Politechnika Łódzka
Instytut Informatyki

PRACA DYPLOMOWA MAGISTERSKA

SEMANTIC IMAGE SEGMENTATION WITH USE OF CONDITIONAL RANDOM FIELDS

Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej
Promotor: dr inż. Arkadiusz Tomczyk
Dyplomant: inż. Aneta Andrzejewska
Nr albumu: 215126
Kierunek: Informatyka
Specjalność: Computer Science and Information Technology

Łódź, 24.02.2019

Instytut Informatyki
90-924 Łódź, ul. Wólczańska 215, budynek B9
tel. 042 631 27 97, 042 632 97 57, fax 042 630 34 14 email: office@ics.p.lodz.pl

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Streszczenie

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Contents

Chapter 1

Introduction

1.1 Purpose and scope of the thesis

[...]

1.2 Goals of the thesis

[...]

1.3 Thesis contents

[...]

Chapter 2

Image Segmentation

As the main goal of the thesis is to perform semantic segmentation of images, this chapter will be focused on providing an explanation of this task and presenting methods that can be used to achieve it.

In order to present the concept of semantic image segmentation, there is a need to define a more general term, which is image segmentation [1]. It is a process aimed to divide an image into regions sharing similar properties, which usually means finding distinct objects and boundaries between them. Depending on an application those partitions will differ, as the goal is to identify regions that provide relevant data for a given problem and this relevance may differ in various applications. Segmentation is often conducted at the preprocessing stage of image analysis, which by grouping pixels into larger regions makes further analysis much simpler and less costly in terms of computations. There are many techniques to tackle the problem of segmentation, however, most of them can be described by one of three categories: threshold, edge-based and region-based methods [2]. Thresholding is the

simplest approach to image segmentation in which individual pixels are assigned to a certain region depending on whether their intensity is in a range of intensities of this region. Edge-based methods are aimed to detect pixels that can be identified as edges between objects, which are characterised by a noticeable change in intensity between neighbouring regions. With all edges detected, it is possible to specify boundaries between objects giving a full segmentation of an image. The last set of techniques, which are region-based methods are aimed to group pixels that have similar properties into sets named regions, which correspond to objects or parts of objects present on an image. This may be obtained by region splitting, which means dividing regions into subregions until all the pixels in a region are homogeneous. An alternative is region growing, which joins neighbouring subregions, if they share similar properties, to obtain larger regions. Very often also a hybrid of these two algorithms is used. Those three methods are the most basic ones when it comes to simple segmentation tasks. Though, for more complicated tasks they are not applicable, they can be used or partially incorporated by more sophisticated algorithms.

2.1 Semantic Image Segmentation

For some applications segmenting an image is not enough, as it may be important to know what each segmented region present. The aim of segmentation is just to partition a given image into regions, without giving information on what these regions mean. The more complicated task of assigning meaning to image regions is named semantic segmentation. It goes a step further into understanding an image, as apart from dividing an image into regions, which correspond to objects, it also recognises

what those regions depict by assigning one of the possible classes to them. Another way of giving a semantic meaning to the image is pixel-wise classification, which means classifying each pixel separately instead of assigning a class to already segmented regions. Either way, the effect is the same. To visualise the objective of semantic image segmentation and example comparing it to ordinary segmentation is presented in Figure 2.1. On the left side, there is a test image presented. It depicts some people standing and other people on bikes in a forest. The middle image shows segmentation of this image into objects. As visible all foreground objects namely people and bikes were detected. However, only the right image shows what is the meaning of segmented regions. Regions marked in pink were classified as people, while green areas were labelled as bikes.

Figure 2.1: *Difference between object segmentation and semantic segmentation. Example image (left), object segmentation (middle), semantic image segmentation (right)*

Source: *The PASCAL Visual Object Classes [3]*

Understanding what objects are present on an image is a key issue in a large variety of practical applications which require image processing. One of the possible uses of semantic image segmentation is in the automotive industry. Thanks to road seg-

mentation, which include obstacle detection and classification, autonomous driving may be possible. It is also applicable to traffic control systems, or smart parking solutions. Similarly, it is needed for robot vision in order to allow decision making based on what is in a field of view of a robot. Furthermore, semantic image segmentation is required for recognition systems for example to provide access to a restricted area based on people biometric features. Similarly, a large set of applications for object recognition and classification is connected with medical imaging, as scanning results are in a form of a substantial number of images, which then have to be analysed by a doctor. Thanks to automatic detection tools, for example to locate tissue pathologies from images, diagnosis can be much easier and less time-consuming. Those are only a few examples of possible applications for semantic image segmentation, however, all of them have one thing in common – they require high precision and accuracy of the results, which can be achieved using various techniques.

2.2 Semantic Image Segmentation methods

For semantic image segmentation, far more complex methods are required than for ordinary segmentation. As in case of any segmentation, there is a need of feature extraction and selection that will allow the distinction between objects. However, apart from specifying object boundaries, those features have to carry enough information to differentiate between objects on a higher level, in order to assign to them a semantic meaning. Conventional approach to this task would be based on manual selection of features. Those features may include previously mentioned colour intensities or detected edges and the same features are used for every pixel in an

image. However, in practice more sophisticated algorithms for feature extraction are preferred. Instead of describing an image as a whole areas significantly different from others are detected. Those areas are named regions of interest, as they are ones on which the segmentation processed is focused. Then, for every such region a vector of important features is constructed with use of feature extraction algorithms. The aim of those algorithms is to take an image as an input and encode information from regions of interest into a series of numbers describing a specific region. They take into consideration features like local spatial information, gradients, histograms, object orientation or texture. Only a proper selection of features will ensure that the segmentation algorithm will be not only able to differentiate one region from the other but also to assign a proper class to each region. For natural images, it is an extremely difficult task as the same object may come in various forms, sizes, shapes, colours etc [4].

Having extracted feature vectors, classification algorithms are aimed to conduct semantic image segmentation basing on them. Traditionally algorithms like Support Vector Machines, Random Decision Forests were involved in this task. Though there are relatively old algorithms, as both were proposed around the 1990s [5, 6], they are still applicable and used for both classification and regression tasks. Support Vector Machine is a supervised learning algorithm which performs classification based on defining decision boundaries between classes. Similarly, Random Decision Forests involve supervised machine learning to solve classification and regression tasks. They are based on composing a classifier ensemble from a set of decision trees, which are independent on each other in a decision making process. It is done in order to obtain more accurate and robust prediction than in the case of a single, complex classifier. Those traditional methods are still applicable for classification tasks, lately more advanced, deep learning methods have become dominant

in algorithms used for semantic image segmentation.

Though, initial approaches to deep learning in this field date back to 2000s [7] a breakthrough of using Deep Neural Networks for image segmentation happened in 2014 [8] when a method of using Fully Convolutional Networks was introduced, which allowed image inputs of arbitrary size as opposed to first implementations that required it to be fixed. Since then, most of the algorithms used for semantic image segmentation have been based on this method. Later in the same year [9], a method combining Fully Convolutional Networks with Conditional Random Fields was proposed, which improved the results by better boundary detection and capturing of details. Conditional Random Fields, which will be further explained in this dissertation can themselves act as an independent classifier that can be used to perform semantic image segmentation.

Chapter 3

Structured Prediction

In various tasks of classification it is not enough to propose an output as a simple scalar value but instead, a complex structure or multiple related outputs like sequences, trees or graphs are needed. One of the examples of such problems is natural language processing, for instance, aimed to assign parts of speech for each word in a sentence [10]. A different example would be a prediction of a structure of biological sequences like proteins or genes [11]. Furthermore, such tasks are common in computer vision to detect and categorise objects present in an image or a video [12]. What is characteristic of all these kinds of applications is that very often the dimensionality of inputs is large.

As a result, a number of potential outcomes can be enormous, making the classification computationally highly demanding. A framework that allows solving such problems is called structured prediction. It is a technique involving supervised machine learning algorithms in which the output is treated as one object instead of a set of individual values. Such an object has a complex structure containing a number of

related entities, which can involve constraints or dependencies. Taking into account those relations is a key concept of structured prediction.

In this chapter explanation of definitions and terms connected with structured prediction will be provided, which are necessary to understand this dissertation. First, the concept of probabilistic graphical models will be described, followed by an overview of the factorisation process. In the next section, Conditional Random Fields, which are a part of the topic of this dissertation, will be introduced. After that, a process of inference in factor graphs and parameter training will be described. Last, but not the least an overview of different methods of energy formulation will be provided.

3.1 Probabilistic Graphical Model

In machine learning very often there is a need to construct a mathematical model that represent real-life objects in terms of data essential for a given problem. An example of such representations is a graphical model, which reflect not only relations between observations and quantities of interest but also between different observations. They offer a compact and intuitive way of representing a large set of related variables, which is of high use in structured prediction.

A graphical model is composed of a set of nodes being a reflection of each random variable. If two or more variables are related, then their associated nodes are connected with an edge.Furthermore, with each edge, there may be a vector of parameters, or weights assigned which describe how much one variable is depended on the other. Depending on an application, elements of a graph represent different

objects. Furthermore, a graph can model also other information than relations between nodes. Giving an example, for image segmentation each node represents a pixel in an image, and edges describe relations between individual pixels. Furthermore, there can be an additional layer of nodes created, with each node representing a class label. Then, between a pixel node and its label node there is also an edge.

The goal of constructing graphical models is to propose a data representation that will allow finding a solution for the specified problem in a clear and direct way. However, for some applications, especially for classification, it is impossible to propose a well-defined model that will give an answer with certainty. In such situations, it is beneficial to propose a representation of an object that encodes not direct, but statistical dependencies between observations and quantities of interest. Probabilistic graphical models offer such functionality. By using them it is possible to obtain not a single, definite answer but a full probability distribution of answers, being a mathematical function representing each possible result with a probability of its occurrence.

