Color Thresholding, Vegetable Classification

PROJECT TOPIC 4

Food classification

Group 2

Karin Pettersen, Tormod Müller, Aleksander Aaboen, Sander Island & Martin Iversen

Table of content

1 Abstract

For this report, our task was to create a script which could classify a type of food in a picture, we solved this topic by segmenting an image, using color thresholding and morphological erosion. Our goal was to find out if a meal contained any vegetables, and if so, highlight them. We have chosen to classify the vegetables based on color.

We chose to classify the colors we though were the most typical for vegetables. The tool we used to implement our approach is the *Color Thresholder* app from MATLAB's *Image Processing Toolbox*. This tool lets us segment images by thresholding the color channels based on different parts of a color space. Using this approach, we were able to create binary images with the specified colors highlighted.

We used image fusing and post processing to produce a resulting image where all the vegetables were highlighted. We found that our approach yielded good results albeit some shortcomings and problems centered around reflective light, dark colors, and the colors orange and yellow. We tested our script for a wide variety of images and through this decided that if further development would happen, we would implement more pre-, and post-processing to eliminate the problems and shortcomings we encountered.

In conclusion we have created a working vegetable classification script, which given an image with good resolution, lighting, minimal reflection, and minimal use of orange and yellow will produce a good resulting image where all the vegetables on the plate are highlighted.

2 Introduction

This project is a part of the Computer Vision course, and the purpose of the project is to receive a different type of learning experience compared to just regular class work. In this, learned to work in group with other students and obtained knowledge on various methods of image segmentation which we can apply creatively to real-life situations.

In today's society diet is important. People nowadays are more conscious about what they are eating, getting daily exercise and wellness in general. With this in mind, we were wondering*: Is it possible for us to create an application whose purpose is to classify the types of food on a plate and give information about the meal based on the types of food on the plate?*

For this project however, we have chosen to try and segment one specific type of food on a plate and highlight it, given limited knowledge and time constraints. We chose to go with vegetables since this food type has a lot of different colors and shapes.

Food classification Is the art of identifying different types of food in an area by some characteristic such as color, shape, or texture. This methodology falls under a broader concept of computer vision, called Image segmentation. Image segmentation is used to segment an image into different parts.

For the project we decided to center our approach around color. We have chosen to focus on colors that we believe are most relevant for categorizing vegetables. Thus, the goal is to implement a script with functions that will classify green, red, purple, yellow and orange colors in a plate of food. To solve this, we will be using color thresholding, image fusing and morphological opening.

We chose this project because it gave us an opportunity to go deep dive into image segmentation and learn more different techniques within this field. We also wanted to learn about how image segmentation technology is used in everyday life (like face recognition, medical imaging, and autonomous driving systems).

In this report we will focus on state of art, method, and implementation. Our discussion will give you a better idea of the process: what worked well and what we could have done differently. And finally, the conclusion will show our results and conclusion on the project topic.

3 State of the art

Image segmentation is widespread in digital image processing and computer vision in general. It is the process of partitioning a digital image into multiple parts. Which can make analyzing, post processing and working with the image easier.

Over the years segmentation algorithms and techniques have been developed to solve problems in specific areas like medical imaging, automated driving, video surveillance and machine vision. For example: Segmentation used in autonomous driving systems to help the system identify and locate vehicles, signs, lines, and other objects on the road (Figure 1).

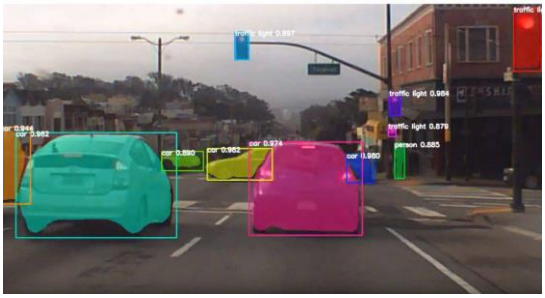

Figure 1 An autonomous car using image segmentation and classification. (Briefs, 2018)

Image segmentation is also widely used in medical context for clinical analysis and visualization of the interior of a body. Examples of some methods using image segmentation is X-Ray Radiography, Computed Tomography (CT) and Magnetic Resonance Imaging (MRI). With these methods it is possible to determine anatomic errors like fractures, tumors, cancer and other defects and diseases (Figure 2).

Figure 2 An MRI scan of a human head. (Mehta, 2014)

For our project we choose to use basic image segmentation techniques, but a different technique we could have used was machine learning. Machine learning is a branch within Artificial Intelligence that focuses on the idea that the system can “learn” from data processing. You have an algorithm, feed it a lot of data and after a large amount of processing the algorithm will be able to compute your desired result.

One increasingly popular algorithm used for image segmentation is the Random Forest algorithm. Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time (Donges, 2020). This algorithm is used in the *Classification of Food Images through Interactive Image Segmentation* research paper. In this study they use Random Forest to distinguish between different types of food. *Boundary Detection & Filling and Gappy Principal Component Analysis methods are applied to restore the missing information - pixels not detected by the algorithm* (Inunganbi, Seal, & Khanna, 2018). In their work, they end up

with results which demonstrate that the Random Forest outperforms the existing methods. (Inunganbi, Seal, & Khanna, 2018)

Another research paper we investigated was the *FOOD IMAGE ANALYSIS: SEGMENTATION, IDENTIFICATION AND WEIGHT ESTIMATION* research paper. Of relevance to us, in this research they used image segmentation to determine the regions where a particular food is located. In the research they use Local Variation image segmentation which *is a graph-based image segmentation method with weight on each edge measuring the dissimilarity between pixels* (He, 2013).

