

# Banana Ripeness Detector

April 10th 2019

CIS\*4720 Final Project

Giavinh Lam,  
Greg Hetherington,  
Petar Kenic

*School of Computer Science at the University of Guelph*

## Abstract

The purpose of this project was to develop an effective solution for detecting the ripeness of a banana in a given photo. After research on different potential algorithms and taking into account the scope and constraints of the project (e.g. time restrictions), a template matching algorithm was chosen to use as the base algorithm. Further features such as background subtraction, multi-scaling, and rotational invariance were used to enhance the base algorithm and were notable achievements. Overall, the results proved to be fairly successful with the majority of the test cases although there were occasional false positives and some results where the area detected could have been improved (e.g. algorithm bounding part of the detected banana instead of all of it).

# Introduction

Determining the ripeness of a banana seems like an easy task for anybody, but to visually impaired people it is nearly impossible. In this report, we will be discussing our proposed solution, a web application that can take in a picture of a banana and output a resulting image highlighting the ripeness of the banana. Some of the challenges are determining the region of the banana and the colour analysis.

In the paper “Image Processing of an Agriculture Produce: Determination of Size and Ripeness of a Banana”<sup>1</sup>, it discussed many types of edge detection including gradient-based, derivative-based and Canny’s method. Edge detection is an image processing technique used to find the boundaries of an object in a image. The Canny’s method one of the better types of edge detection as it has very good detection and has a short response. This is the edge detection method we ended up implementing for our solution.

In the paper “Application of Image Analysis for Classification of Ripening Bananas”<sup>2</sup>, it discusses how RGB is not an ideal color scheme for detecting colours and other colour schemes should be used. These include HSV (Hue Saturation Value) or CIE L\*a\*b\* as they both take into account the luminance/lightness component of the pixel. As well, these systems are great at matching, due to them replicating human perception. Using a better colour space, the algorithm will compare the pixel to a set of thresholds to properly determine if the banana is unripe, ripe or overripe.

## Algorithm Discussion

### Brief Explanation

The algorithm chosen to implement this solution was template matching. Template matching is a method for searching and finding the location of a given template image in a larger image. The algorithm works by overlaying and sliding the template image over the larger image multiple times until complete. While sliding the template image, the algorithm compares the two images pixel by pixel to judge similarity.

---

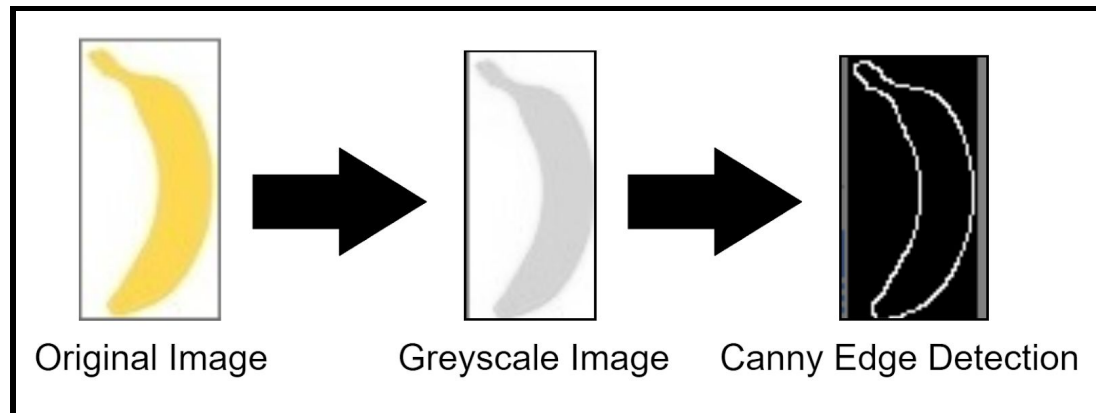
1

[https://www.researchgate.net/publication/4376224\\_Image\\_processing\\_of\\_an\\_agriculture\\_produce\\_Determination\\_of\\_size\\_and\\_ripeness\\_of\\_a\\_banana](https://www.researchgate.net/publication/4376224_Image_processing_of_an_agriculture_produce_Determination_of_size_and_ripeness_of_a_banana)

