Image Processing and Analysis on Forestry Data

Author: Spencer Morse

Client: Dr. Noah Charney

COS 598 Image Processing, Spring 2022, University of Maine
Orono, Maine
spencer.morse@maine.edu

I. Introduction

This document serves as a final write up for the image processing and analysis of forestry data project, for COS 598.

II. Purpose of the Project

The major purpose of this project is to gain experience and understand how to use and potentially write image processing algorithms. The goal is to use these techniques to analyze data and apply that knowledge to real world applications.

A. Background on Project

For this project the intent was to take years worth of aerial forestry images and remove any potential noise, in order to attempt to estimate the amount of biomass; or fallen trees. In this case, the images have large amounts of noise, shadows, that make it very difficult to tell the difference between fallen trees and standing ones. Shadows are considered signal and valid data, but in this case they are unwanted and prevent us from detecting fallen trees.

B. Major Goals for Project

There are a few important goals for this project. The first, and most obvious being, using image processing methods to attempt to remove the unwanted noise in the images. More specifically, I wanted to attempt to automate a process of sending in the forestry images and running image processing methods, like thresholding, noise reduction, and segmentation to attempt to create a mask of the shadows. From there we could then use that mask to remove the shadows from the images, and begin to estimate the amount of biomass.

Being new to image processing, other goals included, trying and testing different image processing techniques to see what results they derive. I largely wanted to gain experience with these techniques and how they can be applied to real world data, in an attempt to solve a problem.

III. Data Description

This section will explain the type of data that was collected and used within this project.

A. What is it and Where does it come from?

The data used in this project comes from Dr. Noah Charney. The data consists of aerial images taken of a conservation land outside Nashville, Tennessee. Each image was gridded out at 150ft x 150 ft, or 30m - 30m, tiles. The data ranges from 2010 - 2018 in 2 year intervals. This means we have images from the same portions of land for every 2 years from 2010 -2018. It is also important to note that the images were taken at various times of the day, making the shadows direction inconsistent.

Fig 1. 41-56_2018 - Data Example

B. *Pre-existing Software and Algorithms*

There was no pre-existing software used for this project. Although there is already software out there that will remove shadows from images, I wanted to create a program from scratch in order to gain the experience and create a program that was more personalized to this data in particular. As for algorithms used, that will be explained in sections to come.

IV. Tools

Throughout this project I used a variety of tools and software to attempt to accomplish the goal. Firstly, the project is done completely in Python 3 programming language. I chose this language because it has an extensive collection of image processing libraries like OpenCV [1] and SimpleITK [2].

As for the coding environment and documentation I used a web-based interactive computational environment called Jupyter Notebook [3]. This allowed for me to do all my coding and testing in one place, as well as provide format for my publishings.

Lastly, I used GitHub [4] to create a repository to store the files and data.

V. Methods

There are several different image processing methods that I used to do this project, including:

- Thresholding
- Hue Saturation Value Masking
- Non-Local Means Denoising
- Segmentation
- Mathematical Morphology Methods
    - Dilation
    - Erosion
    - Opening
    - Closing

Firstly, we will discuss thresholding. Thresholding is a simple type of image segmentation in image processing. It allows us to change the pixels of an image in order to make the image easier to analyze. With thresholding we can accomplish things like taking a grayscale image and using thresholding to create binary images. For the sake of this project, I attempted binary thresholding, however the results were not beneficial to my goal. Another important aspect of thresholding is that we can use histograms to find appropriate threshold values that will be used in the thresholding process. Which brings me to my next method, Hue Saturation Value (HSV).

HSV provides a numerical representation of an image which corresponds to the colors within the image, similar to a histogram. So essentially, HSV separates *luma*, or the image intensity, from *chroma*, or the color information. This is important because using HSV we could find appropriate threshold values that could be used to create a mask of the shadows. This will be explained more in the results section.

Another method used is denoising. This is a method used in image processing and computer vision where we try to estimate the original image by suppressing noise in an image. In particular I used a Non-local means denoising algorithm, defined as:

$$u(p) = \frac{1}{C(p)} \int_{\Omega} v(q) f(p,q) dq \quad [5]$$

where Ω is the area of an image, p and q are two points within the image, u(p) is the filtered value of the image at point p, while v(q) is the unfiltered value. f(p,q) is the weighting function and C(p) is the normalizing factor. In non-local means denoising the noise is expected to be Gaussian.

