1 Segmentation

We wouldn't be able to extract each individual fruit from the image using typical image processing methods like thresholding and contour detection, but we can easily detect and extract each fruit using the watershed algorithm. We must start with user-defined markers when using the watershed method. These markers can be manually defined using point-and-click, or they can be defined automatically or heuristically using thresholding and/or morphological processes.

We must start with user-defined markers when using the watershed method. These markers can be manually defined using point-and-click, or they can be defined automatically or heuristically using thresholding and/or morphological processes. The watershed technique treats pixels in our input image as local elevation (called topography) based on these markers — the method "floods" valleys from the markers outwards until the valleys of various markers meet. The markers must be set accurately in order to produce an accurate watershed segmentation.

Figure 1

Now we need to remove any small white noises in the image. For that we can use morphological opening. To remove any small holes in the object, we can use morphological closing. So, now we know for sure that region near to center of objects are foreground and region much away from the object are background. Only region we are not sure is the boundary region of fruits.

So we need to extract the area where we know the fruits are. The boundary pixels are removed by erosion. So whatever is left, we can be certain it is fruit. That would work if the objects weren't touching. However, because they are touching, another viable method is to find the distance transform and apply the appropriate threshold. The next step is to locate the location where we are certain they are not fruits. We dilate the outcome as a result of this. Dilation widens the gap between the object and the background. Because the boundary region is

gone, we can ensure that any region in the background in the outcome is truly a background.

The remaining regions, whether they are fruits or backdrop, are those about which we have no knowledge. It should be found using the Watershed algorithm. These spots are usually found towards the fruit's edges, when the foreground and background collide (Or even two different fruits meet). By subtracting sure fg area from sure bg area, you can get it.

We receive certain sections of fruits in the thresholded image that we are certain are fruits, and they are now disconnected. In other circumstances, only foreground segmentation is of interest, rather than separating the mutually touching items. We don't need to apply distance transform in this scenario; erosion would enough. Erosion is simply another way to extract a certain foreground area.

Now we know exactly where the fruits are located, as well as the background and other details. As a result, we build a marker (an array the same size as the original image, but with an int32 datatype) and label the regions within it. The areas we know for sure (foreground or background) are labeled with any positive integers, but different integers, while the areas we don't know for sure are simply left blank.

The image's backdrop is labeled with 0, while other items are labeled with integers beginning at 1. However, we know that if the background is set to 0, watershed will treat it as an unknown area. As a result, we'd like to assign a different integer to it. Instead, we'll put a 0 next to the unknown region, which is defined by unknown.

Now our marker is ready. It is time for final step, apply watershed. Then marker image will be modified. The boundary region will be marked with -1. And here's the result after applying the watershred algorithm.

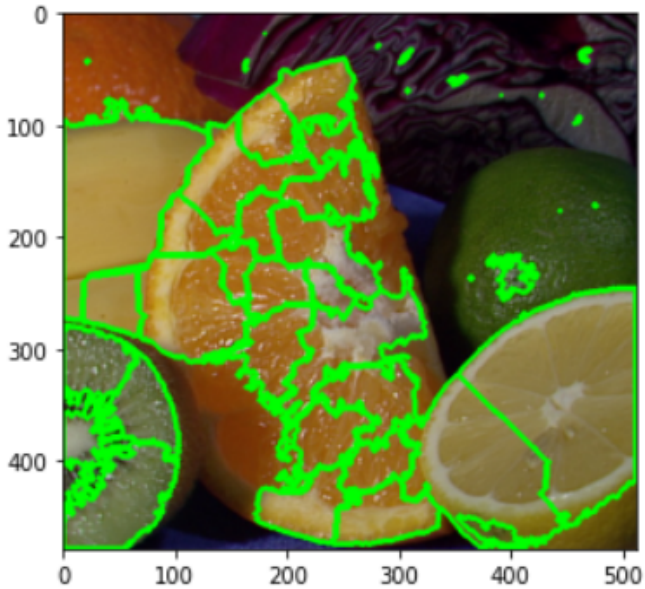

Figure 2

In the above image we can see examples of objects that would be impossible to extract using simple thresholding and contour detection, Since these objects are touching, overlapping, or both, the contour extraction process would treat

each group of touching objects as a single object rather than multiple objects. And therefore using the watershred algorithm was not the best solution.

