Image Segmentation Techniques: An Analysis

Craig Akiri

Department of Computer Science, The College of Wooster

May 10, 2022

Abstract

Image segmentation is a process by which regions of interest are extracted from an image or an image sequence. This is done by dividing the image into multiple segments using various algorithms. This paper explores the three major algorithms used during the image segmentation process. Furthermore, it goes into how they work and compare with each other and presents any issues with those techniques while providing some example images.

Contents

1 Introduction

Digital image processing is the practice of using computer algorithms to manipulate images in order to extract useful information. It is made up of several sub-operations that each contribute uniquely to the final image. Image segmentation is one of the more significant types of image processing. Due to its importance, numerous algorithms have been developed to refine the techniques used to partition images into multiple segments. More precisely, image segmentation involves labeling pixels in an image that share particular visual characteristics. Each new region is comprised of pixels that share some characteristic or computed value such as color, contours, texture, and objects. It the case that there is no perfect algorithm for image segmentation. This is because each image may work best with a different algorithm. Certain characteristics such as color, for example, are best handled by threshold or clustering-based segmentation algorithms. Therefore, hybrid and AI techniques have been used to expand the range of images it will work on. This paper will cover three major algorithms developed to deal with this problem namely, threshold based segmentation, edge based segmentation, K-means clustering.

1.1 The Computer Vision Pipeline

Image segmentation is part of a larger process known as computer vision. This is the name given to the set of image processing techniques involved in attempting to provide computers with a high-level understanding of the visual world by simulating how humans see [10]. Computer vision has allowed a vast range of technological advancements, from robot vision to autonomous driving, facial recognition to cancer cell detection. Industries utilize computer

vision to streamline processes, improve consumer experience and enhance security through biometrics. Manufactures and large-scale farmers can use it to spot defects in products on the assembly line and produce in the fields respectively. In order to fully understand computer vision as a process we can visualize it as a pipeline as seen in Figure 1. The computer vision pipeline is a chain of processes whose output is the input of the next operation in the chain [3].

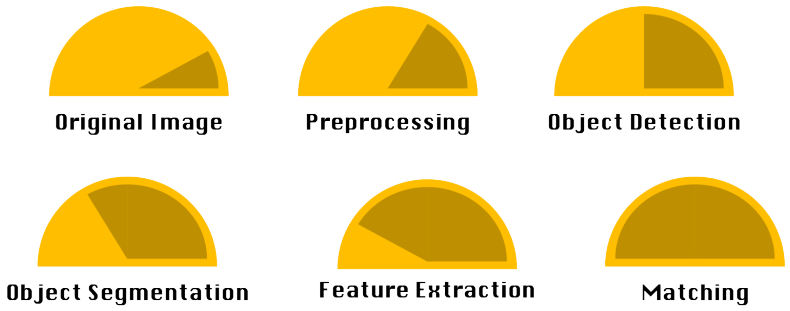

Figure 1: The Computer Vision Pipeline modified from [3]

We start with the original Image which is then taken through some preprocessing this may involve techniques such as noise reduction and rescaling to improve object detection and segmentation accuracy. The processed image is then delivered to the object detection model that Identifies and classifies the subject in the image and creates a rough bounding box around it essentially localizing the subject from the rest of the image. This localization then becomes the input for the object segmentation phase. Through image segmentation we then refine the localization by outlining the object's boundary more precisely [2]. A typical detection and segmentation is shown in Figure 2 below. More idiosyncratic features can then be extracted using the segmentation which is then matched to a known dataset.

In this paper we shall explore in detail some of the methods involved in the object segmentation step.

Figure 2: Example of object detection and segmentation adapted from [2]

2 Image Segmentation Algorithms

Image segmentation allows us to understand an Image at a more granular level compared to other computer vision tasks. While object detection is helpful in locating, classifying and tracking an object within an image or image sequence, image segmentation allows for the definition and understanding of the shapes and boundaries of those objects [5]. The importance of image segmentation can readily be seen in fields such as medicine during cancer cell detection. This is because the shape and relative size of the cells are important in determining whether the cancer is malignant or benign. This cannot be achieved using just object detection and classification because they do not give as much information. Furthermore, recent advancements in object detection have also come to depend on image segmentation. By training detection models using segmentation researchers have been able to improve category recognition, localization accuracy as well as improving context awareness (for example a boat is more likely to appear on the water rather than in the sky) [18]. For this reason, image segmentation can be viewed as a superset of classification and detection tasks and one

of the most important steps in computer vision [2].

