REPORT LAB 3
SEGMENTATION AS REGION CLASSIFIERS

INTRODUCTION

Segmentation is a fundamental task in computer vision that aims to assign a meaning to each pixel in an image. However, to define a pixel is really challenging due to the high variability of appearance, shape, and scale of objects in natural scenes. Therefore, many methods have adopted a region-based approach, where the image is first divided into coherent parts or regions, and then classified into one of the predefined categories (background/foreground). Region classifiers can be defined using the spatial support and boundary information of the regions, as well as the contextual cues from neighboring regions. In this way, we can produce more accurate and consistent segmentation results.

CONTENT

1 THRESHOLD-BASED SEGMENTATION EXERCISE

When we talk about global thresholding we are referring to one of the multiple segmentation methods that are used in computer vision to do segmentation (separate the background and foreground). To determine those regions we apply a threshold. This is a concrete pixel value (between 0 and 255) all the values then are compared with this value and if they are bigger or smaller than the value they are classified as a foreground or background respectively. However, to choose that value there are more efficient or good ways than others. You could create an histogram of values and choose the threshold by considering it, or even you can do it randomly, but maybe it will make no sense. In other way around, you can use what is called automatic thresholding, which basically uses some methods as the adaptive mean or the Otsu's method to find the most adequate threshold.

Let's talk about those methods. First the mean thresholding, what it does is to decide the local threshold, which is a value for a determined region, based on the values of the neighbor pixels. This process is actually very useful for images where the illumination conditions vary or when different parts of the image have different light properties or characteristics. Moreover, in comparison with global thresholding it handles the noise better, it differs from the different textured regions and contrary to global thresholding it adapts dynamically to the different regions of an image. Secondly, the Otsu's method also segments the image in two classes or regions, however the process is different, so as it is more complex the segmentation becomes more refined.

1. First it does an histogram of the pixel's intensity of the image. Then it normalizes the values and computes the cumulative distribution from the normalized values.
2. Secondly, as the objective is to find the threshold that maximizes the inter-class variance, it considers all threshold values (intensity levels) and computes the variances of the two classes (the spread of a pixel intensity within a class).
3. Third, the value that maximizes the inter-class will be considered the optimal one as it will effectively separate the background and foreground.

Actually, the advantages of the Otsu's method is that it automatically determines the threshold based on the different properties of an image, so this process is actually suitable

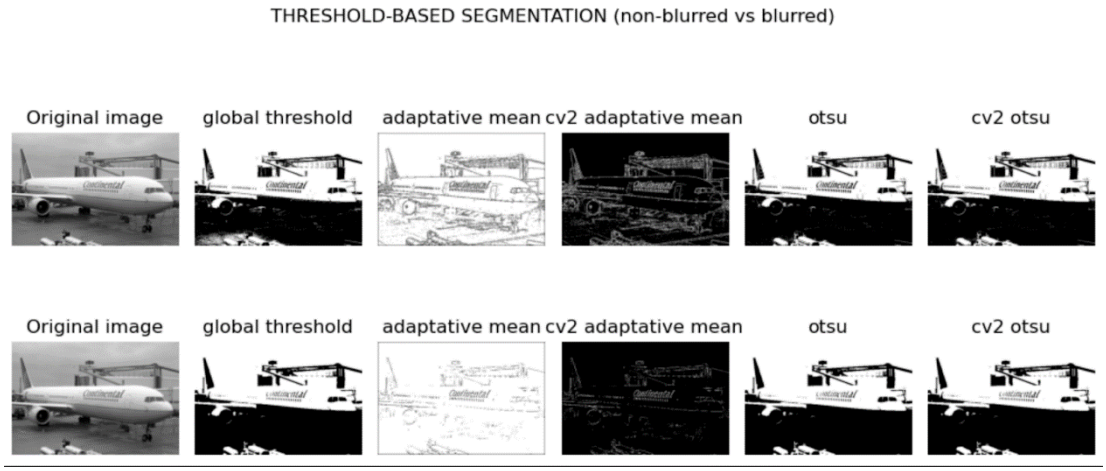
for a wide amount of images without having to adjust any other parameter manually. On the contrary the global threshold may work for an image and not any other.

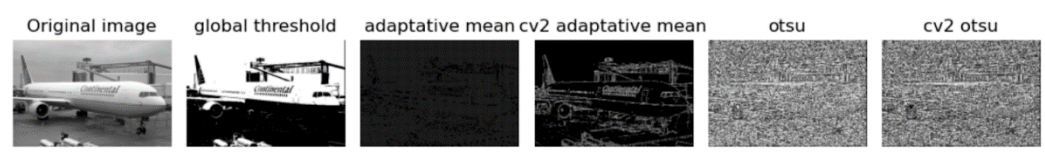
During our process we have applied those processes manually and using python libraries. In case of the Otsu's method we have proven that the manually and the one using library is actually the same because the threshold value obtained is almost the same:

```
Otsu's algorithm implementation thresholding result (manually computed):  136.962890625
Otsu's algorithm implementation thresholding result (CV2 function):  137.0
```

Moreover we have also computed the local thresholding which actually uses all these methods but in a region (in our case we have said to be size 3x3, but can be any). Due to the fact that it has a parameter named size it determines how much of the image will be entered as the image in the different methods. Nevertheless, it iterates until the process has been done in all the image.

This process (local thresholding) does not ensure having better results, however as the defined method is applied in a part/window of an image instead of using the whole, it means that it probably will get better results as some aspects such as light or noise will be better controlled. But, no matter that, we need to remember that computing the local thresholding it is actually involves a more intensive computation as it will need to apply the method (compute thresholds) in each window.





The obtained results are pretty much the same as the previous segmentation representations, as in our case, the size that is defined in the local thresholding is of the same measure that the kernel window (3x3). However, it is relevant to emphasize that because the otsu's method is not so directly implemented the results change (it is not only seeing if a value is bigger or smaller than a thresholding as it does the global).

These functions have used some parameters that are asked to the user. Such as the threshold, the region value, the constant C or the kernel size in case of the last method mentioned. In the previous results (image of the airplane) we have used the following values: Threshold = 127, Region value = 3, constant C = 10 and kernel size = 3.

2 EDGE-BASED SEGMENTATION EXERCISE

Edges are discontinuities in an image that can arise from change of color, depth, illumination or others. The process of edge detection involves detecting sharp edges in the image and producing a binary image as the output. To perform edge segmentation we used three methods: Sobel filters, Canny and Laplacian of Gaussian.

To apply Sobel filters we first need to apply the vertical and horizontal kernels separately (we will call them Gx and Gy respectively) using convolution, and then perform the following operation: Gx2+Gy2. When we do this operation we are getting the image gradient. The gradient of an image is an array where the values represent the direction where intensity increases the most. It is useful because regions without edges will return zero gradient, while the other will return positive or negative values. Finally, we must normalize the gradient. We must apply both the vertical and the horizontal kernel because edges can occur in both directions.

With the Canny Edge Detector there are many more steps. Firstly, we must apply a Gaussian filter to the image using convolution to smooth it and remove noise. Then, we will compute the image gradient and the orientation of the gradient using our sobel_edge_detection function. The orientation is found with the following formula: `arctan2(Gy, Gx)`.

Once we have the image gradient we must apply Non-Maximum Suppression to it, which thins the edges down to a single pixel in width. To do so, we iterate through each pixel in the gradient and find the neighbor pixels which are in the direction of the gradient. If the intensity of the pixel is MAX compared to other pixels in the same direction we will keep the value. Otherwise, the value of the pixel will be set to 0.

After that, we will do Hysteresis thresholding to discard non-edges. It works by applying a high and a low threshold: those edges above the high threshold are sure-edges, the ones below the low threshold are non-edges (which will be discarded), and the ones between both threshold values are weak-edges, which will be further examined. If a weak edge is connected to a sure-edge, it is classified as an edge. However, if a weak edge is not connected to a sure-edge, it is discarded.