1.1 K-Means Segmentation

The technique of decreasing the amount of colors in an image is known as color quantization. One reason for this is to improve memory. Some gadgets may have limitations that limit the number of colors that can be produced. Color quantization is used in those situations as well. For color quantization, we employ k-means clustering. There isn't anything fresh to say here. R, G, and B are three characteristics. As a result, we'll need to resize the image to fit into a Mx3 array (M is number of pixels in image). Following that, we apply centroid values (including R, G, B) to all pixels, resulting in an image with the required number of colors. We'll have to restructure it again to match the original image's shape.

Our main finding is that when the clusters overlap, k-means can be significantly improved using these two tricks. Simple furthest point heuristic (Maxmin) reduces the number of erroneous clusters from 15% to 6%, on average, with our clustering benchmark. Repeating the algorithm 100 times reduces it further down to 1%. This accuracy is more than enough for most pattern recognition applications. However, when the data has well separated clusters, the performance of k-means depends completely on the goodness of the initialization. Therefore, if high clustering accuracy is needed, a better algorithm should be used instead.

When we increased the number of K we found out that the segmentation was more recognizable for fruits.

Use the k-means for segmentation with k=3:

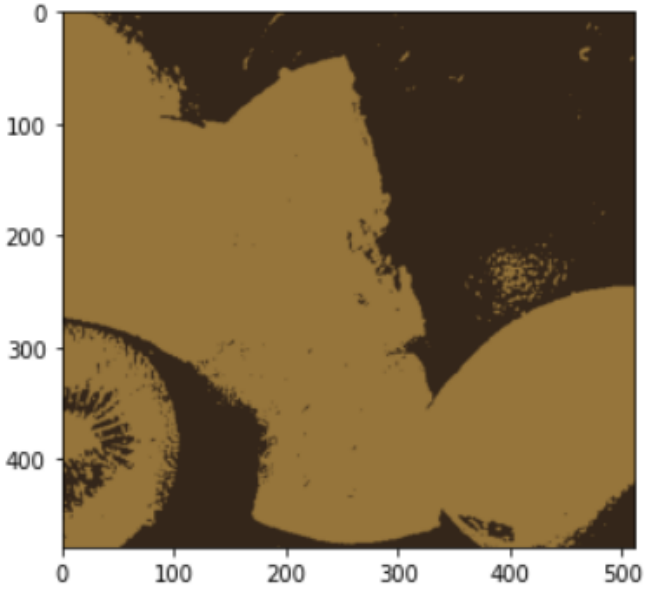

Figure 3

Use the k-means for segmentation with k=16:

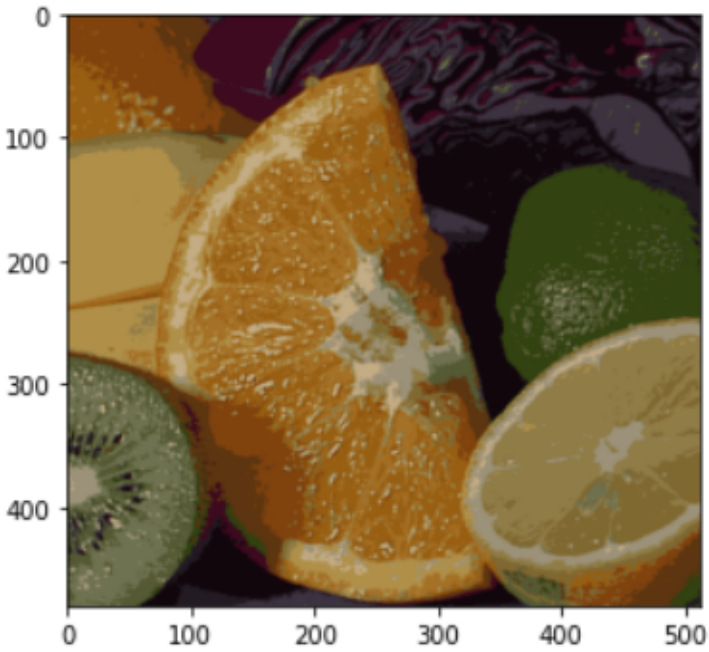

Figure 4

1.2 Mean shift segmentation

The Mean Shift takes usually 3 inputs: A distance function for measuring distances between pixels. Usually the Euclidean distance, but any other well defined distance function could be used. The Manhattan Distance is another useful choice sometimes.

A radius. All pixels within this radius (measured according the above distance) will be accounted for the calculation.

A value difference. From all pixels inside radius r, we will take only those whose values are within this difference for calculating the mean

Here, the function cv2.medianBlur() computes the median of all the pixels under the kernel window and the central pixel is replaced with this median value. This is highly effective in removing salt-and-pepper noise. One interesting thing to note is that, in the Gaussian and box filters, the filtered value for the central element can be a value which may not exist in the original image. However this is not the case in median filtering, since the central element is always replaced by some pixel value in the image. This reduces the noise effectively. The kernel size must be a positive odd integer

The best result was using the k-means for segmentation with k=16. Because it did detect the smallest details and each fruit individually.

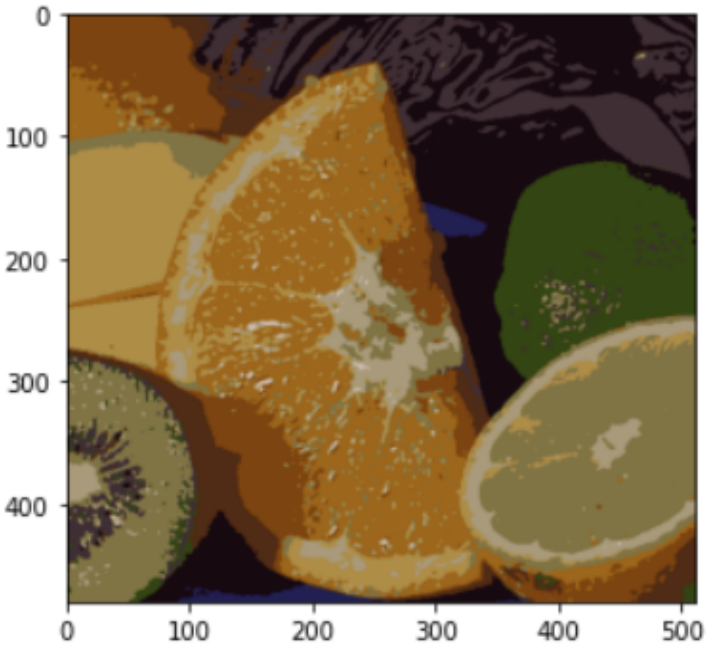

Figure 5

1.3 Thresholding Segmentation

The most basic approach of image segmentation is thresholding. It's a non-linear technique that turns a gray-scale image into a binary image by assigning two levels to pixels that are below or over a threshold value. To put it another way, if a pixel value is bigger than a threshold value, it is assigned one value (which may be white), otherwise it is allocated another value (may be black). We use the cv2.threshold() function in OpenCV.

A single-channel array is subjected to fixed-level thresholding using this function. The function is commonly used to convert a grayscale image to a bi-level (binary) image (compare() could also be used for this) or to remove noise, which involves filtering out pixels with too tiny or too big values. The function supports a variety of thresholding techniques. The computed threshold value and thresholded image are returned by the function.

