CENG 466 - Image Processing
THE 3

Mert Uludoğan
2380996

Yiğitcan Özcan
2521847

I. Task 1: grayscale morphology

For the first task, we implemented different functions for each segmentation goal. They can be listed as

A. *rug_pattern_morph*

This function processes an image to identify rug patterns using morphological operations, edge detection, and masking techniques.
First, the input image is loaded in color (*im_color*) and converted to grayscale (*im_g*). The grayscale image is then converted to 8-bit format and inverted (*im_inv*) to enhance patterns against background details.

Morphological operations (MORPH_OPEN and MORPH_CLOSE) are applied sequentially to the inverted image to smooth the texture and reduce noise, improving pattern recognition.

For edge detection, the Canny edge detector is utilized with a specified *gradient_sigma*. The detected edges are then thickened using dilation with a structuring element (disk(2)) to help define clearer boundaries for segmentation.

Foreground and background markers are set based on intensity thresholds (*lower_bound_pattern* and *upper_bound_background*). These markers identify regions corresponding to patterns and background, respectively. The markers are further refined using a median blur to ensure smoother transitions and clearer segmentation.

Finally, the image is masked by applying an inverted mask based on the refined markers, creating a masked image that highlights the patterns while suppressing the background. This approach effectively isolates rug patterns from their surroundings.

rug_pattern_morph returns:

- im_color: Original color image.
- im_inv: Inverted grayscale image after preprocessing.
- im_gradient: Gradient image derived from edge detection.
- markers: Marker labels for defining foreground and background.
- masked_image: Resulting image with the patterns highlighted and background suppressed.

This implementation focuses on effectively isolating and highlighting rug patterns using a combination of morphological operations, edge detection, and strategic masking techniques to achieve clear pattern segmentation from the background.

B. *wrinkle_finder*

This function detects wrinkles in an image using edge detection and morphological processing.
The input image is read in color (im_color) and converted to grayscale (im_gray). The grayscale image is normalized to a consistent range (0–255) for better edge detection (im_norm).

Then, edges are detected using the Canny edge detector with a specified sigma. Edges are thickened using dilation (disk(2)), and the gradient is normalized to 255.

A binary mask is created by thresholding the gradient image (threshold parameter).

Lastly, small noise is removed using morphological opening. Small gaps in the detected wrinkles are filled using morphological closing (morph_kernel size).
wrinkle_finder returns:

- im_color: Original color image.
- im_gradient: Gradient of the image highlighting edges.
- binary_mask: Postprocessed binary mask of detected wrinkles.

C. *zipper_finder*

This function detects zipper-like regions in an image using edge detection and thickening operations.

The input image is read in color (original) and converted to grayscale (grayscale). The grayscale image is normalized to a (0–255) range.

A median blur (median_blur_k) is applied to smooth the grayscale image. The Canny edge detector is used to compute the gradient with specified thresholds (grad_lower, grad_upper).

Dilation is applied iteratively (5 times) to thicken the detected edges. A structuring element (cv2.MORPH_ELLIPSE) is used for dilation.

A binary mask is created from the thickened edges. The original image is masked with this binary mask to extract regions corresponding to the detected zipper.

zipper_finder returns:

- original: Original color image.
- gradient: Gradient image showing detected edges.
- thickened: Thickened edges highlighting zipper-like regions.
- zipper_rgb: RGB regions of the zipper extracted using the binary mask.

