Universität des Saarlandes
Max-Planck-Institut für Informatik

Scene Segmentation in Adverse Vision Conditions

Masterarbeit im Fach Informatik
Master's Thesis in Visual Computing
von / by

Evgeny Levinkov

angefertigt unter der Leitung von / supervised by

Dr. Mario Fritz

betreut von / advised by

Dr. Mario Fritz

begutachtet von / reviewers

Prof. Dr. Bernt Schiele
Dr. Mario Fritz

Saarbrücken, February 2013

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, February 2013 Evgeny Levinkov

Abstract

Semantic Image Segmentation is the task of segmenting images into homogeneous regions with semantic meaning. In Computer Vision it means assigning a class-label from a known set of classes to every pixel in the image. The standard approach is to take some sample data, divide it into training and testing sets, and to learn the class distribution on the training set, followed by checking the quality of the achieved generalization on the testing set. The generalization ability of the standard approaches is based on the assumption that the underlying class distribution does not change in both training and testing sets and does not evolve with time.

In this work we want to investigate in the problem of performing image segmentation in particularly difficult visual conditions. We want to look specifically into the conditions when the class distribution does change with time, which creates problems for the standard approaches.

As all the datasets, that are currently used in the image segmentation community, try not to break the constant class distribution assumption, we propose a new testing set, which exhibits significant variations in visual appearance of different classes and contains the class distribution different from the one in the training set.

We evaluate a number of standard approaches for image segmentation based on Conditional Random Fields, employing general and more specialized features. We look into the possible ways of performing adaptation to new conditions by means of online learning methods and propose our ideas on how to improve the generalization over previously unseen conditions without experiencing model drift.

Acknowledgements

I am deeply grateful to my supervisor Dr. Mario Fritz, who was leading me on my way to the Knowledge, kindly sharing his experience with me. I am thankful to Prof. Dr. Bernt Schiele and the whole crew of Computer Vision chair for getting me interested into the world of Computer Vision and providing a friendly and cheerful atmosphere during the whole period of lectures and writing my thesis. I am very thankful to Dr. Christian Wojek, who helped me very much by giving valuable pieces of advice. I would like to thank Saarland University for providing an outstanding studying possibilities and International Max Planck Research School for Computer Science for allowing me to use them. I am thankful to the whole Max Planck Institute for providing a home-like atmosphere during the whole period of staying in Germany and letting me know a lot of interesting people from all over the world.

And special thanks I want to give to my parents, to my relatives and friends, who supported me during my studies regardless of the thousands of kilometers that separated us.

In the memory of my grandmother.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Scene segmentation, as the task of separating an input image into homogeneous regions each having a predefined meaning (class-label), is one of the core problems in Computer Vision. Image segmentation helps to simplify an input image and analyze it, in order to help a machine to understand what is depicted there and to allow it to use this information in its work. This is, in particular, a crucial property for systems which are used for navigation of robots or cars in the environment, which is in turn can be used to help people to perform safer or completely autonomous driving or have other important civil application.

Figure 1.1 shows, probably, one of the most important applications of image segmentation in the driving assistance systems. In this case image segmentation can help to turn complicated for understanding road scenes' images into simpler ones, which in turn can be processed by an on-board computer and can, for example, hint the driver about possible dangerous situations on the road and help to avoid colliding with other vehicles and, most important, pedestrians.

Image segmentation can also have other fields of application. For example, it can be used as the lower layer in different object detection algorithm generating proposals for where objects belonging to certain classes might be in the image. Most detectors are based on the sliding-window approach, which means that if there exist a way to perform initial localization of regions of interest, this can significantly reduce the running time of the detection algorithm and improve their accuracy.

Being an important problem in Computer Vision, image segmentation has been studied carefully for a long time and a number of algorithms have been proposed to

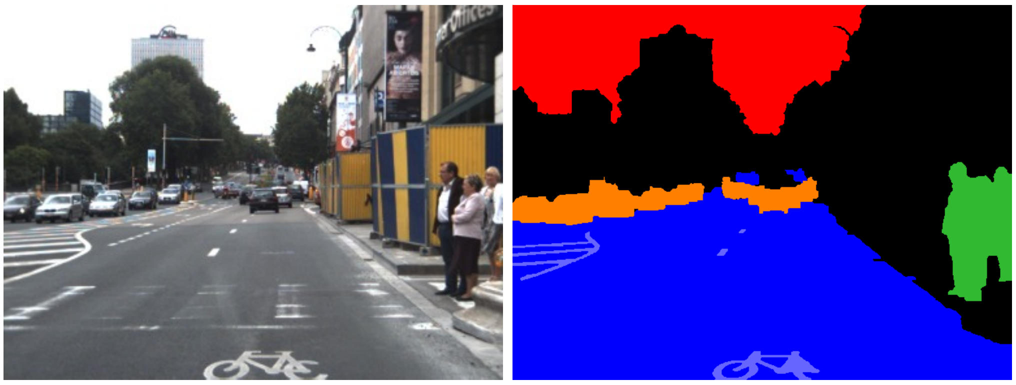
FIGURE 1.1: **Application of image Segmentation in driving assistance**. Left: the original image from an on-board camera installed on a car. Right: handmade segmentation of the original image. Image segmentation helps to simplify the input image in order to improve understandability and outlines, for example, the road surface (blue color), other vehicles (orange), and pedestrians (green), which can help make driving safer.

deal with it. The most conventional approaches are based on learning the underlying class distribution on some training set, and then predicting class-labels for previously unseen instances based on the learned distribution. The main assumption, which must hold in order for such methods to work, is that this class distribution stays the same for the unseen instances. Is this does not hold then these methods start to experience certain difficulties which result in degrade of the accuracy. In real-world application there can be situations, when this assumption might not hold. For example, a car was driving along a highway, but then turns into a country road. Though the surface the car is going on now can still be called a road, but its appearance completely differs from the asphalted highway. Now if we take, for example, a standard classifier which learned the class distribution on asphalted road samples will have difficulties on the country road as the appearance of the road and the underlying class distribution has changed considerably.

In order to deal with possible changes of the class distribution on-line learning techniques can be used. The main idea behind them is that they try to give algorithms the ability to adapt themselves to such changes, improving their robustness to previously unseen conditions, but also ensuring that the algorithm does not experience *model drift*, *i.e.* learning a distribution which differs too much from the original one.

In this thesis we want to explore on-line learning techniques and their particular application to the task of image segmentation. We are aiming for tracking an evolving target distribution over models. An example of such application is presented in Figure 1.2, which shows how image segmentation of a difficult image becomes better as algorithm performs adaptation with time. As all the datasets which are in use nowadays do not exhibit the level of variation in visual appearance we want to test, we collected

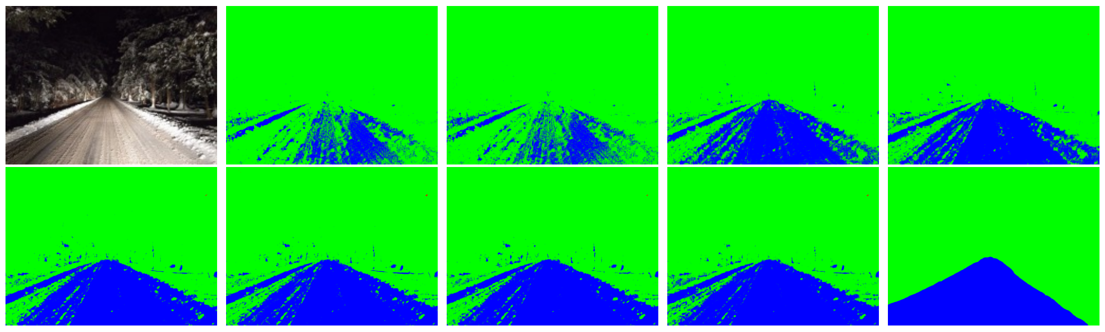

FIGURE 1.2: **An example of image labeling evolution in on-line learning**. First image presents the original image from the new dataset. Next four images show the segmentation after the first four iterations, and next four - after the last fout iterations of our proposed Bayesian Model Update algorithm. The labeling becomes better and gets closer to the ground truth (the last image) with time.

new road scenes' dataset and manually labeled it. We implemented a number of modern approaches which do not account for distribution changes and tested them on the new dataset, employing simple features as well as features specially designed for road detection. We experimented with existing on-line learning techniques and proposed our way of improving stability of on-line learning algorithms using Bayesian Model Update under Structured Scene Prior.

The contributions of this work are:

- a new more challenging road scenes dataset;
- a new Bayesian Model Update under Structured Scene Prior approach, which allows to tack an evolving distribution and therefore perform adaptation to new conditions.

1.2 Related Work

In order to experiment with adaptive techniques we first needed to choose an appropriate classifier and features. From the classifier we needed good accuracy, low training and testing time and initial robustness to noise in the samples' labels, as our approach to performing adaptation is based on on-line learning which uses predicted label in the process of retraining and they can be noisy. We also investigated in how Conditional Random Fields approaches can help to improve segmentation in adverse conditions, as well as other approaches which wotk with structured label space.

Unary Classifiers The idea of combining classifiers (boosting) in order to improve their combined (boosted) quality is due to [34]. A few years later the first boosting algorithm was proposed in [15], which gave birth to very popular boosting algorithms called AdaBoost and GentleBoost. For a long time boosting was a de facto standard classifier for any real-time segmentation algorithm ([38, 31]). Random Forests were introduced in [11] and further explored in [18] and showed very good accuracy ([37]) and speed of both evaluation and testing ([35]) due to easy parallelization, and thanks to these properties they start to become a default choice for a basic classifier in image segmentation ([12, 37, 25]). In particular, [12] used Random Forests to perform image segmentation of videos captures on roads.

Conditional Random Fields The first application of Conditional Random Fields in Semantic Image Segmentation can be regarded to [29], who proposed to use CRFs to improve segmentation results. The work [28] extended the conventional CRF formulation by adding higher order potentials on image segments to improve consistency between them. However, the accuracy of these latter approach is restricted by the accuracy of unsupervised image segmentation, which is used to locate the regions on which the model operates. The most recent work of [26] proposes to use a fully-connected CRF over the image pixels in order to use information contained in the image itself to improve segmentation. This approach helps to preserve very fine details of objects' borders, whereas conventional CRF approaches tend to smooth image boundaries. [26] is currently state-of-the-art algorithm for semantic image segmentation. It is worth mentioning that it is based on the method of [38], the main part of which are Texton features, which we employ in this work. We also employ advanced features for road detection from [45].

While the above mentioned approaches consider only feature space of an image in order to improve segmentation results, [25] propose to use not the single pixel's label, but rather to consider a certain labeling neighborhood around the pixel. This method allows to learn frequently occurring in the training data labeling patterns, which certainly helps to make the segmentation more consistent. We think that this method is particularly useful in the task of segmentation of road scenes, as there one can expect quite constant patterns of objects.

On-line Learning The most obvious way of performing adaptation to changing conditions and, as a result, to the underlying class probability distribution is to use the result of the currently segmented images to improve the classifier in a smart way, in order not to break what is working correctly currently. This can be done by adding recently labeled samples into the classifier and retraining it. This process is called *on-line learning*. The first attempts were made in [31]. Later [33] showed that on-line learning can be done efficiently on the basis of Random Forest. They showed that with time the on-line classifier converges to the *off-line* one (*i.e.* the one which was trained having access to all the samples). But in their approach they use samples together with labels, whereas we're aiming for the scenarios when there are no any labels known, but only predictions of the current-state classifier are given. Domain adaptation techniques [32, 27] can help to adapt to evolving target domain distribution, but they also require at least some sample instances with ground truth labels from the target domain. The most recent work [9] considers exactly this setting, by adding new samples with the predicted labels constraining this process by the prior labeling distribution. But in this work they perform testing on the CamVid [1] dataset, which is quite out-of-date, and their dataset [10], which has binary labeling into "road" and "background" and is not that challenging, to our taste. Our proposed approach of improving stability and robustness of an on-line segmentation algorithm is based on the Condensation algorithm from [21], which was shown to be extremely useful in robotics application for robot localization in [13].

1.3 Outline

The rest of this work is structured as follows:

- Chapter 2 starts the thesis with a brief introduction/review of Conditional Random Fields with an accent on their particular application to the task of Semantic Image Segmentation. Section 2.1 gives the definition of Conditional Random Fields and shows they are applied to image segmentation. We research the process of building a classifier from the very bottom layer, so the next Section 2.2 describes which features we employed in this work (in particular: raw color features 2.2.1, texture-layout features from [38] for general purpose image segmentation 2.2.2, and advanced features for road detection from [45] 2.2.3). Section 2.3 gives description of the two most popular nowadays unary classifiers Boosting 2.3.1 and Random Forests 2.3.2 and gives the pros and cons of each classifier. In our work we employ a number of algorithms which being put on top of the unary classifier allow to enhance the quality of the output segmentation. Section 2.3.3 describes the approach from [25] (Structured Class-Label Prediction) which allows the unary classifier to work not on single pixel's labels but on the whole label patches. Section 2.4.1 describes the method from [26] (Fully connected CRFs) which builds a fully-connected graph on the image pixels and using the output of the unary classifier allows to considerably improve segmentation results.

- Chapter 3 gives a description of on-line learning and outlines the benefits it can bring. Section 3.1 presents the most simple approach for performing on-line learning, while Section 3.2 contains our implementation of ideas from [9] on performing prior-constrained adding of new samples in the process of on-line learning. Section 3.3 introduces our proposed approach for improving stability called Bayesian Model Update under Structured Scene Prior.

- Chapter 4 gives an overview of different datasets available in the image segmentation community and argues why they do not suit our research of domain adaptation. There we introduce a new road scenes' dataset which we gathered and labeled by hand giving a number of examples from it. This dataset features extremely (in comparison to all other conventional road scenes' datasets) difficult (adverse) visual conditions.

- Chapter 5 describes all the experiments we conducted and gives the discussion of the achieved results together with our proposition on their improvement. Section 5.1.2 proves that Random Forests are robust to noisy labels and Section 5.1.1 shows the influence and dependencies of different parameters of Random Forests. Section 5.2 presents the segmentation results of our implementations of the approaches described in Chapter 2 on the MSRC dataset. Section 5.3.1 presents both numerical and visual results of evaluation of the state-of-the-art segmentation algorithm [26] on the dataset proposed in Chapter 4. The evaluation experiments on the proposed dataset are followed by Section 5.3.2, which includes approaches from Chapter 2 and features from Sections 2.2.1 and 2.2.2. Section 5.3.3 shows that Random Forest with advanced features from Section 2.2.3 can successfully compete with state-of-the-art segmentation algorithm [26]. Section 5.4 presents results for two existing on-line learning approaches, whereas Section 5.4.3 contains results of our proposed approach on getting improvement and stability of an on-line learning method.

- Chapter 6 concludes the thesis summarizing all the results of our work and proposing directions for possible improvements.

- Appendix A gives a detailed algorithm of building a Random Forest, together with the algorithm for traversing test samples through a Random Forest.

Chapter 2

Conditional Random Fields Approach

Conditional Random Fields (CRFs) are a class of statistical modelling method often applied in pattern recognition and machine learning, where they are used for structured prediction. Whereas an ordinary classifier predicts a label for a single sample without regard to neighboring samples, a CRF can take context into account.

CRFs are a type of discriminative undirected probabilistic graphical model. It is used to encode known relationships between observations and construct consistent interpretations. CRFs are useful in Computer Vision and particularly in Image Segmentation, because in this task observations (samples) are pixels, which represent the real world, and the latter usually shows strong relations and interactions between neighboring particles. So, taking into account these kinds of natural relations helps to considerably improve segmentation results.

2.1 Overview of Conditional Random Fields

2.1.1 Definition

Given $\mathbf{X}$ as a random variable over data sequences to be labeled and $\mathbf{Y}$ a random variable over corresponding label sequence (components of $\mathbf{Y}$ are assumed to range over a finite label alphabet $\mathcal{Y}$), Lafferty *et al.* [29] define a CRF on observations $\mathbf{X}$ and random variables $\mathbf{Y}$ as follows:

Definition 2.1. Let $G = (V, E)$ be a graph such that $\mathbf{Y} = (\mathbf{Y}_v)_{v \in V}$, so that $\mathbf{Y}$ is indexed by vertices of G. Then $(\mathbf{X}, \mathbf{Y})$ is a conditional random field in case, when

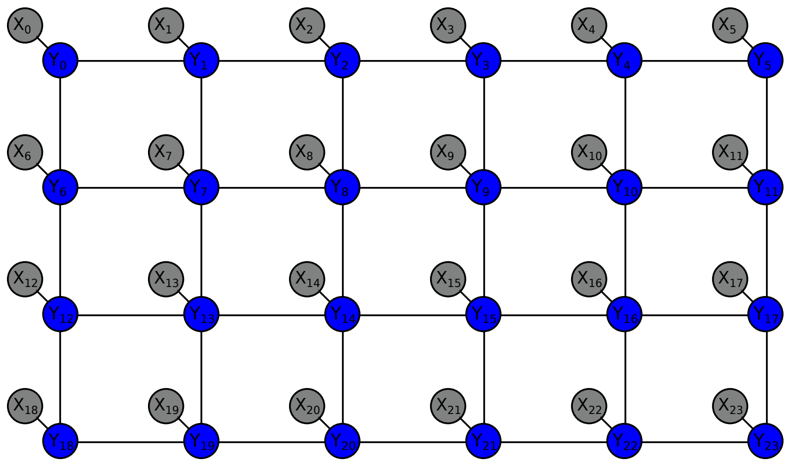

FIGURE 2.1: An example of a simple pairwise connected Conditional Random Field over a 6x4 image. $\mathrm{Y}_0, \ldots, \mathrm{Y}_{23}$ denotes unkown labels of pixels, $\mathrm{X}_0, \ldots, \mathrm{X}_{23}$ denotes observations, grey color means that variable X_i is conditioned on. In this example every pixel is connected to its four closest neighbors.

conditioned on $\mathbf{X}$, the random variables $\mathbf{Y}_v$ obey the Markov property with respect to the graph: $P(\mathbf{Y}_v|\mathbf{X}, \mathbf{Y}_\omega, \omega \neq v) = P(\mathbf{Y}_v|\mathbf{X}, \mathbf{Y}_\omega, \omega \sim v)$, where $\omega \sim v$ means that ω and v are neighbors in graph G.

