Code is referred from https://github.com/ashah01/Efficient-Graph-Based-Image-Segmentation

HOCHSCHULE SRH HEIDELBERG

SRH University Heidelberg

Task 2: Computer Vision

Graph based Image Segmentation

Name:	Shrey Kothavade
Matriculation Number:	11018044
Email Address:	11018044@stud.hochschule-heidelberg.de
Supervisor:	Prof. Dr.-Ing. Milan Gnjatovic
Project Begin Date:	14th Decmber 2022
Project End Date:	21st December 2022

Code is referred from https://github.com/ashah01/Efficient-Graph-Based-Image-Segmentation

Abstract:

Often based on the properties of the picture's pixels, image segmentation is a widely used method in digital image processing and analysis to divide an image into various portions or areas. A excellent picture segment is one that is simple to put together using its own components yet challenging to put together using pieces from other areas of the image. This non-parametric technique captures a wide variety of segment types without requiring neither prior training nor segment type pre-definition or modeling. There are mainly 5 types of image segmentation, in this project we are mainly focusing on Graph based Image segmentation.

Keywords: Graph based Image Segmentation, Image Segmentation, Edge detection, Segmentation, sorting.

I. Introduction:

A wide variety of practical computer vision applications, such as the identification of traffic signs, biology, the assessment of building materials, or video monitoring, all heavily rely on image segmentation. Additionally, driverless cars and Advanced Driver Assistance Systems (ADAS) must use pedestrian detection or identify navigable surfaces.

Image segmentation is a technique that divides a digital image into several subgroups known as Image segments, which serves to simplify future processing or analysis of the image by decreasing the complexity of the original image. In plain English, segmentation is the process of giving pixels labels 1.

The threshold method is the most straightforward approach to segmentation in image processing. By comparing each pixel's intensity with a predetermined value, it separates the pixels in a picture (threshold). When the desired object is more intense than the backdrop, it is helpful (unnecessary parts).

II. Graph Based Segmentation Algorithm:

Definitions of "image area" are necessary for segmentation algorithms (typically depending on the application). High contrast to neighbouring pixels and "homogeneity" are factors we deem to be fair. Distances between (vector-valued) pixels can be used to calculate homogeneity; we find that L2 norms perform better than L1 or pseudo-norms in this regard. It is recommends segmenting the R/G/B component photos separately and combining the findings.

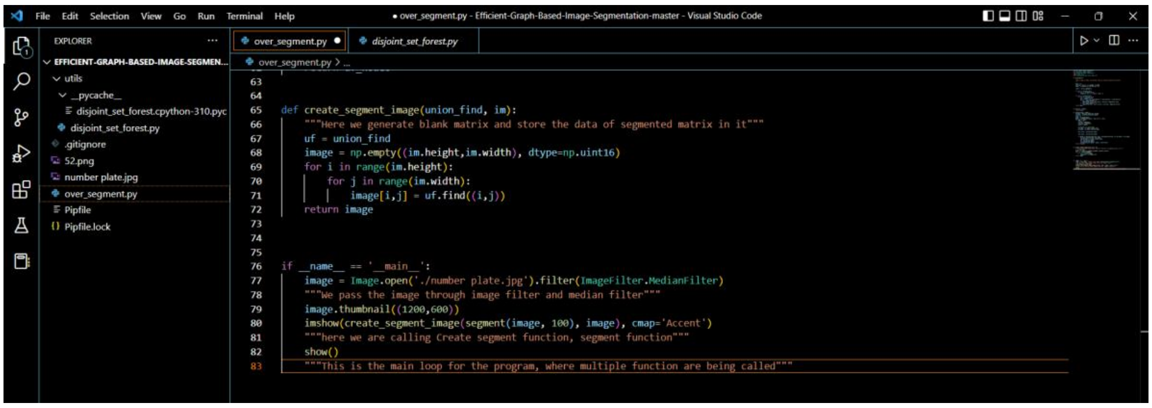
Since object edges are not always visible in all multi-spectral bands, segmenting once using all bands is safer (and undoubtedly quicker). The aforementioned homogeneity measure establishes the weight of edges, keeping in mind the framework for segmenting graphs. How an online graph-cutting heuristic should divide the MST based on edge weight is yet unknown. Because it doesn't take into consideration noise or the general homogeneity of a region, a threshold alone is inadequate. [1]