Depending on a kind of a graph this probability is modelled in a different way. There are two main graph types, directed and undirected graphical models. Directed graphs, also known as Bayesian networks, have edges that are directed from one node to the another, meaning that there is a one-way dependence between variables. Using those graphs causality between different variables can be modelled in a clear way. Bayesian networks encode conditional probability of every label of a node given that its parent nodes have a given label assigned. However, in many applications, especially in computer vision, variables do not have this kind of interactions. This is also a case in image processing, as there is no hierarchy between pixels in

an image. Such problems can be modelled with undirected graphs named Markov Random Fields, which form a base of representing a problem domain in structured prediction. In this case, instead of modelling conditional probability based on parent nodes a joint probability distribution is modelled, which encodes the probability of every label of each variable occurring simultaneously. For large input space, this involves a lot of data processing. In order to store information about it in an explicit way, for example in the form of a table, an enormous number of cells would be needed as every combination of all the variables' possible values should be stored in a separate cell. Because of the fact that in structured prediction a quantity of variables is typically large, there is a need to propose different methods of encoding joint probability distributions and graphical models offer such functionality. In Markov Random Fields, all not connected variables are assumed to be conditionally independent, which means that variables depend only on their closest neighbours. Because of this, it is possible to decompose large distribution functions into smaller distributions containing only related variables, which reduces the number of data needed to be stored [13].

3.2 Factorisation

A process of decomposing a joint probability distribution is named factorisation. As a result of it, a set of factors is defined, which product is equal to a given joint probability distribution. Factorisation can largely decrease the complexity of calculations as it is only required to calculate the outcome of each factor within its scope being usually much smaller than the whole input domain. A base for defining a factor in undirected graphs is a clique that is a subset of interconnected nodes, in

which each variable is connected to every other variable in a set. A clique which is not a subset of any other clique and therefore cannot be extended by adding any more nodes, is called a maximal clique [14]. Given an undirected graph $G = (V, \mathcal{E})$ modelled with a set of nodes, also called vertices V and a set of edges $\mathcal{E}$ a family of joint probability distributions can be obtained as a result of the factorisation process. A joint probability $p(y)$ is understood as a normalised product of factors ψ_c of every clique c from a set of all maximal cliques found in the graph $C(G)$, where y_c denotes all variables nodes from a given clique c, as in equation 3.1.

$$p(y) = \frac{1}{Z} \prod_{c \in C} \psi_c(y_c) \tag{3.1}$$

In those graphs, factors also called clique potentials, do not represent conditional probability. Instead, they are non-negative functions with a property that values with higher potentials are more probable to occur. Because of this, a family of joint probabilities does not need to sum up to 1. That is why a normalising constant Z is introduced. This constant is also known as a partition function and it is expressed as a summation over joint probabilities of all states y in an overall output domain $\mathcal{Y}$.

$$Z = \sum_{y \in \mathcal{Y}} \prod_{c \in C} \psi_c(y_c) \tag{3.2}$$

Factorisation can be presented on a graph which is fully connected, however, it is not very convenient. A far more intuitive way of modelling factorisation of a joint probability distribution is to use factor graphs, which are a type of undirected, probabilistic graphical models. Such graphs contain two kinds of nodes – variable nodes, and factor nodes, with each factor node being connected to variable nodes that depend on it. An exemplary factor graph is depicted in figure 3.1, where variable nodes are marked as green circles, and factor nodes as black rectangles. As presented between every pair of factor and variable nodes there is an edge marked.

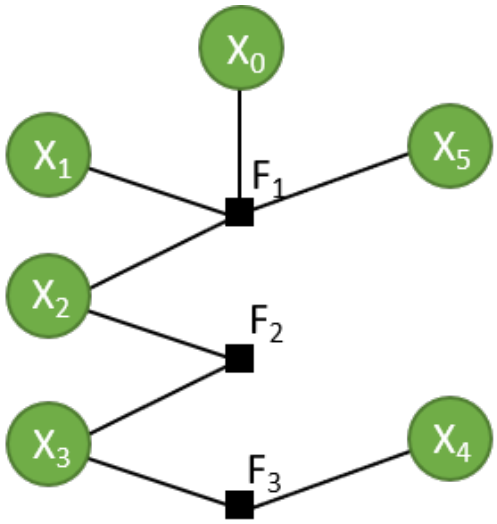


Figure 3.1: *Example of a factor graph with variable nodes marked as green circles and factor nodes as black rectangles*

Then to calculate joint probability $p(y)$ it is enough to compute a product of each factor $F \in \mathcal{F}$ and normalise it with normalising constant Z as in equations 3.3 and 3.4. The scope of the factor F is denoted as $N(F)$ and it represents the set of all variable nodes that are adjacent to this factor.

$$p(y) = \frac{1}{Z} \prod_{F \in \mathcal{F}} \psi_F(y_{N(F)}) \tag{3.3}$$

$$Z = \sum_{y \in \mathcal{Y}} \prod_{F \in \mathcal{F}} \psi_F(y_{N(F)}) \tag{3.4}$$

Hence, for the sample factor graph from figure 3.1 the joint probability $p(y)$ is equivalent to product of factor F_2 dependent on nodes x_5, x_0, x_1, x_2, factor F_2 with x_2, x_3, and factor F_3 dependent on x_3, x_4.

$$p(y) = \frac{1}{Z}(\psi_{F_1}(x_5, x_0, x_1, x_2, y_{N(F_1)}) \cdot \psi_{F_2}(x_2, x_3, y_{N(F_2)}) \cdot \psi_{F_3}(x_3, x_4, y_{N(F_3)})) \tag{3.5}$$

For directed graphs it was clear that factors represent conditional probability, however, for undirected graphs, the meaning of factors is not defined. The only requirement for them is to be non-negative as probability cannot be negative. One of the easiest ways of ensuring that factors will output a positive value is to present a problem in an exponential domain, with use of energy function. Energy is such a function which exponential represents a factor potential as in equation 3.6.

$$\psi_F(y_{N(F)}) = \exp\Big(-E_F(y_{N(F)})\Big) \tag{3.6}$$

Then, the joint probability $p(y)$ can be expressed as a summation of energies for each factor as in equation 3.7.

$$\begin{aligned} p(y) &= \frac{1}{Z}\prod_{F\in\mathcal{F}}\Big(\exp\Big(-E_F(y_{N(F)})\Big)\Big) \\ &= \frac{1}{Z}\exp\Big(-\sum_{F\in\mathcal{F}}E_F(y_{N(F)})\Big) \end{aligned} \tag{3.7}$$

The normalising constant Z has the following form:

$$Z = \sum_{y\in\mathcal{Y}}\exp\Big(-\sum_{F\in\mathcal{F}}E_F(y_{N(F)})\Big) \tag{3.8}$$

Similarly to a factor potential, energy represents how accurate is a given solution [15]. The evaluation is done by assigning a scalar value to each configuration of random variables with an interpretation that the lower an energy is, the more probable is a given configuration. Then to obtain a state $y \in \mathcal{Y}$ with the highest probability it is enough to specify a configuration with the lowest energy as in formula 3.9.

$$\arg\max_{y\in\mathcal{Y}} p(y = Y) = \arg\min_{y\in\mathcal{Y}} \sum_{F\in\mathcal{F}} E_f(y_{N(F)}) \tag{3.9}$$

Factorisation is a highly convenient process for supervised learning in which the number of inputs and possible outputs is extensive, as it reduced the complexity of

the generated model. By considering the problem in terms of factor energy, finding the most likely solution for a problem is interchangeable with identifying the state for which energy of all factors is the lowest, which is much less computationally demanding. Furthermore, factor graphs increase the efficiency of algorithms further described in this dissertation, which are needed to solve prediction problems.

3.3 Conditional Random Fields

For the task of classification, in which observations are globally known it is not necessary to model joint probability distribution over all possible outputs and observed values. Instead, a conditional distribution $P(y|x)$ is modelled, which represent a distribution of output variables on condition that input variables take an observed form. One of the methods that aim to to find the conditional distribution between variables is named Conditional Random Fields [16]. This is a type of Markov Random Fields that can be viewed as undirected graphical models, which represent both observations, and output labels. With the use of factor graphs, factorisation of a probability distribution can be modelled. Then, those graphs contain two different types of variable nodes, inputs and outputs [17]. The first ones are observed input variables x_i, which are dependent on an application. For semantic image segmentation, x_i would denote individual pixels from a single image x taken from the set of all available images X. Input nodes are represented in terms of feature vectors containing observed properties of an individual image pixel expressed in a numerical form. A great advantage of Conditional Random Fields is that they can involve a large variety of different features. Input variable nodes are connected via factors to the second type of nodes, which are known as hidden nodes, as they cannot be

directly obtained from an observed object. Those nodes represent a configuration of output variables denoted as y. For the task of semantic image segmentation, an output variable y_i would represent a label assigned to a given pixel from a set of all possible labels $\mathcal{Y}$, and y would represent a labelling for a whole image. Output nodes are as well connected with each other with the use of factors making it possible to take into consideration relations between neighbouring entities like pixels in an image. Figure 3.2 presents how such a graph can be constructed on an example of an image with size 3×3 pixels.

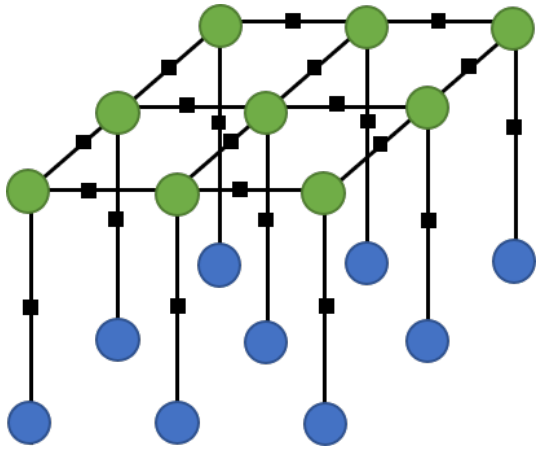


Figure 3.2: *Factor graph representing an image* 3×3 *pixels with input nodes marked in blue, and output nodes in green*

On the graph, blue circles represent input nodes with observed variables, and green circles a hidden layer of output nodes. Factors, which join each output node to the corresponding input node as well as every neighbouring output nodes are marked as black squares. Between each pair factor – node there is an edge presented as a line, which connects them. This is the simplest type of the graph with relations only between input and output nodes, and neighbouring output nodes. However, it is also possible to introduce other dependencies, for example between an output node

and observation nodes related to neighbouring output nodes. For the sake of clearer explanation in following equations it will be assumed that there are relations between one input and one output node, and between two neighbouring output nodes only.

