University of Grongingen

Faculty of science and engineering

User-driven image co-segmentation

Bachelor Thesis

Author:
Niek de Vries

Primary supervisor:
Jiří Kosinka

Secondary supervisor:
Zizhao Wu

July 11, 2019

Abstract

In this bachelor thesis, an application has been implemented that can be used to interactively perform image co-segmentation.

The main steps in this co-segmentation pipeline consist of a superpixel extraction step using SLIC, a feature extraction step including features based on color values, SIFT, and HOG, computing feature vectors for these superpixels, and using these feature vectors in two different co-segmentation approaches consisting of a k-means clustering method and a graph-cut method. For segmentation using graph-cut, the results are evaluated in the form of uncertainty scores for every superpixel.

The pipeline was implemented in a way that each individual step can be performed separately. These steps are integrated in a graphical user interface, providing a user with tools to conveniently select the input images, set the parameters, provide input markings on the images, and view the results of the intermediate steps. Methods to compare the segmentation results to a ground-truth image are also provided.

The application was tested on the CMU-Cornell iCoseg Dataset and showed that it was able to produce good segmentations especially on images where there is a high contrast between the subject to be co-segmented and the background.

Acknowledgements

This project was a collaboration between two students under the supervision of Jiří Kosinka and Zizhao Wu. I would like to thank the other student, Marco Lu, for designing and implementing the graphical user interface for the program and the work he has done to ensure that the integration of the co-segmentation pipeline worked smoothly. I would also like to thank our supervisors for their guidance and helpful suggestions during the project.

Contents

Chapter 1

Introduction

Image segmentation deals with the problem of partitioning a digital image into regions belonging to a common object class. The term co-segmentation refers to an extension of regular segmentation and was first introduced in Rother *et al.* [13]. In co-segmentation, a set of images with a common object is used to segment the common object from every image simultaneously.

After Rother *et al.*, more research has been performed on co-segmentation. Daryanto *et al.* [5] mentions many previous works on this subject, and also mentions several applications for co-segmentation. These applications include: image retrieval, object recognition, and video tracking and editing.

Co-segmentation methods can be categorized as unsupervised approaches and supervised approaches. Unsupervised co-segmentation does not require any human input during the process. An advantage of this approach is that it is fast and free from human error, but has as a drawback that it often does not work well if there is a lot of variance between the images. In contrast, supervised co-segmentation uses human input to perform the co-segmentation, enabling the possibility to detect and correct errors. However, this approach may be very time consuming and is prone to human error.

In this project we introduce a framework to facilitate the task of user-driven co-segmentation. Our goal is to provide the user with a program in which they can perform co-segmentation using a variety of different methods, while providing intermediate feedback of the results of the steps in the co-segmentation process. The individual parts of the segmentation pipeline can be performed separately and can be chosen by the user. The program is built in such a way that the individual steps can relatively easily be extended with more methods.

This work is a collaboration project between two students, of which one is responsible for the graphical user interface of the program, and the other is responsible for the key algorithms used in the co-segmentation pipeline. In this thesis, the latter part is covered. The former part is covered in the Bachelor's thesis by Marco Lu [12]. An overview of this graphical user interface is provided in Figure 1.1.