Code is referred from https://github.com/ashah01/Efficient-Graph-Based-Image-Segmentation

In graph based image segmentation, we followed the following steps: First we took an input image. As the input image was taken, this image is first reduced and converted to gray scale. The data of each pixel of the image is then stored in a 2d array. All the pixel values of the image are assigned as node. Then algorithm for difference between the neighbouring pixel values is calculated and a new table is formed. This table is then sorted in ascending order based on difference between the nodes. A threshold value is declared in the code and below this threshold value all the nodes, i.e., pixels are merged. These merged pixels form a segment, which is then denoted by a colour and process of graph based image segmentation is completed.

III. Working of code

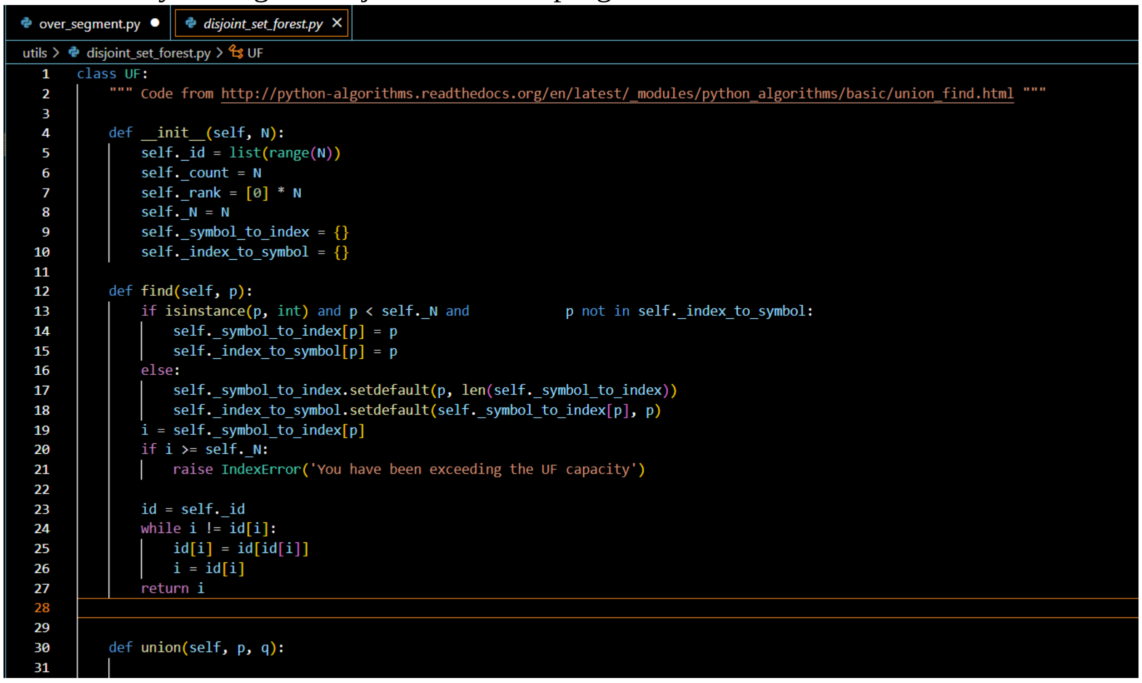
The code starts with taking the input of the image and passing it through the image filter and median filter of python. The size of the image is defined. A nested loop of functions is then initialized in the code.


File Edit Selection View Go Run Terminal Help
● over_segment.py - Efficient-Graph-Based-Image-Segmentation-master - Visual Studio Code
EXPLORER
EFFICIENT-GRAPH-BASED-IMAGE-SEGMEN...
utils
__pycache__
disjoint_set_forest.cpython-310.pyc
disjoint_set_forest.py
.gitignore
52.png
number plate.jpg
over_segment.py
Pipfile
Pipfile.lock
over_segment.py ●
disjoint_set_forest.py
over_segment.py > ...

```
def create_segment_image(union_find, im):
    """Here we generate blank matrix and store the data of segmented matrix in it"""
    uf = union_find
    image = np.empty((im.height,im.width), dtype=np.uint16)
    for i in range(im.height):
        for j in range(im.width):
            image[i,j] = uf.find((i,j))
    return image


if __name__ == '__main__':
    image = Image.open('./number plate.jpg').filter(ImageFilter.MedianFilter)
    """We pass the image through image filter and median filter"""
    image.thumbnail((1200,600))
    imshow(create_segment_image(segment(image, 100), image), cmap='Accent')
    """here we are calling Create segment function, segment function"""
    show()
    """This is the main loop for the program, where multiple function are being called"""
```

