**CS 433 Digital Image Processing – Project Proposal**

**Segmented Images**

An image processing method using Otsu’s thresholding

|  |
| --- |
| Kevin Kemmerer  Winona State University  Kkemmerer15@winona.edu |

**Project Description**

This project aims to implement a project based on Otsu’s method of automatic image thresholding. My implementation of Otsu’s algorithm will ultimately compare k-NN to Otsu’s method. The intensity value of each pixel’s immediate neighbor will be taken into consideration. In layman’s terms I would be comparing the intensity of two pixels and based on those values you can set a desired threshold. I am to complete this project using Java as my desired programming language. I may however dabble with Python to get a better understanding of the algorithm before going further. If I decide that my implementation in Python is complete enough, I might stick with that instead. When using Otsu’s method, it is important for me to pick an image that shows variance in intensity levels from foreground to background. Considering the image that was given to me by Professor Ming Ma, Otsu’s method might be a little challenging. The image contains a lot of noise but I will try to implement it to the best of my ability.

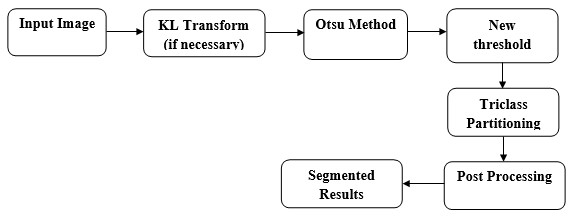
One should assume that this project will include an interface for the user to interact with. My goal for the user interface is to allow for variations in different deep learning algorithms. Ultimately, I want to compare the speed and efficiency of different deep learning algorithms using Otsu’s method. If however I do not get to the point of having multiple algorithms working. I would like the user to have the ability to choose different images to be processed.

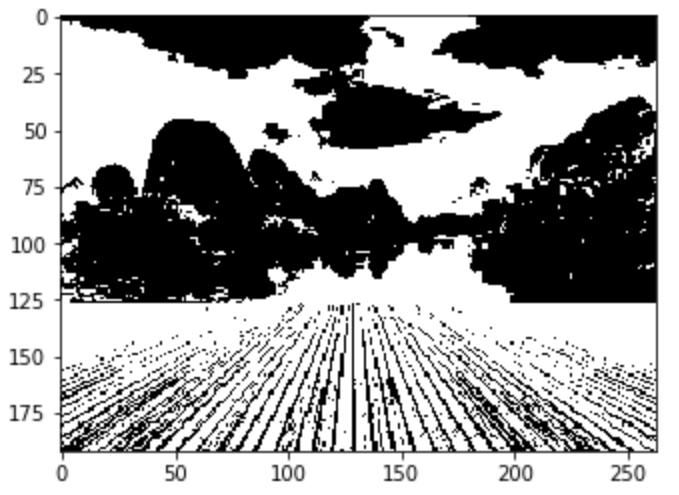
**Significance**

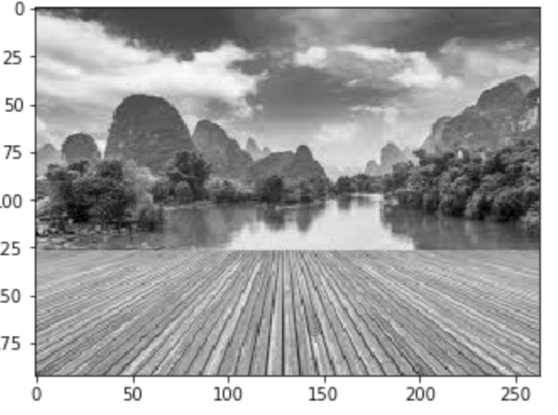
Otsu’s method of image segmentation or versions of it are used extensively in image processing these days. Everyone has a smart phone with the ability to take portrait photos. These portrait photos usually contain a blurred-out background. One way they implement that process is by using a deep learning image segmentation method. From deep fake photos to x-ray cancer images a version of deep learning image segmentation is almost required.

**Method**

In this project, I will compare the performance of the state-of-the-art Otsu’s thresholding to K-Nearest Neighbor. This method will allow the user to segment any image they input as well as display the after image.



****

****

**Acquisition**

I plan to allow the user to input an image but I’m unsure if this will be implemented considering the time limit of the project. The k-NN dataset will be acquired through comparing one pixel to all of it’s neighbors in terms of intensity. Based on the dataset of each pixel the image will be segmented in the form seen above.

