Interactive Local Tone Adjustment

Note: All borrowed code and examples were taken directly from MATLAB2018b documentation, mostly with substantial modification.

Link to the presentation:
https://uwprod-my.sharepoint.com/:p:/g/personal/btruchan_wisc_edu/EYk9O2Kn875PhPAvTPg1LUABdlI8GWgOg2ZMSrLPqtdMiw?e=CfJm4p

Link to the zip file containing our code. (Read the readme, it's important!)
https://drive.google.com/file/d/1PVqcd_X-MErumFMS9w6hitO1_rPJ6ViA/view?usp=sharing

Motivation- Simply being able to adjust the tone (Saturation, luminosity, warmth, other common adjustments) globally is often insufficient. So we wanted to provide users the ability to apply local tone adjustments, using an automatic algorithm instead of something like a magnetic lasso in photoshop, to allow for a far greater range of interpretations of an image. The improvements we made over the magnetic lasso approach include: being faster to use, easier for the user to make their selection, and allowing editing of the parameters that are considered for segmenting to avoid situations where an image can't be easily segmented by a magnetic lasso or other algorithm.

Current state of the art
As far as the current state of the art in "Interactive Local Tone Adjustment" it's probably the tool explained here: https://link.springer.com/article/10.1007%2Fs00371-017-1422-5
The tool is from August of 2017 and cites the paper we're working off of. It also seemingly uses a similar technique relying on input "from a few user strokes" and implements "depth of field simulation, dehazing and tone mapping". The paper itself is behind a paywall.

The approach defined in "Interactive local adjustment of tonal values" would likely be defined as the "current state-of-the-art" in our scenario. The main difference between their approach and ours is we produce an image mask instead of a tone map. Additionally, our segmentation is done first and the user is asked to select from the segments we selected; whereas, their user selects general segments they want in the image.

Why is this problem important?

Usability is key for any software. Software that can only be used by experts can, by definition, can only reach an extremely narrow field of people. Beyond that, at a certain level of tedium, it just isn't worth it to employ certain techniques for the vast majority of people in a vast majority of cases. But what if this difficulty could be dramatically cut down using a new tool, so that someone at say, a hobbyist level could have access to this functionality? Suddenly there could be a relative explosion of new image types due to a massively expanded pool of potential users.

Our approach and its advantages

There is absolutely no doubt that the approach given in the paper is far more powerful and feature-laden than what we're going for. However, the complexity of what they're doing is enormous. Aside from the fact that their solver relies on doctorate level knowledge of linear algebra, among other things, it also runs a gauntlet of complex computations, including an optimization problem. Their approach can't be understood, updated, ported to a new framework/language, or maintained without an enormous amount of subject matter expertise. Our approach is quite straightforward and portable. Our plan can be envisioned as a series of implementation-agnostic, "black box functions" consisting of common algorithms. The whole process consists of largely atomic, common image modification operations. Generally:

1. Segment the image
2. Isolate selected regions in the form of a mask
3. Adjust the isolated image with existing algorithms for hue, contrast, etc
4. Replace the regions on the original image with the modified versions

Our approach was initially inspired by this paper: http://www.cs.huji.ac.il/~danix/itm/.

The Implementation

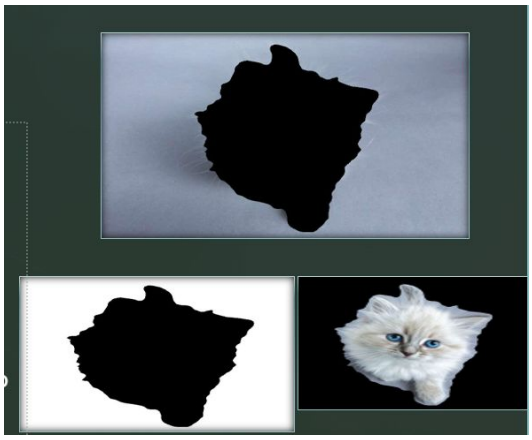
We wrote a couple of functions to be used within MATLAB that will be called to generate a final image. These general functions include: extractmulti.m, extractmultiorig.m, imgSeg_Simple.m, newSeg.m, and stitch.m. In order to use the general functions the user will follow this outlined process:

- Load in an Image(preferably RGB as Greyscale produces inaccurate results, shown later)
- Use either extractmulti.m or extractmultiorig.m to be given a subject, negative, and image mask corresponding to the subject (figure 1)
- Do some Image adjustment to either the subject, the negative, or both(Hue, contrast, flatten, etc.)
- Stitch the subject and negative back together using stitch.m

