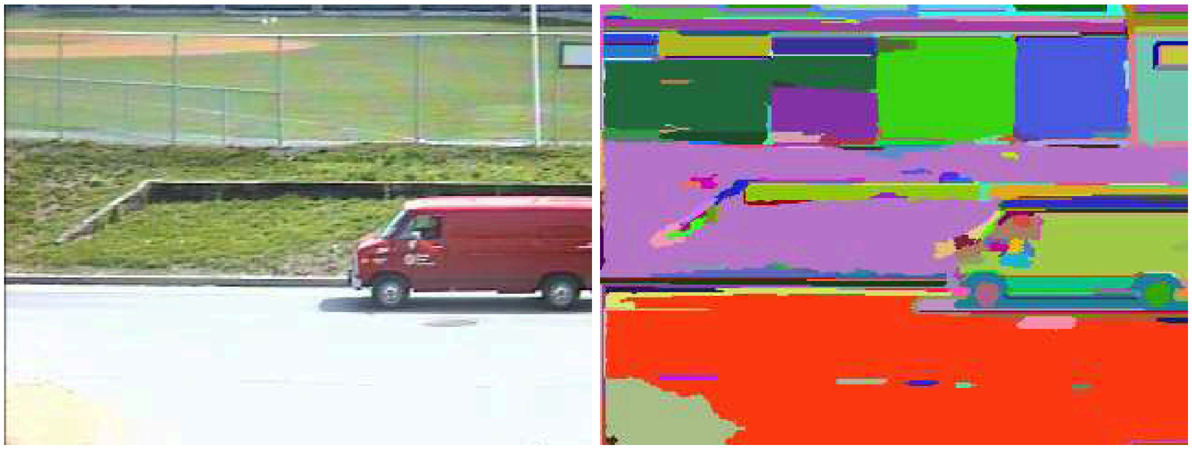
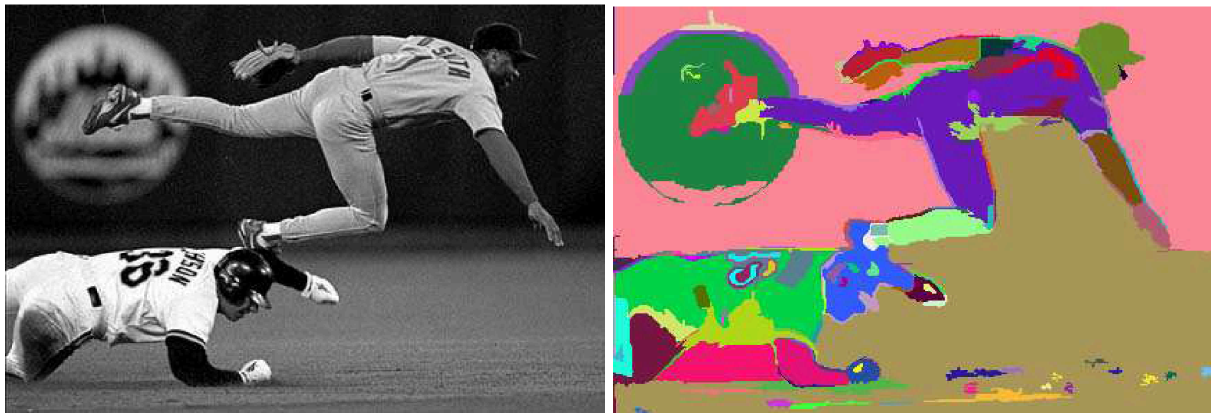
Image Segmentation

