Image Processing and Imaging

Image Segmentation

Dominik Söllinger
Fachbereich AIHI
Universität Salzburg

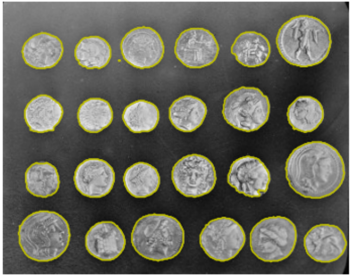
Wintersemester 2022/23

Outline

What is segmentation?

Image Segmentation

- Image segmentation is the division of an image into **regions** or **objects**
- Spatially close regions very often can be combined to an **object**
- Objects are typically what we are interested in (depends on the application). The rest of the image is background.



Complete Segmentation

- Divides an image into non-overlapping regions that match to the real world objects
- Nowadays this is very often tackled with deep learning (DL) based techniques (see **semantic segmentation** or **instance segmentation**)
- Simple tasks can be approached with simple (non-DL based) techniques
  - Text with letters and numbers
  - Objects in front of uniform background or high contrast

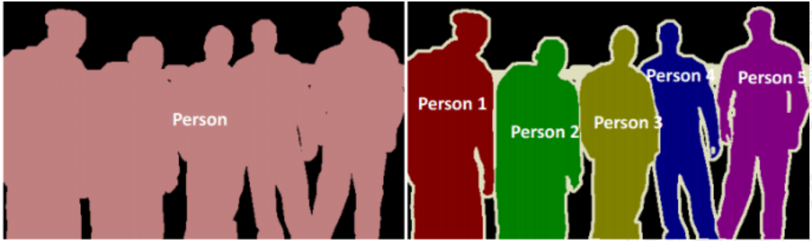


Figure: Semantic segmentation (left) vs. Instance segmentation (right)

Complete and partial image segmentation (2)

Partial Segmentation:

- Regions which are homogeneous with respect to some criterion are identified
- These regions do NOT necessarily correspond to real-world objects
- In many cases, over-segmentation occurs, i.e. to many regions are found

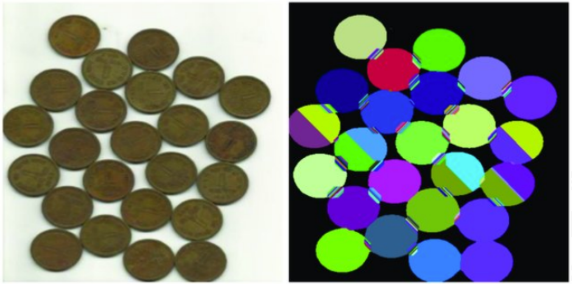


Figure: Example of oversegmentation

Categories of segmentation-based techniques

Segmentation techniques can be grouped into three categories:

1. **Thresholding-based**: Use global information of the image or parts of it (e.g. the histogram).
2. **Edge-based**: An edge filter is applied to the image. Closed edge-chains are generated as object borders
3. **Region-based**: Group together pixels which are neighbours and have similar values and splitting groups of pixels which are dissimilar in value

- The latter two techniques solve a dual problem:
  Each region can be described by its closed border curve and each curve describes a region which is enclosed by the curve

Outline

1. Introduction
2. Thresholding-based segmentation
   - Thresholding Variations
   - Threshold Selection
   - Optimal Thresholding Techniques
3. Region-based segmentation
   - Region Growing
   - Region Merging and Splitting
   - K-Means Clustering Segmentation

Thresholding

- Great due to its **simplicity** and **computational speed**
- A threshold T is determined to separate objects and the background
- Input image $f(x, y)$ is transformed into a binary segmented output image $g(i, j)$ as follows:

$$g(i,j) = \begin{cases} 1 & f(i,j) \geq T \\ 0 & f(i,j) < T \end{cases}$$

- If T is constant over the entire image, we call refer to it as **global thresholding**
- Thresholding is a suited approach if objects are not joined and the object's gray-scale values are different from background's gray-scale values

Outline

Threshold Variations (1)

- Using a one fixed threshold is successful in rare cases only
- **Variable thresholding** is often more successful. The value of T is changed dependent on a local image region's characteristics.