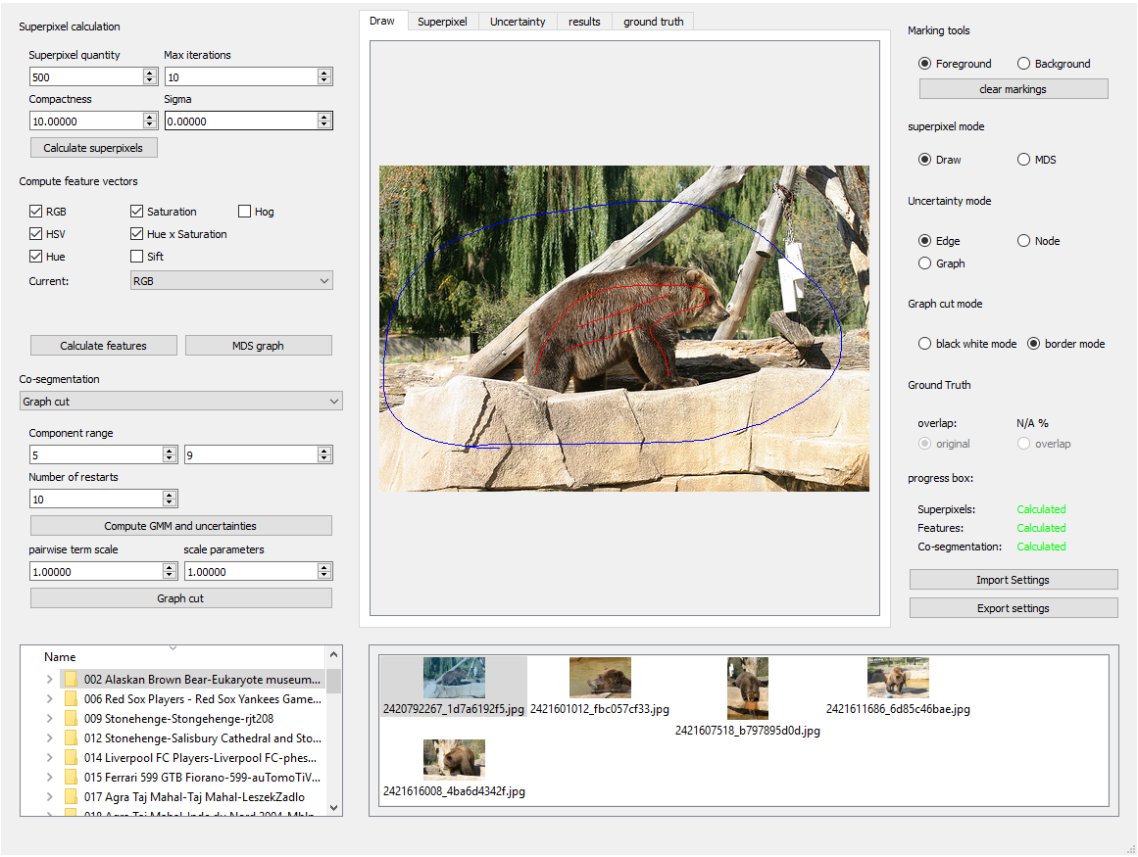

Figure 1.1: Graphical user interface of the program

Chapter 2

Related Work

Extensive research has already been performed on image segmentation as well as co-segmentation. As mentioned in the previous chapter, Daryanto *et al.* [5] provides an overview of this research. In this section we cover some of the works that have been used for the implementation of the co-segmentation pipeline in this project. We divide these works into unsupervised and supervised approaches.

2.1 Unsupervised

2.1.1 Co-segmentation by Histogram Matching

The work of Rother *et al.* [13] was the first appearance of image co-segmentation. This co-segmentation was performed through adding foreground similarity constraints into traditional Markov random field (MRF) based segmentation methods. In here, an L1-norm was used to represent the foreground similarity, and they optimize the energy using their introduced trust region graph cuts (TRGC) method.

2.1.2 Discriminative Clustering

In the work of Joulin *et al.* [8], the authors implemented a method for unsupervised co-segmentation. In this method, existing tools for bottom-up image segmentation such as normalized cuts were combined with kernel methods commonly used in object recognition, which were used within a discriminative clustering framework to perform foreground/background co-segmentation.

In their implementation, the authors use scale-invariant feature transform (SIFT) [11] descriptors, Gabor filters, and color histograms to extract the information needed from the images to perform the co-segmentation.

2.1.3 Multi-class Co-segmentation

While the previously mentioned methods deal with foreground/background segmentation, multi-class co-segmentation segments the images into more than two regions. Kim *et al.* [10] first introduced a strategy to solve this problem. In this strategy, the segmentation task is modeled as a temperature maximization on anisotropic heat diffusion, where there are k heat sources corresponding to k number of classes.

After this, Joulin *et al.* [9] extended their work from [8] to the multi-class case. This was done

by setting up a cost function combining spectral- and discriminative-clustering terms, which was then optimized using an expectation–maximization (EM) algorithm.

2.2 Supervised

2.2.1 iCoseg

(a) Group of related images + multiple scribbles (b) Current segmentations (c) Next scribble region recommendation

Figure 2.1: Overview of iCoseg [2]

The implementation of the co-segmentation pipeline in our project is based for a very large part on the work researched by Batra *et al.* [2]. In their method called iCoseg, they proposed a system for interactive co-segmentation in which a user provides foreground and background markings on one or more images in the set of images to be co-segmented, as illustrated in Figure 2.1.

In iCoseg, the markings received from a user are used to fit a common appearance model shared over all images which in turn is used to build an energy function for each separate image. These energy functions are then used to segment each individual image using a graph-cut algorithm.

In addition to this, they have developed an intelligent recommendation algorithm that can be used to suggest new regions for the user to mark.

2.2.2 Interactive Segmentation

In the work by Yang *et al.* [14], the authors have implemented image segmentation with graph-cut on a graph-based approach on superpixels created using Simple Linear Iterative Clustering (known as SLIC) [1] to decrease the amount of nodes in the graph.

2.2.3 Energy Minimization on Graphs

Both of the previously mentioned works make use of a graph-cut approach to minimize an energy function of a graph. This approach includes the use of a min-cut/max-flow algorithm. The authors of Boykov *et al.* [3] have performed experiments on min-cut/max-flow algorithms for energy minimization in computer vision. In addition to this, they have developed their own algorithm that accomplishes this task, which in many cases works several times faster than previous algorithms.

Chapter 3

Design

The problem of image co-segmentation can be viewed as a labeling problem, where we have a collection of images to which we want to collectively assign labels to their content. The result of this will be that each labeling site with the same label has something in common. This commonality could be that they should be foreground or background, belong to the same object, or that they belong to visually similar regions (e.g. foliage, rock, sky).

To address this problem we build a co-segmentation pipeline based on the iCoseg method where we split the problem up into the steps pre-processing, feature extraction, and co-segmentation. This pipeline is illustrated by Figure 3.1.

