COMP4102
Project Final Write Up

Paint Sheet by Number

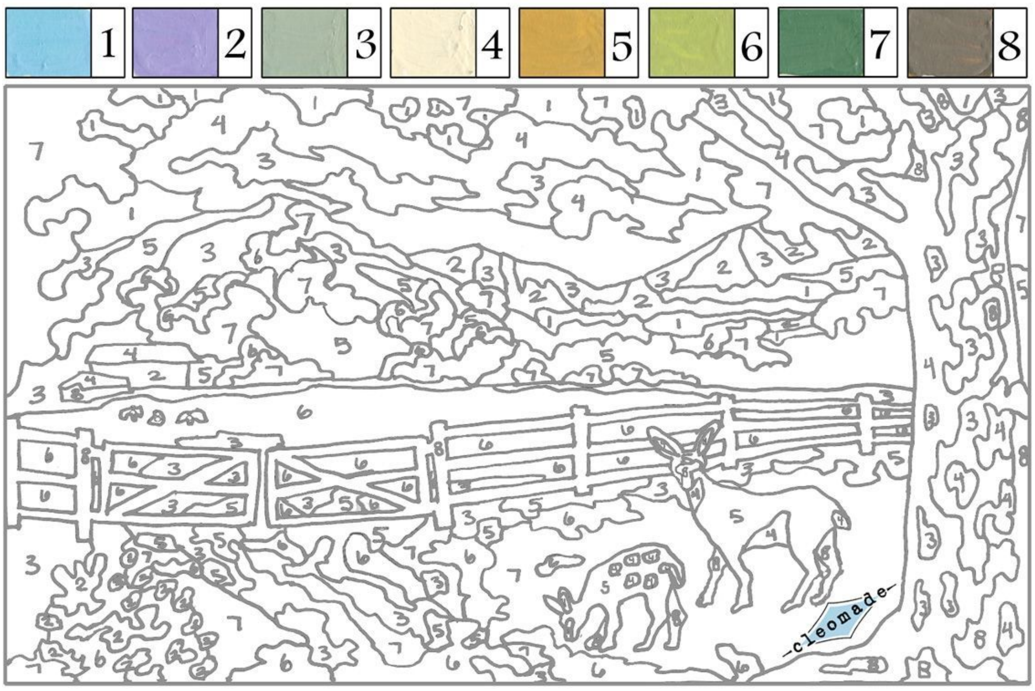
April 18, 2020

Team Members

Cameron Canning-Potter 100977152
David Jatczak 100879164
Pravdeep Deol 101007603

Abstract

A paint by number kit is a fun activity for people of any age where a person would paint each numbered region as a specific colour. After every area is painted over, the result would be a beautiful work of art that looks like the person who painted it actually painted it from scratch from themself. Typically a paint by number kit is a paint sheet with numbered regions that are blank/white with black edges. Regions are referred to as segments in our project. A region would have a single number indicating what colour the region should be; the colour to be used would be specified by a legend, typically on the side of the picture that the person is colouring in, and would have a colour associated with each number. Below is an example paint sheet for a paint by number kit. Our application takes in a coloured input image (.e.g. jpg files, png files, etc) and generates a paint sheet.



(source: https://www.pinterest.ca/pin/295267319295769934/)

Introduction

The aim of this project was to design and implement an application that converts any given coloured image into a paint sheet. The process begins with the original image being quantized using k-means clustering. Each cluster is segmented into its connected components. Using this data the app makes use of a combination of edge detection and dilation to create a paint sheet. After this step each segment is labeled with a number to represent the unique colour for that specific segment. Then it provides a legend of these colours, mapping them to their corresponding numbers. At the end the user is given an image that looks like a paint sheet with accurate colouring information for the image.

This application is relevant to computer vision because the core functionality that we implemented can be used to take any real world image and create an accurate colour mapping.The application demonstrates this when it makes use of “k-means” clustering to detect a set of colours within an image and to define different segments. Someone could take this data and build upon it to be used in future applications where searching for and finding objects is relevant.