D. *Outputs of Morphological Operations*

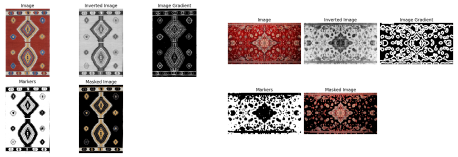

Fig. 1. Rug patterns extracted using grayscale morphology

Fig. 2. Wrinkles detected using grayscale morphology

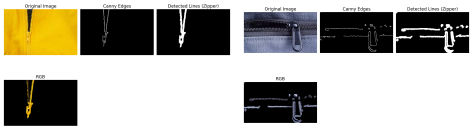

Fig. 3. Zipper detected using grayscale morphology

II. Task 2: K-means clustering using RGB features

For the second task, we defined the functions; preprocess_image, apply_kmeans_rgb, mask_segment_rgg, mask_segment_enhanced_rgb and these functions collectively allow for various enhancements like preprocessing, segmentation, and selective highlighting of regions from an image, which helps a lot in object detection, region-of-interest extraction, and visual analysis tasks.

A. *preprocess_image*

This function preprocesses an RGB image by applying Gaussian smoothing and contrast stretching. These steps help enhance the image and make it suitable for further processing, such as segmentation.

First, it reads the image from the specified path and converts it to RGB format using OpenCV (cv2.cvtColor). OpenCV reads images in BGR format by default, so conversion ensures a proper RGB representation.

Secondly, it smooths the image using a Gaussian filter with a specified kernel size (kernel_size). This reduces noise and smoothens the transitions between colors in the image, which is important for improving the performance of subsequent processing.

In the end, each RGB channel (Red, Green, Blue) of the smoothed image is independently enhanced using contrast stretching. The contrast stretching spreads out the intensity values of the pixels throughout the range (0-255), improving the visibility of details in each channel.

preprocess_image returns:

- original: The original RGB image.
- stretched: The preprocessed image with smoothing and contrast stretching applied.

B. *apply_kmeans_rgb*

This function segments an RGB image into different regions (clusters) based on color similarity using the K-Means clustering algorithm.

At first, it reshapes the 3D image array into a 2D array of shape (num_pixels, 3), where each pixel is represented by its RGB values.

Then, it applies the K-Means algorithm to group pixels into n_clusters clusters based on their RGB values. The random_state ensures reproducibility of the clustering results.

For the last step, each pixel is assigned the color of its cluster center. Reshapes the cluster-labeled pixels back into the original image dimensions to create the segmented image.

apply_kmeans_rgb returns:

- segmented_image: The segmented image where each cluster is represented by a distinct color.
- labels: A 2D array where each pixel is assigned a cluster label (an integer from 0 to n_clusters-1).

C. *mask_segment_rgb*

This function isolates and masks a specific cluster from the segmented image, highlighting the region corresponding to a particular color cluster.

At first, it creates a mask by setting the pixels belonging to the target_cluster to 255 (white) and others to 0 (black). The resulting binary mask is single-channel.

After that, it converts the single-channel binary mask into a 3-channel mask to match the original RGB image.

Finally, it applies the 3-channel mask to the original image using a bitwise AND operation. This operation isolates the pixels belonging to the target_cluster, while others are set to black.

mask_segment_rgb returns:

- masked_image: The original image with only the target_cluster region visible, and all other regions blacked out.

D. *mask_segment_enhanced_rgb*

It is an enhanced version of mask_segment_rgb. We call it enhanced, because it does not take a single target_cluster, but instead it takes a list of target_clusters.