What this means is that a CRF is an undirected graphical model (Figure 2.1) whose nodes can be divided into exactly two disjoint sets $\mathbf{X}$ and $\mathbf{Y}$, the observed and output variables, respectively; the conditional distribution $P(\mathbf{Y}|\mathbf{X})$ is then modeled.

Definition 2.2. A *potential* $\phi(x)$ is a non-negative function of variable x. A *joint potential* $\phi(x_1, \ldots, x_n)$ is a non-negative function of a set of variables $x_1, \ldots, x_n$.

Then we can write down the conditional probability distribution as a Gibbs distribution:

$$P(\mathbf{Y}|\mathbf{X};\theta) = \frac{\exp - \sum_{c \in \mathcal{C}_G} \phi_c(\mathbf{Y}_c|\mathbf{X};\theta)}{Z(\mathbf{X};\theta)}, \tag{2.1}$$

where θ is some parameter vector of the model, $\mathcal{C}_G$ is the set of all cliques of the graph G, $\phi_c(Y|\mathbf{X};\theta)$ is a potential which works on clique c, $\mathbf{Y}_c$ is a subset of $\mathbf{Y}$ of variables which comprise the clique c and the normalizing constant $Z(\mathbf{X};\theta) = \sum_Y \exp - \sum_{c \in \mathcal{C}_G} \phi_c(Y|\mathbf{X};\theta)$ is just a summation over all possible labelings $Y \in \mathcal{L}^N$, where $N = |V|$ is the number of variables.

In the Semantic Image Segmentation task we are interested in finding the labeling of pixels $\mathbf{Y}$ which has the highest probability given the observation values $\mathbf{X}$:

$$\mathbf{Y} = \underset{\hat{Y} \in \mathcal{L}^N}{\operatorname{argmax}} P(\hat{Y}|\mathbf{X};\theta) = \underset{\hat{Y} \in \mathcal{L}^N}{\operatorname{argmax}} \frac{\exp - \sum_{c \in \mathcal{C}_G} \phi_c(\hat{Y}_c|\mathbf{X};\theta)}{Z(\mathbf{X};\theta)} \tag{2.2}$$

One can observe that the normalization factor $Z(\mathbf{X}, \theta)$ does not depend on y, so it can be dropped out. Then we get a simplified formula:

$$\mathbf{Y} = \underset{\hat{Y} \in \mathcal{L}^N}{\operatorname{argmax}} \exp - \sum_{c \in \mathcal{C}_G} \phi_c(\hat{Y}_c | \mathbf{X}; \theta) \tag{2.3}$$

The definition of cliques is quite an arbitrary thing and always depends on particular needs. The most popular definition of cliques is when there is a potential for each pixel $\phi(y_i | x_i; \theta)$ and there is an edge between nearby pixels in the graph G and an associated with this edge potential $\phi(y_i, y_j | x_y, x_j; \theta)$. Then we can rewrite the exponent in (2.3) as:

$$\sum_{c \in \mathcal{C}_G} \phi_c(\mathbf{Y}_c | \mathbf{X}; \theta) = \sum_i \phi(y_i | x_i; \theta) + \sum_{j \in \mathcal{N}(i)} \phi(y_i, y_j | x_i, x_j; \theta), \tag{2.4}$$

where $\mathcal{N}(i)$ denotes neighbors of pixel i in the graph G. The usual choice is to take left, right, top, and bottom neighbors. The neighborhood can be easily extended to be larger, but at the price of increasing inference complexity. We will show later in this work different types of this neighborhood.

2.1.2 Inference

For general graphs, the problem of exact inference in CRFs is intractable. The inference problem for a CRF is basically the same as for an Markov Random Field (MRF) and the same arguments hold. In MRFs one may calculate the conditional distribution of a set of nodes $V' = v_1, \ldots, v_i$ given values to another set of nodes $W' = \omega_1, \ldots, \omega_j$ by summing over all possible assignments to $u \notin V', W'$; this is called *exact inference*. However, exact inference is a n-P-complete problem, and thus computationally intractable in the general case. Approximation techniques such as Markov chain Monte Carlo and loopy belief propagation are often more feasible in practice. However for CRFs there exist special cases for which exact inference is feasible:

- If the graph is a chain or a tree, message passing algorithms yield exact solutions. The algorithms used in these cases are analogous to the forward-backward and Viterbi algorithm [23] for the case of HMMs.
- If the CRF only contains pair-wise potentials and the energy is submodular, combinatorial min cut/max flow algorithms yield exact solutions.

If exact inference is impossible, several algorithms can be used to obtain approximate solutions. These include:

- Loopy belief propagation
- Alpha expansion
- Mean field inference
- Linear programming relaxations

2.2 Features

Features are a crucial part of every classifying system. They define the transformation of the real world into a form most suitable for current classification task. Feature engineering can be called an art, because it requires good feeling and knowledge of the problem and sometimes it takes a lot of time to come up with good features.

Certainly, even very good features cannot do the whole job of building a "perfect" classifier, but one shouldn't underestimate their impact on the overall performance.

2.2.1 Raw Color Features

In Semantic image segmentation task we get as an input color images. So, the most simple kind of features would be just taking RGB values of every single pixel. However, it was mentioned in [37, 38] that CIELAB color space tend to generalize better. In this work we actually need only forward transformation, because we need CIELAB values for the classifier and do not need to convert the image in CIELAB color space back to the usual RGB.

In order to convert RGB values to the corresponding CIELAB representation, we first need to convert RGB to the standard XYZ color space. This can be implemented as a simple matrix-vector multiplication:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = A \times \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

RGB of every pixel should be first per channel normalized so that every value lies in the range $[0, 1]$. There exist a large number of choices for matrix A from different standards. We used the following one:

$$A = \begin{pmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{pmatrix} \tag{2.5}$$

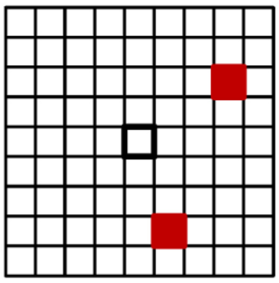

FIGURE 2.2: **Feature patch** (courtesy of [37]). This is an example of a feature patch. Bold-framed square in the center denotes the pixel being currently processed. Two red squares show two possible pixel locations relative to the central one which are actually considered when computing feature response for the central pixel.

After this, with the help of the following formulas XYZ values can be converted to LAB ones:

$$\begin{aligned} L &= 116 f(Y/Y_w) - 16, \\ a &= 500[f(X/X_w) - f(Y, Y_w)], \\ b &= 200[f(Y/Y_w) - f(Z/Z_w)], \end{aligned}$$

where

$$f(t) = \begin{cases} t^{\frac{1}{3}} & \text{if } t > \left(\frac{6}{29}\right)^3 \\ \frac{1}{3}\left(\frac{29}{6}\right)^2 t + \frac{4}{29} & \text{otherwise.} \end{cases}$$

Here (X_w, Y_w, Z_w) are the CIEXYZ tristimulus values of the reference white point. Accordingly with the chosen matrix (2.5) we take $(X_w, Y_w, Z_w) = (0.950456, 1.0, 1.088854)$.

The main improvement one can achieve even with raw color features is by considering information not only from a single pixel, but rather by applying some function to a number of pixels in the neighborhood (Figure 2.2). The following functions has been shown ([37, 25]) to be most useful:

$$\begin{aligned} \varphi(x|\theta_1, \theta_2) &= I_{(u,v,0)+\theta_1} - I_{(u,v,0)+\theta_2}, \\ \varphi(x|\theta_1, \theta_2) &= I_{(u,v,0)+\theta_1} + I_{(u,v,0)+\theta_2}, \\ \varphi(x|\theta_1, \theta_2) &= \left| I_{(u,v,0)+\theta_1} - I_{(u,v,0)+\theta_2} \right|, \end{aligned}$$

where x is a patch from the image with coordinates (u, v), $\theta_i = (\delta u_i, \delta v_i, c_i), i = 1, 2$ are the displacement parameters, with δu_i and δv_i being a displacement relative to the patch center and c_i denoting one of the color channels. These function can capture for example long-distance gradients, homogeneous regions, *etc.* The application of these functions can be easily extended to even higher dimensional feature representations than just RGB or CIELAB color channels. In that case $c_i \in [1, \ldots, D]$, where D is the dimensionality

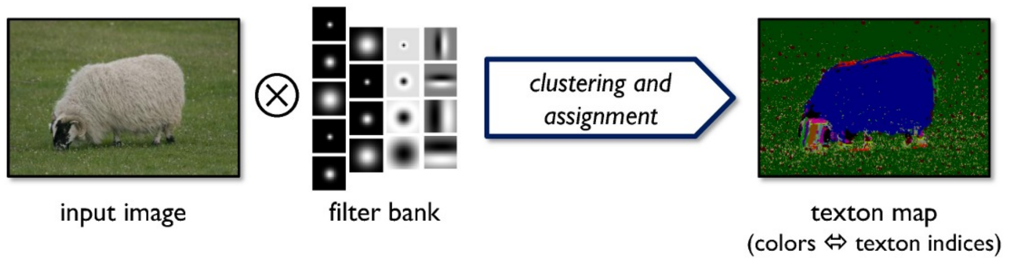

FIGURE 2.3: **The process of image textonization** (courtesy of [38]). An input image is convolved with a filter-bank. The filter-bank responses for all pixels in the training images are clustered. Finally, each pixel is assigned a texton index corresponding to the cluster center nearest to its filter-bank responses.

of the feature space. In case of raw color features $D = 3$.

2.2.2 Texture-Layout Filters

As features *textons* were introduces in [30] and later have been shown to be useful in such applications as material categorization [40] and general object categorization [44]. It is worth mentioning that the term 'texton' itself was introduced by [24] for describing human perception of textures.

The process of textonization is illustrated in Figure 2.3 and proceeds as follows. The training images are convolved with 17-dimensional filter-bank at scales k. We use the same filter-bank as proposed in [44], which consists of Gaussians at scales k, $2k$, $4k$, x and y derivatives of Gaussians at scales $2k$ and $4k$, and Laplacians of Gaussians at scales k, $2k$, $4k$, and $8k$. The Gaussians are applied to all three color channels, while the other filters are applied only to the luminance channel of CIELAB color space. This filter-bank was determined to have full rank in a singular value decomposition in [22] and therefore to contain no redundant elements. The 17-dimensional responses for all training pixels are then whitened (to give zero mean and unit covariance) and clustered in an unsupervised manner. We employ a freely available library by [41], which contains an efficient implementation of k-means Lloyd's and Elkan's algorithms. The latter uses triangular inequality [14] to avoid many distance calculations when assigning points to clusters and is typically much faster than Lloyd's algorithm. However, it uses storage proportional to the square of the number of clusters, which makes it impractical for a very large number of clusters. Finally, each pixel in the image is assigned to the closest cluster center producing a *texton map* T.

Texture-Layout filters were proposed in [38]. Each texture-layout filter is a pair (r, t) of an image region r and a texton t. The center of region r is defined in coordinates relative to the pixel i being classified. For efficiency, we use only rectangular regions, though any other shape is also possible. For simplicity, a set $\mathcal{R}$ of candidate regions is

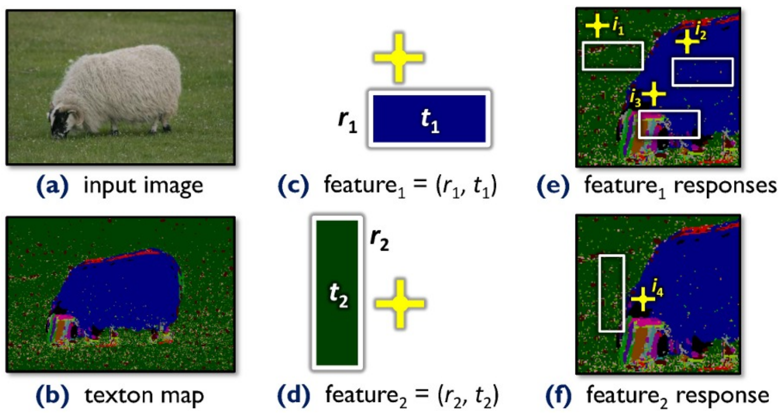

FIGURE 2.4: **Calculating feature responses and capturing textural context** (courtesy of [38]). **(a, b)** An image and its corresponding texton map (colors map uniquely to texton indices). **(c)** Texture-layout filters are defined relative to the point i being classified (yellow cross). In this first example feature, rectangular region r_1 is paired with texton t_1 (mapping to the blue color). **(d)** A second feature where region r_2 is paired with texton t_2 (green). To improve readability, (c) and (d) are shown double sized compared to (e) and (f). **(e)** The response $\upsilon_{[r_1,t_1]}(i)$ of the first feature is calculated at three different positions in the texton map (magnified). In this example $\upsilon_{[r_1,t_1]}(i_1) \approx 0$, $\upsilon_{[r_1,t_1]}(i_2) \approx 1$, and $\upsilon_{[r_1,t_1]}(i_1) \approx \frac{1}{2}$. **(f)** For the second feature (r_2, t_2), where t_2 corresponding to 'grass', the algorithm can learn such points as i_4 belonging to sheep regions tend to produce large values of $\upsilon_{[r_2,t_2]}(i)$, and hence can exploit the contextual information that sheep pixels tend to be surrounded by grass pixels.

sampled at random, such that their top-left and bottom-right corners lie within some fixed bounding box which covers considerable portion of an image (about a half or a quarter of an image). As illustrated in Figure 2.4, the feature response at location i is the proportion of pixels under the offset region $i + r$ that have texton index t:

$$\upsilon_{[r,t]}(i) = \frac{1}{\text{area}(r)} \sum_{j \in (i+r)} [T_j = t] \tag{2.6}$$

From here on [.] is an identity function defined as:

$$[condition] = \begin{cases} 1 & \text{if } condition \text{ is satisfied} \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

Outside the image boundary there is zero contribution to the feature response.

The filter responses can be efficiently computed over the whole image using integral images [43]. Figure 2.5 illustrates this process: the texton map is separated into K channels (one for each texton) and then, for each channel, a separate integral image is computed. The integral images can later be used to compute the texture-layout filter response in $\mathcal{O}(1)$ time: if $\hat{T}^{(t)}$ is the integral image of T for texton channel t, then the

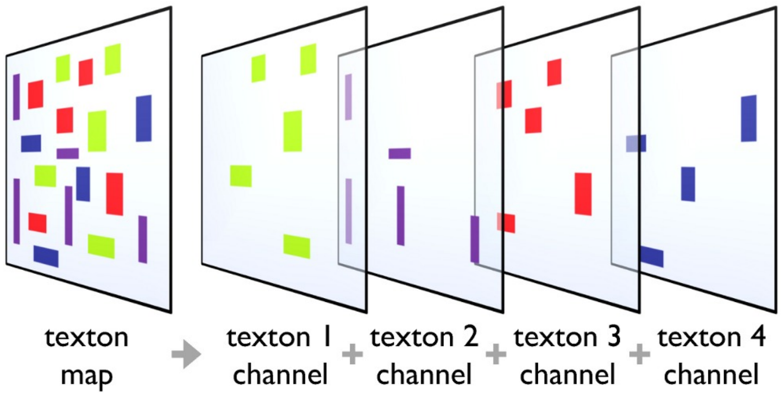

FIGURE 2.5: **Separating the texton map into multiple channels** (courtesy of [38]). The texton map of an image, containing K textons, is split into K channels. An integral image [43] is built for each channel and used to compute texture-layout filter responses in constant time.

feature response is computed as:

$$\upsilon_{[r,t]}(i) = \hat{T}^{(t)}_{r_{br}} + \hat{T}^{(t)}_{r_{tl}} - \hat{T}^{(t)}_{r_{bl}} - \hat{T}^{(t)}_{r_{tr}} \tag{2.8}$$

where r_{br}, r_{tl}, r_{bl}, r_{tr} denote the bottom right, top left, bottom left, and top right corners of rectangle r. [38] investigated into other region shapes rather than simple rectangles. In particular, they evaluated rectangles rotated by 45°, and pairs of rectangles with texton responses added $(\upsilon_{[r_1,t]}(i) + \upsilon_{[r_2,t]}(i))$ or subtracted $(\upsilon_{[r_1,t]}(i) - \upsilon_{[r_2,t]}(i))$. However, it only increased computation time, but didn't produce noticeably better results. So, we also decided to stay with just rectangles.