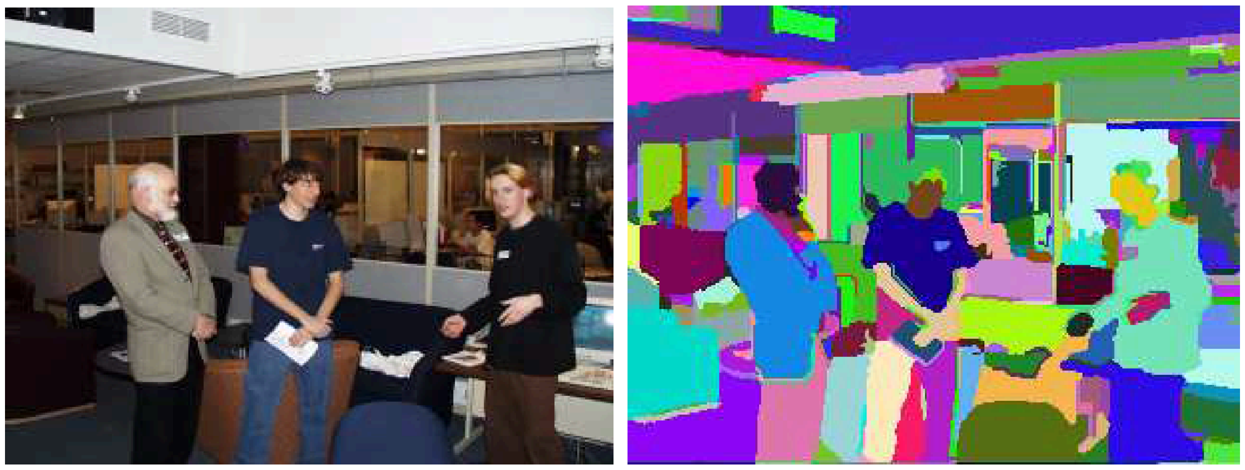
Sample Input/Output



A street scene (320 x 240 color image), and the segmentation results



A baseball scene (432 x 294 grey image), and the segmentation results



An indoor scene (image 320 x 240, color), and the segmentation results

Problem Definition

Image Segmentation

In digital image processing and computer vision, image segmentation is the process of partitioning a digital image into multiple image **segments**, also known as image **regions** or image **objects**.

obeying two high-level principles:

1. **Internal coherence** – pixels inside a region are visually similar.
2. **Boundary significance** – neighboring regions differ enough to justify a boundary.

Some Usages

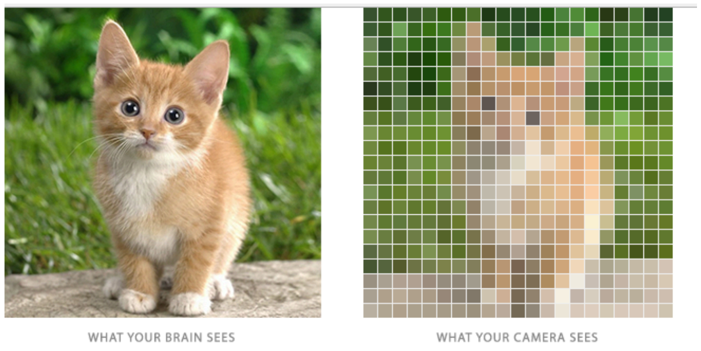
1. Pre-processing for deep-learning models.
2. Automatic figure–ground separation.
3. Interactive selection tools in photo-editing software.

Related Image Terminologies

Digital Image

It's an electronic snapshot taken of a scene or scanned from documents, such as photographs, manuscripts, printed texts, and artwork.

It's made of picture elements called pixels. Typically, pixels are organized in an ordered rectangular array. Each pixel has its own intensity value, or brightness which is represented in binary code (zeros and ones).



Color depth

Also known as **bit depth**, is the number of bits used to indicate the color of a single pixel. The size of an image is determined by two factors:

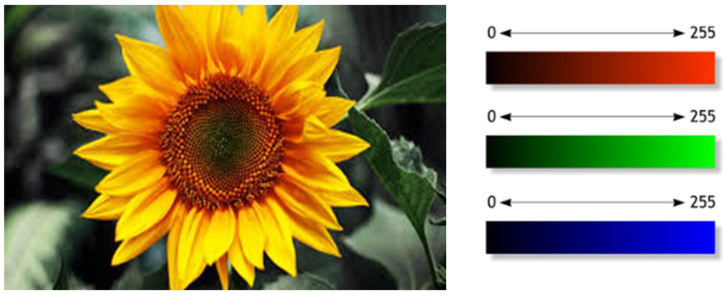
1. Dimensions: width and height of the 2D pixel array.
2. Color depth: number of bits per pixels

