PGM Tutorial:
Graph Cuts Algorithm for Image Segmentation

Marcus Klasson

January 23, 2019

1 Introduction

Probabilistic graphical models can be divided into two groups of models, namely *directed* and *undirected* graphical models. Directed graphical models, or Bayesian Networks, are representations of a probability distribution, where conditionally dependence is defined as directed edges between random variables. In undirected graphical models, the direct dependencies between the variables cannot be naturally described, so instead we identify which variables that are dependent and then define the strength of their interactions [5, 7]. These models can also be referred to as Markov Random Fields (MRFs) and one of its most important application is computer vision, where it has been successfully used for so called *early vision* tasks, such as image segmentation, image denoising and stereo reconstruction [3]. In this tutorial, we will focus on how to use MRFs for image denoising and segmentation, where the latter will be solved using the GrabCut algorithm.

When applying MRFs in vision and image processing, every pixel i is represented as having a hidden state x_i and a corresponding value y_i that we can extract from the observable image. The hidden states x_i models the "reasons" for why y_i has been generated and can either be discrete or continuously valued. Adjacent pixels typically have strong relationships to each other, e.g. the pixels describing a cow will certainly have surrounding pixels that also describe the cow, and this is modeled by connecting neighboring hidden state variables. Usually, each pixel is connected to 4 or 8 neighboring pixels, but it is possible to connect them in less symmetric ways. Figure 1 illustrates a MRF based on an image, which also can be described as the joint distribution $p(\mathbf{x}, \mathbf{y})$ where $\mathbf{x} = \{x_1, \dots, x_N\}$, $\mathbf{y} = \{y_1, \dots, y_N\}$ and N is the total number of pixels.

The connections between the pixels are represented as undirected edges. In practice, two adjacent pixels i and j has two edges (i, j) and (j, i) with a corresponding nonnegative cost or weight , which corresponds to the strength of the pixels relationship. There are many different approaches to compute these pairwise costs with some hyperparameters to tune. For now, we let ω be the hyperparameter set for computing the pairwise costs.

The goal is to obtain more knowledge about the hidden states $\mathbf{x}$ and we do so by estimating their posterior probabilities with Bayes' rule, as

$$p(\mathbf{x}|\,\mathbf{y}, \omega) = \frac{p(\mathbf{x}, \mathbf{y}|\,\omega)}{p(\mathbf{y}|\,\omega)} = \frac{p(\mathbf{y}|\,\mathbf{x}, \omega)p(\mathbf{x}|\,\omega)}{p(\mathbf{y}|\,\omega)} \propto p(\mathbf{y}|\,\mathbf{x}, \omega)p(\mathbf{x}|\,\omega), \tag{1}$$

where $p(\mathbf{y}|\,\mathbf{x}, \omega)$ is the observation likelihood and $p(\mathbf{x}|\,\omega)$ is the prior assumption of the hidden states. For MRFs, it is common to write the posterior as a sum of energies, as

$$p(\mathbf{x}|\,\mathbf{y}, \omega) = \frac{1}{Z(\mathbf{y}, \omega)} \exp\{-E(\mathbf{x}, \mathbf{y}, \omega)\}, \tag{2}$$

where

$$E(\mathbf{x}, \mathbf{y}, \omega) = \sum_i U_i(x_i, y_i) + \sum_{(i,j)\in\mathcal{E}} V_{ij}(x_i, x_j, \omega). \tag{3}$$

$Z(\mathbf{y}, \omega)$ is the partition function, which gives the posterior the property of a probability density function. U is called the unary term and is the likelihood $p(\mathbf{y}|\,\mathbf{x}, \omega)$ of the pixel values given the hidden states. V is the pairwise cost of the edge between neighboring pixels in the set $\mathcal{E}$ containing all pixel pairs. However, the partition function $Z(\mathbf{y}, \omega)$ is intractable to compute. The most common approach in computer vision is then to use Maximum-a-Posteriori (MAP) estimation.