We used the same number of textons as in [38] equal to 400. This results in very high memory requirements, for example storing all 400 integral images for each image from MSRC [4] training data-set (this was about 45% of all images) would require about 30GB of memory. Therefore we subsample the train images and use w. l. o. g. only every 5-th pixel in each dimension at training time, but compute feature responses for test images at full resolution. This is possible as the responses are normalized by the area of the rectangle.

2.2.3 Advanced Features for Road Detection

We found out features from Christian Wojek and Bernt Schiele [45] to deal very good with the task of semantic image segmentation of road scenes.

Texture and location features The texture features are computed from the first 16 coefficients of the Walsh-Hadamard transform. This transformation is a discrete approximation of the cosine transform and can be computed efficiently [20, 8] even in real-time (for example using modern graphics hardware). The features are extracted at multiple scales from all channels of the input image in CIELAB color space. As a preprocessing step, a and b channels are normalized be means of gray world assumption to cope with varying color appearance. The L channel is mean-variance normalized to fit a Gaussian distribution with a fixed mean to cope with global lighting variations. They also found that normalizing the transformation's coefficients according to [39] is beneficial. They propose to L_1-normalize each filter response first and then locally normalize the responses at each image pixel. They take the mean and variance of the normalized responses as a feature and the grid point's coordinates within the image as a location cue.

As the result of this feature transform we get a 194-dimensional vector for each pixel. This makes storing all feature-transformed images in memory very problematic, so we perform a sub-sampling and use only every 4-th pixel in each dimension from training images. But we still perform feature transform for test images at their full resolution. These features were used only in the setting, when no any function was applied to feature patches, so there is no problem of matching between train and test images' feature responses.

We used the implementation kindly provided to us by the authors.

2.3 Unary Potentials

The main components of any CRF-based formulation is the *unary potentials*. They are responsible for making the initial prediction about sample's label, before any other enhancing technique is applied. It is exactly them who incorporate our knowledge about the behavior of individual samples and give us a strong clue about what class the sample might belong to. Any other higher-level technique expects to get a reasonably good prediction about each sample, because otherwise, despite whatever clever the algorithm is, it will not do much useful if already the initial guess is wrong. Therefore, it is extremely important to devote considerable amount of designing time to build a strong classifier which will be able to deal with the task already well enough. Here "strong" is quite a tentative term, because certainly one would try to find a classifier which has the lowest test error for a particular task. But on the other hand, if one aims for real-time application the question of running time becomes also very important. As a usual matter in life, there is a trade-off between the overall accuracy and the running time which turn out to be reciprocals (in the meaning) of each other: if ones designs an algorithm which has very good accuracy the running time is very probable to be very high, and vice versa.

Running time (both at training and testing) was a crucial matter for us. So, we tried to find a classification algorithm, which would have both reasonably (for our particular task; certainly, there is no “ideal” classifier in the Universe) good accuracy and as small as possible both training and testing time. Having this in mind we finally came up with two options: **Boosting** (in particular GentleBoost) and **Random Forest**.

2.3.1 Boosting

Boosting (the original idea is due to [16]) is a general method for improving the performance of any learning algorithm. In theory, boosting can be used to significantly reduce the error of any *weak* learning algorithm that consistently generates classifiers which need only be a little bit better than random guessing. Despite the potential benefits of boosting promised by the theoretical results, the true practical value of boosting can only be assessed by testing the method on “real” learning problems.

Boosting works by repeatedly running a given weak learning algorithm (although any learning algorithm can be used; weak learners are just faster both at training and testing, therefore opening way to real-time application of the whole classification algorithm) on various distributions over the training data, and then combining the classifiers produced by the weak learner into a single composite classifier. The first provably effective boosting algorithms were presented by R.E. Schapire [34] and Y. Freund [15].

The general case Boosting algorithm is given as follows:

```
**Data**: a set of training points $(x_i, y_i)_{i=1}^N$, a Weak Learner $f(x)$, number of iterations $M$
**Result**: final classifier $F(x) = \text{sign}(\sum_{m=1}^M \alpha_m f_m)$
initialize weights: $\gamma_i^1 = \frac{1}{N}, i = 1, \dots, N$;
**for** $m \leftarrow 1$ **to** $M$ **do**
    1. Train a classifier $f_m$ using a base method (Weak Learner) and the weighted
    training data $(x_i, y_i, \gamma_i^m)_{i=1}^N$;
    2. Recompute the weights $\gamma^{m+1}$, where the weights of wrongly classified training
    points are usually increased and the weights of correctly classified points are
    decreased;
    3. Compute coefficient $\alpha_m$ for the current Weak Learner $f_m$, which is either 1 or
    depends on the error of the classifier;
**end**
```

Algorithm 1: General Boosting algorithm

The intuition about Boosting is that it tries to aggregate classifiers which in general perform not good into a stronger classifier by paying more attention to the points which are wrongly classified at each iteration of the algorithm. That’s why the algorithm 1 increases weights for incorrectly classified points at step 2 to make the new Weak Learner to try to perform well exactly on this points, as the other points, which have low weights, were already correctly classified at previous iterations.

From the statistical point of view iterative updates of Boosting can be seen as minimization of a convex loss function over a convex set of functions or descent in function space:

$$F(x) \mapsto F(x) + \alpha_m f_m(x), \tag{2.9}$$

where

1. f_m is a descent direction based on the current F,
2. α_m is the step-size in the descent step

Particularly, the loss being optimized by the Boosting algorithm is essentially the empirical exponential loss of the final classifier F:

$$L(F) = \frac{1}{N}\sum_{i=1}^{N} exp(-y_i * F(x_i)) \tag{2.10}$$

There is a large variety of particular implementations of the Boosting algorithm 1. One of the most popular implementations are AdaBoost [16] (which stands for Adaptive Boosting) and GentleBoost. The latter one has a real-valued output (whereas in AdaBoost all WeakLearners must be binary-valued functions), which can be seen as the confidence of the classifier about certain classes. And besides, with further per pixel normalization its output can be converted into a probability distribution, which in turn can be used in some other enhancing algorithms, which are described later in this work.

The GentleBoost algorithm looks like this:

```
**Data**: a set of training points $(x_i, y_i)_{i=1}^N$, a *real-valued* Weak Learner $f(x)\colon \mathcal{X} \mapsto \mathbb{R}$,
    number of iterations $M$
**Result**: final classifier $F(x) = \text{sign}(\sum_{m=1}^M f_m(x))$
initialize weights: $\gamma_i^1 = \frac{1}{N}, i = 1, \ldots, N$;
**for** $m \leftarrow 1$ **to** $M$ **do**
    1. Fit the Weak Learner $f_m(x)$ into the weighted training data $(x_i, y_i, \gamma_i^m)_{i=1}^N$
    (Weak Learner uses weighted squared loss $L(f_m) = \sum_{i=1}^N \gamma_i (y_i - f_m(x_i))^2$;
    2. Update the weights $\gamma^{m+1}$ as
    $\gamma_i^{m+1} = \gamma_i^m \exp(-y_i f_m(x_i))$;
    3. Re-normalize $\gamma$ so that $\sum_{i=1}^N \gamma_i^{m+1} = 1$;
**end**
```

Algorithm 2: GentleBoost algorithm

Then the binary classification can be performed just as $\text{sign}(F(x))$. In case of K-class classification one builds K different boosting classifiers (built using one-vs-all strategy). Then, at test time, every sample is put into each of the K boosting classifiers resulting in a vector $c \in \mathbb{R}^K$ of confidences of each classifier in the sample belonging to a class $i, i = 1, \ldots, K$. After this the final prediction can be done as:

$$k = \underset{i \in [1,\ldots,K]}{\text{argmax}}\, c_i \tag{2.11}$$

At the same time vector c can be converted to a probability distribution, where high confidence means high probability of certain classes.

An interesting observation can be done for the GentleBoost algorithm:

Proposition 2.3. *If the Weak Learner f_m is real-valued, $f_m(x)\colon \mathcal{X} \mapsto \mathbb{R}$, then the update step $F_{m+1}(x) = F_m(x) + f_m(x)$ of the GentleBoost algorithm is an approximate Newton step in order to minimize the empirical exponential loss.*

Proof. We can expand the risk up to the second term using the Taylor series of $\exp(x)$,

$$\begin{aligned} R(F+f) = \mathbb{E}\left[L(F+f)\right] = \mathbb{E}\left[\exp(-y(F(x)+f(x))\right] = \\ = \mathbb{E}\left[\exp(-yF(x))(1 - yf(x) + \frac{1}{2}f(x)^2)\right]. \end{aligned} \tag{2.12}$$

At each iteration we are trying to find such f which minimizes the above risk $f = \text{argmin}_{\hat{f} \in \mathcal{F}} R(F + \hat{f})$, and observing that the first term in parenthesis in (2.12) does not depend on f we can modify it without changing the minimizer,

$$\begin{aligned} &\mathbb{E}\left[\exp(-yF(x))(1 - yf(x) + \frac{1}{2}f(x)^2)\right] = \\ &= \mathbb{E}\left[\exp(-yF(x))(\frac{1}{2} - yf(x) + \frac{1}{2}f(x)^2)\right] = \\ &= \mathbb{E}\left[\exp(-yF(x))(y - f(x)^2)\right]. \end{aligned} \tag{2.13}$$

We also used the fact that $y \in \{-1, 1\}$, so $y^2 = 1$ always holds. One can notice, that the expression under expectation in (2.13) is a weighted squared loss with $\gamma_i = \exp(-y_i F(x_i)) \Rightarrow$ each Weak Learner minimizes a second order approximation of the exponential loss. $\square$

This proposition 2.3 shows, that it's not just a luck that Boosting happens to work good in practice, but rather it has mathematical guarantees to converge and decrease the overall error of the boosted classifier, provided that the updates of weights γ (and coefficients α for some implementations) are done in a proper way.

2.3.2 Random Forests

Random Forests have gained a lot of attention recently, particularly in the filed of Semantic Image Segmentation. The main reasons are that Random Forests show very good performance in terms of accuracy (at the level or even better than Boosting) and being very fast both to train and test. Random Forests were introduced by L. Breiman [11] and further developed by P. Geurts *et al.* [18].

Definition 2.4. A *Random Tree* is a function $f(x, \theta_m)\colon \mathcal{X} \to \mathcal{Y}$ or shortly $f_m(x)$, where $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{1, 2, \ldots, n\}$ and θ_m represents a set of parameters learned while training.

In the above definition θ_m denotes any stochastic parameters (like splitting functions, splitting values, *etc.*) which define behavior of nodes of the tree while traversing each sample down the tree. Then we can say:

Definition 2.5. A *Random Forest* is a collection of M Random Trees $\mathcal{F} = \{f_1, f_2, \ldots, f_M\}$.

On the other hand, Random Tree is a binary Decision tree with *decision* and *leaf* nodes. Every decision node has an associated test of the form $g(x) > \theta$ which decides on the left/right propagation of a sample. $g(x)$ can be any real- or vector-valued function which output should be meaningful in terms of comparison with parameter θ. Leaf (or terminating) nodes hold probability distribution (or labeling). Detailed algorithms for building Random Forests can be found in Appendix A.

Training The algorithm starts at the root node having all the train instances (or a fraction of them in order to reduce correlation between different trees) S and then tries to separate the given set into two L (left) and R (right) sub-sets (that is if $L \subseteq S$ then $R = S \setminus L$) so that to decrease the overall impurity:

$$L(S) = \frac{|R|}{|S|} E(R) + \frac{|L|}{|S|} E(L) \tag{2.14}$$

$E(S)$ can be any measure of uncertainty. The usual choices are the Shannon Entropy $E(S) = -\sum_{i=1}^{n} p_i \log p_i$ or Gini index $E(S) = \sum_{i=1}^{n} p_i(1 - p_i)$, where n is the number of classes and p_i is the probability of i-th class in the given set S. Finding the best split is basically a combinatorial n^p-hard problem, because one needs to try every possible splitting function with every possible parameter. This is unreasonable in practice, so the general approach is to uniformly randomly sample a splitting function, its parameters (if any) and a splitting value (that is why Decision Trees built in such way are called Random). This is done K times and then the splitting setting $(g_i(x), \theta_i)$ which results in the highest impurity decrease is chosen and assigned to the decision node. P. Geurts *et* *al.* [18] showed that taking $K = \lceil\sqrt{d}\rceil$, where d is the dimensionality of the feature space, gives a suboptimal solution. This procedure goes on in a recursive manner until some stopping criterion is satisfied: maximum depth D is reached, the number of samples in the node is less than some integer value n, the impurity of the node $E(S)$ falls below some threshold, *etc.* Then a leaf node is grown and assigned a probability distribution (or labeling) of classes from the samples that reached the leaf node.

As the training samples may be unbalanced, that is the number of instances of each class is not the same, it is reasonable to compute weights $w_c = \sum_{i=1}^{N}[y_i = c]^{-1}$, where N is the number of samples, for each class $c \in \mathcal{Y}$ and multiply the distribution in all nodes with these weights, followed by L_1-normalization to ensure that the result of this operation is a probability distribution. This is quite a natural assumption, because some classes (like grass, sky, road, *etc.*) may occupy a lot of space on an image, while others (like pedestrian, a pet, poles, *etc.*) may be not that prominent, but still very important to be classified correctly. And for a classifier interested only in total error the most profitable strategy would be to predict rather a more frequent class than a rearer one. We think, that all classes should be treated equally. This procedure usually slightly increases the total error, but significantly decreases per class average error, which is a good trade-off in general.

Due to the way Random Trees are grown, one can put new samples into the tree even at test time, thus enabling to adapt the whole classifier to changing circumstances without the need to retrain unlike all other classifiers. This procedure is called on-line learning (A. Saffari *et al.* [33]).

Predicting Given a Random Forest and a number of samples the algorithm traverses each of the samples through each of the Random Trees until terminating in a leaf node and retrieving the corresponding probability distribution associated with that leaf node. Then all the probability distributions from all the trees of the ensemble are averaged per sample:

$$p(y|x) = \frac{1}{T}\sum_{m=1}^{M} p_M(y|x) \tag{2.15}$$

Finally, the class which has the highest probability is chosen for each sample:

$$C(x) = \underset{c\in\mathcal{Y}}{\operatorname{argmax}}\, p(c|x) \tag{2.16}$$

One can notice here that each sample can be tested in each tree absolutely independently which gives a way for efficient parallelization.

Breiman [11] gives the following definition concerning Random Forests:

Definition 2.6. The *margin function* for a random forest is defined as

$$mr(x, y) = p(y|x) - \max_{\substack{k \in \mathcal{Y} \\ k \neq y}} p(k|x) \tag{2.17}$$

and the strength of the set of classifiers $\mathcal{F} = \{f_1, f_2, \ldots, f_M\}$ is

$$s = E_{(\mathcal{X},\mathcal{Y})} mr(x, y). \tag{2.18}$$

The margin function (2.17) is basically the difference of true label probability and the highest probability of a class other than the true one. So, it defines the confidence of the whole classifier, and, obviously, for a correct classification $mr(x, y) > 0$ should hold. Using margin function (2.17) the generalization error can be defined as:

$$GE = E_{(\mathcal{X},\mathcal{Y})}(mr(x, y) < 0), \tag{2.19}$$

where the expectation is measured over the entire distribution of (x, y).

Using the definition of the strength (2.18) Breiman [11] defines an upper bound on the generalization error as:

$$GE \leq \bar{\rho}\frac{1 - s^2}{s^2}, \tag{2.20}$$

where $\bar{\rho}$ is the mean correlation between pairs of trees in the Random Forest (measured on how similar their predictions are). This is useful for analysis of the overall performance of the Random Forest as well as individual Random Trees. Ideally, in order to get as low as possible generalization error the individual trees should be highly independent and have good accuracy (strength). That is why each tree usually gets only a small portion of the whole training set in order to decrease correlation among trees.

Besides being accurate and fast, Random Forests also possess a number of properties we find very useful for our problem:

- They can handle noisy training data (wrong labels) very well. Being tolerant to label errors without significant loss in accuracy is a nice property on its own, but it is a crucial one for us.
- They are very fast both for training and testing. It was shown by T. Sharp [35] that both training and testing can be easily parallelized on GPU resulting in very little running times which open way to real time application of Random Forests as a classifier.

- As shown by A. Saffari *et al.* [33] it is possible to perform on-line learning during right during testing.

It is worth mentioning, that due to the way the splitting algorithm works with the train samples there is one significant drawback of Random Forests. As at the splitting node the algorithm might access potentially any sample from the train set and there is no way to somehow predict which samples might be used, the whole train set should always be stored in operating memory of the computer. And when parallelizing the building algorithm, for example by letting each of the trees to be built independently in a separate thread/process, one must assure that any thread gets access to the whole data it needs. For example, when separating the work between several computers without shared memory, a copy of the whole train set should be kept in the memory of each of the computers, thus creating additional traffic on the net and introducing some latency. When it comes down to parallelization of the algorithm on a GPU, one should always remember about this memory constraint, because modern graphic card have quite limited amount of available memory. It is actually an ongoing research on how to benefit from parallelization of the Random Forest building algorithm without dealing with the memory constraining problems.

