National University of Computer & Emerging Sciences, Karachi
Artificial Intelligence-School of Computing
Fall 2024, Lab Manual - 05

Course Code (AI4002)	**Course: Computer Vision Lab**
Instructor(s):	**Sohail Ahmed**

Objectives:

- ✓ Introduction to Image Segmentation
- ✓ Techniques/Methodologies of Image Segmentations
- ✓ Region-based segmentation
- ✓ Edge detection segmentation
- ✓ Clustering-based segmentation

1. **Image Segmentation** is the process of partitioning an image into distinct, non-overlapping regions or segments, each of which corresponds to a meaningful object or part of the scene. The goal is to separate an image into areas that share similar visual characteristics, such as color, texture, or intensity.

Mathematics Behind Image Segmentation

The mathematics behind image segmentation involves various techniques and algorithms, including:

- ✓ **Thresholding:** Image segmentation can begin with simple thresholding, where a pixel's intensity value is compared to a predefined threshold. If the intensity is above the threshold, it belongs to one region; otherwise, it belongs to another.

- ✓ **Edge Detection:** Edge detection algorithms identify rapid changes in intensity, which often correspond to object boundaries. Common edge detection methods include the Sobel operator, Canny edge detector, and Laplacian of Gaussian (LoG).

- ✓ **Clustering:** Clustering techniques, like K-Means clustering or Mean-Shift clustering, group similar pixels together based on feature similarity. These methods consider multiple attributes, such as color values, to form clusters.

- ✓ **Graph Theory:** In graph-based segmentation, pixels are represented as nodes in a graph, and edges between nodes represent the similarity between pixel values. Graph algorithms like minimum spanning trees or graph cuts can be used to partition the image into segments.

Real-Life Examples

- ✓ **Medical Imaging:** In medical image analysis, segmentation is used to identify and isolate specific anatomical structures or tumors in X-rays, CT scans, or MRIs.

- ✓ **Object Detection:** In autonomous vehicles, image segmentation is employed to detect and locate objects like pedestrians, other vehicles, and road signs.

- ✓ **Satellite Imagery:** Segmentation of satellite images can help classify land cover types, monitor deforestation, and assess urban growth.

- ✓ **Biomedical Imaging:** In microscopy images, segmentation is used to count cells, identify cell nuclei, and diagnose diseases.

- ✓ **Augmented Reality:** Image segmentation is crucial for scene understanding in AR applications, helping to identify and track objects in real-time video.

2. Image Segmentation Techniques

Image segmentation is a critical task in computer vision, and various techniques and methodologies are employed to achieve accurate and meaningful segmentations. Here are some common techniques and methodologies used in image segmentation:

Thresholding

Methodology- Divides an image into two regions based on pixel intensity values. Pixels above a certain threshold belong to one region, and those below belong to another.
Use Cases- Simple and effective for segmenting objects with distinct intensity differences.

Edge Detection

Methodology- Detects boundaries between objects by identifying rapid changes in intensity or color.
Use Cases- Useful for extracting object contours and boundaries.

Region Growing

Methodology- Starts with a seed pixel and expands to neighboring pixels that are similar in some criteria (e.g., intensity, color).
Use Cases- Effective for segmenting regions with uniform characteristics.

Watershed Segmentation

Methodology- Treats the image as a topographic map and simulates flooding to identify segment boundaries.
Use Cases- Suitable for segmenting touching or overlapping objects.

Clustering-Based Segmentation

Methodology- Groups pixels into clusters based on feature similarity using techniques like K-Means, Mean-Shift, or DBSCAN.
Use Cases- Effective for segmenting objects with complex characteristics.

Deep Learning-Based Segmentation

Methodology- Utilizes deep learning models, such as Convolutional Neural Networks (CNNs), for semantic and instance segmentation.
Use Cases- Suitable for complex tasks and large datasets.
Frameworks- TensorFlow, PyTorch, and libraries like Keras provide pre-trained models and tools for deep learning-based segmentation.

Active Contour Models (Snakes)

Methodology- Deformable models that evolve and adapt to object boundaries based on image gradients and constraints.
Use Cases- Effective for object tracking and boundary refinement.