Thresholding Variations (2)

Using a single threshold T typically works well if the distribution of the is bimodal.
If the histogram has, for example, three dominant modes (i.e., cause by two types of light objects on a dark background) we require **multiple thresholds**.

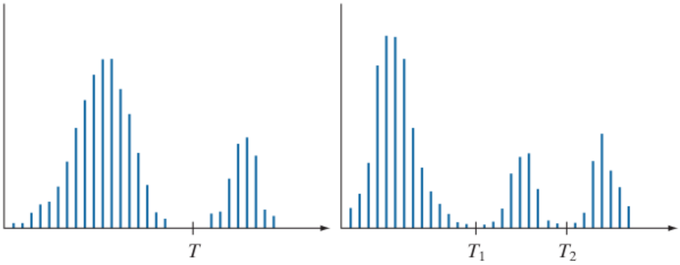


Figure: Intensity histograms that can be partioned by a single threshold (left) and by dual thresholds (right)

Band Thresholding:

- Grayscales values in a specific grayscale range are determined to be object pixels
- Example application: CT imaging — Different tissue types cause different grayscale values

$$g(i,j) = \begin{cases} 1 & f(i,j) \in D \\ 0 & \text{otherwise} \end{cases}$$

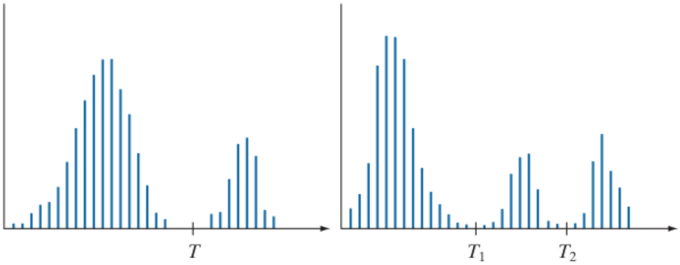


Figure: Intensity histograms that can be partioned by a single threshold (left) and by dual thresholds (right)

Thresholding Variations (4)

Multi Thresholding: Similar to band thresholding but the output image is not binary

$$g(i,j) = \begin{cases} 1 & \text{for } f(i,j) \in D_1 \\ 2 & \text{for } f(i,j) \in D_2 \\ 3 & \text{for } f(i,j) \in D_3 \\ \ldots \end{cases}$$

Semi Thresholding: Removes the background but keeps the grayscale information in the objects

$$g(i,j) = \begin{cases} f(i,j) & \text{for } f(i,j) \geq T \\ 0 & \text{otherwise} \end{cases}$$

Outline

The crucial question — How to choose a threshold?

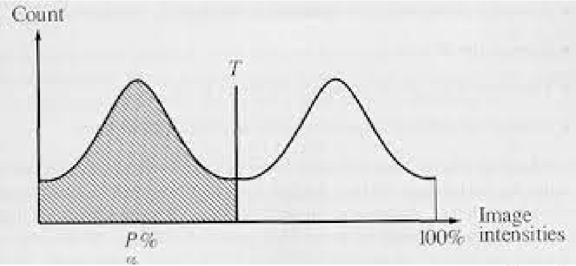
In general we can say, the more **priori information** the better.

Illustrive examples:

- Shape of the object region
- Position or orientation of the object
- Known initial and final point of the boundary
- Relation of the region considered to other regions with required properties (e.g., above or inside)

- **Assumption:**
  We have dark objects against a light background and can estimate the area/size of the objects present in the image (e.g., written letters on a sheet)

- If p % of the image is occupied by the objects of interest, choose the threshold T so that the p % darkest pixels are classified as object pixels.
- **How?** Compute the commulative histogram distribution. Select T such that $1/p$ of the image is $\leq T$.



Optimal thresholding

- Optimal thresholding selects a threshold value that is statistically optimal
- If the histogram is **bimodal**, the threshold is selscted to be the minimal value between the two extrma
- If the histogram is **multimodal**, the thresholds can be selected between two corresponding maxima.

Attention: A biomodal histogram does not guarantee a correct segmentation.
Example: 50% B/W pixel mixed or concentrated on one image side result in an identical bimodal histogram.