But the main disadvantage of Random Forests is that as every tree is essentially a binary tree, then the upper bound on the number of elements (nodes) of the Random Forest is $\mathcal{O}(M2^{D+1})$. Which is quite a large number on its own, but also grows exponentially in the maximum depth D. For the sake of optimization we use array representation of each binary tree and this means that we have to allocate array of length $2^{D+1}-1$ elements before even starting to build the tree. Depending on the platform the implementation of Random Forest aims at, one should carefully choose the maximum depth D both in order to build a good classifier, but also not to run out of memory.

2.3.3 Structured Class-Label Prediction

In all ordinary approaches in Semantic Image Segmentation the pixel's label is taken as the label of the sample, thus making each sample absolutely independent in terms of label space $\mathcal{Y}$. But the natural observation says that usually label space $\mathcal{Y}$ shows some topological structure, which can be used as a good clue when segmenting an unseen image. Although this topological structures are present already in any training set as a part of ground truth images, they are still not considered as an important part of learning process by most of the classification approaches. As a result, most of such approaches result in very noisy labeling, *e.g.* see Figure 2.9. An approach overcoming this shortage was proposed by P. Kontschieder *et al.* [25]. They propose an algorithm,

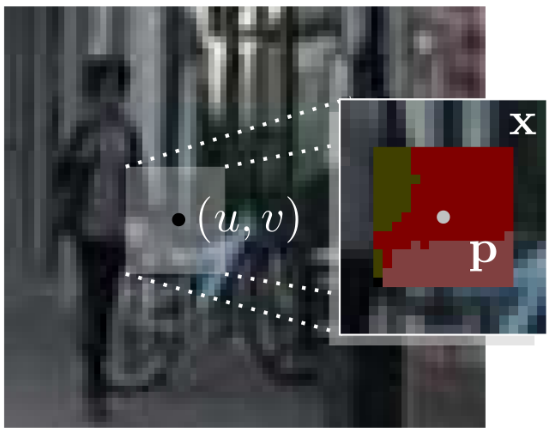

FIGURE 2.6: **An example of a patch label** (courtesy of [25]). Whereas ordinary approaches would consider only label of a pixel (u, v), they propose to use the local neighborhood **p** and therefore learn valid labelling transitions among adjacent object categories. Here: person, building, bicycle.

which uses *label patches* around each training pixel instead of just its single label. An example can be seen in Figure 2.6.

Their *Structured label space* $\mathcal{P}$ consists of $d \times d$ label patches, *i.e.* $\mathcal{P} = \mathcal{Y}^{d \times d}$. With $p_{(i,j)}$ the denote the (i, j)-entry of a label patch $p \in \mathcal{P}$. Additionally, they index the entries of every patch in a way that position $(0, 0)$ denotes the center of the patch and (i, j) are given relative to the central position. Using the new labels every training sample has now form of (x, p) with $x \in \mathcal{X}$ and $p \in \mathcal{P}$. Basically, this approach redefines the way any learning algorithm works with the labels. For example, they use an ordinary Random Forest with patches of simple color-based features 2.2.1. But in general with certain adaptations any other algorithm can be changed in order to work with the new structured label space $\mathcal{P}$. As we are also using Random Forest in this work, we adopted their approach.

Training As it was mentioned in the previous paragraph in case of Random Forests incorporation of the new approach is very easy. As the features are dealt with in the same way as before, all splitting procedures and growing of trees are done in the same manner as before. The only difference at this step is that one label patch π is stored in each leaf node instead of distribution of classes. For example, consider a number of label patches $\mathcal{P}_t \subseteq \mathcal{P}$ that reached a leaf node t during the procedure of growing a tree. Then one patch which represents the mode of distribution of patches in $\mathcal{P}_t$ is selected as the final labeling. In order to keep the cost of this step low they propose to use the pixel independence assumption when computing the joint probability of a patch p as

$$P(p|\mathcal{P}_t) = \prod_{i,j} P^{(i,j)}(p_{(i,j)}|\mathcal{P}_t), \tag{2.21}$$

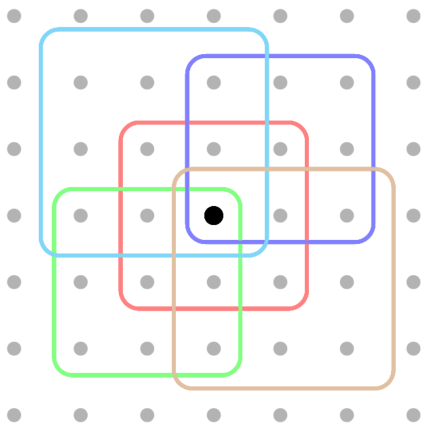

FIGURE 2.7: **Fusion of structured predictions** (courtesy of [25]). Each pixel collects class hypothesis from the structures labels predicted for itself and neighboring pixels, which have to be fused into a single class prediction. For clarity reason, only 5 out of 9 label patches are drawn.

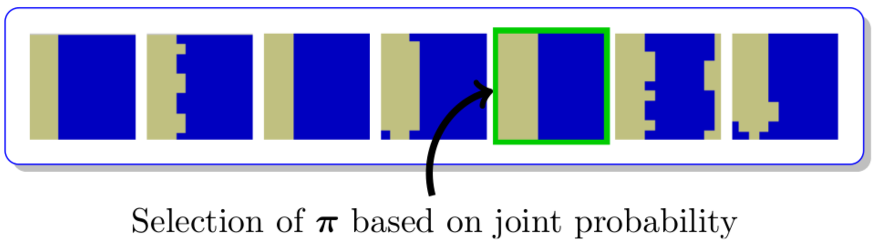

FIGURE 2.8: **Label patch selection based on joint probability** (courtesy of [25]). Example of label patches reaching a leaf during training. A label patch π is selected based on the joint probability distribution of labels in the leaf.

where $P^{(i,j)}(p_{(i,j)}|\mathcal{P}_t)$ represents the marginal class distribution of pixel position (i,j) over all label patches $\mathcal{P}_t$. The label patch π is finally selected for the leaf node t as the one in $\mathcal{P}_t$ which maximizes the the joint probability as

$$\pi = \operatorname*{argmax}_{p \in \mathcal{P}_t} P(p|\mathcal{P}_t). \tag{2.22}$$

An example of such patch selection is given in Figure 2.8. In order for the splitting algorithm of Random Forest to work correctly, in each splitting node equation (2.14) should be evaluated and as this function awaits some probability distribution over one pixel instead of over patches a random patch position (i,j) is chosen and marginal probability $P^{(i,j)}(\cdot|\mathcal{P}_t)$ is used.

Predicting At test time every sample is routed through each tree of the forest $\mathcal{F} = \{f_1, \dots, f_M\}$ and a bunch of label patches $\mathcal{P}_\mathcal{F}$ is gathered. Then again the patch which maximizes the joint probability is chosen

$$p = \operatorname*{argmax}_{\hat{p} \in \mathcal{P}_\mathcal{F}} P(\hat{p}|\mathcal{P}_\mathcal{F}), \tag{2.23}$$

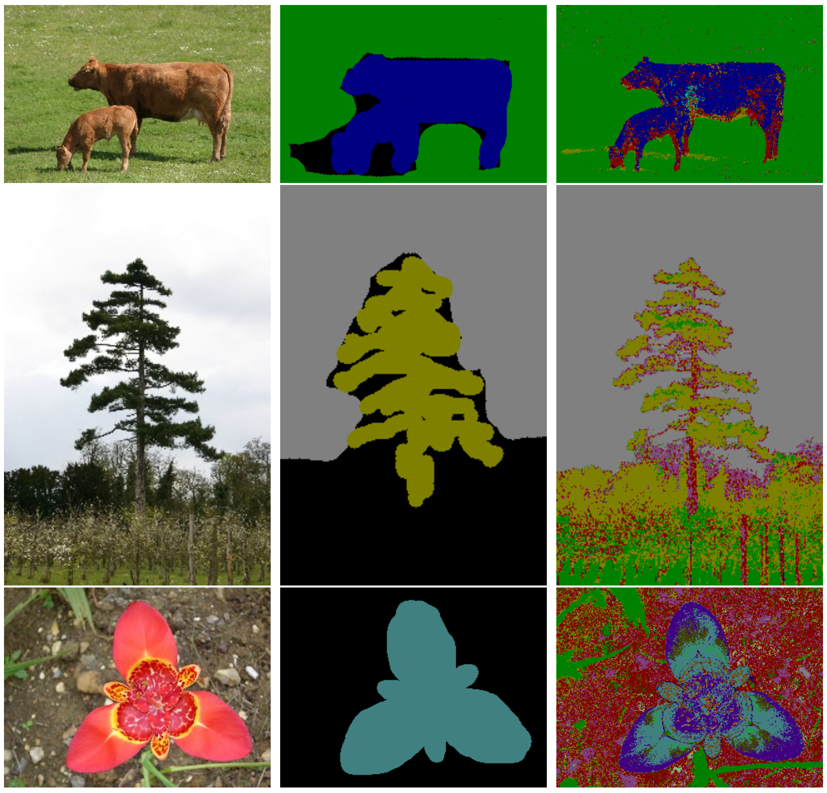

FIGURE 2.9: **Example of noisy labelings**. Columns denote from left to right: original image, ground truth image, output of a classifier (Random Forest in this case).

where $P(\hat{p}|\mathcal{P}_{\mathcal{F}})$ is defined as (2.21). As opposed to the standard semantic segmentation algorithms which assign one label to every test sample, structured class-label classifier produces a label patch which also covers some neighboring pixels. Particularly, if a label patch p has width $d \times d$ dimensions, then every pixel (except for the boundary ones of course) is covered by exactly d^2 label patches. So, here the question of fusion of those patches arise. One can think of many ways of doing this but we adopt the simplest, but still good performing, strategy as proposed by the authors, when the most voted class per pixel is selected as the pixel label. An example of this fusion approach can be seen in Figure 2.7.

We found this paper to be particularly interesting in our task, because this new approach of dealing with labels was easy to incorporate into existing framework of our Random Forest implementation and we are interested in scenarios of road scenes segmentation, which always show well-structured patterns especially for road, lane markings, cars, *etc.*, so being able to learn this is very helpful in our task.

2.4 Pairwise Potentials

Figure 2.9 shows that even a good classifier but which works on individual pixels produces a labeling which is subject a lot of noise. But the real world is in general piecewise smooth, so the intuition is that the resulting labeling of an image should be also piecewise homogeneous in its labels. As unary potentials cannot cope with this problem on their own an extension to the framework is necessary by adding pairwise potentials between different pixels in order to let them influence their peers and create a more natural looking segmentation. The most conventional approach is to connect each pixel only to a limited number of pixels in its neighborhood [38, 42]. The main reason for this is the restriction of the inference time which remain linear in the number of variables complexity as long as there is some fixed (relatively small) neighborhood of each pixel. This setting still allows (up to some fixed constant) linear inference time, but if one tries to connect each pixel to every other pixel in the image, the inference complexity becomes quadratic and therefore totally impractical, because it takes hours or even days [26] using traditional approaches like Markov Chain Monte Carlo sampling or graph cut based inference algorithms.

2.4.1 Fully connected CRFs

Basic CRF models are composed of unary potentials on individual pixels or image patches and pairwise potentials on neighboring pixels or patches [38, 17, 19, 42]. The resulting *adjacency* CRF structure is limited in its ability to model long-ranged connections within the image and generally results in excessive smoothing of object boundaries. Instead, Philipp Krähenbühl and Vladlen Koltun [26] propose to use a *fully connected* CRF that establishes pairwise potentials on all pairs of pixels in the image. An example of a fully connected CRF can be seen in Figure 2.10 and looks much more complicated in comparison with a usual approcah like 2.1.

The main restriction here is inference time because even a low resolution image will have tens of thousands of nodes and billions of edges and the inference algorithm has complexity $\mathcal{O}(n^2)$ in the number of vertices. In their paper they propose a highly efficient inference algorithm for fully connected CRF models in which the pairwise edge potentials are defined by a linear combination of Gaussian kernels in an arbitrary feature space. The algorithm is based on a mean-field approximation to the CRF distribution. This approximation is iteratively optimized through a sequence of message passing steps, each of which updates a single variable by aggregating information from all other variables. Their main observation in this context was that a mean field update of all variables in a fully connected CRF can be performed using Gaussian filtering in feature space. This

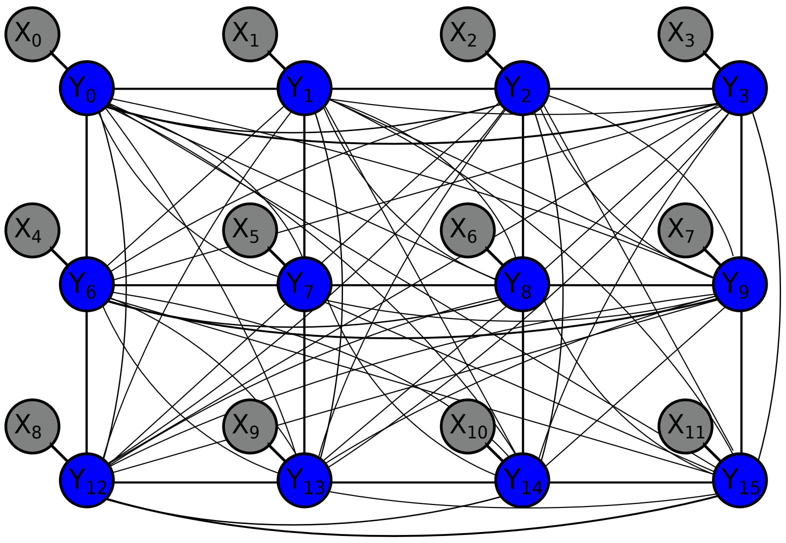

FIGURE 2.10: This is an example of a fully connected Conditional Random Field over a 4x3 image. $\mathrm{Y}_0, \dots, \mathrm{Y}_{11}$ denotes unkown labels of pixels, $\mathrm{X}_0, \dots, \mathrm{X}_{11}$ denotes observations, grey color means that variable X_i is conditioned on. Here every pixel is connected to every other pixel in the image. The number of edges grows qaudratically in the number of pixels.

allowed to reduce the computational complexity of message passing from quadratic to linear in the number of variables by employing efficient approximate high-dimensional filtering [7]. The resulting approximate inference algorithm is sublinear in the number of edges in the graph.

As it was mentioned in 2.1 probability of a certain labeling $\mathbf{Y} \in \mathcal{L}^N$ can be expressed as:

$$P(\mathbf{Y}|\mathbf{X};\theta) = \frac{\exp - \sum_{c \in \mathcal{C}_G} \phi_c(\mathbf{Y}_c|\mathbf{X};\theta)}{Z(\mathbf{X};\theta)}, \tag{2.24}$$

where the term $E(\mathbf{Y}|\mathbf{X};\theta) = \sum_{c \in \mathcal{C}_G} \phi_c(\mathbf{Y}_c|\mathbf{X};\theta)$ is usually called the Gibbs energy of the labeling $\mathbf{Y}$. The maximum a posteriori (MAP) labeling of the random field is $\mathbf{Y} = \operatorname{argmax}_{\hat{Y} \in \mathcal{L}^N} P(\hat{Y}|\mathbf{X};\theta)$.

Then in the fully connected pairwise CRF model, where G is a complete graph on $\mathbf{Y}$ and $\mathcal{C}_G$ is the set of all unary and pairwise cliques, the Gibbs energy can be written as

$$E(\mathbf{Y}|\mathbf{X};\theta) = \sum_i \psi_u(y_i|x_i;\theta) + \sum_{j \neq i} \psi_p(y_i, y_j|x_i, x_j;\theta), \tag{2.25}$$

where i and j range from 1 to the number of pixels in the image N. The unary potential $\psi_u(y_i|x_i;\theta)$ is computed independently for each pixel by a classifier that produces a probability distribution over the label assignment y_i given pixel observation x_i.

The pairwise potentials in their model have the form

$$\psi_p(y_i, y_j|x_i, x_j; \theta) = \mu(y_i, y_j)k(\mathbf{f}_i, \mathbf{f}_j) = \mu(y_i, y_j)\sum_{m=1}^{K}\omega^{(m)}k^{(m)}(\mathbf{f}_i, \mathbf{f}_j), \tag{2.26}$$

where each $k^{(m)}$ is a Gaussian kernel $k^{(m)}(\mathbf{f}_i, \mathbf{f}_j)) = \exp\left(-\frac{1}{2}(\mathbf{f}_i - \mathbf{f}_i)^T \Lambda^{(m)} (\mathbf{f}_i - \mathbf{f}_i)\right)$, the vectors $\mathbf{f}_i$ and $\mathbf{f}_j$ are feature vectors of pixels i and j correspondingly in an arbitrary feature space which are the result of some feature transform applied to observations x_i and x_j, $\omega^{(m)}$ are linear combination weights, and μ is a label compatibility function. Each kernel $k^{(m)}$ is characterized by a symmetric, positive-definite matrix $\Lambda^{(m)}$, which defines its shape. A simple label compatibility function μ is given by Potts model, $\mu(y_i, y_j) = [y_i \neq y_j]$.