The primary challenge of this project was the implementation of a segmentation algorithm that could segment any image. This was a challenge because the k-value used in the k-means clustering algorithm directly affects how noisy the resulting image looks. This posed a problem because if a too small of a k-value was chosen then you would miss potential segments of different colours. However, choosing too large of a k-value meant that the image generated was too noisy and was not a good candidate image as a paint sheet. Towards the end of the project we felt that giving the user control over the k-value during runtime would allow the user to find the best k-value by trial and error.

Background

The four main image segmentation techniques used in practice are region-based, edge detection, clustering, and mask R-CNN [3]. In our testing we found region based and edge detection required an image to have high contrast between its segments and failed in many of our test cases. While R-CNN's are currently world class image segmenters they come with limitations. R-CNN's need to be trained on images with known object labels [3]. For example finding cars, streetlights, and signs in a picture of an intersection. This kind of segmentation would ignore segments that do not have a class, like the sky, the road and trees. We want our project to work on almost every image so this specificity is a detriment. Not to mention the labeled datasets we would need to obtain, the long training times and long prediction time [2]. For these reasons we have decided to use clustering for our image colour quantization.

Clustering attempts to group alike data in such a way that objects in the same clusters are similar [1]. This alike data can then be further processed into its connected components solving the problem of image segmentation. We choose k-means as it offers flexibility in the number of clusters and the fastest run time [1]. Unlike R-CNN's clustering needs to know nothing about what the data is modeling, which means it will work on any image. While the majority of clustering algorithms determine the number of clusters by minimising some cost function we found this unfavourable. We did some experiments in calculating the number of clusters utilizing elbow method on k-means. Our implementation of the elbow method runs k-means on values from 2 through 10 and calculates the mean squared error for each k [4]. Given a simple input (Figure 1), the elbow method (Figure 2) would say 4 is the best number of clusters. But it would not be unreasonable to assume the user would want 3 clusters, e.g. green, white, and brown. For this reason we decided to let k be a hyper parameter adjustable by the user during run time. While the majority of clustering algorithms run in O(n^2), k-means runs in O(n), n being the number of pixels in the image [1]. These reasons led our group to choose k-means as our clustering algorithm.

