-neighbor searches and improved the performance of the algorithm, especially with high-dimensional pixel data.
5. **Parallel Processing**:
   I incorporated parallel processing to compute pixel clustering across multiple cores. This reduced runtime, particularly for large images, by handling multiple pixel computations simultaneously.
6. **Label Assignment**:
   After clustering the sampled pixels, I assigned each pixel in the original image to the nearest cluster center (mode). This generalized the clustering results from the sampled pixels to the entire image, creating the final segmented output.
7. **Stopping Criteria**:
   I stopped iterations when the change in cluster centers (modes) fell below a threshold or a maximum number of iterations was reached. This ensured that the algorithm converged efficiently without unnecessary computations.

Each decision I made was to address the computational complexity of the Mean Shift algorithm while preserving its ability to produce high-quality segmentations. Let me know if you need further details or refinements.

OPTIMIZATION EXPLANATION

By combining **KDTree**, **sampling**, and **parallelism**, the algorithm optimizes the performance. **Sampling** reduces the number of pixels that need to be processed, while **parallelism** allows for processing multiple pixels at the same time, speeding up the computation. **KDTree** enables fast nearest-neighbor search, which is essential for efficiently updating pixel positions during the Mean Shift iterations. We explain each in the followings:

KDTree Explanation

The KDTree is a data structure used to efficiently store and query multi-dimensional data points, especially in scenarios that involve nearest-neighbor searches. In the context of this Mean Shift segmentation algorithm, the KDTree is used to speed up the search for neighboring pixels. Specifically, the `KDTree` helps to find the pixels within a given spatial radius (`spatial_bandwidth`) around the current pixel being processed.

Here's how it works:

- **Tree Construction**: The `KDTree` is built using the `sampled_pixels`, which include both spatial coordinates (x, y) and color values (R, G, B) of a subset of the image pixels. The KDTree organizes these pixels in a way that allows fast retrieval of nearby points.
- **Querying the Tree**: When processing a specific pixel, the KDTree allows for fast queries to find which other pixels are within a given radius from the current pixel. The function `tree.query_radius([point], r=spatial_bandwidth)` is used to find all pixels within a radius `r` of the `point` (current pixel). This is crucial for Mean Shift, where the algorithm iterates over neighboring pixels to update the position of the current pixel.

The use of KDTree significantly improves performance compared to a brute-force approach, where each pixel would need to be compared with every other pixel to find its neighbors. By using KDTree, we efficiently reduce the computational complexity of these nearest-neighbor searches, especially for large images.

Sampling

Sampling is the technique of selecting a subset of pixels from the original image to work with, rather than using all the pixels. In this case, a fraction (`subsample_fraction`) of the total pixels is randomly selected for processing. By reducing the number of pixels, **sampling** helps speed up the computation while maintaining the overall accuracy of the segmentation. This is particularly useful when working with large images, as it reduces the number of calculations required, making the process more efficient.

Parallelism

Parallelism refers to the process of executing multiple tasks simultaneously, which significantly reduces the overall time required for computation. In this algorithm, the Mean Shift process for each pixel is independent of others, so **parallelism** is implemented using the `ThreadPoolExecutor`. This allows multiple pixels to be processed at the same time, rather than sequentially. By utilizing multiple threads, the algorithm can process many pixels simultaneously, thus speeding up the overall Mean Shift process.

On next page I am going to explain the implemented code in Vs code environment.

CODE EXPALANATION

We have added comment to the code itself to explain each function functionality in detail.

CONFIGURATION

We must address three parameters value:

1) Epsilon (Convergence Value)
2) Spatial bandwidth
3) Range bandwidth

BANDWITH CONFIGURATION

Whether the spatial bandwidth should be larger or smaller than the range bandwidth depends on the nature of the image and the goals of segmentation. Here's a detailed explanation of different scenarios:

- Spatial bandwidth is larger than range bandwidth:
  - **Effect:** Clustering is primarily influenced by spatial proximity, even if the color differences within a region are small.
  - **Potential issue:**
    - Fine details or small color variations may be lost, as the focus is more on spatial closeness.