In our function we first tried to find the right values for the high and low threshold by doing the 90 percentile of the gradient for the high_t, and for the low_t we multiplied the high_t by 0.1 to obtain a fraction. This way we can obtain thresholds that adapt to each image instead of setting a fixed value for all images.

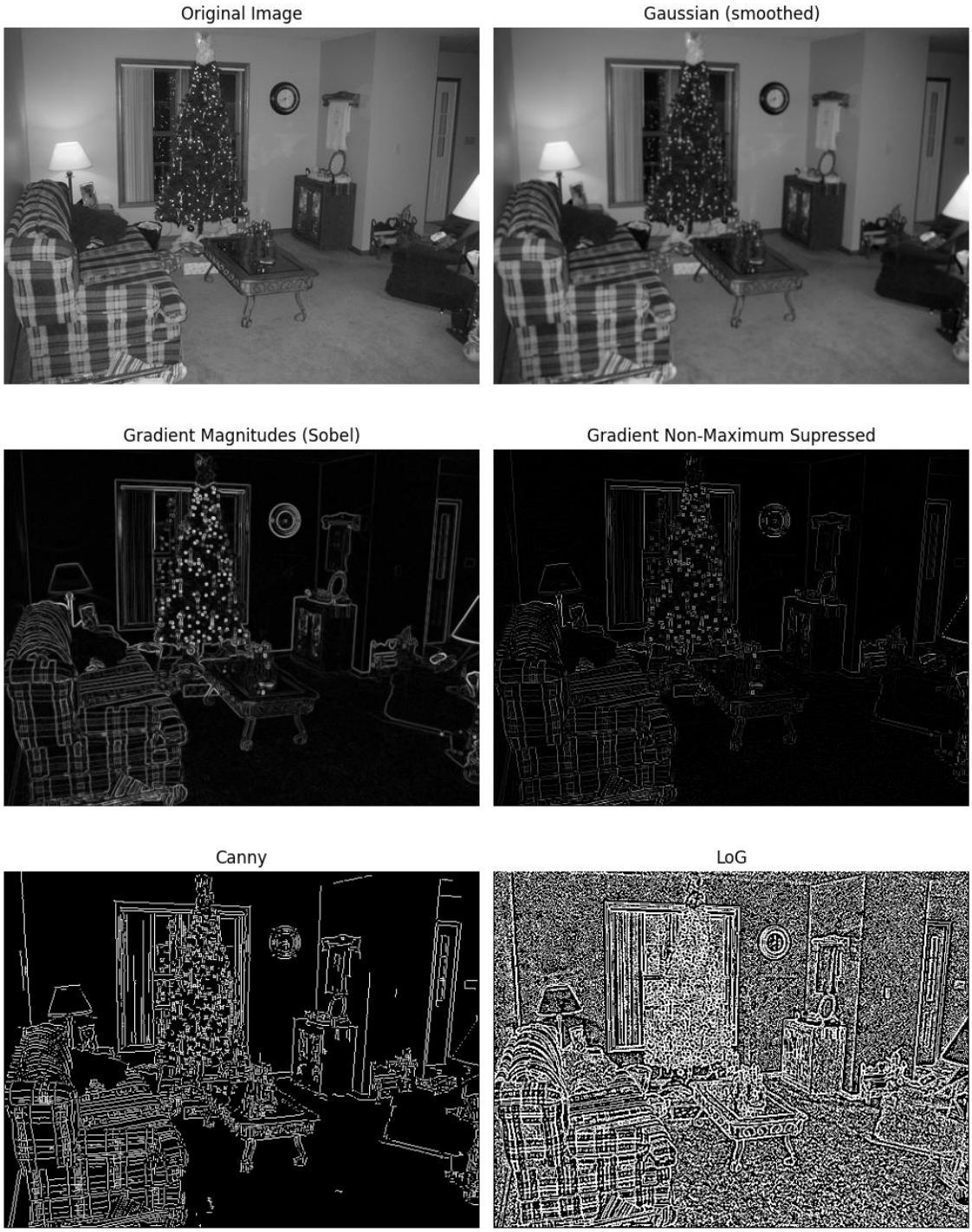
After that, we started to classify the edges based on the threshold values we set, and then we iterated through the weak edges to see if they were attached to some sure-edge by looking at its neighbors. If so, the edge was set to 255 (sure-edge value). If not, the edge was set to 0 (non-edge value).

The last operation was the Laplacian of Gaussian. The Laplacian of an image highlights regions of rapid intensity change, so features with a sharp discontinuity will be enhanced.

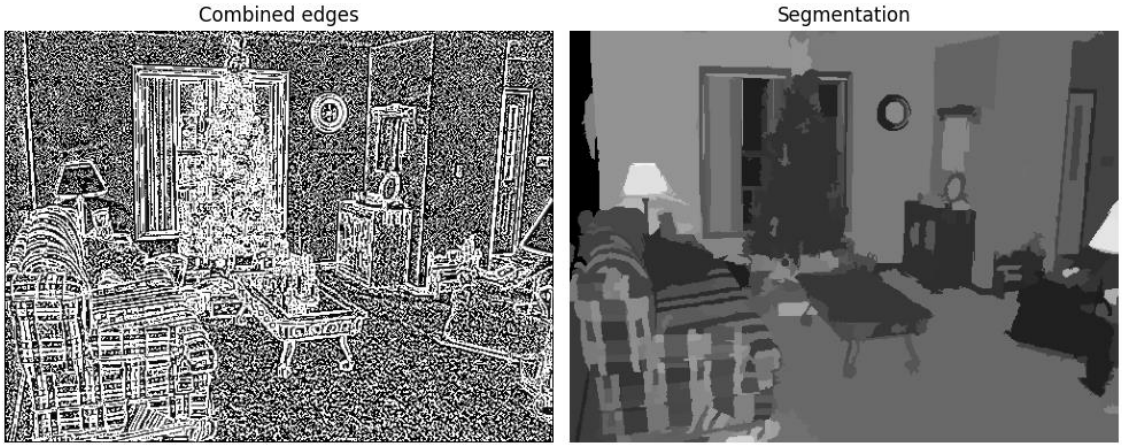
It does it by computing the second derivative for each pixel because edges are at between pixels whose second derivatives sharply transition from one sign to the other (Zero-crossing).

In this method we first apply a Gaussian filter to remove the noise and then apply the Laplacian kernel using convolution. We end the operation applying Zero-Crossing, which consists of finding if pixels are surrounded by changes of sign values around them. If so, we will set them to 255.

This are our results:



In addition, we had to find a method to fill the connected regions to obtain segmented regions (instead of edge regions). We will apply image segmentation using Felzenswalb algorithm, which is a bottom-up segmentation algorithm that groups pixels into segments based on their color similarity and proximity. The key concept is to iteratively merge adjacent image regions if the cost of merging them is lower than a certain threshold. Here are the results:



3 WATERSHEDS

Watersheds is an image segmentation algorithm that visualizes the image as a topographic landscape where high intensity represents peaks and hills and low intensity represents valleys. To apply Watersheds, we apply the following steps:

3.1 IMAGE PREPROCESSING:

- Convert RGB image to grayscale and to binary using thresholding.
- Apply opening to the binary image to remove noise and emphasize edges. In this case we used a 5x5 kernel.

3.2 GETTING SURE FOREGROUND, SURE BACKGROUND AND UNKNOWN REGIONS

- We use dilation to identify the sure background (the pixels that are black for sure) since we add edges to the objects.
- We apply Distance transform to the sure background result to identify the regions that are likely to be foreground (white). We do this calculating the distance of every white pixel to the nearest black pixel using Eucledian distance. The output is a numpy array where the non-black (non-0) values are the distance result.
- We threshold the Distance Transform to get the sure foreground region. We will set the threshold level at 50% of the maximum distance found by the Distance Transform, so pixels with distance higher that this threshold are set as sure foreground.
- To identify the unknown regions (the ones that are not sure background or foreground, the regions near the edges) we simply subtract the sure background and the sure foreground.