Figure 1

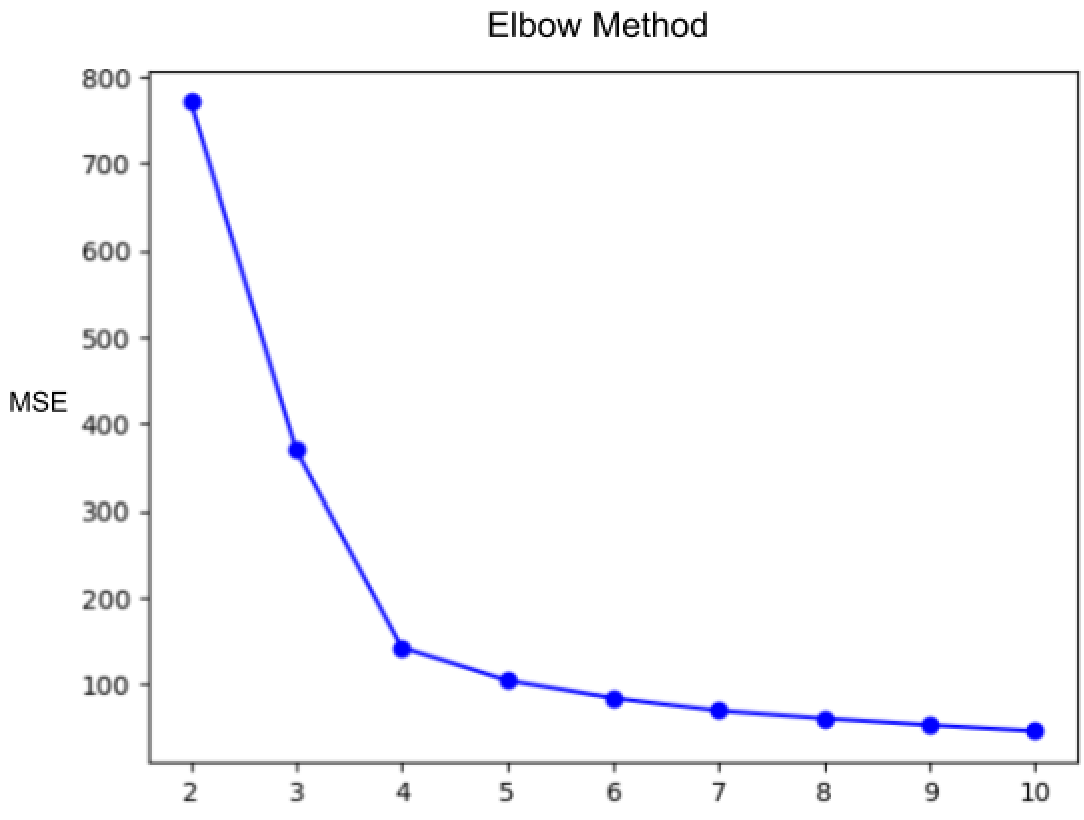


Figure 2

Approach

The Application is divided into five separate classes. PaintSheet contains Segmenter, a collection of Segment’s, and other variables & functions. This was the main logic of calculating a paint sheet where it would contain the original image, the quantized image by k-means, and the final product image. Window class is an encapsulation of PaintSheet and various other settings for the user interface. It would handle the logic of displaying, key presses (shortcuts), and telling PaintSheet when to recalculate when given new options like dilation or a k-value for k-means. The Legend class handles two main operations. These include the creation of the legend that contains a mapping of colours to numbers, and drawing numbers that match the legend and that fit within the respective segment. Finally Border class handles edge generation of Segments and Miscellaneous functions are held within a helper header.

Our application starts with segmentation. The Segmenter takes as input a coloured image and a k value. The image is preprocessed into a vector of colour values. The k-means algorithm labels each pixel corresponding to a center (in our case a colour). The pixel positions and quantized colour value are stored in what we call a Segment. Since it is possible for a Segment to have multiple connected components initially, e.g. a black and white checkerboard, each Segment is split into multiple Segments corresponding to each of its connected components. Since a Segment is just a collection of points and a colour value, it needs to be rebuilt into a Mat when needed. This is done with asBinaryMat(...) and asMat(...). Additionally the Segmenter contains public functions that are used by the rest of the application. Segmenter stores all the generated segments and as such can provide all k of the segment colours, a generated quantized image, and the entirety of the segment data.

The edges, called Borders in our project, are obtained within the Borders class which utilizes an input image’s dimensions and the segments generated off that image. An edge for a single segment is calculated upon the binary Mat provided by Segment::asBinaryMat(..) with a Gaussian blur then a canny edge detector. A loop over all the segments is utilized for constructing the final edge Mat. A temporary Mat variable for the summation in the loop is utilized, i.e. a black (blank) image with the same dimension as the original image. All these edge Mats are summed onto the temporary Mat variable thus end up with the result of a black & white image containing just the edges of the segments. This is done by a static method in

Borders class, Borders::create(..) which utilizes Borders::getEdges(..) on each segment.

Within the context of our project we also wanted a way to modify edge detection to create more complete regions and also gain a paint sheet aesthetic. For both of these problems the solution was dilation. Dilation is the process of increasing the thickness of lines in an edge detection image. Not only did this operation create our desired paint sheet aesthetic, it also helped in extending edges to create more complete regions. However, with the introduction of the k-means clustering algorithm to our program the use of dilation had become purely aesthetic. As such the dilation function is one of the last steps before creating the legend. After testing the application we decided to implement an option for the user to not include dilation in the final result because of unwanted interactions between dilation and k-means clustering with high k-values.

For the number placement a rather slow and brute force method was used where each segment would be scanned for a square shaped area that the number can fit in, meaning the number is not on an edge and is entirely within a segment of its own colour. In other words every pixel in the square shaped area is entirely one colour. The side length of this square is the font size. This is done for each and every segment. If the number cannot find a place to fit in then the font size for the number is adjusted to half its size and reduces the square shaped search area as well. The algorithm described in the previous paragraph would be done again. Each time a place is not found then the size gets halved each iteration until the font size is less than 15% of the original font size otherwise no number will be drawn. This gives varying sizes of numbers on the paint sheet with some segments not having a number.

The legend is created by looping over every element in the list of segments to get each segment's colour. This colour is taken and used to create a rectangle to the right of the image, that is filled with that segment's colour. Then a number is drawn next to this rectangle to indicate that this number corresponds to it. The number is determined with the index ‘i’ used to loop over the list of segments.