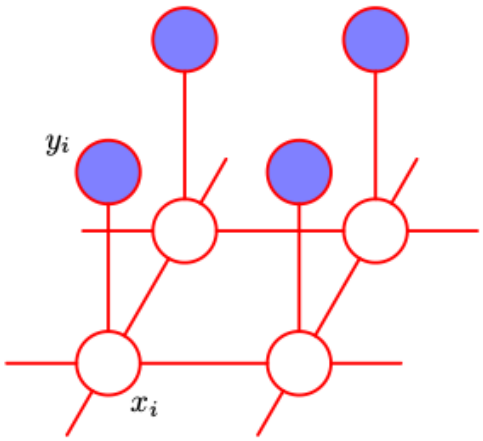


Figure 1: An undirected graphical model representing a Markov Random Field for vision tasks, in which x_i is the state of pixel i (often unknown) and y_i denotes the value pixel i in the observed image. Figure is taken from [1].

Inferring $\mathbf{x}$ with MAP estimation is done by computing $\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} p(\mathbf{x}|\,\mathbf{y}, \omega)$, or equivalently minimizing the energy function as

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} E(\mathbf{x}, \mathbf{y}, \omega). \tag{4}$$

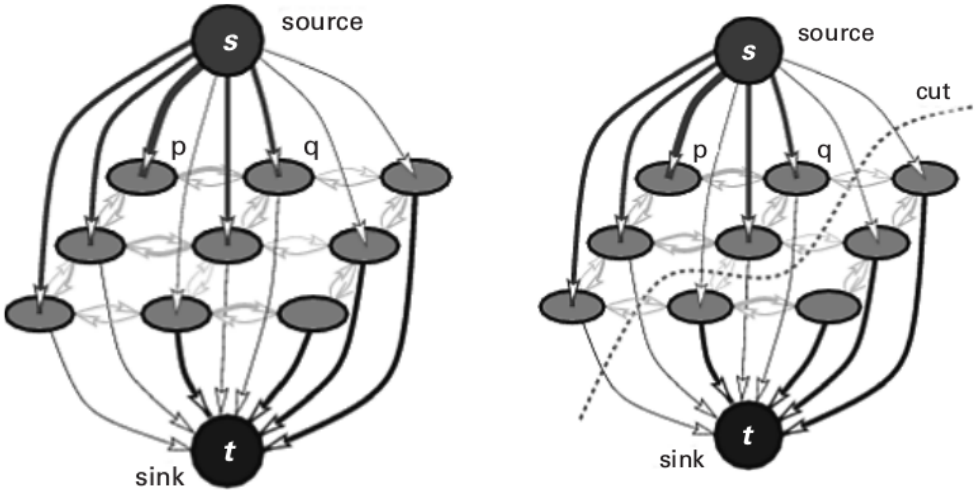
The advantage of MAP estimation is that we never need to evaluate $Z(\mathbf{y}, \omega)$, since we optimize the energy function with respect to $\mathbf{x}$ and not $\mathbf{y}$. The energy functions are as above written as sums of unary and pairwise terms and can be derived from the MAP estimate in a straight forward way:

$$\begin{aligned}
\hat{\mathbf{x}} &= \arg\max_{\mathbf{x}} p(\mathbf{x}|\,\mathbf{y}, \omega) && (5)\\
&= \arg\max_{\mathbf{x}} \prod_{i=1}^{N} p(y_i|\,x_i, \omega) p(\mathbf{x}) && (6)\\
&= \arg\max_{\mathbf{x}} \sum_{i=1}^{N} \log p(y_i|\,x_i, \omega) + \log p(\mathbf{x}) && (7)\\
&= \arg\max_{\mathbf{x}} \sum_{i=1}^{N} \log p(y_i|\,x_i, \omega) - \sum_{(i,j)\in\mathcal{E}} V_{ij}(x_i, x_j, \omega) && (8)\\
&= \arg\min_{\mathbf{x}} \sum_{i=1}^{N} -\log p(y_i|\,x_i, \omega) + \sum_{(i,j)\in\mathcal{E}} V_{ij}(x_i, x_j, \omega) && (9)\\
&= \arg\min_{\mathbf{x}} \sum_{i=1}^{N} U(x_i, y_i) + \sum_{(i,j)\in\mathcal{E}} V_{ij}(x_i, x_j, \omega) && (10)
\end{aligned}$$