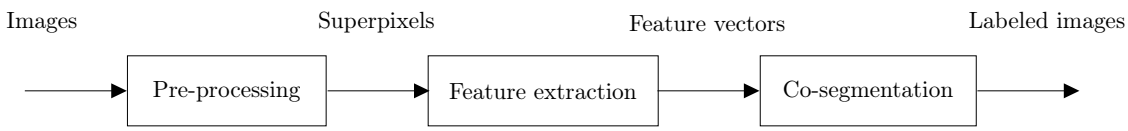

Figure 3.1: Co-segmentation pipeline

The following subsections describe these steps in more detail and serve as an overview of the methods that are used. In the next chapter we cover in more detail how these methods work and how they are implemented in our program.

3.1 Pre-processing

The first step in our co-segmentation pipeline is the segmentation of the image into superpixels. These superpixels consist of groups of connected pixels of similar color and grayvalue. An example of this can be seen in Figure 3.2. The intuition behind this step is that more often than not, the pixels within a given superpixel are connected to each other in some way and thus would receive the same label. This means that instead of having to find a label for every single pixel, we can label entire superpixels at once, reducing our problem size significantly and speeding up the process as a result.

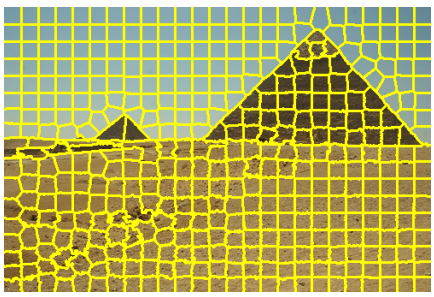

Figure 3.2: Image divided into superpixels

3.2 Feature Extraction

After splitting the images up into superpixels, we need to obtain information from the image that can be used to determine the label of the superpixels. Raw pixel data cannot be used directly to do this, so an abstraction of the image information, known as image features, needs to be extracted from it.

Our goal in this step is to obtain a suitable summary for every superpixel that can be used to differentiate between them. We do this by computing features for each of the superpixels using the pixels belonging to it.

Generally, one feature alone is not sufficient to provide a good description of a superpixel. This is why we do not compute a single feature, but instead compute multiple different features and combine these into a vector called a feature vector. An example feature vector for a superpixel is illustrated in Figure 3.3.

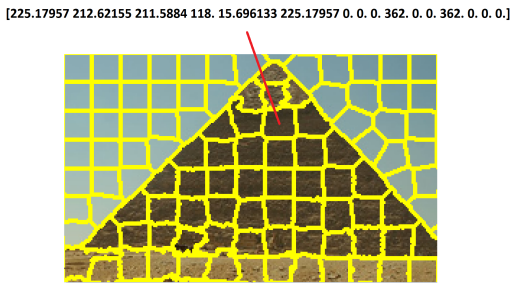

Figure 3.3: Feature vector for a superpixel

We provide several features that can be computed in our program. The user is to choose which of these features they want to include or exclude in the created feature vector. If applicable to the feature, the user can also adjust the parameters that are used in its computation.

The following subsections provide a brief overview of the different features that are included. We separate these into the categories of Color, SIFT, and HOG.

3.2.1 Color

The choice of the features based on color values that we include in our program is based on the color feature vector described in Hoiem *et al.* [7]. These features are calculated from the RGB and HSV color spaces and include:

1. The means for RGB.
2. The means for HSV.
3. A one-dimensional color histogram for hue with five bins.
4. A one-dimensional color histogram for saturation with three bins.
5. The entropy for the above histograms.

In our implementation of the histograms in our program, the user is able to adjust the number of bins freely. In addition, we include a two-dimensional color histogram for hue and saturation combined.

3.2.2 SIFT

The features described previously only use the value of the pixels and do not take into account their positions in the image. One feature vector that does do this is a descriptor obtained from SIFT. SIFT is a method that is usually employed in object recognition. This method uses a keypoint detection algorithm to find keypoints in an image that are invariant to scale and orientation. At these keypoints, a descriptor is computed that uses local image gradients and their orientations in

a region around the keypoint. These descriptors are feature vectors of 128 elements long and are relatively invariant to change in illumination and 3D viewpoint.

In our program we do not use the keypoint detection algorithm, but we use the center of the superpixels as keypoint locations. We let the user specify the size of the region around this keypoint and use a fixed orientation. After this we compute the SIFT descriptor for the region around this keypoint as normal. Please note that as we do not use the keypoint detection algorithm, our obtained descriptors are not invariant to scale and orientation.

3.2.3 HOG

In addition to SIFT, we include a Histogram of Oriented Gradients (also known as HOG) feature descriptor [4]. This feature descriptor is similar to SIFT and is often used for object detection. HOG differs from SIFT in that SIFT uses a keypoint detection algorithm to find the locations of the regions to compute the descriptors, while HOG computes descriptors over the entire image on a dense grid. A SIFT descriptor is always of length 128 and is computed with specific dimensions, while the HOG descriptor length may vary and depends on the dimensions that may be specified by the user.

3.3 Co-segmentation

After having obtained a feature vector for every superpixel in the set of images, these feature vectors can be used in co-segmentation. In our program we include two different methods of approach, which we categorize as unsupervised and supervised co-segmentation.

3.3.1 Unsupervised

In the category of unsupervised methods we include methods that do not require any further input from the user. One of these methods could be to perform a simple clustering on the entire set of all feature vectors using a simple clustering algorithm like k-means. More sophisticated unsupervised methods exist (e.g. [8]), however, as the main focus of our project is to provide interactive co-segmentation, we do not cover these in our program.