3.3 GETTING MARKERS (DIFFERENT OBJECTS) OF THE IMAGE:

Now we will define the markers using the sure foreground regions. We will do this by iterating through the pixels, and if they're white (sure region) and they're not labeled yet, we will start a new marker and mark all the white pixels attached to it to the same label using BFS. With BFS we search for all neighboring pixels and if they're white, we assign the current label to it. The result should be an image were the different markers/regions are labeled with a different number.

To distinguish these markers with the background, we increment all values of the markers by 1 and set the unknown regions to 0. This way, sure background pixels are labeled as 1 and sure foreground pixels are labeled starting from 2, while unknown regions are represented as black (0).

3.4 ASSIGN UNKNOWN REGION TO BACKGROUND OR TO ONE OF THE MARKERS:

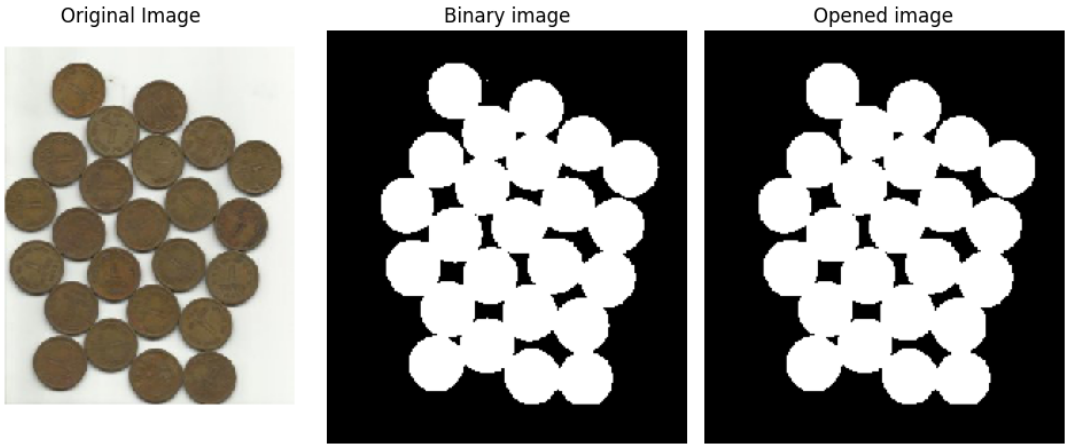
Now we will apply "cv2.watershed" to the marker image. This function modifies the markers image: it marks the borders of the objects with -1 and the actual objects with different positive integers. Therefore, the unknown regions must be assigned to background or to foreground (to one of the objects). But how does it actually work?

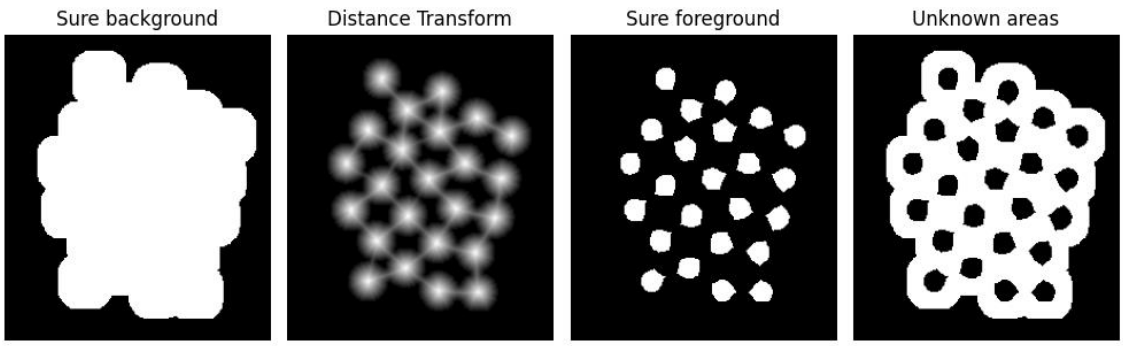
- Flooding: we expand each marker based on the gradient of the image. In this context, the gradient represents the topographic elevation (high intensity = peak, low = valley). Each pixel is assigned to one of the markers depending on which marker's "flood" it reaches first. If a pixel reaches more than one marker equally, we will keep it as unknown for now.
- Dam Construction: as the flooding processes, floods from different markers will start to meet. When they do, we create a boundary of -1 to separate them.

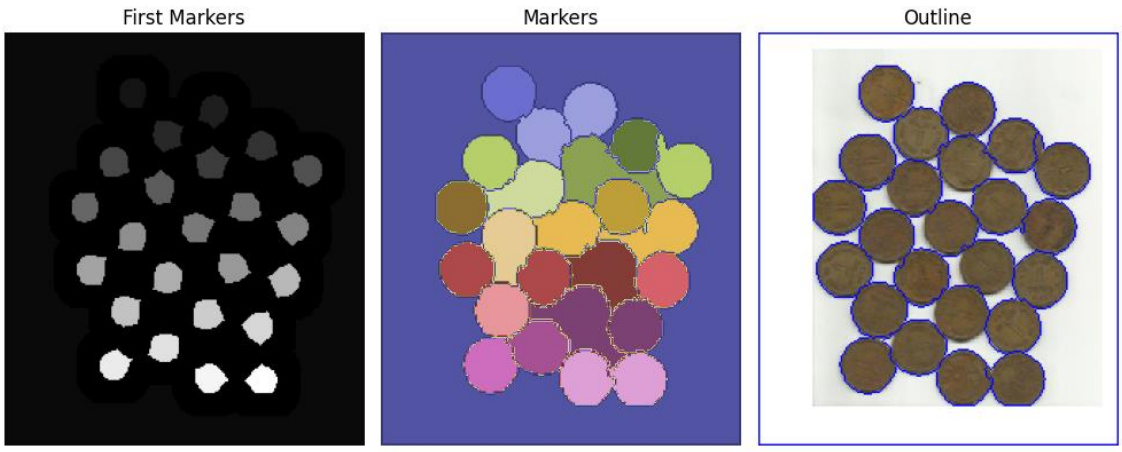
3.5 GET FINAL IMAGE WITH CONTOURS:

Once we have our markers defined, we will draw in the original image the contours of each object using cv2.findContours. To find each object contour separately, we compute a binary image for each marker were the only white part is the marker region.

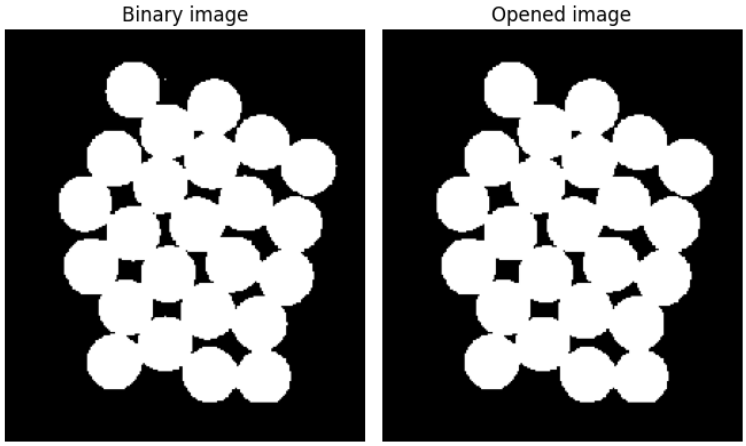
Our first approach to apply Watersheds was using the cv2.distanceTransform, cv2.connectedComponents and cv2.watershed functions. This are the results:

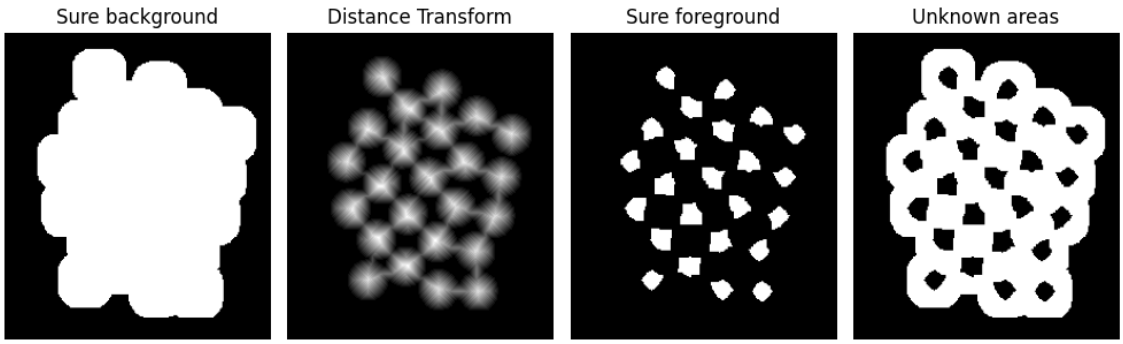


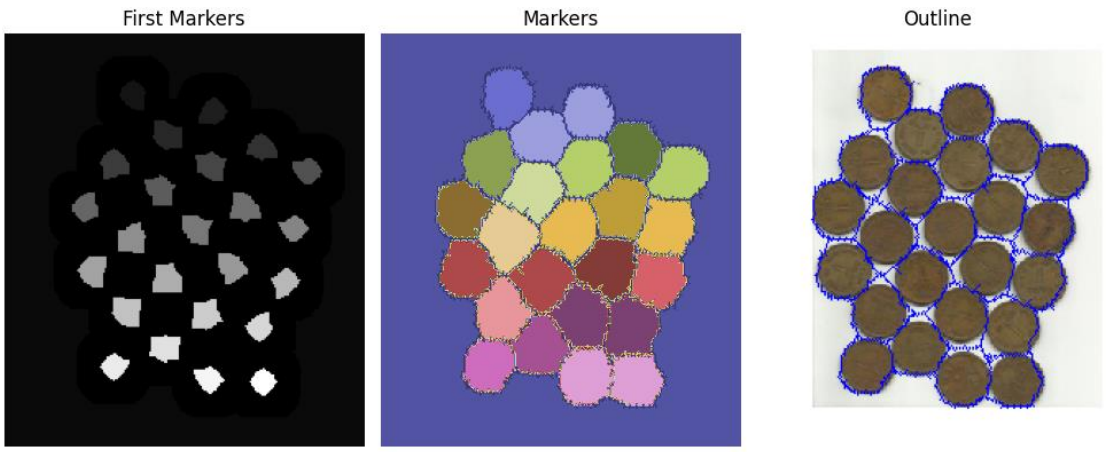




However, we wanted to implement a manual approach were we didn't use any of the cv2 functions mentioned but our own: DistanceTransform(), GetMarkers() and DoWatersheds(). This are the results:







4 REGION-BASED SEGMENTATION EXERCISE

If we talk about region-based segmentation we are referring to those kinds of algorithms that divide an image in different parts based on a certain criteria, such as the intensity of a pixel, the color or the texture. The aim of those methods is to group together pixels that share similar characteristics and form coherent regions in the image. Some methods are region growing, split-and-merge approach, flood fill or mean-shift algorithm. However not all algorithms work in the same way and so it involves that the effectiveness of region-based segmentation method can be influenced by factors such as noise levels, image resolution, and the nature of the objects to be segmented.

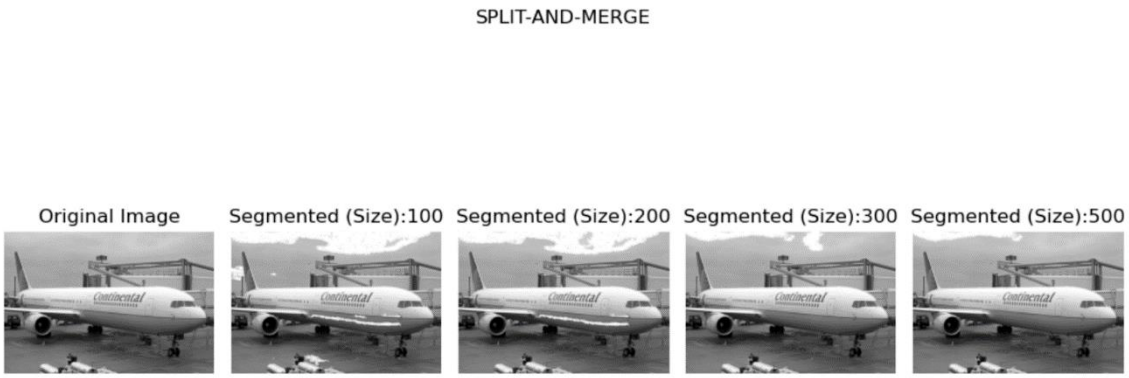
Let's look into some properties of those methods. The region growing algorithm is a pixel-based image segmentation algorithm that groups together neighboring pixels with similar properties to form coherent regions or objects. The algorithm starts with one or more seed points (they are selected by the user randomly or based on a criteria) and iteratively grows the region by adding adjacent pixels. In our algorithm, the adjacent pixels are examined and classified into the seed points if they have the same intensity value. It is an iterated process until there are no changes in two successive iterative stages, so then we assume that the condition is no longer applicable.

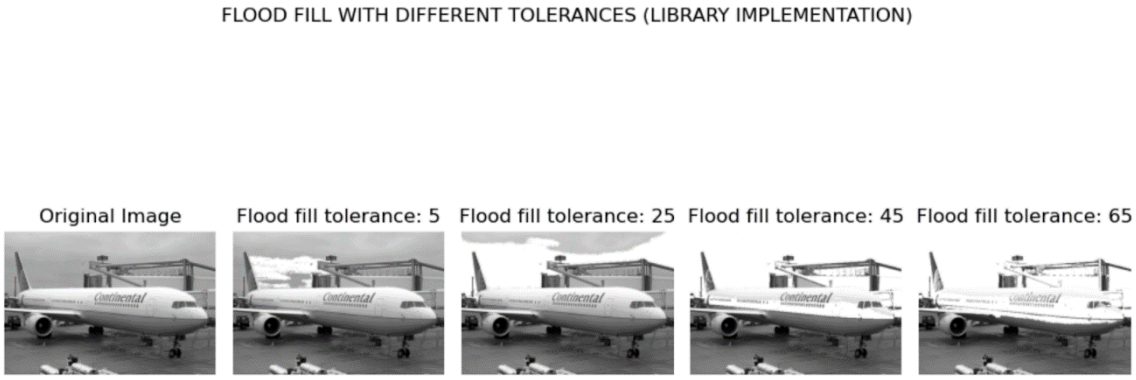
In comparison to this algorithm, mean-shift does not depend on the initialized/starting points. What it does is instead of considering the value of concrete data parameters it works identifying dense regions on the image. It makes a supposition that changes due to a kernel matrix that computes the weights of the neighbor values and re-estimate the supposed mean. At the end the center of a region is shifted towards the mean position. This is done until it converges. This algorithm apart from not being influenced by the initialization it also does not make assumptions of the distribution nor or require previous knowledge of cluster design. Besides, it can handle arbitrary data shapes, sizes and also noise. On the contrary of all those benefits, it is actually computationally expensive and it is sensitive to the choice of bandwidth (which also determines the quality of segmentation).