Conditional Random Fields model distribution $P(y|x)$ of a single labelling y based on observed features of an image x. This distribution can be expressed in terms of energy as in formula 3.10, with constant Z computed as in equation 3.11.

$$P(y|x,w) = \frac{1}{Z(x,w)} \exp\left(-E(x,y,w)\right) \tag{3.10}$$

$$Z(x,w) = \sum_{y \in \mathcal{Y}} \exp\left(-E(x,y,w)\right) \tag{3.11}$$

Then, assuming an underling factor graph, energy of the whole system can be expressed as a summation of energies of individual factors, as it has already been presented in formula 3.7. However, for Conditional Random Fields a factor graph is composed of two types of nodes, input and output nodes. As a result, there are two sets of factors, a set $\mathcal{F}_1$ of factors between an output node and an input node, and a set $\mathcal{F}_2$ of factors between two neighbouring output nodes. Hence, the energy of the whole system can be expressed as two summations, one over the set of factors $\mathcal{F}_1$ and the second one over a set $\mathcal{F}_2$. As each factor $F \in \mathcal{F}_1$ is bound exactly to one input node and one output node the first summation over the set $\mathcal{F}_1$ may be expressed as a summation over all output nodes $i \in V$, which will compute energies for factors that are between the current output node an an input node assigned to it. Similarly, summation over factors from $\mathcal{F}_2$ is equivalent to summation over every pair of output nodes $i, j \in V$ that will compute energy for a factor that is between them. Hence, the formula for energy for a factor graph modelling Conditional Ran-

dom Fields is presented in formula 3.12.

$$\begin{aligned} E(x,y,w) &= \sum_{F\in\mathcal{F}_1} E_F(x_F,y_F,w) + \sum_{F\in\mathcal{F}_2} E_F(y_F,w) \\ &= \sum_{i\in V} E_1(y_i,x_i,w) + \sum_{i,j\in V} E_2(y_i,y_j,w) \end{aligned} \tag{3.12}$$

Factor energy is defined differently depending on the factor type. Energy E_1 of factors $\mathcal{F}_1$ is called a unary potential. It models a relationship between observed features and a predicted label. It should assign high values of energy for a wrong label assigned to an output node of a given input node, and low energy values for proper assignments. It is used to promote a situation in which two nodes which are similar in terms of their feature vectors have the same label. Energy E_2 for factors from the set $\mathcal{F}_2$ is named a pairwise potential and it reflects a relation between labels of neighbouring output nodes. It introduces a penalty if neighbouring output nodes have a different label by assigning high energy values to such configurations, which as a result lower probability of their occurrence. What is more, factor energy can be parametrised with a set of weights w, which can be obtained by a training process. Then, energy is dependent on three types of variables, inputs, outputs, and parameters [18], and can be expressed as . A more detailed explanation of energy function and its formulation will be provided in section *3.6 Energy formulation*.

In order to make predictions about an unobserved object, it is necessary to find such a combination of output variables in a factor graph that will give the lowest energy as this would be a most probable setting. However, it is computationally intractable to calculate graph energy for every possible label configuration especially for problems involving large data structures. On an example of semantic image segmentation, even for a small problem of segmenting an image with size 100×100 pixels into two classes, it would require to calculate energy for $2^{100\times100}$ combinations and real-life problems are much more complicated. Because of this, in most

cases, it is impossible to obtain an exact solution for classification problems based on factor graph energy. Though, it is possible to approximate a solution.

3.4 Inference in factor graphs

A correctly defined model, either by training or by expertise, is used to make predictions about the unobserved data in a process called inference. There are two most popular approaches to this task, marginal, and maximum a posteriori (MAP) inference. Marginal inference is aimed to compute marginal distributions, which give a probability of a variable taking a given value as opposed to every other possibility. An example of it would be to segment an image into foreground and background, where one object is considered foreground and all other objects are assigned as background. On the other hand, a goal of MAP inference is to find the most probable configuration of unobserved variables' values basing on the values of observed data. Both types of inferences are used in factor graphs and consist of establishing observed data and finding such values of output variables that minimise the total energy. In the thesis for the task of minimising the energy MAP inference was used. The reason behind this is choice is that MAP inference outputs directly a state $y* \in \mathcal{Y}$ that has the maximal probability of occurrence given an underling factor graph, observation x, and learned weight vector w and this is the goal of semantic image segmentation.

State $y*$ represents the most optimal configuration of labels of each individual pixel. Finding the optimal state would require calculating energy for each possible configuration of outputs. This is too computationally demanding to be performed, however, there are methods to provide the final result without a need of extensive

calculations. Two most popular types of methods that are used for inference are Monte Carlo methods and variational algorithms [17]. Monte Carlo algorithms are a type of stochastic algorithms which instead of estimating conditional probability distributions for a given test object base the approximation on a repetitive generation of random samples from a distribution of interest. The idea is that given enough number of samples their mean will approximate desired output. On the other hand, variational algorithms are aimed to find a single configuration that is an approximation of a most probable output, by changing inference into an optimisation problem. Though Monte Carlo methods are more accurate with a large number of iterations they will converge to the real result, variational algorithms tend to be much faster.

One of the most popular approaches towards variational inference in probabilistic graphical models is named Belief Propagation. It is an iterative framework based on message passing that exchange data between variable and factor nodes [19]. Those messages, also named beliefs, are vectors that can be understood as a measure of certainty of related nodes that other nodes belong to some state. It is based on a concept of dynamic programming, which instead of doing calculations for each factor independently uses a recursive procedure to propagate beliefs throughout the whole graph. Recursive nature reduces the number of computations because of reusing previously computed data. Belief Propagation was first used to find exact marginals on tree [20], tough it is also applicable in a general case of graphs that may contain loops, however, without a guarantee of finding a global optimum. This method is applicable to both of the previously mentioned inference problems. Sum-product algorithm is used for computing marginal distribution, and max-product algorithm for solving MAP inference.

3.4.1 Sum-Product algorithm

Sum product algorithm is method used to compute marginal distributions of a given test sample and a normalising constant Z with the use of a trained model. Those outcomes can be described by formulae 3.13 and 3.14.

$$p(Y = y|x, w) = \frac{1}{Z} \exp\left(- \sum_{f \in \mathcal{F}} E_f(y_f) \right) \tag{3.13}$$

$$Z = \sum_{y \in \mathcal{Y}} \exp\left(- \sum_{f \in \mathcal{F}} E_f(y_f) \right) \tag{3.14}$$

In Sum-Product algorithm, factor energy can be computed based on information sent from adjacent nodes in a form of messages, which are aimed to propagate observed data throughout the whole model. They are sent through each edge of a factor graph in both directions. The messages sent from variable nodes to factors are denoted as $q_{Y_i \to F}$ and factor-to-variable message can be expressed by $r_{F \to Y_i}$, where i is an index of a variable node [21]. Variable-to-factor messages depend on previously calculated factor-to-variable messages that are associated with factors adjacent to this variable. It can be expressed as a summation of messages from each factor F' from the variable neighbourhood M with an exception of the factor F to which the initial message is directed. Such a belief is calculated for each variable node i and for every label y from the output domain of a given problem as in formula 3.15.

$$q_{Y_i \to F}(y_i) = \sum_{F' \in M(i) \setminus \{F\}} r_{F' \to Y_i}(y_i) \tag{3.15}$$

Similarly, when it comes to computing messages from factors to variables there is a summation of messages from each variable j from the neighbourhood N of the factor F apart from a variable i which is a recipient of the initial message. Furthermore, factor-to-variable messages take into consideration factor potentials expressed in

terms of energy. These messages are again calculated for every label y separately. Given the fact that one component of factor energy is a pairwise potential, which is dependent on labels of every variable connected to the factor, while computing these messages there is an extra summation over all possible states of neighbouring variables. This is needed to calculate energy for every factor state y'_F. This state can be understood as a label configuration of variables adjacent to the factor knowing that the variable to which the message is directed has been labelled with y. A formula for a factor-to-variable message is presented below.

$$r_{F\to Y_i}(y_i) = \log\left(\sum_{\substack{y'_F\in\mathcal{Y}\\ y'_i=y_i}} \exp\left(-E_F(y'_F) + \sum_{j\in N(F)\setminus\{i\}} q_{Y_j\to F}(y'_j)\right)\right) \tag{3.16}$$

Both of those kinds of messages are depended on previous computations, however, for trees, there are two situations in which they can be calculated without prior knowledge. For leaf nodes that are only connected to one factor, a variable-to-factor message will be equal to 0, as the neighbourhood $M(i)\setminus\{F\}$ will have no elements. Message from factor to variables can be also calculated for factors that are only connected to one variable. In such a situation neighbourhood $N(F)\setminus\{i\}$ will have 0 elements and consequently only an energy term is needed for the message to be calculated. If a graph contains no cycles, using those calculated beliefs recursively other messages can be computed [22]. For tree-like structures, message computations start with choosing an arbitrary node as a root node. Then, there is a two-step procedure, which is forward and backward propagation of beliefs. Initially, starting from leaf nodes, messages are computed and passed to the root node. Messages are transferred from every node to its parents after all the messages from the node's children are received. They carry information about an optimal label of the root node. After all messages are computed the propagation starts again but in

reverse direction, which is from the root to all leaves. In this step, optimal labels for all non-root nodes are predicted. In those two steps, messages of every edge of the tree-like graph are established and can be used to calculate marginals. Then for every variable node marginals can be computed based on factor-to-variable messages from all adjacent factors. Again, normalisation with a partition function Z is needed for the marginals to carry probabilistic information. It is defined based on the summation of factor-to-variable messages directed towards the root node r for every possible label y. Formulae below are used to compute both marginals and their normalising constant.

$$p(Y_i = y_i) = \exp\left(\sum_{F \in M(i)} r_{F \to Y_i}(y_i) - \log Z\right) \tag{3.17}$$

$$\log Z = \log \sum_{y_r \in \mathcal{Y}} \exp\left(\sum_{F \in (r)} r_{F \to Y_r}(y_r)\right) \tag{3.18}$$

To obtain a final result, which is the most probable configuration of labels for a set of variables $i \in V$ it is enough to calculate marginals for each variable and every possible label y and to chose the one with higher probability.