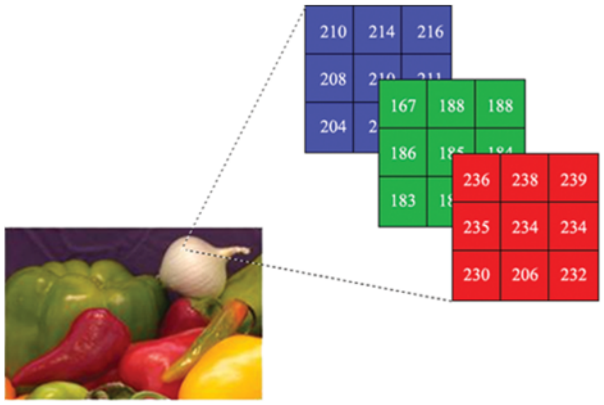
$$Image\ Size\ =\ Width \times Height \times Color\ Depth$$

Intensity

The intensity of a pixel is expressed within a given range between a minimum and a maximum value [Inclusive], based on the color depth of the pixel.

True Color images have intensity from the darkest (0) and lightest (255) of three different color channels, **R**ed, **G**reen, and **B**lue. Each channel has a range from **0 to 255** as shown in Figure below. So we need 8+8+8=24 bits to represent 1 pixel color which means we have 2^{24} = 16,777,216 different colors.





FIRST: GRAPH REPRESENTATION

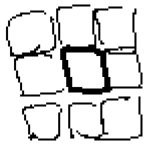
This section describes exactly how the input image is converted into a **weighted, undirected graph**—the data structure that every later stage will consume.

Pixels ⟶ Vertices

Let

$$G\ =\ (V,\ E),\ \ V\ =\ \{\, v_i \mid p_i\ is\ a\ pixel\ of\ the\ image\}.$$

Each vertex v_i represents one pixel p_i.



Neighbor Relationship (8-Connected Grid)

Two vertices v_i, v_j are connected by an edge $(v_i, v_j) \in E$ **iff** their pixels are direct neighbors in the standard 8-connected grid, its N, S, E, W and four diagonal neighbors.

This yields **at most eight edges per pixel**, so

$$|E| = m = \Theta(n),$$

where $n = |V|$ (number of pixels).

Edge-Weight Function

Weight is the absolute intensity difference:

$$w((v_i, v_j)) = |I(p_i) - I(p_j)|$$

Pre-processing: Apply a light Gaussian blur with $\sigma = 0.8$ *once* to the whole image before computing weights. This removes single-pixel digitization noise yet leaves visible content unchanged.

Color Images

Handle color pictures as three independent monochrome layers:

1. Run the entire graph build + segmentation pipeline on **R**ed, **G**reen, and **B**lue channels separately.
2. After segmentation, **intersect** the three label maps: two neighboring pixels belong to the same final region **only if** they are in the same component in *all* ***three*** single-channel results.

SECOND: REGION-FINDING STRATEGY

This section specifies how to decide whether a provisional **boundary between two regions** should be **preserved or eliminated**.

The rule relies on contrasting ***between-region*** dissimilarity **with *within-region*** variability, while automatically adapting to region size.

Main Idea

The method measures the evidence for a boundary between two regions by **comparing two quantities**:

1. one based on **intensity differences across the boundary**, and

2. The other is based on **intensity differences between neighboring pixels** within each region.

Intuitively, the intensity **differences across the boundary** of two regions are perceptually important if they are **large relative** to the **intensity differences inside** at least one of the regions. The algorithm is based on this idea.

Consider the image shown in the top left of the following Figure. Most people will say that this image has three distinct regions:

1. a rectangular-shaped intensity ramp in the left half,
2. a constant intensity region with a hole on the right half, and
3. a high-variability rectangular region inside the constant region.

This example illustrates **two perceptually important properties** that should be captured by a segmentation algorithm. The remaining parts of the Figure show the three largest regions found by the segmentation algorithm.

1. **Widely varying intensities should not alone be judged as evidence for multiple regions.**
   Such wide variation in intensities occurs both in the ramp on the left and in the high variability region on the right. Thus, it is not adequate to assume that regions have nearly constant or slowly varying intensities.
2. **The three meaningful regions cannot be obtained using purely local decision criteria.**
   This is because the intensity difference across the boundary between the ramp and the constant region is actually smaller than many of the intensity differences within the high variability region. Thus, in order to segment such an image, some kind of adaptive or non-local criterion must be used.