We include an implementation of k-means clustering to do unsupervised co-segmentation. The k-means clustering algorithm is a popular general-purpose clustering method that scales well to a large number of samples. This can be a useful property if a user is co-segmenting a large number of images at once.

3.3.2 Supervised

In our supervised approach we have a user provide markings on sections of the images indicating that certain superpixels are supposed to be foreground or background. We then perform segmentation using a graph-cut algorithm in a graph-based approach described in iCoseg.

In this approach, a directed weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is built for every image, where the vertices are superpixels and their edges correspond to their adjacency in the image. For these graphs, energy functions are defined as follows:

$$E(L) = \sum_{i \in \mathcal{V}} E_i(L_i) + \lambda \sum_{(i,j) \in \mathcal{E}} E_{ij}(L_i, L_j) \ ,$$

where L is a labeling of the image, the first term is the data (unary) term indicating the cost of assigning a superpixel to foreground or background, and the second term is the smoothness (pairwise) term used for penalizing label disagreement between neighbours. λ is a scalar which can be set by the user.

We now want to find a labeling that minimizes this energy, which is known as the maximum a posteriori estimate (MAP) of the solution. The technique employed here is that the graph is constructed for the energy function to be minimized in such a way that the minimum cut on the graph also minimizes the energy (either globally or locally).

According to the max-flow min-cut theorem by Ford and Fulkerson [6], the minimum cut of a graph is equivalent to the maximum flow, for which efficient algorithms exist. We use the max-flow/min-cut algorithm in Boykov *et al.* [3] to find the minimum cut in our graph.

Chapter 4

Implementation

Our implementation of the program is written in Python. This language is chosen because open-source Python implementations of the algorithms that we need are readily available, and Python makes it relatively easy to bring these implementations together.

In our program we make extensive use of `opencv-python`, a wrapper package for OpenCV Python bindings. OpenCV is a popular library used in computer vision. Before we begin with our co-segmentation process, we read the images using this library.

What follows is an explanation on how each of the steps of the co-segmentation pipeline work and how they are implemented.

4.1 Superpixel Extraction

We perform our superpixel extraction using SLIC. SLIC is appealing because of its relative simplicity and efficiency. Moreover, SLIC can be used to generate compact and nearly uniform superpixels, which is a useful property when extracting local image features for a superpixel in the feature extraction step.

SLIC uses a k-means clustering approach in CIELAB color space to generate superpixels. In this approach, k corresponds to the number of superpixels that are generated. SLIC also has a parameter for the compactness of a superpixel, which balances color proximity and space proximity. A higher compactness results in superpixels that are more square.

The open-source `scikit-image` library provides a segmentation module that includes an implementation of SLIC. This implementation was chosen as it was faster than several other Python implementations of SLIC that we tested.

We give the user control over the parameters for the approximate number of segments, compactness of the superpixels, the maximum number of iterations of k-means, as well as a parameter σ, which determines the width of a Gaussian smoothing kernel that can be used as an optional smoothing step before performing SLIC.

4.2 Superpixel Neighbours

To be able to calculate the pairwise terms for the energy functions in graph-cut, the neighborhood of each superpixel needs to be computed. Efficient methods to compute neighbours for each superpixel exist (e.g. using Delaunay tesselation). However, while testing these implementations

they sometimes produced wrong results, so in the current implementation it was decided to compute the adjacency of superpixels by computing the adjacency pixel-wise, which is slower but guarantees correct results.

4.3 Superpixel Centers

For our implementation of SIFT and HOG, we use keypoints at the center of every superpixel. One could use the centers that are used by SLIC during the computation of the superpixels for this. However, the current implementation that is used for SLIC does not provide this data as output, so we use the medians of the x and y coordinates of the pixels in a superpixel. The median is chosen here instead of the mean as the median should be more robust to outliers. This is a simple approach but may result in centers that are outside of the superpixel when a superpixel is an odd shape (e.g. long shape with curvature). The user can prevent this by making sure the computed superpixels are approximately square.

4.4 Features

In our program we provide methods to compute feature vectors for superpixels from features that are described in the following subsections. These individual feature vectors are combined into one larger feature vector per superpixel. A user can specify which of these individual feature vectors they want to include.

4.4.1 Color Means

These are the most basic feature vectors that are included in our program. They consist of two feature vectors of three features each, corresponding to the means of the pixel values in the RGB and HSV color spaces. We retrieve these features using the mean function from OpenCV.

4.4.2 Color Histograms

A color histogram is a representation of the distribution of color. To compute a color histogram, the range of possible pixel values is split up into a number of bins. The number of pixels that fall into each bin range is then counted. The length of the resulting feature vector depends on the number of bins, which is chosen by the user.

We include two one-dimensional color histograms for the hue and saturation channels, and one two-dimensional histogram for both hue and saturation channels combined of the HSV color space. These histograms are computed using the `calcHist` function from OpenCV.

4.4.3 Entropy

We include the entropy of the color histograms as an optional feature. For this entropy we use the Shannon entropy, which is a measure of disorder of a distribution and is computed using the following formula.

$S = -\sum_i (P_i \cdot \log P_i)$, where P_i is the ith element in the distribution P.