Considering that I want the user to be able to input an image, there must be some sort of preprocessing. That pre-processing includes resolution scaling and turning that image to grayscale.

**Current Progress**

My progress on this project so far is less than I wanted it to be. I have however been looking at multiple Otsu’s method implementations. Also, I have created a preliminary k-NN algorithm using something more simple than images as you will see below.

# Global Variables  
k = 16  
training\_folder = #training dataset path  
minn\_folder = training\_folder + 'Minnesota/'  
health\_folder = training\_folder + 'Health/'  
tech\_folder = training\_folder + 'Tech/'  
  
def remove\_punctuation(s):  
 return re.sub(r'[^\w\s]','',s) # Changed remove punc function  
  
def file\_list(folder):  
 return [f for f in listdir(folder) if isfile(join(folder, f))]  
  
def all\_file\_list():  
 minn\_files = file\_list(minn\_folder)  
 for i in range(len(minn\_files)):  
 minn\_files[i] = minn\_folder + minn\_files[i]  
  
 health\_files = file\_list(health\_folder)  
 for i in range(len(health\_files)):  
 health\_files[i] = health\_folder + health\_files[i]  
  
 tech\_files = file\_list(tech\_folder) # added for tech files  
 for i in range(len(tech\_files)):  
 tech\_files[i] = tech\_folder + tech\_files[i]  
  
 return minn\_files + health\_files + tech\_files  
  
def file\_to\_word\_list(f):  
 fr = open(f, 'r')  
 text\_read = fr.read()  
 text = remove\_punctuation(text\_read)  
  
 return text.split()  
  
def get\_vocabularies(all\_files):  
 voc = {}  
 for f in all\_files:  
 words = file\_to\_word\_list(f)  
 for w in words:  
 voc[w] = 0  
  
 return voc  
  
def load\_training\_data():  
 all\_files = all\_file\_list()  
 voc = get\_vocabularies(all\_files)  
   
 training\_data = []  
  
 for f in all\_files:  
 tag = f.split('/')[7]  
 point = copy.deepcopy(voc)  
  
 words = file\_to\_word\_list(f)  
 for w in words:  
 point[w] += 1  
  
 d = {'tag':tag, 'point':point}  
 training\_data.append(d)  
  
 # print(training\_data) # test to make sure training data is correct  
 return training\_data  
  
def get\_distance(p1, p2):  
 sq\_sum = 0  
  
 for w in p1:  
 if w in p2:  
 sq\_sum += (p1[w]-p2[w])\*(p1[w]-p2[w])  
 else:  
 sq\_sum += p1[w]\*p1[w]  
  
 return math.sqrt(sq\_sum)  
  
# This function is implemented for seeing insights of training data  
def show\_distances(training\_data):  
 for i in range(len(training\_data)):  
 for j in range(i+1, len(training\_data)):  
 print('d('+str(i)+','+str(j)+')=', end='')  
 print(get\_distance(training\_data[i]['point'], training\_data[j]['point']), end=' ')  
 print()  
 for i in range(len(training\_data)):  
 print(training\_data[i]['tag'])  
  
def test(training\_data, txt\_file):  
 dist\_list = []  
 txt = {}  
 item = {}  
 max\_i = 0  
  
 words = file\_to\_word\_list(txt\_file)  
 for w in words:  
 if w in txt:  
 txt[w] += 1  
 else:  
 txt[w] = 1  
 # print(txt) # for testing  
 # print(item) # for testing  
 print("How related your article is to the current training data (0.0 is identical): ") # for testing+format  
  
 for pt in training\_data:  
 item['tag'] = pt['tag']  
 item['distance'] = get\_distance(pt['point'], txt)  
 print(item['distance']) # for testing+format  
  
 if len(dist\_list) < k:  
 dist\_list.append(copy.deepcopy(item))  
 else:  
 for i in range(1, k):  
 if dist\_list[i]['distance'] > dist\_list[max\_i]['distance']:  
 max\_i = i  
 if dist\_list[max\_i]['distance'] > item['distance']:  
 dist\_list[max\_i] = item  
  
 vote\_result = {}  
  
 # print(dist\_list) # for testing  
 for d in dist\_list:  
 if d['tag'] in vote\_result:  
 vote\_result[d['tag']] += 1  
 else:  
 vote\_result[d['tag']] = 1  
  
 print() # for pretty print out format  
 print("Total number of articles used for comparisons: "+str(vote\_result)) # for testing+format  
 sorted\_items = sorted(dist\_list, key=lambda items: items['distance']) # Sorting dist\_list by distance  
 # print(sorted\_items) # for testing  
  