Tough Belief Propagation was designed to be used for non-cyclic, tree-like structures it is also possible to apply its concepts to the general case of graphs that may contain cycles, which are the most common type in computer vision tasks [23]. With graphs that contain loops it is impossible to determine a root node, hence, a previously stated algorithm for exact determination of marginals is inapplicable, however, it can be modified to allow approximating marginals for graphs with undirected cycles. In this algorithm, named Loopy Belief Propagation, messages are first initialised with random variables and then iteratively updated until they converge. This method is based on a concept that the local neighbourhood of nodes can be viewed as a tree, and as a result, message passing algorithms are applicable to

it. Message passing can be done in parallel or in a sequential scheme, in a random or predefined order [24]. For parallel processing computation of messages for every edge is done simultaneously and after that, they are propagated to neighbours. A sequential scheme is more similar to the original Belief Propagation method as initially messages for one node are computed and then propagated to neighbouring nodes and used for further computations. Though formula for factor-to-variable message $r_{F \to Y_i}$ is the same as in an original algorithm, the equation for messages from variable to factors is modified to contain a normalising factor.

$$\bar{q}_{Y_i \to F}(y_i) = \sum_{F' \in M(i) \setminus \{F\}} r_{F' \to Y_i}(y_i) \tag{3.19}$$

$$\delta = \log \sum_{y_i \in \mathcal{Y}} \exp\left(\bar{q}_{Y_i \to F}(y_i)\right) \tag{3.20}$$

$$q_{Y_i \to F}(y_i) = \bar{q}_{Y_i \to F}(y_i) - \delta \tag{3.21}$$

In Loopy Belief Propagation the exact marginal distribution is not known, however, it is possible to compute approximate marginals. Furthermore, as in this algorithm a notion of a root node does not exist it is not possible to calculate the exact partition function. That is why local normalising constants are used. The final equation for approximate marginals is also normalised according to formulae presented below.

$$\bar{\mu}_i(y_i) = \sum_{F' \in M(i)} r_{F' \to Y_i}(y_i) \tag{3.22}$$

$$z_i = \log \sum_{y_i \in \mathcal{Y}} \exp\left(\bar{\mu}_i(y_i)\right) \tag{3.23}$$

$$\mu_i(y_i) = \exp\left(\bar{\mu}_i(y_i) - z_i\right) \tag{3.24}$$

Using the presented method it is possible to find the most probable label for each node of a factor graph even for non-tree-like structures.

3.4.2 Max-Product algorithm

Finding the most probable label for each node individually may not result in the most optimal label configuration of the whole structure. MAP inference, with an example of Max-Product algorithm of Belief Propagation, addresses this problem. It is similar to the Sum-Product algorithm, as it is also based on message propagation between nodes and is exact for trees and gives an approximate result for general graphs. Max-Sum and Min-Sum versions of this algorithm operate in logarithmic space, using energies instead of factor potentials. The difference between Max-Product and Sum-Product algorithms lies in the computation of factor-to-variable messages, as depending on an algorithm variation either maximisation or minimisation over factor states is performed instead of marginalisation. For Max-Sum inference, the formula for messages from a factor to a variable is as follows.

$$r_{F\to Y_i}(y_i) = \max_{\substack{y'_F\in\mathcal{Y}\\ y'_i=y_i}} \left(-E_F(y'_F) + \sum_{j\in N(F)\setminus\{i\}} q_{Y_j\to F}(y'_j) \right) \tag{3.25}$$

Similarly, in a Min-Sum variant of this algorithm, factor-to-variable messages can be computes as in equation 3.26.

$$r_{F\to Y_i}(y_i) = \min_{\substack{y'_F\in\mathcal{Y}\\ y'_i=y_i}} \left(E_F(y'_F) + \sum_{j\in N(F)\setminus\{i\}} q_{Y_j\to F}(y'_j) \right) \tag{3.26}$$

When it comes to computations of variable-to-factor message the only difference is in a formulation of normalising factor, which instead of computing logarithm of a sum of messages in exponential form, takes the average value of those mes-

sages.

$$\bar{q}_{Y_i \to F}(y_i) = \sum_{F' \in M(i) \setminus \{F\}} r_{F' \to Y_i}(y_i) \tag{3.27}$$

$$\delta = \frac{1}{|\mathcal{Y}|} \sum_{y_i \in \mathcal{Y}} \bar{q}_{Y_i \to F}(y_i) \tag{3.28}$$

$$q_{Y_i \to F}(y_i) = \bar{q}_{Y_i \to F}(y_i) - \delta \tag{3.29}$$

Computing of all messages for every edge of the graph allows finding such a configuration of labels, which has the highest probability of occurrence. The optimal prediction y^* is obtained by determining a state which has the maximal belief for every variable, which is equivalent to finding the state of lowest energy.

$$y_i^* = \arg\max_{y_i \in \mathcal{Y}} \mu_i(y_i) \tag{3.30}$$

$$\mu_i(y_i) = \sum_{F' \in M(i)} r_{F' \to Y_i}(y_i) \tag{3.31}$$

Presented inference algorithms allow predicting the optimal configuration of labels for unclassified test samples given the observed data and an established model in a form of a factor graph. The model can be defined by expertise, however, usually it is trained based on known train samples.

3.5 Parameter training

Parameter estimation, also called parameter training, is a process that adjust weights of the model so that it reflects real dependencies between input and outputs. In Conditional Random Fields properly estimated weights are necessary to compute factor

energy during an inference process. In order to train parameters of the model a supervised learning is used, which is a machine learning task that uses a ground truth to optimise an objective function so that it will be able to correctly map input into outputs. For the task of semantic image segmentation, ground truth is a set training samples which is composed of properly labelled images. Having them it is possible to construct such a factor graph in which not only features for observed variable nodes X are known, but also labels for hidden nodes Y. Similarly to the task of inference, there are two main ways of parameter learning, first involving computation of marginal distributions named Maximum Likelihood Learning and the second requiring only MAP labelling [19]. The latter method known as Max-Margin Learning has recently gained recognition as an alternative towards Maximum Likelihood Learning, however, it is based on Support Vector Machines and not on Conditional Random Fields, which are the key topic of this dissertation.

Maximum Likelihood Learning is a standard and most popular approach to parameter learning in Conditional Random Fields. In order to perform any training process to find optimal parameters of the model an objective function is required. This is a function that evaluates the ability of the model to properly map inputs into outputs given a chosen set of parameters. In Maximum Likelihood Learning the objective function is called likelihood and it is used to maximise conditional probability $P(y|x,w)$ based on training objects with known inputs x and outputs y, and unknown weight vector w. Usually, the concept of log-likelihood is used instead of likelihood, as computations in logarithmic scale tend to be less complex. Furthermore, knowing that the state with the lowest energy of a factor graph is, in fact, the most probable state, the task of maximising probability is equivalent to minimising energy. Hence, the objective function for Maximum Likelihood Learning is expressed as a negative log-likelihood denoted as $\mathcal{L}$. Hence, basing on the

definition of conditional probability $P(y|x,w)$ presented in equation 3.10 that can be transformed into logarithmic scale, the objective function $\mathcal{L}$ needed for the process of supervised learning can be expressed as in equation 3.32.

$$\mathcal{L}(w) = \lambda \|w\|^2 + \sum_{n=1}^{N} \Big(E(x^n, y^n, w) + \log Z(x^n, w) \Big) \tag{3.32}$$

Energy of a single sample can be expressed as a linear relation between a current weight vector w and a function φ that is dependent on inputs and outputs of the factor graph constructed for this sample, as presented in formula 3.33. Then, the partition function Z can be also expressed in terms of this relation, as in equation 3.34.

$$E(x^n, y^n, w) = \langle \varphi(x^n, y^n), w \rangle \tag{3.33}$$

$$Z = \sum_{y \in \mathcal{Y}} \exp\Big(- \langle \varphi(x^n, y^n), w \rangle \Big) \tag{3.34}$$

Apart from an energy term calculated for each of N training samples and their normalising constants Z, in the objective function there is also a regularisation term λ. Regularisation is one one of methods used to prevent overfitting of the model. Its aim is to penalise large values of weights so that less complex models are learned. As λ is a hyperparameter of an objective function, it should be additionally chosen using for example cross-validation. Then, the process of parameter learning is aimed to find such a weight vector w^* for which the objective function will be minimal, as in equation 3.35.

$$w^* = \arg\min_{w} \left(\lambda \|w\|^2 + \sum_{n=1}^{N} E(x^n, y^n, w) + \sum_{n=1}^{N} \log Z(x^n, w) \right) \tag{3.35}$$

Finding a minimum of a function is a typical task in machine learning and can be solved with various methods. The fundamental and most straightforward method of solving optimisation problems is named Gradient Descent [25]. It is an algorithm

that updates parameters of an objective function in such a way to minimise an error between expected outputs of training examples and the actual output of the trained function. It is done by iteratively moving into a direction specified by a negative of a gradient of the objective. The algorithm starts by assigning arbitrary values to the parameters of the function and changing them slightly in each iteration, according to the calculated gradient. At each iteration $t+1$ parameters are updated by subtracting part of a value of the gradient. The size of this part is defined by a hyperparameter α named step size, which can be either constant or changeable throughout a process of learning. Weight update is done according to the formula 3.36.

$$w_{(t+1)} = w(t) - \alpha_t \nabla \mathcal{L}(w_t) \tag{3.36}$$

The process continues until the error is small enough to state that a local minimum, or global in case of convex functions, was reached. To apply the Gradient Descent method in an optimisation problem, an objective function has to be differentiable. Fortunately, a regularised conditional probability specified that was presented in formula 3.32, is a smooth and convex function, hence Gradient Descent method can be used to find a global minimum of it.