2

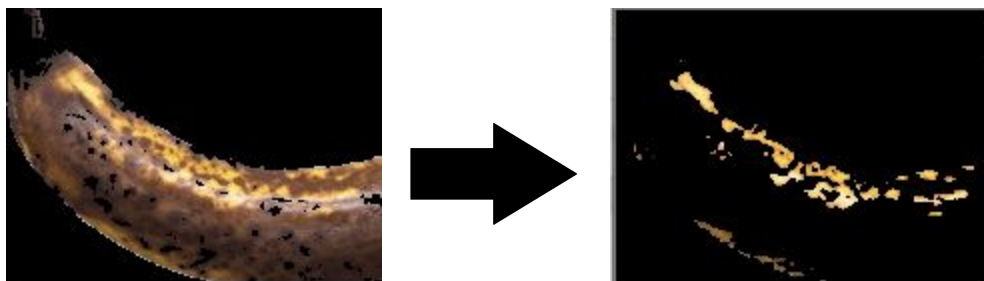
[https://www.researchgate.net/publication/229734831\\_Application\\_of\\_Image\\_Analysis\\_for\\_Classification\\_of\\_Ripening\\_Bananas](https://www.researchgate.net/publication/229734831_Application_of_Image_Analysis_for_Classification_of_Ripening_Bananas)

Our solution's algorithm has multi-scaling functionality, which allows the algorithm to detect bananas varying in size (that are similar enough to match the given template image). As well, rotational invariance to a degree (using multiple template images) has been implemented so that bananas with different degrees of rotation can be found which increases the accuracy and flexibility of the algorithm.



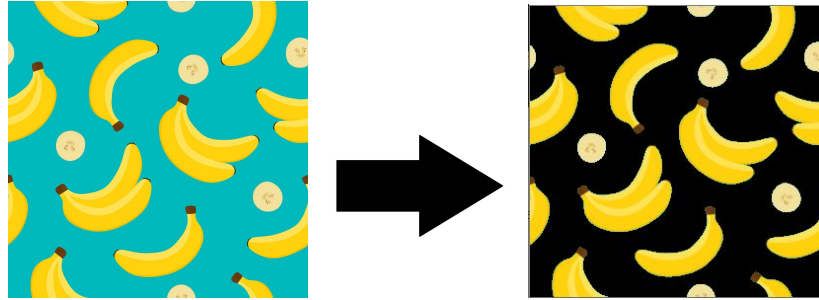
**Figure 1:** Preparing a template image example with Canny Edge Detection

The algorithm uses a template image in order to detect bananas. In figure 1, it shows how each template image was converted to the template used in the Canny edge detection method. However, to disregard rotation as a factor, 23 template images were provided to the algorithm which was effectively the same template image rotated in 15-degree increments.



**Figure 2:** Example Process of Ripeness Function (yellow mask extracted and then compared)

Figure 2 shows how the ripe region (yellow-like colours) of the banana is extracted. These images are then compared to the original image using the mean squared error to determine if it is a match. If there is too large of a difference in similarity then the banana is further analyzed to determine whether it is unripe or overripe using similar methods.



**Figure 3:** Example Background Subtraction Comparison

An example of background subtraction working on a test image can be seen in the figure above. Based on the colour thresholds for green, yellow, and brown colours used, the algorithm will ignore any undesired colors.

