# Food classification vegetables

PROJECT TOPIC 4

Food classification

Group 2

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

# Table of content



Notes to TA:

Is the concept section to vague?, more math or more in depth discussion?

Is the method and implementation implemented correctly

Do we need to provide individual sources for all 30 test images? Or is unsplash.com ok?

**TODO:**

- Finish abstract will be written after the report is finished
- Write conclusion will be written after feedback

## Abstract

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 the project part of the course, we will learn to work in group with other students and receive knowledge on various areas of learning. That we will apply creatively to real-life situations, where the purpose of the project work in this class is to be able to classify a specific part of an image.

For this project, we have chosen to analyze an image of a dinner plate with the help of MATLAB threshold color app and find out if the plate has vegetables. We have chosen to categorize the vegetables in different colors, that we believe are typical accurate for vegetables, and thus classify if the dinner plate contains vegetables.

The tool we have used in the project is the MATLAB threshold color image. The app lets you segment color images by thresholding the color channels based on different color spaces. So, by using this app in our project, we can create binary segmentation mask for a color image.

## Introduction

In today's society diet is important, people nowadays are more conscious about what they're 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. 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, and texture. This technology falls under a broader concept called Image segmentation. Image segmentation is a technology within computer vision used to segment an image into different regions.

We have chosen to only focus on colors that we believe are most relevant for categorizing vegetables given our limited knowledge and experience. Thus, the goal is to implement functions that will classify green, red, purple, yellow and orange colors in a plate of food. To solve this, we will be using the Image Processing Toolbox from MATLAB.

We chose this project because it gave us an opportunity to go through numerous image segmentation techniques to learn more about these different techniques and image segmentation in general. We also wanted to learn about the basics behind image segmentation technology surrounding us in everyday life like face recognition, CGI and autonomous cars.

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.

## 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. This can make analyzing, post processing and morphing the image easier.

Over the years segmentation algorithms and technics 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.

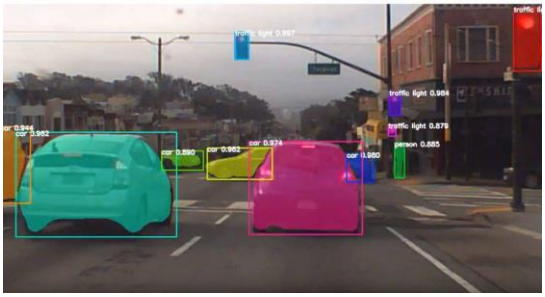


*Figure 1 Autonomous driving system (Brief, 2018)*

Image segmentation of food is widely used in food industry. The main use of image segmentation in the food industry is the use of postprocessing in certain sections of a dish in order to make the dish more appetizing. Another example of food segmentation in the food industry is using the segmentation and classification techniques for the verification of the quality of the cooking process of the food.


*Figure 2Commercial for Big Mac, is this accurate? (Beltrone, 2017)*

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.

## Method

We started 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 and colors within images.

The goal with image segmentation is to make an image easier to analyze, post process or discuss. There are different ways to go about this; we tried using the k-means image segmentation algorithm, a clustering algorithm which partitions objects into clusters creating sections in an image with similar pixels.

This would be a valid approach to our problem however, we are creating one script working for multiple images which would be hard given that using k-means requires that you specify the number of clusters you want which would vary for different plates. After looking for other image segmentation techniques we decided on the (MATLAB team, Unknown) to classify different colors on a plate.

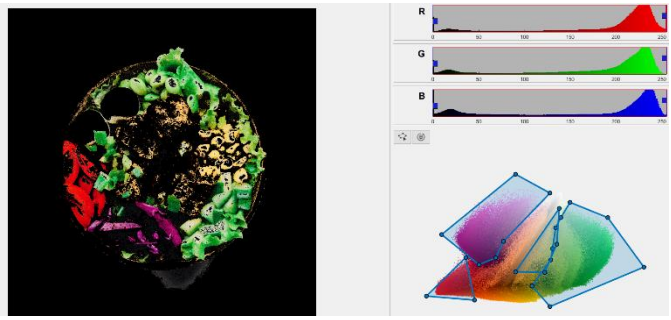



*Figure 3The Image Thresholder app*

The Color Thresholder app is an application used to select regions of color from an image. You can then export this into a generic function. This function selects pixels which satisfy the specified color regions. "Hits" will become white, and "Misses" will become black. We figured that this method was the best for our use-case after the use of Machine Learning. By using the Color Thresholder, we are in most cases able to easily distinguish between for instance meat and vegetables (for the exceptions and problems see test results), or also the different types of vegetables if desired.