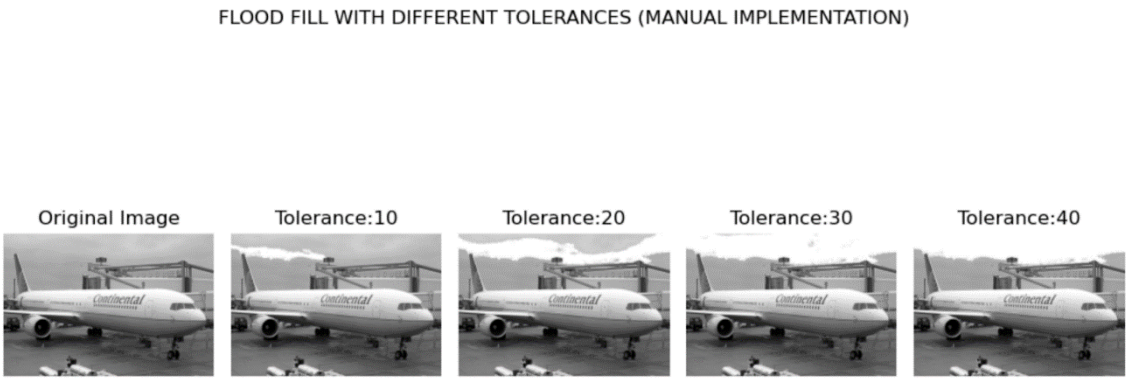
Furthermore, we also have a method that divides an image to elaborate and differentiate the regions. However, instead of doing it separately it uses a homogeneity criterion. This process or algorithm is named split-and-merge. Apart from dividing the image it also merges those regions that are similar. This process is repeated until all regions pass the homogeneity test. We could say that this process is aiming to group similar pixels together, just as the growing algorithm, but they do so in different ways. The main difference between the two methods is the way they approach the segmentation process. While split-and-merge starts by dividing the image into smaller regions and then merges similar ones, region growing starts with seed points and expands the regions by adding similar neighboring pixels.

Aside from those algorithms we also have the flood fill algorithm which actually has some similarities with the growing algorithm as it grows in a concrete situation, a decided section. However it also considers other things as the value of the chosen point and it uses recursivity to compute the new values of the unvisited neighbors. This process is controlled with a tolerance that determines how much it expands its recursivity and so the new color defined by the user. Let's study it more in detail. This algorithm is used to fill an area of a bitmap or an image with a specified color. It works by starting at a pixel location (seed_point) and checking if the color of the starting pixel is the same as the

target color (determined by the user). Then If it is the same as the target color, fill it with the replacement color and mark it as visited. Lastly, it recursively checks its neighbors until the entire area has been filled (have the same color). At the end this algorithm is effective when dealing with connected regions of similar intensity or color. Its simplicity and efficiency make it suitable for various image segmentation and processing tasks, especially in scenarios where the shape and extent of regions may vary (e.g. to separate foreground and background regions in an image. Besides to fill small holes or gaps in images)







5 CLUSTER-BASED SEGMENTATION EXERCISE

5.1 DESCRIPTION OF ALGORITHMS

Another method to segment an image is Cluster-based segmentation. This method is an unsupervised machine learning algorithm used to group data points into groups/clusters based on their similarity. K-means starts by randomly initializing K cluster centroids (K = desired number of clusters). Then, each pixel is assigned to the cluster whose centroid is closest in terms of distance (usually Euclidean distance). This way the initial clusters are formed, but then we will recalculate the centroids of the new clusters as the mean of all the pixels assigned to that cluster. We will repeat this process until convergence, which is when the assignment of pixels to clusters and the update of centroids is stabilized.

K-means has some limitations:

- It is sensitive to the initial placement of the centroids, so the algorithm could converge to a local minimum.
- It assumes that clusters are spherical and of equal size, but in the real world they have complex shapes and their size varies.
- The fixed number of clusters from the start is a problem is the K is chosen wrongly, since it can lead to over-segmentation or under-segmentation.
- It is sensitive to outliers, which can affect the centroids, leading to suboptimal segmentation.
- If the clusters in an image have uneven distributions, K-means may struggle to segment them accurately.

A different approach from K-means is Gaussian Mixture Models. GMM is also an iterative clustering algorithm, but they differ in many ways:

- GMM assumes that pixels are generated from a mixture of several Gaussian distributions, and each cluster is associated with a Gaussian distribution.
- Unlike K-means, GMM does a soft assignment for each data point: instead of assigning one cluster (hard assignment), it assigns probabilities of belonging to each cluster.
- GMM has more parameters to estimate than K-means: for each cluster it needs to estimate mean, covariance matrix and weight. The covariance matrix allows clusters to have different shapes, so there is more flexibility. On the other hand, K-means only estimates clusters' centroids.

In conclusion, GMM is preferred when clusters have varying shapes and sizes and when we want soft assignments, while K-means is best for cases where clusters are spherical and equally sized and when computational efficiency is crucial.

5.2 ANALYSIS OF FUNCTIONS

In this case, the analysis of the functions and its parameters will be specific for the "lenna" image. Depending on the image, we would probably change the parameters to adjust which performs the best. The analysis of the results that we include here in the Report is also included in the notebook file.

5.2.1 K-MEANS FUNCTION

5.2.1.1 Analysis of hyperparameters

- image: np.ndarray, original RGB input image.
- color_space: str, color space to convert the image ('rgb', 'hsv', 'lab', etc.).
- auto_select_clusters: bool, whether to perform an automatic selection of the number of clusters.
- max_clusters: int, maximum number of clusters.
- init: str, parameter for KMeans().
  - If init == "k-means++": selects initial cluster centroids using sampling based on empirical probability distribution of the point's contribution to the overall inertia. Speeds up convergence. (default)
  - Elif init == "random": choose n_clusters rows at random for initial centroids.
  - Elif init == array of shape (n_clusters, n_features): it gives initial centers.
- n_init: int, parameter for KMeans(): number of iterations with different centroid seeds. (default: n_init=10)
- max_iter: int, parameter for KMeans(): maximum number of iterations. (default: max_iter=300)
- tol: float, parameter for KMeans(): tolerance of Frobenius norm of difference in centroids of 2 consecutive iterations. (default: tol=1e-4)
- random_state: int, parameter for KMeans(): random number generation for centroid initialization. If int -> randomness is deterministic. (default=None)

5.2.1.2 Analysis of results

For the analysis we will try different parameters to see which adjust better to our image.

1. **color_space:** K-means can vary across color spaces because the intensity values for the pixels are different. RGB represents colors with Red, Green and Blue channels; HSV separates color information in hue, saturation and value components; and Lab consists of 3 components: L* (lightness), a* (green to red), and b* (blue to yellow).

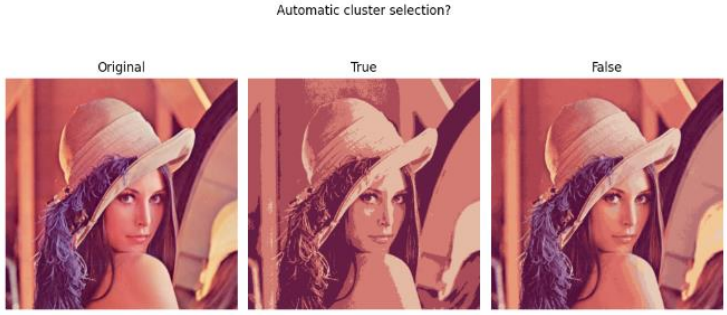
   In Lab, for instance, the perceptual difference of colors is more consistent, that's why its clustering result looks less detailed even if it has used the same K value as HSV.

   Let's consider an image where color information is represented by hue, that would be an image with different colored cobjects. In this case, performing K-means clustering in HSV color space may result in more natural clusters compared to RGB.

2. **auto_select_clusters:** If we don't apply the autoselection of clusters, which selects which is the best number of clusters for each image (best K), the number of clusters will be set to the max_clusters (in this case 20). As we can see, the result of the image segmented without selecting the best K is not well segmented.

But, how does it select the best K? It performs K-Means iteratively for different values of k from 2 (minimum possible k) to the maximum number of clusters we implied. For each K result:

First it performs fit_predict, which computes cluster centers and predicts cluster index for each sample. Then, it performs calinski_harabasz_score with the reshaped image and the labels we got from fit_predict. This function computes a score defined as a ratio of the sum between-cluster dispersion and of within-cluster dispersion. This way we can know how well a K performed. We will keep the best score we get and use that k for KMeans().