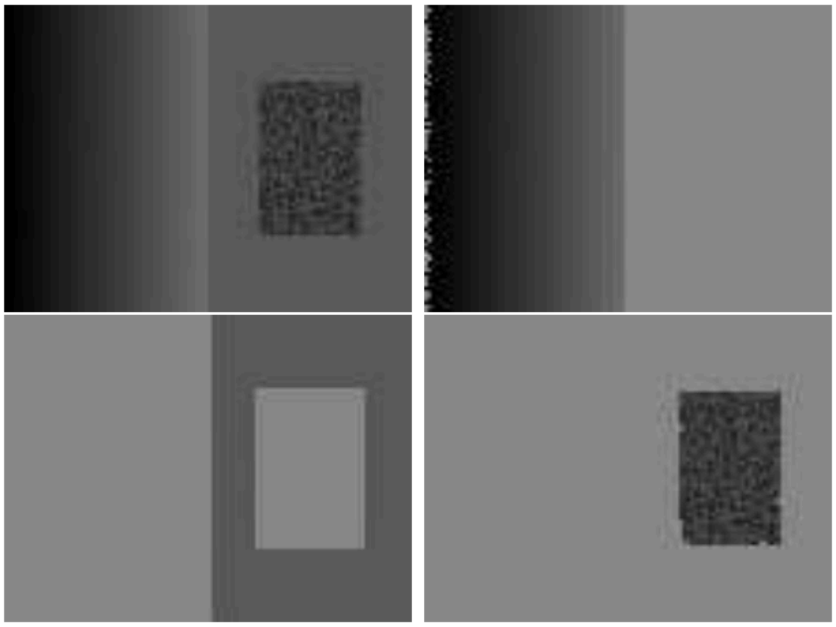


Figure: A synthetic image with three perceptually distinct regions, and the three largest regions found by the segmentation method (image 320 x 240 pixels; algorithm parameters σ = 0.8, k = 300).

Internal Difference

The internal difference of a component $C \subseteq V$ is defined as the **largest weight edge** that is **necessarily** to **keep the component connected**. That is, without this edge, the component becomes disconnected. In other word, **larger edges** are **not necessary** to keep the component connected while **smaller edges** are necessary to keep it connected.

$$(C) = \max_{e \in C} w(e)$$

Difference Between Two Components

The difference between two components $C_1, C_2 \subseteq V$ is defined to be the **minimum weight** edge connecting the **two components**. That is,

$$Dif\left(C_1, C_2\right) = \min_{v_i \in C_1,\ v_j \in C_2, (v_i, v_j) \in E} w((v_i, v_j))$$

If there is no edge connecting C1 and C2 we let $Dif(C_1, C_2) = \infty$.

Region Comparison Predicate

The region comparison predicate evaluates if there is evidence for a boundary between a pair or components by checking if the **difference between the components**, $Dif(C_1, C_2)$, is **large relative** to the **internal difference** within at least one component, $Int\left(C_1\right)$ and $Int\left(C_2\right)$.

A **threshold function** is used to control the degree to which the difference between components must be larger than minimum internal difference. The pairwise comparison predicate is defined as,

$$D(C_1, C_2) = \begin{cases} \text{true} & \text{if } Dif(C_1, C_2) > MInt(C_1, C_2) \\ \text{false} & \text{otherwise} \end{cases}$$

- **D = true**: Sufficient evidence of a real boundary ⟶ keep regions separate.
- **D = false**: Boundary not strong enough ⟶ regions may be merged.

where the minimum internal difference, $MInt$, is defined as,

$$MInt(C_1, C_2) = \min(Int(C_1) + \tau(C_1),\ Int(C_2) + \tau(C_2)).$$

The threshold function τ controls the degree to which the difference between two components must be greater than their internal differences in order for there to be evidence of a boundary between them (D to be true). For small components, $Int(C)$ is not a good estimate of the local characteristics of the data. In the extreme case, when $|C| = 1$, $Int(C) = 0$. Therefore, we use a threshold function based on the size of the component,

$$\tau(C) = k/|C|$$

where $|C|$ denotes the size of C, and k is some constant parameter.

Project Requirements

Provided Implementation

1) Image loading code.
2) Image displaying code.
3) Gaussian filter code.