During the training process, in each iteration a value of gradient $\nabla \mathcal{L}(w)$ needs to be computed in order to update weights of the model. This requires summation over all training samples, which for large training sets is highly time consuming. Furthermore, to obtain the part of the gradient that is associated with a partition function also a summation over all possible configurations of labels for every pixel is needed, which even for small images with only few classes requires a lot of calculations. Formula for gradient calculation is presented in 3.37.

$$\nabla_w \mathcal{L}(w) = 2\lambda w + \sum_{n=1}^{N} \left(\varphi(x^n, y^n) - \log \sum_{y_z \in \mathcal{Y}} \exp\left(\varphi(x^n, y_z^n) \right) \right) \tag{3.37}$$

Fortunately, it is not necessary to directly compute the gradient, as there are methods that are able to approximate it. Similarly as in the case of the inference problem, a Loopy Belief Propagation can be used to estimate the marginal probability and therefore the gradient needed for parameter update. However, this would mean that in each iteration the Loopy Belief Propagation has to be performed, which require a lot of time and resources. Instead, the second type of algorithms that have already been introduced in section *3.4: Inference in factor graphs*, namely Monte Carlo methods, can be applied. One of such methods is a stochastic variant of the Gradient Descent algorithm which is used when the number of training examples is too high to compute the gradient efficiently [26]. The key idea of Stochastic Gradient Descent is to update the parameters of the objective function not after processing the whole training set but after a random batch of training examples, or only after just one example. In this way, the gradient is not computed exactly but only estimated. Tough this estimate is usually very rough and more iterations are required to converge, the simplification introduced by this method largely reduce the time of each iteration making the whole algorithm much faster. However, even if the sum over all training examples is not required, still in order to obtain the partition function the summation over all configurations of labels is needed, which is computationally infeasible. Fortunately, similarly as in case of the gradient, the partition function does not need to be computed directly. Instead, an estimation is proposed, which is based on an assumption that large populations usually have a lot of redundancy as they tend to contain very similar objects. Hence, instead of taking into account the whole population it is enough to propose such a set of samples which will reflect the probability distribution of the population. This concept is based on the notion of Monte Carlo integration [27], which states that for sufficiently large number of independent samples S summation of outputs for each sample is roughly equal to

the integral of the function, as in equation below, with $p(x)$ denoting distribution of chosen samples.

$$\int_a^b f(x)p(x)dx \cong \frac{1}{S}\sum_{s=1}^{S} f(x_s) \tag{3.38}$$

The main issue of this idea is that it is not known what is the required number of samples for the distribution to reflect the whole population. However, there are techniques named Markov Chain Monte Carlo methods to generate such samples that will fulfil the Monte Carlo integration principle. One of the most popular examples of thee methods is named Gibbs Sampling. It is an algorithm that is used to create mutually dependent samples form the probability distribution of an underlying model. It works by constructing a Markov Chain based on a conditional distribution of random variables. A Markov Chain is a model representing a sequence of states, through which it is possible to move freely with a probability depending only on one, previous state. Gibbs Sampling is an iterative algorithm which initially creates a random Markov Chain representing a configuration of labels for a given object. Then, in each iteration, every variable is replaced by the most probable state, given the conditional probability on all the other variables. Hence, for every possible label of a given variable, an energy of the whole configuration is calculated and a label with the lowest energy, meaning the highest probability of occurrence, is chosen to be involved in the final sequence. After every variable is processed, the resulting Markov Chain forms one sample, which is then used to initialise variables in a sequence for the next sample. This forms a set of Gibbs samples created in such a way that they can be used to estimate the part of the gradient associated with the partition function Z. Thus, the formula for gradient estimation during a Stochastic Gradient Descent after the process of Gibbs Sampling is presented in 3.39.

$$\nabla_w \mathcal{L}(w) = 2\lambda w + \varphi(x^n, y^n) - \log\left(\frac{1}{S}\sum_{s=1}^{S}\exp\left(\varphi(x^s, y^s)\right)\right) \tag{3.39}$$

The process of computing sampling distribution is far less demanding than the computation of full conditional distribution, as instead of taking into consideration every possible configuration of labels it is enough to process distribution of only one random variable at a time. However, if a chosen set of samples S is large, in every iteration still a lot of calculations are needed to be performed. Tough in this step yet another simplification can be introduced. For a set of samples containing only one element the number of computations is largely reduced. As an effect, again an estimate of a gradient will be very rough. However, as each Gibbs sample is created based on the previous sample with every iteration it is more close to the approximation of the true distribution. Hence, given a large number of iterations, the algorithm will eventually reach a direction which is an approximation of the true gradient. Hence, the final formula to compute the gradient needed for parameter update is presented in equation 3.40.

$$\nabla_w \mathcal{L}(w) = 2\lambda w + \varphi(x^n, y^n) - \varphi(x^s, y^s) \tag{3.40}$$

Typically Gradient Descent Methods tend to choose the shortest path towards the minimum of the objective function. The key idea behind all the presented simplifications is not find the shortest path, but instead to make more steps that are less accurate but easier to compute. Both approaches are aimed to converge near the global minimum, though the second one is faster are more applicable for problems with a lot of training data and large output space. Having defined the objective function and a method to estimate the gradient it is possible to efficiently perform a supervised parameter learning.

3.6 Energy formulation

A basis of both inference and parameter training is a proper formulation of an energy function, which reflects a probability of occurrence of a given configuration of variables in a system. In semantic image segmentation, without correctly defined energy function it will not be possible to differentiate objects of different classes. As it has already been presented, energy is composed of two terms: unary potential and pairwise potential.

$$E(y,x) = \sum_{i \in V} E_1(y_i, x_i) + \sum_{i,j \in V} E_2(y_i, y_j)) \tag{3.41}$$

Proper formulation of both of those terms is needed for Conditional Random Fields successfully perform their assigned tasks.

3.6.1 Unary potential

In semantic image segmentation, the unary component of the energy function predicts a label of a given pixel or region, based on some observed features of this part of an image. Though it can be modelled with a number of different methods it is always responsible for assigning the same label to input nodes that are characterised by similar features. A feature can be described as a numeric representation of row data that is fetched from an image. The process of extracting and selecting those representations is a challenging task as without properly chosen features that provide all the required information to the model, it would not be possible to perform the assigned task. What is more, it usually accounts for the vast majority of the time that is needed to perform a machine learning task [28]. Proper formulation of a feature vector is a crucial step in machine learning as it reflects not only the level of

the model complexity but also its performance. However, the relevance of available features is strictly bound to data and to the chosen model and due to to the large diversity in both data and models it is extremely difficult to generalise the process of feature engineering. Depending on the task some models can perform well given a large number of features while others require less, but more informative features. Though there exist some automatic or semiautomatic tools for feature engineering [29, 30, 31], that aim to make feature extraction less problem specific, still the vast majority of machine learning tasks use the traditional approach of manual feature selection [32]. Features used for any task within the area of image processing can be either low- or high-level [33]. Low-level features are such features that can be automatically extracted from an image such as pixel intensity, gradients, colours, edges or textures. They do not give any information about spatial relations between regions of an image. On the other hand, high-level features correlate data obtained from low-level features with a content of an image to provide contextual information about shapes and objects, thus having a semantic meaning. Feature engineering is a time-consuming and demanding task of its own and it is not a part of this dissertation. Therefore, to stay within the scope of the thesis, feature extraction is limited only to features that are based solely on colour-related data.

In the most basic form unary potential of a given factor between an input node x_i and an output node y_i is specified by a linear relation between feature function $\varphi(x)$ and learned parameters $w(y)$ [21] as in equation 3.42.

$$E_1(y_i, x_i, w) = \langle w(y_i), \varphi(x_i) \rangle \tag{3.42}$$

In the most basic form, feature function outputs a feature vector ϕ that is composed of colour values of each individual pixel. A colour value can be expressed in a numerical form with the use of a chosen colour space. A colour space is a

mathematical model representing a colour value as a tuple of typically three or four numbers. The most popular colour space being based on the physiology of a human eye is RGB model, in which an arbitrary colour value is denoted as a combination of values of three primary colours, which are red, green, and blue. A different colour space, named CIELAB, was created in other to mimic a human perception of colours. The model is constructed on three axes, first being axis L* representing Lightness with values between 0 and 100. Axes a* and b* encode green-red and blue-yellow components respectively. There are many other colour spaces such as CMYK, HSV, HSL or YUV, each of them being more suitable for certain tasks [34]. Nevertheless, the choice of colour space in which colour features will be encoded in the prediction model can influence its performance. For simplicity, in this chapter RGB colour space will be assumed. Hence, a feature vector ϕ for a given pixel is composed of three features, each representing a different component of the chosen colour space as in equation 3.43.

$$\phi(x_i) = \begin{bmatrix} \varphi_R \\ \varphi_G \\ \varphi_B \end{bmatrix} \tag{3.43}$$

When it comes to parameters of the unary potential they are expressed in terms of weights that are learned in the training process. With every label y there is a separate weight vector w associated, as in as in equation 3.44, and to compute unary potential a concatenation of weight vectors for each label is needed.

$$w(y) = \begin{bmatrix} w_R \\ w_G \\ w_B \end{bmatrix} \tag{3.44}$$

This distinction is required as each object class can be best characterised by differ-

ent features. For example, given an object with label *grass* a green component of the feature vector should be prioritised, while for label *sky* it should be a blue component. As the task of image segmentation, meaning finding an optimal prediction y^* for each pixel in an image, is equivalent to the problem of energy minimisation, weights which are associated with prioritised features should have relatively small values. Hence, with the same example of two labels *grass* and *sky* proper configuration of weights would be as in equation 3.45.

$$w_{grass} = \begin{bmatrix} 1.0 \\ 0.0 \\ 1.0 \end{bmatrix} \qquad w_{sky} = \begin{bmatrix} 1.0 \\ 1.0 \\ 0.0 \end{bmatrix} \tag{3.45}$$

Then, for example for fully blue pixels, with feature vector $\phi = [0.0, 0.0, 1.0]$, the unary component of energy will be equal to 0.0 for label *sky* and 1.0 for label *grass*. Thus, label *sky* will be chosen for such pixels, as the energy for this label is smaller than for other labels. Hence. to obtain the unary component of the energy function for a given pixel x_i with a given label y_i from a set of labels L, a corresponding weight vector needs to be multiplied by the feature vector as in equation 3.46.

$$E_1(y_i, x_i, w) = \begin{cases} \left\langle w_{1,0}, \phi(x_i) \right\rangle, & \text{if } y_i = 0 \\ \left\langle w_{1,1}, \phi(x_i) \right\rangle, & \text{if } y_i = 1 \\ \ldots & \ldots \\ \left\langle w_{1,L}, \phi(x_i) \right\rangle, & \text{if } y_i = L \end{cases} \tag{3.46}$$