4 Method

We kicked off the project by browsing the web to learn about the different project topics we could choose from. Eventually we chose the Food-Classification topic. From there we started exploring the different methods to classify specific parts within images. After this we started to look for research articles and papers with similar topics to ours (He, 2013) , we concluded that our project fell in under the category image segmentation. Since we were classifying vegetables, we needed to define characteristics specific to vegetables.

The two main characteristics a vegetable has is color and texture. Regarding texture, we could use Image Entropy and Image filtering. These methods could help us to classify different regions within the plate of food, this could be used as a form of pre-processing before applying more segmentation techniques. We concluded that using texture analysis would be too complicated for our project case given that vegetables are vastly different in texture. Therefore, we ended up centering our project around image segmentation using colors. Given that some vegetables have the same color scheme as some types of meat and grains we wanted to apply some preprocessing to remove unwanted types of food.

4.1 K-Means clustering

We tried using the k-means image segmentation algorithm, a clustering algorithm to separate our image into two sections: vegetables and non-vegetables, to make classification easier.

Figure 3 Graph showing off k-means functionality, black dots = clusters, colored dots = cases (Yobero, 2018)

This algorithm partitions elements into a specified number of clusters k. The simplest version of this algorithm operates in two steps: the assignment step and the update step:

The assignment step assigns each pixel to the closest cluster (this being the one with the nearest mean) and for every new pixel added to the cluster it updates the mean value and all the pixels in the cluster (update step). The result is an image where similar data points is assigned to a cluster.

$$\sum^{k}\left(x_i^j - c_j\right)^2 \; k = number\ of\ cluster, x = data\ point, c = cluster$$

For each of the clusters the algorithm k, the function computes the distance between a cluster c and the data point in question x and moves the data point accordingly. It iterates through each "case" before moving the centroids and repeating the process.

This would have been a valid approach to our pre-processing however, we wanted to create one script working for multiple images. This would be hard with k-means given that it requires you to specify the number of clusters you want, which would vary for different plates of food with different colors. Because of this we decided to scrap our idea of segmenting the image before doing color classification.

4.2 Image Thresholding

Having decided on basing the project around color classification, we started looking for different segmentation/classification methods centered around colors. Selecting a specific range of colored pixels and excluding all other pixels in the image was the approach we chose to go for, this would allow us to target vegetables with specific color schemes and exclude unwanted food-types. When looking for image segmentation methods for color classification in our tool of choice: MATLAB, we found MATLAB's color thresholder app from the *Image Processing Toolbox*.

This app uses color thresholding: Thresholding is a spatial domain process, a transformation operator is applied to an image f(x, y) resulting in a transformed image g(x, y) (Rafael C. Gonzalez, 2008)

$g(x,y) = T[f(x,y)]\ g = resulted\ image,$

$T = transformation, f = original\ image$ (Rafael C. Gonzalez, 2008)

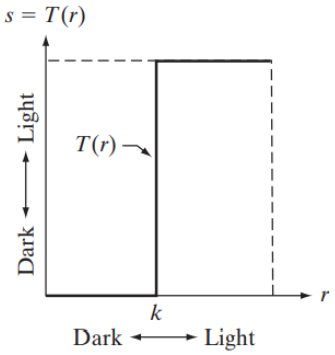

Figure 4 Basic thresholding T = color value (Rafael C. Gonzalez, 2008)

As seen in Figure 4, you specify a pixel value, if a given pixel is less or more than the specified threshold (pixel value) it results in a miss and reversely a pixel hitting above the threshold will result in a hit. The result in our case is a binary image where the hits are white, and the misses are black.

The Color Thresholder app (Figure 3) is an application which lets a user select regions of pixels in a color space from a specified image (*Figure 3*). You can then export this into a generic function which selects pixels which satisfy the specified regions of the color space using a mean RGB value. The result of this exported function is a binary image only containing pixels in the specified section of the color space.

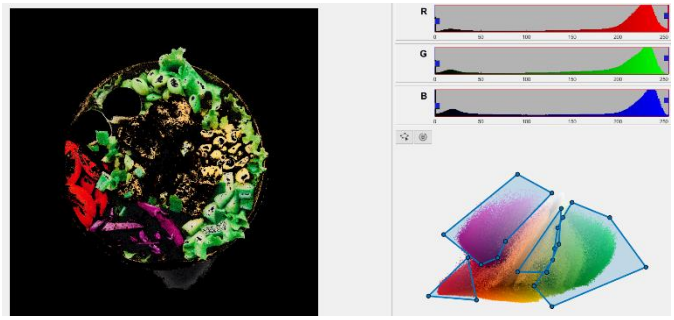

Figure 5 Color Thresholder App Graphical User Interface.

By using color thresholding, we are in most cases able to easily distinguish between the plate of food and vegetables (for exceptions and problems see test results and discussion). The *Color Thresholder app* also lets a user choose between different color spaces however for this project we only tested for RGB (given more time we would have compared RGB to HSV). The app lets a user select an image, choose a color space, select a section of the image's color space and export this into a function (Figure 6).