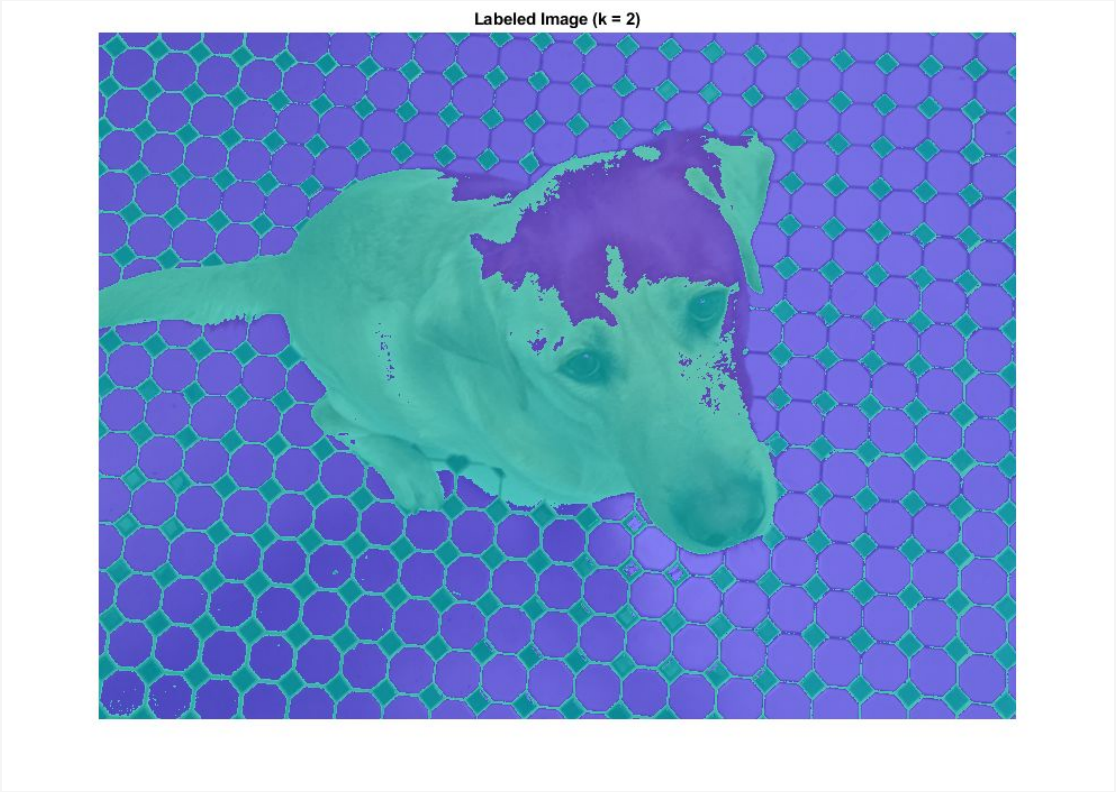
Our segmentation process is done within extractmulti.m or extractmultiorig.m and can be broken down into two different methods, which can be applied to different scenarios.

Method 1

This method corresponds to the code within extractmultiorig.m and imgSeg_simple.m. The segmentation done in imgSeg_simple.m is an improved version of k-means image segmentation as defined in the MATLAB R2018b documentation. This improved version is used to supplement the image with information about the texture in the neighborhood of each pixel. We obtain the texture information by filtering a grayscale version of the image with a set of Gabor filters. Additionally, spatial qualities are accounted for in this improvement as well. This additional information allows the k-means clustering algorithm to prefer groupings that are close together spatially. Following (figure 1) is a visual walkthrough of Method 1s process:

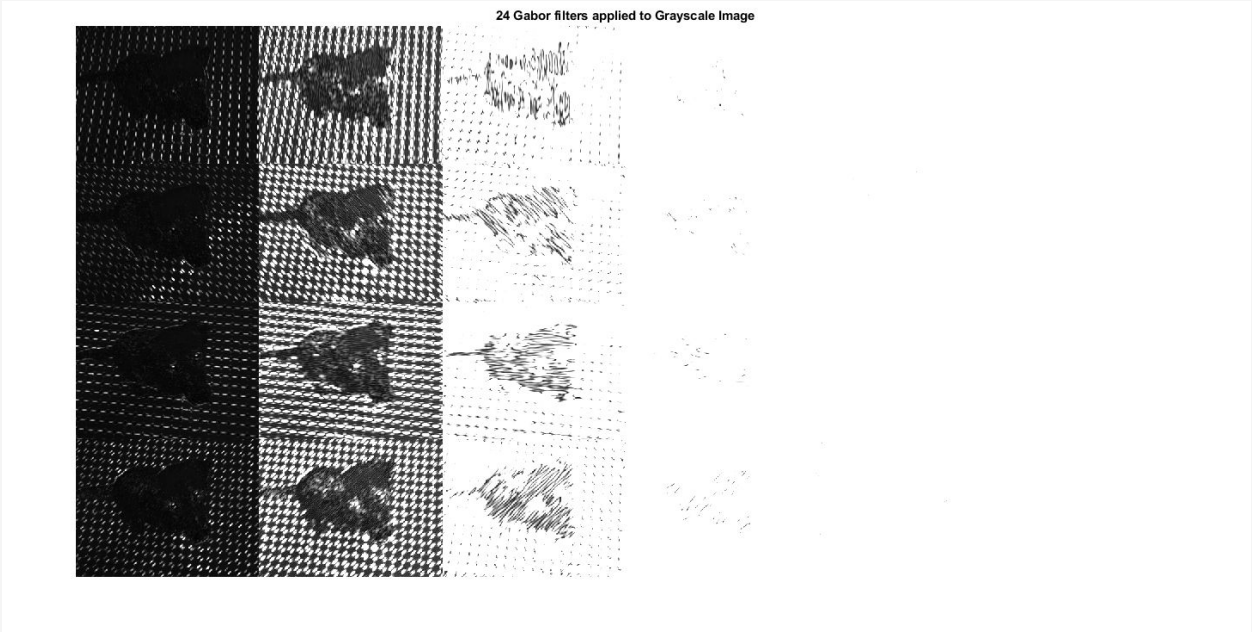


(figure 1) Top: Subject Image, Bottom Left: Image Mask, Bottom Right: Negative Image

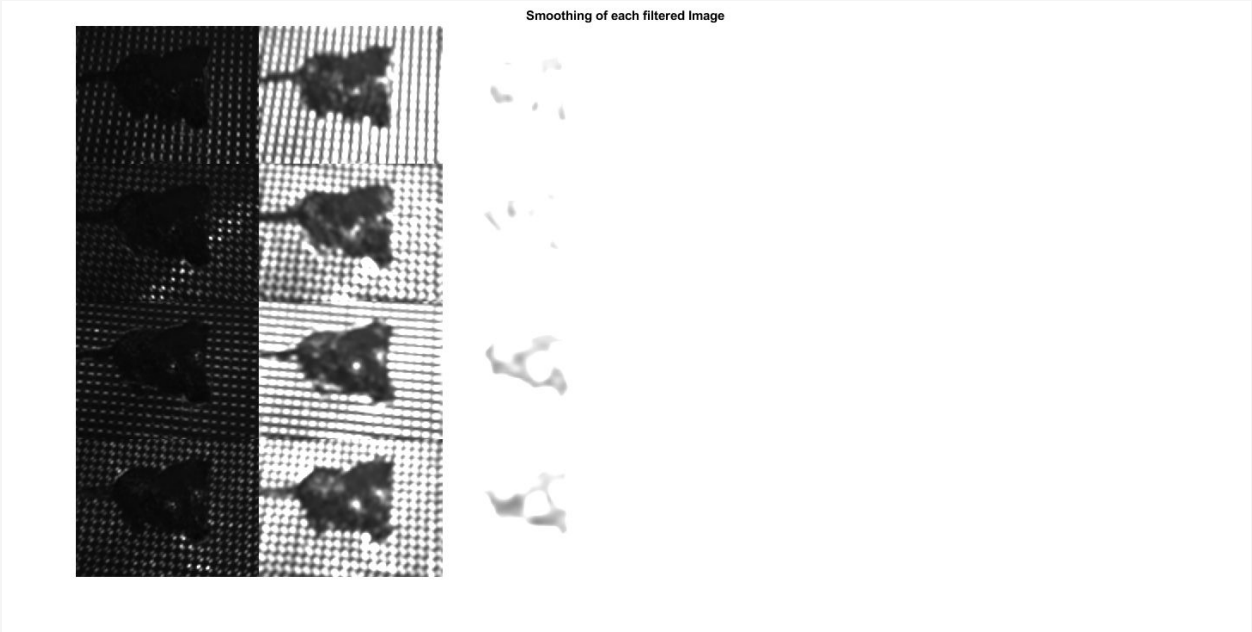


This is an example result from basic K-means clustering with no improvements. As you can see, the selection would be too inaccurate to give good results selecting the background or the dog.