Thresholding Based Techniques

1. **Global Thresholding**
   Methodology: A single global threshold value is applied to the entire image.
   Use Cases: Effective when there is a clear intensity difference between the objects and background.

```
import cv2

image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
_, binary_mask = cv2.threshold(image, 128, 255, cv2.THRESH_BINARY)
```

2. **Adaptive Thresholding**
   Methodology: Uses different threshold values for different regions of the image, which adapt to local variations in intensity.
   Use Cases: Ideal for images with varying lighting conditions.

```
import cv2

image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
binary_mask = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
```

3. **Otsu's Thresholding**
   Methodology: Automatically selects an optimal threshold value to maximize the inter-class variance between object and background pixels.
   Use Cases: Suitable when the distribution of pixel intensities is bimodal.

```
import cv2

image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
_, binary_mask = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
```

4. **Color-Based Thresholding**
   Methodology: Applies thresholding in multiple color channels (e.g., RGB, HSV) to segment based on color.
   Use Cases: Useful for segmenting objects with distinct colors.

```
import cv2
import numpy as np
```

```
image = cv2.imread('image.jpg')
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
lower_bound = np.array([30, 50, 50])
upper_bound = np.array([60, 255, 255])
mask = cv2.inRange(hsv_image, lower_bound, upper_bound)
```

5. **Hysteresis Thresholding (Canny Edge Detector):**

Methodology: Utilizes two threshold values, a high threshold to detect strong edges and a low threshold to link weak edges.

Use Cases: Typically used for edge detection but can be adapted for segmentation.

```
import cv2
image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
edges = cv2.Canny(image, 100, 200)
```

Region Growing Segmentation Techniques

1. Region Growing Based on Intensity

Region growing is a region-based image segmentation technique that groups neighboring pixels with similar properties into segments or regions. There are different variations of region growing techniques in image segmentation. This technique grows regions by considering the intensity similarity of neighboring pixels.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Load the image
image = cv2.imread('ents.jpg', cv2.IMREAD_GRAYSCALE)

# Initialize seed point (starting point for region growing)
seed_point = (10, 10)

# Define a function for region growing
def region_growing(image, seed, threshold):
    # Create an empty mask to store the segmented region
    mask = np.zeros_like(image, dtype=np.uint8)

    # Create a stack for pixel traversal
    stack = [seed]

    # Get the intensity value of the seed point
    seed_intensity = image[seed]

    while stack:
        x, y = stack.pop()

        # Check if the pixel is within the image boundaries
        if x < 0 or x >= image.shape[0] or y < 0 or y >= image.shape[1]:
            continue

        # Check if the pixel is unvisited
        if mask[x, y] == 0:
            # Check if the intensity difference is below the threshold
            if abs(int(image[x, y]) - int(seed_intensity)) < threshold:
                # Add the pixel to the segmented region
                mask[x, y] = 255

                # Add neighboring pixels to the stack
                stack.extend([(x + 1, y), (x - 1, y), (x, y + 1), (x, y - 1)])

    return mask

# Define a threshold
threshold = 50

# Perform region growing
segmented_image = region_growing(image, seed_point, threshold)

# # Display the segmented image
# cv2.imshow('Segmented Image', segmented_image)
# cv2.waitKey(0)
# cv2.destroyAllWindows()

plt.imshow(segmented_image)
plt.axis('off')
plt.show()
```

Watershed segmentation

Watershed segmentation is a popular image segmentation technique that is particularly useful for segmenting objects with complex shapes or objects that are touching each other in an image. It is based on the idea of treating the pixel intensity values as topographic heights in a 3D surface, where watersheds are used to separate different regions.

Watershed Segmentation Technique

Preprocessing

- ✓ Convert the input image to grayscale if it's a color image.
- ✓ Apply suitable preprocessing techniques like noise reduction and contrast enhancement if necessary.

Marker Generation

- ✓ Identify markers that represent the regions you want to segment. These markers can be manually created or generated automatically based on specific criteria. Common approaches include:
- ✓ Manual placement of markers by the user.
- ✓ Thresholding to create markers based on intensity levels.
- ✓ Distance transform to create markers around the object centers.

Gradient Calculation

- ✓ Calculate the gradient of the image to highlight the boundaries of objects. You can use techniques like Sobel or Scharr filters.

Marker Labeling

- ✓ Label the markers created in step 2 using different integer values.

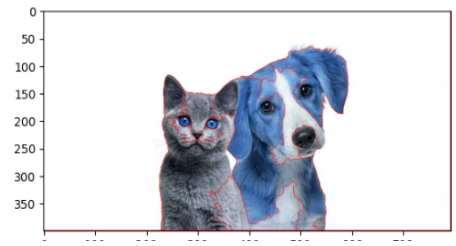
Watershed Transformation

- ✓ Treat the gradient image as a topographic surface where high gradient values represent peaks.
- ✓ Fill basins in the topographic surface with different colors, starting from the labeled markers.
- ✓ The boundaries where the basins meet represent the segmented regions.

Post-processing

- ✓ Optionally, you can apply post-processing techniques like morphological operations (erosion, dilation) to refine the segmentation.