Concept of Optimal Thresholding

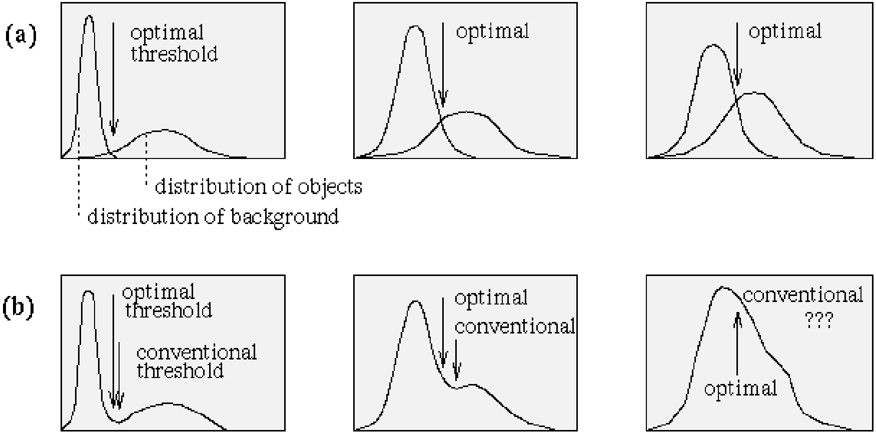


Figure: Concept of optimal thresholding, first row represents the individual distribution, second row is the combined distribution.

Assumption: Existence of regions with two dominant grayscale values.

Algorithm:

1. Select an initial estimate for the global threshold T^0.
2. Produce two regions by global thresholding with T^0: Foreground and object region
3. Compute the mean intensity value μ_B^t of the background and object region μ_O^t.

$$\mu_B^t = \frac{\sum_B f(i,j)}{\text{number of background pixel}}$$
$$\mu_O^t = \frac{\sum_O f(i,j)}{\text{number of object pixel}}$$

4. Compute a new threshold value midway between μ_B and μ_O.

$$T^{t+1} = \frac{\mu_B^t + \mu_O^t}{2}$$

5. Repeat Steps 2 through 4 until the difference between values of T^{t+1} in successive iterations is smaller than a predefined value ΔT.

How to choose the initial threshold value?

- Assuming that the corner pixels contain background values → Choose the average corner pixel value
- Choose the global average pixel value
- **Note:** The initial intensity value must be greater than the minimum an less than the maximum intensity value.

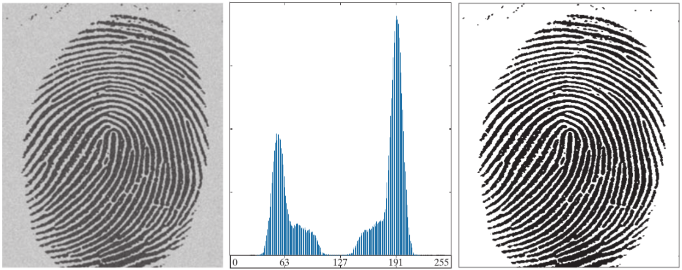


Figure: Example of segmentation using the preceding iterative algorithm. Application of the algorithm resulted in the threshold T=125 4 after three iterations, starting with T equal to the average intensity of the image, and using $\Delta T = 0$.

Optimal thresholding using Otsu's method (1)

Idea: View thresholding as statistical-decision theory problem → Minimize the average error incurred in assigning pixels to two or more regions (also called classes).

How?

By choosing a threshold T that **minimizes the intra-class variance** σ^2_{intra}, defined as a **weighted sum of variances** of the two classes.

$$\sigma^2_{intra}(T) = q_O(T)\sigma^2_O(T) + q_B(T)\sigma^2_B(T)$$

where

$\sigma^2_O(T)$ ... Threshold-dependent variance the object region
$\sigma^2_B(T)$ ... Threshold-dependent variance of the background region
$q_O(T)$, $q_B(T)$ ... Weight terms

The **weight terms** are chosen based on the **probability of the class** occuring.