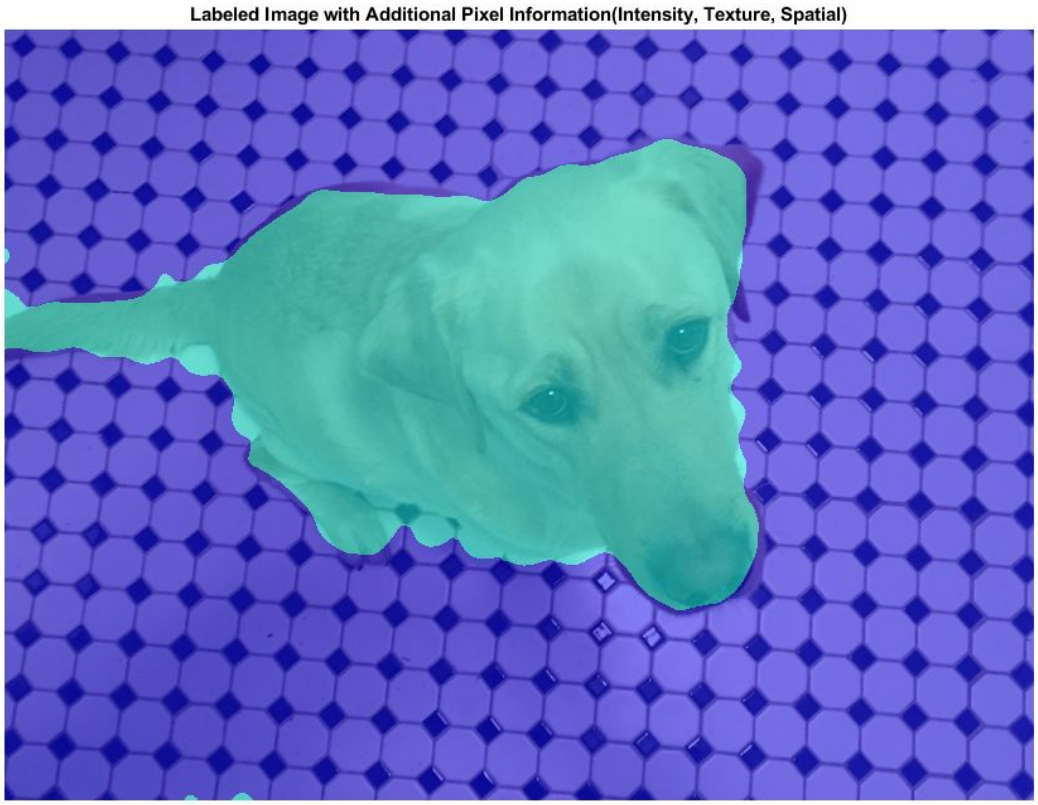
So this method was improved by adding Gabor filters and Spatial information.



Here are the Gabor filters being applied to the above image.



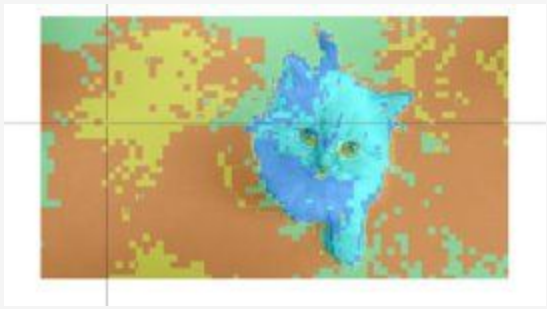
The Gabor filters are then smoothed. This texture information along with each pixel's x-y values are concatenated and clustered via K-means. This then gives the following improved result:



This segmented image is then displayed by extractmultiorig.m and the user is asked to select which segment(s) they would like to edit and to click off the image when they are done. After that, the selected segment(s) are saved as the image subImg.jpg, the unselected parts as negImg.jpg, and an image mask corresponding to the selected part called imgMask.jpg. Then these parts will be adjusted by some MATLAB code(look at flattenfield.m as an example) and then the subject and the negative are stitched back together using stitch.m

Method 2

The extraction and segmentation in Method 2 corresponds to the functions extractmulti.m and newSeg.m. The segmentation in newSeg.m is L*a*b color space segmentation using k-means as defined by the MATLAB 2018b documentation. An example of the results from newSeg.m is shown below.

