

CS221 Project Final Report

Automated Selfie Stick Remover

Yiju Hou, Wendi Liu, Qingyang Xu

1. Task Definition

1.1 Motivation

Nowadays, when people take pictures with selfie sticks, the long selfie stick often occupies a conspicuous portion of the image and makes the photo awkward looking. Hence many people want an image processing software that can automatically remove the selfie stick from the picture and leave behind minimum traces. On a related note, one of the reasons why lightweight indoor drones are gaining popularity rapidly is that they can take pictures from bird-eye view without blocking any space. In this project, we implement such an algorithm which automatically identifies and removes the selfie stick from photographs and fills in the removed part with the natural extension of the remaining background.

1.2 Input and Output

Our input is the photograph with a selfie stick and the two ends of the selfie stick clearly labeled with red circles. We ask the user to label the approximate positions of the two ends of the selfie stick in order to reduce our working space into a smaller region containing the selfie stick. It is also a technique used by many photo editing softwares such as Photoshop. Our output is the original image with the selfie stick cleanly removed and replaced by new pixels that blend naturally into the background, which should look like a photo taken by an indoor drone.

1.3 Evaluation Metrics

Since the perception of the performance of the selfie stick remover can be rather subjective, simply evaluating the percentage of the selfie-stick pixels the algorithm has removed and replaced is not the most effective approach (see more details in the error analysis section). We decided to visualize the segmentation result as an evaluation of the effectiveness of our algorithm. We compared the performance using K-Means and Hierarchical Agglomerative Clustering with different features and present the result in the error analysis section. Based on the segmentation result, we categorize our data and analyze what kind of images do our algorithm perform best on, and in which cases the selfie sticks are not effectively removed.

2. Infrastructure

Our training samples consist of over one hundred images taken with selfie sticks from Google as well as photos with hashtag “selfie stick” from Instagram. After collecting these images, we manually labeled both endpoints of the selfie stick in each picture with two small red circles with RGB values (255, 39, 0).

Since segmenting the selfie stick from a given picture essentially amounts to extracting the pixels belonging to the selfie stick from all other pixels, we first transform all the pixel RGB values into LAB color space. We chose to use LAB color space because it is very close to human perception, which means that the distance calculated using LAB values has relatively high correlation with the color similarity perceived by human eyes.

Since clustering on the full image is both highly inefficient and likely to yield errors, we first need to extract the much smaller patch of the original picture which contains the selfie stick by identifying both the endpoints of the selfie stick labeled by the user with red circles. To do this, we first loop through all pixels in the labeled image to

find the position coordinates of marker pixels with RGB (255, 39, 0) matching the color of the circle. Then we take the minimum and maximum over the x and y coordinates of these pixels and extract the patch which contains the selfie stick. We use this extracted patch as the input to our clustering and inpainting algorithms.

3. Approach

3.1 Overall Strategy: Divide and Conquer

We divide our task into two parts:

1. Identify and remove the pixels belonging to the self-stick from the image.
2. Inpaint the removed pixels with the weighted averaged of their neighboring pixels with Biharmonic equation based algorithms.

3.2 Attempted Solutions to the Segmentation Problem

3.2.1 Basic K-means Clustering

Intuitively, selfie sticks have some very distinctive features such as relatively uniform metallic texture and color, which could be identified through clustering on the pixels' color values. Instead of initializing the centroids randomly, we pick some pixel inside the endpoints of the selfie stick as the initial centroid for the selfie-stick cluster, and another point not on the selfie stick as the second centroid, which should speed up convergence and reduce the chance of getting stuck at local minima in the clustering. We use the Euclidean distance between the LAB values of the 2 pixels as the loss function which we minimize by cluster assignments.

We find that basic k-means clustering is fast since its time complexity is linear with the number of data and it is also easy to interpret and evaluate the clustering results. However, k-mean often produces clusters with relatively uniform size even if the selfie stick occupies a small portion of the input images. It is also sensitive to outliers and easily gets trapped in local optima.