- Range bandwidth is larger than spatial bandwidth:
    - Effect: Clustering becomes more sensitive to color differences, even if pixels are spatially far apart.
    - Use case:
        - Works well for images where small color variations are critical, such as textured images or those with intricate details and high color variability.
        - Ideal for segmenting images with many small, distinct regions of color.

- Spatial and range bandwidths are equal:
    - Effect: A balance is achieved between spatial proximity and color information.
    - Use case:
        - Appropriate when there is no clear preference for prioritizing spatial or color factors, and a balanced approach is desired.
        - Often provides reasonable and consistent results in most cases.

EPSILON CONVERGENCE:

The choice of the **epsilon convergence value** (ϵ\epsilonϵ) in the Mean Shift algorithm determines the threshold for convergence during the iterative process, i.e., how small the shift between iterations must be for the algorithm to consider a point as having converged. Whether it should be **high** or **low** depends on the image and the desired outcome:

Low Epsilon Value (e.g.: epsilon < 1e-3ϵ)

- **Effect:** The algorithm performs more iterations per point until the shift is very small. This leads to highly precise convergence.
- **Use case:**
    - When fine-grained segmentation is needed, especially for images with detailed textures or small, distinct regions.
    - Ideal for high-resolution images where subtle variations matter.
- **Drawback:** Computationally expensive and slower, as more iterations are required for every point to converge.

High Epsilon Value (e.g., epsilon > 1e-3ϵ)

- **Effect:** The algorithm converges faster because it stops iterating when the shift is relatively large. This may lead to slightly less precise clustering.
- **Use case:**
    - For general-purpose segmentation, or when computational efficiency is more critical than extreme precision.
    - Suitable for images with smooth transitions or when rough segmentation is acceptable.
- **Drawback:** May result in under-segmentation, where fine details or small regions are missed.

FINAL CONFIGURATION:

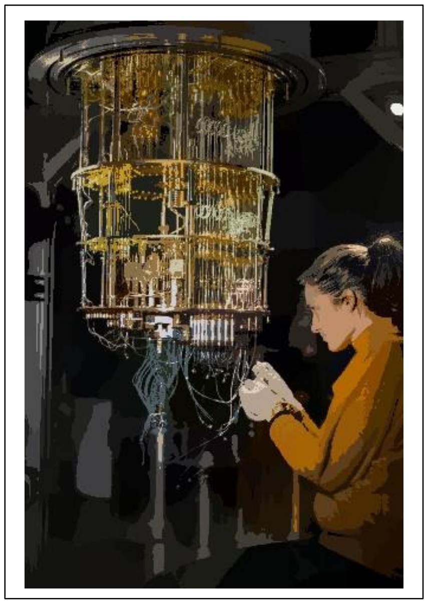
According to my picture which is shown below we need to consider higher value for color bandwidth than the spatial. Also for convergence criteria I have started from 1e-3 then reduce it to 1e-5.