 # Sorting dist\_list such that the result is the article with the lowest distance.  
 # Meaning the result is the training section which the article is most similar to  
  
 result = sorted\_items[0]['tag'] # Result is the 'tag' with the lowest distance i.e the most similar  
  
 # Removed voting because it was not taking into account how similar  
 # the article was to the training articles. This was only taking into account  
 # the number of articles in the dist\_list  
  
 # for vote in vote\_result:  
 # if vote\_result[vote] > vote\_result[result]:  
 # result = vote  
  
 return result  
   
def main(txt):  
 td = load\_training\_data()  
 # show\_distances(td) # for test usage only  
 print("Your article's category according to our training data is: " + test(td, txt))  
  
testing = input("Which article would you like to test? ")  
print() # for pretty print out format  
main(testing)

**Discussion**

The code above is testing an implementation of k-NN. The user inputs an “article” which is really just any kind of text document. The input text document is compared to training data that is already put into the program. The program will return a list of vectors showing how similar or dissimilar the users input article is to the training data and determine a category for the users article.

I will be using this sort of idea when implementing and comparing k-NN to Otsu’s method. The above code will be modified such that it takes an image as an input.

**\*\*README\*\***

**Original image required to be processed for the project is shown in the .pdf**

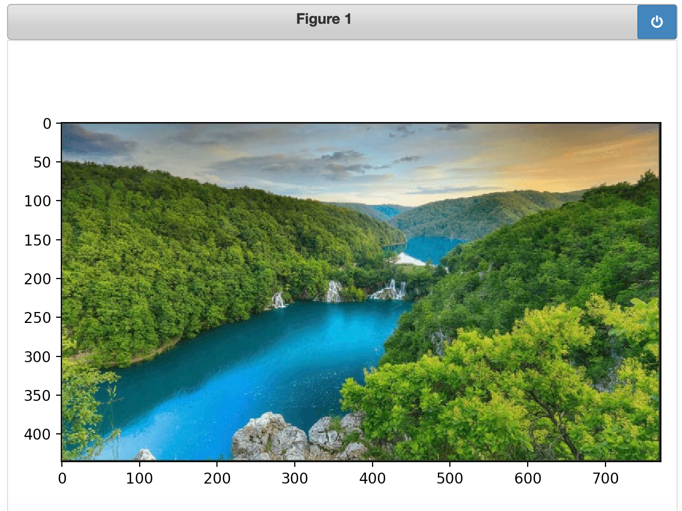
My application is fairly simple in operation. It is using Jupyter Notebook with built in widgets. The ipywidgets library allows for labels/text/boxes/buttons and much more providing a light weight GUI for users.

My application allows for users to input any image from a filepath or image name such as “image.jpg”. The user inputs their image into the text box and it is read from a file and displayed.

User input (File path/image with extension):

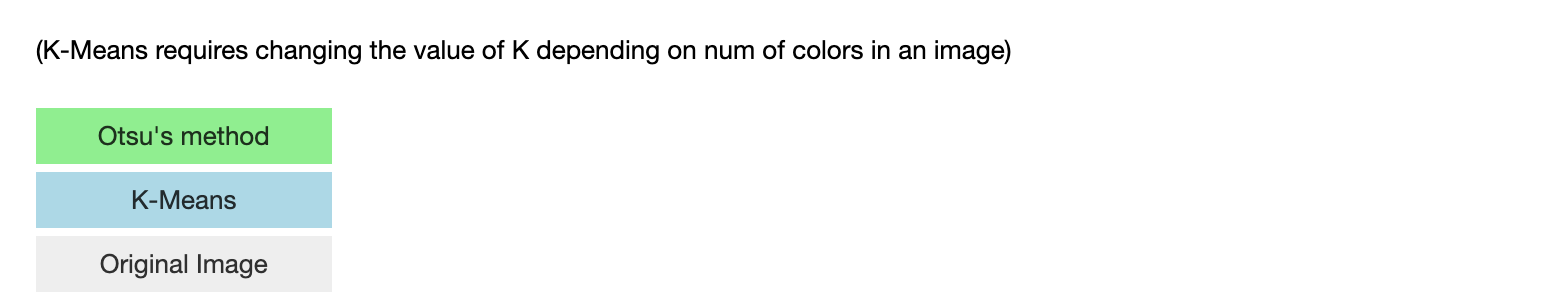


Image displayed for user:



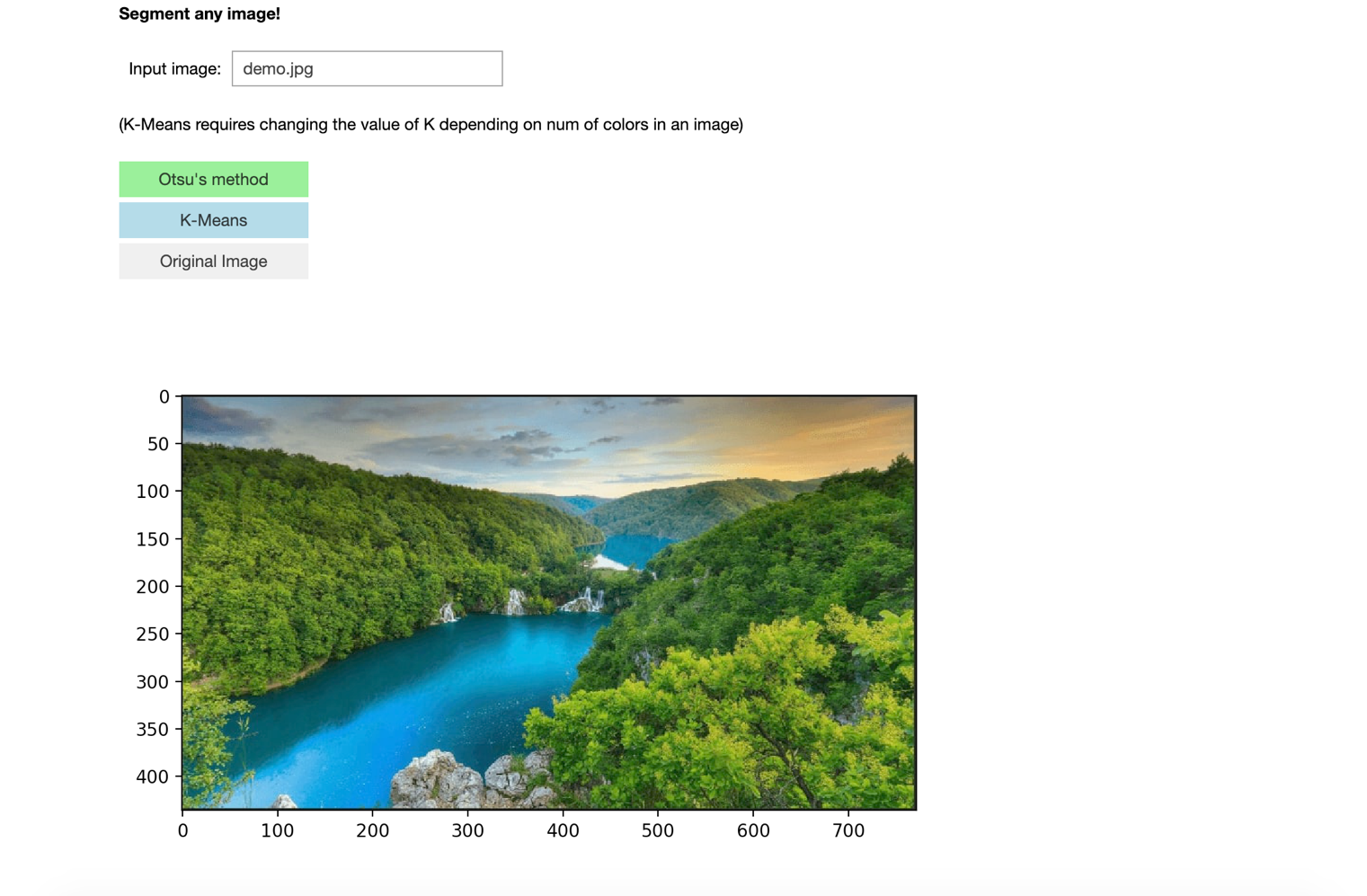
Once the initial image is displayed users can decide to “stop interaction” with his photo to save it in the widget.