For multi-class image segmentation and labeling they use contrast-sensitive two-kernel potentials, defined in terms of color vectors I_i and I_j and positions p_i and p_j:

$$k(\mathbf{f}_i, \mathbf{f}_j) = \omega^{(1)}\exp\left(-\frac{|p_i - p_j|^2}{2\theta_\alpha^2} - \frac{|I_i - I_j|^2}{2\theta_\beta^2}\right) + \omega^{(1)}\exp\left(-\frac{|p_i - p_j|^2}{2\theta_\gamma^2}\right), \tag{2.27}$$

where $\mathbf{f} = \{I_R, I_G, I_B, p_x, p_y\}$ which contains RGB color information of a pixel at location (x, y). The first term in this equation (*appearance kernel*) is inspired by the observation that nearby pixels with similar colors are likely to have the same labels. The degrees of nearness and similarity are controlled by parameters θ_α and θ_β. The second term (*smoothness kernel*) removes small isolated regions [38].

In general, the proposed framework of this form (2.26) is a very powerful and highly customizable tool for expressing almost any possible combination of pixels' interactions, despite being constrained only to Gaussian-based potentials.

Instead of computing the exact distribution $P(\mathbf{Y}|\mathbf{X}; \theta)$, the mean field approximation computes a distribution $Q(\mathbf{Y})$ that minimizes the KL-divergence $\mathbf{D}(Q||P)$ among all distributions Q that can be expressed as a product of individual marginals $Q(\mathbf{Y}) = \prod_i Q_i(y_i)$. Minimizing the KL-divergence, while constraining $Q(\mathbf{Y})$ and $Q_i(y_i)$ to be valid distributions, yields the following iterative update equation:

$$Q_i(y_i = l) = \frac{1}{Z_i}\exp\left(-\psi_u(y_i) - \sum_{l' \in \mathcal{L}}\mu(l, l')\sum_{m=1}^{K}\omega^{(m)}\sum_{j \neq i}k^{(m)}(\mathbf{f}_i, \mathbf{f}_j)Q_j(l')\right). \tag{2.28}$$

This update equation leads to the following iterative algorithm:

```
initialize $Q$;
while not converged do
    $\tilde{Q}_i^{(m)}(l) \leftarrow \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l)$ for all $m$;// message passing step
    $\hat{Q}_i(y_i) \leftarrow \sum_{l \in \mathcal{L}} \mu^{(m)}(y_i, l) \sum_{m=1}^{K} \omega^{(m)} \tilde{Q}_i^{(m)}(l)$;// compatibility transform
    $Q(y_i) \leftarrow \exp\left(-\psi_u(y_i) - \hat{Q}_i(y_i)\right)$;// local update
    normalize $Q_i(y_i)$;
end
```

Algorithm 3: Mean field approximation algorithm in fully connected CRFs

A closer look at the message passing step helps to realize that this process can be rewritten in the form of convolution with a Gaussian kernel $G_{\Lambda^{(m)}}$ as follows

$$\tilde{Q}_i^{(m)}(l) = \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l) = \sum_{j \in V} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l) - Q_i(l) = \left[G_{\Lambda^{(m)}} * Q(l) \right](\mathbf{f}_i) - Q_i(l). \tag{2.29}$$

They subtract $Q_i(l)$ from the convolved function $\bar{Q}_i^{(m)}(l) = \left[G_{\Lambda^{(m)}} * Q(l) \right](\mathbf{f}_i)$ because convolution sums over all elements, while message passing does not sum over Q_i.

It is well known that convolution can be performed efficiently in Fourier domain by just multiplication of complex values of the transformed signal with the corresponding values of the transformed convolution kernel, and the discrete Fourier transform (DFT) can be efficiently performed in time $\mathcal{O}(n \log n)$, *e.g.* using a freely available implementation FFTW [2]. But the authors went even farther and made use of a paper by A. Adams *et al.* [7] who presented an algorithm which allows to perform high-dimensional Gaussian convolution in linear in the number of elements time $\mathcal{O}(d^2 N)$, thus allowing to perform message passing step of the inference algorithm in linear time and making the complexity of the whole inference algorithm linear in the number of variables. This is the main reason why the proposed inference algorithm is so highly efficient. And also the reason why all potentials in their framework can be only Gaussians.

In this work we used the implementation provided by the authors which can be found at http://graphics.stanford.edu/projects/densecrf/.

Chapter 3

On-line Learning

The most obvious way of performing adaptation to changing conditions and, as a result, to the underlying class probability distribution is to use the result of the recently segmented images to improve the classifier in a smart way, in order not to break what is working correctly currently. One possible way, which we study in details in this work, is *on-line learning.*

On-line learning is a method of incorporating arriving information at the time of testing. This method is designed to help any classification algorithm to perform adaptation to new conditions on-the-fly. It is based on the fact that we put new samples, which have just been classified by the algorithm, together with the predicted label straight into the classifier and re-train it. Obviously, the main challenge here is how to choose the new samples which we can trust and reject the false positives, because, unlike in off-line learning, algorithms which try to learn at testing time do not have access to ground truth labels. Correspondingly, if too many incorrectly labeled samples are added into the classifier, then it might happen in some time that the algorithm will start working incorrectly in terms the initially specified aim, *e.g.* it may start performing incorrect classification or outputting just noise due to getting confused by the lately added samples. This affect is usually called *model drift.*

As on-line methods look attractive for real-life applications, where conditions might change from those present in the training data, there has been made several researches in this direction. One of the earliest is [31], who experimented with boosting, but the best results were obtained in [33], who showed that on-line learning can be easily performed with Random Forests. They also show that on-line Random Forests converge to off-line ones with time, so we in our experiments perform not on-line but rather *batched learning*, as this is just a lower bound on accuracy one can achieve with pure on-line learning. And on-line learning can be seen as just a technique of speeding up the re-training process. We rather concentrate on the problem of adding faithful samples.

By batched learning we mean, that we take a number of consequent images (in all experiments we took 10 images), perform segmentation with the current classifier and then process these images trying to find faithful samples and add them to the training set of the classifier together with the predicted labels for these samples. We want to stress that we do not use the ground truth images in any possible way other than for computing test error rates.

We also combine off-line learning together with on-line learning by first training a classifier on the old training set (off-line part) and then refine it by adding new samples at the testing time (on-line part). Correspondingly, we call the old training set, which is used for training in off-line manner, as *off-line set*, and the new testing set, which is used for on-line learning, as *on-line set*.

3.1 Naive Approach

As the name of this method suggests, the idea behind this method is extremely simple: perform segmentation of a new batch of images, then select a number of pixels of each class and add them into the classifier, which is later re-trained on the old and new new samples.

At testing time we get an output probability distribution $P(x_{(i,j)})$ from our classifier for each pixel (i, j) and the predicted class-label for it

$$c^* = \operatorname*{argmax}_{c \in \mathcal{Y}} P(x_{(i,j)} = c). \tag{3.1}$$

Then, as in such setting there is now way of checking whether the given segmentation is correct or not, we take features of only those pixels, for which the following holds

$$P(x_{(i,j)} = c^*) > \lambda, \tag{3.2}$$

where λ is some threshold parameter. We found $\lambda = 0.8$ to perform best in our experiments. High probability $P(x_{(i,j)} = c^*)$ should indicate high confidence of the classifier in the predicted label. As there are no any ways of somehow checking the correctness of the predicted label, we can only rely on the confidence of the classifier.

The new samples may be unbalanced, *i.e.* have different number of representatives of each class, so we sort the samples of each class $c \in \mathcal{Y}$ in decreasing order of corresponding

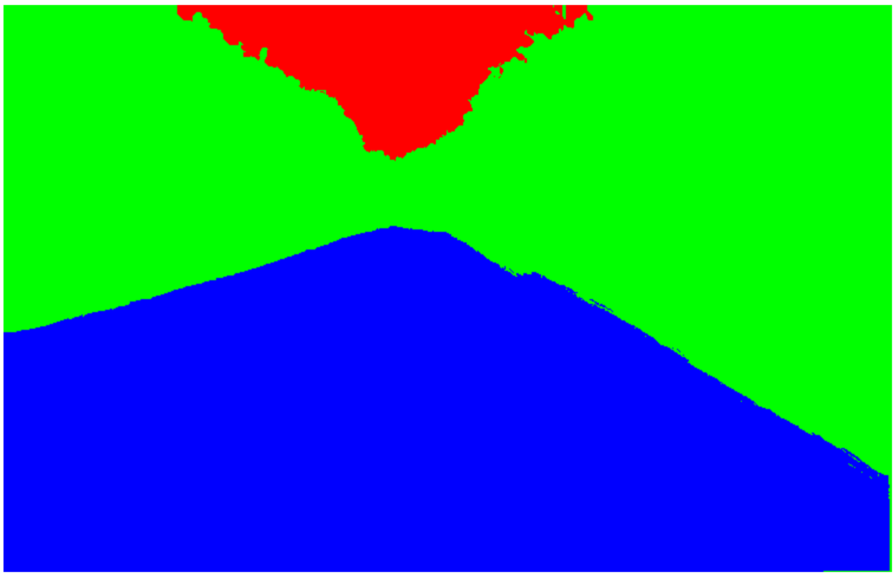

FIGURE 3.1: **Prior labeling computed on the old set**.

probabilities $P(x_{(i,j)} = c)$, then find the minimum number of samples N_{min} among all classes and take only $0.8N_{min}$ new samples of each class $c \in \mathcal{Y}$. This helps to keep balance between classes and does not allow one class to prevail on the others. We employ this strategy in all subsequent approaches.

This approach depends very much on the quality of the off-line classifier trained on the off-line data, because if it does not perform reasonably good segmentation initially, there no way to correct it in the future. Basically, this is rather an empirical approach as it does not have any mathematical guarantees to eventually converge regardless of the initialization.

3.2 Adding Constraint in the Form Prior Labeling

It was mentioned in the Section 3.1, taking new samples with the predicted labels which have high confidence is not a reliable and optimal way of performing on-line learning. As we also use off-line trained classifier, then a natural extension to the naive approach would be to somehow employ the information contained in the off-line data. We adopt method from J. Alvarez *et al.* [9] who proposed to compute the class-histogram on the off-line data, then normalize it per-pixel and use it as a prior distribution to weigh the output probability distribution of the classifier at testing time.

We believe that adding the constraint in form of the prior labeling is reasonable, because the classifier is designed to be used for classification tasks which must have something in common with the off-line data. For example, Figure 3.1 shows the prior labeling computed for the old training data. Although our classifier will work with on-line data, which differ very much in appearance from the off-line data, it is still reasonable to assume that in any circumstances the “road” tends (in other words is more likely to be found here than anywhere else in the image) to be in front of the camera, around the middle of the lower part of the image, “sky” tends to be above the “road”, and “background” is likely to surround the “road”.

Computing per-pixel histogram by considering only labelings from the off-line dataset may leave certain bins empty due to not fully representative off-line data. So, we initialize all bins with the default value equal 1 in order to account for very low-probable situations but which still may occur in real-life. Otherwise, the corresponding prior probabilities will be 0 and this is extremely bad, because this won't be changed in the future, as probabilities are multiplied, and multiplication of arbitrarily large number with 0 will always result again in 0.

The images in the off-line dataset have dimensions 752x480. So, we compute histogram (initialized to all 1's) for each pixel and after per-pixel L_1-normalization we get a prior $P_{pr}^{(i,j)}$ for each pixel $(i,j), i = 1,\ldots,752, j = 1,\ldots,480$. Images in the on-line dataset all have various dimensions, so we perform nearest-neighbor sampling from the prior distribution $P_{pr}^{(i,j)}$. Then at testing time output probability distribution $P(x_{(i,j)})$ from our classifier for an image with dimensions $W \times H$ is element-wised multiplied with the corresponding prior:

$$\tilde{P}(x_{(i,j)}) \propto P(x_{(i,j)}) P_{pr}^{(\lfloor i\frac{480}{H}\rfloor, \lfloor j\frac{752}{W}\rfloor)}, \text{for all pixels } (i,j). \tag{3.3}$$

We also have the initial predicted class-label

$$c^* = \underset{c \in \mathcal{Y}}{\operatorname{argmax}} P(x_{(i,j)} = c). \tag{3.4}$$

Then we can check, that if

$$\tilde{P}(x_{(i,j)} = c^*) > \lambda, \tag{3.5}$$

where λ is some predefined threshold parameter, then we take the corresponding pixel's features together with the predicted label c^* as a new sample.

3.3 Bayesian Model Update for Scene Segmentation under Structured Scene Prior

We propose a new model to incorporate unlabeled data for scene labeling. Our approach is motivated by Bayesian tracking formulations like Condensation [21] and particle filters, as they are widely used in vision and robotics [13]. In contrast to previous work, we track an evolving model over time and use a scene segmentation prior in order to avoid drift.

Bayesian tracking models consist of two steps: first, the propagation of a state distribution over states x_{t-1} given the observations Z_{t-1} to the next time step, and, second, update by an observation at the new time step z_t. In robotics application there is frequently a control input c_{t-1} added. The propagation is performed in a Bayesian manner by integrating out the unobserved states x_{t-1}:

$$p(x_t|Z_{t-1}) = \int p(x_t|x_{t_1}, c_{t-1})p(x_{t-1}|Z_{t-1})\,\mathrm{d}x_{t-1}, \tag{3.6}$$

where $p(x_t|x_{t_1}, c_{t-1})$ is a state transition model. In robotics applications it is common to condition it on a potential control input c_{t-1}.

The second step takes a measurement $p(z_t|x_t)$ into account and arrives at a new distribution over states x_t via application of the Bayes' rule:

$$p(x_t|Z_t) = \frac{p(z_t|x_t)p(x_k|Z_{t-1})}{p(z_t|Z_{t-1})}. \tag{3.7}$$

In our application we are interested in tracking an evolving target distribution over models. Our aim is to add unlabeled data at each time step, but avoid model drift as it occurs in ordinary "self-training" scenarios. Rather than sticking to a single model hypothesis, we seek to model a distribution over model hypothesis. Therefore, we aim at updating a distribution over model hypothesis given labels $p(h_t|L_t)$. At each time step we are given a set of unlabeled images u_t that we use to update our model hypothesis h_t. The unobserved variables, that we are interested in, are the unknown scene labelings l_t and current distribution of model hypotheses h_t. There is also the unlabeled data u_t at time step t that we'd like to use in order to update our model. Figure 3.2 depicts the Bayesian network for our approach.

We describe the incorporation of the unlabeled examples in a Bayesian framework by integrating over all model hypothesis:

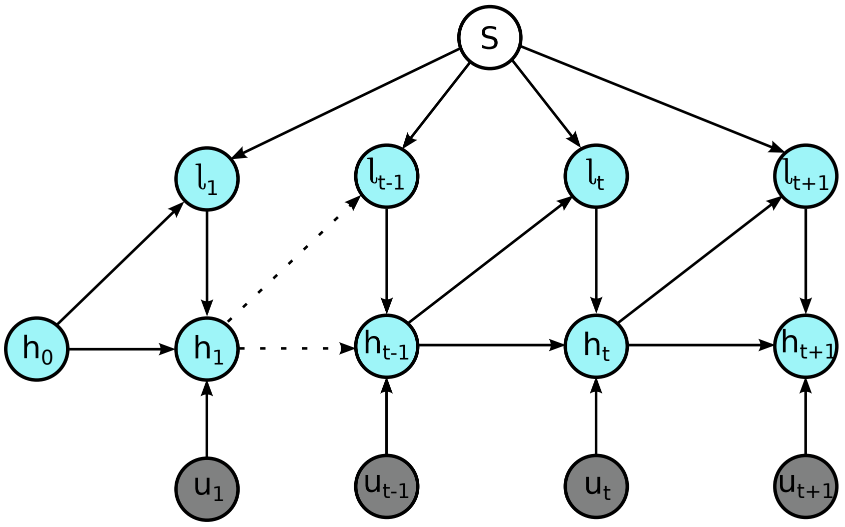

FIGURE 3.2: **Proposed Bayesian Network**. Gray color denotes observed variables which are images u_t to be labeled. Aqua color denotes hidden variables, which are unknown labelings l_t and current set of model hypotheses h_t. Parameter S is the scene labeling prior

$$p(h_t|L_{t-1}) = \int p(h_t|h_{t-1}, u_t)p(h_{t-1}|L_{t-1})\, \mathrm{d}h_{t-1}. \tag{3.8}$$

In the measurement step, we apply the Bayes rule in order to get the updated distribution over model hypothesis:

$$p(h_t|L_t) = \frac{p(l_t|h_{t-1}, S)p(h_t|L_{t-1})}{p(l_t|L_{t-1})}, \tag{3.9}$$

with

$$p(l_t|h_{t-1}, S) = p(l_t|h_{t-1})p(l_t|S), \tag{3.10}$$

where $p(l_t|h_{t-1})$ is the probability of a certain scene labeling prediction given a model hypothesis h_{t-1} and $p(l_t|S)$ is a scene labeling prior.

Consequently, scene labeling will also be performed by marginalization over the model distribution:

$$p(l|L_t) = \int p(l|h_t)p(h_t|L_t)\, \mathrm{d}h_t, \tag{3.11}$$

where l is the labeling of a test image, for which we want to do the prediction.