E. *Outputs of K-Means clustering using RGB features*

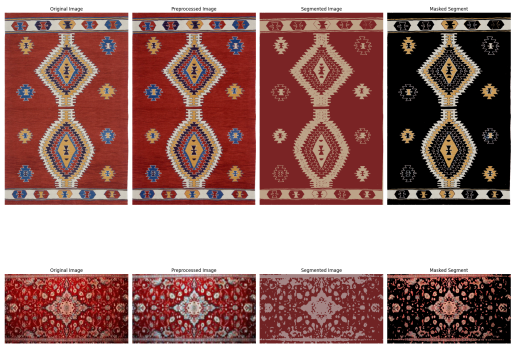

Fig. 4. Rug patterns extracted with K-Means clustering using RGB features

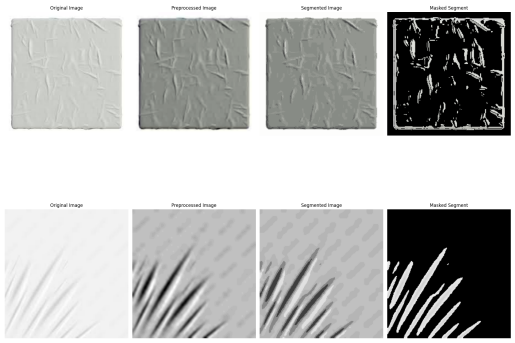

Fig. 5. Wrinkles detected with K-Means clustering using RGB features

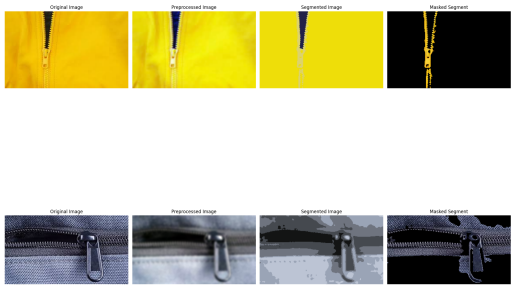

Fig. 6. Zipper detected with K-Means clustering using RGB features

III. Task 3: K-means clustering using LBP features

The third task was the most difficult and time consuming one for us, because LBP is a non-intuitive feature descriptor that requires understanding the spatial relationships between pixels and how these relationships encode texture information. We encountered the outputs that make sense to human visual system but, they were hard to segment.

A. *compute_lbp*

It computes the Local Binary Pattern (LBP) for a grayscale image. LBP is a texture descriptor that captures local spatial patterns by comparing pixel intensities.

This function takes the grayscale input image, the radius of the circular neighborhood for the LBP computation (default is 1 pixel), and the number of sampling points in the circular neighborhood (default is 8 points).

It uses the local_binary_pattern function from skimage.feature, to calculate the LBP for each pixel. Compares the intensity of the center pixel with its neighbors in a circular pattern. It assigns binary values (1 or 0) based on whether a neighbor's intensity is greater than or less than the center pixel. These binary values form a binary number, which is converted to a decimal value and stored in the LBP image.

compute_lbp returns:

- An LBP image where each pixel represents the LBP value corresponding to its neighborhood.

B. *apply_kmeans_lbp*

This function clusters the LBP image into clusters using the K-Means algorithm.

First, it flattens the 2D LBP image into a 1D array of pixel values (lbp_flat) for clustering.

Then, it uses the K-Means algorithm to group the pixel values into n_clusters. Each pixel is assigned a cluster label based on its similarity to the cluster centers.

In the end, it converts the cluster labels back into the original 2D image shape.

apply_kmeans_lbp returns:

- clustered_image: A 2D array where each pixel is assigned a cluster label.
- labels: The reshaped array of cluster labels.

C. *mask_segment_enhanced_lbp*

This function isolates and highlights regions in the original image corresponding to specific clusters in the LBP segmentation.

It converts the original image to uint8 format, normalizing it to the range [0, 255] if necessary. Then, creates a blank binary mask and sets pixels corresponding to target_clusters to 255 (white) in the mask.

After that, it resizes the mask if its dimensions differ from the original image and converts the single-channel binary mask into a 3-channel mask to match the original RGB image.

In the final, it uses a bitwise AND operation to isolate the specified regions in the original image.

mask_segment_enhanced_lbp returns:

- masked_image: The original image with only the regions corresponding to the target_clusters visible, while other regions are blacked out.

D. remove_small_objects_morph

Before implementing this function, some of our images had unnecessary segment that is caused by slight shadow changes. This function performs morphological operations to dilate and remove small objects in the LBP image, enhancing the visibility of larger structures.

Firstly, it creates an elliptical structuring element of size (kernel_size, kernel_size). Elliptic structuring element is the most suitable one for our noise-like small objects' removal.

