Markov Random Fields

Tutorial for KTH PGM DD2420

1 Introduction and Overview

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 [4, 5]. These models can also be referred to as Markov Random Fields (MRFs). One important applications where MRFs have been used is computer vision for tasks as image segmentation, image denoising and stereo matching [1]. In this tutorial, we will give an introduction to the formal definition and properties of MRFs followed by showcasing how MRFs can be used as a represention for images. In the coding exercises, we will demonstrate how two algorithms, namely *loopy belief propagation* and *graph cuts*, can be used for solving image denoising tasks.

1.1 Markov Random Fields

We turn to the Misconception example (see Section 4.1 in Koller and Friedman [4]) where four students meet in pairs to work on a homework assignment for a class. The pairs that work together are illustrated by the edges in the undirected graph in Figure 1, which forms the pairs are (A, B), (B, C), (C, D) and (D, A). For each pair, we want to represent how likely the students are to agree with each other on how to solve the assignment. We let this interaction be modeled with a nonnegative score function ϕ called a *factor*, where a higher scoring value corresponds to how probable a certain interaction is going to occur. As an example, the factor defining the interactions between A and B could defined as

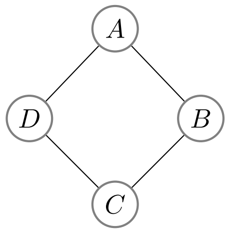


Figure 1: Undirected graph representation of the Misconception example from Koller and Friedman [4].

$$\phi(A, B) = \begin{cases} 100 & \text{if } A = B = 1 \\ 100 & \text{if } A = B = 0 \\ 1 & \text{otherwise,} \end{cases}$$

which would indicate that students A and B are more likely to agree with each other than to disagree.

We also want to model all interactions that can occur in the whole graph with a joint probability distribution. In a Bayesian network, the joint distribution is defined by following the directed structure in the graph and taking the product of all conditional probability distributions. Similarly, all existing factors are combined into an unnormalized distribution by multiplying them:

$$\tilde{p}(A, B, C, D) = \phi(A, B)\phi(B, C)\phi(C, D)\phi(D, A), \tag{1}$$

The final joint distribution is then defined as

$$p(A, B, C, D) = \frac{1}{Z}\tilde{p}(A, B, C, D), \tag{2}$$

where the normalizing constant $Z = \sum_{A,B,C,D} \tilde{p}(A, B, C, D)$ ensures that the distribution sums to one.

A formal definition of a Markov Random Field is a probability distribution over variables $x_1, ..., x_n$, where each x_i correspond to a node in an undirected graph. The distribution p is defined as

$$p(x_1, ..., x_n) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \phi_c(x_c), \tag{3}$$

where $\mathcal{C}$ denotes the set of cliques in the graph, and where the factor ϕ_c is a nonnegative function over all variables in a clique c. The normalizing constant, or *partition function*, is defined as

$$Z = \sum_{x_1, ..., x_n} \prod_{c \in \mathcal{C}} \phi_c(x_c), \tag{4}$$

where the sum is over all possible settings of each variable $x_1, ..., x_n$.

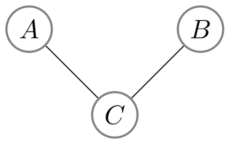
1.2 Independencies in Markov Random Fields

In directed graphs, we could check conditional independencies between variables using *d-separation*, which involved testing if the paths connecting two sets of nodes were blocked. However, this definition of paths being blocked becomes somewhat vague whenever a *v-structure* of directionalities, e.g. $A \rightarrow C \leftarrow B$, is present in the graph. Since the parent-child node relation is removed in undirected graphs, the definition of what independencies that can be described become much more simple: the variables A and B are dependent if they are connected by a path with unobserved variables. For example, suppose there is an undirected graph $A — C — B$, then the nodes A and B are dependent when C is unobserved. If C is observed, then the path between A and B will be blocked and thus they will be independent, i.e. $A \perp\!\!\!\perp B | C$.

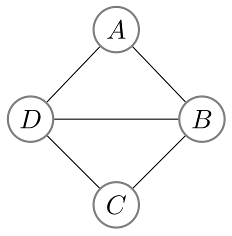
The *Markov blanket* MB(X) for a node X is defined as the minimal set of nodes that should be observed in order to make X independent of all other nodes in the graph. This definition holds for both directed and undirected graphs, but takes a simpler form in undirected graphs, because a node is independent of all other nodes conditioned only on its neighboring nodes.