Fig1:Code1

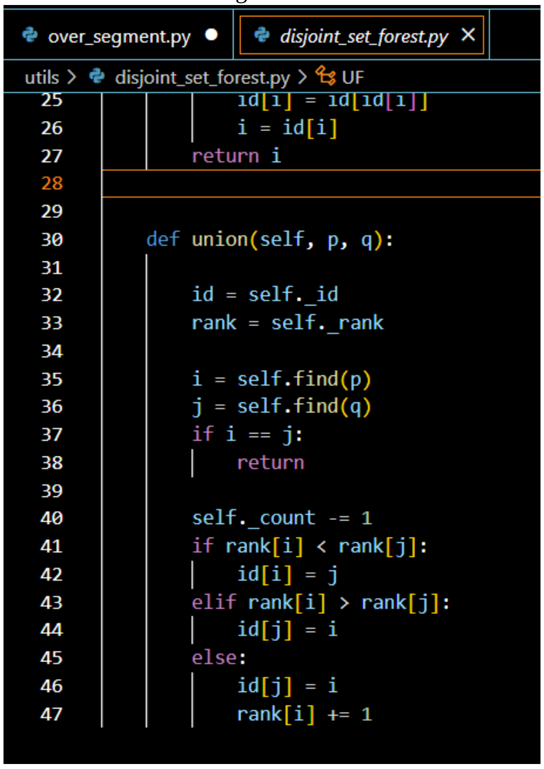
In this, first Segment function is called. In Segment function, we first determine the size of the matrix by calling the disjoint set forest program.



```
class UF:
    """ Code from http://python-algorithms.readthedocs.org/en/latest/_modules/python_algorithms/basic/union_find.html """

    def __init__(self, N):
        self._id = list(range(N))
        self._count = N
        self._rank = [0] * N
        self._N = N
        self._symbol_to_index = {}
        self._index_to_symbol = {}

    def find(self, p):
        if isinstance(p, int) and p < self._N and            p not in self._index_to_symbol:
            self._symbol_to_index[p] = p
            self._index_to_symbol[p] = p
        else:
            self._symbol_to_index.setdefault(p, len(self._symbol_to_index))
            self._index_to_symbol.setdefault(self._symbol_to_index[p], p)
        i = self._symbol_to_index[p]
        if i >= self._N:
            raise IndexError('You have been exceeding the UF capacity')

        id = self._id
        while i != id[i]:
            id[i] = id[id[i]]
            i = id[i]
        return i


    def union(self, p, q):
```

Code is referred from https://github.com/ashah01/Efficient-Graph-Based-Image-Segmentation