The app gave us a nice preview allowing us to get a great understanding which color values we selected, because of this we decided to use the Color Thresholder app from the Image Processing toolbox as our base. The exported functions converted

our image into a binary image (from RGB), instead of giving us the RGB values, this way, we can classify specific parts of the image easily.

After selecting the base methodology for our script, we had to decide on how to approach the project:
We could either create one function which classified all 5 colors (*see figure 2*). This would save time and make the system less complex in addition we wouldn't be able to fine tune specific colors, however, creating a detailed result using this approach would take a lot of testing and the use of one image containing all of our colors all under specific conditions.

The other approach was to create one function per color before fusing the five images to create one resulting image. This would allow us to fine tune specific colors providing more modularity making the testing process easier. We ended up choosing this approach since it would make modification and testing easier. This approach did make our code more complex (having to fuse images). The process continued by finding test images with different difficulty levels. The results our approach produced were acceptable however a lot of distortion and noise in the imaged made it hard to make out specific sections in the image.

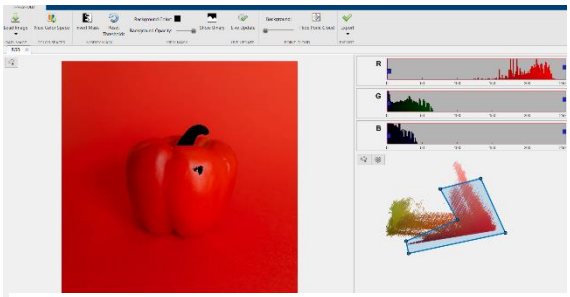
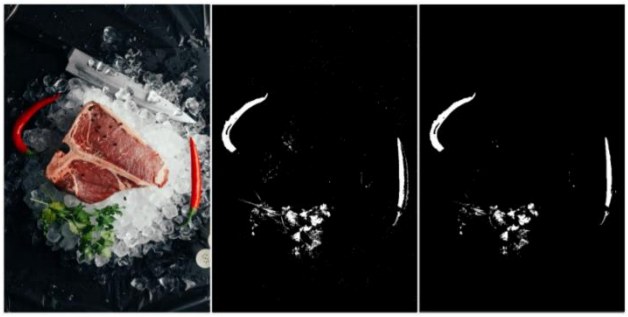


*Figure 4 Using the Color Thresholder app to select one color*

Because of this we had to "clean up" the image using some form of post processing. We decided on morphological opening since this method erodes the image removing lone pixels and general distortion. The images we tested are classified as either easy, medium, hard, or problematic. All the images were run through the functions we made and from there we performed post image processing in the form of morphological opening.



By performing morphological opening, we remove small objects from an image while preserving the shape and size of larger objects in the image (Morphological Operations). In our case, this means that we make a clearer object of the different vegetables and ideally are able to fight against reflection etc. a little bit better. 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.

# Technologies

This section will be dedicated to introducing the different concepts we're using in our implementation, note that there will be brief introductions, no in-depth discussion of the concepts.

## K-Means clustering

An image segmentation algorithm which partitions elements into a specified number of clusters. The simplest version of this algorithm operates in two steps, the assignment step and the update step.