1.3 Exercises

1. **Cliques in MRFs**
   (a) What is the definition of a clique in a graphical model?
   (b) What are the cliques in graph $\mathcal{G}_1$ and $\mathcal{G}_2$?



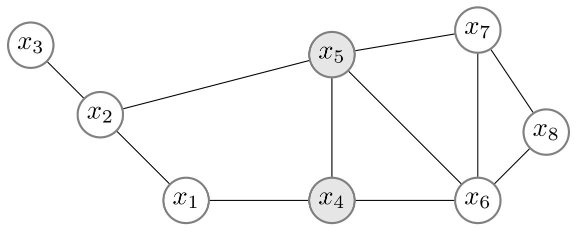
($\mathcal{G}_1$)



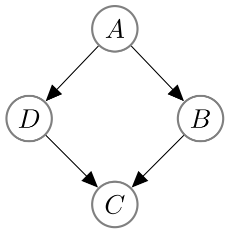
($\mathcal{G}_2$)

2. **Independencies**
   (a) Which nodes (or set of nodes) are independent in graph $\mathcal{G}_3$?
   (b) What is the Markov blanket for node x_2 and node x_6 in graph $\mathcal{G}_3$?

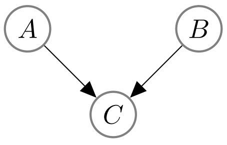
3. **Graphical Representations**
   (a) Suppose that we want to define a model with a graph that satisfies the indepence conditions $A \perp\!\!\!\perp C | \{B, D\}$, meaning A and C are independent if B and D are observed, and also $B \perp\!\!\!\perp D | \{A, C\}$. Explain why this model cannot be represented by the directed graph $\mathcal{G}_4$?
   (b) The directed graph $\mathcal{G}_5$ satisfies $A \perp\!\!\!\perp B$ when C is unobserved. Explain why can't this model be represented by an undirected graph?



($\mathcal{G}_3$)



($\mathcal{G}_4$)



($\mathcal{G}_5$)

2 Markov Random Fields as Image Representations

One popular application for undirected graphs is image processing and computer vision. A general assumption for images is that neighboring pixels should be similar, unless there pixels is nearby an edge. It turns out that Markov Random Fields (MRFs) are suitable for representing the spatial dependencies between neighboring pixels, which is illustrated in Figure 4. Images are described by observed pixel intensities $\mathbf{y} = \{y_1, ..., y_N\}$ and hidden states $\mathbf{x} = \{x_1, ..., x_N\}$, where the number of pixels N is $N = 4$ and $N = 9$ for the graphs in Figure 4. What the hidden states represent is dependent on the task. In many vision problems, the hidden states are discrete and can then be referred to as *labels*. For example, in image segmentation, each label then corresponds to some entity or object present in the image. The problem is how to determine reasonable values for each hidden state when we can only observe the pixel intensity values. This procedure is called *inference* and we will introduce how to perform inference in MRFs in the following sections.

2.1 Exercises

1. Explain the types of cliques that exist in the undirected graphs in Figure 4.
2. Which nodes are in the Markov Blanket of the hidden state x_5 in graph $\mathcal{G}_2$ in Figure 4?
3. Write an expression using factors over clique variables for the joint probability distribution $p(\mathbf{x}, \mathbf{y})$ of the graph $\mathcal{G}_1$ in Figure 4. Remember to define the normalizing constant Z.

2.2 Energy Formulation

We will onward refer to the hidden state, or label, x_i as the true pixel value, while y_i is called the observed pixel intensity. This distinction will make sense later in the exercises for binary image denoising.

We assumed earlier in the MRF that there is some correlation between the observed pixel intensity y_i and its pixel x_i. We also assumed that there is a correlation between a pixel x_i and its neighbours x_j, which comes naturally from studying an image. In computer vision, both these correlations are typically expressed with some arbitrary energy function that is computed using the cliques. These energy functions correspond to the factors ϕ_c over a clique c described in Section 1.1, so the energy functions be defined as nonnegative as well. The nonnegativity of the individual energy functions allows us to write the complete energy function as a sum over all energies:

$$E(\mathbf{x}, \mathbf{y}) = \sum_i \phi(x_i, y_i) + \sum_{\{i,j\}} \phi(x_i, x_j), \quad (5)$$