```
In [40]: import cv2
import numpy as np
from matplotlib import pyplot as plt

# Load the image
image = cv2.imread('dog.jpeg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Threshold the image to create markers (you can adjust the threshold value)
_, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

# Perform morphological operations to remove noise
kernel = np.ones((5, 5), np.uint8)
opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=2)

# Create sure background and sure foreground markers
sure_bg = cv2.dilate(opening, kernel, iterations=3)
dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
_, sure_fg = cv2.threshold(dist_transform, 0.2 * dist_transform.max(), 255, 0)

# Subtract sure foreground from sure background to get unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg, sure_fg)

# Label markers for watershed algorithm
_, markers = cv2.connectedComponents(sure_fg)
markers = markers + 1
markers[unknown == 255] = 0

# Apply watershed algorithm
cv2.watershed(image, markers)
image[markers == -1] = [255, 0, 0]  # Mark boundaries with red color

plt.imshow(image)
plt.axis('on')
plt.show()
```



Clustering-Based Segmentation

Clustering-Based Segmentation is a technique that segments an image into regions by grouping pixels into clusters based on certain similarity criteria. It's commonly used for image segmentation when you want to group pixels that share similar characteristics, such as color or intensity. K-Means clustering is a popular algorithm for clustering-based segmentation.

Clustering-Based Segmentation Technique

1. Feature Extraction

- ✓ Choose the feature(s) that represent each pixel's characteristics. For color images, these features could be RGB values or color spaces like LAB or HSV. For grayscale images, intensity values are commonly used.

2. Cluster Initialization

- ✓ Decide on the number of clusters (K) you want to create. This choice depends on the specific image and application.
- ✓ Initialize K cluster centroids. These centroids represent the initial cluster centers, which can be randomly chosen or based on some heuristic.

3. Clustering Algorithm (K-Means)

- ✓ Iterate through the following steps until convergence.
- ✓ Assignment Step: Assign each pixel to the nearest cluster centroid based on feature similarity (e.g., Euclidean distance).
- ✓ Update Step: Recalculate the cluster centroids as the mean of the pixels in each cluster.
- ✓ Convergence is reached when the cluster assignments no longer change significantly.

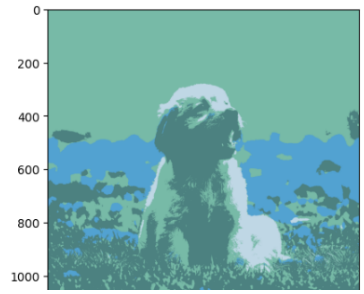
4. Segmentation

- ✓ After convergence, the image is segmented based on the final cluster assignments. Each cluster represents a segment or region in the image.

5. Post-processing

- ✓ Optionally, apply post-processing techniques to refine the segmentation, such as merging or splitting clusters based on specific criteria.

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Load the image
image = cv2.imread('dogimg.jpg')

# Reshape the image to a 2D array of pixels
pixel_values = image.reshape((-1, 3))

# Convert to float32 for K-Means clustering
pixel_values = np.float32(pixel_values)

# Define criteria and apply K-Means clustering
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)
K = 4  # Number of clusters (you can adjust this)
_, labels, centers = cv2.kmeans(pixel_values, K, None, criteria, 50, cv2.KMEANS_RANDOM_CENTERS)

# Convert back to 8-bit values
centers = np.uint8(centers)
segmented_image = centers[labels.flatten()]

# Reshape the segmented image back to its original shape
segmented_image = segmented_image.reshape(image.shape)

# Display the segmented image
cv2.imshow('Segmented Image', segmented_image)
cv2.waitKey(0)
cv2.destroyAllWindows()

plt.imshow(segmented_image)
plt.axis('on')
plt.show()
```



Explanation

We load the input image and reshape it to a 2D array of pixels (pixel_values) where each row represents a pixel and its color values (R, G, B).

The pixel values are converted to float32 for K-Means clustering.

We define the criteria for the K-Means algorithm, including the maximum number of iterations and a small epsilon value for convergence.