In conjunction with being one of the most important domains in computer vision, Image segmentation is also among the oldest, hence there have been numerous techniques developed to tackle this problem [2]. However, this paper will limit its scope to three of these techniques; Threshold based segmentation, Edge based segmentation, and K-means clustering. A banana was picked as the subject that will segmented using these various techniques. This was a rather arbitrary decision nonetheless it was also selected because it contains distinct enough features that can be segmented for using the aforementioned methods. In order to implement these algorithms, we will use the OpenCV-python library which has some of these parameters predefined as functions.

2.1 Threshold based segmentation

This is the simplest segmentation technique where a threshold is set based on the variation of intensity between the pixels belonging to the object and those belonging to the background [8]. In order to differentiate the pixels, we are interested in from the rest (which will be discarded), we perform a comparison of each pixel's intensity value with respect to a threshold that is determined based on the subject of the image. The threshold can be considered as a constant value where pixels that have values greater than the threshold are set to 1 (white) while pixels with values less than the threshold are set to 0 (black) [17]. The image is thus converted into a binary mask. This process is often referred to as binarization [2]. It is important to note that this process requires that the image has little to no noise (random undesired variation in brightness and color) but most importantly the subject should have a

higher intensity than the background to avoid contaminating the mask. One way to mitigate color contamination is by defining a range of threshold values. Take for example our subject, a banana, we know that its color may range from greens to yellows and even browns, hence colors like blues or purples can be ignored. In doing so we have refined the criteria for the threshold to only include colors that appear on the subject as seen in Figure 3.

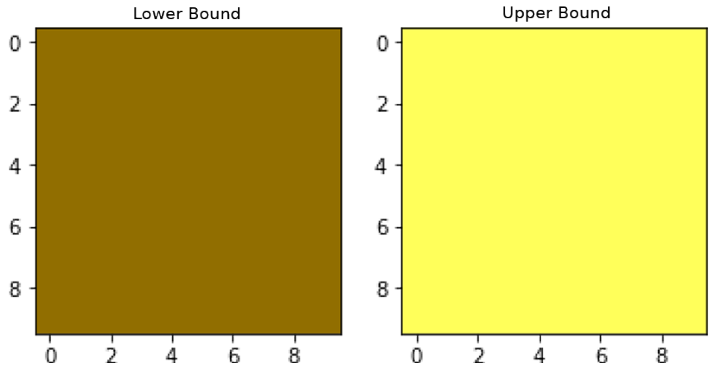

Figure 3: Banana color range

After Identifying the color range, we need to convert the image into the HSV color space in order to perform the threshold. HSV is a good choice for segmenting by color because individual colors are more localized and visually separable compared to the standard RGB color space [13]. Furthermore, the HSV color space is more meaningfully related to the way humans perceive color. This color space is also more robust towards external lighting changes this gives us more legroom when it comes to defining our bounds. When put into action we can see already that thresholding using a color range results in a more accurate mask than before Figure 4.

Depending on the image this can be the final step in the thresholding process, however, as we can see in the in the image above there are still some unwanted sections. We can clear these sections by applying some morphological operations. Morphological operations are a