Required Implementation

Requirement	Performance
1. Graph construction + weight computation	**Time:** should be **bounded by O(N^2)**, N is one image dimension (width/height)
2. Image segmentation	**Time:** should be **bounded by O(M log M)**, M is number of pixels.
3. Image segments visualization	

Input

1. Image (2D array of pixels)
2. K: constant parameter to control the threshold

Output

1. **Text file contains the following**:
    1. Number of segments (regions).
    2. Size (number of pixels) for each segment, **sorted in descending order**, one per line.
2. **Image that shows your segmentation**: every region is colored with a distinct color and combined with the original picture so the detected regions are easy to see.

Test Cases

Sample Test:

- **Goal:** test the correctness
- **Given:** sample images with the expected O/P

Complete Test:

- **Goal:** test efficiency (besides correctness)
- **Given:** 3 levels
    1. Small: image size O(100's KBs)
    2. Medium: image size O(10's MBs)
    3. Large: image size O(100's MBs)

Deliverables

Implementation (60%)

1. Graph construction
2. Image segmentation
3. Image segments visualization

Document (40%)

1. Code of graph construction.
2. Code for Image segmentation algorithm.
3. Code for image segments visualization.
4. Detailed analysis of the above codes.

Allowed Codes

1. C# Startup Template to open and display the images. Refer to **Appendix: Template Code Description** for more details.

2. Data structures code other than the graph (either C# built-in data structures or other open-source data structures). You **MUST understand** and **analyze** it!

Important Delivery Notes

1. **Test Cases & Output:**
   - The project should support the option to **browse any image** from the disk.
   - Write the number of segments & size of each one in a **text file** with the **same format** specified in the project document.
   - **Visualize** the segmentation result using separate color for each region.
2. **Documentation:**
   - You must get the **document printed** (softcopy is not accepted).
   - **Analysis** must be **detailed** (don't write the whole complexity only).
   - In case of using any **built-in/external data structures** or **functions**, you must **analyze them**.

Milestones

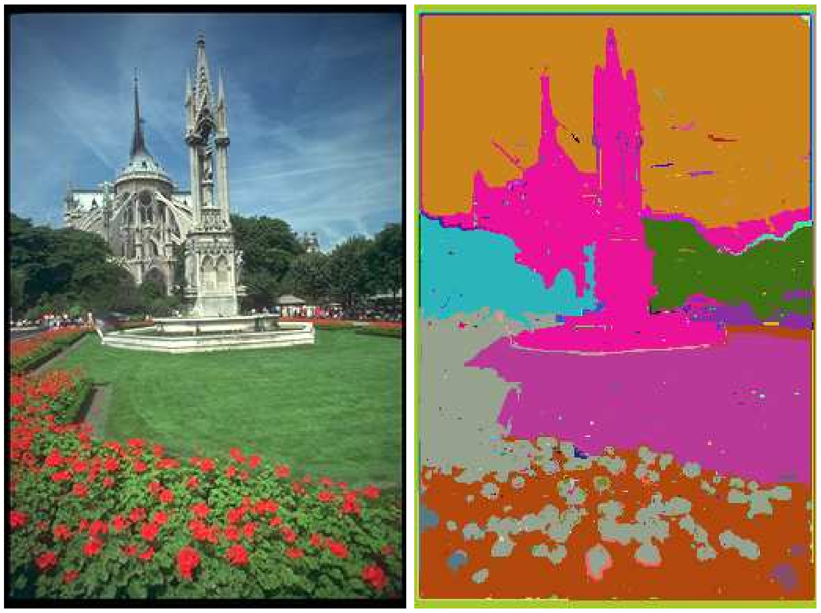
	Deliverables	**Due to**
Milestone1	1. Graph construction 2. Image segmentation algorithm 3. Image segments visualization 4. Documentation	Final Delivery [LAB EXAM WEEK]
o **MUST** deliver the required tasks and **ENSURE** it's worked correctly o **MUST** deliver in your scheduled time (TO BE ANNOUNCED)		

BONUSES

Bonus B1 — Nearest-Neighbor Graph Segmentation

Instead of linking only the eight surrounding pixels, map every pixel to a point in the 5-D feature space (x, y, R, G, B) and build a graph that connects each point to its **k nearest neighbors in that space** (k≈10 works well).