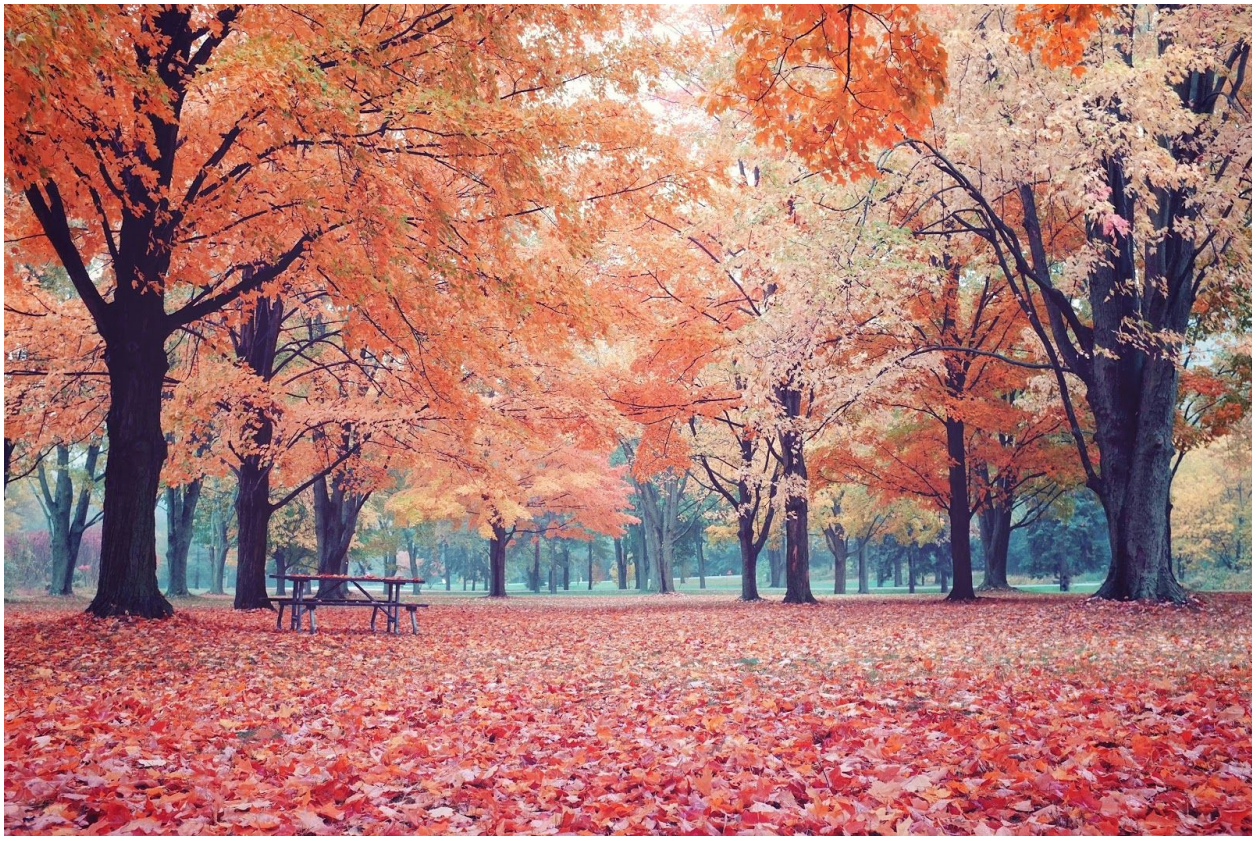


As you can see above, the segmentation around the edges of the cat are very tight when using the Lab color space, but the eyes are included in the same segment as the background due to their similar coloration. Here we came up with a novel solution that solves a host of seemingly unrelated problems. First, we'll show some of of these, and then we'll show the fix.

Parts of Grass have been grouped with the cat & butterfly due to similar coloration.

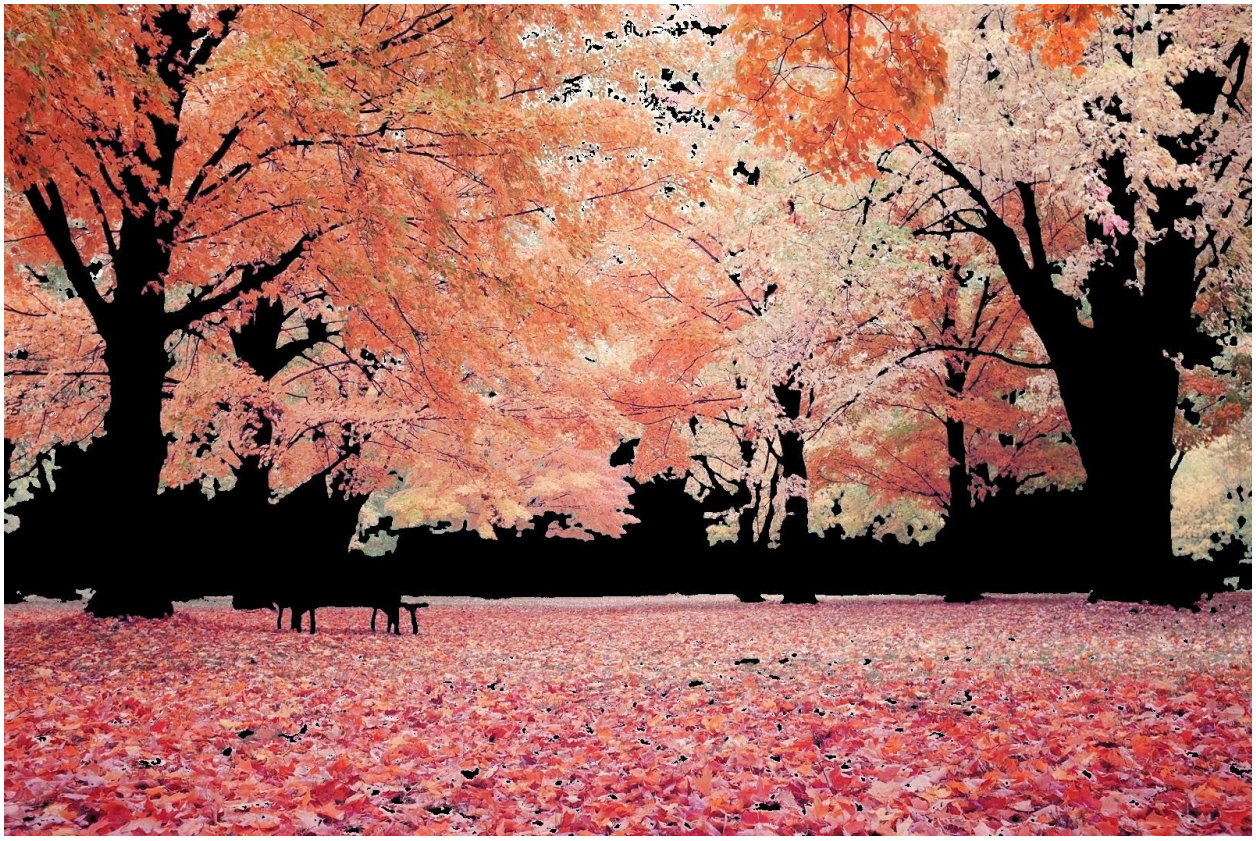
Even though humans can clearly distinguish 3 regions in the following image (ground, middle area, and canopy), the algorithm does not segment in this way. Therefore, an operation like: change the colors of the leaves on the trees but leave the ones on the ground the same is not