### Known Limitations

However, there are limitations to the algorithm implemented. For example, the algorithm was implemented to find the ripeness of only one banana in the image. It was designed with this due to the safe assumption that images containing multiple bananas would intuitively have the same ripeness for every banana (e.g. at a supermarket, bananas would always have the same ripeness). Therefore images containing multi-colored bananas will not work properly. Other limitations include the algorithm covering mainly the side view of bananas. However, the threshold for the banana matches is not too strict therefore certain other slightly different orientations of bananas are considered when doing template matching. A non-strict threshold also comes with certain disadvantages like having false positives however I feel that this is justified because bananas can be shaped very differently and the algorithm will always choose the best match (therefore it will choose the actual banana over a banana-like object).

## Pseudo-Code

This algorithm was implemented in Python 3.7 and took advantage of libraries such as openCV (a library for real-time computer vision) and numpy (a library for more advanced computing). The following is example pseudocode for the algorithm:

```
# where template is a path to a folder containing template images
# where image is the input image to the program (loaded by cv2.imread() function)
def bananaRipeness(template, image):
    resize(image) # load input image and resize if it's too large (1)
    imageFix(image) # apply background subtraction to input image (2)
    cvtGrayscale(image) # convert to grayscale
    for image_path in os.listdir(templatePath): (3)
        cvtGrayscale(template) # converts template to grayscale (4)
        cannyEdge(template) # applies Canny edge detection on template (5)
        for scale in np.linspace(0.1, 1, 50)[::-1]: # loops 50x from 1 to 0.1 (6)
            resize(image) # resizes input image using scale variable (7)
            cannyEdge(image) # applies Canny edge detection on image
            match = matchTemplate(template, image) (8)
            if match > threshold: (9)
                storeResults() # keep track of important variables (10)
    if match not found: (11)
        return original image with "no banana found" text
    else:
        checkRipeness(image) # determine ripeness of detected banana (12)
        if ripe == 1:
            return image with "ripe" text
        else if (ripe == 0)
            return image with "overripe" text
        else:
            return image with "unripe" text
```

**Figure 4:** Algorithm Pseudocode

The pseudo-code shown above does not reflect the actual code for the algorithm. Many of the functions in the code above are abstractions of blocks of code and do not include every argument used in the actual Python implementation of the program. The following table details the justification and reasoning for the different code blocks labelled in the figure above.

**Table 1:** Significance and Analysis of Pseudocode Functions

Code Section	Analysis	Equivalent Python Program Code
(1)	This method resizes a given image if its dimensions are too large. This is to increase both the algorithm's accuracy and computational runtime (large images take longer to process and analyze pixel by pixel). Since the algorithm resizes the input image down to 10% (as shown in code sections 6 and 7), this means that very large images even at 10% of their original size may still be too big for pixel-by-pixel comparison with the template (resulting in inaccurate images).	Lines 23 to 27
(2)	This function applies background subtraction to the input image. It ultimately removes any non-banana colors (e.g. colors that are not white, yellow, green, brown, or black) from the image as they are irrelevant details. This is important as it reduces unnecessary edges in the Canny edge detection output.	Line 29 and lines 133 to 137 (function body)
(3)	This line of code is necessary to loop through the folder containing the template images. This is necessary so that the algorithm uses multiple template images when using template matching allowing the algorithm to accept different rotations of a banana (matches highest result based off of these images).	Line 33
(4)	This pseudocode function was used to convert the given template image to grayscale. This is done to prepare it as input for the Canny edge detection algorithm.	Line 38
(5)	This method takes the grayscale image and converts it to an edge-based version using the Canny edge detection algorithm. This algorithm has four main stages: noise reduction, intensity gradient calculation, non-maximum suppression and hysteresis thresholding <sup>3</sup> . Template matching was done using these edge-based versions as the shape of the banana was a more significant factor in the algorithm's accuracy then the color (as a banana can range from green to yellow to black).	Lines 39
(6)	This line of code loops from 1 to 0.1 ([::-1] causes it to loop in reverse) in 50 decrements/iterations. These values were important in tweaking the algorithm's runtime and accuracy (e.g. increasing iterations will have reduced benefits the higher you go and will cause a longer runtime). The value being updated by the loop was used to resize the input image for template matching.	Lines 42
(7)	This function resizes the input image using the scale factor (variable) updated by code block 6.	Line 44
(8)	This function uses the cv2.matchTemplate() function which returns a correlation map. The cv2.minMaxLoc() function is then used to find max intensity values (high matching) in this map and to obtain the matched area.	Line 53