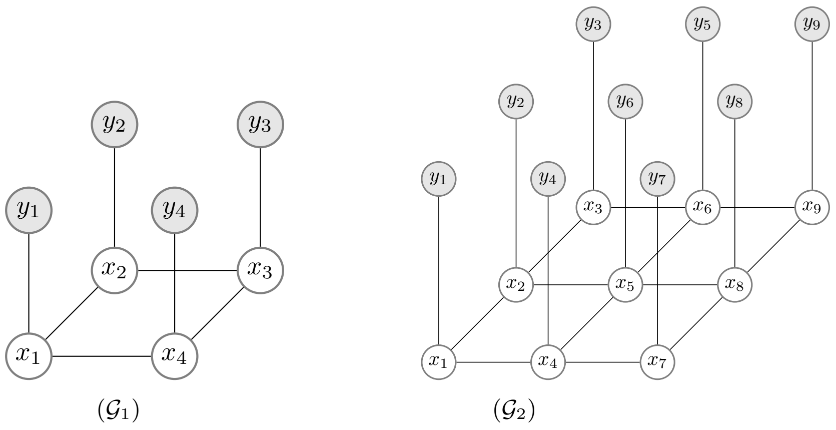

y_1 y_2 y_3 y_4
x_1 x_2 x_3 x_4
$(\mathcal{G}_1)$
y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8 y_9
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9
$(\mathcal{G}_2)$

Figure 4: An undirected graphical model representing a Markov Random Field for a 2×2 and 3×3 image, where y_i corresponds to the pixels and x_i the hidden states for each pixel.

where $\phi(x_i, y_i)$ and $\phi(x_i, x_j)$ are the energy functions representing the correlations (or energies) in the MRF. The joint probability distribution $\mathbf{x}$ and $\mathbf{y}$ is then given by

$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \exp\{-E(\mathbf{x}, \mathbf{y})\}, \tag{6}$$

where Z is the partition function for turning the expression into a valid probability distribution.

Now when we have formalized an expression for $p(\mathbf{x}, \mathbf{y})$, we can perform inference involving computing the posterior $p(\mathbf{x}|\mathbf{y})$ to find reasonable values for the pixels $\mathbf{x}$. Commonly, we can settle for assigning the value to each x_i that gives the highest probability in the posterior. This method is called Maximum-a-posteriori (MAP) inference, which involves computing $\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} p(\mathbf{x}|\mathbf{y})$ or equivalently $\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} E(\mathbf{x}|\mathbf{y})$.

2.3 Exercises

1. What is the computational advantage with using MAP inference?
2. Explain which parts of the joint probability distribution $p(\mathbf{x}, \mathbf{y})$ the energy functions $U(x_i, y_i)$ and $V(x_i, x_j)$ corresponds to.

3 Approximate Inference in Images

Inference can quickly become a computationally heavy task in graphical models. For example, a binary image with 100 pixels will have 2^{100} different combinations of possible images $\mathbf{x}$. Exact inference algorithms will basically not work easily with images. Luckily, we can settle for approximate inference algorithms for solving vision tasks. In this section, we will introduce two popular techniques for that, namely Loopy Belief Propagation and Graph Cuts. This section will include coding in Matlab, where both methods will be applied to reduce noise from binary images.

3.1 Loopy Belief Propagation

Loopy Belief Propagation (LBP) is a message passing algorithm for performing inference on complex graphs. Let x_i and x_j be two neighboring nodes. The node x_i has to wait until it has recieved the messages from all its neighbors, except x_j, until x_i can pass its message to x_j. This procedure is performed, given some ordering of directions to pass messages, for a fixed number of steps or until convergence (when the messages are unchanged). Note that LBP is only performed on the hidden nodes.

A message $msg_{i\rightarrow j}(l)$ represents node x_i's belief about node x_j being assigned with label l. All messages are initialized to some constant depending on the selected message update equation. Given the energy functions $\phi(y_i, x_i)$ and $\phi(x_i, x_j)$, here are three different message update algorithms that can be used for the inference:

Sum-Product algorithm

1. Initialize all messages to 1.
2. Message update summing over all labels l':
$$msg_{i\to j}(l) = \sum_{l'} \exp\left(-\phi(y_i, x_i = l')\right) \exp\left(-\phi(x_i = l, x_j = l')\right) \prod_{k \in N(i)\setminus j} msg_{k\to i}(l')$$
3. Normalization:
$$msg_{i\to j}(l) = \frac{msg_{i\to j}(l)}{\sum_{l'} msg_{i\to j}(l')}$$
4. Belief update:
$$\beta(x_i = l) = \exp\left(-\phi(y_i, x_i = l)\right) \prod_{k \in N(i)} msg_{k\to i}(l')$$