Here, the matter is straight forward. If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black). The function used is cv2.threshold. First argument is the source image, which should be a grayscale image. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value. OpenCV provides different styles of thresholding and it is decided by the fourth parameter of the function. Different types are:

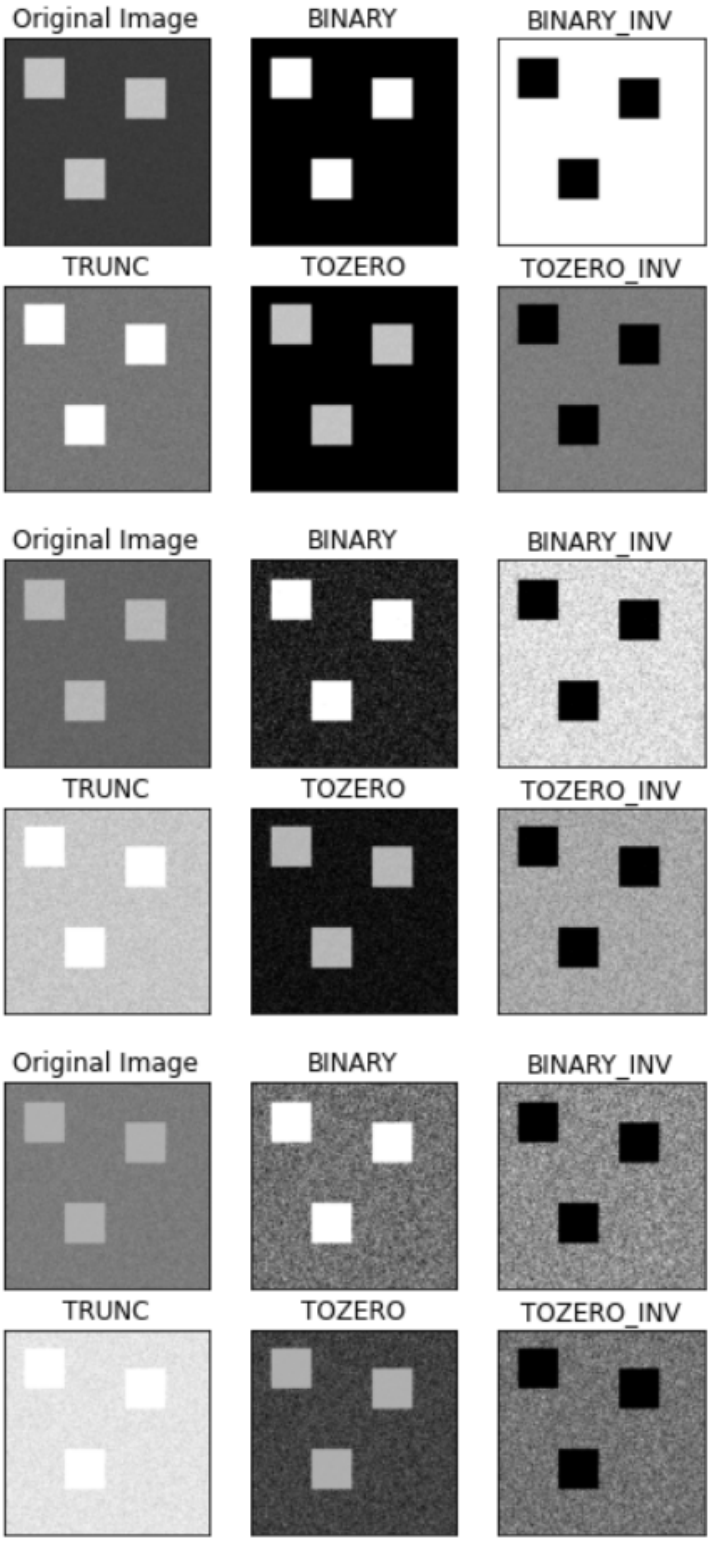
Original Image
BINARY
BINARY_INV
TRUNC
TOZERO
TOZERO_INV
Original Image
BINARY
BINARY_INV
TRUNC
TOZERO
TOZERO_INV
Original Image
BINARY
BINARY_INV
TRUNC
TOZERO
TOZERO_INV

Figure 6