Edge weights are the Euclidean distances between these feature points. Running the same region-merging algorithm on this graph lets the algorithm group **visually similar but spatially distant pixels**—for example, as shown in the figure, scattered red flowers can merge into one logical region even if pure grid connectivity would keep them separate.



Segmentation using the nearest neighbor graph can capture spatially nonlocal regions (σ = 0.8, k = 300)

To keep the graph linear in size and the runtime near O(n log n), fetch neighbors with any fast approximate-nearest-neighbor routine; the classic reference is:

> *S. Arya and D. M. Mount. Approximate nearest neighbor searching. Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 271-280, 1993.*

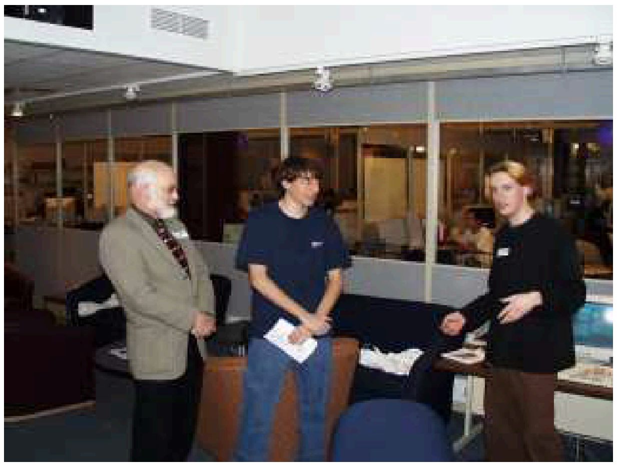
Bonus B2 — Interactive “Click-to-Merge” UI

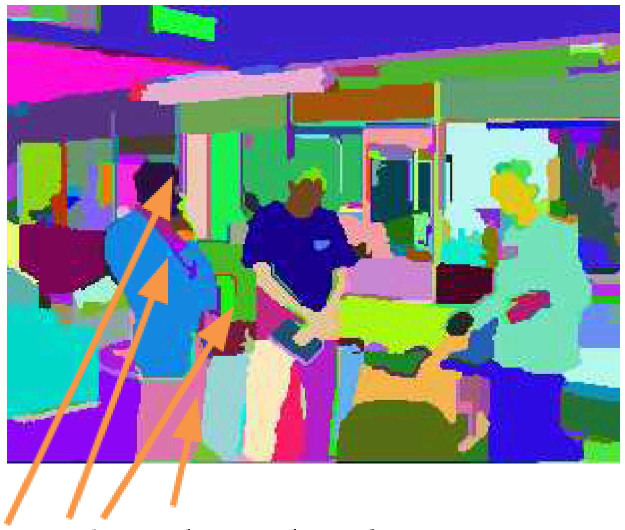
After the algorithm produces its color-coded segmentation preview, display that overlay in a simple GUI window.

Allow the user to **click on any number of colored regions to select them and press a single key (or button) to merge the selection into one segment**.

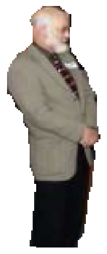
When the merge is confirmed, the program refreshes the view and displays a new image where **ONLY** the **newly merged segment is rendered with its original pixel colors**.

This tool lets you quickly repair cases where automatic segmentation splits an object that should stay whole. For example, the person in the left was initially segmented into many regions (face, hands, suit jacket, pant,… etc). By merging these regions, we can segment the person as a single region.





By selecting these regions and merge them, the person will be segmented correctly



Appendix: Template Code Description

C# Code contains **`ImageOperations`** class with the following functionalities:

1. Open image & return it in a 2D array of type `RGBPixel`[1]`[,]`

```
RGBPixel [,] OpenImage(string ImagePath)
```

2. Get width and height of the image matrix

```
int GetHeight(RGBPixel [,] ImageMatrix)
int GetWidth(RGBPixel [,] ImageMatrix)
```

3. Display an image on a given `PictureBox` control

```
    void DisplayImage(RGBPixel [,] ImageMatrix,PictureBox
PicBox)
```

4. Apply Gaussian filter on an image

```
RGBPixel [,] GaussianFilter1D(RGBPixel [,] ImageMatrix,
                int filterSize, double sigma)
```

[1] `RGBPixel` is a structure defined in the code to hold the Red, Green, Blue values of each pixel