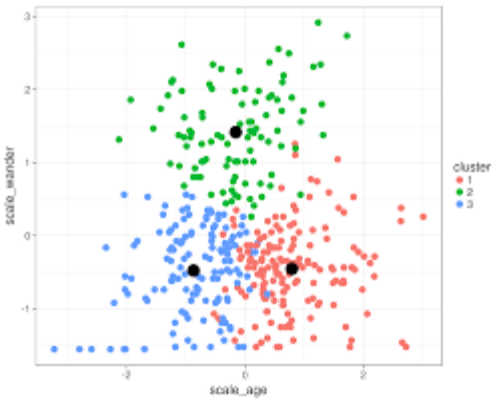
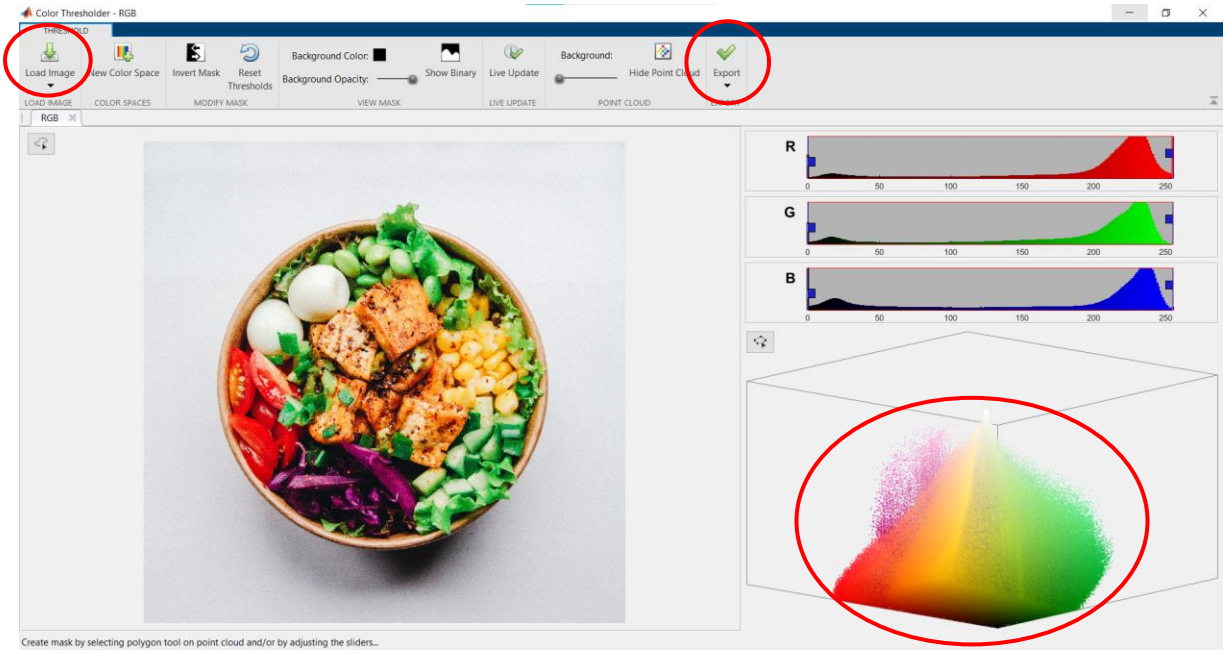



*Figure 5 Graph showing off k-means functionality (Yobero, 2018)*

The assignment step assigns each pixel to a 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. The end result is an image where each cluster is highlighted.

## Image Thresholder app

This app lets a user select an image, choose a color space, select a section of the color space for a specific image and export this into a function.