We implement this computation using the entropy function from the stats module of the open-source Python library `SciPy`.

4.4.4 SIFT

Our SIFT feature vectors are obtained by computing a SIFT descriptor around keypoints with a user determined size but fixed orientation at the center of each superpixel. These feature vectors have a length of 128 elements and are computed as follows (see Figure 4.1):

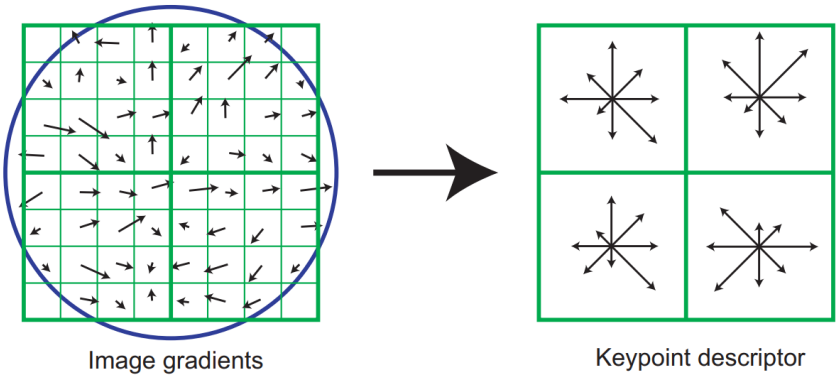

Figure 4.1: Visual representation of the computation of a 4x8 SIFT descriptor adapted from Lowe [11].

1. A 16x16 sample array is constructed for the region around the keypoint, where each of the elements is the dominant magnitude and orientation of the image at the sample point.
2. Samples located closer to the center of the keypoint need to contribute more. This can be achieved by weighting the elements using a Gaussian window.
3. A 4x4 grid is constructed where each grid cell is a histogram consisting of 8 orientation bins spaced apart by 45°. The 16x16 sample array is accumulated into these 4x4 sub-regions to obtain the descriptor.
4. This 4x4x8 descriptor is flattened to form a feature vector of length 128.

For the computation of this descriptor we use the SIFT implementation from OpenCV.

4.4.5 HOG

For our HOG feature vector we compute the HOG descriptor for an image patch around the center of the superpixel. The dimensions of this patch are set by the user. We use the implementation of OpenCV to obtain these HOG descriptors, which are computed as follows:

1. At every pixel, a gradient is computed using its neighbouring pixels. These gradients have a magnitude and an orientation.
2. The image patch is divided into cells and for each of these cells a histogram of gradients is computed, which stores the accumulated magnitude of each orientation range. The dimensions of these cells and the number of bins in the histogram can be specified by the user.
3. The user specifies the dimensions and stride of a block that is used to normalize the histograms. This block is moved over the image patch with a spacing depending on the stride. In each step, the histograms of the cells that are located in the block are concatenated and normalized.
4. The vectors obtained from the previous step are concatenated to form the final feature vector.

4.5 Unsupervised Segmentation

By now we have obtained a feature vector for every superpixel in the data. This data can now be used to cluster the superpixels using k-means clustering. We use the implementation found in the `sklearn.cluster` module from the `scikit-learn` library to perform this task. What follows is an overview of how k-means clustering works.

4.5.1 k-means Clustering

Given a set of n samples X, k-means tries to divide these samples into a set of k clusters C, where each cluster is described by a mean μ_j (also known as centroid) of the samples in the cluster, while trying to minimize the within-cluster sum of squares. This is described by the following formula:

$\sum_i^n \min_{\mu_j \in C}(||X_i - \mu_j||^2)$, where $||X_i - \mu_j||$ is the Euclidean distance between X_i and μ_j.

Finding this solution is NP-hard, but a local minimum can be found using Lloyd's algorithm, which works as follows:

First, a set of initial k means μ_k is chosen (e.g. chosen randomly from the samples). Then, the algorithm alternates between the following two steps until convergence:

1. Each sample is assigned to a cluster with the nearest mean according to the least squared Euclidean distance, following:

$$C_k = \{X_n : ||X_n - \mu_k||^2 \leq ||X_n - \mu_j||^2 \ \forall j, 1 \leq j \leq k\}$$

2. New means are calculated for these new clusters by:

$$\mu_k = \frac{1}{C_k} \sum_{X_j \in C_k} X_j$$

This algorithm is typically performed several times and the best result is chosen.

4.6 Supervised Segmentation

4.6.1 Graph-based

Our graph-cut algorithm for the graph-based approach is implemented using `PyMaxflow`, a Python wrapper for the `C++ maxflow` library by Vladimir Kolmogorov, implementing the algorithm described in Boykov *et al.* [3]

To be able to use this max-flow algorithm to find a minimum-cut in our graph described in the previous chapter, two additional vertices need to be added to this graph, which are the terminals s (source) and t (sink). These two terminals will both have edges to every superpixel vertex in the graph, and will represent foreground and background respectively. The unary term of our energy function is now represented in the graph by the edges between the terminals and the superpixel vertices, while the pairwise term is represented by the edges between the superpixels.

Following the iCoseg approach, we fit a foreground and background Gaussian Mixture Model for the user-labeled superpixels. Our unary term edge weights are then set as the negative log-likelihood of a superpixel's feature vector given these models. This task is accomplished using the `sklrean.mixture` package from the `scikit-learn` library.