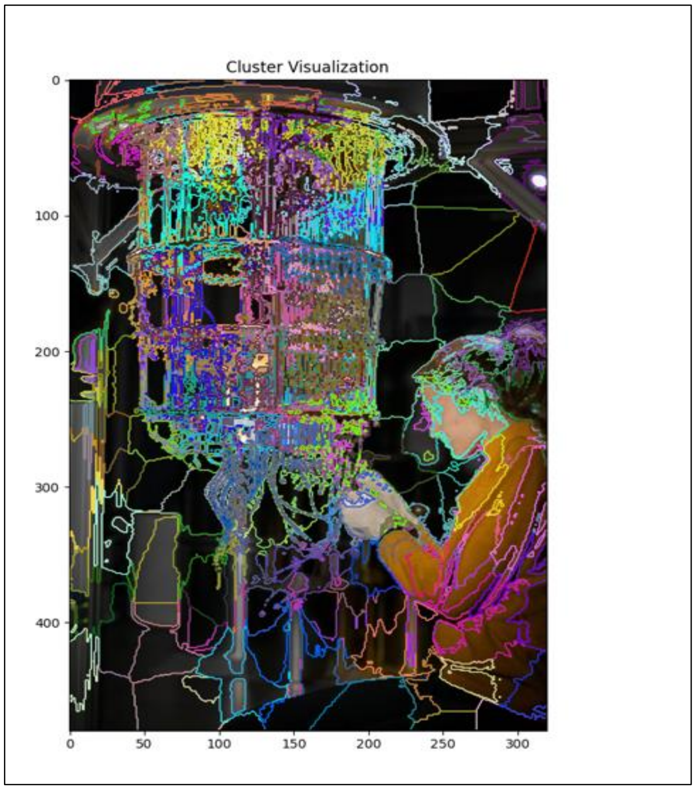
MEAN SHIFT SEGMENTATION

MEAN SHIFT SEGMENTATION

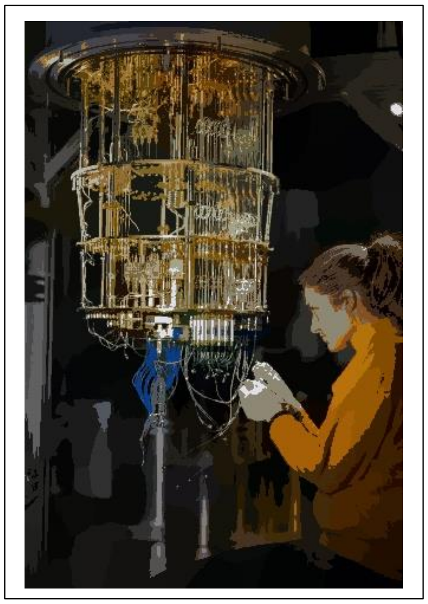
THE OUTPUT

EPSILON 1e-3:





EPSILON 1e-5:



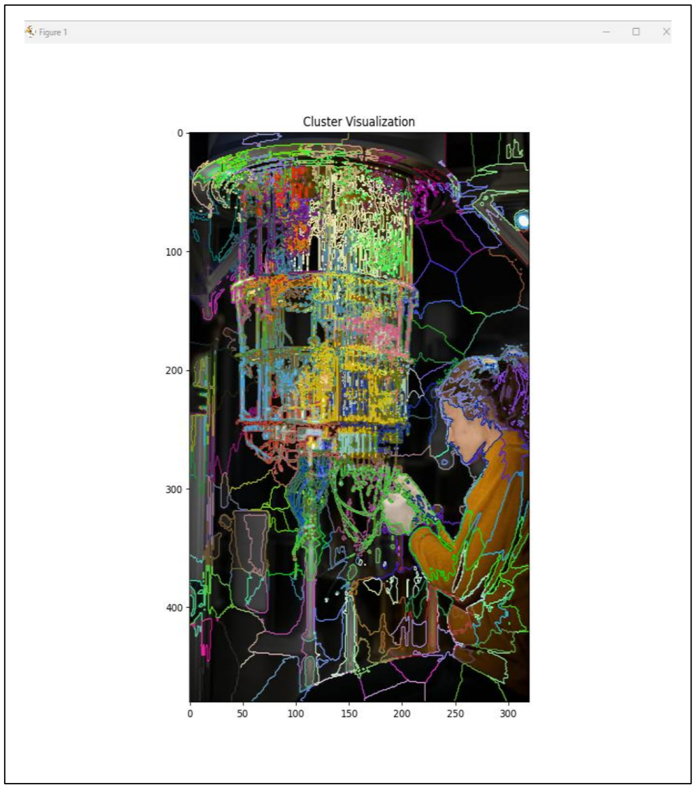


Figure 1
Cluster Visualization
0
100
200
300
400
0
50
100
150
200
250
300

THE END