Fig2:Code2



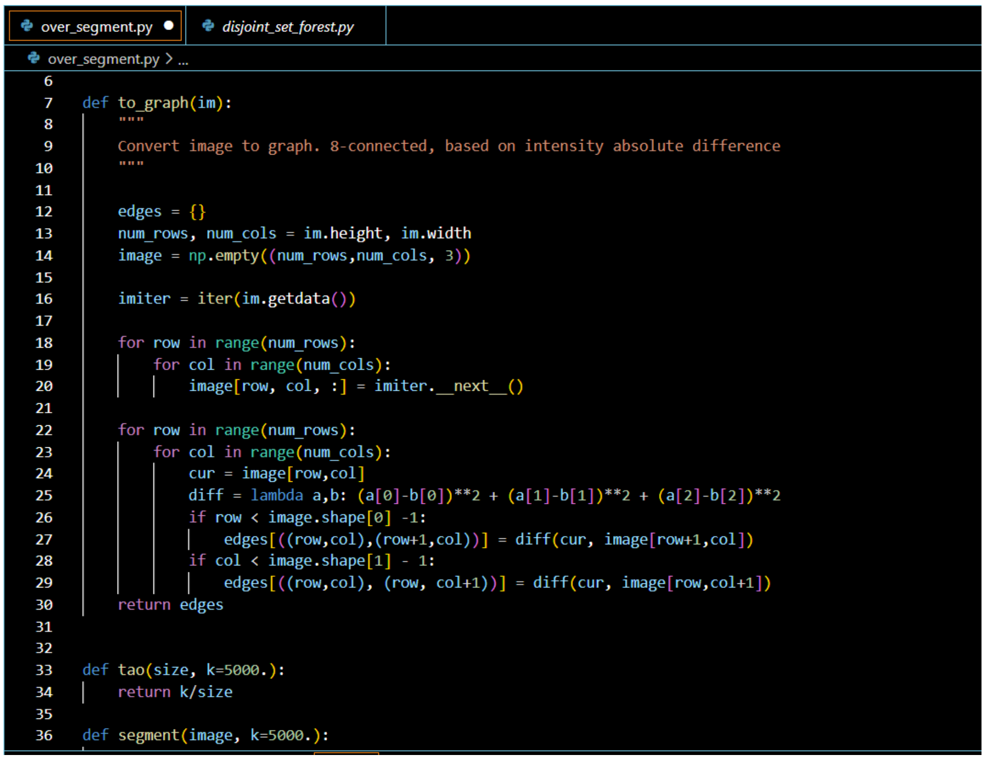
over_segment.py ● | disjoint_set_forest.py ×

utils > disjoint_set_forest.py > UF

```python
            id[i] = id[id[i]]
            i = id[i]
        return i


    def union(self, p, q):

        id = self._id
        rank = self._rank

        i = self.find(p)
        j = self.find(q)
        if i == j:
            return

        self._count -= 1
        if rank[i] < rank[j]:
            id[i] = j
        elif rank[i] > rank[j]:
            id[j] = i
        else:
            id[j] = i
            rank[i] += 1
```

Fig3: Code3

This data is later processed to generate the edge detection in graph function. In graph function, a null matrix is generated and the difference between the consecutive nodes is calculated and then this data is sent back to segment function.