Another important method used is basic mathematical morphology. Essentially mathematical morphology in image processing is simple operations done on images, typically binary images, to solve a problem within the image. These operations take in an image and a structuring element, which decides the nature of the operation. In this project I used the dilation, erosion, opening, and closing morphological transformations from the OpenCV library. Simply put, erosion is where all pixels near a boundary will be discarded depending on the size of the kernel or structuring element. Dilation is the opposite of erosion where it will increase the size of the foreground object. This is useful in things like noise removal where we would use erosion to remove the white noise and then dilate it to join the broken parts of an object. This is also known as opening and closing, which is essentially just a combination of either erosion and dilation or vice versa. This can be iterated as many times as necessary.

VI. Results

Here I will discuss the results from my efforts so far. Firstly, looking at figure 1 from above, you can see that there are many shadows in the image and it is quite noisy and hard to depict. The first step of action I took was creating a custom HSV scaler. The goal of this was to be able to read an image into it and find the appropriate threshold values in real time.

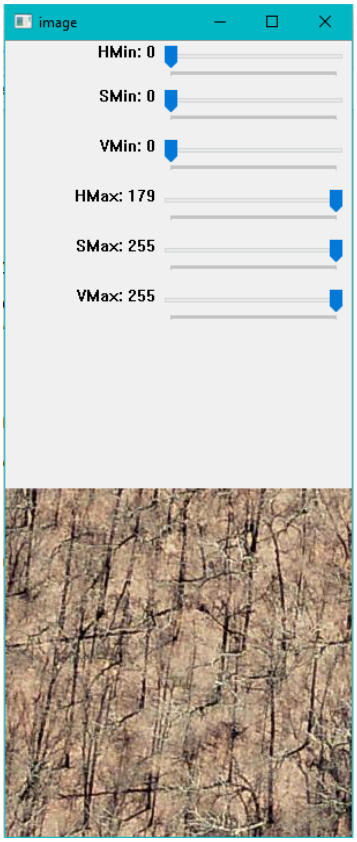

Fig 2. HSV Scaler

The source code for this scaler is located in the Jupyter Notebook on the GitHub for this project [4]. From this scaler I was able to adjust values and make a threshold mask. To accomplish this using the OpenCV library we use the code:

hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)

This takes in the image as an input and converts it from BGR (blue green red) to HSV. From there we use this code to make the mask:

mask = cv2.inRange(hsv, lower, upper)

This takes in that converted HSV image and uses the supplied values we give it from our scaler to create a mask of the image. From there we want to slice the mask to separate it from the image, denoted by:

output = cv2.bitwise_and(img,img,mask=mask)

So using that original image I attempted this approach. I found an appropriate lower value by changing the VMin value to about 130 while keeping everything else the same. This yielded a mask like:

Fig 3. VMin Threshold Mask

From the figure above, we can see that the HSV scale did a decent job of highlighting those shadows. Alternatively, you could adjust the VMax value and reverse the image so that now everything but the shadows are highlighted. However, this wasn't extremely helpful. You can still see that there are a lot of random isolated pixels and quite a bit of noise in the image.

This brought me to my next method which was non-local means denoising. I needed to find a way to potentially mitigate some of the noise in the image since they aren't perfect, and OpenCV supplied a method for this. The method is denoted as:

cv.fastNlMeansDenoisingColored()

This method takes in a few common parameters. One being a value, h, that decides the filter strength. The higher the value h, the better it removes noise but also removes detail. It also takes in a value for the template and search window size, typically odd numbers. It is also expected that the noise is Gaussian, when using this method. To implement this we use the following code:

dst = cv.fastNlMeansDenoisingColored(img,None,10,10,7,21)

Here we take in the image, none is just an output variable. I found 10 to be a good value for the filter strength, and 7 and 21 to be good values for the template and window size. So when running this on our original image I got something like:

Fig 4. Denoised Image

As you can see, this is quite a step up from before. We can now better see just the big shadows and even what looks like a fallen tree in the bottom middle of the image. The goal was to then use this image to potentially get a better mask. To do this I then took this denoised image and tried applying the HSV threshold again. That yielded this image:

Fig 5. Denoised Threshold Image

This gives us a much better mask than the previous one that hadn't had the denoising algorithm applied to it. However, it is still not perfect. My next idea was to try and see if the image could be cleaned up using mathematical morphology. To accomplish this I tried dilation and erosion separately on the image. I used a structuring element of a 3x3 kernel. OpenCV also contains methods for these transformations that are denoted like:

dilation = cv2.dilate(img, kernel, iterations=1)
erosion = cv2.erode(img, kernel, iterations=1)

These methods take in an image, along with the structuring element or kernel (3x3 in my case) and the number of iterations you want to apply the method. I didn't gain any significant results from these methods alone so I tried applying a combination of them, also known as opening. The code for this is:
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)

Unfortunately, this didn't work how I was hoping as it mostly just blurred out the threshold, opposed to fixing the isolated pixels and creating a better threshold mask.

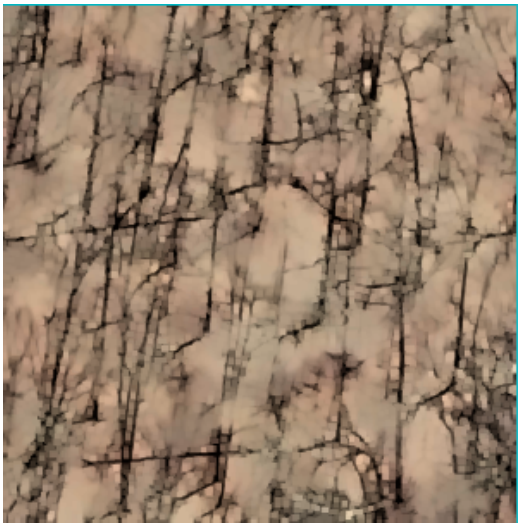

Fig 6. Opening on Threshold Image

That concludes the results I have gathered to this point.

VII. Future Work

There is a great deal of work that could still be done, given more time. I believe that firstly, being able to successfully implement a form of segmentation, where we are taking a mask of the shadows and subtracting them from the image to get a shadowless image.

In future work I think the algorithms and methods used could be improved to where we are getting better and more clear images from those methods. This will help to accomplish that goal of mitigating those shadows.

Lastly, a great future work would be to make the process more automated. Currently it is really only set up to where you put one image in and get one image out at a time. This is nice for small scale

projects, but something of this size where we have years and years worth of data, tt would be much more efficient if we could streamline images to the program and have it properly handle them.

VIII. Conclusions

In conclusion, I took this project on as a late project idea due to my original project falling through. With that being said this is just a starting point to the potential solutions to solving this problem. I believe there are many ways to accomplish this goal and with more time I would have liked to attempt some. Overall, this project and class has taught me so much about image processing and how we analyze images. I came into this with no prior experience or knowledge on the subject and am amazed at how much goes into image processing and even computer vision.

Acknowledgment

I would like to thank Dr. Noah Charney for supplying the copious amount of data used for this project. I would also like to thank Dr. Terry Yoo for giving me the opportunity to do this project, and the assistance he provided.

References

[1] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

[2] Z. Yaniv, B. C. Lowekamp, H. J. Johnson, R. Beare, "SimpleITK Image-Analysis Notebooks: a Collaborative Environment for Education and Reproducible Research", J Digit Imaging., doi: 10.1007/s10278-017-0037-8, 31(3): 290-303, 2018.

[3] Kluyver, T. et al., 2016. Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides & B. Schmidt, eds. *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. pp. 87–90.

[4] Spencer Morse, COS598 Final Project, (2022), GitHub repository, https://github.com/SpencerM0415/COS598-Final-Project

[5] *Non-local means*. (2011, November 26). Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Non-local_means

[6] *Image processing tutorials in OpenCV*. OpenCV. (n.d.). Retrieved from https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_table_of_contents_imgproc/py_table_of_contents_imgproc.html#py-table-of-content-imgproc

[7] Image processing. OpenCV. (n.d.). Retrieved from https://docs.opencv.org/3.4/d2/df0/tutorial_js_table_of_contents_imgproc.html