The legend is drawn to the right of the result image. This means that the image has to be extended; but a hard coded number of pixels does not work, because if the input image is too small then the resulting legend would be cut off. To prevent this issue from occurring the entries in the legend will expand vertically downwards until it reaches the end of the image, where it will begin a new column. This method of handling the issue will prevent an image of small size from cutting off the legend information.

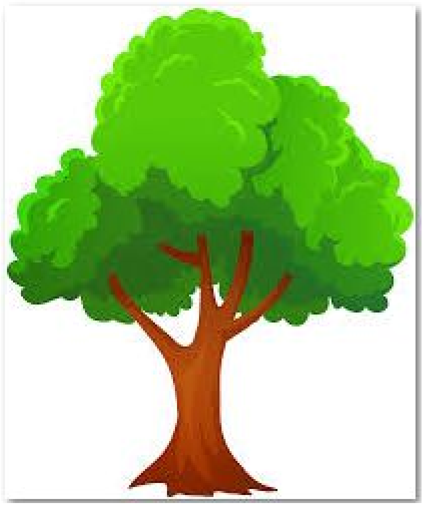
The OpenCV createTrackbar(..) functionality was used for letting the user decide what k-value to give to the k-means algorithm. Unfortunately in an earlier prototype it would cause PaintSheet to constantly recalculate upon dragging the slider. It is now robust enough to use without lag from calculations and also limits the user to proper k-values.

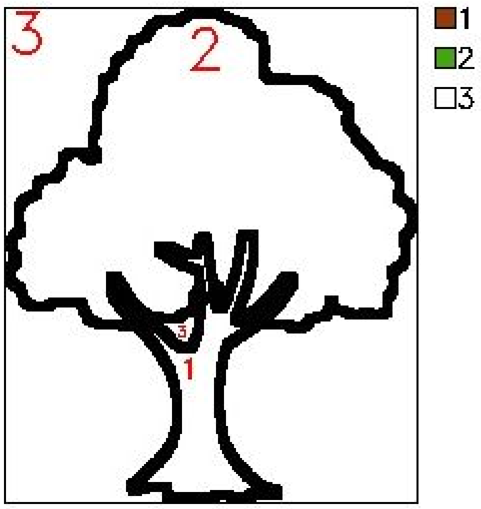
The user interacts with our application via keyboard key presses to enable/disable dilation, save the image, and compare the quantized (coloured in) image to the original image. This is done within the Window class. Keyboard key presses are an easier alternative to implementing an actual UI with a bloated library like Qt since it is easier for users to pick up our project and compile due to less dependencies in our project.

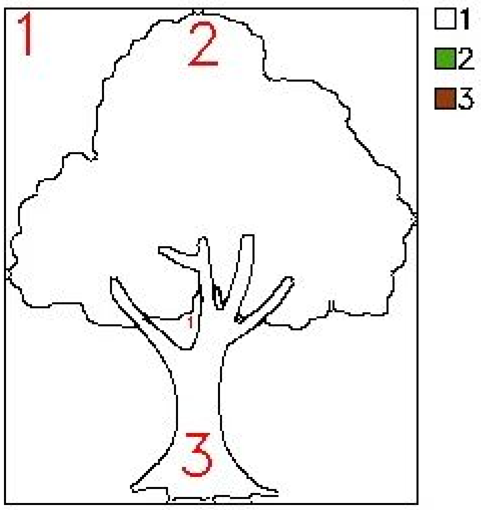
Results

The end result of this project is a successful paint by number application that will take in any image as input and output a paint sheet. We defined success as generating a recognisable paint sheet with almost every segment containing a number. We achieved this in most of our test cases. However, during the development and testing of our project some failure cases were discovered. The primary case involved the choice of an optimal k-value. For example, choosing a small k-value can result in the entire image being divided into two or three segments. This results in an image that does not accurately represent the original image and not very interesting to use as a paint by number page. However, choosing a larger k-value resulted in more detailed results. This contributed to the problem of the resulting image being too noisy and is compounded with the use of dilation which thickens the edges and makes small segments even smaller. Another potential failure case is that depending on the noise of the original image and higher k-values, smaller segments do not allow for enough space to display a number labeling of the segment.