3. **max_clusters**: By choosing different values for the max_clusters, since we have the auto_select_clusters=True, we are choosing how many values of K the algorithm considers. If the auto_select_clusters = False, the max_clusters value would set the K value for the algorithm. The smaller the K value, the more segmented the image will be, but a very low number can lead to over-segmentation. On the other hand, the larger the K value, the less segmented the image will be, and it can lead to under-segmentation.

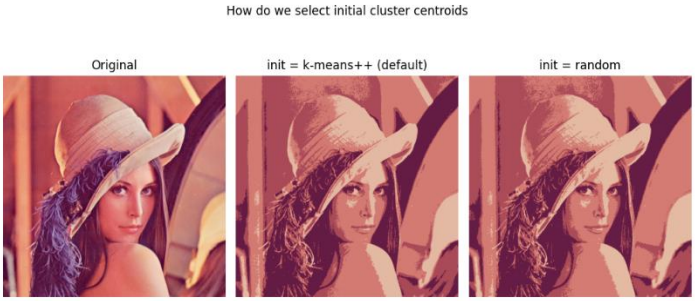
In this case, we can see that even if we increase the number of maximum clusters, the algorithm will always choose the most optimal K (which is 4 in this image), but since the number of maximum clusters is higher, it will look for more options than a low number of maximum clusters. By looking at more K options, the algorithm is more computationally expensive: it takes less that 4 minutes to compute the first 3 examples (5, 10 and 20 clusters) but more than 8 minutes to compute the last example with max_clusters=50.

4. **init:** By changing the "init" parameter we are changing the way the algorithm will choose the initial centroids, which is crutial for the final clustering result. If we choose the "k-means++" mode, it will select the initial centroids using sampling based on empirical probability distribution of the point's contribution to the overall inertia. On the other hand, by choosing "random" mode, it will choose the values at random.

   If we choose "random" instead of "k-means++", the initialization will be faster (because it's random) and we can have a more diverse set of starting points. However, it is more probable to converge to a local minima if bad initial points are chosen, and it will give less deterministic results. This is why the default mode is recommended.

   There is another mode that we haven't tried, which is passing an array with (n_clusters, n_features), which means that we would choose exactly the centroids the algorithm starts with.

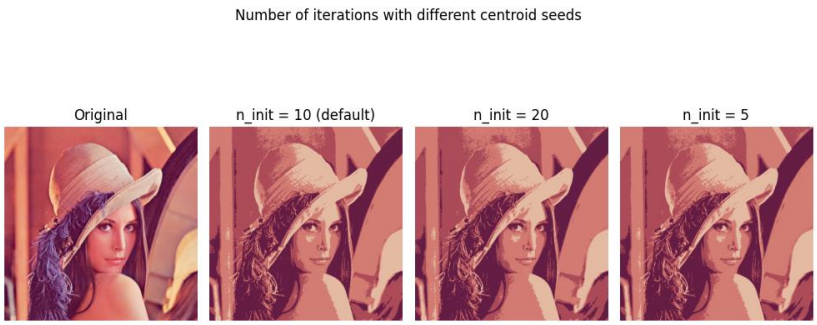
   In this case, the results don't change based on this parameter.



5. **n_init:** The n_init parameter controls the number of times the K-means algorithm is run with different centroid seeds. he final result is the one with the lowest inertia, which is the sum of squared distances between data points and their assigned centroids.

   If we choose a lower n_init the computation will be faster because the algorithm is executed fewer times, but it will make the algorithm more sensitive to the initial placement of the centroids, because with less initializations there is a higher risk of getting stuck in local minima (a suboptimal solution).

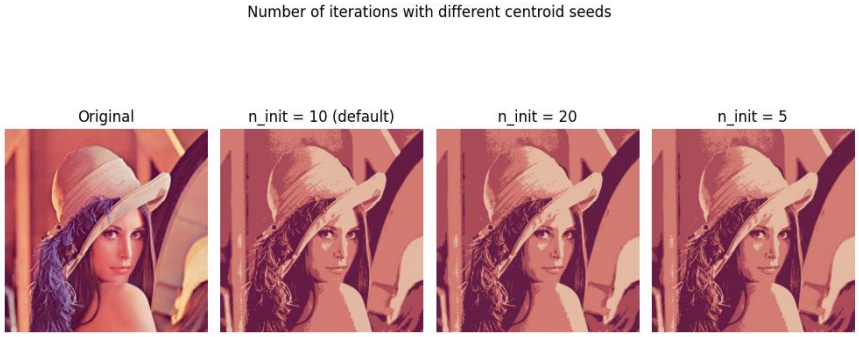
   On the other hand, choosing a higher n_init will increase the robustness of the algorithm (reducing initialization sensitivity), so it is more likely to find the global optimal solution. It also will have more probability to find the best clusters since it will explore a larger search space. However, the computational cost will be increased because we increase the number of executions.



6. **max_iter:** The max_iter parameter in K-means clustering determines the maximum number of iterations the algorithm is allowed to run during a single initialization. It controls how many times the centroids are updated in an attempt to converge to a stable solution. If the algorithm does not converge within the specified number of iterations, it stops, and the current centroids are considered as the final result.

   If we choose a lower max_iter, it can lead to a faster convergence, especially if clusters are well separated, but it may cause the algorithm to converge to a suboptimal solution because it stops before reaching the true convergence point.

   On the other hand, choosing a higher max_iter provides more iterations for the algorithm to converge. This can be useful when dealing with complex data or when the clusters are not well-separated. However, it will be more computationally expensive.



5.2.2 GMM FUNCTION

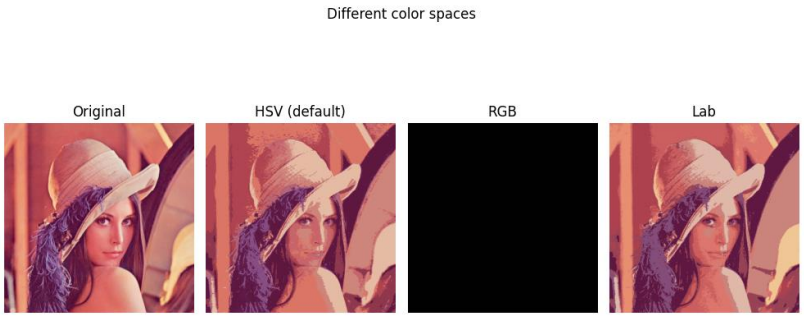
5.2.2.1 Analysis of hyperparameters

- resize_factor: int, factor by which the image is resized to reduce computation.
- color_space: str, color space to convert the image ('rgb', 'hsv', 'lab', etc.).
- n_components: int, number of components for GMM. (default = 1)
- max_components: int, number maximum of components for GMM.
- covariance_type: str, type of covariance parameters to use.
  - ‘full’: each component has its own general covariance matrix. (default)
  - ‘tied’: all components share the same general covariance matrix.
  - ‘diag’: each component has its own diagonal covariance matrix.
  - ‘spherical’: each component has its own single variance.
- normalization: bool, whether to normalize the image data.

5.2.2.2 Analysis of results

For the analysis we will try different parameters to see which adjust better to our image.

1. **color_space:** As we said before, the same image in different color spaces has different intensity values for each channels because they represent different color information. Therefore, we will see different clustering results.

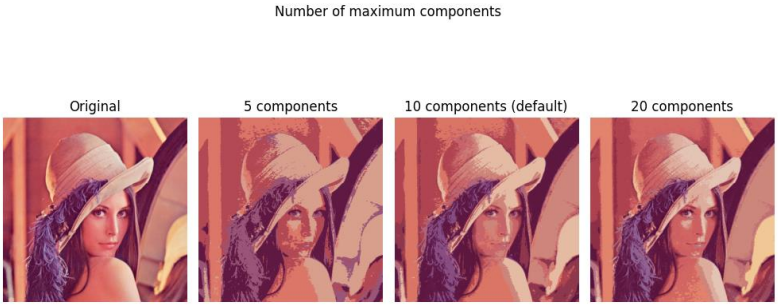


2. **max_components:** The max_components parameter controls the maximum number of components (clusters) that the GMM will try to fit to the data. We are trying to find the best value for the number of clusters by doing the following:

We consider a range for the number of clusters from 2 to max_components. The higher this parameter, the more numbers we will consider it for the most optimal number of clusters. If the number is too high, there is a risk of overfitting.