<sup>3</sup> [https://docs.opencv.org/3.1.0/da/d22/tutorial\\_py\\_canny.html](https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html)

Code Section	Analysis	Equivalent Python Program Code
(9)	This code block compares the result of the match to the threshold value (an integer determined through testing) to see if it is within the reasonable range of being a banana.	Line 57
(10)	This code block runs if the <i>if</i> condition returns true. In this block of code, any important variables and values are stored so that this match can be saved and further analyzed for the banana ripeness output.	Lines 61 to 63
(11)	This block of code occurs when no reasonable match was found in the algorithm. In the case this occurs, the text 'No banana found' is appended to the output image which is then returned by the function.	Line 65
(12)	In the case a potential banana match is found, a function is run on the match to determine the ripeness of the banana. In this function, yellow and brown masks are subtracted from the matched image to create a new image and this new image is compared to the input image using mean squared error to determine the degree of difference (the smaller the better). The similarity between these images is compared to a threshold value to determine whether the banana is unripe, ripe, or overripe. Mean squared error is the average squared difference between the estimated values and what is estimated.	Line 73 and lines 111 to 128 (function body)

# Experiments

## Experiment Layout

The experiments conducted will test different types of images with unique and distinct features. Each experiment will include a *Description* section which will describe why the experiment is being tested, a *Results* section which includes the images being tested and their respective results, and an *Analysis* section which will discuss the results in-depth. The following is a summary of the different experiments that will be discussed:

- Case Study: Single Banana
- Case Study: Multiple Bananas
- Case Study: Different Backgrounds
- Case Study: False-Positives
- Case Study: No Bananas or False-Positives

## Evaluation Metric

The evaluation metric will be a result of a subjective eye-test (visual) that the tester will undergo themselves. This is due to the fact that the best classifier for a banana under these set of circumstances is the human eye. The evaluation metric will measure the correctness of these two different things tested by the algorithm:

1. Was there a banana detected?
2. If there was a banana detected, is it unripe, ripe, or overripe?

The metric for whether or not a banana is detected will include 2 possible values:

- YES: There was a banana detected
- NO: There was no banana detected

This metric will be considered a success if the value produced (YES/NO) is correct according to the subjective visual inspection of the image by the tester.

The metric for whether the detected banana is unripe, ripe, or overripe will include 3 possible values:

- Unripe: The banana is unripe
- Ripe: The banana is ripe
- Overripe: The banana is overripe

This metric will be considered a success if the value produced is correct according to the subjective visual inspection by the tester.


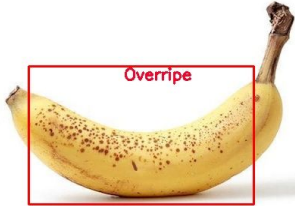

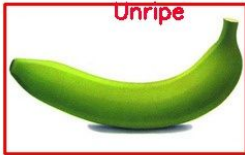

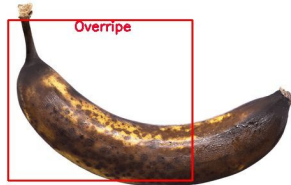


## Experiment #1) Case Study: Single Banana

### Description

*Case Study #1* is focused on testing an image with a single banana that has a white background. The idea for this test case is that it is the easiest test possible for our banana ripeness algorithm. The test will test three different images, where there will be one image that should result in a ripe, unripe, and overripe result.