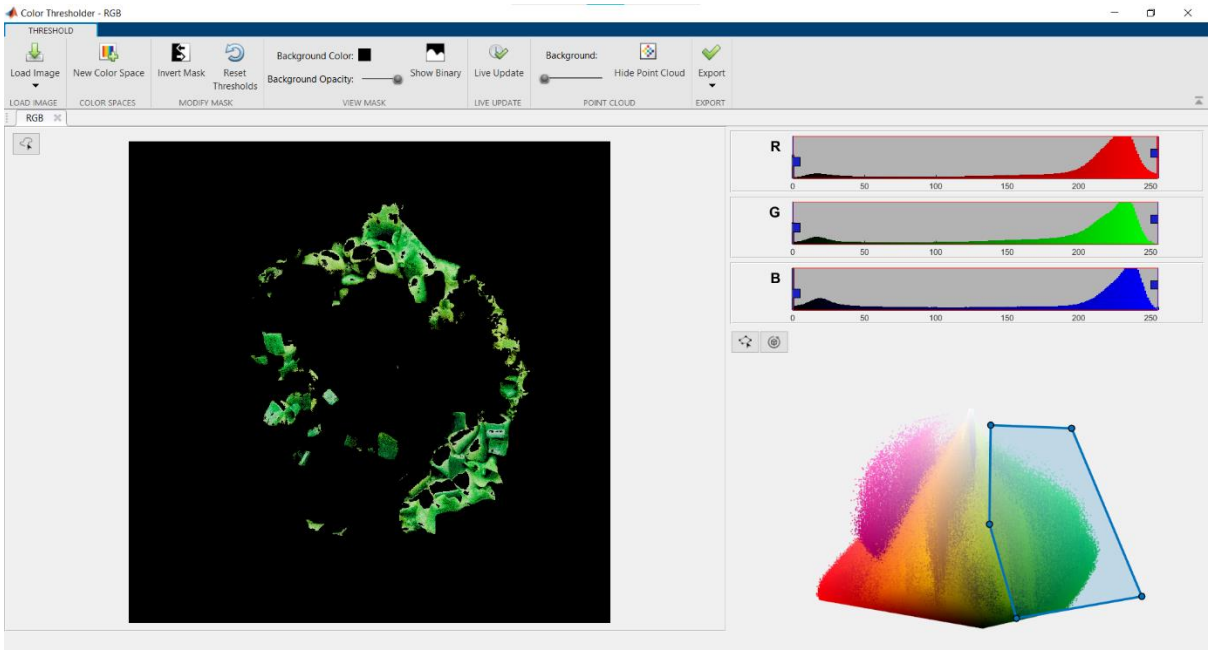



*Figure 6 Selected the green section of the image above*

Exporting a function from the app will create a function which will create a binary mask based on the selected colors. Sending in a RGB image to the function will filter the passed in RGB image and the binary mask created with the app.

*Figure 7 Binary mask applied to an image*

## Fusing images

Using the function imfuse this function creates a composite image of two images, in our case we specified a method either blend which overlays the images with alpha blending or diff which creates an image with the difference between the images. For our script we fused all the different colors with diff and blended both greens in order to get the most accurate representation of the image.

## 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.

## Implementation

We started development by trying image clustering to classify different regions of the image. However, we quickly realized that using a basic image segmentation algorithm would not work for our case; finding test images, we realized that the food in a lot of cases is mixed unevenly on the plate (see figure 1). Using a basic clustering algorithm to classify different sections of a plate or bowl like figure 1 the algorithms we tested did not give us any usable clusters (see figure 4).

*Figure 1 A mix of different vegetables with no clear borders*

Implementing the Image Thresholder app was straightforward:

*Figure 8 Image after applying kmeans image clustering with one cluster for each desired color(5)*

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 an image of the plate with the vegetables we classified highlighted. When testing we tried finding images where there is a clear distinction between vegetables and the rest of the dish. We tested a wide variety of images to get a good overview of what our script is capable of.

# 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) for all the tests.

## 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 that our scripts struggle with darker colors. Vegetables like spinach, broccoli and other dark vegetables were ignored by our scripts. This was an expected outcome given that we focused on the greener vegetables like lettuce when creating our green binary mask.

Given that 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).

*Image 2 Results, Dark green and yellow*

Another problem we encountered when testing these images was classification of the color yellow: We created our yellow binary mask 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 mask 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 since it removes most of the noise in the image while still leaving the main parts of the mask.

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 results in a poor classification result since there are fewer pixels in the food section of the image.

*Image 6 Result, low resolution yields poor results*

## Results normal images

For the normal category of images, we tried to find images with more complex structure, plates of food where there were less clear sections (see image 19). 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 is a picture of a pizza, it contains some tomatoes and some lettuce, the initial result of our script found all the vegetables in addition to some orange/yellow dough, after post processing however the tomatoes are gone.