We try all the numbers in the range by computing the Gaussian Mixture and calculating a score of quality. This score is computed with .aic (Akaike), which is a measure of quality that takes complexity into account. The lower this score is, the better. We will choose the number of components that give the best score.

In this case we see that by increasing the number of components, for this image the algorithm chooses a higher number of clusters, which means that it is more optimal than choosing a lower number of clusters.



3. **covariance_type:** The covariance_type parameter in GMM specifies the type of covariance matrices used by the model.

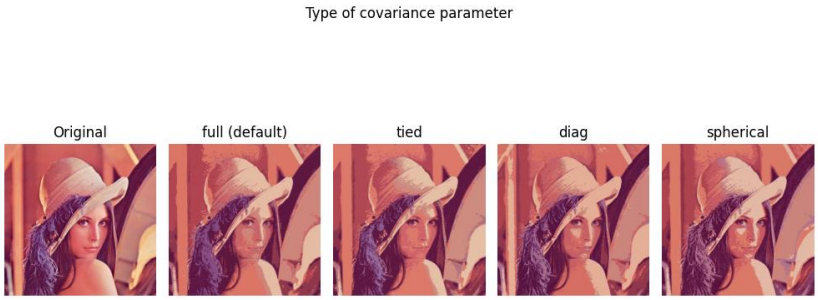
"full": is the default option where each component in the mixture has its own general covariance matrix. This is the most flexible option but also requires estimating a large number of parameters, so it is computationally more expensive. It allows for elliptical, rotated, and correlated clusters.

"tied": here all components share the same general covariance matrix. This reduces the number of parameters to estimate and can be computationally more efficient. But, it assumes that all clusters have the same shape, size, and orientation.

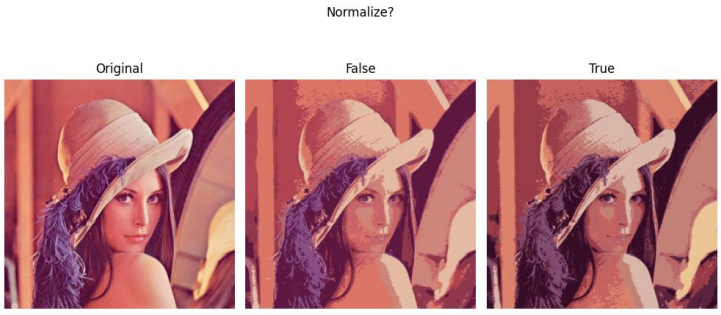
"diag": now each component has its own diagonal covariance matrix (no cross-covariances). This assumes that the dimensions are uncorrelated and can result in clusters aligned with the coordinate axes.

"spherical": each component has its own single variance (scalar). This assumes that the dimensions are uncorrelated and have equal variance. It leads to spherical-shaped clusters.

In this case, the results change depending on which covariance matrix we use. In our opinion, the full and diag look the best for this image.



4. **normalization:** The "normalization" parameter determines whether the input image data is normalized or not before applying the GMM algorithm. If we normalize it, the algorithm will be less sensitive to variations in the absolute intensity of teh colors, but we will lose information about the intensity. This might be important if the segmentation task requires preserving the original intensity characteristics of the image. On the other hand, if we don't normalize, we preserve original intensities but it can become sensitive to variations. To choose one or another depends on our objectives: for scale-invariant tasks it is better to normalize, but for intensity-sensitive tasks is better not to.



5. **resize_factor:** The resize_factor parameter controls the factor by which the input image is resized before applying the GMM algorithm. The resizing operation can have implications for the computational efficiency of the algorithm and the visual appearance of the segmentation results. If we resize the image we will reduce computation time but also we will lose details in the segmentation.

5.3 CONCLUSIONS

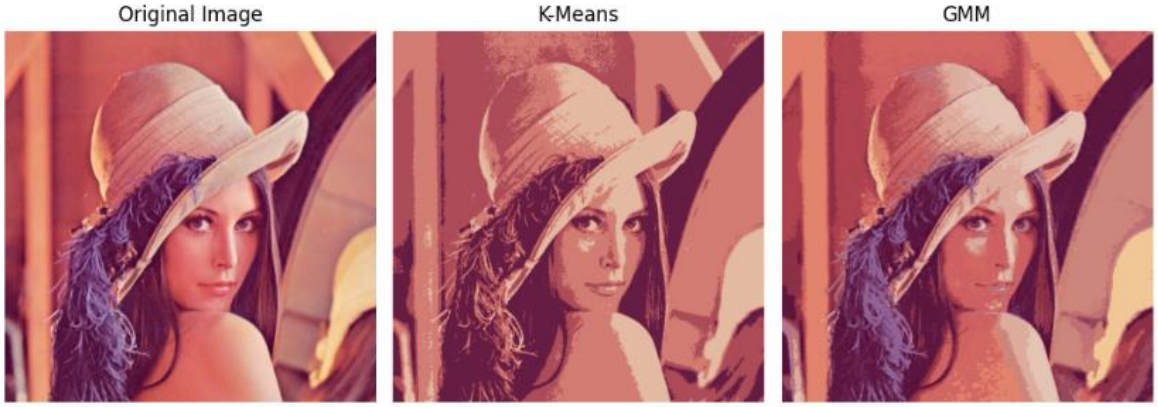
Let's compare the best parameters of each method for lenna image and analyze which method is best:

For the K-means algorithm, the best parameters for this image were:

- max_clusters = 5, because the optimal number of clusters is 4.
- auto_select_clusters = True, because we find the optimal number of clusters.
- Other parameters are the default ones.

For GMM algorithm, the best parameters for this image were:

- color_space = "hsv"
- max_components = 20, because it showed that a higher number of clusters gave a more optimal solution.
- covariance_type = "full"
- resize_factor = 1, we should not resize the image to not lose details.
- normalize = false, we should not normalize to get original intensities of image.
- Other parameters are the default ones.



We think that the method that looks the best is GMM, however, it takes much more time to execute than K-means, but the results are more detailed. K-Means took 4 seconds while GMM took 5 minutes.

5.4 EXTRA FUNCTIONALITIES

We added the resize_factor and the normalization to the GMM function.

6 DISCUSSION

1. Compare and contrast the output of edge-based and region-based segmentation techniques. In what situations would you prefer one over the other? Explain why and provide examples on how each method performs better than the other.

Edge-based and region-based segmentation are two techniques for image processing, but they are used for different purposes.

Edge-based segmentation focuses on identifying the edges in an image, which are represented by significant changes in intensity, color, texture… We could say it

emphasizes local pixel relationships because to identify an edge it normally compares to the neighboring pixels.

It is preferred when the object has clear and distinct boundaries, for example, it is effective to detect the edges of a car in a road. Following the same topic, it is also used for object recognition and tracking tasks, like identifying a license plate. Also, it is used when details of the image need to be preserved, like in medical imaging.

On the other hand, region-based segmentation focuses on grouping pixels with similar properties into regions based on color, intensity or texture. It utilizes global image information instead of local criteria.

It is preferred then when objects have a uniform texture or color within their boundaries, for example, for segmenting the sky from the ground in a landscape image. Also it is used in cases where object boundaries are not well defined (like cells in a microscopy image). Finally, it is also preferred when the image contains a lot of noise.

2. How does the choice of parameters (like kernel size or threshold values) affect the outcome of different segmentation algorithms? Since it can affect differently depending on the algorithm, explain how it affects each of them.

Threshold-based algorithms:

1. Global thresholding: the threshold value determines the pixel intensity that separates the image into foreground and background. A low threshold will result in more foreground details but can be sensitive to noise; A high threshold simplifies the segmentation but may lead to under-segmentation.
2. Adaptative Mean Thresholding: the region size specifies the size of the local neighbourhood for calculating adaptive thresholds. The larger the size, the more smooth segments (but may loss details), the smaller the size, it will adapt more to local variations.
3. Otsu's thresholding: we don't directly give a parameter but compute the most optimal threshold in the algorithm.