### Results

Description	Original Image	Resulting Image	Success /Failure
Ripe banana in a white background.			Success
Unripe banana in a white background.			Success
Overripe banana in a white background.			Success

### Analysis


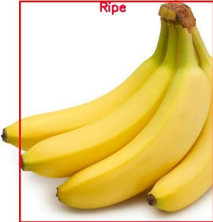

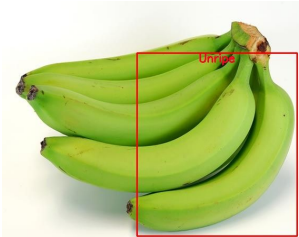

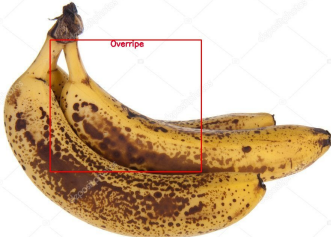
After running the algorithm on the above images, we can see that all three images were successfully identified. What this proves is that the algorithm will work on the most basic case in a real life application, where the user will take a picture of the banana with a white background behind it.

## Experiment #2) Case Study: Multiple Bananas

### Description

*Case Study #2* is focused on testing images of multiple bananas that share the same level of ripeness. The background is also free of false-positives and different elements. The idea is to see if the algorithm can deal with multiple bananas in the same image, and if it can return the correct result.

### Results

Description	Original Image	Resulting Image	Success/ Failure
Multiple ripe bananas.			Success
Multiple unripe bananas			Success
Multiple overripe bananas			Success

### Analysis


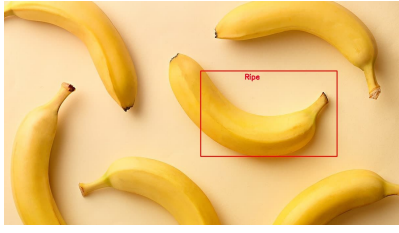
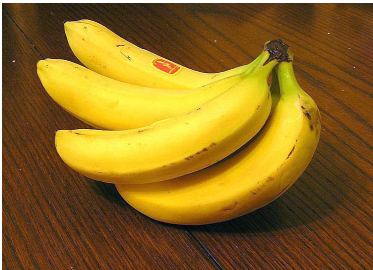
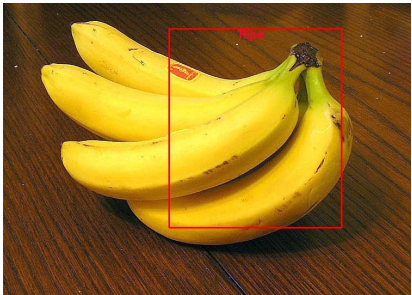

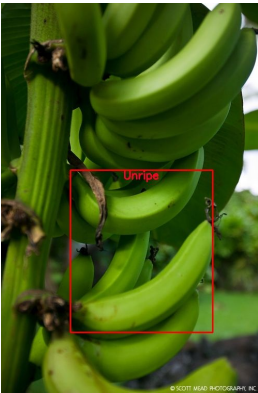
The tested images above were successfully identified. What can be concluded from the above results is that if a user were to take a picture of multiple bananas of the same level of ripeness (which expected in nature), the algorithm would be able to identify the level of ripeness for the group of bananas correctly.

## Experiment #3) Case Study: Different Backgrounds

### Description

*Case Study #3* is focused on testing images where there are an arbitrary number of bananas that have a background that may have false-positives or different elements. This is an important test case because it will verify how well the algorithm is able to detect the banana with different elements in the background that may cause an incorrect result. The incorrect result could be a result of the banana not being detected in the image or the background having an influence over whether or not the banana is ripe or not.

### Results

Description	Original Image	Resulting Image	Success/ Failure
Multiple ripe bananas with a yellow background.			Success
Multiple ripe bananas on a wooden brown table.			Success
Multiple unripe bananas with a green background with other green elements.			Success

## Analysis

The algorithm did a good job detecting the banana(s) with different background colours. The different background colours were chosen with the intent to try and skew the results that the algorithm would produce.