For our smoothness term we use a Potts model defined as:

$$E(L_i, L_j) = I(L_i \neq L_j)\ exp(-\beta d_{ij})\ ,$$

where $I(\cdot)$ is 1(0) if the input is true(false), d_{ij} is the distance between the features of superpixels i and j (we use Euclidean distance), and β is a scale parameter which can be set by the user.

This model is contrast sensitive, meaning that it assigns high costs to adjacent superpixels that are close together in Euclidean space and vice versa. This encourages segmentations that follow strong edges in the image, which may or may not be preferred behaviour depending on the object to be segmented. Because of this, iCoseg uses a distance metric learning algorithm to learn these distances d_{ij} from user markings.

The max-flow algorithm can now be used to find the minimum cut separating the terminals; see Figure 4.2. After this cut, each superpixel will be attached to either the s or t terminal, and therefore be labeled as foreground or background accordingly.

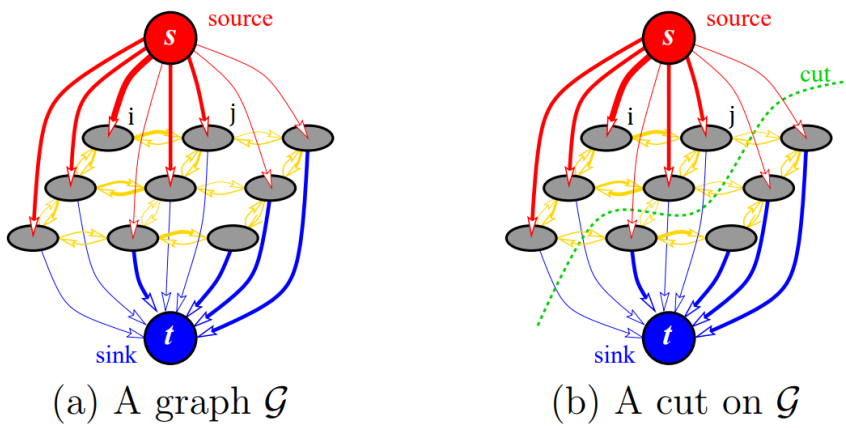

Figure 4.2: Visual representation of graph-cut adapted from Boykov *et al.* [3]

Chapter 5

Evaluation

5.1 Uncertainties

For the supervised segmentation, the segmentation can be evaluated using uncertainties as described in iCoseg. Using the methods described, we compute scores for every superpixel based on the results of the segmentation. These scores are a measure of the uncertainty there is for the label assigned to the superpixel.

We compute uncertainty scores based on three criteria, which are explained in the following subsections. These methods give us scores for every superpixel, which we can use to form three separate heat maps for every image.

5.1.1 Node uncertainty

The node uncertainty score is a score based on the entropy of the node beliefs. From the fit Gaussian Mixture Models, we retrieve the likelihoods of each Gaussian for a superpixel. The foreground and background likelihoods are normalized and the entropy of this distribution is calculated and used as our uncertainty score. A more uniform distribution means that there is more uncertainty at this superpixel, which is represented as a higher entropy.

5.1.2 Edge uncertainty

To compute this uncertainty, we find for every superpixel its k (=10) nearest neighbours from the user-labeled superpixels. We then use the entropy of the proportion of foreground and background neighbours as our uncertainty score for this superpixel. A higher entropy here again means more uncertainty.

We implement k-nearest neighbours using the `sklearn.neighbors` module from `scikit-learn`.

5.1.3 Graph-cut uncertainty

For this uncertainty, we use the difference in the energies between opposite labels of a superpixel. A lower energy here means that there is more uncertainty about this assignment.

Chapter 6

Results

The program is tested on images of the CMU-Cornell iCoseg Dataset [2]. This dataset was intended by the authors of iCoseg as a benchmark for future work. It contains groups of images of various subjects and also includes hand-labeled pixel-level ground-truth images for comparison with segmentation results.

6.1 k-means

From testing on several groups of images in the dataset it is observed that using k-means clustering can yield decent segmentations on images where the subject to be segmented from the image is entirely made up from colors that have a high contrast with the background. This method works well if the subject does not contain any colors that are also present in the background. An example of these results can be seen in Figure 6.1.