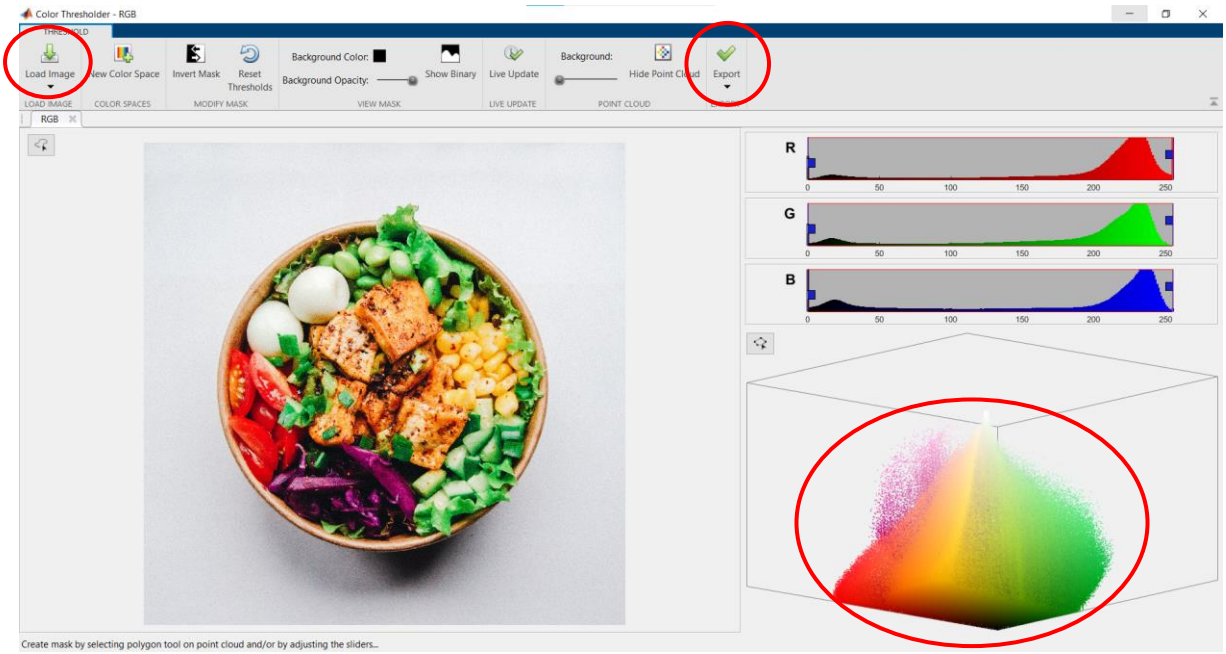

Figure 6 Color Thresholder app: 1. Load image option, 2 export option, 3 Color space

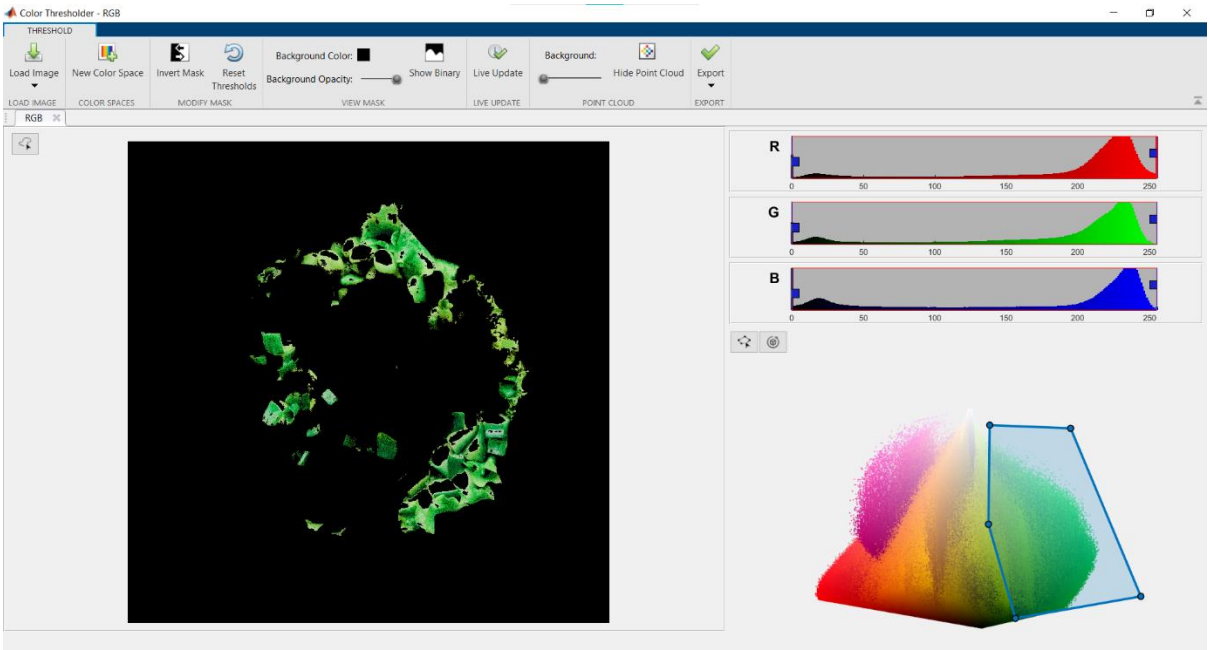

Figure 7 Selecting parts of the color space with the Color Thresholder app.

Exporting a function from the app will create a function which will create a binary mask based on the selected parts of the color space (Figure 7). The app gave us a nice preview of which pixels we selected which made selecting specific colors easy.