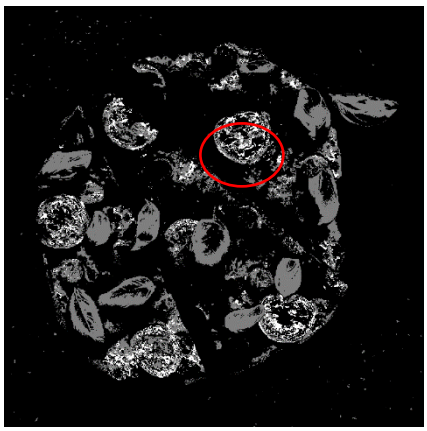


*Image 19 before post processing*

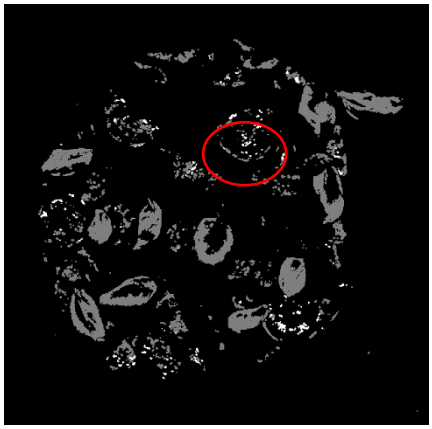


*Image 19 after post processing*

Testing images with the colors yellow and orange yielded results like the ones in the easy images section, since orange is a very popular color among meat, bread and other types of cooked food, our script cannot differentiate between orange vegetables and orange foods making this color problematic, although in some images post processing solved this problem.

Reflection and bright colors were the scripts biggest problems throughout testing. To illustrate the problem, we will be looking at image 17. This image has a lot of different colors and clear sections. Since the colors of this image are quite intense so this image should not be any problem for our script however, the result ends up being lackluster.

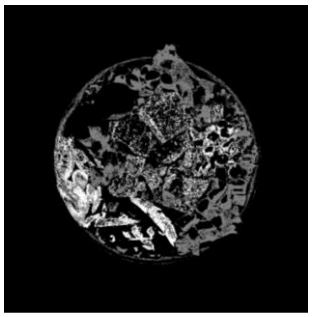


*Image 17 before post processing*

*Image 17 result*

The initial result has a lot of noise, especially around the chicken in the middle (illustrating our problem with orange colors). 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.

One of the best results we got from the script was image 16. 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.

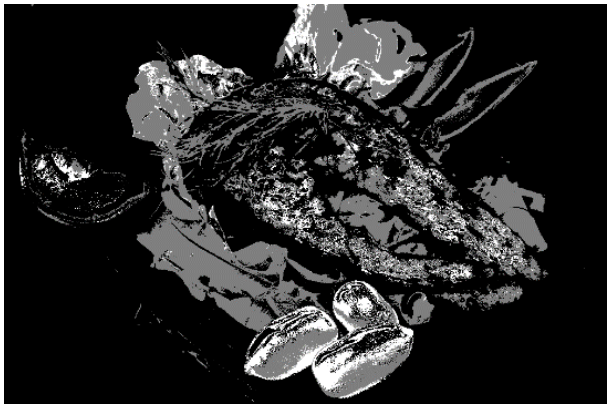


*Image 19 Result*

## 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 contains bowls with different types of food and vegetables, we can see from the post processed image that almost no green vegetables are picked up. This illustrates our light and dark green problems well.