3.3 Further Optimizations

3.3.1 K-means Feature Engineering

Since the selfie sticks have several features, such as uniform texture, color, and shape, which might distinguish them from the background, we added three extra features: pixel position, pixel color gradient, and pixel texture (using Matlab's entropy filter function) into k-means clustering.

3.3.2 Using Feathering for Inpaint

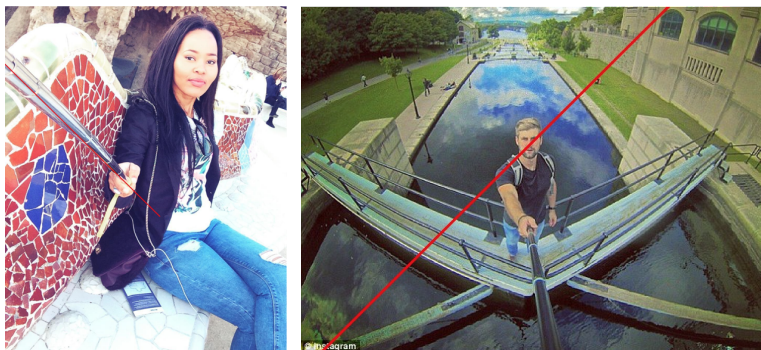
After removing the selfie-stick pixels, we use a simple feathering algorithm to further remove the neighbor pixels of each pixel within 8-pixel radius, since we want to capture as much as selfie-stick as possible, even at the cost of removing some backgrounds, which is tolerable because these false positive pixels are very likely to be inpainted with similar values.

3.4 Baseline and Oracle:□

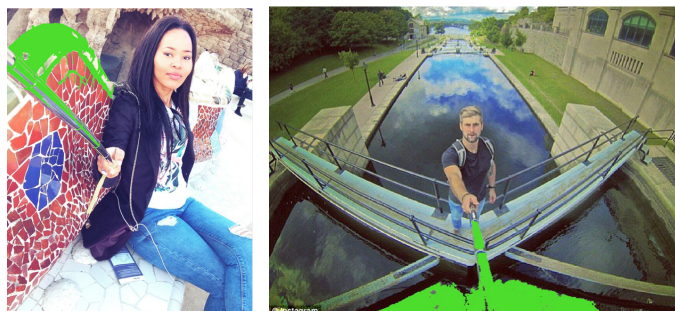
For the baseline, we first run Canny edge detection and Hough Line Transformation on an image with high θ value to identify all long and straight edges. Then we leverage the fact that the selfie stick should intersect with the boundary of the picture and select the longest edge from all qualified candidates as our prediction of the selfie stick position.

From the two typical output images from our baseline algorithm, we conclude that it has decent performance on photos without long straight objects besides the selfie stick. However, if either the selfie stick itself is not long

enough, or there exist other objects with long edges (e.g. bridge, stairs), the baseline algorithm can't segment the selfie stick from the background accurately.



For the oracle, we use the “magic wand” option in Photoshop to segment and remove the selfie stick from the picture. Though Photoshop is fairly effective at identifying the edges, it fails when the selfie stick consists of multiple segments and only manages to remove one of the segments. Therefore the user has to select each segment individually to remove it. Moreover, if the selfie stick surface reflects its surroundings is non-uniform in color, Photoshop will also fail the task.



4. Literature Review

Image segmentation is one of the central challenges in computer vision and image processing. So far tremendous progress has been made along two fundamental approaches: clustering and deep learning. Perhaps the simplest yet most effective general-purpose clustering algorithm is k-means [1], along with seeding heuristics such as k-means++ [2] to prevent poor clustering performance due to local minima. In recent periods, many novel methods have been introduced and implemented specifically for image segmentation. Notable examples include “superpixels” [3], where pixels are grouped into perceptually meaningful regions and clustered as a whole for higher efficiency and accuracy. In order to eliminate the necessity to manually specify the number of clusters, Oliver *et al* [4] proposed a staged version of k-means clustering where the algorithm begins by detecting the main contours of different objects and then automatically determines the optimal number of clusters. Chen *et al* [5] further integrated dynamic programming methods and represented the clustering problem as a decision graph which allows one to more accurately identify the cluster centers and boundaries. While these novel approaches are designed for segmenting objects in generic images, our approach places much focus on optimizing the feature sets which most effectively identifies the boundary of the selfie stick.