After selecting the base method for color classification, we had to decide on how to approach the segmentation: We could export one function which classified all five colors. This would save time and make the system less complex, at the expense of not being able to fine tune specific colors without re-creating the entire function. In addition, we had to find one image containing all our desired colors. Creating a detailed result using this approach would take a lot of testing and to use a very specific image.

The other approach was to create one function per color. This would allow us to fine tune specific colors providing more modularity thereby making the testing process easier. We ended up choosing this approach since it would make modification and testing easier in addition it would result in more specific color classification. Se figure 8

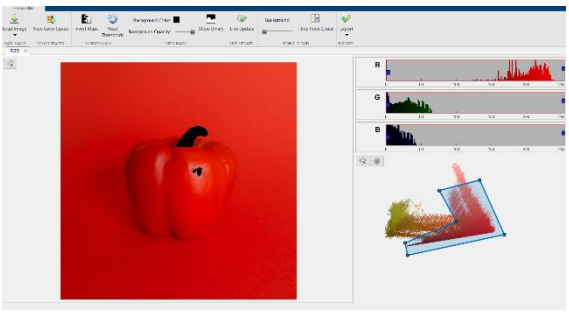

Figure 8 Selecting specific pixel regions using the Color Thresholder app

4.3 Fusing images

Having classified the colors, we had to merge our results to represent the result in a good way. Having five result images, one for each color is undesirable. We decided to fuse the images together, creating one image with all the colors classified by the five images.

Figure 9 Fusing multiple binary images together.

The function *imfuse*() from MATLAB's *Image Processing Toolbox* creates a composite image of two images where you can specify a fuse method, either blend, which overlays the images with alpha blending, or diff which creates an image containing difference between the input images.

For our script we fused all the different colors with diff except green, here we blended light green (avocado and cucumber core) and a more neutral green in order to get the most accurate representation of green vegetables in the image (Figure 9).

The results our approach produced were acceptable, however a lot of distortion and noise in the images made it hard to make out specific sections in the image. Because of this we had to "clean up" the image using some form of post processing (remove noise).

4.4 Morphological opening

This technique iterates through an image, removing all pixels without any neighboring pixels effectively creating an image with less noise. To implement this, you first define a kernel using strel(), you provide this function with a shape and a size in pixels. You then use this morphological structuring element to iterate through the image and remove pixels.

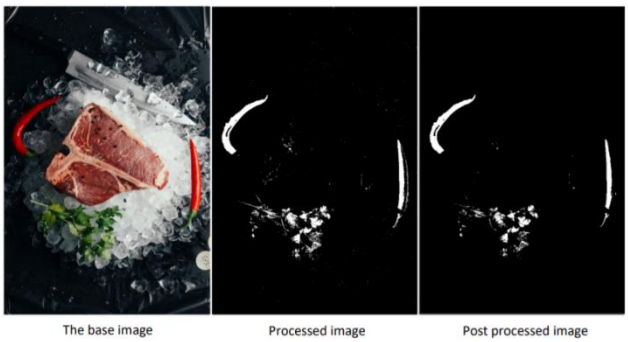

Figure 10 The resulting image before and after morphological erosion

$$A \ominus B = \{z|(B)_z \cap A^c = \emptyset\}$$

$$For\ each\ data\ point\ in\ z\ satisfying$$

$$B\ and\ A^c\ gets\ put\ in\ \emptyset\ (the\ resulting\ image)$$

$$A = image\ B = construct\ element\ and$$

$$\emptyset = result\ image$$

By performing morphological opening, we remove small objects from an image while preserving the shape and size of larger objects in the image (Figure 11). In our case, this means that we make a clearer object of the different vegetables. For instance, is the “water” and seeds inside a tomato, still a part of the tomato, even though it does not have the same red color as the outside of the tomato. Or parts of a vegetable that have a lot of reflection, still counts as part of the vegetable. We decided to use morphological opening since this method erodes white pixels in image, making it perfect for our use case.

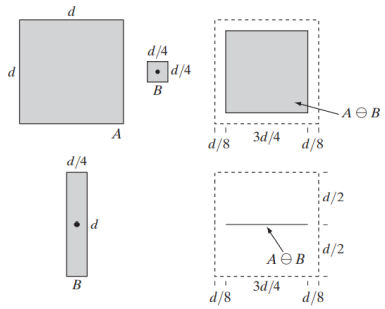

Figure 11 Visualization of formula Ø (Rafael C. Gonzalez, 2008)

5 Implementation

We started development by trying to implement image clustering in order to classify different regions of the image, we tried clustering and edge detection in the form of *k-means clustering*, *Sobel edge detection* and *Canny edge detection*. However, we quickly realized that using a basic edge detection or clustering algorithm would not work for our case; we realized that the food in a lot of cases is mixed unevenly on the plate (Figure 10) in addition to having a lot of different colors. Using a basic clustering or edge detection to classify different sections of a plate or bowl like Figure 12, the algorithms we tested did not give us any usable results given the variance in color in the vegetables (Figure 13)

Figure 12 Unevenly distributed food, hard to classify.

Having scrapped edge detection and clustering we started working on color classification, we started development by creating single color functions:

We selected our color-space (in our case RGB given that this is the color-space we had the most experience in, more on this in discussion), selected appropriate parts of the color space, we tried to select images of single vegetables with vibrant colors to capture as much of their color spectrum as possible when creating these functions. We then exported the function and tested it thoroughly with several images containing vegetables with this specified color scheme, if we got poor results, we would modify the selected regions of color before testing again.