*Image 23*

*Image 23 after post processing*

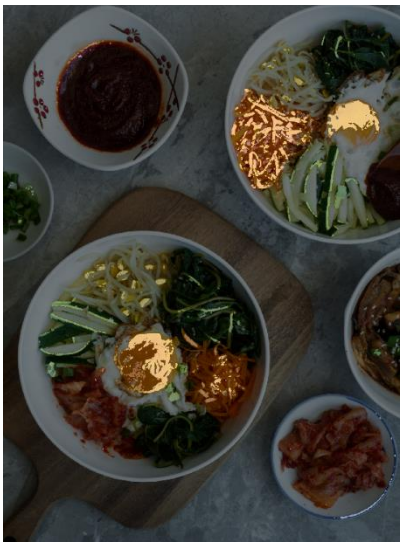


*Image 23 Result*

The scripts problem with orange is also prevalent here. Looking at image 25, it contains a lot of green vegetables in addition to orange salmon. And looking at the result before post processing we can clearly see the scripts struggle with orange and lighter green colors.

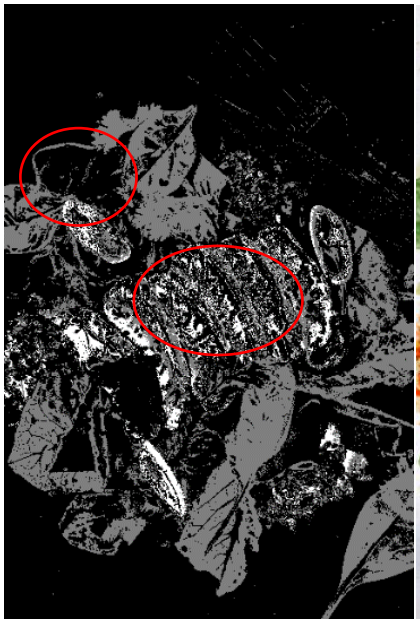


*Image 25 before post processing*

*Image 25*

For these types of images the post processing had to be rough in order to remove the non vegetable sections picked up by the script.The end result looks okay, some of the green is still present however through post processing a lot of the vegetables are lost.

*Image 25 after post processing*

*Image 26 result*

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 twentysix and twentynine, both have a lot of green lettuce scattered around however, image twentynine has yielded a marginaly better result than twentyfive.

We suspect this is due to the difference in illumination, more light gives us more intense colors which are easier for our script to detect.

*Image 26 result*

*Image 29 result*

# Discussion

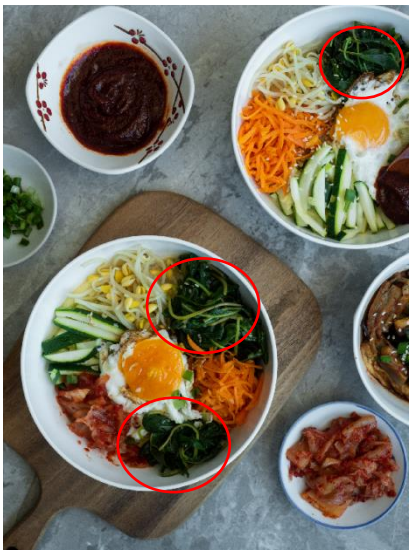
Throughout testing we found that our current version of the script has three main flaws:

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

## Bright and Dark vegetables

The current version of our script's biggest flaw/shortcoming 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 the *Color Thresholder* app from the *Image Processing Toolbox* in MATLAB's library.

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



*Darker vegetables won't get classified*

The same type of problem is apparent when trying to classify bright regions of vegetables cucumber, 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 backgrounds, plates and cutlery getting classified. Fine tuning our color mask functions would in theory be possible however doing this with our tool would be unintuitive and too time consuming.

*Classifying brighter regions would yield poor results*

Both of these issues are apparent throughout most of our test images, one of the approaches we could've taken to solve these problems is to