The first test case had a bunch of bananas separate from each other, laying on a surface that had a constant background that resembled a yellow-orange colour. The algorithm produced a successful result, as one of the bananas was detected and labeled as being ripe. As mentioned in the algorithm discussion, the algorithm only detects one banana in the image, so the fact that it did not detect multiple bananas does not make it a failure.

The second test case had a group of bananas attached to each other and laying on a brown wooden table. The idea was to exercise the possibility of a part of the brown table being detected as an overripe banana, or for the selected region to correctly label the group of bananas as a banana but to incorrectly label the region as being overripe. This would be possible because the algorithm could detect the the banana as being ripe of the selected region of the image has its ripeness influenced by the brown colour of the table.





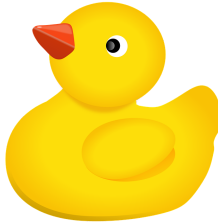
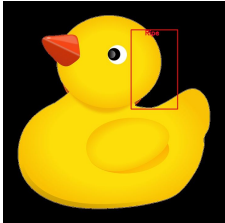
The third test case was an image that included only green (unripe) bananas and other green elements throughout the image. The reasoning for this test case was to see how the algorithm deals with different elements in the background of the image that share the same colour as the banana to be detected. However, the algorithm successfully detected the banana in the image and correctly labeled it as being unripe.

## Experiment #4) Case Study: False-Positives

### Description

In *Experiment #4*, different images that might resemble a banana in some way will be tested to see if the algorithm will be able to recognize that the image does not contain a banana in it.

## Results

Description	Original Image	Resulting Image	Success/ Failure
A person in a banana suit			Fail
A sunflower			Fail
A yellow rubber ducky toy			Fail

## Analysis

Based on our results above, the algorithm does not do a good job in detecting false positives. There is built-in code that is supposed to detect whether or not there is a banana located in the image or not, and that mechanism fails with the above test cases. This is due to the fact that the threshold that the region of pixels has to pass is not that high, and the algorithm picks the highest region of the image that passes this threshold. Due to this design method, false positives are not expected to be ignored.

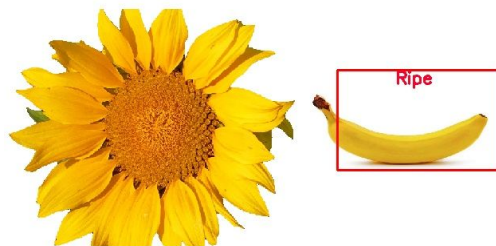
The person in a banana suit test case failed in an unexpected way. The banana suit was not detected as a banana, but rather the individual's shoe was detected as being a banana. A reasoning for this could be because the colour of the shoe could possibly be a banana, and the template-matching portion of the code passed the threshold for being a banana or not. The shoe was in the end, the part of the photo that most resembled a banana and was thus identified as a banana.

The sunflower image had petals of different shapes and sizes, and it was little surprise as to why it was detected as being a banana. Being detected as ripe is seen as a positive in this test case (due to the yellow colour), but unfortunately the algorithm was not sophisticated enough to detect that there is in fact no banana in the image.

The rubber ducky image was detected as being a banana. The same reasoning for why this occurred was used in the first paragraph of this experiment discussion

An improvement to be made to the algorithm to detect false positives would be to add a machine learning component to the algorithm. The idea would be to train a neural network to determine whether or not an image is a banana or not. However, this would take a lot of resources (time and data), so it was not considered to be a good idea for this project. In addition, the idea of this program is to determine whether a banana is ripe, not if the image captured contains a banana or not. The user testing the photo should be only testing it on images that truly do contain bananas.

As mentioned earlier, the algorithm finds the region in the photo that best resembles a banana. The image in figure 5 contains the same sunflower that had a petal detected as a banana, and a banana which was placed in the image. By examining it, we can see that the actual banana (not the petal) was detected as being the banana in the image. This supports the claim that the algorithm picks the best region of pixels that look like a banana.