really possible using a naive approach.



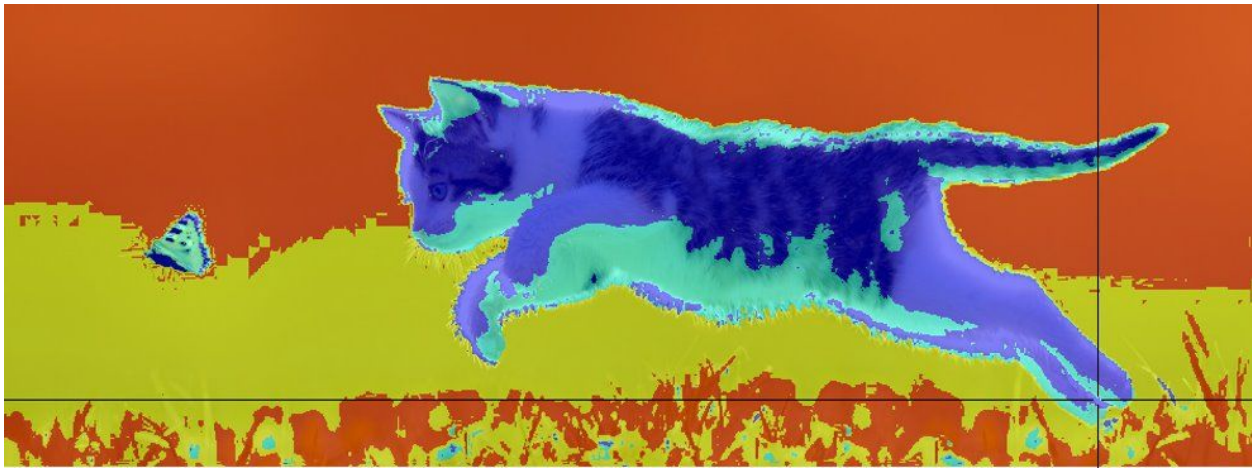
After segmenting this image, it’s clear to see that both the top and bottom of the image have been grouped together due to their similar coloration.As it stands, it’s not possible to edit just the top or just the bottom.

This is what is returned when you attempt to segment out either the canopy or the ground.

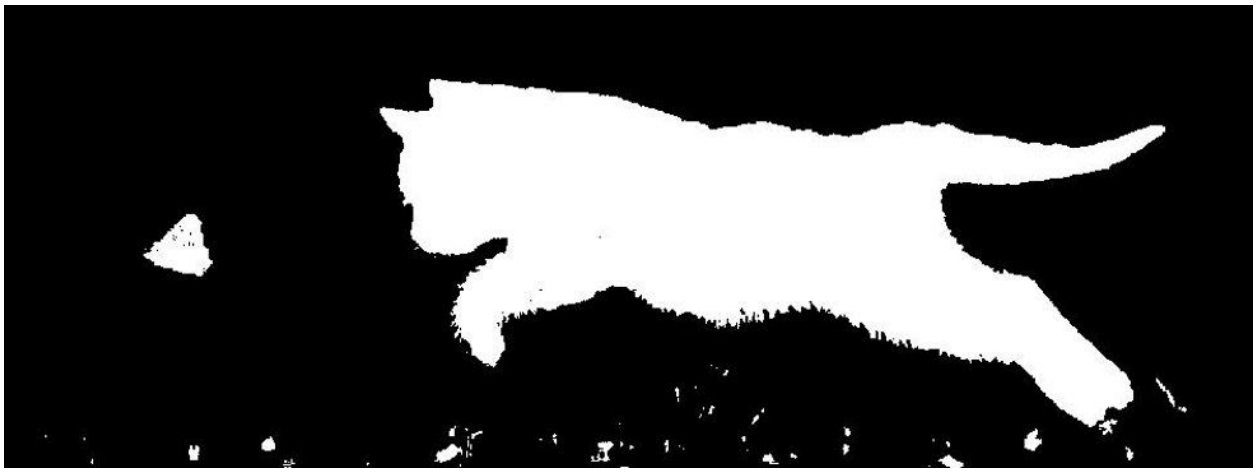


The Solution to everything:
Internally, the object to be segmented out is simply a binary image mask that is applied to the color image. Therefore, we can do an extra stage where we label each disconnected object. Then, it's as simple as asking the user to confirm the objects they wish to select, then creating a binary mask of just those pixels. This is one of those cases where user input is able to solve the problem far more efficiently than trying to fix things programmatically. Here's a demonstration of this process.