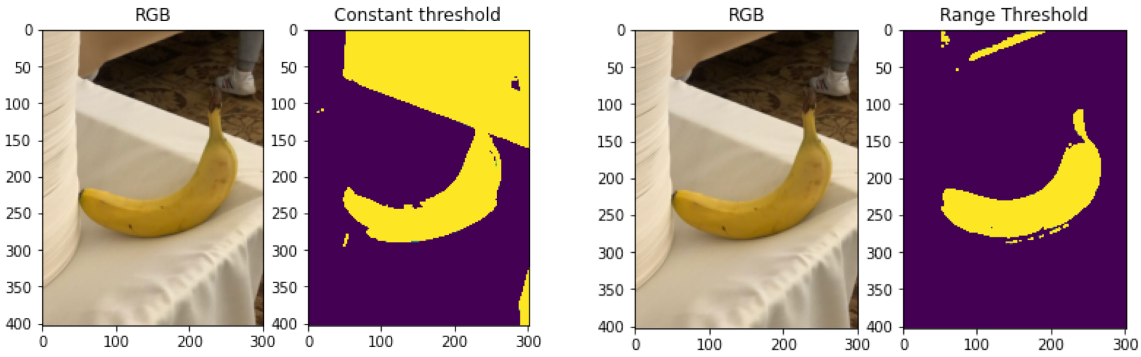

Figure 4: Comparison between Global thresholding and HSV Range thresholding

set of operations that process images based on shapes. These operations include erosion and dilation and are used to remove noise from the mask as well as separate loose parts [9]. Erosion works by defining a kernel, which is the shape used to apply the transformation, the function then iterates through each pixel such that if the pixel is within the bounds of the kernel are maintained and those outside if their value is 1, they are set to 0. This overall decreases the white region of the image by diminishing its features. On the other hand, dilation works in the opposite manner growing the white region. When the two are used together they help to reduce the noise and create a cleaner mask as shown in Figure 5.

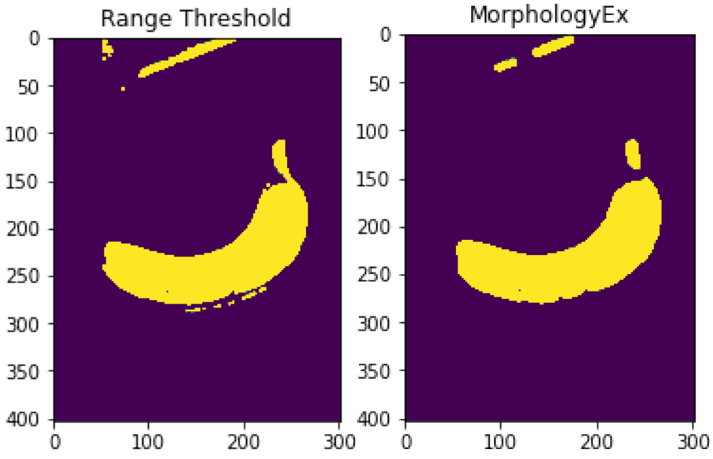

Figure 5: Before and After Morphological operations

Now that we have cleaned our mask, we can remove the remaining patches by estimating the area of each patch and selecting the largest one. The largest area is assumed to be the object being segmented because along the image segmentation process, we are separating the foreground from the background. This process makes the subject of the image clearer and more prominent such that we can conclude, most of the time, that the subject has been correctly identified. The final mask as shown in Figure 6 is the result of successful segmentation.

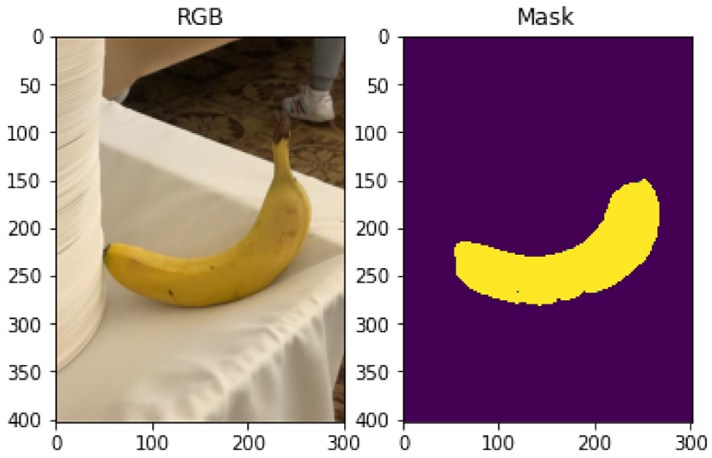

Figure 6: Final mask after selection

There may be some instances where even after all these steps the mask might have holes, in this case we perform a flood fill operation on the mask. This basically creates a duplicate of the mask and fills the empty areas from the top left pixel to the bottom right. The flood filled mask is then inverted and combined with the original mask using a bitwiseand operation (a boolean operation that maintains the union of the two).