Inference We solve the above inference problem by a Particle Filter approach. At each time step the model distribution $p(h_t|L_t)$ is represented by a set of particles $s_t^{(n)}$ (random forests in our case) with weights $\pi_t^{(n)}$. Next the models are propagated to the next time step via $p(h_t|h_{t-1}, u_t)$. In traditional tracking application this transition is modeled with a deterministic part and a stochastic component. We propose a similar scheme that applies to the propagation of models. For the number n of desired particles, repeat:

1. Perform sampling (pick a particle s^i_{t-1} from $s^{(n)}_{t-1}$) from $p(h_{t-1}|L_{t-1})$ according to the weights $\pi^{(n)}_{t-1}$;
2. Sub-sample set of unlabeled images u_t to $\hat{u}_t$;
3. Predict labels $\hat{l}_t = \text{argmax}_l\, p(l|h_{t-1})$ for subset $\hat{u}_t$;
4. Retrain model using $\hat{l}_t$ and L_{t-1}.

Traditional tracking approaches would now follow up with a measurement in order to update the weights $\pi^{(n)}_t$. Similarily, we update the weight π^i_t of each sample s^i_{t-1} (model hypothesis) according to Equation (3.9) on the next batch of images. $p(h_t|L_{t-1})$ is the distribution represented by our particles after the propagation step from above and $p(l_t|h_{t-1}, S)$ is the product of the likelihood of the labeling times the likelihood of the labeling given the scene labeling prior. We don't compute the denominator - but rather directly normalize the weights of the particles $\pi^{(n)}_t$ to sum up to 1.

Chapter 4

A New More Challenging Road Scenes Dataset

In order to study the problem of adaptation we needed to find a dataset, which would exhibit considerable amount of appearance variation still representing the same scenario. Image segmentation is a very old and very well studied field of Computer Vision, but we were not able to find a dataset which would suit the above mentioned conditions. Conventional datasets are either just a collection of different images depicting different objects which have very little in common and such images cannot by put into one scenario, or the images in a dataset do not exhibit sufficient variations of visual conditions. Eventually, we decided to collect our own dataset of road scenes' images, which would feature the visual conditions we were looking for and represent one coherent scenario. We decided to take images of roads in different conditions, because this can allow direct application of our research in real-life.

4.1 Existing Datasets

There exist a large number of various datasets compiled on purpose for testing different segmentation approaches representing different conditions. The most popular datasets for testing general purpose segmentation algorithms are the MSRC [4] and VOC [6] datasets. In our work we made use of the MSRC dataset, as it was also used as the test set in all the original papers of the methods we implemented, to check the quality of our implementations.

We think that the most suitable setting for investigating into adaptive techniques is road scenes datasets. First of all, such datasets represent the field of application of our research which benefits most of all from the ability to adapt to changing visual conditions, which can help make, for example, driving safer. Besides, such datasets exhibit the inherent process of evolution of visual conditions through time which is the ideal setting for any adaptive algorithm, as the adaptation cannot be done instantly.

4.1.1 The Microsoft Research Cambridge (MSRC) dataset

The MSRC [4] dataset is a publicly available collection of photographs taken at different parts of the world and capturing a large number of objects classes. It contains 591 images and 23 classes, but all researchers leave out horse and mountain classes, as there are too few samples of them for training, so it is difficult for any classifier to faithfully learn them. But even the resulting 21 classes represents a wide variety of objects of different nature: building, grass, tree, cow, sheep, sky, airplane, water, face, car, bicycle, flower, sign, bird, book, chair, road, cat, dog, body, boat. The main disadvantage of this dataset is the fact that the provided ground truth images show very rough labeling of the objects. Besides, not all pixels are labeled and one has to account for background ("void") class both while training and testing. For certain methods (like [25]) this may require particular handling of such "void" pixels. When testing, the pixels, marked as background in the ground truth images, are not considered when computing error rates.

We use this dataset to show that our implementations of the methods described in Chapter 2 work exactly (up to the used features and certain parameters of the classifiers) as stated in the original articles. We use the standard for this dataset splitting into train and test images from [26] which can be found here `http://graphics.stanford.edu/projects/densecrf/textonboost/split.zip`, with the only difference that we combine Train and Validation set together.

4.1.2 Road scenes dataset

As our main interest lies in the field of image segmentation of road scenes we used the following dataset from [45] to evaluate implemented classification algorithms and features. This dataset was taken by the authors using an auto-mounted camera and driving along some German highways and roads, capturing real-time road scenarios. The dataset is available at `http://www.d2.mpi-inf.mpg.de/tudds`. This dataset contains 88 training labeled images and 88 testing labeled images. It also has a number of possible object class to color mappings, but we considered only the mapping which contains 3 classes: road, sky and all other category classes were merged into one class called background. We believe that this is a reasonable assumption, because we are mainly interested in correct detection of the "road" (blue) and all other classes in this

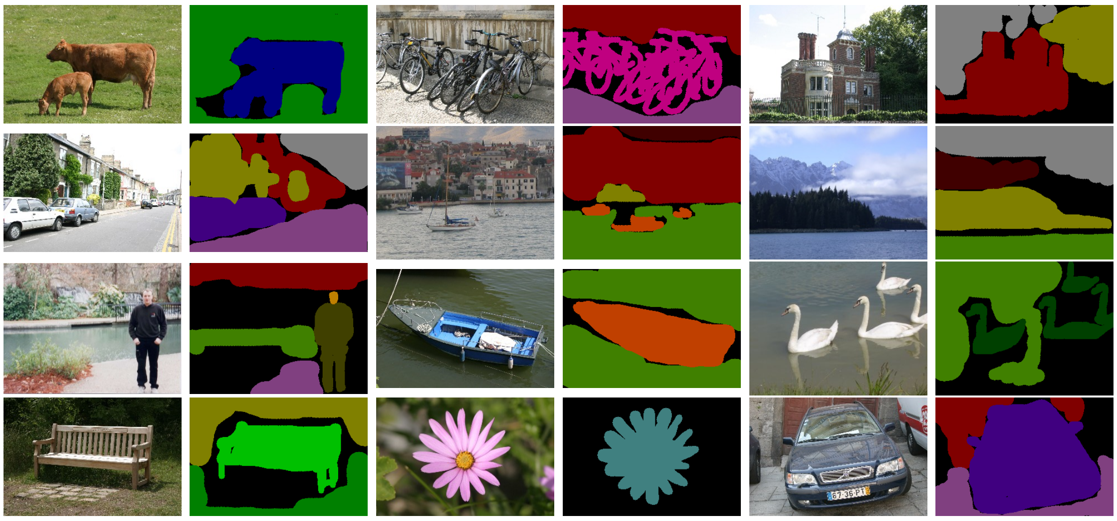

FIGURE 4.1: Some example images with the corresponding ground truth images from the MSRC [4] dataset

case may be considered as just “background” (green), and we also add “sky” (red) class as it occupies a lot of space in the provided images and helps not to turn our task into a binary problem.

We decided to take this dataset as an initial point in our research, because it was captured in real-life situations and we are particularly interested in real-life road scenarios. Besides, already this dataset exhibits some visually difficult situations like changes in object appearances due to motion blur effect, deep shadows which appear and disappear suddenly, changes in lightning conditions like over- or under-saturated regions.

There also exists a dataset from [10], but it has very limited labeling, which consists of just two classes (“road” and “background”), which we found to be not corresponding to our aims. And besides that dataset does not show any really difficult visual conditions.

4.2 The New Dataset

As we mentioned before, MSRC [4] dataset is just a collection of pictures of a variety of object classes and it does not suit our aim at all. We use it only for evaluation of the implemented methods, as all of them were evaluated on the original papers on this dataset. The dataset from [45] represents real-world road scenarios and exhibits visual conditions we are looking for, but still it was too little for us. As this dataset has only pictures of asphalted relatively clean highways going through forests or fields and we would like to look into even more challenging conditions.

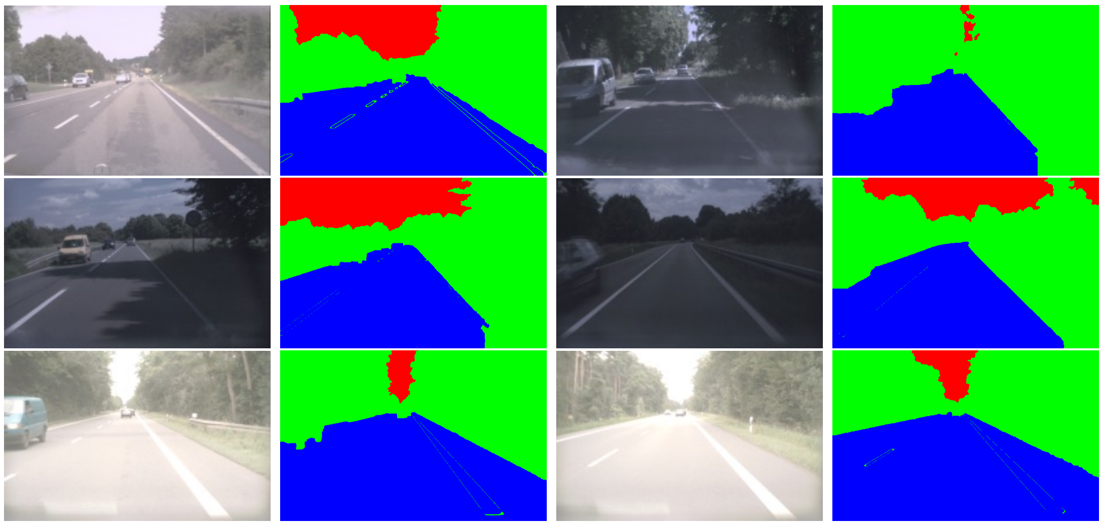

FIGURE 4.2: Some example images with the corresponding ground truth images from the [45] dataset

For this purpose we searched over the Internet, and particularly considered Flickr® (http://www.flickr.com/), looking for images depicting roads mostly in conditions which we called "autumn" and "winter", as ordinary datasets try to avoid such situations. We used the search engine of Flickr® and used such tags as "dirty roads", "autumn roads", "roads with mud", "winter roads". The result of our searching is 220 images about half of which represent roads in autumn conditions and another half roads in winter conditions. Figures 4.3 and 4.4 show a bunch of examples from each of the "seasons". We performed hand labeling of the gathered images using the GNU Image Manipulation Program and a tablet into three classes: road (blue), sky (red), and background (green). Unlike in the MSRC [4] dataset, we treat background as a meaningful class and both learn and predict it.

We believe that the introduced dataset introduces new challenges into the segmentation community, the ones which previously were omitted. Fore example, our autumn images may have leaves on the road (which are on their own of yellow or red color, which is different from the green color seen in the training set), different road cover (like asphalt of different colors or simply no any asphalt at all, just ground), sometimes there dirt or snow (or even no any road visible, but rather just path in the snow). If autumn images have at least something in common with the train set (and in all our experiments we use train set from [45]), then winter images have totally different visual appearance, which must create problems for most ordinary off-line methods.

For notational convenience, from here on we use the word *old* when we refer to the training and testing datasets from [45], and *new* when we mean the introduced more challenging dataset. And in all the following experiments (apart from the ones on the MSRC [4] dataset) we always use the *old* training set for training.

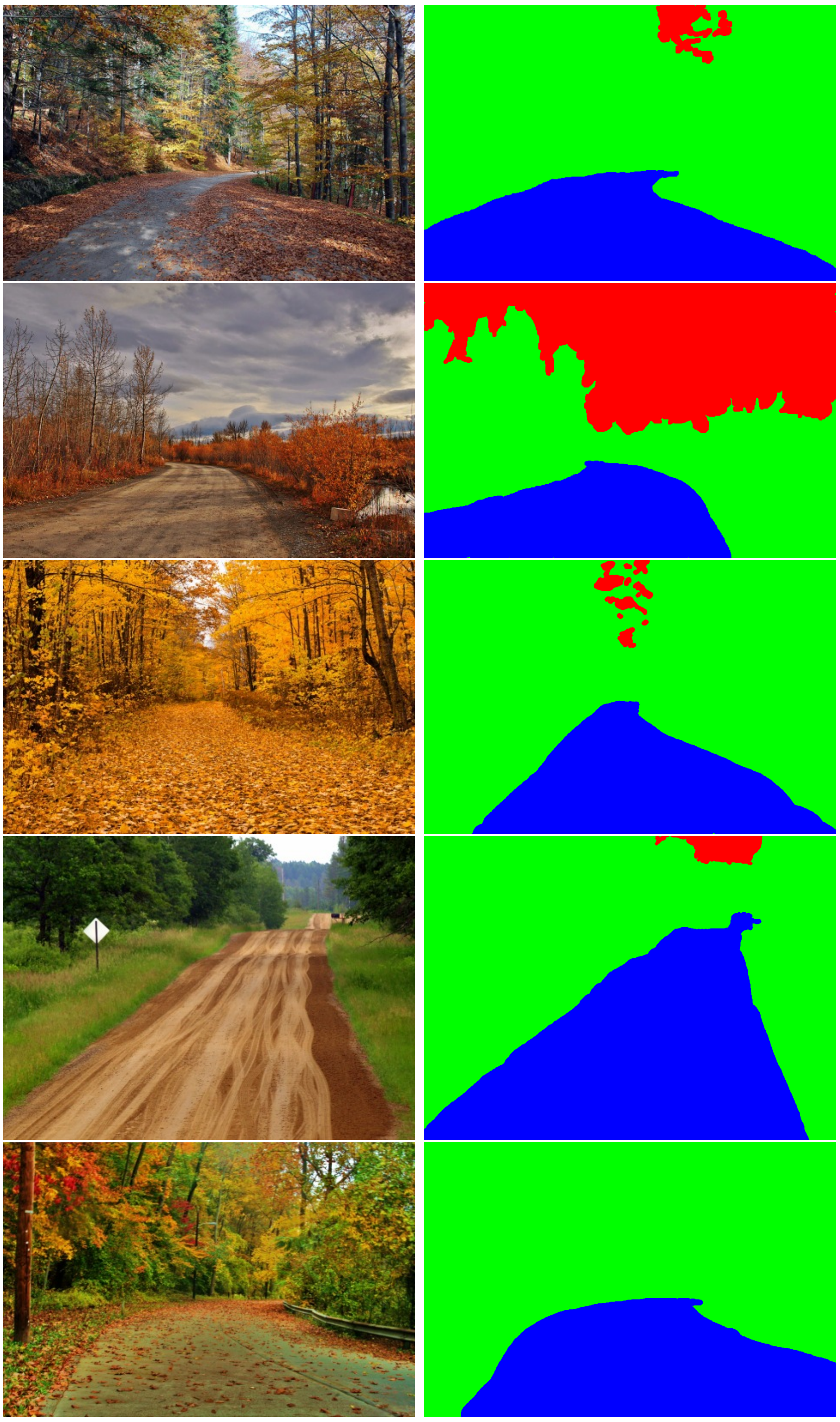

FIGURE 4.3: Some example images with the corresponding ground truth images from "autumn" part of the proposed dataset.

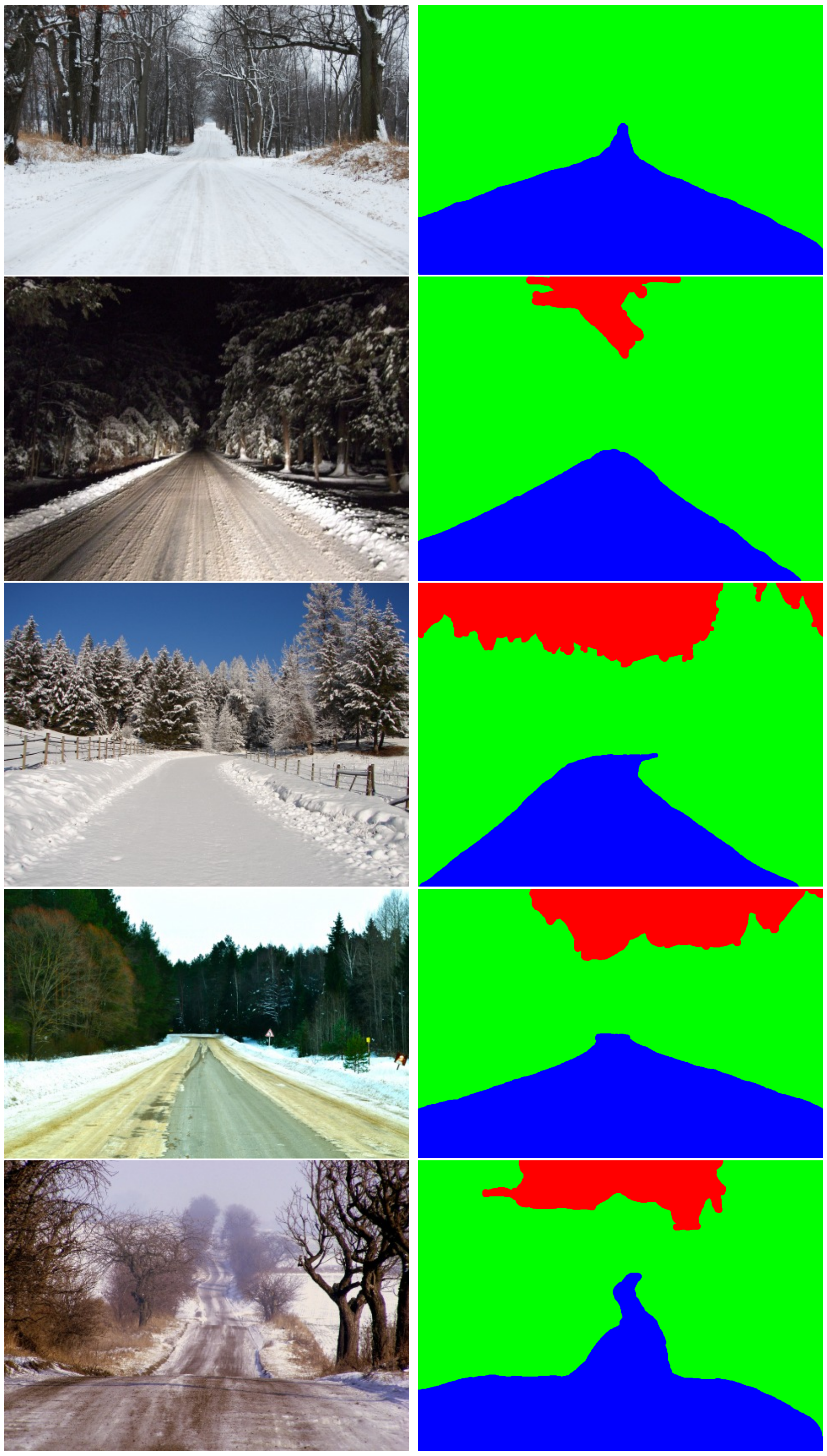

FIGURE 4.4: Some example images with the corresponding ground truth images from "winter" part of the proposed dataset.