Figure 13 Figure 10 after k-means clustering.

After creating all the color functions, we fused them together using *imfuse()* (from MATLAB's *Image Processing Toolbox*) and applied morphological opening in order to remove noise from fusing and classification. Script steps:

1. Read the image
2. Run the image through each color function:
    a. Red (tomato, bell pepper)
    b. Green (lettuce and cucumber)
    c. Light Green (more greens)
    d. Orange (carrots)
    e. Yellow (corn and yellow bell peppers)

    f. Purple (onion)
3. Fuse the images
    a. Fuse green and light green with blend
    b. Fuse all the colors with blend
4. Run the resulting image through a morphological opening algorithm

The resulting image will be a binary image of the plate with the vegetables highlighted. When testing we tried a wide variety of images, easy images with clear regions and hard images where all the food is mixed. We tested a wide variety of images to get a good overview of what our script is capable of.

6 Results and discussion

For testing we divided images into three groups: Easy, Normal and Hard. We tested ten images per category and discussed the results of each category separately. We have chosen to include some test results in the report to illustrate points, however, see the *Test Result Computer Vision Project* pdf document (found in the git repository: https://git.gvk.idi.ntnu.no/MartinIversen/cvprojectfoodclassification) for all the test results.

6.1 Results Easy images

For the easy images, we tried to pick out images which had little to no reflection and had simple structure. The images we tested showed us that our scripts struggle with darker colors, vegetables like spinach, broccoli and other dark vegetables were ignored by our green color function. This was an expected outcome given that we focused on the greener vegetables like lettuce when creating our green function's.

If the image we use has a large resolution, a lot of colorful vegetables and minimal reflection. Our script gives a very satisfying result (see testing image 10 and 9 from test document).

Figure 14 Yellow sections and dark green sections are hard to classify.

Another problem we encountered when testing these images was classification of the color yellow: We created our yellow function with corn and yellow bell peppers in mind, resulting in a binary mask which has a hard time differentiating between egg yolk, yellow gravy and the vegetables we based the function on.

In terms of post processing, we used a size value between four and eight, depending on the image and the classification result. We found it hard to pick out one size value which resulted in desirable post processing results for every image we tested however, five is the value we ended up choosing for the script it removes most of the noise in the image while still leaving an acceptable number of pixels.

Lastly our script is largely dependent on images with a high resolution: Since our script consists of functions which compare RGB values of pixels to create binary masks, the accuracy of our script is heavily correlated with the number of pixels in the image. Because of this, using images where the picture is taken far away from the food or if the image has a poor resolution results in a bad result since there are fewer pixels in the image.

Figure 15 Low resolution yields poor results

6.2 Results normal images

For the normal category of images, we tried to find images with a more complex structure, plates of food where there were fewer clear sections (see Figure 18). We also tried to find some images with a lot of different colored vegetables and some reflection.

The main problem our script encountered when trying to classify these more complex images was post processing accurately: Unlike most of the easy images these pictures were more cluttered, making it hard for our morphology function to differentiate between the lone pixels we wanted and the ones we didn't want: Image 19 from the test document is a picture of a pizza, it contains some tomatoes and lettuce, the initial result of our script finds all the vegetables in addition to some orange/yellow dough, after post processing however the tomatoes are gone (see Figure 16-17).

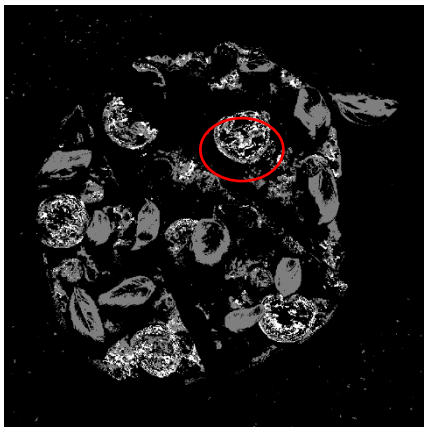

Figure 16 Image 19 from the test document before post processing

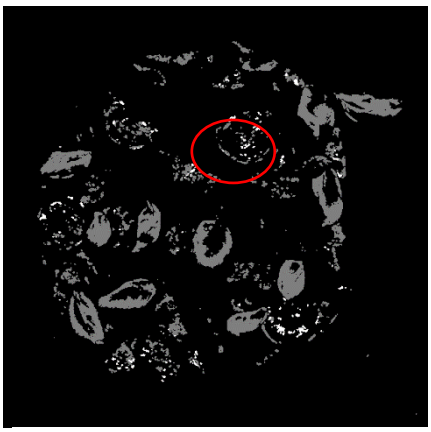

Figure 17 Image 19 after post processing

Testing images with the colors yellow and orange yielded results like the ones in the easy images section, since orange and yellow is a very popular color among meat, bread and other types of cooked food, our script struggled to differentiate between orange vegetables and orange foods in images containing these foods, making these colors problematic, although some images post-processing removed some of the noise this problem created.

Figure 18 Image17 containing a lot of bright color and light reflection.

Reflection and bright colors were the scripts biggest problems throughout testing. To illustrate the problem, we will be looking at Figure 18. This image has a lot of different colors and reflection. Since the colors of this image are quite intense this image should not be any problem for our script however, the result ends up being quite poor.