Iterated Conditional Modes, Loopy Belief Propagation and Simulated Annealing are just some methods for computing the MAP estimate [3, 5, 6]. Another popular algorithm and often the most efficient solution is called graph cuts, which we will focus on in this tutorial.

2 Energy Minimization with Graph Cuts

Suppose $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ is a directed graph, where the set $\mathcal{V}$ contains vertices (nodes) corresponding to image pixels and $\mathcal{E}$ contains the edges that connect neighboring pixels. There also exists two terminal nodes called the source s and the sink t, such that $s, t \in \mathcal{V}$, where s has directed edges going to all pixels and all pixels have directed edges to t. See Figure 2a for an illustration of a 3×3-graph including the sink and source nodes. An s-t–cut C is a partition of the nodes in $\mathcal{V}$



(a) A graph $\mathcal{G}$.

(b) A cut on $\mathcal{G}$.

Figure 2: Example of a directed graph (left) and when a cut has been performed (right). The thickness of a directed edge represents a larger cost of performing the cut across the edge. The figure is taken from [3].

into two disjoint sets S, T, such that $s \in S$ and $t \in T$. Every possible cut has a cost defined as the sum of costs associated to all edges that go from S to T. The minimum s-t–cut problem, or min-cut problem, is to find a cut C with the smallest cost, and Figure 2b illustrates such a cut. This problem was proved by Ford and Fulkerson to be equivalent to maximum flow from source and sink (the same as drain t), so min-cut and maximum flow is often used interchangeably in the context of computer vision. Note that a cut can be viewed as a binary labeling, e.g. in segmentation the pixels belonging to the background of a scene can be assigned to the source node and foreground pixels to the sink node. There exists generalizations of min-cut with more terminal nodes involved than s and t, but such generalizations are NP-hard [6, 3].

Consider the graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ above, where $\mathcal{V} = \{v_1, \ldots, v_N, s, t\}$. Each cut on $\mathcal{G}$ has some cost, and thus $\mathcal{G}$ represents the energy function which maps any possible cut to the sum of non-negative unary and pairwise terms. Let $\mathbf{x} = \{x_1, \ldots, x_N\}$ be a set of binary variables, then these variables may correspond to the vertices in $\mathcal{G}$ except s, t, such that $x_i = 0$ when $v_i \in S$ and $x_i = 1$ when $v_i \in T$. Hence, the energy function E can be viewed as a function with N binary variables, where $E(x_1, \ldots, x_N)$ is equal to the cost of a cut defined by the values of $x_1, \ldots, x_N$ ($x_i \in \{0, 1\}$). The conclusion is that the min-cut on $\mathcal{G}$ corresponds to minimizing the energy function E [6].

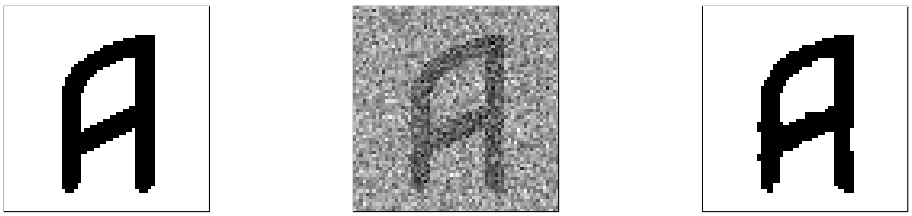
3 Exercises

The programming exercises includes implementing a solution to the image denoising problem and also a segmentation algorithm called GrabCut. The task is to fill in code skeletons in the provided Matlab files and generate plausible results. Make sure that the Image Processing Toolbox is installed into Matlab, since we will use some of its functions to import images and so on.