Edge-based algorithms:

1. Sobel Operator and Laplacian of Gaussian: the size of the kernel determines the size of the sobel and Log filters. The larger the kernels the more it will smooth the image, the smaller the kernels the more details it will keep.
2. Canny Edge Detector: 1. The sigma controls the blurring in the initial Gaussian smoothing: an appropriate sigma balances noise reduction and edge preservation; 2. The thresholds in hysteresis process control the edge detection, they choose whether to classify an edge as a sure-edge, weak edge or non-edge. We need to adapt this values depending on the image.

Region-based algorithms:

1. Watersheds: a noisy gradient image can lead to over-segmentation.
2. Split-and-Merge: the threshold value affects the balance between over and under-segmentation (lower thresholds lead to more merging)
3. Flood Fill: the seed point (starting point for region growing) affects which region is grown. Different seeds result in different segmentations.

4. K-Means and Gaussian Mixture Models (GMM): we explain all of the parameters and how they affect the results in section 5.

3. Analyze the computational complexity of the different segmentation methods. Which methods are more suitable for real-time applications? Provide "big O" notation and an estimation of the execution time for the methods.

Threshold-based algorithms:

1. Global thresholding: O(1) - efficient, execution times scales linearly with number of pixels.
2. Adaptative Mean Thresholding: O(N), N = block size - relatively fast for moderate block sizes.
3. Otsu's thresholding: O(L), L = number of intensity levels - generally fast.

Edge-based algorithms:

1. Sobel Operator: O(1) - with small kernels is computationally efficient.
2. Laplacian of Gaussian: O(K), K = kernel size - small kernels are fast but don't capture fine details well.
3. Canny Edge Detector: O(N), N = number of pixels in image - execution time depends on image size.

Region-based algorithms:

1. Watersheds: O(N log N), N = number of pixels. Involves computing a priority queue.
2. Split-and-Merge: O(N log N), N = number of pixels. Suitable for real-time applications for moderately sized images.
3. Flood Fill: O(1) - generally fast.
4. K-Means: O(I * K * N * d), where I is the number of iterations, K is the number of clusters, N is the number of pixels, and d is the number of features. Not suitable for real-time if K is too large.
5. Gaussian Mixture Models (GMM): O(I * K * N * d * d). Similar to K-Means, but with an additional dimension for covariance. Not as suitable for real-time applications with large datasets.

Most suitable for real-time applications: global and otsu's thresholding, sobel operator, watersheds, split-and-merge, flood fill.

4. Discuss the challenges of segmenting images with low contrast. What methods can be particularly effective in these scenarios?

Segmenting images with low contrast is a challenge due to the lack of clear intensity differences between objects and background.

Challenges:

- Objects and background may have similar intensity levels, leading to not clear distinction of boundaries.
- Low-contrast images are often more susceptible to noise.
- The gradient of intensity may be too small to detect edges accurately.
- Regions within the objects may appear homogeneous due to low contrast.

Methods to solve this challenges:

- Histogram equalization: enhances the contrast of an image by redistributing intensity levels.
- Adaptive Histogram Equalization: improves local contrast by applying histogram equalization in smaller regions.
- Local Binary Pattern: describes the local structure of an image by comparing each pixel with its neighbors.
- Gradient-Based Edge Detectors with adaptive thresholding: adapts to local variations, allowing for better edge detection in low-contrast regions.

5. Investigate the impact of image resolution on the performance of various segmentation algorithms. Are they sensitive or not?

Image resolution refers to the amount of detail that an image holds (measured in pixels). It is a key parameter in digital imaging and is crucial for computer vision applications. However, it is sensitive. For example, higher resolution can help capture intricate details and improve the accuracy of instance segmentation or algorithms that use local features and edges for segmentation may perform better as they have more detailed information for analysis. In summary, the sensitivity of segmentation algorithms to image resolution is context-dependent and algorithm-specific. It needs to consider the characteristics of the images and the design of the segmentation algorithm and its working method.

6. Explore the use of segmentation in object tracking (segment and recognize the same object in a video). Which methods are most effective and why?

Object tracking involves predicting the position of a given target object in each video frame. Segmentation in object tracking provides more accurate object boundaries and handles occlusions, which can improve the performance of object detection and tracking algorithms.

There are a lot of methods, however the usefulness of them depends on the requirements of the user (real-time processing, number of objects to track, complexity of the scene). For example, PointTrack is highly effective for online MOTS (Multi-Object Tracking and Segmentation) due to its speed and accuracy, while TraDeS (Track to Detect and Segment) is beneficial when detection and tracking need to be performed simultaneously.

7. Discuss the limitations of traditional segmentation techniques in handling complex images and suggest potential improvements.

Traditional segmentation techniques, such as thresholding, edge detection, region-based techniques, and clustering, often struggle or have issues while dealing with complex images. These techniques may have difficulty defining "meaningful regions" due to the wrong visual perception and the diversity of human comprehension. They may also find it difficult to handle complex scenes, occlusions, and variations in lighting conditions. Additionally, these techniques often require significant fine-tuning to support specific use cases with manually designed heuristics.

8. Evaluate the effectiveness of combining multiple segmentation techniques. Can hybrid approaches yield better results? Propose some hybrid methods. Are ensemble methods enough for improving results?

Combining multiple segmentation techniques (hybrid approaches) can be effective in image segmentation but it all depends on the condition and requirements of the task. Using this may allow them to use different segmentation techniques and at the same time to overcome their individual limitations.

Hybrid approaches can yield better results as they can handle a wider range of scenarios and complexities in images. For example, a hybrid approach combining region splitting and clustering techniques has been proposed for image segmentation.

Ensemble methods, which involve using multiple learning algorithms to obtain better predictive performance, can also be effective in image segmentation. However, it can depend on the image complexity or the way the algorithms are combined for instance. Even so, it is true that they can improve the robustness and stability of the segmentation results by reducing the risk of selecting a poorly performing model.

9. Analyze the role of color spaces in image segmentation. How does the choice of color space affect segmentation performance?

The choice of color space is a critical factor in image segmentation. They represent color details as different color channels (components) in a 3 or 4 dimensional. However, an optimal choice of color space may vary depending on the specific requirements of the task. It can significantly influence the segmentation performance as color can distinguish thousands of shades, making color-based segmentation more useful in extracting information compared to intensity or texture-based segmentation.

Moreover, the color space's choice can increase the quality of performance of processes such as segmentation and feature matching. Certain regions of interest in images belonging to a particular domain can be segmented better when represented in a certain form of color space than another.

CONCLUSIONS

In conclusion, the exploration of segmentation algorithms are various techniques applied depending on the characteristics and challenges in image processing. The implementation of these algorithms is crucial for understanding their principles and applications. About thresholding we could say that those methods separate objects from the background. Instead, the edge-based segmentation focuses on detecting abrupt changes in pixel intensity, commonly indicating object boundaries. Besides, while doing this project we have learned that the Watershed algorithm employs topographic surface concepts to delineate catchment basins in the gradient magnitude of an image. This method provides a powerful means of segmenting images into distinct regions. We also codify some segmentation algorithms as Split-and-Merge and Flood Fill (Region-based). These methods are versatile and can adapt to different image characteristics. Some other methods based on the clustering segmentation such as the GMM uses probabilistic models of hard and soft-segmentation.

In conclusion, the diverse range of segmentation algorithms promotes having the option to choose an algorithm depending on the specific characteristics of the images and the desired segmentation goals.

To end we want to remark that this project has given the opportunity to implement the theoretical learning into practice while enjoying the project and playing with different images. Nevertheless, it is true that as we are getting to know more and more the different exercises become more difficult to codify and so do what we are asked for. However, this has been a challenge for both of us and a way to gain and remember more easily all about the segmentation algorithms.