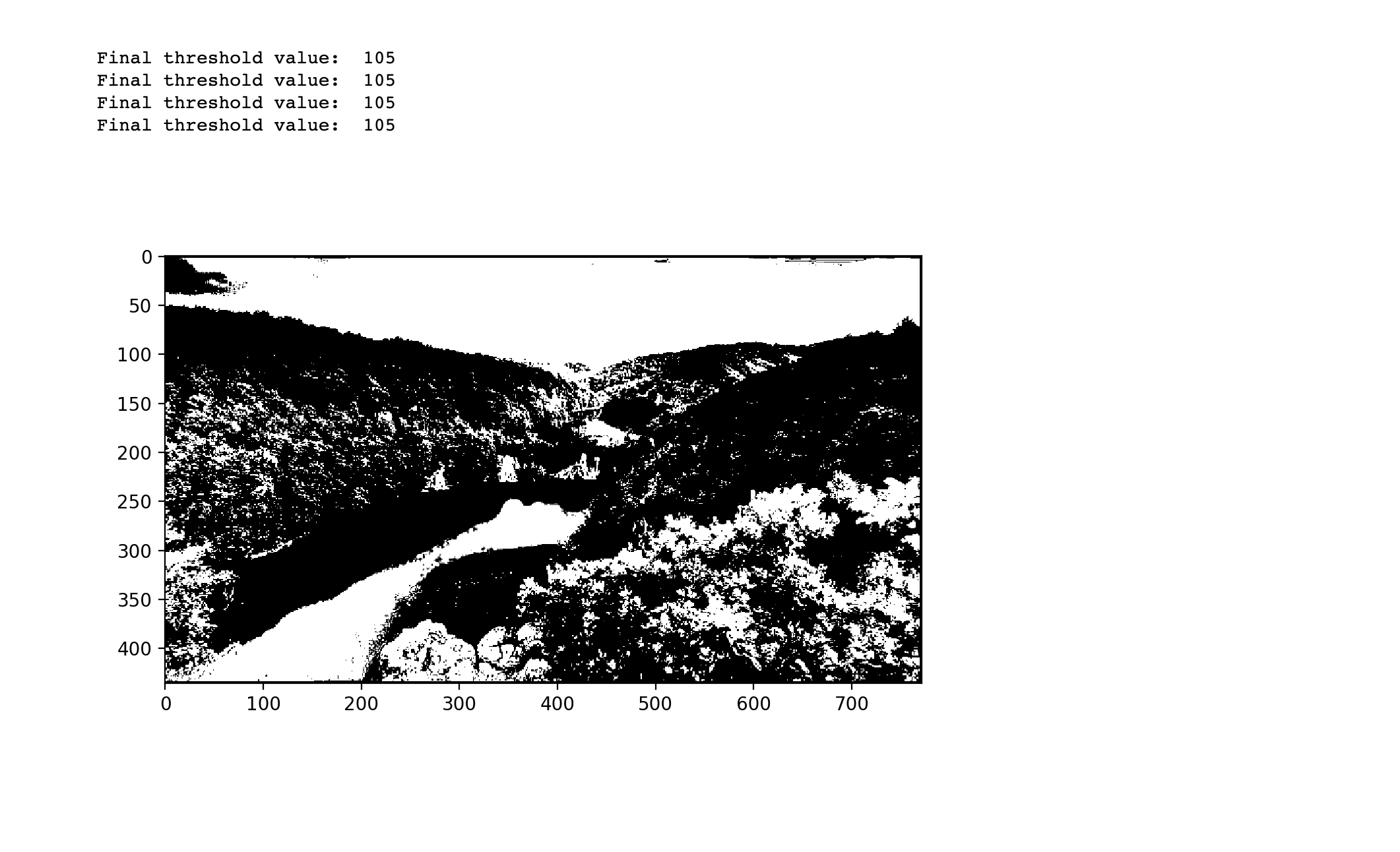
Once the user decides to stop interaction or not with the initial image(soft save to widget). There are buttons available for users to select how they would like to segment the image.