over_segment.py ● | disjoint_set_forest.py

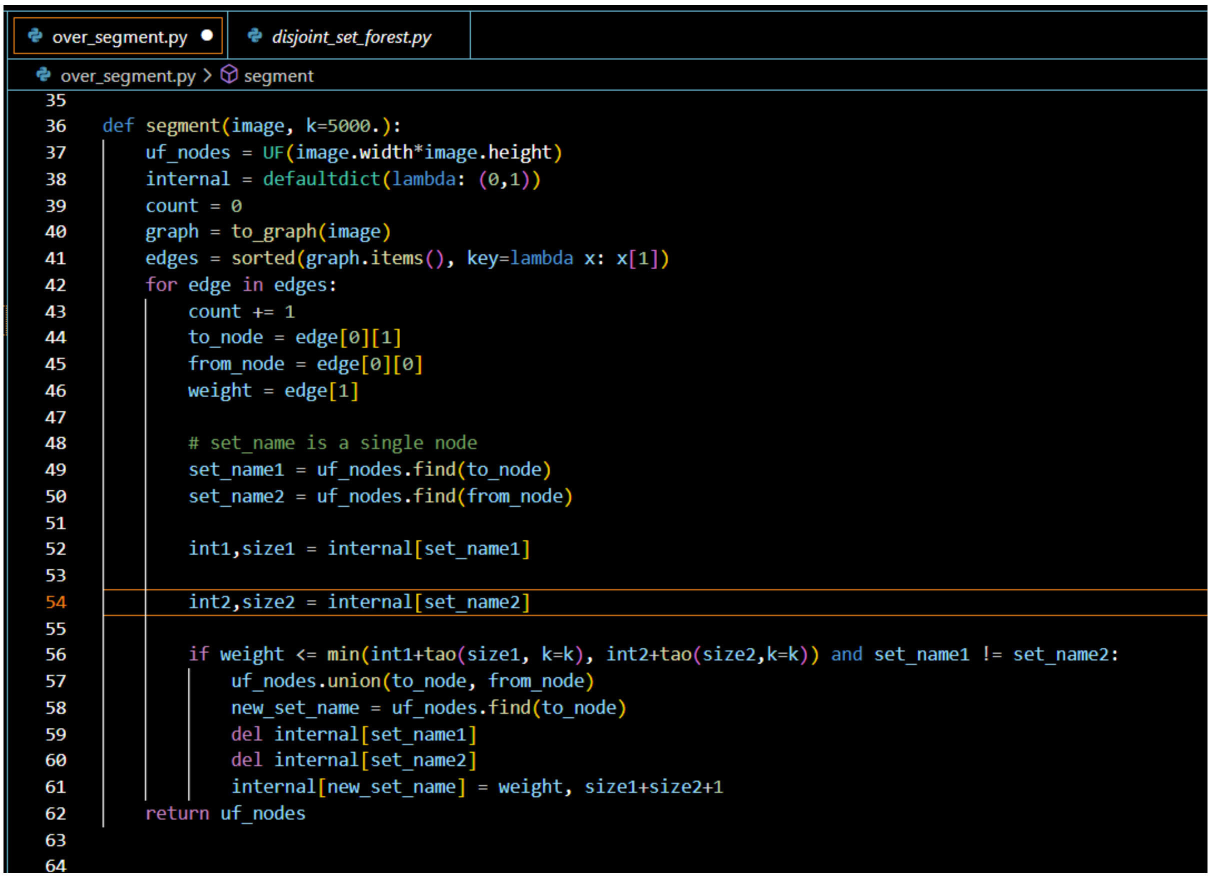
over_segment.py > ...

```python

def to_graph(im):
    """
    Convert image to graph. 8-connected, based on intensity absolute difference
    """

    edges = {}
    num_rows, num_cols = im.height, im.width
    image = np.empty((num_rows,num_cols, 3))

    imiter = iter(im.getdata())

    for row in range(num_rows):
        for col in range(num_cols):
            image[row, col, :] = imiter.__next__()

    for row in range(num_rows):
        for col in range(num_cols):
            cur = image[row,col]
            diff = lambda a,b: (a[0]-b[0])**2 + (a[1]-b[1])**2 + (a[2]-b[2])**2
            if row < image.shape[0] -1:
                edges[((row,col),(row+1,col))] = diff(cur, image[row+1,col])
            if col < image.shape[1] - 1:
                edges[((row,col), (row, col+1))] = diff(cur, image[row,col+1])
    return edges


def tao(size, k=5000.):
    return k/size

def segment(image, k=5000.):
```

Fig4: Code 4

This data is then sorted in ascending order and stored in weight matrix. Now according to the threshold the nodes are ordered and segments are generated.

Code is referred from https://github.com/ashah01/Efficient-Graph-Based-Image-Segmentation



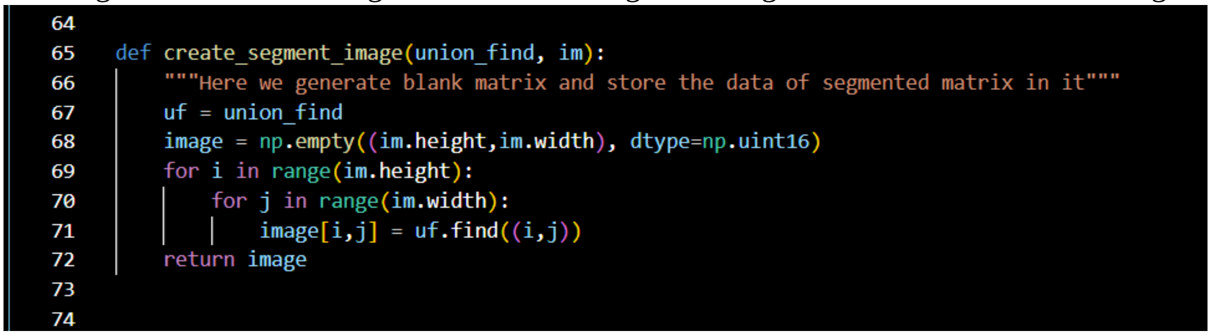
over_segment.py ● | disjoint_set_forest.py