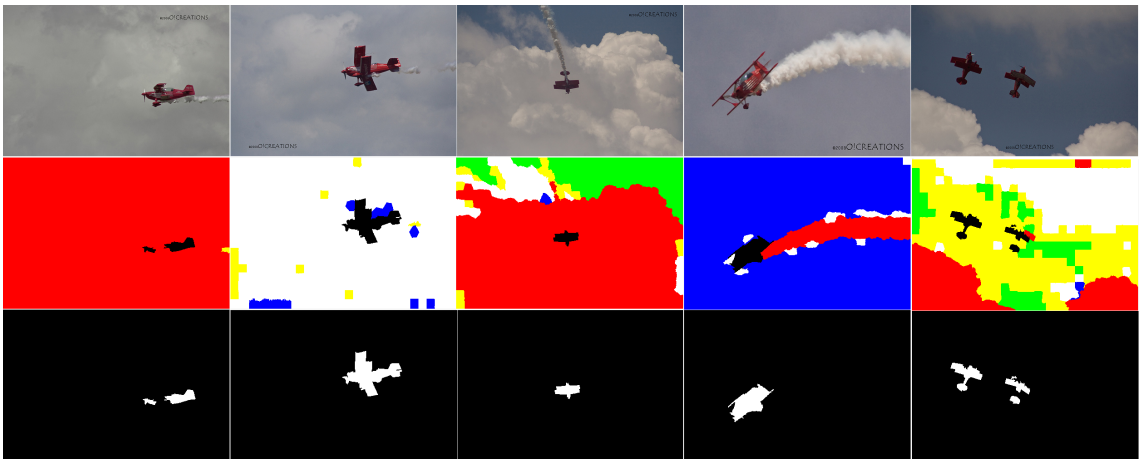

Figure 6.1: Top: original images. Middle: k-means clustering results (k=6). Bottom: only subject cluster visible

This approach is not suitable for images with a similar color palette between subject and background and when the subject shares colors with the background, as seen in Figure 6.2.

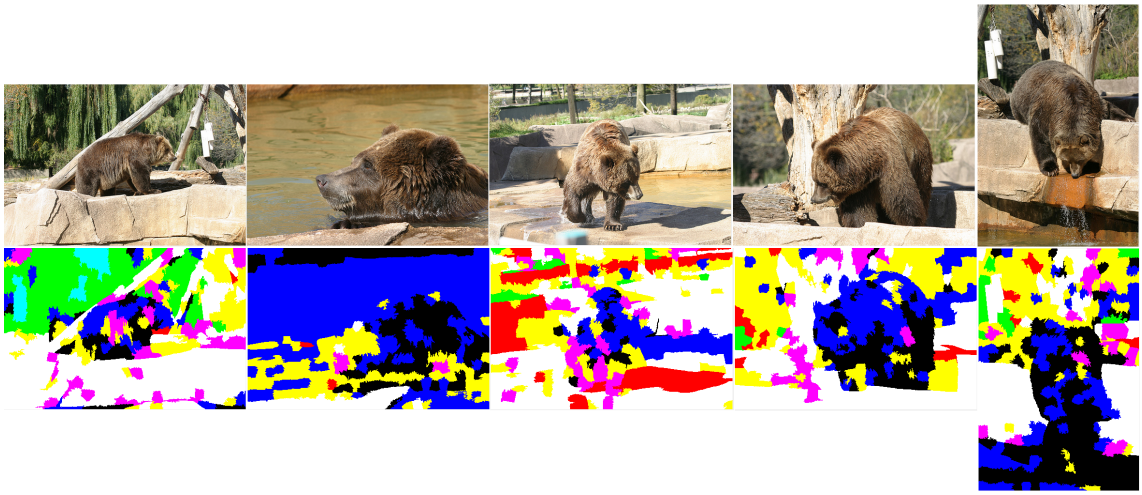

Figure 6.2: Top: original images. Bottom: k-means clustering results (k=8)

6.2 Graph-cut

As with k-means, the best results for graph-cut are also images where the foreground and background do not share any colors and the foreground and background consist of contrasting colors.

The following results are obtained using the color feature vector of length 16 described in Hoiem *et al.* [7], being the means for RGB and HSV, color histograms for hue (5 bins) and saturation (3 bins), and the entropy of the histograms.

With these settings we are able to obtain results with over 95% overlap with the ground-truth images on image sets with near optimal conditions. An example of these results can be seen in Figure 6.3.

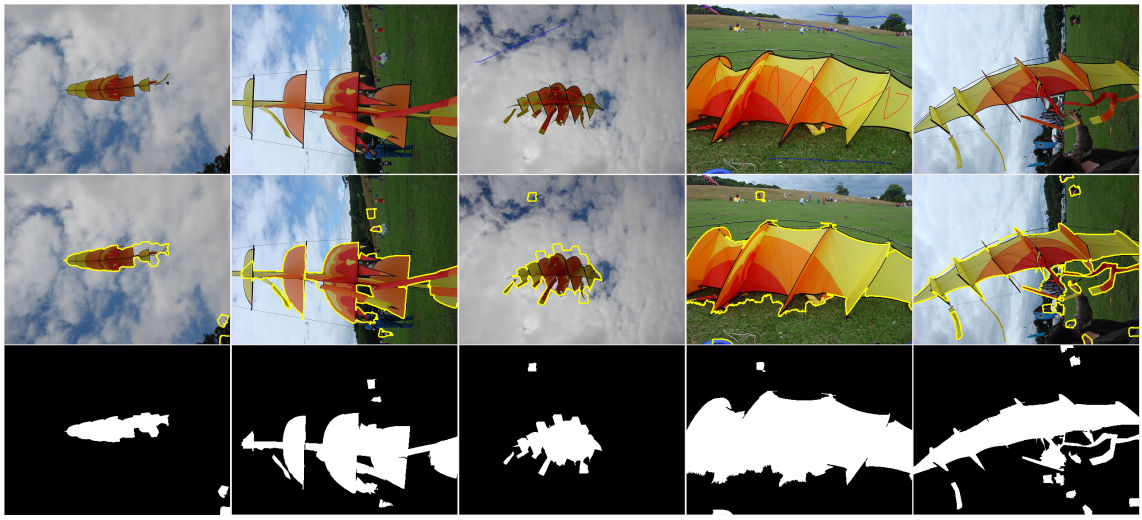

Figure 6.3: Top: original images with markings on the third and fourth image from the left. Middle: Graph-cut results. Bottom: Graph-cut results in black and white.