2.2 Edge based segmentation

Edge based segmentation, which is also referred to as edge detection, is the task of identifying the different edges of objects in an image in order to separate them from the background [7]. From a segmentation perspective, it divides the image by observing the change in intensity or pixels of an image [7]. This means that the edges mark areas of discontinuity in gray levels, color values, texture, etc. Edge detection has become a popular segmentation technique because, just as thresholding, it is simple and quick to implement. This allows for unwanted and unnecessary information to be removed quickly reducing image size and intricacy [17]. The quality of the segmentation can be improved be connecting all the edges into edge chains that match the image boarders more accurately to create a seamless boarder. This makes it easier for post-processing techniques, such as flood filling, to be applied as well as reduce the number of chunks made of small irregular edges. The resulting image is an intermediate segmentation of the object and not a full mask hence more operations have to be carried out after detecting the edges [15]. Therefore, the goal of edge-based segmentation is to provide a reference point from which other segmentation tasks can start from [15].

There are many edge-based segmentation methods developed. Canny, Prewitt, Deriche and Roberts cross are some of the most popular edge detection operations [17]. These algorithms work by creating a filter that is calculated image gradients in the x and y coordinates of the spacial plane then convolving these filters over the given image [12]. In this paper we shall look at the canny edge detection algorithm due to its reliability in use and it is readily implemented in OpenCV.

2.2.1 Canny Edge detection Algorithm

Canny edge detection was developed by John F. Canny in 1986. Regardless of its age it has become one of the standard edge detection methods [11]. JFC laid out a general criterion for how an optimal edge detection algorithm should perform. These criteria include: [11]

1. Detection: it should have a low error rate, which means the probability of falsely detecting non-edge points should be minimized by maximizing the signal to noise ratio.

2. Localization: the detection should closely match real edges in the source image.

3. Number of responses: this is implied in the first criterion, nonetheless, it can be explicitly defined as, one real edge should not result in more than one detected edge

These three criteria make the canny edge detection algorithm one of the most strictly defined methods that provides good and reliable detections. However, all the processing is not handled by the algorithm alone and since edge detection is an intermediate segmentation method it requires some pre and post processing. Before the segmentation is carried out the region of interest must be identified with good contrast between the object and its background. Next the image is converted into gray scale to reduce the computational requirements as seen in Figure 7.

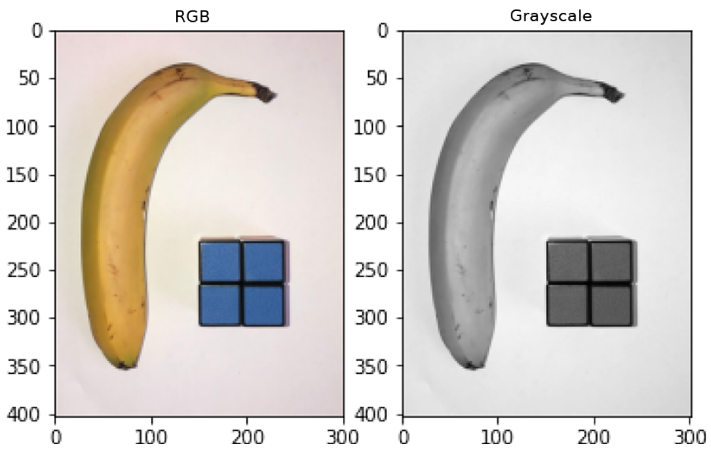

Figure 7: Converting RGB image to Grayscale

After applying the necessary pre-processing techniques, the edge detection algorithm can be applied. The canny algorithms runs in 5 steps this paper shall not go into too much detail more information can be found at [11]:

1. Noise reduction: This involves blaring the image using a Gaussian filter to remove noise as shown in Figure 8.

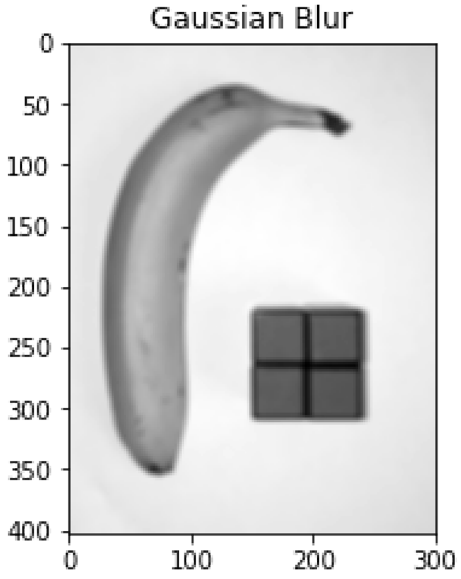

Figure 8: Applying Gaussian Blur filter

2. Finding gradients: The gradients are defined where the grayscale intensity of the image changes the most.

3. Non-maximum suppression: The blurred edges are turned into sharp edges by preserving all local maxima.

4. Double thresholding: Remove any unwanted edges by thresholding, edges that are within the threshold are maintained.

5. Edge tracking by hysteresis: Final edges are determined by suppressing all edges that are not connected to a very certain strong edge.