Chapter 5

Experimental Results

We conducted a number of experiments to show that adaptation to new conditions is an important and required technique for situations when the underlying class-distribution changes over time. First, we start by experiments on the Random Forests to figure out the effect of different parameters and to show that Random Forests are inherently robust to noisy labels.

Then we continue by comparing our implementations of methods described in Chapter 2 on the MSRC dataset. We do it to investigate how different approaches perform relative each other and the state-of-the-art for general purpose image segmentation. Our main aim here was to find an approach which would have good accuracy, but also low training and testing time, because we needed this classifier to be used in on-line learning, which we chose to be our way of performing adaptation.

We conduct several experiments on the new dataset proposed in Chapter 4. We compare the performance of the state-of-the-art method for general purpose segmentation on this new dataset and the old one from [45]. We show that existing datasets try to avoid difficult visual conditions and that some standard approaches which do not perform adaptation to changing distribution tend to fail there.

In the end, we carried out a number of experiments with on-line learning methods described in Chapter 3. We show that even a naive on-line learning approach can improve segmentation results compared to standard approaches. We also show that our proposed Bayesian Model Update under Structured Scene Prior allows to adapt to changing visual conditions and as a result to improve segmentation results.

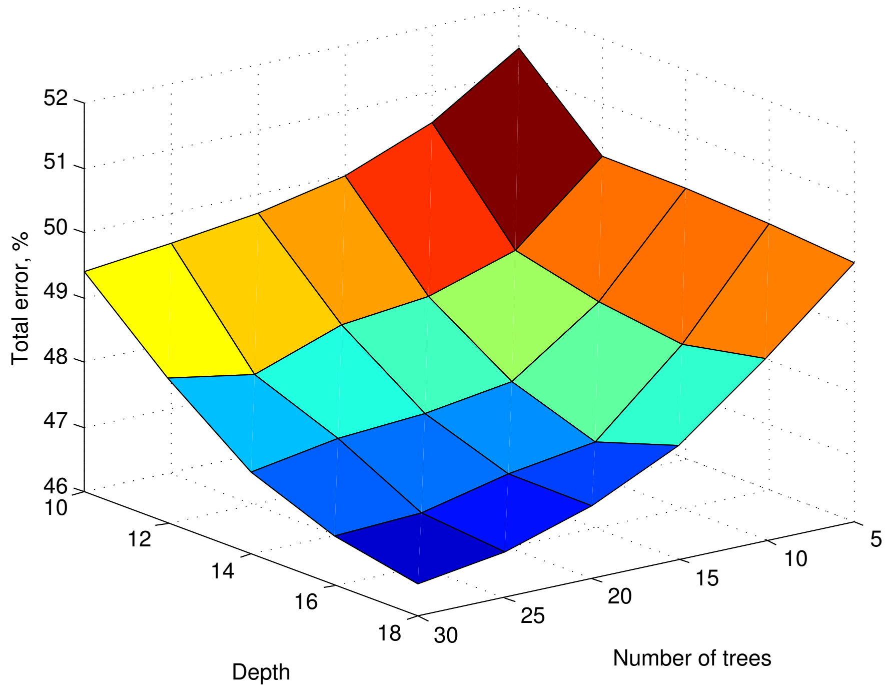

Figure 5.1: **Dependency of Test Error of Random Forest on parameters**.

5.1 Experiments with Random Forests

Random Forests have become very popular recently in image segmentation [12, 37, 25] for having good accuracy and low training and testing time, being robust to noise. To check these properties we conducted the following experiments.

5.1.1 Effect of Different Parameters

Usually in practice the minimum number of samples in a leaf node is set to quite a small number 3-5 and the parameter K is set to $\lceil\sqrt{d}\rceil$. So, the only parameters which are adjusted in a particular application task are the number of trees M in a Random Forest and the maximum depth of each tree D.

We studied the behavior of Random Forest by conducting the following experiment on the MSRC [4] dataset. We used the same splitting of the dataset as in [26] with the only difference that we added Validation set to the Training one. Then we trained and tested Random Forest the following combinations of parameters:

$$S_p = D \times M, \tag{5.1}$$

where $D = \{10, 12, 14, 16, 18\}, M = \{5, 10, 15, 20, 25, 30\}$.

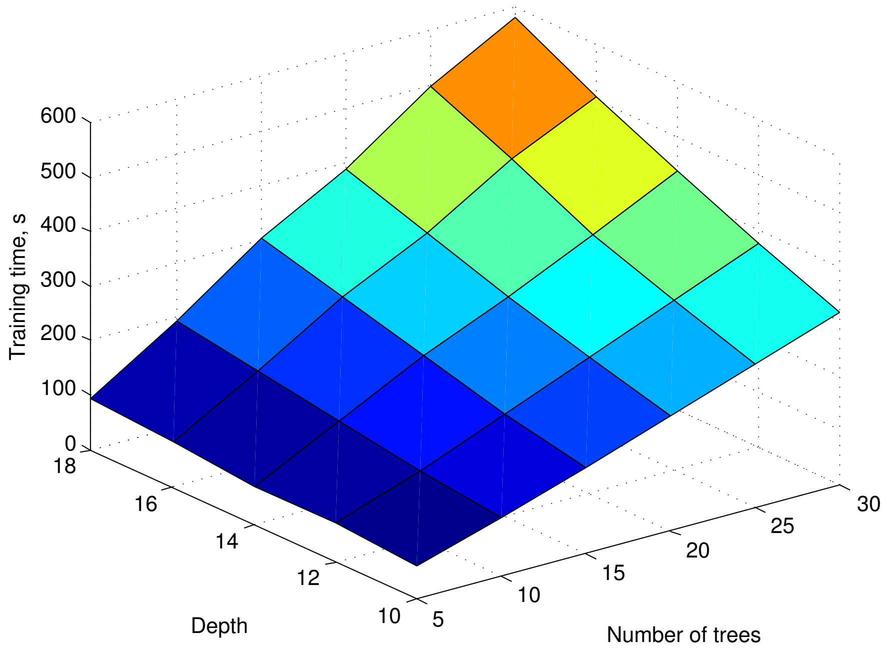

Figure 5.2: **Dependency of Training time of Random Forest on parameters**.

Figure 5.1 shows how Test Error changes with the change of parameters. The main observation here is that by increasing both the maximum depth D and the number of trees in the ensemble M the Test Error goes down. But parameter D tends to have greater contribution in the decrease of the error. This is due to the fact that by increasing D we allow exponentially more splits to be done which allows for the trees to perform better separation of the samples. But this also exponentially increases the size the Random Forest occupies in memory. But, in general, large parameters D or M are not required as Random Forest converge pretty quickly and further increase in parameters does not necessarily results in corresponding increase in accuracy, but definitely increases required memory. For example, [36] use only 3 trees but each of depth 20 and [37] use 5 trees each of depth 10.

Figure 5.2 shows the dependency of training time on the parameters. This are times for multi-threaded implementation with the help of Intel TBB [3]. Training time depends almost linearly on either of the parameters.

Figure 5.3 shows how test time depends on different parameters. Again, the dependency is almost linear in either of the parameters. The number of test images here was 256, so there is a whole bunch of parameters which allow for the test time to be less than a second, which can be called real-time performance, yet showing almost optimal accuracy.

Figure 5.4 shows the overall Running time for a Random Forest classifier *vs.* the Test Error. Here Running time is a sum of Training and Testing time for the corresponding

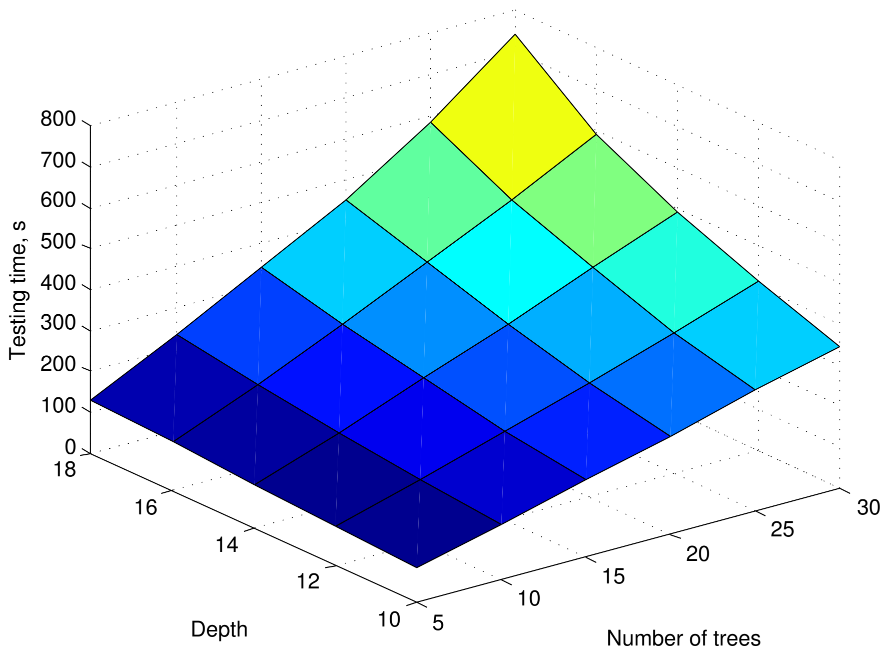

FIGURE 5.3: **Dependency of Testing time of Random Forest on parameters**.

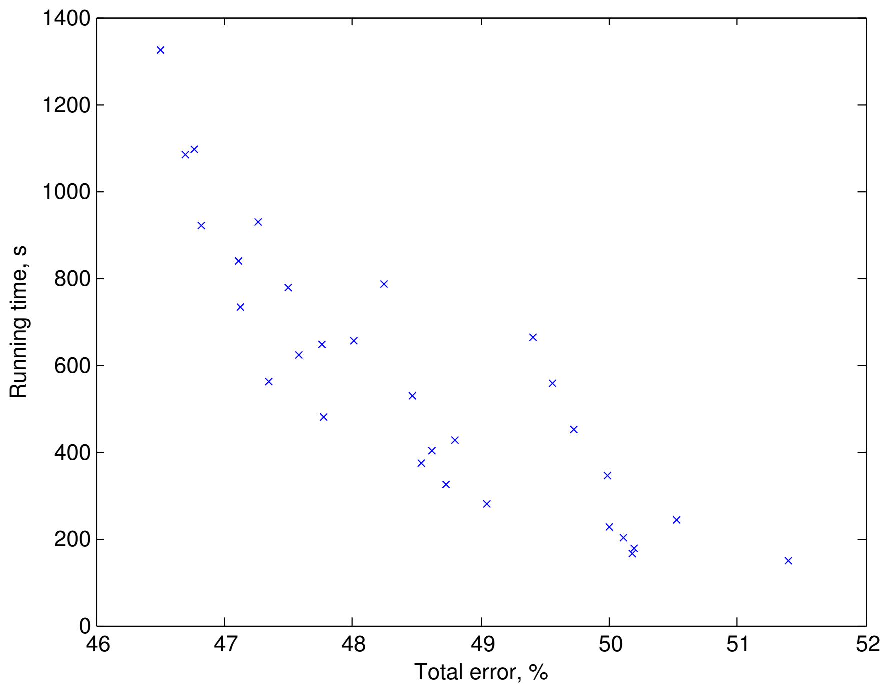

FIGURE 5.4: **Running time *vs*. Test Error for Random Forests**.

pair of parameters. Obviously, making Random Forests larger and deeper allows to decrease generalization error, but results in corresponding increase of the Running time. It is remarkable that Random Forests tend to converge rather quickly and further increase of parameters does not bring noticeable decrease of Test Error, while considerably (after some point already exponentially in dependency of Test Error) increase the overall Running time.

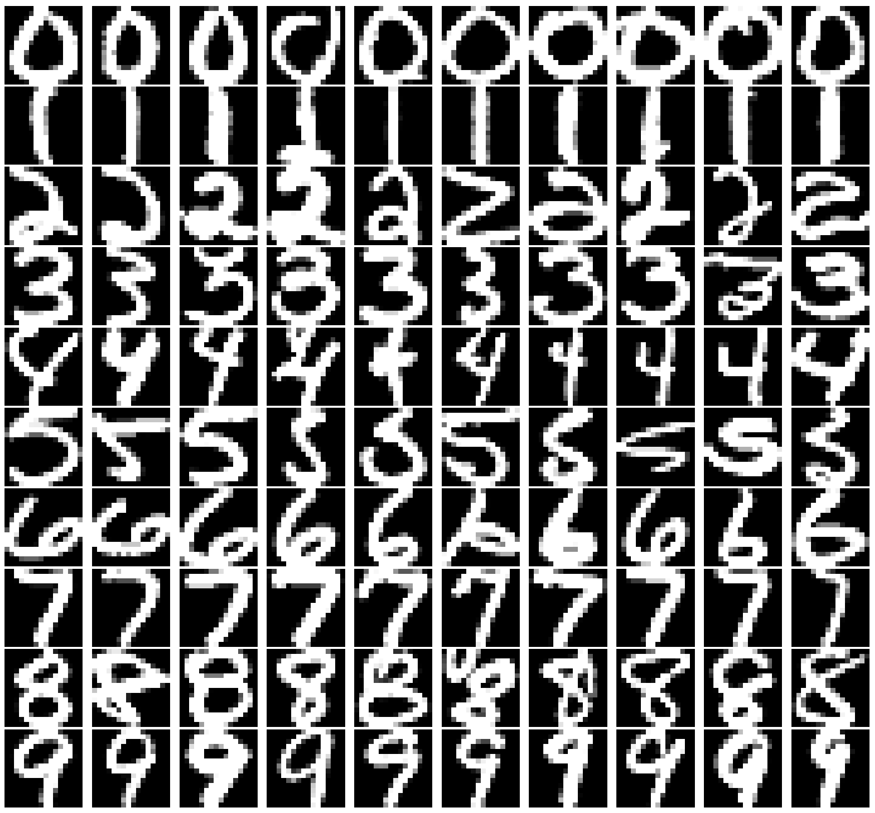
FIGURE 5.5: Some example images for each class from the USPS [5] dataset.

5.1.2 Robustness to Noise

In has been noticed [11, 18] that Random Forests are particularly robust to errors in the provided labels. The main reason for this behavior is that Random Forest does not try to find exact dependency between features and labels. When trying to find an optimal split at each splitting node, it basically works only with features and only then computes distribution of labels in the resulting splits. So, a moderate number of incorrectly provided labels will (hopefully) not distract the distribution so much, that the decision about the best split will be done incorrectly.

To test this property we conducted the following experiment. We took a subset (around 2000 images with each class having around the same number of images) of USPS [5] dataset which consists of images of handwritten number from 0 to 9 (some examples can be seen in Figure 5.5).

Then we randomly permuted labels for a portion of the training data, train several classifiers and tested on a non-permuted test-set. The results can be seen in Figure 5.6. On the X-axis you can see how many samples from the training data-set were changed their labels and on the Y-axis test error (averaged over 5 independent runs) of each of the classifiers trained on the whole train data set with the corresponding number

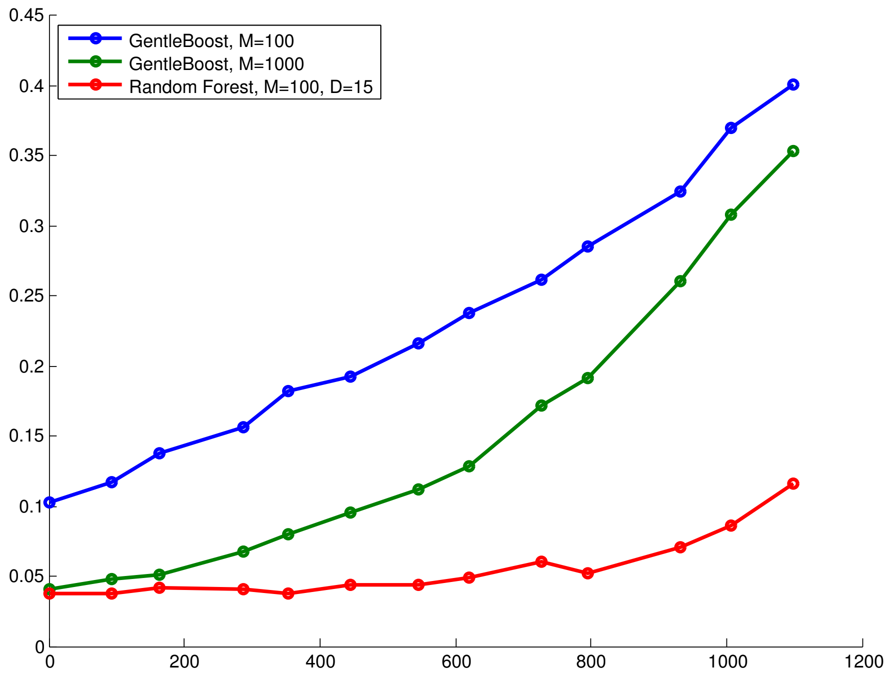

FIGURE 5.6: **Random Forest *vs.* GentleBoost on noisy train data**.

of changed labels. We compared here GentleBoost classifier with 100 and 1000 weak learners and a Random Forest with 100 trees each of depth 15. L. Breiman [11] agues that for example "AdaBoost is essentially Random Forest". He states that the number of iterations of Boosting roughly corresponds to the number of trees in a Random Forest. That is why we chose one GentleBoost with 100 weak learners as a direct competitor of our implementation of Random Forest with 100 Random Trees and also one GentleBoost with 1000 weak learners as even a higher challenge for the Random Forest classifier.