Max-Product algorithm

1. Initialize all messages to 1.
2. Message update taking maximum over all labels l':
$$msg_{i\to j}(l) = \max_{l'}\ \exp\left(-\phi(y_i, x_i = l')\right) \exp\left(-\phi(x_i = l, x_j = l')\right) \prod_{k \in N(i)\setminus j} msg_{k\to i}(l')$$
3. Normalization:
$$msg_{i\to j}(l) = \frac{msg_{i\to j}(l)}{\sum_{l'} msg_{i\to j}(l')}$$
4. Belief update:
$$\beta(x_i = l) = \exp\left(-\phi(y_i, x_i = l)\right) \prod_{k \in N(i)} msg_{k\to i}(l')$$

Min-Sum algorithm

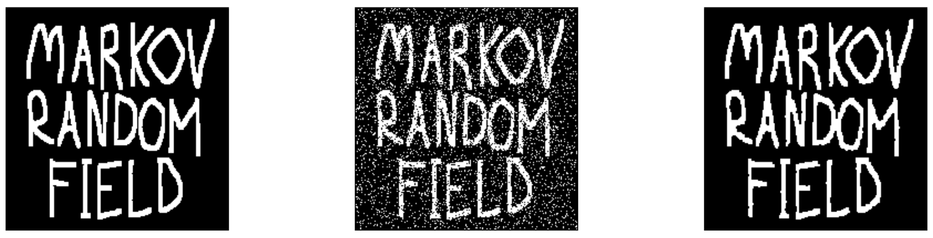
1. Initialize all messages to 0.
2. Message update taking minimum over all labels l':
$$msg_{i\to j}(l) = \min_{l'}\ \phi(y_i, x_i = l') + \phi(x_i = l, x_j = l') + \sum_{k \in N(i)\setminus j} msg_{k\to i}(l')$$
3. Normalization:
$$msg_{i\to j}(l) = msg_{i\to j}(l) - \log\left(\sum_{l'} \exp\left(msg_{i\to j}(l')\right)\right)$$
4. Belief update:
$$\beta(x_i = l) = \phi(y_i, x_i = l) + \sum_{k \in N(i)} msg_{k\to i}(l')$$

3.1.1 Coding Exercise

Problem formulation: Let $\mathbf{x}$ be the true binary image seen in Figure 5a. We add noise to the image by flipping every pixel from intensity 1 to 0 and vice versa with some probability θ, which gives us the noisy image $\mathbf{y}$ (see Figure Figure 5b). We select the factors to be given by $\phi(x_i, y_i) = |y_i - x_i|$ and $\phi(x_i, x_j) = [x_i \neq x_j]$, which results in the energy function

$$E(\mathbf{x}, \mathbf{y}) = \tau \sum_i |y_i - x_i| + \lambda \sum_{i,j} [x_i \neq x_j], \tag{7}$$

where τ and λ are positive constants. The factor $\phi(x_i, x_j)$ is called the Potts model, and $[x_i \neq x_j] = 1$ when the condition is true.



(a) (b) (c)

Figure 5: (a) Original image. (b) Noisy image using probability $\theta = 0.1$ for flipping pixel value. (c) Restored binary image obtained using LBP with min-sum algorithm.

1. Finish the implmentation of LBP by filling in the empty lines of code. Use one of the message updating algorithms outlined above.
2. Plot the inferred binary image and the value of energy function. What parameters τ and λ gives satisfactory results? Search for suitable parameter values in the range $[1, 10]$. How many number of iterations were needed?

3.2 Graph Cuts

Suppose $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ is a directed graph. 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 6a 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 $\mathcal{V}$ 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. Figure 6b illustrates such a cut. A min-cut can be viewed as a binary labeling, where the sum of all costs corresponds to the energy function. The costs between the terminal nodes correspond to $\phi(y_i, x_i)$, while the pairwise costs corresponds to $\phi(x_i, x_j)$. We can therefore use this method for the image denoising task, which we will do in the exercise below. Note that there exists generalizations of min-cut with more terminal nodes involved than s and t, but such generalizations are NP-hard problems [1].