Instead of having to define all these steps the canny algorithm has already been implemented in the OpenCV library. This allows use the robust algorithm with little effort. The canny function takes in a preprocessed image as discussed above and a kernel size. A kernel is basically a matrix that represents how you should combine a window of n-by-n pixels to get a single pixel value. With regards to edge detection a kernel is the window of n-by-n pixels that is used to convolve the edge filter onto the image. The result is the perimeter of the objects in the image as defined by the edges seen in Figure 9.

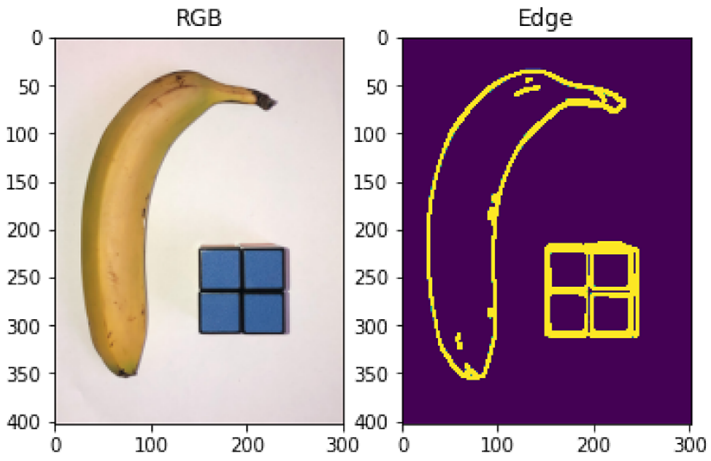

Figure 9: Canny Edge Detection

Post processing can then be applied to the edge detection to further separate the subject and create a binary mask. Such operations may include filling in the contours using the draw contours method described by the OpenCV library or applying a morphological operation similar to the one used in thresholding-based segmentation to close the perimeter of the objects. The final image is an exceptional binary mask of the subject as seen in Figure 10.

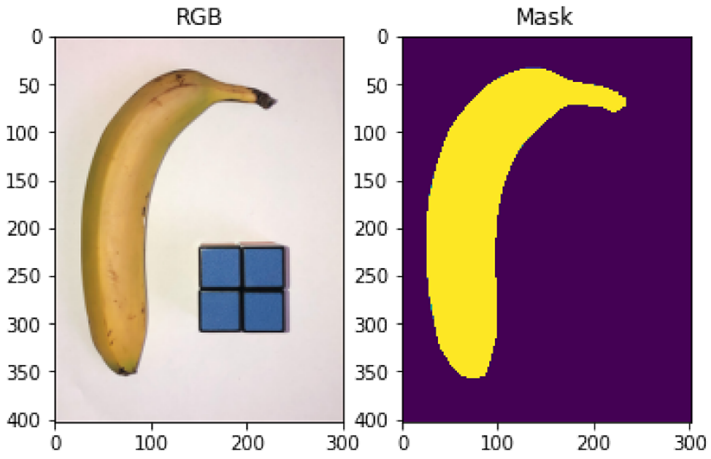

Figure 10: Final mask after post processing

2.3 Clustering based segmentation

Clustering algorithms are the simplest form of unsupervised learning algorithms. An unsupervised learning algorithm is a type of algorithm that discovers patterns from unlabeled data. These algorithms can help with finding features in a data set that can be useful for categorization [6]. As the name suggests clustering involves dividing a data set into several groups or clusters such that each data point in the group is like another data point within the same group [12]. Some of the popular clustering algorithms include fuzzy c-means (FCM) and k-means clustering [17]. For the purposes of image segmentation k-means will be used. This is because it is a simple and efficient algorithm to implement. On top of that there are pre-define methods defined in the OpenCV library.