In images where this is not the case, our graph-cut method is able to produce better results than k-means, but struggles with assigning similarly colored superpixels correctly to foreground or background. An example of this can be seen in Figure 6.4.

The segmentations obtained from our graph-cut method can vary widely between computations, even when using the same markings. This can be attributed to the use of Gaussian Mixture Models

Figure 6.4: Top: original images with markings on the first and last image. Bottom: graph-cut results.

in the unary term. Our used implementation fits these models using an expectation-maximization (EM) algorithm, which tries to find the best model given the data. This method may not always find this best model, but may get stuck in a local maximum and thus output different models between computations, resulting in different segmentations when using these models in graph-cut.

Our tests with using graph-cut on only SIFT feature vectors are not very successful. Because we use keypoints with a fixed orientation and scale, the best results with this are obtained when the subject appears in the images in roughly the same size and orientation. In Figure 6.5 it is observed that when only marking the left and right image, the method is still able to segment the middle image, however, the superpixels on the remaining two images are completely left out as foreground.

Figure 6.5: Top: original images with markings. Bottom: graph-cut results with SIFT.

Chapter 7

Conclusion

In this work we have implemented a pipeline consisting of various methods that can be used to successfully co-segment groups of images. The quality of the co-segmentations are dependent on the methods chosen, parameters used, and the conditions of the images.

The main parts of the pipeline that was implemented consist of superpixel extraction using SLIC, a feature extraction step with various local features, and a segmentation step with either k-means clustering or graph-cut.

The work performed in the other part of this project provides a graphical user interface that lets a user perform these methods and set their parameters in an intuitive way. The results of the intermediate steps can be displayed to the user so the user can adjust the parameters to their liking.

The co-segmentation can be evaluated in the form of uncertainty scores that can be displayed in the graphical user interface. This provides insight to the user as to which regions of the images are difficult to segment for the program with the current settings.

The current iteration of the program works best on images with a high contrast between the subject to be co-segmented from the images and the background, while there being no shared color between these. Further improvements can be made to improve the results on images where this is not the case.

Chapter 8

Future work

In this section we mention several ways in which our co-segmentation pipeline could be improved.

More image features could be added to our current selection of features. This could provide a user with more tools to perform the co-segmentation process and improve the results from the segmentation step.

The results from co-segmentation using our implementation of SIFT and HOG are mixed. Because we use the centers of the superpixels as keypoints and use keypoints of fixed size and orientation, these points on the subject may differ greatly between images, resulting in a bad segmentation. Further improvements could be made on this.

We provide uncertainty scores to evaluate the results. In iCoseg, these results are used to provide suggestions to the user on where to provide new markings. Such a system could also prove to be useful in our graphical user interface implementation.

Our unsupervised segmentation approach only includes a simple clustering with k-means. This approach could be extended using more sophisticated methods such as in Joulin *et al.* [8], [9].

In the supervised segmentation approach our graph-cut implementation now only makes use of one type of method each to set the weights of the data term and smoothness term. Providing more methods to set these weights may give the user more ways to perform this step and obtain the results that they are looking for.

Bibliography

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. SLIC superpixels. Technical Report 149300, EPFL, June 2010.

[2] D. Batra, A. Kowdle, D. Parikh, J. Luo, and T. Chen. iCoseg: Interactive co-segmentation with intelligent scribble guidance. In *Computer Vision and Pattern Recognition*, 2010. CVPR '10.

[3] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on PAMI*, 26(9):1124–1137, September 2004.

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, June 2005.

[5] T. Daryanto, S. Ari, and S. Yang. Survey: Recent trends and techniques in image co-segmentation challenges, issues and its applications. *International Journal of Computer Science and Software Engineering*, 6(5):99–114, 2016.

[6] L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[7] D. Hoiem, A. A. Efros, and M. Hebert. Geometric context from a single image. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 1, pages 654–661 Vol. 1, Oct 2005.

[8] A. Joulin, F. Bach, and J. Ponce. Discriminative clustering for image co-segmentation. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1943–1950, June 2010.

[9] A. Joulin, F. Bach, and J. Ponce. Multi-class cosegmentation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 542–549, June 2012.

[10] G. Kim, E. P. Xing, L. Fei-Fei, and T. Kanade. Distributed cosegmentation via submodular optimization on anisotropic diffusion. In *2011 International Conference on Computer Vision*, pages 169–176, Nov 2011.

[11] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[12] M. Lu. *Interactive image co-segmentation*. Bachelor's thesis, Rijksuniversiteit Groningen, 2019.

[13] C. Rother, T. Minka, A. Blake, and V. Kolmogorov. Cosegmentation of image pairs by histogram matching - incorporating a global constraint into mrfs. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 993–1000, June 2006.

[14] S. Yang, Y. Zhu, and X. Wu. An interactive segmentation method based on superpixel. *MATEC Web of Conferences*, 22:01027, 01 2015.