Assuming that $P(i)$ denotes the normalized histogram, $q_O(T)$ and $q_B(T)$ can simply be computed as follows:

$$q_O(T) = \sum_{i=0}^{t-1} P(i) \qquad \text{and} \qquad q_B(T) = \sum_{i=T}^{L-1} P(i)$$

Calculating the class variance $\sigma_O^2(T)$ where $P_{O,T}(i)$ is the probability distribution of class O:

$$\begin{aligned}
\sigma_O^2(T) &= \sum_{i=1}^{T} (i - \mu_O(T))^2 P_{O,T}(i) \\
&= \sum_{i=1}^{T} (i - \mu_O(T))^2 P(i|O) \qquad \text{[Apply Bayes rule]} \\
&= \sum_{i=1}^{T} (i - \mu_O(T))^2 P(O|i) P(i) / P(O)
\end{aligned}$$

Finally we obtain for $\sigma_O^2(T)$:

$$\sigma_O^2(T) = \sum_{i=1}^{T} (i - \mu_O(T))^2 \frac{P(i)}{q_O(t)}$$

Similary, we can derive the **formula for the object mean** $\mu_O(T)$:

$$\mu_O(t) = \sum_{i=1}^{t} i \frac{P(i)}{q_O(t)}$$

(background mean and variance by analogy)

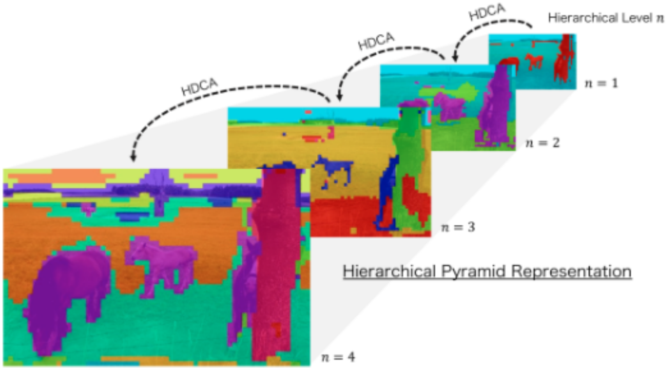
Last but not least, all we need to do is just **run through the full range of** T **values** and **pick the value that minimizes** $sigma_{intra}^2$.

Final note: Having to do this computation for the entire range of pixel values is not very efficient computationally. However, there also exists a **fast recursive approach**.

The recursive approachs, exploits the relationship between the intra-class σ^2_{intra} and inter-class σ^2_{inter} variances $\rightarrow$ Minimizing the intra-class variance is the same as **maximizing inter-class variance**.

There exists a fast recursive algorithm to maximizing inter-class variance (see script for more details).

Basic idea: View the image as a pyramidal data structure. Identify regions in the low resolution image and refine them in the higher resolutions



Advantage: Robustness against noise, since the initial segmentations start with a significantly smoothed image.

Variants:

- Segment the low-resolution image. In the next higher resolution pixels close to the region border are re-assigned to object(s) or background (depending on a similarity criterion) $\rightarrow$ Repeat until the highest resolution
- Apply a *significant pixel detector* (*) in the lowest resolution which identifies pixels different to their neighbourhood.

  The corresponding image part in the full-resolution image is thresholded with T being between the gray-scale of the significant pixel and the average of the other 8-neighbours (this is done locally with different thresholds for different regions).

  (*) e.g, a 3x3 masks which indicates if the central pixel is different from its neighbors

Further variants - Variable Local Thresholding (1)

Basic idea: Adapt the threshold for every point in the image based properties of the local neighborhood (e.g., based on its mean and standard deviation).

Illustrative approach:
Let m_{xy} and σ_{xy} denote the mean and standard deviation of a pixel (x, y)'s local neighborhood.

For instance, we can compute the segmentation mask $g(x, y)$:

$$g(x,y) = \begin{cases} 1 & f(x,y) \geq a\sigma_{x,y} \text{ AND } f(x,y) \geq b\mu_{x,y} \\ 0 & \text{otherwise} \end{cases}$$

where a and b are nonnegative constants.

(a) (b) (c)

Figure: Segmentation of a yeast image. (a) Original yeast image. (b) Yeast image segmented using a global dual thresholding approach. The mid-gray regions on the right side of the image were not separated properly. (c) Yeast image segmented using variable thresholding ($a = 30$ and $b = 1.5$). All regions are nicely separated.