The initial result has a lot of noise, especially around the chicken in the middle (illustrating our problem with orange colors see Figure 19). After post processing the orange colors are gone however, the trade of is that a lot of the lighter colors and reflecting parts gets removed, resulting in little to no classification.

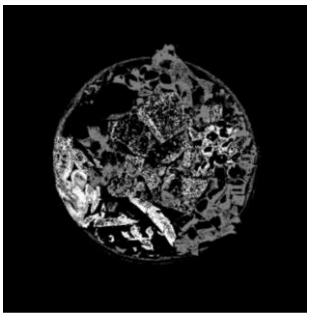

Figure 19 Image 17 before post processing

Figure 20 Image 17 result

One of the better results we got from the script was image 16 (see Figure 22). This image has a variety of colors, textures, and some reflection. As seen in the image below the script picks up almost all the vegetables save for some of the light green avocado, showing us that the script is heavily dependent on good lighting/illumination in order to produce good results.

Figure 22 Image 16 Original image

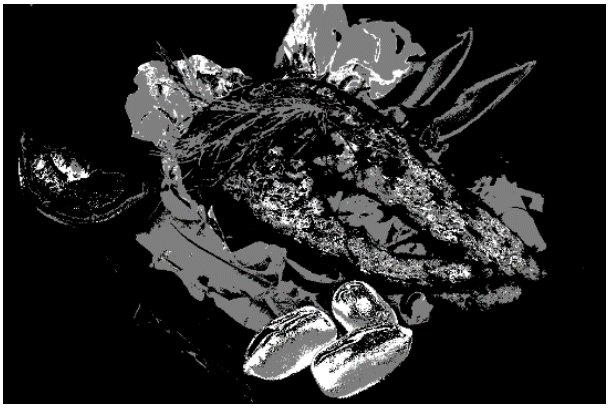

Figure 21 Image 16 pre post processing

Figure 23 Image 16 result

6.3 Results hard images

For these harder images we tried to find images which highlights the problems we experienced in the easy and normal images: Images containing large amount of reflection, cluttered plates, different variants of green, orange, and yellow.

The results we got coincided with the ones for easy and normal images, our script struggles with dark and light green: image 23 (see figure 26) contains several bowls with different types of food and vegetables, we can see from the post processed image that almost no green vegetables are picked up capturing our problem regarding the color green.

Figure 26 Image 23 Original

Figure 25 Image 23 after post processing

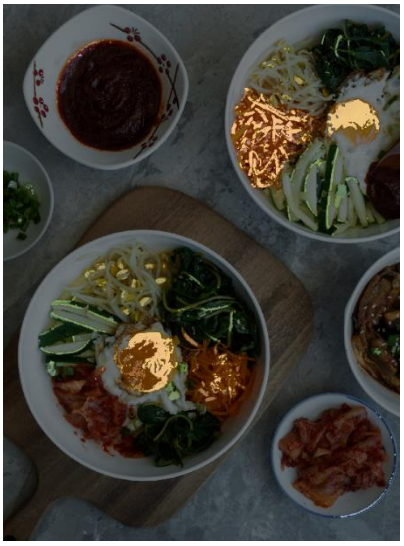

Figure 24 Image 23 Result

The scripts problem with orange is also prevalent in these images. Looking at image 25 (see figure 27), it contains a lot of green vegetables in addition to salmon. Looking at the result before post processing we can clearly see the scripts struggle with orange and lighter green colors(see Figure 28). For these types of images, the post processing had to use a big construct in order to remove the non-vegetable sections picked up by the script. The result looks acceptable (see Figure 30-31), some of the green is

Figure 28 Image 25 Pre Post Processing.

Figure 27 Image 25 Original

still present however through post processing a lot of the vegetables are lost.

Figure 30 Image 25 After Post Processing

Figure 29 Image 25 Result

We found that the result when classifying green colors does vary a lot depending on the amount of illumination on the plate. Looking at the results of image 26 and 29 (see Figure 29-31), both have a lot of green lettuce scattered around however, image 29 (Figure 31) has yielded a marginally better result than 25 (Figure 29). We suspect this is due to the difference in illumination, good lighting gives us more intense colors which are easier for our script to detect in addition to producing minimal reflection.

Figure 31 Image 29 Result, bright green colors makes it easy to classify

7 Discussion

Throughout testing we found that our current version of the script has three main flaws:

- Classification of bright and dark vegetables
- Classifying vegetables reflecting a lot of light
- Differentiating between vegetables and other food with the same color scheme

7.1 Bright and Dark vegetables

The current version of our script's biggest flaw is its inability to detect very bright colors and very dark colors. This is mainly due to the way we approached the project topic: We decided early on to base our solution around Color thresholding, limiting our classification complexity a lot.

Since this methodology is focused solely on color, differentiating between a dark background and a dark part of a vegetable (using only our tool) is impossible, since the RGB values of these regions of the vegetables are too similar. Because of this, for some vegetables like Broccoli, Eggplant, Avocado, Onion and some types of salad will go through the script unnoticed.