5. Error Analysis

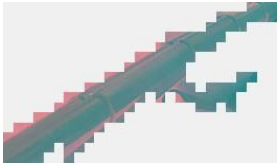





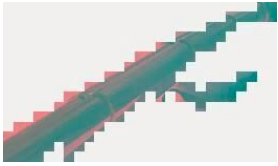


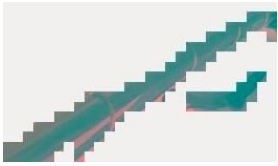


As mentioned above, directly evaluating the final output images can be very subjective. Since both the feathering and in-paint algorithms work with the segmentation results, it is the segmentation that determines how well our program performs. We assume that the intermediate segmentation result can be a good indication of how successful our algorithm is. Initially, we came up with the following score function to calculate the error rate for each segmentation:



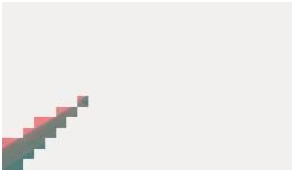
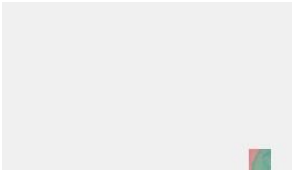
$$Score(segmentation) = -2.5 * \frac{\#false\ negative\ pixels}{\#pixels\ in\ the\ ground\ truth} - 1.0 * \frac{\#false\ positive\ pixels}{\#pixels\ in\ the\ ground\ truth}$$



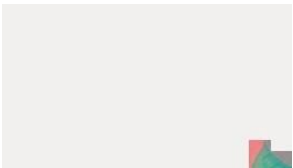

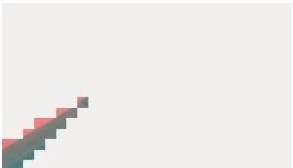
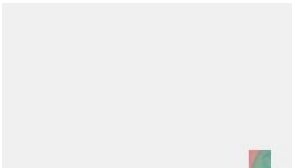

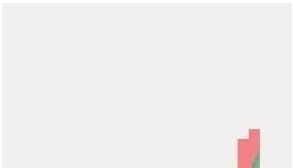
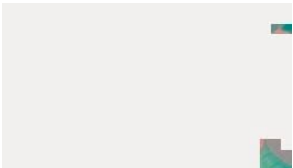
where false negative is defined as all the pixels that are the part of the selfie stick but are not segmented with the majority of the stick in our algorithm, and false positive is defined as the pixels in the background of the image that we recognize as the selfie stick. The coefficients are defined based on the fact that we could tolerate more false positives than false negatives, i.e. we can still produce a reasonable image even if some background pixels are segmented as the stick, but our result would be less ideal if some parts of the stick remain in the background. After making this computation on several images, we find that our scoring system does not accurately reflect the performance of the segmentation. Our segmentation problem is different from regular image segmentation problems, where the segments need to be as accurate as possible. As mentioned above, as long as our algorithm identifies the selfie stick and is able to remove all of it from the image, including parts of the background as part of the stick cluster is not necessarily an indication of a bad segmentation. For example, we have seen the following scenario, where segmentation (a) separates the left and right half of the stick into two clusters and segmentation (b) leaves some pixels along the edges of the stick unrecognized. Both segmentations receive a similar score using the score function above. After running our feathering and in-paint algorithm, we can tell that segmentation (b) is actually a better segmentation because the edges of the stick can still be covered up with the background color, while segmentation (a) would result in an image leaving an obvious partial stick. Therefore we conclude that quantitative evaluation might not be the right way to evaluate our program. We wrote more code to visualize the segmentation of each image patch. With this, we are able to compare how well different algorithms and feature extractions work. Then, we examine how our algorithm performs on different categories of images. Based on these analysis, we are able to find future improvements on our model.