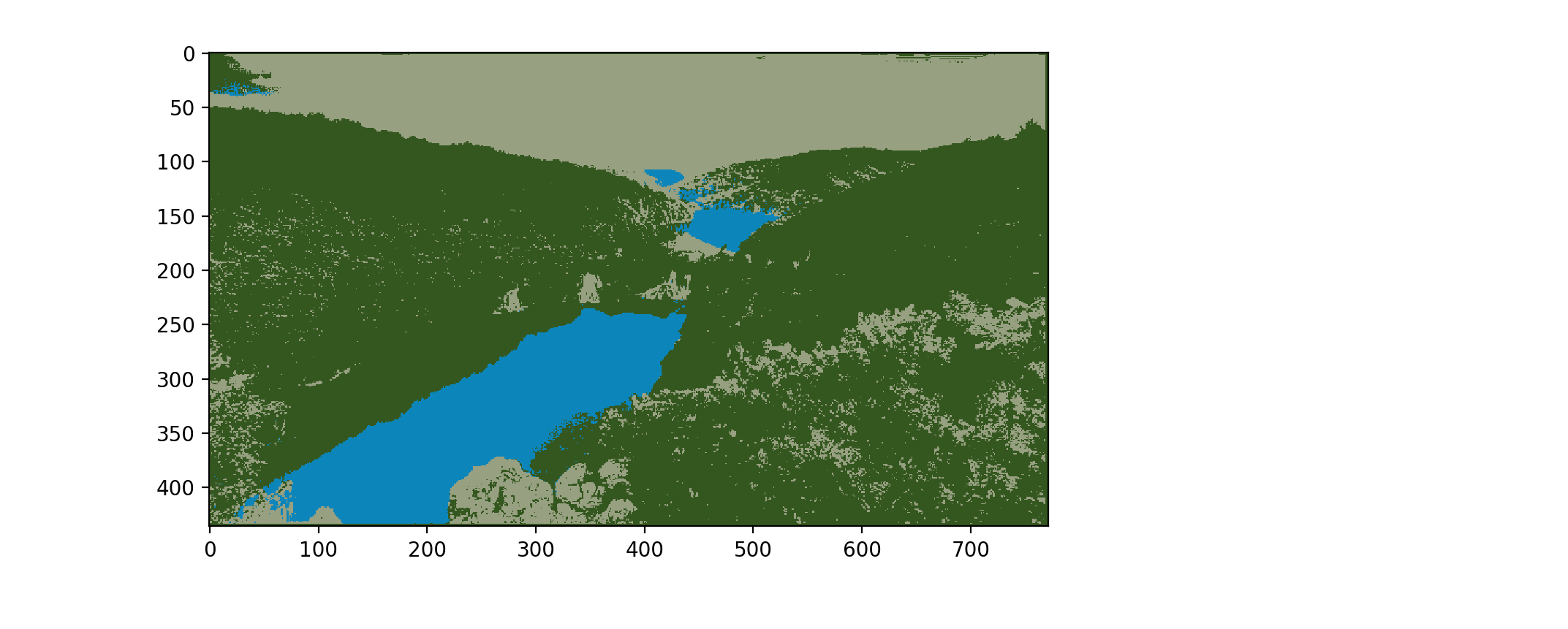


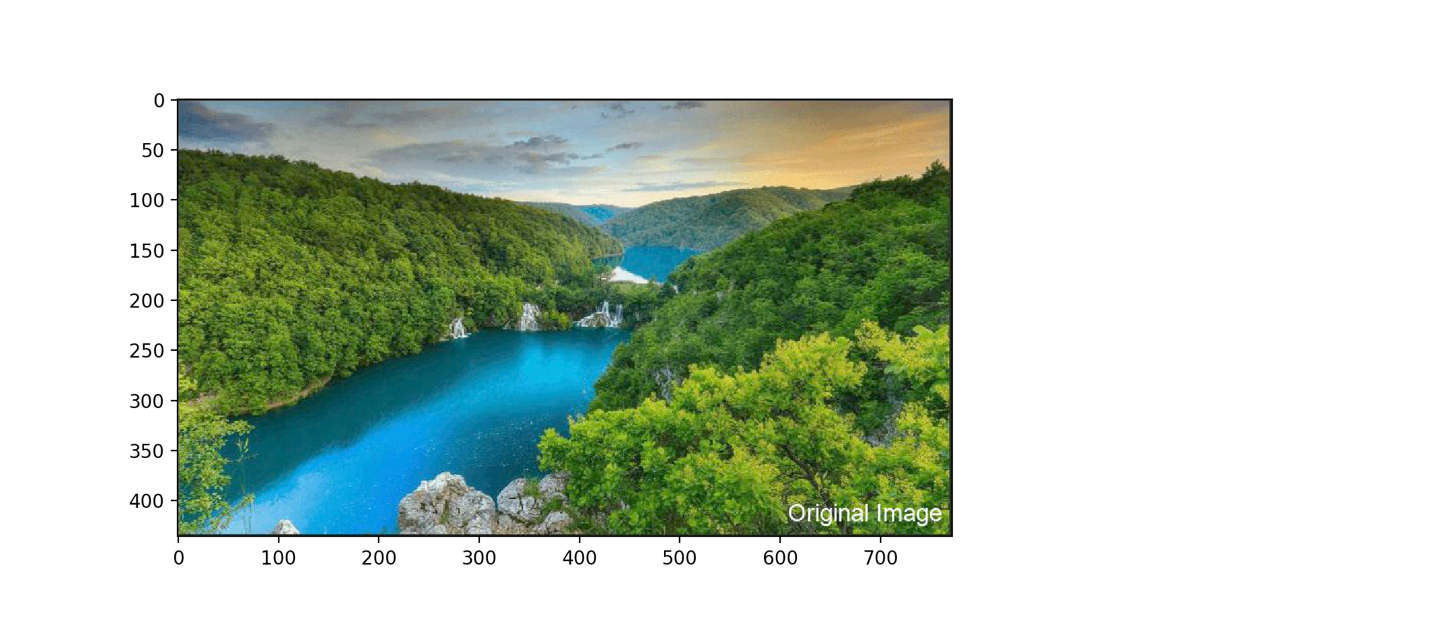
As seen above the user can select whether they want to segment their image using Otsu’s method or K-Means. Also the option to show the original image with a watermark if the user would like to do so. K-Means segmentation in this instance is colored and the value K determines how detailed the segmentation on the image is. It is intended such that the more detailed pictures require a higher K value. This K value can currently be changed only inside the code.

Final image of each segmentation function being used (Original, Otsu’s method, K-Means respectively, Original with watermark):









**Discussion and Conclusion**

This project lead me in many different directions and most of them were figuring out how to correctly write the kNN(K – nearest neighbor) function. Initially the kNN function that I wrote was based on a paragraph or article of text. This lead me to realize that kNN would be difficult to implement without some previous data to compare to. In the case of a single image there are only a few ways to implement kNN and make sure its accurate which include having previous information about the pixels in an image.

kNN requires some information(dataset) to start working correctly so I decided to shift to K-Means for image segmentation. K-Means is similar to kNN in that it clusters different types of pixels together to segment the original image but doesn’t require any additional information.

My K-Means implementation in this instance since it’s not the main focus of my project does use some libraries like cv2.kmeans. Additionally the Otsu’s method function in my program includes some code that I began to understand from a lot of research into how each step would work. On paper the mathematics for Otsu’s method was quite hard to follow so I really had to spend a week or so digging into how I could efficiently code Otsu’s method by almost segmenting each step myself.