Outline

Region-based techniques

- **Idea:** Partition the image into regions of maximal homogeneity
- Primary applied to noisy images or in circumstances where the difference between objects is not "just" the luminance (e.g. they exhibit a different structures — texture)

Homogeneity:

- We need a metric for measuring the similarity between pixels/regions
- For example: average gray-value, shape of a local histogram, texture properties
- Target regions should exhibit the following property (apart from their zero intersection):

$$
\begin{aligned}
H(R_i) &= \text{True} \qquad && i = 1, \ldots, S \\
H(R_i \cup R_j) &= \text{False} \qquad && i \neq j \text{ AND } R_i \text{ adjacent to } R_j
\end{aligned}
$$

  S ... number of regions, $H(R_i)$ is a binary evaluation of homogeneity of R_i
- This means that regions are homogeneous and maximal (i.e. if they would be larger, homogeneity is lost)

Outline

Idea: Group pixels or subregions into larger regions while the homogeneity criterion is satisfied.

Compare one pixel with its neighbors. If the homogeneity criterion is satisfied, the pixel belong to the same region.

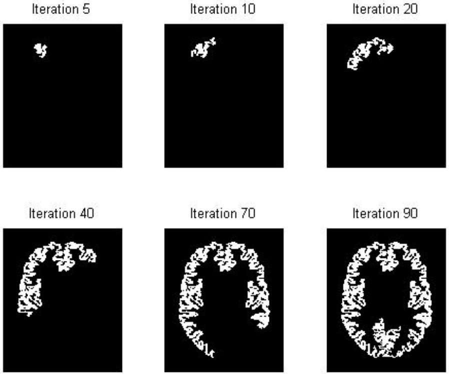


Figure: Illustrative example of region growing

Region Growing Algorithm:

1. Choose seed pixel(s)
2. Check neighbouring pixels following a chosen strategy and add them to the region if they are "similar" to the seed pixel
3. Repeat (2) and (3) for the newly added pixels; stop if no more pixels can be added

Outline

Region Merging and Splitting (1)

Region Growing requires seed points.

Instead, **Region Splitting and Merging** subdivides the entire image into a set of disjoint regions and then merges/splits those regions.

Region Merging and Splitting (2)

Region Splitting:
The goal is to recursively split the image into regions until all regions satisfiy the homogeneity requirement.

Algorithm:

1. Start with the whole image. If the homogeneity criterion is not satisfied, split the image into four disjoint quadrants.
2. Keep subdividing the quadrants until the homogeneity is satisfied for all quadrants of them.

The splitting technique has a convenient representation in the form of structure called **quadtree**.

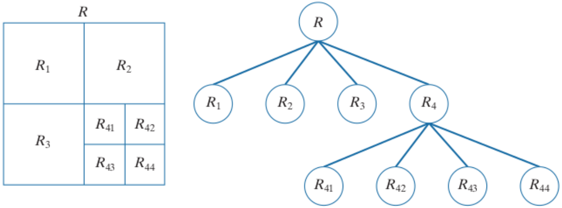


Figure: Left: Partitioned image. Right: Corresponding quadtree

Region Merging and Splitting (3)

Note that if only splitting is used, the final partition often contains adjacent regions with identical properties. We can combine those regions with **region merging**.

Region Merging Algorithm:

1. Start with a set of regions which satisfy the homogeneity requirement.
2. Keep merging regions until the homogeneity requirement is no longer satisifed.

Region Merging and Splitting (4)

Splitting/merging techniques differ in terms of their initial segmentations and the different criteria for homogeneity.

Improvements: Additional employment of edge information

- Neighbouring regions are merged if a significant share of their common border consists of weak edges
- Result of edge relaxation can be used to determine if an edge is weak or not

Region Splitting and Merging Comparison (1)

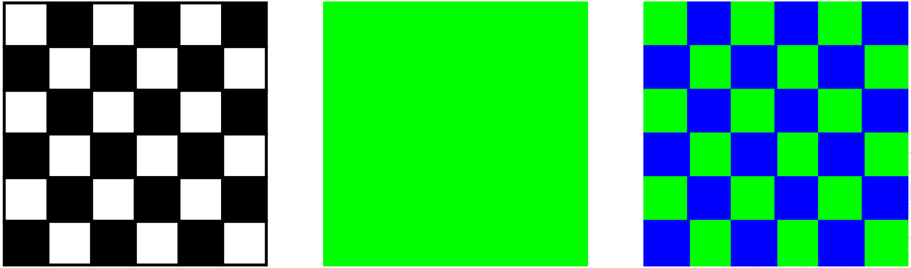
Remark: Region splitting does not result in the same segmentation even if the same homogeneity criteria are used.