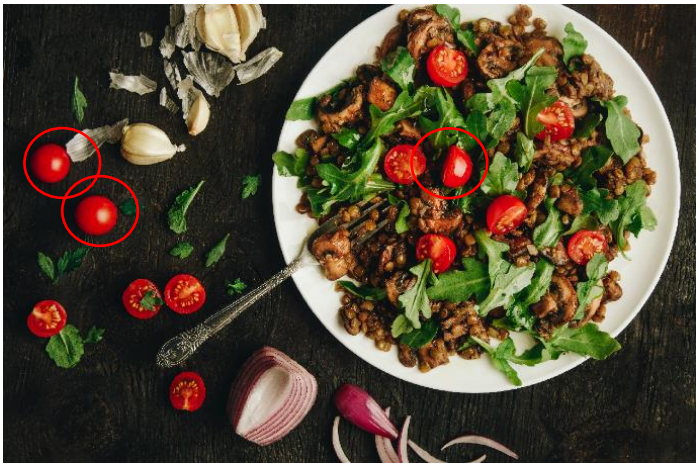
implement an advanced border detection algorithm, by doing this we could've removed regions of the image which aren't relevant for our topic before the classification process this would let us implement darker and brighter color functions, resulting in a more detailed result.

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.

## 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 sensitive to color, making it hard for the script to differentiate between reflecting light and white/bright colors.

Solutions we think could've solved this issue would either be some 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.



*FTomatoes with alot of reflecting light*

## 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 color's. Looking at the RGB values of our orange color function we have: 151, 71, 12

*Mean value Orange.m*

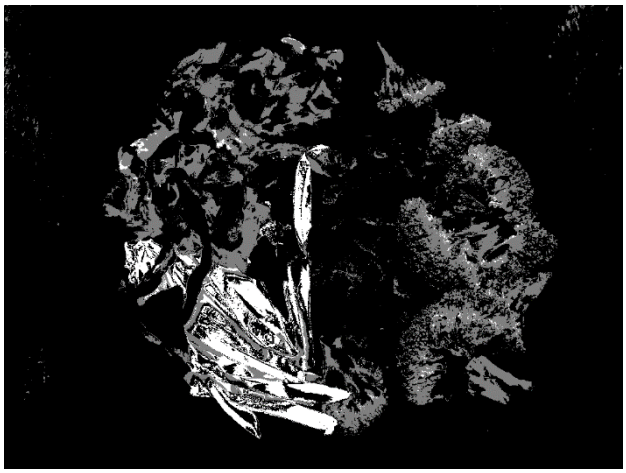
For *orange.m,* this value is the mean of the sections we selected in the *Color Thresholder app* it lies between orange and brown. For the function *Yellow.m* we have the mean value of 157, 128, 55. Looking at these colors we observe that they're quite vague. This is because of how the *Color Thresholder* app exports a function when you select more than one point cloud (it will create a mean RGB value out of all the clouds you've selected). This results in the script being unable to differentiate the foods since the RGB values are similar.

*Mean value Yellow.m*

To solve this issue, we could've 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.

## Post processing

For post processing we used basic morphological opening to remove excessive noise and distortion from the images after classifying colors. We created a morphological construct which we used to select a threshold for pixel removal.



*Result before post processing*

We tested different sizes for this construct. Testing gave us an estimate that a value between four and eight would remove enough noise, for the script we settled on the value 5 however, the size varied for each individual image.

*Construct size 5*

*Construct size 10*

## Optimal pictures

There are other minor flaws in our script, to achieve optimal results the image should preferably have these qualities:

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

If these conditions are met our script will achieve good results.

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 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.

## Conclusion

In conclusion:

- Creating a script which works **well** for different plates of food I hard using our methodology
- Provided with a specific set of images, using the image thresholder app works
- Classifying every part of a vegetable regardless of reflection and bright colours takes a lot more image processing using different types of algorithms.
- Without any image segmentation differentiating between foods with the same colour scheme as vegetables (salmon, chicken and rice) and vegetables with those colour schemas is hard.

Write a proper conclusion using these points and more

# Bibliografi

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

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

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

Matlab accosiation. (u.d.). *Mathworks.* Hentet fra mathworks.com: https://www.mathworks.com/help/images/morphological-dilation-and-erosion.html

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

Matlab/Mathworks association . (u.d.). *Mathworks.* Hentet fra mathworks.com: https://se.mathworks.com/discovery/image-segmentation.html

Silva, L. P. (2012). *Food Classification using Color Imaging.* Hentet fra 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*. Hentet fra rpubs.com: https://rpubs.com/cyobero/k-means