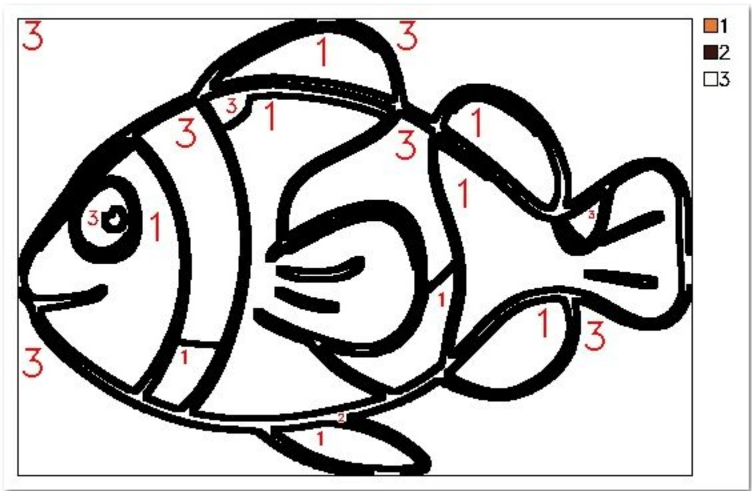
Test Case #1

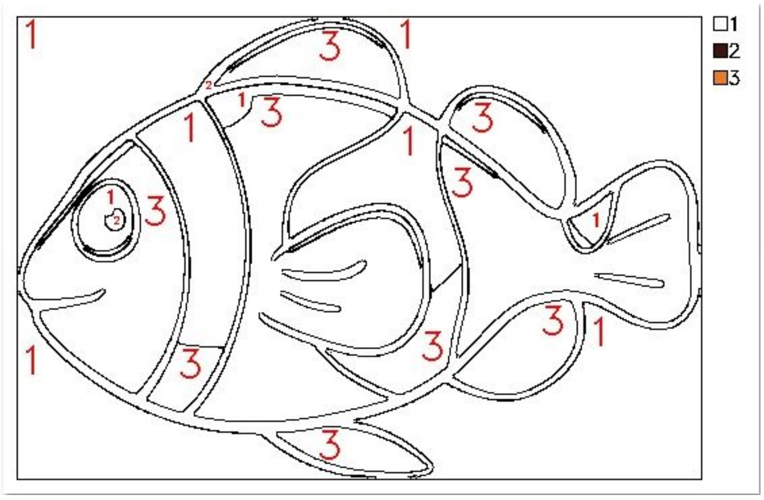




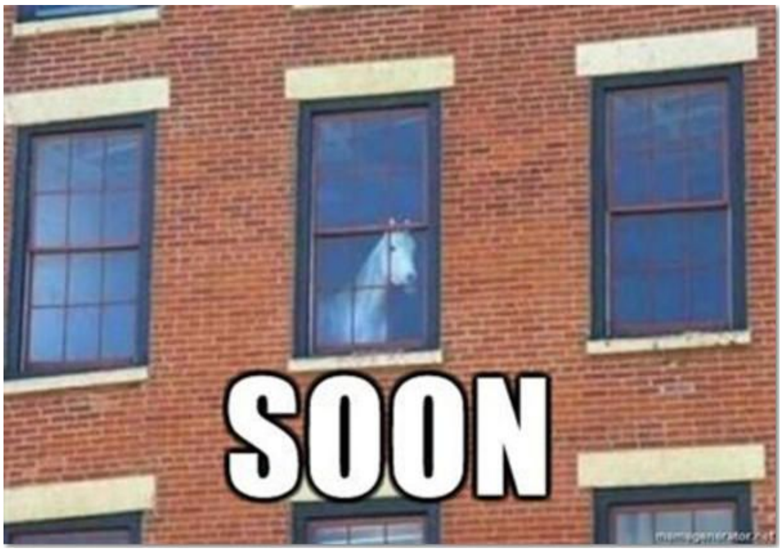


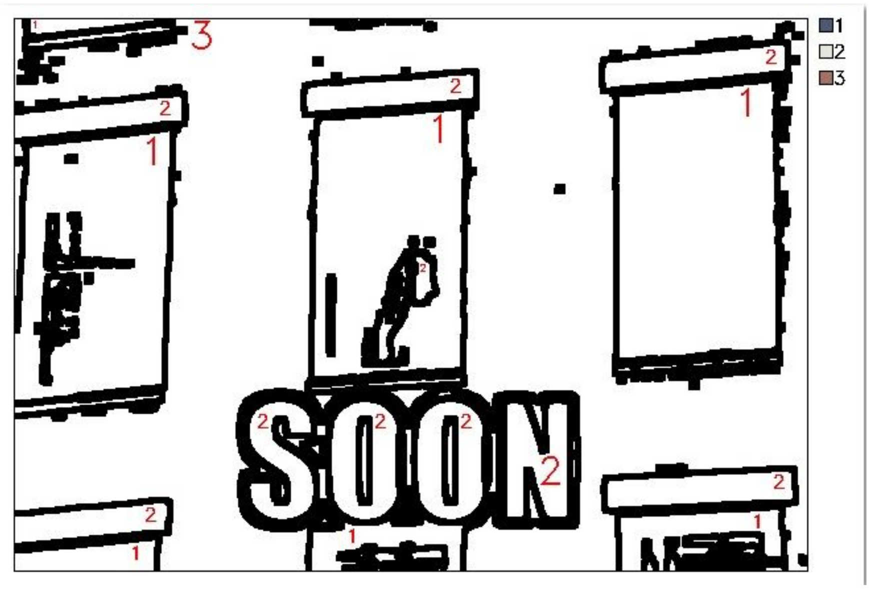
Test Case #2

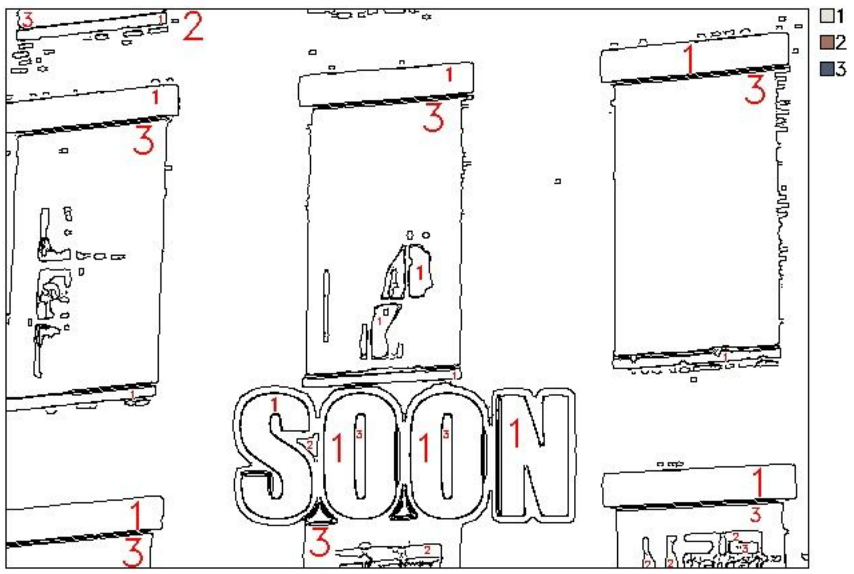




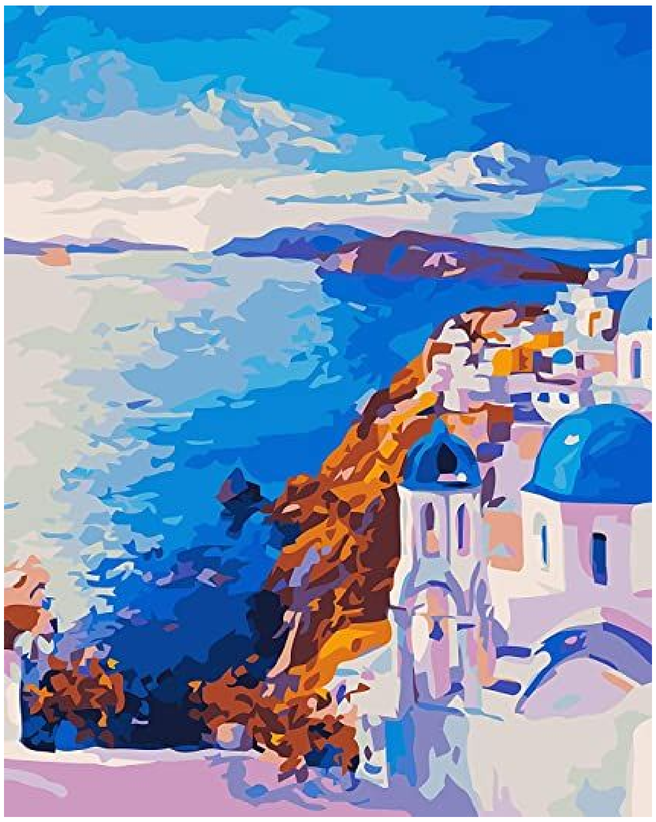
Test Case #3

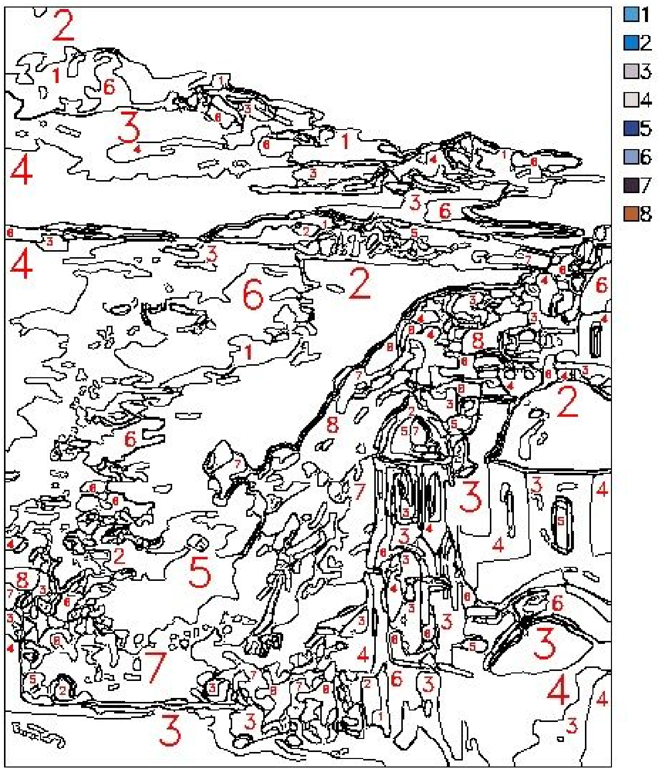






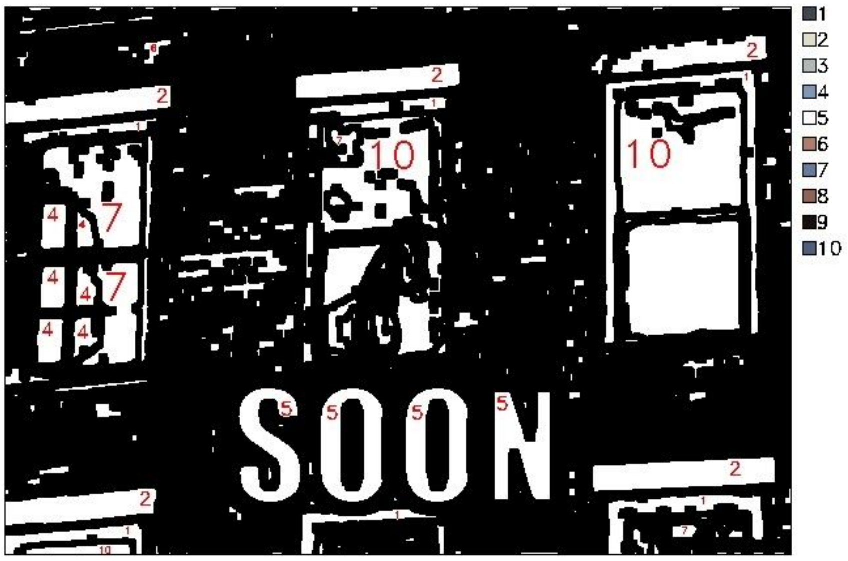
Test Case #4



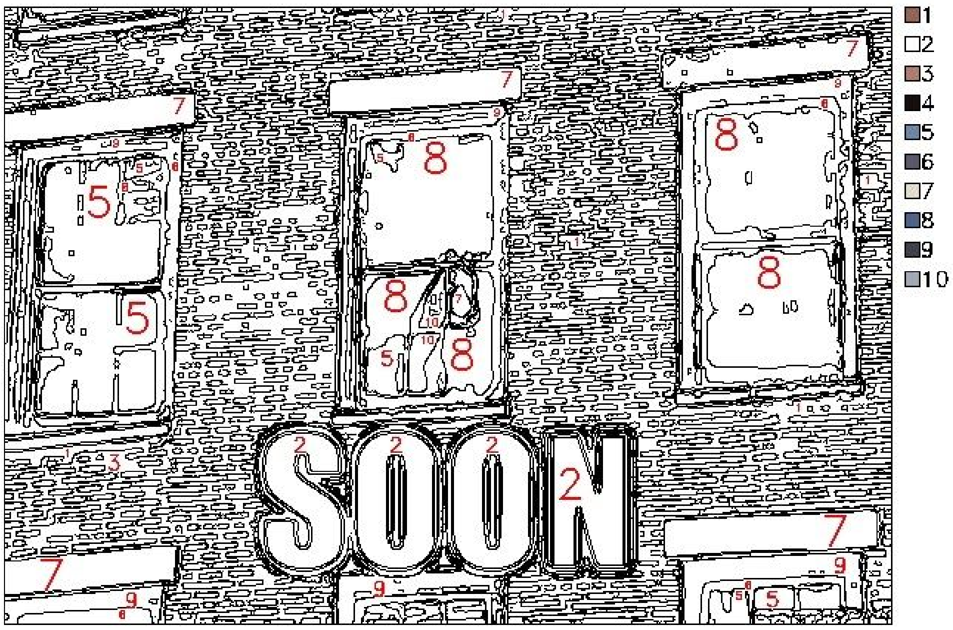


Failure Cases

High k-value with dilation, dilated edges completely covering other adjacent small segments



High k-value without dilation, brick segments too tiny for being numbered



List of Work

David