When we want to solve min-cut problems, we typically download some software package since these algorithms can be cumbersome to implement in an efficient way. Therefore, we will use a Matlab package with the Boykov-Kolmogorov algorithm in the exercises[1][2]. This package uses MEX files, which allows running C/C++ code in Matlab that usually results in faster execution of programs. These files has to be compiled in Matlab. Go to the folder `maxflow` and run `make.m`. The Matlab command window prints "MEX completed successfully" if the compilation was made without errors and you can sanity check that the maxflow implementation works by running `test1.m` and `test2.m`.

[1] `https://se.mathworks.com/matlabcentral/fileexchange/21310-maxflow`

[2] `https://vision.cs.uwaterloo.ca/code/`



(a) (b) (c)

Figure 3: (a) Original binary image. (b) Original image distorted by adding Gaussian noise. (c) Restored binary image obtained using the graph cut algorithm.

The main work load in a programming sense will be on designing appropriate unary and pairwise terms, which helps us assign specific pixels to one of the terminal nodes while handling the discontinuities between the pixels.

3.1 Exercise 1: Image Denoising

A nice example of undirected graphs in general is removing noise from binary images. In Figure 6b, we have added Gaussian noise to the image in Figure 6a. The noisy observed image has pixels $y_i \in \{0, 1, \ldots, 255\}$, where the index $i = 1, \ldots, N$ runs over all pixels. As illustrated in Figure 1, we also assume that there exist an unknown noise-free image with pixel values $x_i \in \{0, 1\}$, which $x_i = 0$ corresponds to the foreground and $x_i = 1$ to the background in this image. The goal is to recover the original binary image given the noisy observed image (see Figure 3c). In this example, we use $x_i = 0$ as foreground, since it is easier to inspect the restored binary image when the letter is represented with black pixels. Foregrounds are typically represented with $x_i = 1$ though.

To restore the binary image from the distorted one, we will minimize the energy function

$$E(\mathbf{x}) = \sum_i D(x_i, y_i) + \sum_{(i,j)\in\mathcal{E}} \gamma\,[x_i \neq x_j]. \tag{11}$$

The unary term $D(x_i = 0, y_i)$ is the cost for assigning label 0 to pixel i. We define D for pixel i as

$$D(x_i, y_i) = \begin{cases} y_i & \text{if } x_i = 0 \\ 255 - y_i & \text{if } x_i = 1 \end{cases}. \tag{12}$$

Thus, assuming y_i is small, i.e. a dark pixel, the cost will be low for assigning label $x_i = 0$ and high if we assign label $x_i = 1$ to y_i.

> **Task 1.1** Complete the code for computing the unary terms with Equation 12 in the function `compute_unary_terms.m` in the folder `exercise1`.

The second term in Equation 11 defines the pairwise costs. If $[x_i \neq x_j] = 1$, then pixel i and pixel j belong to different segments (foreground and background), and $[x_i \neq x_j] = 0$ otherwise. Therefore, if two neighboring pixels belong to the same segment, the term γ does not contribute to $E(\mathbf{x})$. In this exercise, we set $\gamma = 50$ and simply give the weight γ to all neighboring pixels in the image.

> **Task 1.2** Read through the code in function `compute_pairwise_terms.m` in the folder `exercise1` and make sure that you understand how the pairwise terms are computed. How does varying the value of γ affect the results?

We will use the maximum flow algorithm to minimize the energy function E. As mentioned in the previous section, we will use a Matlab package that runs maximum flow/min-cut and focus on building the graph. The algorithm is executed with the following command:

```
[flow, labels] = maxflow(pairwise_terms, unary_terms)
```

The output `labels` is a binary vector, where `labels(i)` is 0 or 1 if pixel i is assigned to the source s or the sink t. The scalar `flow` is the maximum flow value, which is equivalent to the resulting cost of the cut.