Presented formulation of the unary potential is the most trivial one, however, it can also be applied also for more complex models and the only difference is in a way the feature vector is constructed. The goal of energy function is to provide a measure of how well pixels in an image are labelled and any feature vector that can be

expressed in a numerical form is possible to be applied for the unary term of the energy. A feature function does not need to provide a feature vector that is composed of low-level features extracted from an image. Instead, a higher level of abstraction can be used to make features of a given pixel dependent on other classifiers. One example would be a Decision Tree Field approach in which factors that form unary potential are dependent on Decision Trees [35]. Another and a very popular and widely described approach to semantic image segmentation is to model a feature vector for unary potential by utilisation of outputs of a Convolutional Neural Network [18, 36, 37]. However, incorporating an additional classifier into the unary potential of Conditional Random Fields is not the only possibility to improve its performance. Another way is to treat feature function as a probability density function representing a relation between labels and features. Hence, instead of operating on a feature vector that directly represents some properties of a given pixel, or on outputs that were assigned by an auxiliary classifier, such feature function is introduced that will be able to provide a conditional probability of assigning a label to the current pixel given its features. This probability is expressed in terms of log-likelihood as in equation 3.47.

$$\varphi(x_i, y_i) = -\log p(y_i|x, i) \tag{3.47}$$

A given pixel i is represented as a set of features, hence, computation of unary potential is equivalent to finding probability distributions of every label y_i conditioned on a set of features f_i, which can be parametrised with additional parameter θ.

$$p(y_i|f_i, \theta) \tag{3.48}$$

Conditional probabilities of $p(y_i|f_i)$ are not known, however, according to Bayes theorem, they can be found if there is a prior knowledge on a reverse probabilities

$p(f_i|y_i)$ as in equation 3.49.

$$p(y_i|f_i) = \frac{p(f_i|y_i) \cdot p(y_i)}{p(f_i)} \tag{3.49}$$

As probabilities of all possible labellings for a given pixel needs to sum to 1, a term $p(f_i)$ acts as a normalising constant in this equation. Therefore, it can be obtained by equation 3.50.

$$p(f_i) = \sum_{y_i \in \mathcal{Y}} (p(f_i|y_i) \cdot p(y_i)) \tag{3.50}$$

Hence, to calculate a unary potential for a pixel in a given image it is necessary to know the probability of occurrence of the chosen label and a probability of pixel features conditioned on this label. Both of them can be found from a probability distribution that needs to be modelled in an additional training process. With a set of correctly labelled training examples a label probability $p(y_i)$ is straightforward to calculate as it enough to find the ratio between pixels with a given label assigned and all pixels in a training set. Computation of $p(f_i|y_i)$ can be fairly complex, as a dimension of this distribution is dependent on the number of chosen features. However, assuming that chosen features are independent of each other, a problem of finding N-dimensional probability distribution can be transformed into N problems of finding 1-dimensional distributions as in equation 3.51.

$$p(f_i|y_i) = \prod_{k=1}^{K} p(f_{i,k}|y_i) \tag{3.51}$$

Any number and kind of features can be chosen as long as they can be expressed in a numeric way. The method of assessing a conditional probability between a feature and a label is dependent on a feature type. In general, features can be either discrete or continuous. The first type also known as categorical data, represent such features that can have only one value from a set of predefined values. Then,

to get a probability $p(f_{i,k}|y_i)$ for a pixel from a given test image, it is enough to have probabilities of every possible value of the given feature computed beforehand. Once computed during the training process, such probabilities are applicable for any new test image, as for discrete features the whole input domain is covered. For instance, if an image will be first subjected to the process of colour quantisation in order to limit the number of available colours in an image for example by using a 32 colour palette, then a feature representing a colour of a pixel would be a discrete feature taking one out of 32 possible values. Hence, in order to get the probability distribution of such feature, it is enough to count how many times a given colour value appeared in the training set.

On the other hand, for continuous features the input domain is infinite, and therefore it is infeasible to compute a probability of occurrence of each value beforehand. A probability of continuous variables is represented by a probability density function. This function describes a relative likelihood that the variable of interest will have a given value. Then, the probability that the variable will fall within a certain range of values is equal to the integral of the variable density over this range [38]. However, usually no information is available on the type of features distribution, nor on parameters that define it. Fortunately, given a training set of data samples, it is possible to estimate the probability density function of a continuous variable even without knowing its type. It can be achieved by kernel density estimation, also known as Parzen–Rosenblatt window method. The idea of this method is based on superposing kernel functions that are placed over observations and later scaling them in order to create one smooth function that is an approximation of all data points. Hence, to obtain probability $p(f_{i,k}|y_i)$ of a chosen feature k of pixel i there is a need to estimate probability distribution $p(f_{i,k})$ based on all N pixels labelled

with y_i according to formula 3.52.

$$p(f_{i,k}) = \frac{1}{Nh_k} \sum_{n=1}^{N} \mathcal{K}(\frac{f_{i,k} - f_{i,k}^n}{h_k}) \tag{3.52}$$

Kernel, denoted as $\mathcal{K}$, is a probability density function with known distribution that is symmetric around 0. The choice of kernel determines the shape of those individual functions around data points which will be averaged. Gaussian function, which models normal distribution, is an example of a function that can be used as a kernel. It is presented in formula 3.53.

$$\mathcal{K}(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2} \tag{3.53}$$

Other examples of kernel functions include uniform function, also known as rectangular window as in equation 3.54,

$$\mathcal{K}(u) = \frac{1}{2} \tag{3.54}$$

triangular function, presented in equation 3.55,

$$\mathcal{K}(u) = 1 - \|u\| \tag{3.55}$$

or parabolic function, named Epanechnikov kernel, which is shown in equation 3.56.

$$\mathcal{K}(u) = \frac{3}{4}(1 - u^2) \tag{3.56}$$

Bandwidth, denoted as h, is a hyperparameter of the kernel density estimation responsible for the level of data smoothing as it controls the width of individual superposed functions. Consequently, it reflects the range in which data points have a larger effect on an individual observation. The optimal choice of bandwidth is required for the good performance of the method. For large values of h the function will be underfitted, meaning it will be too simple to model known observations, nor

it will be able to predict probabilities of new data points. On the other hand, for values of h that are too small, the model will exhibit overfitting. It will almost perfectly learn all training data, including noise, however, it will not be able to generalise for new data points. As any hyperparameter, bandwidth value should be subjected to the optimisation process before the estimation takes place. A commonly used method to find optimal values of hyperparameters is called cross-validation [39]. In this method for every value of a hyperparameter that is under consideration, a set of observed data is divided into two partitions, a training set and a validation set. Then, the model is trained only with the first set and its performance is evaluated on the validation set. Typically, training and validation steps are performed multiple times using different partitions, so that most of the observed data will be validated against. Then, the results of each individual evaluation are aggregated to provide a measure of the model prediction performance. Optimal hyperparameter is the one for which this performance is the highest.

After finding the optimal value of the bandwidth h and choosing the kernel function $\mathcal{K}$ kernel density estimation can be used to predict probability $p(f_{i,k})$ of a feature k from pixel i from an unknown test image. This requires iterating through all the training data in order to compute the difference between the value of this feature and the observed value that is needed for a kernel function. For large training sets, this becomes computationally complex. Moreover, this needs to be repeated for every feature of all pixels in a test image, which makes it even more time-consuming. Hence, for large training sets, other methods of probability estimation should be used.

Parzen–Rosenblatt window method provides an accurate way to estimate the probability of an unknown sample based on the aggregated probability density function.

However, this function is freshly generated each time estimation is needed. In order to limit the number of required calculations, there is a need to propose a method that will model the underlying probability density only once, during the training phase, and not every time probability of a new observation needs to be assessed. Unfortunately, without knowing parameters describing the generated probability distribution function there is no way to regenerate it without iterating through all data points. However, the problem can be simplified if the training data would be firstly binned. Binning is the process aimed to discretise continuous data by grouping them into intervals of equal size that are called bins. The simplest way to model binned data is to use histograms, which give information on the number of data samples that fell into a particular bin. Then, to obtain the probability of a feature k of an unknown pixel i, it is enough to check to which bin its value falls and to calculate the ratio between the number of observations in a bin denoted as $b_{f_{i,k}}$ and all data points N as in equation 3.57.

$$p(f_{i,k}) = \frac{b_{f_{i,k}}}{N} \tag{3.57}$$

The number of observations in each bin can be computed only once, in a training phase, and it gives enough information to estimate the probability of a feature of any unknown pixel. The accuracy of this estimation is reflected by the number of bins used to create a histogram. If too few bins were used in a discretisation process, important data about observations, especially at the boundaries of bins, would be lost. On the other hand, the more bins are used, the more computationally complex the problem becomes. The number of bins is a hyperparameter of a histogram model, and as in the case of bandwidth in the Parzen–Rosenblatt window method, it should be found in an optimisation process.

After choosing an appropriate method of probability density estimation, it is pos-

sible to obtain a probability distribution of each feature $p(f_{i,k}|y_i)$ and consequently the whole distribution for a given image $p(f_i|y_i)$. This distribution is needed for Bayes' theorem, which was presented in formula 3.49. Then, for every pixel from an unknown test image, it will be possible to get a probability of each label $p(y_i|f_i)$, and therefore calculate the unary potential $E_1(y_i, x_i, w)$.

3.6.2 Pairwise potential

When it comes to the second component of the energy function - a pairwise potential, it is calculated for factors that connect outputs of neighbouring nodes. It is aimed to handle situations in which there is some noise as it is responsible for smoothness of the predicted labels. It penalises a situation in which nodes that are in close proximity to each other have a different label assigned. Thus, in the semantic image segmentation, a sample pixel that is surrounded by pixels with some label will be more likely to have the same label, even if the unary term is more prone to assign a different label to this pixel because of its noised features. In the simplest form, a weight vector φ is a two-dimensional vector with only two possible values $[1,0]$ for pairs with the same label, and $[0,1]$ for pairs with a different label, as presented in equation 3.58

$$\varphi_{y_i=y_j} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \varphi_{y_i \neq y_j} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{3.58}$$

Then, a weight vector is a two dimensional vector composed of weight $w_{2,0}$ which describes a situation in which neighbouring pixels have the same label, and weight

$w_{2,1}$ for a pair differing in assigned labels.

$$w(y_i, y_j) = \begin{bmatrix} w_{2,0} \\ w_{2,1} \end{bmatrix} \tag{3.59}$$

Similarly to the unary component of the energy function, the pairwise component can be described as a linear relation between weight and feature vectors. This is, in fact, equivalent to expressing it in terms of two weights which directly reflect pairwise energy value. If two pixels that form neighbouring nodes in a factor graph have the same label assigned, then the energy of the factor between them will be equal to weight $w_{2,0}$, if they have a different label assigned, then it will be $w_{2,1}$ as in equation 3.60. Again, as the state of smaller energy is the one that is going to be chosen for a given factor, if the situation of two neighbouring pixels having the same label is to be promoted, then $w_{2,0}$ should have a smaller value than $w_{2,1}$.

$$E_2(y_i, y_j) = \begin{Bmatrix} w_1 & , y_i = y_j \\ w_2 & , y_i \neq y_j \end{Bmatrix} \tag{3.60}$$

The pairwise term of energy function provides higher-order information about relations neighbouring pixels, however, the notion of neighbours is different depending on the model used. In general, a neighbour is any pixel that is adjacent to the given pixel, however, neighbourhoods of larger size can also be used. In such a situation factor graphs are highly beneficial as they allow to model relations between even larger groups of pixels in a clear and straightforward way. What is more, pairwise potential can be utilised to take into account even more complex relations between pixels by introducing a measure on how different their classes are from each other. Then, the feature vector, instead of having only two possible values 0 or 1, can take any value within the range of 0 and 1 which reflects a measure of similarity between pixel classes. Then, the formula for a feature vector used to calculate a

pairwise potential can be expressed as in the equation below.

$$\varphi(y_i, y_j) = \begin{bmatrix} 1 - \left|(y_i - y_j)\right| \\ \left|(y_i - y_j)\right| \end{bmatrix} \tag{3.61}$$

Assuming that the difference between labels $\left|(y_i - y_j)\right|$ is either 0 if labels are the same or 1 if labels are different, equation 3.61 is equivalent to equation 3.58.