2.3.1 K-means algorithms

K-means is an iterative clustering algorithm which maximizes for the highest value after each iteration. At the end of operation, the k-means algorithm partitions (N) observations into (K) clusters. Each observation is assigned to a cluster based on the nearest mean. Take for example a 2D data set for better visualization, Figure 11 [16]. Initially we have an unsorted array of data points, (K) number centroids are then randomly placed on the data set to begin clustering. A centroid simply defines the localization of which one of the mean values are calculated. The observations are assigned to the cluster with the least distance from it until each observation is classified. The centroids are then recalculated based on the mean of the cluster then the next iteration is executed. The process then continues until the algorithm converges to a good solution [16].

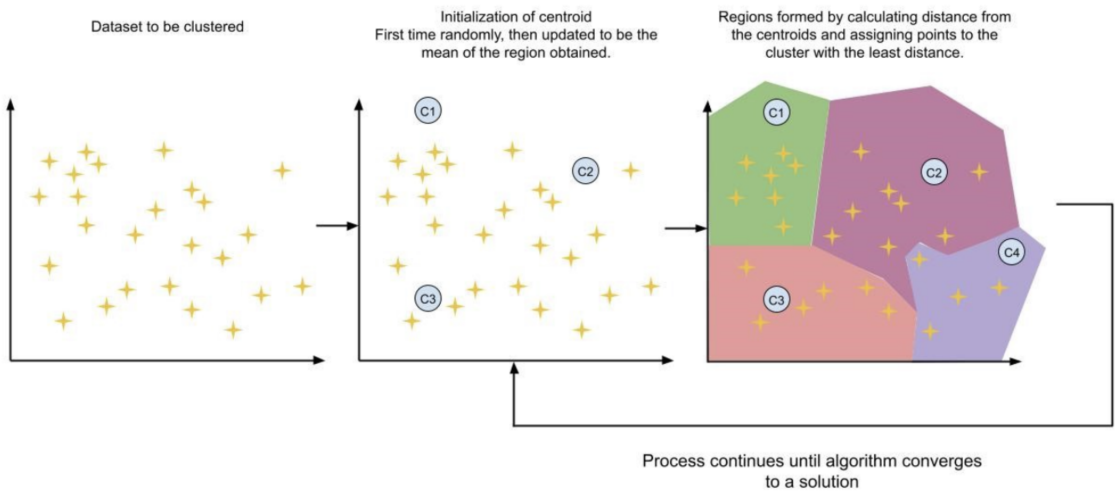

Figure 11: KMeans Clustering workflow [16]

Now that we have defined how the algorithm works, we can go into a bit more detail in how each step is executed. Although, K-means is an iterative algorithm we can define it as a series of 5 steps [1]:

K value The K value is user defined, although, this is antithetical to the unsupervised learning idea and adds friction to the use of the algorithm, it is still a useful algorithm for image segmentation. Since we are using images, calculating the k value does not require extensive knowledge of the domain [1]. Instead, the K value is the number of colors present in the image. Take for example the image in Figure 12. The banana is resting on a black background this means the appropriate K value is 2.

Calculating the centroids Before calculating the centroids the image needs to be re-shaped. This is because k-means works best with 2D data sets whereas images represent a 3D data set (height, width, RGB value). To fit the 3D image into a 2D space we flatten the image to create a 2D array the points representing the rows and the RGB values being the

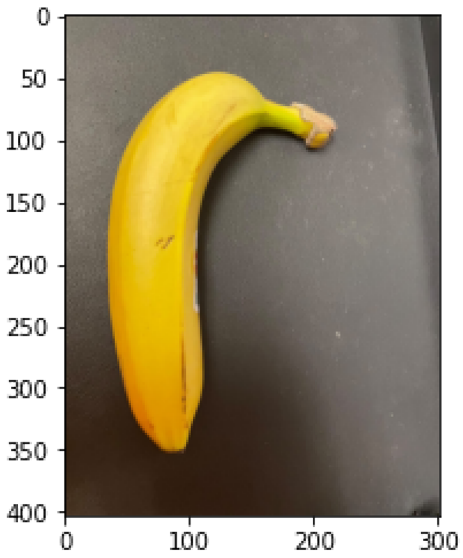

Figure 12: Banana on black background

column. For each K value a center a centroid needs to be calculated as a starting point for the clusters. The first calculation of the centroid does not need to be precise; this is because the algorithm runs iteratively refining the centroids. An arbitrary centroid can be calculated using a random/ pseudo random generator. Fortunately, OpenCV defines one particularly for this use case.