Task 1.3 Run the script `image_denoising.m` and save the restored image. through the code in function `compute_pairwise_terms.m` in the folder `exercise1` and make sure that you understand how the pairwise terms are computed.

3.2 Exercise 2: Segmentation with GrabCut

The graph cut algorithm is also commonly used in image segmentation. The following task will again be about estimating which pixels that belongs to the background and foreground (or object), but we will use a more worked through method called GrabCut. In this exercise, we will use images with pixels in RGB space, which usually demands more complex modeling of the unary and pairwise terms than in the previous exercise. The unary terms can be estimated by using a Gaussian Mixture Model (GMM) to model the color distributions of each segment. An alternative and simpler approach is to let histograms represent the background and foreground color distributions, but the GMMs typically results in better segmentations. The pairwise terms will now take into account the differences of neighboring pixel values in order to encourage stronger connections for similar pixels [8].

The GrabCut algorithm uses an iterative optimization scheme to minimize the energy objective and ends with a user editing step to refine the resulting segmentation. Let $\mathbf{y} = (y_1, \dots, y_N)$ be image pixels in RGB color space and $\mathbf{x} = (x_1, \dots, x_N)$ represents the hidden variables for each pixel, where $x_i \in \{0, 1\}$ with 0 representing the background and 1 the foreground. Note that the values of $\mathbf{x}$ are still unknown and we need to estimate these. Algorithm 1 shows a summarized schema of the GrabCut algorithm. A disclaimer in the notation is that even though the pixel values y_i are three dimensional in RGB space, they will not be denoted with bold letters to keep consistency in the notation.

3.2.1 Defining the Unary and Pairwise Terms

We formalize the energy minimization objective as

$$E(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}, \mathbf{k}) = U(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}, \mathbf{k}) + V(\mathbf{x}, \mathbf{y}), \tag{13}$$

where U and V are the unary and pairwise term respectively. Before we present how these terms are computed, it is necessary to provide some notation about the Gaussian Mixture Models.

A GMM consists of K Gaussian components, where each component has its own mean $\boldsymbol{\mu_k}$ and covariance $\boldsymbol{\Sigma_k}$ indexed by k. Note that the number K needs to be predefined by the user. The Gaussian components are learned from the data points in an unsupervised fashion along with a mixing coefficient $\boldsymbol{\pi}$, which is the ratio for how many data points that belongs to each component, such that $\sum_k \pi_k = 1$. These parameters are determined by maximum likelihood estimation, typically using the Expectation-Maximization (EM) algorithm [1]. We will train two separate GMMs, one for the background and foreground. We also introduce an additional vector $\mathbf{k} = (k_1, \dots, k_i, \dots, k_N)$ with $k_i = \{1, \dots, K\}$, which assigns to each pixel a unique GMM component either from background or foreground according to if $x_i = 0$ or 1. Thus, the parameters of the GMMs can be denoted as the set

$$\boldsymbol{\theta} = \{\boldsymbol{\mu}(x_i, k_i), \boldsymbol{\Sigma}(x_i, k_i), \boldsymbol{\pi}(x_i, k_i); x_i \in \{0, 1\}, k_i \in \{1, \dots, K\}\}, \tag{14}$$

which consists of the means $\boldsymbol{\mu}$, covariances $\boldsymbol{\Sigma}$ and mixture coefficients $\boldsymbol{\pi}$ of the $2K$ Gaussian components for the background and foreground distributions [8].

The unary term U is defined as the negative log-likelihood of the GMM components given the pixels $\mathbf{y}$,

$$U(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}, \mathbf{k}) = \sum_{i=1}^{N} \sum_{k=1}^{K} U_i(x_i, y_i, \boldsymbol{\theta}, k) = -\log \prod_{i=1}^{N} \sum_{k=1}^{K} \mathcal{N}(y_i | \boldsymbol{\mu}(x_i, k), \boldsymbol{\Sigma}(x_i, k)) \boldsymbol{\pi}(x_i, k), \tag{15}$$

where $\mathcal{N}(\cdot)$ is the Gaussian probability density function with mean $\boldsymbol{\mu}(x_i, k_i)$ and covariance matrix $\boldsymbol{\Sigma}(x_i, k_i)$ and $\boldsymbol{\pi}(x_i, k_i)$ is the mixture weighting coefficient.