Penalising a situation in which adjacent pixels have different labels is well suited for regions in which there are some noised pixels inside. However, not every time when two neighbouring pixels have a different label assigned necessarily means that there is some noise in one of those pixels. On object boundaries assignment of different labels should be promoted and not penalised. One of the most popular methods used to represent pairwise potential in Markov Random Fields is named Potts model [40]. This model represents spatial relations between neighbouring pixels taking into account their similarity. Neighbouring pixels affect each other just like in the previously described definition of a pairwise potential, however, in this model, only when pixels are alike the assignment of the same label is promoted. Potts model is applicable to problems in which labelling is locally constant and regions are separated by clear boundaries. Analogously to unary potential, a similarity between pixels is defined by their features. A pairwise feature vector $\varphi(y_i, y_j)$ of a factor between output nodes i and j that is constructed with the use of Potts model is defined as in formula 3.62. Depending on the number of pairwise features chosen the size of this vector will be different. For one feature it will have two elements only - a term defining pixel similarity, and a unit element that allows incorporation

of bias to the system.

$$\varphi(y_i, y_j) = \begin{bmatrix} \exp\left(-\beta_0 \left\|\varphi_0(x_i) - \varphi_0(x_j)\right\|^2\right) \\ \exp\left(-\beta_1 \left\|\varphi_1(x_i) - \varphi_1(x_j)\right\|^2\right) \\ \dots \\ \exp\left(-\beta_n \left\|\varphi_n(x_i) - \varphi_n(x_j)\right\|^2\right) \\ 1 \end{bmatrix} \tag{3.62}$$

Potts model requires a measure of similarity between pixels in order to check if there is an edge between them. This measure is represented as the difference between feature vectors of those pixels $\left\|\varphi(x_i) - \varphi(x_j)\right\|$ and a way of its calculation is dependent on a type of the chosen features. For example, a contrast sensitive Potts model that measures the difference between colour features of neighbouring pixels can be used [41], as differences in colour intensity are the simplest indicator of edges in an image. Then $\varphi(x_i)$ and $\varphi(x_j)$ would denote a three-dimensional vectors representing colours in a chosen colour scheme, of a given pixel pair i and j. Depending on the problem a similarity of pixels can be calculated differently. If an image was pre-classified so that each region is represented by one discrete state, it is enough to propose a binary method of pixel similarity measure. Then the difference $\varphi(x_i)$ and $\varphi(x_j)$ would take value 0 if pixels are of the same class, and value 1 otherwise. For images that were not pre-classified, this measure can take any numeric value. For example, for colour features, it would the distance between colours in the chosen colour scheme. The CIELAB scheme would be a beneficial choice, as its way of modelling colour differences is the most similar to human colour vision.

The described way of representing pairwise feature vectors requires also a hyperparameter β. This is an image dependent parameter that reflects the spatial correlation

between adjacent image pixels [42]. For small values of β classification will remain noisy, while for too large β the result will be over-smoothed. Proper selection of this hyperparameter is crucial for denoising abilities of the pairwise term, however, it is difficult to fix this value a beforehand, as spatial organisation of images tend to differ. Therefore, a value of the hyperparameter β should be adjusted separately for each image. One of the ways to make β image specific is to introduce a dependency between this hyperparameter and an average value of a given feature difference $\left\langle \left\| \varphi(x_i) - \varphi(x_j) \right\| \right\rangle$ in the image, as in formula 3.63.

$$\beta(x) = 2 \cdot \left\langle \left\| \varphi(x_i) - \varphi(x_j) \right\| \right\rangle^{-1} \tag{3.63}$$

Knowing how to compute feature difference for each of the chosen pairwise features, and how to obtain the value of the parameter β it is possible to construct a feature vector that is needed to compute the pairwise potential.

Chapter 4

Semantic Image Segmentation system

blah blah introduction

4.1 Dataset

miał być VOC, a jest paint

4.2 Preprocessing

The task of semantic image segmentation is aimed to assign one label to each pixel of an image. The system that is described in this dissertation required extensive

computations both in training, and in inference phases. Consequently, for large images and extensive training sets, semantic image segmentation with use of Conditional Random Fields becomes a time and resource consuming process. A lot of improvement can be achieved by incorporating all the simplifications that were described in chapter 3 Structured Prediction, however, still every computation needs to be performed for each pixel in an image. That is why, in the described system a preprocessing stage was introduced that is aimed to limit the number of required computations. In this stage, each image that will be later used in any of the image sets, namely training, validation and testing sets, is subjected to the process of superpixel segmentation. The aim of this process is to divide an image into a number of regions named superpixels that are composed of multiple pixels that share similar colour properties. This segmentation is performed in such a way that each pixel in an image is assigned to exactly one superpixel, therefore, superpixels are non-overlapping. Moreover, if the division into superpixels is done correctly objects in the image should be divided into multiple superpixels, but no superpixel should be split by an object boundary [43].

Division into superpixels was the only part of the system that was accomplished with the use of an external library. This library takes three arguments on input, first one being an image on which segmentation should be performed, second is an expected number of superpixels, and the third one is a compactness coefficient. The last parameter defines the importance of uniformity in superpixels shape. For large value of compactness coefficient, resulting superpixels will have smooth boundaries and will be roughly in the same size, while for small values adherence to object boundaries is more promoted than the uniformity in superpixel size. The goal of this segmentation is to limit the number of inputs in a constructed factor graph to as large extent as possible at the same time sustaining information about object

boundaries. Hence, proper selection of the expected number of superpixels and a compactness constant is crucial for optimality of the image segmentation system described in this dissertation.

Choice of input parameters for the process of superpixel division should be image specific. When it comes to an expected number of superpixels the choice is rather simple, as this parameter directly reflects the number of inputs in the image factor graph. Hence, it should be mainly dependent on the image size and on the extend to which number of inputs should be reduced. On the other hand, the choice of the compactness coefficient is not that simple, as the relation between this parameter value and a resulting superpixel shape is not so straightforward. The importance of the proper choice of this parameter can be visualised basing on an example of one image. Figure 4.1 shows a sample image that will be subjected to the process of division into superpixels. The chosen image has the size of $512px \times 512px$. For this

Figure 4.1: *A sample input image for the task of initial segmentation into superpixels*

Source: *Public-Domain Test Images for Homeworks and Projects [44]*

sample image, the parameter defining the expected number of superpixels has been

chosen to 300. When it comes to compactness coefficient three values were chosen, 500, 50 and 2. First set of images shown in figure 4.2 presents one sample image divided into superpixels per one compactness coefficient value. A colour of each pixel in those images is the same as a mean colour of a superpixel which contains the given pixel. As visible, for large value of compactness coefficient being 500 as

(a) *Compactness 500* ***(b)*** *Compactness 50* ***(c)*** *Compactness 2*

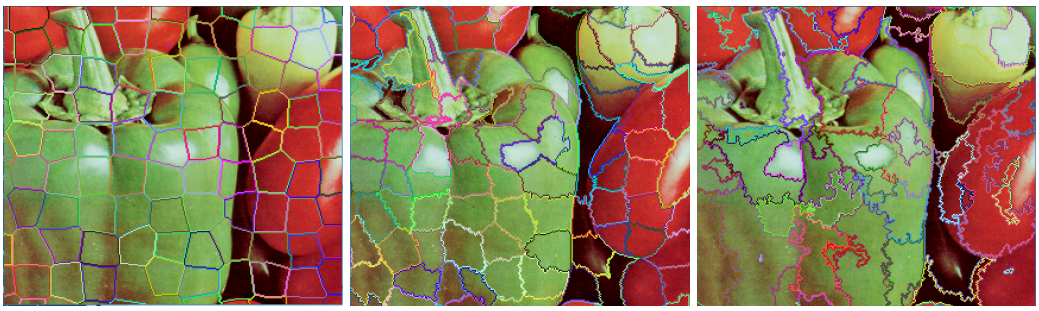
Figure 4.2: *Division of a sample image into superpixels based on different values of compactness coefficient, with mean colour of each superpixel marked*

shown in figure 4.2a, colours of the test image are blurred and it is hard to notice what this image presents without seeing it before the segmentation process. Segmented images with compactness coefficient for values 50 and 2 are quite similar. Next set of images shown in figure 4.3 present the results of the same processes of segmentation, however, this time superpixel borders are marked on the original image. As depicted in figure 4.3a for large values of compactness coefficient the resulting superpixels have smooth boundaries and are roughly in the same size and square-like shape. A similar division is presented in figure 4.3b for compactness coefficient of 50. Tough boundaries of of created superpixels are less smooth, still the size of them is pretty similar. Uniform size of superpixels is more suitable if an image is to be modelled with a graph, as they produce a regular lattice [45], which