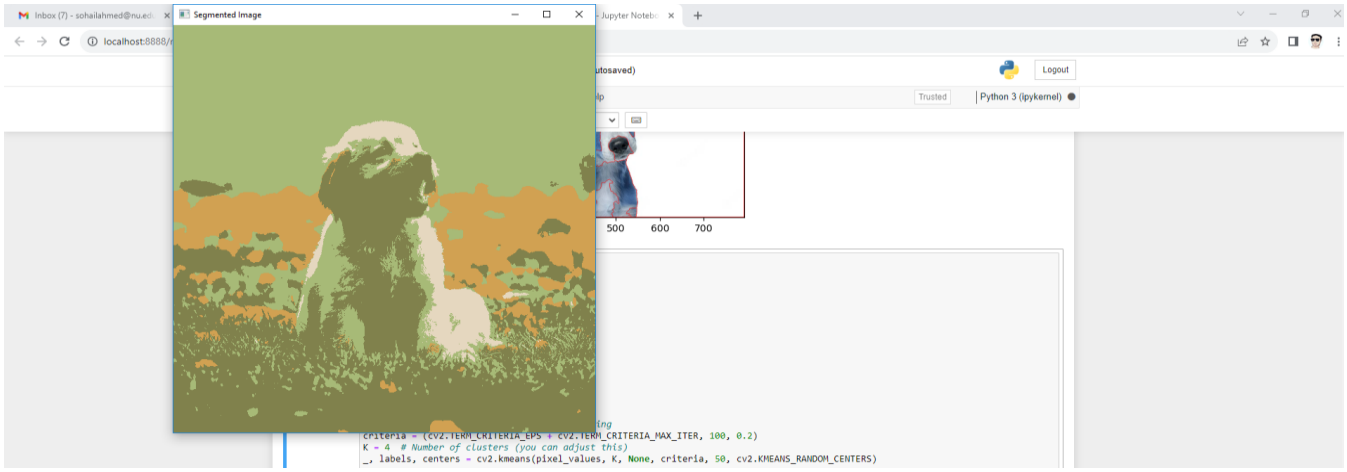
K is set to the desired number of clusters. You can adjust this based on your needs.

K-Means clustering is applied using cv2.kmeans. The resulting labels represent the cluster assignments for each pixel, and centers represent the cluster centroids.

The cluster assignments are used to generate a segmented image by mapping each pixel to its corresponding cluster centroid.

The segmented image is reshaped back to its original shape, and the result is displayed.

You can adjust the number of clusters (K) and other parameters to control the granularity of the segmentation.



Lab Tasks

Task 1: Thresholding-Based Segmentation

Scenario- You have a grayscale medical X-ray image of a bone fracture. The area of interest (the fracture) is significantly darker than the surrounding bone. Perform thresholding-based segmentation to isolate the fracture.

Task 2: Region Growing Intensity-Based Segmentation

Scenario- You have a microscopic image of cells. Choose a seed point in one of the cells, and perform region growing-based segmentation to identify and separate that cell from the rest.

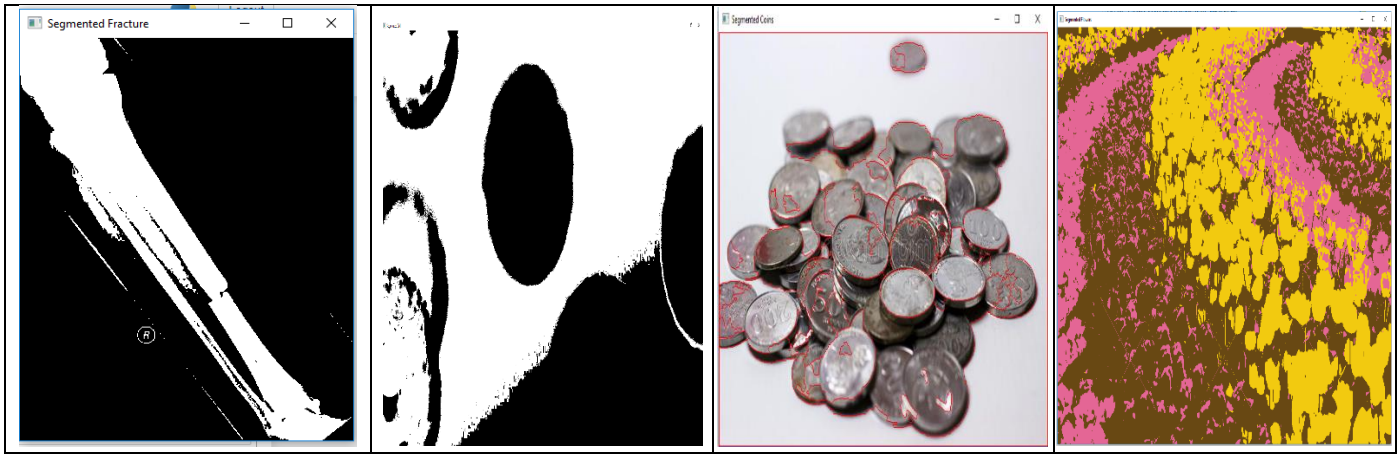
Task 3: Watershed Segmentation

Scenario- You have an image of overlapping coins on a table. Perform watershed segmentation to separate and count the individual coins.

Task 4: Cluster-Based Segmentation

Scenario- You have an image of colorful flowers in a garden. Perform cluster-based segmentation to separate different types of flowers based on color.

Expected Outputs



Original Images

The End ☺