Task 2.1

Derive the negative log-likelihood function in Equation 15. Note that the pixels in $\mathbf{y}$ are assumed to be i.i.d and that pixel y_i belongs to either the foreground ($x_i = 1$) or background ($x_i = 0$).

The pairwise term is in [8] defined as

$$V(\mathbf{x}, \mathbf{y}) = \gamma \sum_{(i,j)\in\mathcal{E}} [x_i \neq x_j] \exp(-\beta ||y_i - y_j||^2), \tag{16}$$

where $[x_i \neq x_j]$ is an indicator function and $\mathcal{E}$ is the set of all edges for neighboring pixels. When $\beta = 0$, smoothness is encouraged everywhere to a degree determined by γ. However, setting $\beta > 0$ relaxes the smoothness in regions with high contrast, which has been shown to be more effective [4, 8].

3.2.2 Initialization of GrabCut

Now we have introduced the essential parts of the GrabCut algorithm, such that we can implement the segmentation method. Open the file `run_grabcut.m` in the folder named `exercise2`. In this script, we can select which image from the folder `images` that we wish to use for the segmentation task. Now we need to define which part of the image that belongs to the foreground/background by putting a bounding box around the object that we want to segment. The bounding box is set with the function `init_foreground_region.m`, but the the predefined boxes may also be used and is preferred for the provided images (especially when debugging the code). We also need to set how many Gaussian components the GMMs should use. In [8], it was preferred to use $K = 5$, but it can be decreased to $K = 3$. The issue that may arise when using too many components is that some components become very similar, which can lead to that no pixels get assigned to only one of those Gaussian components. This leads to some mixture coefficients $\boldsymbol{\pi}_k$ for some component k to become zero when updating the GMMs.

We use so called trimaps T to indicate which pixel indices that belong to the background and foreground. In GrabCut, the pixels outside the bounding box should belong to the background trimap T_B and the pixels within the box are labeled as unknown in the trimap T_U. The foreground trimap is initialized as the empty set, i.e. $T_F = \emptyset$. Since the hidden variables $\mathbf{x}$ are binary, we initialize it by setting the pixel with index $n \in T_U$ as $x_i = 1$ and pixels belonging to the background as $x_i = 0$.

3.2.3 Expectation-Maximization Algorithm

The parameters of the background and foreground GMMs are learned from the pixel data with the Expectation-Maximization (EM) algorithm. We first run the K-means algorithm to initialize the means, covariances and mixture coefficients of the GMM. The cluster centers from K-means are set to be the means $\boldsymbol{\mu}_k$ for Gaussian component k. We can then estimate the covariance matrix and mixture coefficients as

$$\begin{aligned}
\boldsymbol{\Sigma}_k &= \frac{1}{N_k - 1} \sum_{i=1}^{N_k} (y_i - \boldsymbol{\mu}_k)(y_i - \boldsymbol{\mu}_k)^T, \\
\boldsymbol{\pi}_k &= \frac{N_k}{\sum_k N_k},
\end{aligned}$$

where N_k is the number of pixels in the trimap that were assigned to cluster k. The EM algorithm then goes as follows:

E-step: Evaluate the responsibilities p_{ik} using the current parameter values

$$p_{ik} = \frac{\pi_k \mathcal{N}(y_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(y_i | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}. \tag{17}$$

M-step: Re-estimate the parameters with the computed responsibilities p_{nk}

$$\boldsymbol{\mu}_k^{new} = \frac{1}{N_k} \sum_{i=1}^{N} p_{ik}\, y_i \tag{18}$$

$$\boldsymbol{\Sigma}_k^{new} = \frac{1}{N_k} \sum_{i=1}^{N} p_{ik}\, (y_i - \boldsymbol{\mu}_k^{new})(y_i - \boldsymbol{\mu}_k^{new})^T \tag{19}$$

$$\pi_k^{new} = \frac{N_k}{N}, \tag{20}$$

where $N_k = \sum_{i=1}^{N} p_{ik}$.