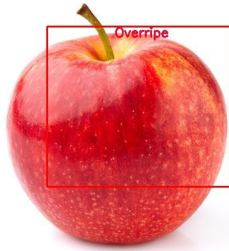




## Experiment #5) Case Study: No Bananas or False-Positives

### Description

The idea behind *Experiment #5* is to test the algorithm on images where there are no bananas or false-positives. The absence of false-positives in the image is a subjective measure. The algorithm will only be considered successful if there is no banana found in the tested image.



## Results

Description	Original Image	Resulting Image	Success/ Failure
A night sky			Success
An apple			Fail
A field of green vegetation			Fail
A black/dark grey Mazda sedan			Success

## Analysis

The banana detection portion of the algorithm was successful with half of the images tested. The algorithm does not rely on colours to detect if there is a banana or not, but rather uses template matching. Due to this being the case, the failures in this experiment are rather interesting.

The first image tested was a night sky, where no banana was detected. The reasoning for testing this image is because the image does not have any distinct objects in it. It is one of the most basic test cases for testing if the algorithm can recognize that there is not banana in the image. With that being said, the image consists of blue shades of colour, with a strip of yellow near the bottom of the image. The algorithm did a good job recognizing that there was no banana in the image.

The second image is a red apple. The reasoning behind testing this image was that an apple is an object that you would see regularly in a grocery store, and it would be important for the algorithm to recognize that an apple is in fact not a banana. Interestingly enough, a banana was detected in this image. A reason behind this could be that the apple has different shades of red on the skin in the area detected as being a banana. The algorithm could be too sensitive to these colour changes and thus viewed these different shades of being objects themselves. One of these patterns found could then have been resembling a banana. The following image below is the image that was processed by the detection portion of the algorithm:



The above image shows that the algorithm recognizes these lighter shades of red to be a valid shade (colour) for a banana. Due to this, the algorithm is able to find a banana in the above image. These shades are naturally occurring, so testing was done on a cartoon apple (solid colour red) below to test further support this theory:

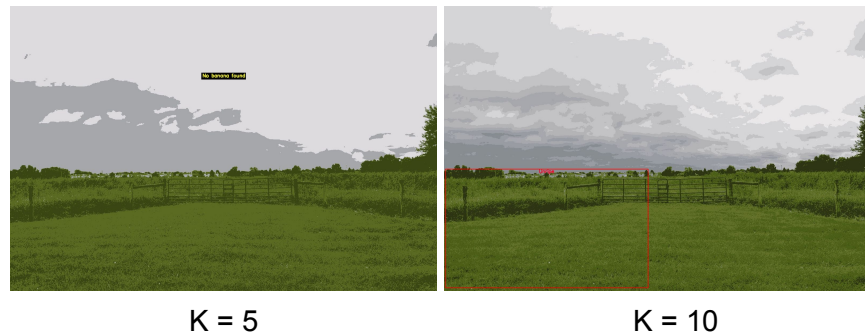


As we can see, no banana was detected in the above image. The apple has a consistent shade of red across it, thus avoiding the aforementioned problem with apples in real life. This also means that the curvature of the apple is not the reason for the detection of a banana.

The third image tested was a field of vegetation. In this image there are different shades of green and black, which resulted in a banana being detected. To support this theory, a different approach will be taken than we did in the previous example with the red apple. The



image of the field will be processed through a K-Means Clustering algorithm which will reduce the amount of detail in the photo (edges) and will produce the following:



As seen in the two images above, having K equal to 5 will result in the image having less detail (edges) and will decrease the chance that a banana is detected (there was no banana detected with K=5). When having a K value of 10, the image more closely resembles the original image and is thus more susceptible to having a banana detected by accident (a banana was detected to be overripe).

The fourth image tested was a car. The image consisted of shades ranging from black to white, while also having some orange and red on the lights of the car. The reason for testing this image was to see if the image would detect a banana if it consists of different shades of the same colour. This occurred in the previous two test cases so this was another attempt at testing this flaw. However, no banana was found in this image which thus resulted in a successful test.