5.1 Comparison of Segmentation Result Using Different Algorithms and Features

Original Picture			
K-means	Cluster 1	Cluster 2	Cluster 3
Feature Selection			





Color			
Color + Position			
Color + Texture			
Color + Gradient			


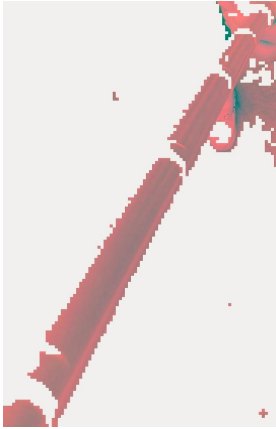



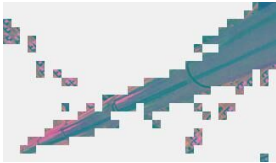


Original Picture			
HAC Feature Selection	Cluster 1	Cluster 2	Cluster 3
Color			

Color + Position			
Color + Texture			
Color + Gradient			









5.2 Comparison of Segmentation Results on Different Categories of Images





Category 1: Selfie stick is well separated from the background

Original Image	Cluster 1	Cluster 2	Cluster 3
			







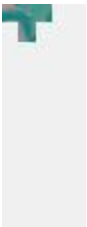



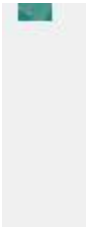

Category 2: Selfie stick is very similar to the background (either its color is similar to background color or it is reflective)

Original Image	Cluster 1	Cluster 2	Cluster 3
			
			

			
similar to background 			
HAC does better 			

Category 3: There is difference within the stick (either color change or stick is segmented by design)

Original Image	Cluster 1	Cluster 2	Cluster 3
color change on stick			

			
segmented stick 			
HAC does better 			

Category 4: Image is of poor quality and we do not have that many pixels to work with

Original Image	Cluster 1	Cluster 2	Cluster 3
too few pixels 			

5.3 Important Findings and Discussion

As seen in 5.1, using K-Means with color and texture features give us the best overall result. However, our segmentation is less successful with selfie sticks that are either internally segmented or are similar to background, or with images of poor quality. Surprisingly, HAC is able to segment the stick really well in the first two cases. We

believe that this is because HAC is able to preserve connectivity within image data, which K-Means cannot. This gives us an important insight into one possible extension of our model: we can categorize our image first to pick the most appropriate algorithm to use. This requires that our algorithm could recognize image patterns, which could be achieved using deep learning methods. We think that deep learning could also help with both segmentation and painting the stick. It has been proven that convolutional neural networks perform really well in image segmentation. We read extensively into de-convolutional neural network, which is one of the mostly commonly used framework for image segmentation. However, deep learning work well only with large amount of data. Our algorithm requires very specific type of images, i.e. selfies with selfie sticks. We would love to train a deconvnet model if we can find more data and more time, which is another reason we did not choose this approach. Also, our current algorithm produces a blurry in-paint result. In order to generate a more realistic in-paint, we can use generative models to create an appropriate replacement of the stick.

References

1. J. MacQueen, *et al.*, “Some methods for classification and analysis of multivariate observations”, in: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1, Oakland, CA, USA., 1967, pp. 281–297.
2. D. Arthur, S. Vassilvitskii, “k-means++: the advantages of careful seeding”, in: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics Philadelphia, PA, USA. pp. 1027–1035.
3. X. Ren, J. Malik, “Learning a classification model for segmentation” in *International Conference on Computer Vision*, Vol. 1, Nice, 2003, pp. 10-17.
4. A. Oliver, *et al.*, “Improving clustering algorithms for image segmentation using contour and region information”, in: *Automation, Quality and Testing, Robotics, 2006 IEEE International Conference on*, Vol. 2, IEEE, 2006, pp. 315–320.
5. Z. Chen *et al.*, “Image segmentation via improving clustering algorithms with density and distance” in: *3rd International Conference on Information Technology and Quantitative Management*, ITQM 2015, Volume 55, 2015, Pages 1015-1022