(a) *Compactness 500* ***(b)*** *Compactness 50* ***(c)*** *Compactness 2*

Figure 4.3: *Division of a sample image into superpixels based on different values of compactness coefficient, with borders of each superpixel marked*

is important for proper definition of superpixel neighbour that is required for computing of pairwise potentials. On the other hand, for small values of compactness coefficient, as in figure 4.3c, the created superpixels adhere well to object boundaries, however, they differ largely in shape and size. Also the created lattice is highly irregular, as some objects are composed with only few superpixels, while in others even a small detail or a change of brightness is marked as a separate superpixel, which would make it difficult to assign this superpixel into a proper class. Difference between shape of created superpixels is best visible on object boundaries. Figure 4.4 depicts a zoomed view of superpixels that were created on the boundary between green and red peppers. It is clear that for compactness coefficient of 500 the created superpixels do not comply with the already presented rule that no superpixel should be split by an object boundary. With such superpixel division semantic image segmentation would not be successful, as the information about object boundaries would be lost during the preprocessing stage. Hence, such a large value of this parameter does not give acceptable results. For compactness of 2, as shown in figure 4.4c, created superpixels have totally different shapes. Furthermore, some



(a) Compactness 500 *(b) Compactness 50* *(c) Compactness 2*

Figure 4.4: *Division of a sample image into superpixels based on different values of compactness coefficient, with boundaries of each superpixel marked*

superpixels are composed of only a really small number of pixels and they usually present artefacts of an image, which do not carry any important information for semantic segmentation and would only increase the computational complexity of this task. Therefore, too small values of compactness coefficient should also be avoided. Figure 4.4b depicts superpixel segmentation that is the closest to optimal from all the presented divisions. The resulting superpixels properly define object boundaries at the same time creating a regular grid. What is more, small regions with changes of brightness that are irrelevant for semantic segmentation are not separated from surrounding pixels. Therefore, such division into superpixels would be applicable for further processing.

Thanks to the preprocessing stage, the number of inputs in the factor graph constructed upon a given image is largely lowered as instead of hundreds of thousands of pixels only few hundreds superpixels are needed. As a result the complexity of calculations and a required processing time are reduced. What is more, by division into superpixels pieces of image that are irrelevant for the task of semantic image segmentation are averaged and incorporated into one superpixel together with adja-

cent regions that carry more important information about the object they represent. This smoothing property is important for example for some slightly noised pixels or regions with the sudden change of brightness in few adjacent pixels.

Bibliography

[1] R. C Gonzalez and R. E Woods. “Digital Image Processing (2nd Edition)”. In: New Jersey. Prentice Hall, Jan. 2002. Chap. Image Segmentation. ISBN: 0201180758.

[2] C. A. Glasbey and G. W. Horgan. *Image Analysis for the Biological Sciences*. 1995. ISBN: 0-471-93726-6.

[3] *PASCAL VOC*. `http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html/`. [Online; accessed 19.06.2018].

[4] B. Fan, Z. Wang, and F. Wu. “Local Image Descriptor: Modern Approaches”. In: Jan. 2015. Chap. Classical Local Descriptors.

[5] T. K. Ho. “Random Decision Forests”. In: *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*. ICDAR ’95. IEEE Computer Society, 1995, pp. 278–. ISBN: 0-8186-7128-9.

[6] C. Cortes and V. Vapnik. “Support-Vector Networks”. In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. ISSN: 1573-0565.

[7] D. Ciresan et al. “Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images”. In: *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, pp. 2843–2851.

[8] J. Long, E. Shelhamer, and T. Darrell. “Fully convolutional networks for semantic segmentation”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015, pp. 3431–3440.

[9] L. Chen et al. “Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs”. In: *CoRR* abs/1412.7062 (2014).

[10] Y. Altun, I. Tsochantaridis, and T. Hofmann. “Hidden Markov Support Vector Machines”. In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*. ICML’03. AAAI Press, 2003, pp. 3–10. ISBN: 1-57735-189-4.

[11] Y. Liu, E. P. Xing, and J. Carbonell. “Predicting Protein Folds with Structural Repeats Using a Chain Graph Model”. In: *Proceedings of the 22Nd International Conference on Machine Learning*. ICML ’05. ACM, 2005, pp. 513–520. ISBN: 1-59593-180-5.

[12] A. Torralba, K. Murphy, and W. T. Freeman. “Contextual models for object detection using boosted random fields”. In: *NIPS* (Oct. 2004).

[13] J. Pearl. “Causality: Models, reasoning, and inference, second edition”. In: *Causality* 29 (Jan. 2000).

[14] P. Pardalos and J. Xue. “The Maximum Clique Problem”. In: *Journal of Global Optimization* 4 (Apr. 1994), pp. 301–328.

[15] S. P. Chopra. “Factor Graphs for Relational Regression”. PhD thesis. 2008. ISBN: 978-1-109-90255-6.

[16] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML '01. Morgan Kaufmann Publishers Inc., 2001, pp. 282–289. ISBN: 1-55860-778-1.

[17] C. Sutton and A. McCallum. "An Introduction to Conditional Random Fields". In: *Found. Trends Mach. Learn.* 4.4 (Apr. 2012), pp. 267–373. ISSN: 1935-8237.

[18] P. Krähenbühl and V. Koltun. "Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials". In: *Advances in Neural Information Processing Systems 24*. Curran Associates, Inc., 2011, pp. 109–117.

[19] A. Blake, P. Kohli, and C. Rother. *Markov Random Fields for Vision and Image Processing*. The MIT Press, 2011. ISBN: 9780262015776.

[20] J. Pearl. "Reverend Bayes on inference engines: a distributed hierarchical approach". In: *in Proceedings of the National Conference on Artificial Intelligence*. 1982, pp. 133–136.

[21] S. Nowozin and C. H. Lampert. "Structured Learning and Prediction in Computer Vision". In: *Found. Trends. Comput. Graph. Vis.* 6 (Mar. 2011), pp. 185–365. ISSN: 1572-2740.

[22] J. S. Yedidia, W. T. Freeman, and Y. Weiss. "Exploring Artificial Intelligence in the New Millennium". In: Morgan Kaufmann Publishers Inc., 2003. Chap. Understanding Belief Propagation and Its Generalizations, pp. 239–269. ISBN: 1-55860-811-7.

[23] B. Kim, A. Yedla, and H. D. Pfister. "Message-Passing Inference on a Factor Graph for Collaborative Filtering". In: *CoRR* abs/1004.1003 (2010).

[24] C. Wang, N. Komodakis, and N. Paragios. "Markov Random Field Modeling, Inference & Learning in Computer Vision & Image Understanding: A Survey". In: *Computer Vision and Image Understanding (CVIU)* 117.11 (Nov. 2013), pp. 1610–1627.

[25] Q. V. Le et al. "On Optimization Methods for Deep Learning". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Omnipress, 2011, pp. 265–272. ISBN: 978-1-4503-0619-5.

[26] F. E. Curtis and K. Scheinberg. "Optimization Methods for Supervised Machine Learning: From Linear Models to Deep Learning". In: *The Operations Research Revolution*. 2017, pp. 89–113.

[27] P. M. Lee. "The Gibbs sampler and other numerical methods". In: *Bayesian Statistics: An Introduction*. John Wiley & Sons, 2012, p. 281. ISBN: 9781118359778.

[28] A. Zheng and A. Casari. "Feature Engineering for Machine Learning. Principles and Techniques for Data Scientists". In: O'Reilly Media, Apr. 2018. Chap. The Machine Learning Pipeline, pp. 1–5. ISBN: 978-1-491-95324-2.

[29] *AutoML: Automatic Machine Learning*. `http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html/`. [Online; accessed 16.01.2019].

[30] *TPOT: Data Science Assistant*. `https://epistasislab.github.io/tpot/`. [Online; accessed 16.01.2019].

[31] *auto-sklearn*. `https://automl.github.io/auto-sklearn/stable/`. [Online; accessed 16.01.2019].

[32] U. Khurana et al. "Cognito: Automated Feature Engineering for Supervised Learning". In: *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*. Dec. 2016, pp. 1304–1307.

[33] M. Nixon and A. S. Aguado. *Feature Extraction & Image Processing for Computer Vision, Third Edition*. 3rd. Academic Press, Inc., 2012. ISBN: 9780123965493.

[34] A. Saglam and N. A. Baykan. "Effects of color spaces and distance norms on graph-based image segmentation". In: *2017 3rd International Conference on Frontiers of Signal Processing (ICFSP)*. Sept. 2017, pp. 130–135.

[35] S. Nowozin et al. "Decision tree fields". In: *2011 International Conference on Computer Vision*. Nov. 2011, pp. 1668–1675.

[36] G. Lin et al. "Efficient piecewise training of deep structured models for semantic segmentation". In: *CoRR* abs/1504.01013 (2015).

[37] A. Kirillov et al. "A Generic CNN-CRF Model for Semantic Segmentation". In: (Nov. 2015).

[38] D. R. Anderson. "Continuous Probability Distribution". In: *Modern Business Statistics*. Oct. 2017. ISBN: 9781538803660.

[39] P. Refaeilzadeh, L. Tang, and H. Liu. "Cross-Validation". In: *Encyclopedia of Database Systems* 532–538 (Jan. 2009), pp. 532–538.

[40] M. Hao et al. "A contrast-sensitive Potts model custom-designed for change detection". In: *European Journal of Remote Sensing* 47.1 (2014), pp. 643–654.

[41] J. Shotton et al. "TextonBoost for Image Understanding: Multi-Class Object Recognition and Segmentation by Jointly Modeling Texture, Layout, and Context". In: *International Journal of Computer Vision* 81.1 (Jan. 2009), pp. 2–23. ISSN: 1573-1405.

[42] M. Pereyra et al. “Estimating the Granularity Coefficient of a Potts-Markov Random Field Within a Markov Chain Monte Carlo Algorithm”. In: *IEEE Transactions on Image Processing* 22.6 (June 2013), pp. 2385–2397. ISSN: 1057-7149.

[43] P. Neubert. “Superpixels and their Application for Visual Place Recognition in Changing Environments”. PhD thesis. 2015.

[44] F. a. p. petitcolas. *Public-Domain Test Images for Homeworks and Projects"*. `https://homepages.cae.wisc.edu/~ece533/images/`. [Online; accessed 29.01.2019].

[45] R. Achanta et al. “SLIC Superpixels Compared to State-of-the-Art Superpixel Methods”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.11 (Nov. 2012), pp. 2274–2282. ISSN: 0162-8828.