- Everything in Borders class
- Combined and cleaned up people's code frequently
- Organized everyone's code into PaintSheet class, Window class, Legend class
- PaintSheet class
    - Encapsulated everyone's code
- Window class
    - Created it
    - Display logic
    - Trackbar slider for k-values
    - Qt buttons for clicking (scrapped)
- findContour(..) (scrapped)
- Legend class (contributed)
    - Encapsulated Pravdeep's code
    - Implemented number placement algorithm
        - Dynamic sizing of numbers
        - Colour checking
- CMake configuration and tutorial for compilation
- Writeup
    - Abstract
    - Some of approach
    - References
    - Collaborated and peer reviewed with team
- Licensed software under GPLv3

Cam

- Elbow method analysis script
- Segmenter class
    - k-means
    - Connected components
- Window (contributed)
    - Keystroke interface
- Overall application design
- Writeup
    - Background
    - Approach (contributed)

Pravdeep

- Add Dilation to edge detector
- Create legend from colour data
- Enable legend to create new columns when lots of colours

- Drawing numberings at centers of segments/contours
- Sizing the image properly
- Write up
    - Introduction
    - Some of approach
    - results

GitHub Page

https://github.com/Davidj361/PaintSheetGenerator

References

[1] Clustering: A Comprehensive Survey of Clustering Algorithms
Dongkuan Xu & Yingjie Tian
https://link.springer.com/article/10.1007/s40745-015-0040-1

[2] R-CNN: Rich feature hierarchies for accurate object detection and semantic segmentation
Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik
https://arxiv.org/abs/1311.2524

[3] Various Image Segmentation Techniques: A Review
Dilpreet Kaur , Yadwinder Kaur
https://ijcsmc.com/docs/papers/May2014/V3I5201499a84.pdf

[4] Finding Best Possible Number of Clusters using K-Means Algorithm
K. Maheswari
https://www.ijeat.org/wp-content/uploads/papers/v9i1s4/A11191291S419.pdf