Iterate between the E- and M-step for some number of iterations or until the log-likelihood of the parameters have converged [1].

Task 2.2

Implement the equations for the E- and M-step of the EM algorithm in the function `EM_algorithm.m` in folder `exercise2`.

3.2.4 Computing the Pairwise Terms

Regarding the pairwise terms, we will make some minor generalizations of Equation 16. The summation over neighboring pixels can be omitted, since it is implicitly included in the graph cut algorithm. Also, by removing the indicator function $[x_i \neq x_j]$ we are only required to compute the pairwise terms once without any performance loss. Otherwise we would have to recompute these terms after every new estimate of the hidden variables $\mathbf{x}$. Thus, each pairwise term v can be computed as

$$V_{ij} = \gamma \exp(-\beta ||y_i - y_j||^2). \tag{21}$$

Task 2.3

Fill in the lines of code in the function `compute_pairwise_terms.m` into the folder `exercise2` with Equation 21 for computing the pairwise cost.

Recall that $V_{ij} = V_{ji}$. Also, note that we have ignored the indicator function between x_i and x_j, since it was done in [9]. We set the smoothing parameter $\gamma = 50$, which was a versatile setting for several different images according to [8]. Parameter β is supposed to ensure that the exponential term switches appropriately between high and low contrast. Setting it to $\beta = 0.02$ works well for the provided images.

3.2.5 Iterative Minimization

Before GrabCut, graph cuts was used as a "one-shot" algorithm for segmentation. The advantage with Grabcut is that the hidden variables $\mathbf{x}$ are updated iteratively through the re-estimations of the GMM parameters with the newly labeled pixels in T_U. The first step of the re-estimation is to assign the most likely Gaussian component for each pixel y_i based on the value of x_i. In other words, we pick the most likely Gaussian component from the foreground GMM if $x_i = 1$ for pixel y_i, otherwise if $x_i = 0$, we pick a component from the background GMM. The k'th Gaussian component is assigned to pixel y_i by minimizing the unary term with respect to k, as

$$k_i = \arg\min_k U_i(x_i, y_i, \boldsymbol{\theta}, k), \tag{22}$$

where $k = \{1, \ldots, K\}$.

The second step is to re-estimate the GMM parameters based on the assigned pixels to each components. Assume that we wish to estimate the parameters of the foreground GMM. For a

given GMM component k', we have the subset of pixels $F(k) = \{\mathbf{y}_n : k_i = k', x_i = 1\}$. The new mean $\boldsymbol{\mu}(x_i, k_i)$ and covariance matrix $\boldsymbol{\Sigma}(x_i, k_i)$ are estimated with the sample mean and covariance formulas, which in Matlab can be computed with the `mean` and `cov` functions. The mixture coefficients are estimated as $\boldsymbol{\pi} = |F(k')| / \sum_k |F(k)|$, where $|\mathbf{S}|$ denotes the size of set $\mathbf{S}$.

The third step is performing the min cut and then we need to compute the unary terms between the terminal nodes and all pixels. The costs are represented as the negative log-likelihoods that the pixel y_i belongs to either the background and foreground with the corresponding GMM parameters. Thus, the cost is computed by marginalizing out k as

$$U_i(x_i, y_i, \boldsymbol{\theta}) = -\log \sum_{k=1}^{K} \boldsymbol{\pi}(x_i, k)\, \mathcal{N}(y_i | \boldsymbol{\mu}(x_i, k), \boldsymbol{\Sigma}(x_i, k)). \tag{23}$$

Task 2.4

Compute the unary terms with the negative log-likelihood in the function `compute_unary_terms.m` in folder `exercise2`. You may use the expression of the log-likelihood that you derived in Task 2.1.