## Reflection

### Results

The algorithm was tested on different test cases and thus revealed its strengths and weaknesses. These strengths and weaknesses were reflected on both the ability for the algorithm to detect whether a banana is present in the photo, and what level of ripeness the banana is (unripe/ripe/overripe).

With regards to detecting bananas in the image when they are present, the algorithm never failed to detect a banana in any of the test cases, even when there were false-positives in the image to try and get the algorithm to detect the false-positive instead of the actual banana. However, in the absence of a banana and in the presence of a false-positive, the algorithm would identify the false-positive as a banana. In addition, when testing images that subjectively did not have any false-positives, the algorithm would sometimes detect in a region of pixels that subjectively do not look like or resemble anything close to a banana (example: a field in *Experiment #5*).

The algorithm was inconsistent in some cases when trying to determine the ripeness of a banana. This was due to the fact that the algorithm did not isolate the shape of the banana from the rest of the image, but rather would measure the ripeness based on the selected rectangular area where the banana was found to be located. This resulted in inconsistent results where a yellow banana could be detected as overripe due to darker shades in the background (*Experiment #4*).

Experiment #	# of Successful Tests	Total # of Tests
1	3	3
2	3	3
3	3	3
4	0	4
5	2	4
<b>TOTAL</b>	<b>11</b>	<b>17</b>

Based on the table above, the algorithm had a 64.71% success rate through our test suite of images in this report. The failures can be attributed to the algorithm's inability to test effectively for false-positives and its tendency to be lenient when detecting a banana.

## Improvements

Improvements to the algorithm could be made to both the ability to detect if a banana exists and to the detected ripeness level of the banana match.

The algorithm could improve banana detection by being more strict on what is classified as a banana (e.g. by making certain thresholds such as the banana match threshold stricter) . However, the question of how necessary this is has to be asked. The algorithm already does a good job detecting the banana if it is in fact in the image, and the whole point of the algorithm isn't to be a classifier for if there is a banana in the image but rather to be a ripeness level detector. Therefore the application and purpose of the algorithm is different than the application of classifying objects.

The ripeness detection can also be potentially improved in different ways. One possible way to improve it is modifying the checkRipeness() function. Instead of the current technique, of creating new images using HSV masks, and comparing the differences using mean squared error (MSE), the algorithm can instead find the median/dominant pixel color in the matched area and then pick that pixel colour as the colour to be tested for ripeness. Another idea would be to use structural similarity (SSIM) instead of mean squared error (MSE) to compare similarity

between two images. As SSIM uses comparison of local regions instead of global regions like MSE, there is an increase in accuracy but decrease in performance<sup>4</sup>.

Finally, it is important to note that there are other algorithms that have the potential of being a much more effective and accurate banana ripeness detector. For example, machine learning algorithms such as the Haar cascade classifier algorithm are capable of detecting multiple bananas varying in size, orientation, and angle to a high degree of accuracy. However, these algorithms come with the cost of time. Based on past experiences with machine learning, we felt that we would not have the time and data to train and test an accurate banana classifier within the project deadline.

## Code

The web application makes it easy to test different images one by one. The code and the README can be found here: <https://github.com/peken97/BananaRipenessDetector>

The standalone Python 3 script for running the algorithm on many input images in a folder can be found here: <https://github.com/GV79/bananascript/blob/master/bananaIdentification.py>. This code was used as the basis for column 3 in Table 1 above.

## Conclusion

In this paper, we created and tested a web application that was able to detect and grade the ripeness of a banana in a image. Python was used to implement our template matching algorithm which took advantage of the Canny edge detection method to detect the banana and the HSV colour space to determine whether it was unripe, ripe, or overripe. Future work will include more testing and implementation of some of the improvements listed in the paper.

---

<sup>4</sup> <https://www.pyimagesearch.com/2014/09/15/python-compare-two-images/>