3.2.1 Coding Exercise

Problem formulation: Let $\mathbf{x}$ be the true binary image seen in Figure 7a. We add Gaussian noise to $\mathbf{x}$, which gives us the noisy image $\mathbf{y}$. The factor $\phi(y_i, x_i)$ is given by

$$\phi(x_i, y_i) = \begin{cases} y_i & \text{if } x_i = 0 \\ 255 - y_i & \text{if } x_i = 1 \end{cases}. \tag{8}$$

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. The pairwise cost is given by $\phi(x_i, x_j) = [x_i \neq x_j]$, which results in the energy function

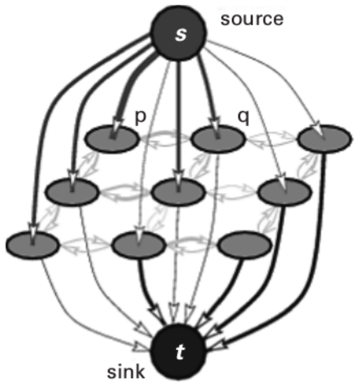
$$E(\mathbf{x}, \mathbf{y}) = \sum_i \phi(x_i, y_i) + \lambda \sum_{i,j} [x_i \neq x_j], \tag{9}$$

where λ is a positive constant.

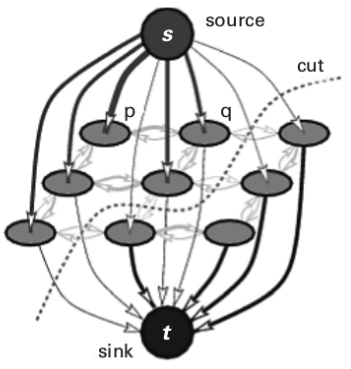
It can be cumbersome to implement an efficient graph cut algorithm. We will therefore use a Matlab package with the Boykov-Kolmogorov algorithm in this exercise[1][2]. This package uses MEX files, which allows running C/C++ code in Matlab, which usually results in faster execution of programs.

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

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



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



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

Figure 6: 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 [1].

We will use the maximum flow algorithm to minimize the energy function E. Note that maximum flow and min-cut are equivalent. 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:

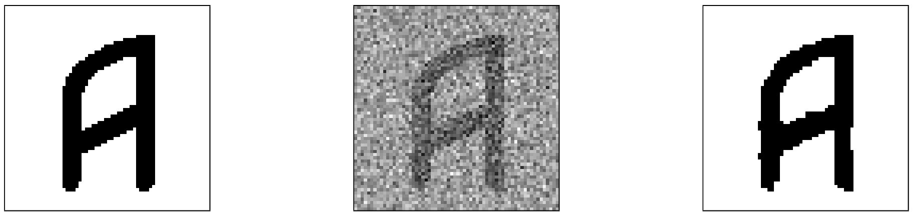
```
[flow, labels] = maxflow(pairwise_terms, unary_terms)
```

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. Complete the following steps:

1. Set up the terms $\phi(x_i, y_i)$ by completing the code lines in `compute_unary_terms.m`. Read through the code in `compute_pairwise_terms.m` which sets up the pairwise factors $\phi(x_i, x_j)$.
2. Plot the inferred binary image. What parameter values for λ gives satisfactory results? Do a coarse search for a suitable parameter value in the range $[1, 200]$. How does varying λ affect the result?

To run the code, the MEX files has to be compiled in Matlab. It might be that you already have a supported compiler installed, otherwise you will have to install one. An easy way out is to find the nearest computer with Linux installed, they commonly have a C++ compiler installed that can be used for compiling MEX fiels in Matlab.

Compile the MEX files by going 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`. If this step succeeds, then the script `image_denoising.m` should run.



(a) (b) (c)

Figure 7: (a) Original binary image. (b) Original image distorted by adding Gaussian noise. (c) Restored binary image obtained using graph cut.

4 Summary and Further Reading

In this tutorial, we introduced the main idea and properties of a Markov Random Field. Moreover, we showed how MRFs can be used for representing images and gave two examples of inference methods – loopy belief propagation and graph cuts – used for these kinds of undirected graphs. The inference methods were applied for solving image denoising problems, where we implemented the setup of the pairwise cost terms.

More examples on applying MRFs to other computer vision problems than image denoising can be found in the book [1]. Graph cuts has been extended to foreground/background segmentation in [2, 7]. This website [3] gives an introduction for using loopy belief propagation in stereo vision, where the task is to do depth estimation in images. Loopy belief propagation is applied in other fields than computer vision, e.g. in [6] where it is used for error-correcting codes on noisy communication channels.

References

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

[2] 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.

[3] Nghia Ho. Loopy belief propagation, markov random field, stereo vision. `http://nghiaho.com/?page_id=1366`. Accessed 2019-01-15.

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

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

[6] Kevin P Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 467–475. Morgan Kaufmann Publishers Inc., 1999.

[7] 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.