The final issue I had was finding a good way to make a GUI using Jupyter Notebook. I know I could’ve found a more complicated way using strictly tkinter or some other library. Other libraries were difficult for me to implement and pulled me away from the focus of the project (Otsu’s method). So I decided to stick with a more simple light weight embedded gui using widgets. It may not be the prettiest but it allowed me to focus more on the implementation of each function.

**Project Timeline**

|  |  |
| --- | --- |
| Task  (Schedule your own tasks below. Adjust it as time goes if needed.) | Time  (Schedule your own time below. Adjust it as time goes if needed.) |
| Research Otsu’s method of thresholding and k-NN. Create project proposal | 1/26 ~ 2/5 |
| Prepare k-NN algorithm and begin coding using sample data | 2/6 ~ 2/13 |
| Make sure k-NN implementation is working correctly and then apply images | 2/14 ~ 2/28 |
| Implement method that takes an image input and outputs a segmented image | 3/1 ~ 3/14 |
| Make sure that each algorithm is working correctly based on Otsu’s method of thresholding | 3/15 ~ 3/29 |
| Write paper | 3/30 ~ 4/5 |
| Finalize | 4/6 ~ 4/18 |

**References**

[1] Hidayah, Maulidia & Akhlis, Isa & Sugiharti, Endang. (2017). Recognition Number of The Vehicle Plate Using Otsu Method and K-Nearest Neighbour Classification. Scientific Journal of Informatics. 4. 66. 10.15294/sji.v4i1.9503.

[2] Block diagram taken from: https://www.pantechsolutions.net/matlab-code-for-segmentation-of-image-using-otsu-thresholding