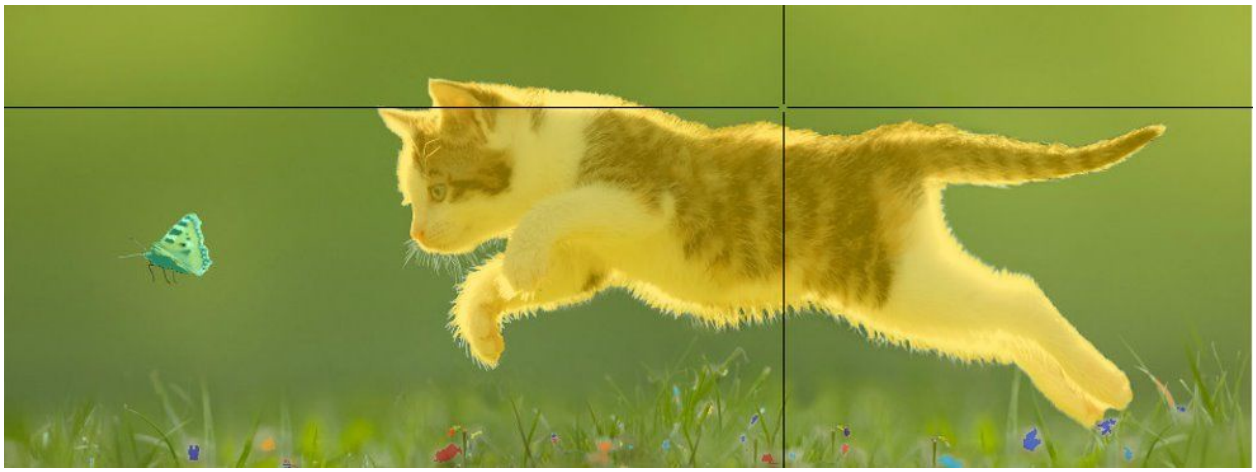
The user is faced with the choice to select their desired segments.



This is the internal representation of the cat after the user selects both blue segments.



Here the user is asked again to select the object they want to keep. Not only will this eliminate the noise, it's now possible to segment the butterfly and the cat separately.



The results of the fix:

The noise is almost entirely eliminated.

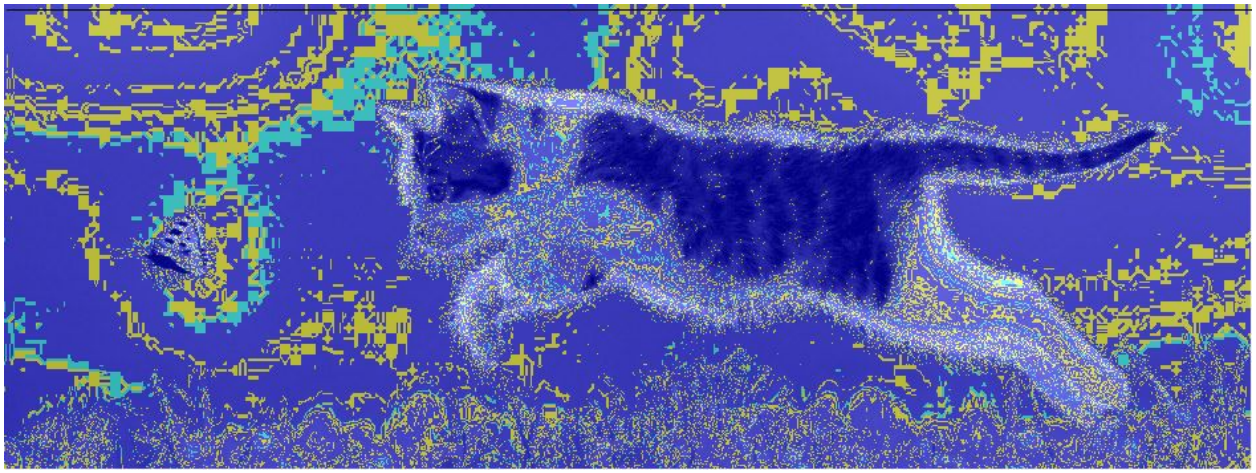
Because the canopy and the ground are disconnected, the user can just select the piece they want, and we get this nice result.

Problems-

We could do a better version of stitching with a little bit of smoothing to improve how the final image looks under visual inspection. This would remove areas of the image that make the stitching of the final image obvious. As shown below,

This is using method 1 segmentation,method 2 gives below result. We decided the results from method 2 are pretty good, so we should not need to distort the image with smoothing.