3.2.6 User Editing

After the iterative minimization scheme, the resulting segmentation is displayed and it is optional to perform some user editing steps for refining the segmentation. If the user editing step is selected, then it will pop up a window requesting the user to first fix some foreground pixels and secondly background pixels. The unary terms of the fixed pixels needs to be changed, such that there is a high cost for labeling a fixed foreground pixel as background. When the pixels have been fixed, we update the unary terms and run graph cuts a final time.

3.2.7 Expected Results

A showcase of segmentation results for the provided images are displayed in Figure 4 and 5. They typically need user editing after the iterative minimization, which certainly improves the segmentation in most cases. What would probably have improved the results is a border matting step, e.g. the one introduced in the original GrabCut paper [8]. This would deal with the problem of blurred and mixed pixels around object boundaries and create more smooth transitions between the foreground and background.

(a) (b)

Figure 4: Results from GrabCut without user editing. (a) Original image. (b) First segmentation.

(a)

(b)

(c)

(d)

Figure 5: Results from GrabCut with user editing. (a) Original image, (b) First segmentation, (c) User edits by fixing pixels. Fixed foreground pixels are white and background pixels are red, (d) Second segmentation.

Data: Image pixels $\mathbf{y}$.
Result: Binary labeling of $\mathbf{x}$.
Initialize trimap by supplying only T_B from bounding box. The foreground is set to $T_F = \emptyset$ and $T_U = \bar{T}_B$ to the complement of the background;
Initialize $\alpha_i = 0$ if $i \in T_B$ and $\alpha_i = 1$ if $i \in T_U$;
$[\text{GMM}_{fg}, \text{GMM}_{bg}]$ = EM Algorithm;
Iterative Minimization;
while $n_{iter} <= 3$ **do**
1. Assign GMM components for each pixel;
2. Update GMM_{fg} and GMM_{bg} based on pixels assigned to each GMM component;
3. Compute unary terms U.;
4. Estimate segmentation with graph cuts algorithm.

end
User editing: Fix some pixels either to $\alpha_i = 0$ (background brush) or $\alpha_i = 1$ (foreground brush) and update trimap accordingly. Perform step 3 and 4 above, just once.;

Algorithm 1: The GrabCut algorithm.

4 Summary

This tutorial has been dealing with formalizing energy minimization objectives for two computer vision tasks - image denoising and binary segmentation - and we have solved both tasks by using the efficient graph cuts algorithm. One of the main tasks with MRFs in vision is to build graphs and coming up with clever ways to measure the similarities between pixels (pairwise terms) and their relations to what is present in the image (unary terms), e.g. foreground and background, or sky, grass, cow, etc. For future work with the GrabCut algorithm, the border matting step should be added, investigate other methods to compute the pairwise terms and also the initialization of the Gaussian components in the GMMs could be changed. When your implementation of GrabCut works properly, feel free to try it on some random photo (see Figure 6).

(a) (b)

Figure 6: New years day 2011, Luleå, Sweden.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] Mårten Björkman. Lab 3: Image segmentation. Computer Lab in course DD2423 Image Analysis and Computer Vision given at KTH, 2017.

[3] Andrew Blake, Pushmeet Kohli, and Carsten Rother. *Markov Random Fields for Vision and Image Processing*. The MIT Press, 2011.

[4] Y. Boykov and M. P. Jolly. Interactive graph cuts for optimal boundary amp;amp; region segmentation of objects in n-d images. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, 2001.

[5] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009.

[6] V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159, 2004.

[7] Volodymyr Kuleshov and Stefano Ermon. Introductory course to probabilistic graphical models. `https://ermongroup.github.io/cs228-notes/`, 2018.

[8] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. In *ACM transactions on graphics (TOG)*, volume 23, pages 309–314. ACM, 2004.

[9] Justin F. Talbot and Xiaoqian Xu. Implementing grabcut. Technical report, Brigham Young University, January 2006.