Clustering Once the K-value is defined, and the centroids have been calculated the algorithm goes through each row in the 2D array and compares its column values to that of the centroid. If the value is closest to that particular centroid, then it gets put in the cluster. This is repeated for each point in the array until completion. The size of the array can simply be calculated as the area of the image because the algorithm loops through each pixel. For example, for a 500 x 500 pixel image the algorithm will have to go through 250,000 data points. This can become a problem for high resolution images as the number of data points can slow down the algorithm since it will be run iteratively.

Recalculating the centroids After each iteration the centroids are adjusted gradually until its position is the mean of all members in the cluster.

Stopping Criteria The clustering and recalculating stages are repeated until a predefined accuracy level is reached or after a certain number of iterations has been achieved. The output is an array of labels of each pixel assigning it to a particular cluster and an array of centroids with a corresponding cluster label. An image can be derived from this data by reshaping it back into 3D space as seen in Figure 13. A mask can then be generated by separating the desired cluster and thresholding image into a binary mask.

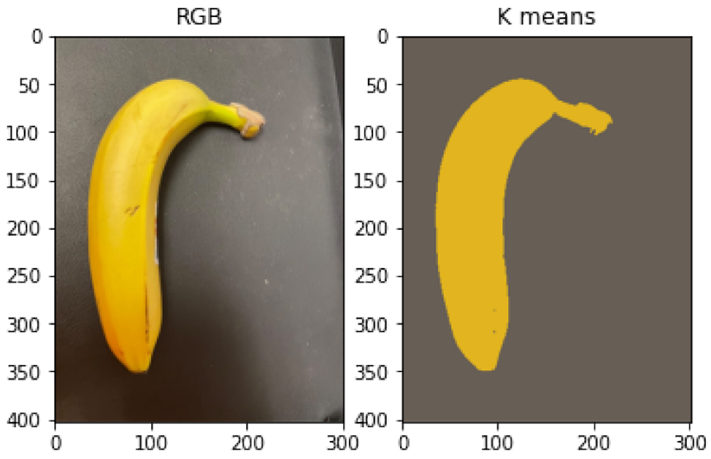

Figure 13: Comparing RGB image and KMeans segmentation

3 Software Application

This section will detail the programming language and external libraries used.

3.1 Software Description

The software portion of this paper is a demonstration of the segmentation techniques discussed using a Jupyter notebook and the python programming language along with some external libraries. A Jupyter notebook is an open standard that defines an environment which allows for live code execution, data visualizations, annotation and explanatory text. This is very useful because it allows for everything required during coding to be displayed within the programming window instead of having to deal with external pop-up windows. It also has added benefit of being able to view intermediate data that is used to infer other steps in the programming process. The notebook can also be saved and shared without losing data calculated during its operation. Python was chosen because of its extensive libraries and its compatibility with the Jupyter standard.

The python notebook, demo.ipynb, contains the implementation for each algorithm discussed in the order represented in the paper. The notebook is divided into executable cells which when run perform the segmentation algorithms on a set of images stored in an images folder. The result is then displayed as a graph showing the original image alongside the segmented mask.

3.2 External Libraries

During software implementation I used the following external libraries:

OpenCV-python This is an open-source computer vision library that formed the basis of this research. Most of the methods implemented in the software were drafted from the predefined operations provided by the library.

NumPy It provides python with more elaborate support for manipulating multi-dimensional arrays and matrices. It is mostly used in this software to manipulate images and apply matrix calculations on them.

Pandas The pandas library provides fast, flexible and extensive data structures for working with relational or labeled data.

Matplotlib This is a well-established library for data visualization in python. It is used in the notebook to create graphs to compare each segmentation to its original images.

4 Results

This section includes the findings after benchmarking the three segmentation algorithms.