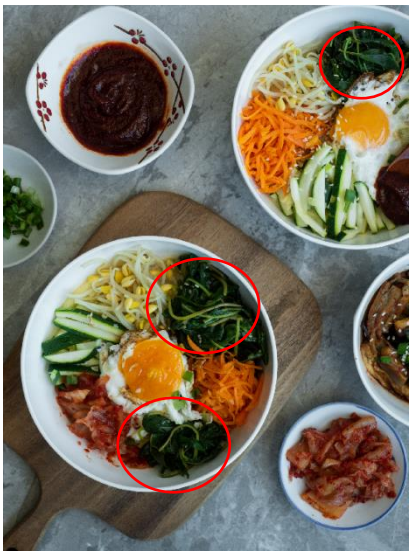

Figure 32 Image 23 Original, alot of dark green colors, making the image hard to classify

The same type of problem is apparent when trying to classify bright regions of vegetables like cucumber cores, avocado and the insides of tomatoes. These sections RGB values are too bright, creating a color mask function for these bright colors would result in white backgrounds, plates and cutlery getting classified due to light reflection. Fine-tuning our color mask functions would in theory be possible however doing this with our tool would be unintuitive and too time consuming for this project.

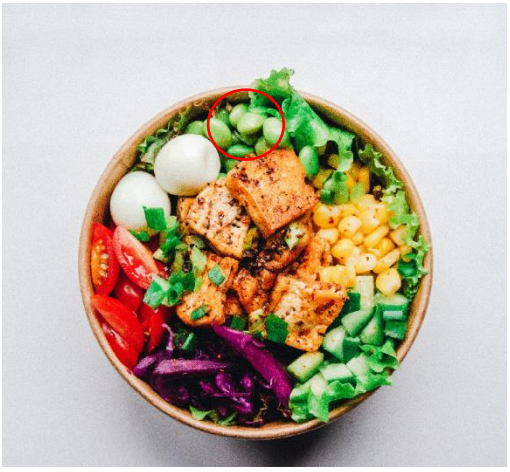

Figure 33 Image 17 Original: Reflective light makes the image hard to classify

Both issues are apparent throughout most of our test images, one of the approaches we could have taken to solve these problems is to implement an advanced border detection algorithm, by doing this we could've removed regions of the image which aren't relevant for our classification approach, this would let us implement darker and brighter color functions, resulting in a more

detailed result containing the problematic colored vegetables.

As for the border detection algorithm, one based around deep learning would give us the results we desire however, this technology lies outside our curriculum and would require a lot of time and computational power to implement properly.

7.2 Reflective light

The other big problem our current script has is its inability to deal with light reflection in the vegetables, our current solution is as stated above only classifies color, making it hard for the script to deal with reflective light.

Solutions we think could have solved this issue would either be an advanced morphological operation used to remove all the reflecting light in the image before classification, or an advanced deep learning algorithm, able to detect reflective light and remove it. Implementation of pattern analysis or homomorphic filtering to remove illumination in the image before classifying the image would also be a valid solution for this issue.

Figure 34 Image 15 Original, reflective light on tomatoes makes them hard to classify.

7.3 Vegetables and other food with the same color scheme

We encountered this problem throughout all our testing: When the script classifies the colors: orange and yellow it cannot differentiate between foods like carrots, tomatoes and meats like salmon and chicken, since these color values are too similar.

This also concerns corn and broth´s with yellow colors. Looking at the RGB values of our orange color function we have: 151, 71, 12 For our orange function (see Figure 35) this value is the mean of the sections we selected in the *Color Thresholder app* it lies between orange and brown. For the yellow color function we have the mean value of 157, 128, 55 (see Figure 36). Looking at these colors we observe that they are quite bland. This is because of how the *Color Thresholder* app creates a function when you select more than one point cloud (it will create a mean RGB value out of all

Figure 35 Mean RGB value for orange.m script

Figure 36 Mean RGB value for yellow.m script

the sections of the color-space you have selected). This results in the script being unable to differentiate the foods since the RGB values are similar.

To solve this issue, we could have implemented a border detection algorithm to separate the different sections of food, before running a texture analysis function to classify different types of meat and remove them thereby solving our problem another solution would be to use another color classification tool.

7.4 Post processing

For post processing we used morphological erosion to remove excessive noise and distortion from the images after classification. We created a morphological construct with a threshold for pixel removal.

We tested different sizes for this construct. Testing gave us an estimate that a value between four and eight would remove enough noise to produce good results.

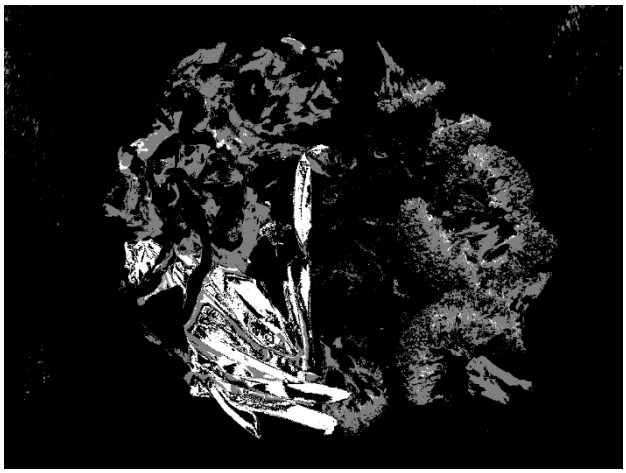

Figure 37 Image 30, low Morhological construct radius

Figure 39 Image 35 Medium Morphological construct radius

Figure 38 Image 30 High Morhological construct radius

For the script we settled on the value five (see Figure 38-39) however, the size varied for each individual image.