Example: Segmentation of the chessboard pattern using either region splitting or merging. We consider the homogeneity criterion $H(R_i)$.

$$H(R_i) = \begin{cases} TRUE & \text{if the region contains the same number of B/W cells} \\ FALSE & \text{else} \end{cases}$$

Region Splitting and Merging Comparison (2)

$$H(R_i) = \begin{cases} TRUE & \text{if the region contains the same number of B/W cells} \\ FALSE & \text{else} \end{cases}$$



source splitting merging

Figure: Left: Non-segmented chessboard image. Center: Region splitting (the upper pyramid level is homogeneous, no splitting possible). Right: Region merging (lowest pyramid level initially consists of inhomogeneous regions, no merging possible)

Split-and-Merge Algorithm

In practice, typically a combination of splitting and subsequent merging is applied. The employed data structure is often a quad-tree.

1. Define an initial segmentation into regions, a criterion for homogeneity and a pyramidal data structure
2. In case a region in the data structure is not homogeneous, it is split into its four children regions; In case four regions corresponding to the same parent can be merged, they are merged. If no further region can be processed, GOTO 3)
3. In case two neighbouring regions (either in different levels of the pyramid or with different parent nodes) can be merged according the criterion for homogeneity they are merged
4. Regions being too small are merged with the most similar neighbouring region

Outline

K-Means Clustering Segmentation (1)

Idea: Image segmentation can be viewed as a clustering task. "Similar" pixels should be clustered into one region

Procedure:

1. Represent each pixel in the image with a vector (e.g. intensity, colour and location, texture descriptors, etc.)
2. Choose distance weights (which vector component is more important than others, e.g. color vs. location)
3. Apply k-means clustering
4. Pixels belong to the segment corresponding to cluster centers

K-means Clustering Algorithm:

Let $\{x_1, \ldots, x_n\}$ be a set of pixels (observations), where each observation x is a vector and has the form $x = [x_1, ..., x_D]^T$. For instance, in case of a grayscale image, x is a scalar.

The aim is to partition the n pixels into K sets $S = \{S_1, S_2, \ldots, S_K\}$ in a way that the **within-cluster sum of squares (WCSS) is minimized**.

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{x_j \in S_i} ||x_j - \mu_i||^2$$

where μ_i the mean or centroid of pixels in cluster S_i

Lloyd Algorithm

1. Specify an initial set of means μ_i.
2. Assign samples to clusters: Each sample should be assigned to the cluster set whose mean is the closest.

$$S_i^{(t)} = \{x_j : ||x_j - m_i^{(t)}||^2 \leq ||x_j - m_{i^*}^{(t)}||^2 \; \textit{for all } i^* = 1, ..., k\}$$

3. Update the cluster centers (means)

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

4. Repeat (2) and (3) until the WCSS is less than certain threshold T.

K-Means Clustering Segmentation (4)

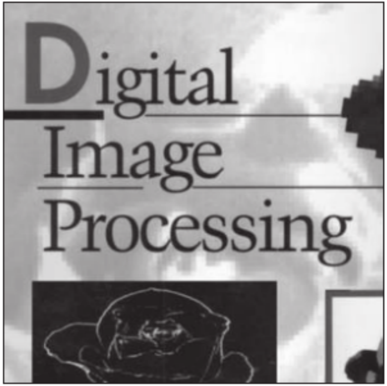


Figure: Left: Original image. Right: K-Means clustered image with $k = 3$. Clustering was done solely based on the pixel intensities

K-Means Clustering Segmentation (5)

- Finding the optimal solution is an **NP-hard problem**. When $T = 0$, the algorithm is known to converge in a finite number of iterations to a local minimum. It is not guaranteed to yield the global minimum.
- In practical applications, heuristic approximations are used
- "Stability" Test: Run Lloyd's algorithm multiple times using different random initializations for the cluster means each time.
- Lloyd's algorithm has a complexity of $\mathcal{O}(knT)$ with k is the number of clusters, n is the number of data points and T is the number of iterations