over_segment.py > segment

```python

def segment(image, k=5000.):
    uf_nodes = UF(image.width*image.height)
    internal = defaultdict(lambda: (0,1))
    count = 0
    graph = to_graph(image)
    edges = sorted(graph.items(), key=lambda x: x[1])
    for edge in edges:
        count += 1
        to_node = edge[0][1]
        from_node = edge[0][0]
        weight = edge[1]

        # set_name is a single node
        set_name1 = uf_nodes.find(to_node)
        set_name2 = uf_nodes.find(from_node)

        int1,size1 = internal[set_name1]

        int2,size2 = internal[set_name2]

        if weight <= min(int1+tao(size1, k=k), int2+tao(size2,k=k)) and set_name1 != set_name2:
            uf_nodes.union(to_node, from_node)
            new_set_name = uf_nodes.find(to_node)
            del internal[set_name1]
            del internal[set_name2]
            internal[new_set_name] = weight, size1+size2+1
    return uf_nodes


```

Fig5: Code5

This segmented data is then goes to the create segment image function to recreate the image.



```python

def create_segment_image(union_find, im):
    """Here we generate blank matrix and store the data of segmented matrix in it"""
    uf = union_find
    image = np.empty((im.height,im.width), dtype=np.uint16)
    for i in range(im.height):
        for j in range(im.width):
            image[i,j] = uf.find((i,j))
    return image


```

Fig6: Code 6

Using the Matplotlib libraries imshow() function we add colors to this image. This image is now displayed using a show command.

Code is referred from https://github.com/ashah01/Efficient-Graph-Based-Image-Segmentation