7.5 Color space

Given that the *Color Thresholder* lets the user select a color space we looked at each color space's strengths and weaknesses: RGB is the color space we chose to go for since this is the color space, we have the most experience with and because it is the color space mainly used when displaying images on a screen. HSV: If we had more experience with this color-space we would have chosen HSV since this color-space is widely used for image processing and robot-vision however, since our methodology is quite straightforward and our knowledge of the HSV color-space is limited, we decided not to use it. Given more time (or a second try at the project) we would have tested our script with this color-space and compared the results against the RGB color-space.

YCbCr: We knew nothing about the advantages of this color space and after some research we concluded that this color space did not have any advantages over RGB and HSV for our use-case. Regarding L*a*b since this color-space is centered around approximating human vision we concluded that it was not useful for our project.

7.6 Optimal pictures

To achieve optimal results the image should preferably have these qualities:

- High resolution
- No strong shadows
- Minimal reflective light
- Good illumination
- Intense and distinctive colors

If these conditions are met our script will achieve good results (see figure 23).

If we were to redo the project, we would have approached the topic differently: We would base our project around edge detection and texture analysis instead of basing it solely around color. Taking this approach, we could have split the images into different regions, classify them using texture before running them through a color classification algorithm in order to solve our current script's problems.

8 Conclusion

Looking at our current script with regards to the goal we set in our introduction: *We have chosen to try and segment one specific type of food on a plate and highlight it.* Our script classifies vegetables on a given plate and gives a resulting image with the vegetables highlighted. It struggles with images with poor illumination and a lot of reflection and has no way of differentiating between different types of food with the same color.

Given a picture which is well illuminated, has a high resolution and no food which has the same color as the vegetables. Our script does have the functionality we set out to implement in the start of the project. We found that creating a script that works for every dish/picture is hard using our approach because of our focus around color. We encountered a lot of unforeseen variables like lighting conditions for the images in addition to the importance of resolution when using thresholding.

If we were to improve the script, we would implement a machine learning algorithm like the R*andom Forest algorithm* in order to make our script more versatile and to remove undesired food types before doing the color classification. We would also implement more post processing in order to remove unwanted textures, reflection and shadows in the resulting image.

Our testing approach worked well, testing with a wide variety of images helped us realize the limitations regarding our use of image segmentation. Although we did not have time to address a lot of the problems our testing process highlighted it made us reflect and discuss them.

If we were to redo the testing process we would have done it iteratively in order to try and address the problems and limitations our script had. In addition we would have tested the images with the HSV color-space in addition to the RGB color-space to compare the two color-spaces and their results.

Basing the project around color thresholding did require a lot of fine tuning and testing in order to achieve reasonable results. Implementing texture analysis, machine learning and edge detection would most likely have been a better solution to our project case, since we would have been able to remove unwanted food types before classifying colors. However, we believe our approach yielded good results and if more time were spent on pre-, and post-processing we believe our script would have been a good foundation for developing a food classification app.

9 Source List

All testing images are taken from Unsplash and are public domain stock photos. (n.d.). *Unsplash* . Retrieved from unsplash.com: https://unsplash.com/

Beltrone, G. (2017, Januar 19). *AdWeek*. Retrieved from Adweek: https://www.adweek.com/creativity/mcdonalds-poses-existential-question-big-mac-bacon-still-big-mac-175654/

Brief, R. (2018, October 31). *Research Brief* . Retrieved from Research Brief : https://www.cbinsights.com/research/startups-drive-auto-industry-disruption/

Briefs, R. (2018, October 31). *CB insights.* Retrieved from cbinsights.com: https://www.cbinsights.com/research/startups-drive-auto-industry-disruption/

Donges, N. (2020, September 3). *builtin*. Retrieved from www.builtin.com: https://builtin.com/data-science/random-forest-algorithm

He, Y. X. (2013). *FOOD IMAGE ANALYSIS: SEGMENTATION IDENTIFICATION AND WEIGHT ESTIMATION.* Retrieved from ncbi: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5448794/

Inunganbi, S., Seal, A., & Khanna, P. (2018, February 14). *Springer Link.* Retrieved from www.link.springer.com: https://link.springer.com/chapter/10.1007/978-3-319-75420-8_49#aboutcontent

Matlab accosiation. (n.d.). *Mathworks.* Retrieved from mathworks.com: https://www.mathworks.com/help/images/morphological-dilation-and-erosion.html

MATLAB team. (Unknown, Unknown Unknown). *Mathworks helpcenter*. Retrieved from Mathworks: https://se.mathworks.com/help/images/ref/colorthresholder-app.html

Matlab/Mathworks association . (n.d.). *Mathworks.* Retrieved from mathworks.com: https://se.mathworks.com/discovery/image-segmentation.html

Mehta, A. (2014, February 26). *BrainFacts.org.* Retrieved from brainfacts.org: https://www.brainfacts.org/in-the-lab/tools-and-techniques/2014/brain-scans-technologies-that-peer-inside-your-head

Rafael C. Gonzalez, R. E. (2008). *Digital Image Processing Third Edition.* New Jersey: Pearson.

Silva, L. P. (2012). *Food Classification using Color Imaging.* Retrieved from citeseerx: https://citeseerx.ist.psu.edu/viewdoc/downloaddoi=10.1.1.217.6709&rep=rep1&type=pdf&fbclid=IwAR0SX-Y_hTDK0dqF8iOGhKxK4hyl6aVhrQJ7jR4fuiTMyPmtxrfikhA8dEQ

Yobero, C. (2018, February 1). *RPubs*. Retrieved from rpubs.com: https://rpubs.com/cyobero/k-means

10 Figure List