Then, it applies dilation using the morphological kernel for the specified number of iterations. Dilation thickens the structures in the image, effectively removing small objects.

remove_small_objects_morph returns:

- dilated_image: The LBP image after dilation, with small objects removed and larger objects enhanced.

E. adjust_pixel_intensity

This function adjusts the intensity of an image by darkening low-intensity pixels and lightening high-intensity pixels, enhancing contrast.

First, it converts the grayscale image to a float32 format to allow for scaling during intensity adjustments.

Then, it creates a mask for pixels with intensity less than lower (darken) and another for pixels with intensity greater than upper (lighten).

It multiplies the intensity of pixels in the darken_mask by darken_factor to darken them, also multiplies the intensity of pixels in the lighten_mask by lighten_factor to lighten them. It ensures all pixel values remain within the valid range [0, 255] too.

adjust_pixel_intensity returns:

- adjusted_image: The intensity-adjusted grayscale image.

F. Outputs of K-Means Clustering Using LBP Features

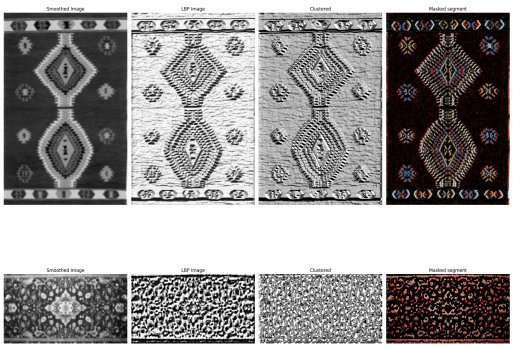

Fig. 7. Rug patterns extracted with K-Means clustering using LBP features

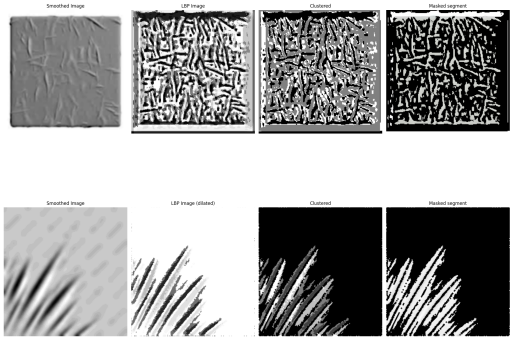

Fig. 8. Wrinkles detected with K-Means clustering using LBP features

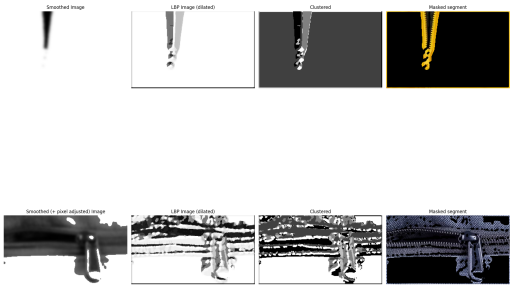

Fig. 9. Zipper detected with K-Means clustering using LBP features

IV. GENERAL ANALYSIS OF THE METHODS

A. *Segmentation Quality vs. Choices of Parameters*

*1) **Task 1: Morphological segmentation**:*

Parameters:

lower_bound_pattern, upper_bound_background: controls separation between pattern and background; morphological kernel size, number of iterations: defines the amount of smoothing out

Effects:

Little kernel size will not delete noise and as such can give fragmented patterns. Smoothing too much risks bleaching the edges between the pattern and background, potentially decreasing segmentation accuracy. For example, lower_bound_pattern and upper_bound_background both tend to be sensitive; small changes can make either value too large or too small, preventing adaptiveness across different image intensities.

*2) **Task 2: RGB K-Means Clustering**:*

Parameters:

n_clusters: This controls the amount of detail in the segmentation.

Effects:

With fewer clusters, the result is under-segmentation, and distinct regions get merged. With too many clusters, there is over-segmentation; homogeneous regions might get divided into multiple clusters. Contrast stretching increases the distinguishability of the clusters but may also enhance noise during the pre-processing step.