IV. Results:

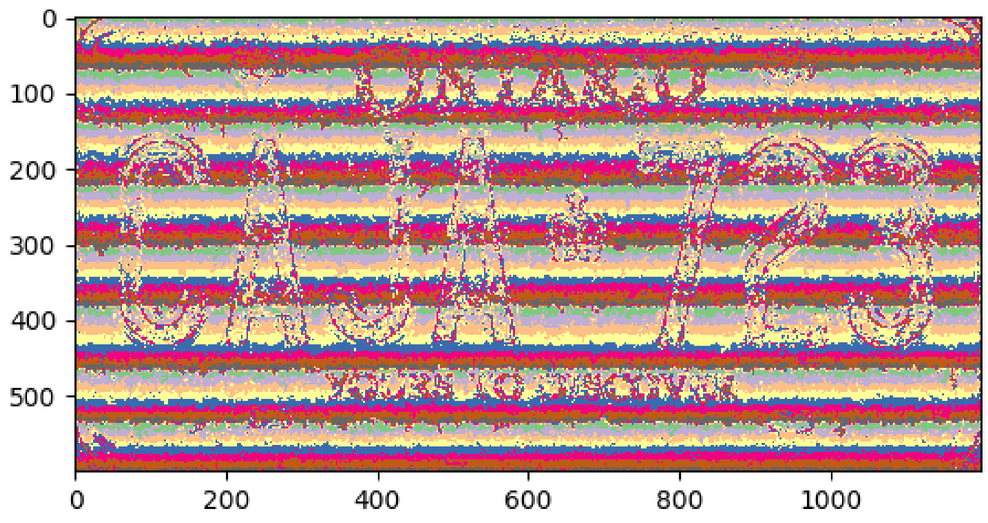


Fig7: Output at Threshold 10

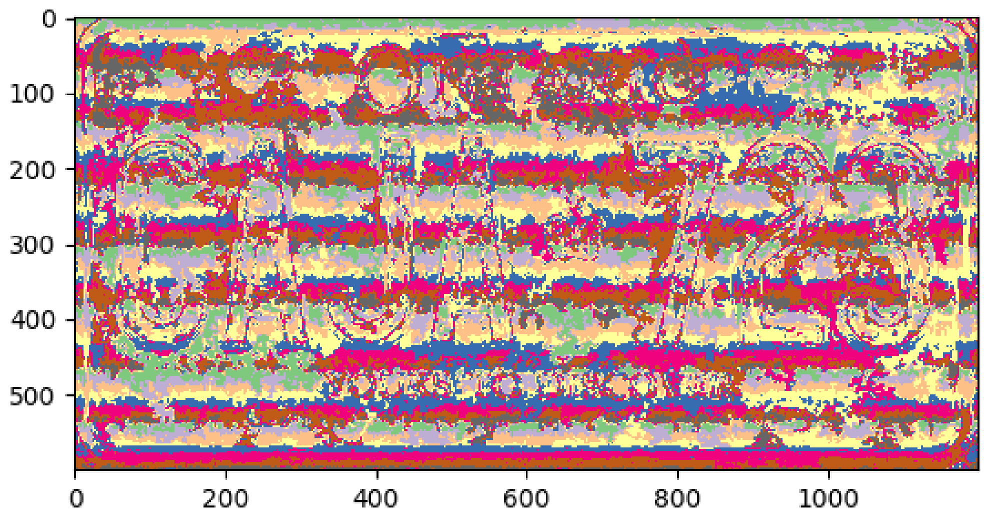


Fig8: Output at Threshold 100

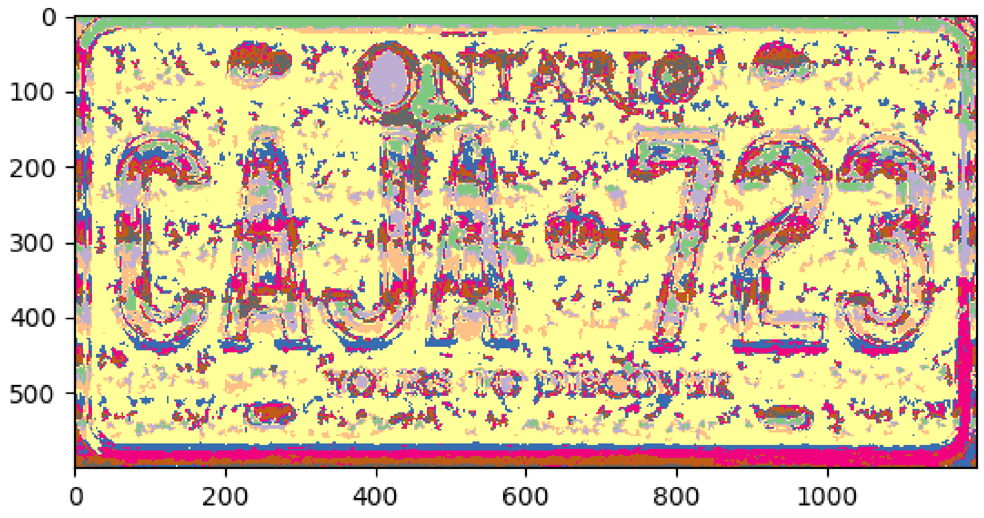


Fig9: Output at Threshold 245

Code is referred from https://github.com/ashah01/Efficient-Graph-Based-Image-Segmentation

V. Approach and challenges faced in the project

After understanding the concept, I was able to write code in python from scratch till Gray image conversion. After that, I was not able to develop the logic of edge detection algorithm. As the logic was taught in the class, we were able to create for loops for the same but the desired output was not there. I tried referring to the codes given online, but most of the codes were developed with specific libraries.

After again asking professor, the code was clear, but as I was coding in python, it was difficult to convert the code in python. So, I shifted my work on processing tool. Yet I got stuck after edge detection algorithm.

VI. References

[1]J. Wassenberg, W. Middelmann, and P. Sanders, "An Efficient Parallel Algorithm for Graph-Based Image Segmentation," Lecture Notes in Computer Science, pp. 1003–1010, 2009, doi: 10.1007/978-3-642-03767-2_122.

[2] https://github.com/ashah01/Efficient-Graph-Based-Image-Segmentation

[3] https://www.youtube.com/watch?v=2IVAznQwdS4&t=6s

[4] https://github.com/luisgabriel/image-segmentation

[5] https://github.com/salaee/pegbis

[6] https://github.com/sirius-mhlee/graph-based-image-segmentation

[7] https://github.com/yubajin/Graph-BasedImageSegmentation

[8] https://github.com/Jia-Xueyu/Graph_based-Image-Segmentation