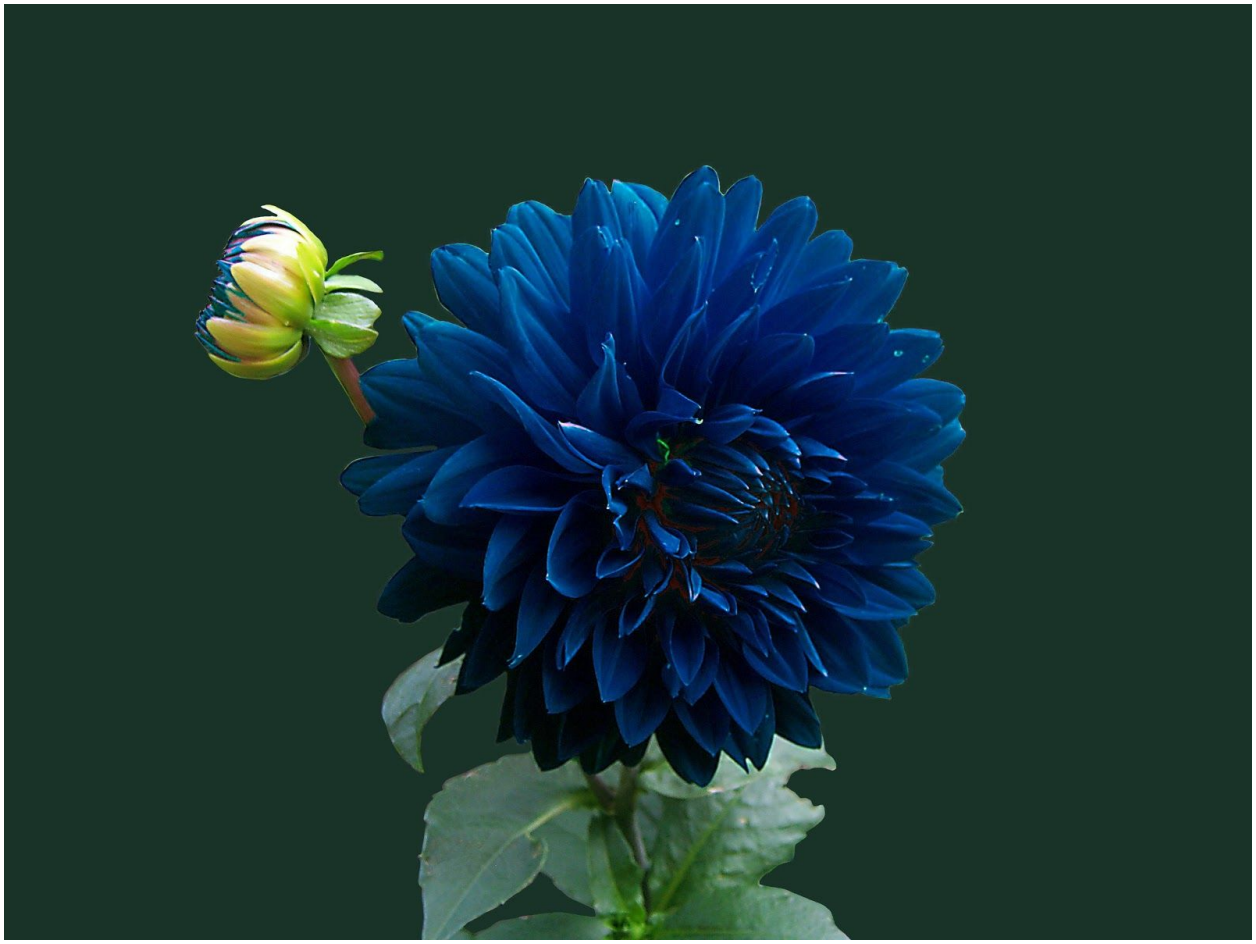
The next example shows how our k-means clustering isn't robust to images where the the background and the subject are too close in intensity. To demonstrate this we show a greyscale image, but k-means clustering will also fail in an RGB image when the intensity difference isn't great enough.



Another problem we face is demonstrated below. What is happening in the below images is as follows: The blue sky is being selected as the subject, the subject is tinted darker blue to look later at night, and then the subject is stitched back in. In the second image, if you look between the branches of the trees in the bottom middle you can see some of the old sky color. This is not the intended result as there is similar color sky very close by.

Results

Here the cat and the butterfly (subjects) have their contrast increased and the negative is turned to grayscale.



The above two images demonstrate the red in the flowers being segmented out and shifted blue.

The following two images demonstrate a case where the background is already blurred, and you want to edit only the subject area. (Corresponds to LocalLaplacian.m)

The above image is an example of the difference between applying a flattenfield to the entire picture(left) vs. just the background(right)(corresponding to flattenfield.m)

Here we tried an extension where we take a background image, match it to the size of the subImg, and then stitch them together. There are strange distortions/artifacts around the edge of the masked out area (the black cat shape on the galaxy). We suspect that this is from stretching the background image improperly, but the source of this bug remains a mystery.