*3) **Task 3: LBP and Texture Analysis**:*

Parameters:

radius, n_points influence the granularity of texture description. K-Means parameters (n_clusters) and morphological kernel size affect texture-based segmentation.

Effects:

Small radius and n_points may lose fine texture details, while larger values increase computational cost without significant gains for simple textures. The result of segmentation is strongly related to the LBP pattern distribution that may vary strongly from image to image.

B. *Advantages and Disadvantages of Each Approach*

*1) **Task 1: Morphological Segmentation**:*

Advantages:

- Has produced satisfying results in cases where clear intensity differences existed between the foreground and the background.
- Morphological operations are able to enhance edges and eliminate noise.

Disadvantages:

- Performs poorly on images with complex textures or overlapping intensity ranges.
- It may be tricky to tune the parameters for a diverse set of images.

*2) **Task 2: RGB K-Means Clustering**:*

Advantages:

- Handles color-based features very well, hence suitable for multicolored images.
- Easy to implement and interpret in RGB segmentation tasks.

Disadvantages:

- Does not perform well on grayscale or low-color-variance images.
- Sensitivity to lighting conditions and quality of preprocessing.

*3) **Task 3: LBP and Texture Analysis**:*

Advantages:

- Highly effective for texture-based segmentation, capturing fine details.
- Robust against intensity variations, focusing on patterns instead.

Disadvantages:

- Computationally expensive, especially with larger radius and n_points.
- Limited performance in images dominated by color variations with little texture.

C. *Algorithm Performance per Image*

Image of Simple Intensity Patterns-for example, rugs or simple patterns: Best Approach: Task 1 (Morphological Segmentation) Works well because the intensity differences are well defined and the structures are uncomplicated. Why: Morphological smoothing efficiently removes noise and segments patterns.

Complexly Colored Image Example-: multicolored Fabrics: Best Algorithm: Assignment 2 RGB K-Means Clustering Best captures the differentiation in color to determine demarcation of regions Why: K-means utilizes color clustering to an advantage, beating possible alternatives that might have utilized intensity or texture.

High-texture-complexity image-for example, with wrinkled surfaces or complex designs: Best Approach: Task 3-LBP and Texture Analysis LBP captures texture details, which is important for robust segmentation. Why: In such images, texture features are more appropriate than those based on intensity or color.

D. The Impact of Image Complexity

Simple Images: Task 1 is better because the morphological operation can separate different regions clearly. Preprocessing by morphological smoothing cleans the segmentation well.

Color-Dominant Images: Task 2 is better because RGB K-Means directly targets color features. Preprocessing by contrast stretching enhances the separation between clusters.

Texture-Dominant Images: Task 3 is optimal because it leverages texture patterns, which are independent of color or intensity. Postprocessing through morphological operations refines results by removing small, irrelevant details.

5. Impact of Preprocessing and Postprocessing

Preprocessing: Essential to enhance quality in segmentation, Gaussian smoothing in Task 2 cleans the noise and enhances clustering results. LBP computation in Task 3 relies on preprocessing to normalize the variations in intensity.

Postprocessing: Improves segmentation robustness: Morphological operations refine noisy segmentation masks (Task 1 and Task 3). Thresholding and masking ensure clean separation in Tasks 2 and 3.

Final Insights:

Best Method: Task 1 is best for simple images with marked intensity contrasts. Task 2 is ideal for color-rich images with little or no texture. Task 3 excels in texture-dominant images. Overall Conclusion: The choice of method will depend on the complexity of the image. Results always improve when pre- and post-processing are adapted to the applied method and image characteristics.

REFERENCES

Rafael C. Gonzales and Richard E. Woods, *Digital Image Processing*. New Jersey: Pearson Education Inc., 2008.

Hazal M. Ozcan, Itir O. Ertugrul, Fatos T. Yarman Vural, *Fundamentals of Image Processing with Python*. Wiley Inc., 2025.