4.1 Procedure

To compare these segmentation techniques against each other, the CPU run time of the algorithm was tabulated across different image resolutions. The run time was calculated using Jupyter's in built module for function timing and the graphs and console output was turned off except for the final mask. The benchmark was run for 10 iterations at 4 different resolutions and the mean of each run was recorded. The source image used was 12 megapixels (MP) whose equvalent is a resolution of 4032 x 3024 pixels. It was then down scaled using a scale factor of 0.1, 0.5 and 0.75 to create the other resolutions. No hardware acceleration was implemented due to compatibility issues with the M1 processor and lack of documentation on the problem.

4.2 Results

After benchmarking it is clear that the resolution of the image plays a significant role in runtime of the segmentation algorithms as seen in Figure 14. Thresholding is the fastest among the three segmentation algorithms. This was expected as it also the simplest to implement requiring little to no actual calculation. Edge detection comes in second, however, it falls apart with higher resolutions leading to incorrect masks being produced. This is because edge detection is susceptible to noise. Higher resolution images contain unnecessary details that reduce the signal to noise ratio of the algorithm. Thus, lower resolution images

Figure 14: Table showing the CPU performance of the segmentation algorithms

Image Size		CPU Runtime (ms)		
h * w	Resolution (MP)	Thresholding	Edge	K-means
403 * 302	0.122	121	112	161
2016 * 1512	3.05	122	160*	1790
3024 * 2268	6.86	184	259*	4160
4032 * 3024	12.19	253	375*	7540

Performance figures reported in milliseconds (ms)

*Incorrect mask was produced

are preferred when using edge detection. Similarly, high resolution images can be downscaled to improve performance.

The K-means algorithm on the other hand suffers notably from an increase in image resolution. This is because the operation runs repeatedly for a given number of iterations. During each iteration the algorithm performs matrix calculations on a pixel-by-pixel basis for the entire image. The higher the resolution the higher the total number of pixels required to calculate, multiply this by the number of iterations and we begin to see where we are losing precious time. Therefore, since resolution begins to see diminishing scaling the image can help reduce the time complexity.

5 Conclusion

The performance of a computer vision system depends highly on its ability to perform image segmentation. This is certainly a very complicated task and al the discussed techniques have their advantages and shortcomings. Thresholding for one is fast and requires no prior information but it fails to consider spatial information and is dependent on setting accurate threshold values. Edge detection is simple and performs well in high contrast situations. Its limitation is that it is very sensitive to noise. Finally, K-means generates really good segments, however, it is heavily constrained by its computational intensity. Recent research in image segmentation techniques seeks to solve some of these shortcomings by applying novel approaches to the problem. Exploring hybrid techniques that combine two or more image segmentation algorithms could also be the topic of a future revision of this paper.

References

[1] Vatsal . K-means explained, Feb 2021.

[2] Hmrishav Bandyopadhyay. A gentle introduction to image segmentation for machine learning, 2022.

[3] Sunila Gollapudi. *Learn Computer Vision using opencv: With deep learning cnns and RNNS*. Apress, 2019.

[4] Mircea Ionescu. Tomato shape, 2020.

[5] Dan Jameson. Image segmentation guide, 2019.

[6] Daniel Johnson. Unsupervised machine learning: Algorithms, types with example, Mar 2022.

[7] Waseem Khan. Image segmentation techniques: A survey. *Journal of Image and Graphics*, pages 166–170, 2014.

[8] OpenCV Library. Basic thresholding operations, 2000.

[9] OpenCV Library. Eroding and dilating, 2000.

[10] Arm Ltd. What is computer vision.

[11] Thomas B Moeslund. Canny edge detection, Mar 2009.

[12] Pulkit S. Computer vision tutorial: A step-by-step introduction to image segmentation techniques (part 1), Aug 2019.

[13] Rebecca Stone. Image segmentation using color spaces in opencv + python, Nov 2020.

[14] S. Sural, Gang Qian, and S. Pramanik. Segmentation and histogram generation using the hsv color space for image retrieval. In *Proceedings. International Conference on Image Processing*, volume 2, pages II–II, 2002.

[15] Mrinal Tyagi. Image segmentation: Part 1, Jul 2021.

[16] Mrinal Tyagi. Image segmentation: Part 2, Jul 2021.

[17] Pavan Vadapalli. Image segmentation techniques [step by step implementation], Feb 2021.

[18] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *CoRR*, abs/1905.05055, 2019.