As it follows from the Figure 5.6, Random Forest is indeed very robust to noise. Even when around half of all the training samples are basically noise, it still shows reasonably good performance being more than 4 time better than GentleBoost.

Good robustness to noise of Random Forest was one of the main reasons why we eventually decided to use exactly Random Forests as the basis classifier in this work.

5.1.3 Discussion

Random Forests show good convergence rates and allow to get good performance for already small number of trees, which allows to perform fast training and testing using them. And this is particularly important for our application, because we need to perform re-training of the classifier preferably in real-time in our on-line learning approaches. And low test-time is a crucial property for real-world application, because there the segmentation of new images must be carried out in as little time as possible to ensure the least latency, because this will allow to use such classifier in systems which should be able to analyze and react in real-time. In these and the following experiments we made an observation that if the parameter setting of $M = 10, D = 15$ does not result in the expected accuracy then it is worth to rather consider further engineering in features or some other aspects, than trying to tune parameters of the Random Forest.

Robustness to noise is a very useful property in our application, because in on-line learning approaches we use samples with predicted labels which might exhibit a certain level of label noise. The on-line learning algorithms on their own try to minimize the number of wrong labels, but it is good that already the classifier is robust to noise, thus making the overall system even more stable.

5.2 Experiments on the MSRC dataset

In the next step of our research we implemented a number of image segmentation algorithms showing good performance in the original papers and which we think are suitable for our purposes. We needed to check several possibilities for the classifier and choose the best one from the point of view of the trade-off between accuracy and speed. We would like to stress that we would prefer a faster working classifier, because we need as low as possible running time in order to use this classifier later for on-line learning and we are also aiming for the real-world application, which requires the whole system to be able to perform in real-time. We carried out all experiments on the MSRC [4] dataset, because this is the dataset used in all the original papers, which describe the segmentation algorithms we employ in this work. And this dataset is just a de facto standard for comparing general purpose image segmentation algorithms.

We used Random Forest as a classifier in all our implementations. We started with just Random Forest which was working on individual pixels and CIE color values (Section 5.2.1). In Section 5.2.1 we expanded the classifier to be able to work on feature patches around a pixel (Figure 2.2). After that we added method of [25] to improve output labeling and make it looking more consisting, as this method allows to significantly decrease the level of noisy labels (Section 5.2.3). We also implemented algorithm of [38] (Section 5.2.4), because their textons are used in [26] and the latter is currently state-of-the-art algorithm showing the best result on the MSRC [4] dataset. Finally, we apply Fully-Connected CRF approach from [26] (Section 5.2.5), which improves not only visual segmentation results, but also considerably decreases the Test error (we use the code, kindly provided by the authors). We use the standard for this dataset splitting into train and test images from [26] which can be found

Method	Total error, %	Average error, %	Training time	Testing time
Random Forest	56.3	77.5	**19.65s**	1m 5.1s
RF on feature patches	47	69.4	51.64s	1m 24.1s
SCL	45.7	69.3	1m 10.1s	2m 40.35s
RF with Texton features	37.1	44.6	13m 44.3s	**36.83s**
Above + FC-CRF	26.9	36.6	13m 44.3s	4m 4.86s
Krähenbühl *et al.* [26]	**14.2**	**21.6**	around 1 day overall	

Table 5.1: Comparison of our implementations of methods described in Chapter 2 on the MSRC [4] dataset. Detailed description of the experimental settings can be found in 5.2.

here `http://graphics.stanford.edu/projects/densecrf/textonboost/split.zip`, with the only difference that we combine Train and Validation sets together.

All our implementations were using Random Forest as the basis classifier. The parameters were set to: number of trees in the ensemble $M = 10$, the maximum depth of each tree $D = 15$, number of random samples given to each tree was around $\frac{1}{4}$ of all the samples in the training set, unless stated otherwise the number of random split tries per splitting node was set to the default value of $\lceil\sqrt{d}\rceil$. As it is common for all experiments on the MSRC [4] dataset, we used the stride of 5 pixels in each dimension. We used simplest CIE color values as features, as we were mostly interested to see how different algorithms perform in the same conditions. Numerical comparison of different experimental settings can be seen in Table 5.1.

It is worth mentioning that the performance numbers of all the methods can be increased by simply training more trees, or making them deeper, or by changing other parameters of the classifier. But we were interested in performance numbers relative to each of the settings, but not the absolute performance numbers.

5.2.1 Setting 1: Pure Random Forest

In this experiment we trained a Random Forest which considered only local information of each pixel independently of the others. This is the most naive among all possible approaches, but even such a simple method shows results which are significantly better than random guessing. Images, which show more or less correct labelings, still suffer from high labeling noise, mainly because in this case the classifier works on individual pixels and does not consider any interactions between pixels.

Figure 5.7 shows some example outputs of this approach. An interesting observation is that even the most simple Random Forest is able to locate objects in the scene and very accurately follow their borders. But unfortunately it makes a lot of mistakes when

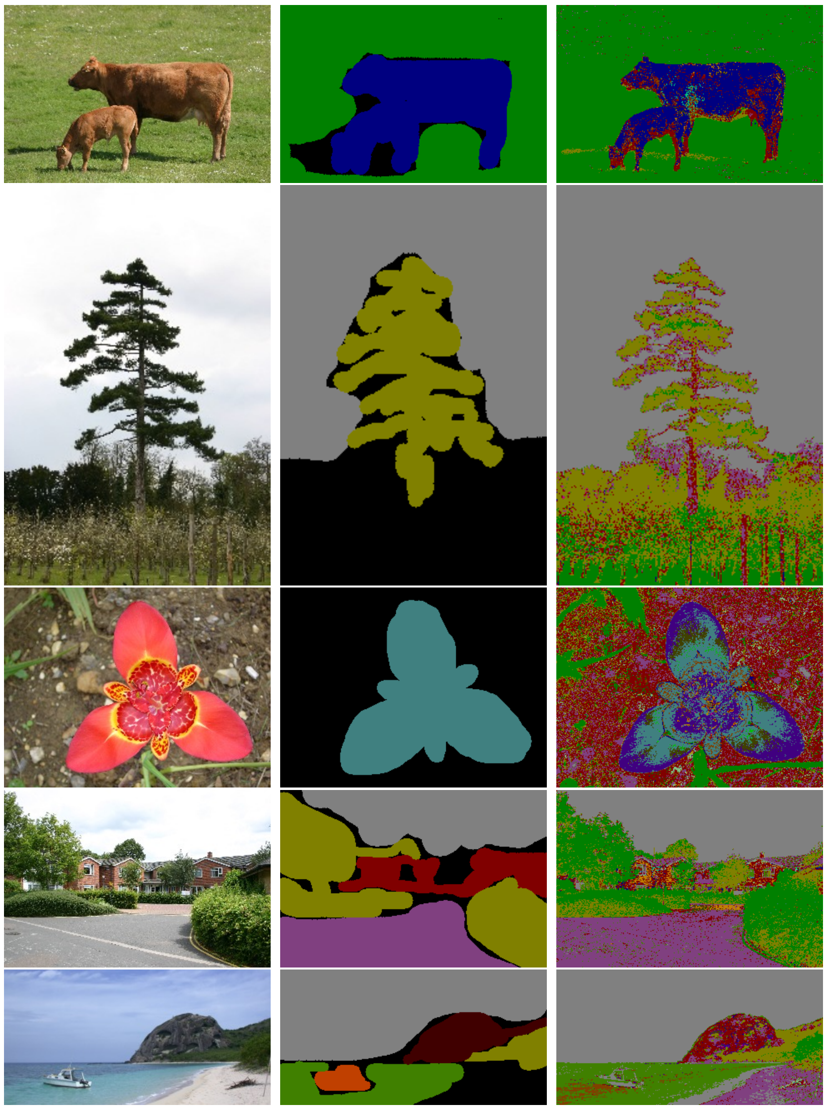

FIGURE 5.7: Columns denote from left to right: original image, ground truth image, output labeling of Random Forest 5.2.1.

trying to predict class-labels for segmented objects. So, just Random Forest can be used to successfully perform image segmentation, but not the semantic one.

5.2.2 Setting 2: Random Forest with Feature Patches

Allowing the Random Forest classifier to work on some neighborhood (Figure 2.2) in the feature space around each pixel allows to considerably improve segmentation accuracy, even without modifying the features themselves. Apart from taking raw feature value from one of the neighboring pixels we also made use of functions described in 2.2.1. Though feature patches do not modify the features themselves, but the change the way the classifier works with pixels and allows for it to capture, for example, long-distance gradients which are more distinctive than just pixel's color values.

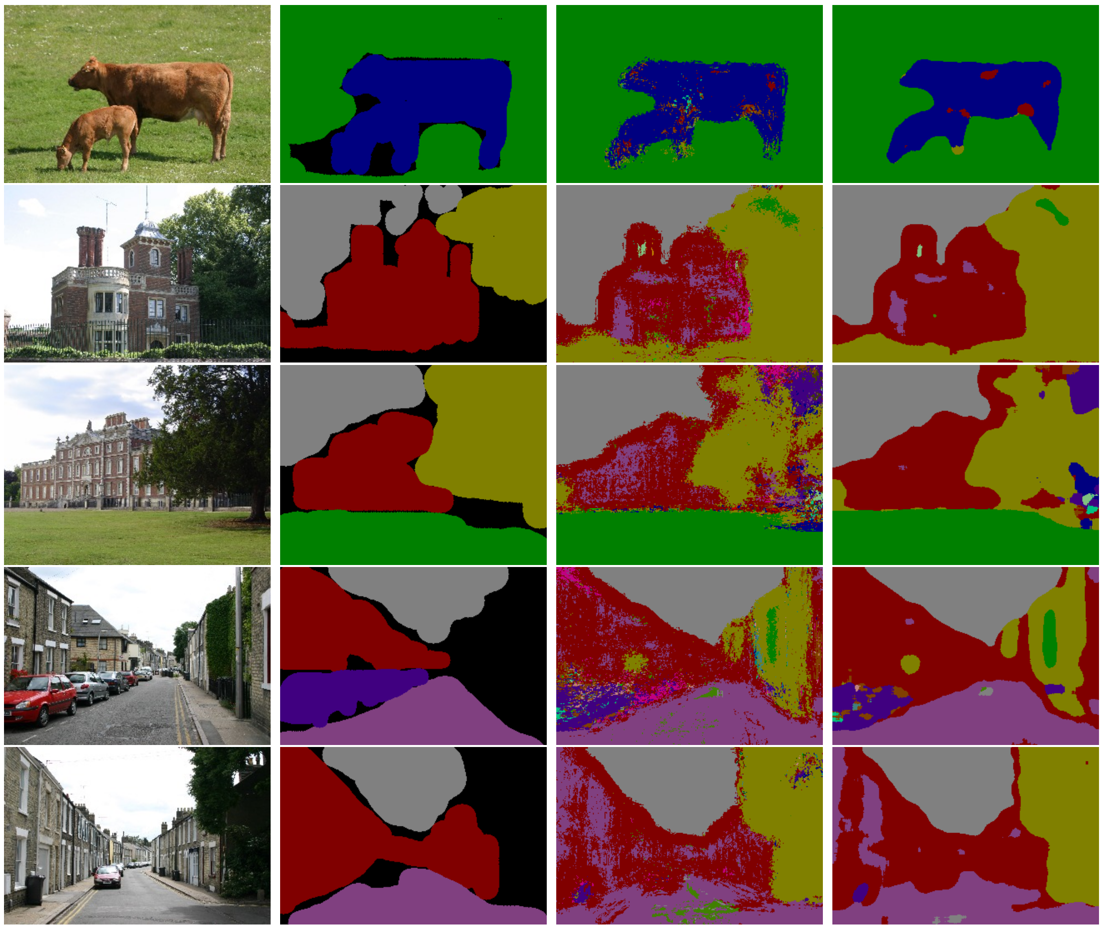

FIGURE 5.8: Columns denote from left to right: original image, ground truth image, output labeling of Random Forest with feature patches 5.2.2, output labeling of Structured Class-Labels 5.2.3.

Third column of Figure 5.8 shows some results of this approach. This approach can nicely segment large objects in the images, but it still produces a lot of noise in the output labelings.

In this experiment the number of random splits per split node was set to $K = \lceil \sqrt{dw^2} \rceil$, where $d = 3$ is the dimensionality of he original feature space and $w = 25$ is the dimensions of a square feature patch around each pixel, to reflect the increased feature space dimensionality.

[37] report Total Error of 50.3%. Our result is a bit better, because in [37] they trained 5 trees each of maximum depth 10.

5.2.3 Setting 3: Random Forest with Feature Patches and Structured Class-Labels

In this experiment we took the Setting 2 5.2.2 and added approach from [25] (description of the method can be found in 2.3.3) which helps to improve the output segmentations results by removing a lot of noise and making the labeling more consistent. Though the improve in terms of numbers is not that much significant (∼1%), but the improve in visual segmentation quality is evident. Because this method works only in the label space and does not consider features or original image, or even the probability distribution for each pixel, it is not able to correct wrong guesses, which can be the output of the classification process.

In this experiment the feature settings were the same as in 5.2.2. The width of a square label patch was chosen to be 7 pixels.

Figure 5.8 (forth column) shows some results of this approach. If compared to just Random Forest from the previous section which does not consider labels' consistency among neighboring pixels, we observe significant reduction of label-noise.

[25] report total error of 39.2%, which is better than we get, but this difference might be due to the fact that they used more complicated features (CIE raw channel intensities, first and second order derivatives as well as HOG-like features, computed on L-channel). The main observation here is that even with the simplest features this approach allows to improve the results.

5.2.4 Setting 4: Random Forest with Texton Features

Method proposed in [38] is used in [26] and together they compile the current state-of-the-art method for general purpose image segmentation. We implemented this method as described in the original article [38] with the main difference, that we used Random Forest instead of Boosting as the basic classifier. We augmented our implementation of Random Forests to be able to work with the new features (detailed description can be found in 2.2.2)

In this experiment we precomputed textons for both train and test images, so the measured running times do not reflect the duration of this process. As proposed by the authors we also used subsampling of the texton maps in order to decrease memory requirements. This does not effect the accuracy, because while computing Integral images we aggregate information of the whole patch that is being subsampled, so there is no loss of texton information, we just correspondingly decrease the spatial resolution of the output classifier. Due to the specifics of random nature of Random Forests, in each spit node we sampled 10 random rectangles for each of 400 textons in order to be sure not to miss important information. Obviously, this results in increased training time because of the increased search space. But the testing time still stays very low.

Figure 5.9 (third column) shows some example segmentations for this approach. One can notice, that the resulting segmentation is quite noisy and does not preserve object boundaries. So, this approach is good for quickly locating object in the scene, but it is not that faithful for determining exact object boundaries, so some additional post-processing step (at least a simple CRF as in the original paper) is absolutely necessary.

[38] report total error of ∼30%. But it is not clear from the article with which parameters they got this result. We downloaded the code the authors provide with this article, ran it and the result was 37.3%, which is almost exactly the same, as what we achieved.

5.2.5 Setting 5: Random Forest with Texton Features and Fully-Connected CRF (FC-CRF)

This approach was proposed in [26] and, same as Structured Class-Labels method, can be seen as a method of enhancing resulting labeling of images by introducing a fully-connected CRF over the image pixels and, which is most important, allowing to perform inference into such complicated graph in linear time. Unlike the Structured Class-Labels method, this approach considers both the feature-transformed image and the probability distributions of each pixel which is the output of the unary classifier. Basically, this means that this approach can be applied on top of any classification algorithm, which is able to produce probability distributions. In our experiments we used the code provided by the authors and applied it to the output of Random Forest with Texton Features from the previous section. As this approach does not change anything in the training process and is only applied while testing, the training time does not change at all. As a result we observe a significant boost (almost -10% for total error) in the overall classification accuracy.

Figure 5.9 shows some example labelings of this approach. One can notice, that the segmentation has very fine details in terms of object boundaries, unlike over-smoothed object boundaries for ordinary CRF approaches in which each pixel is connected only to a small number of neighboring pixels.

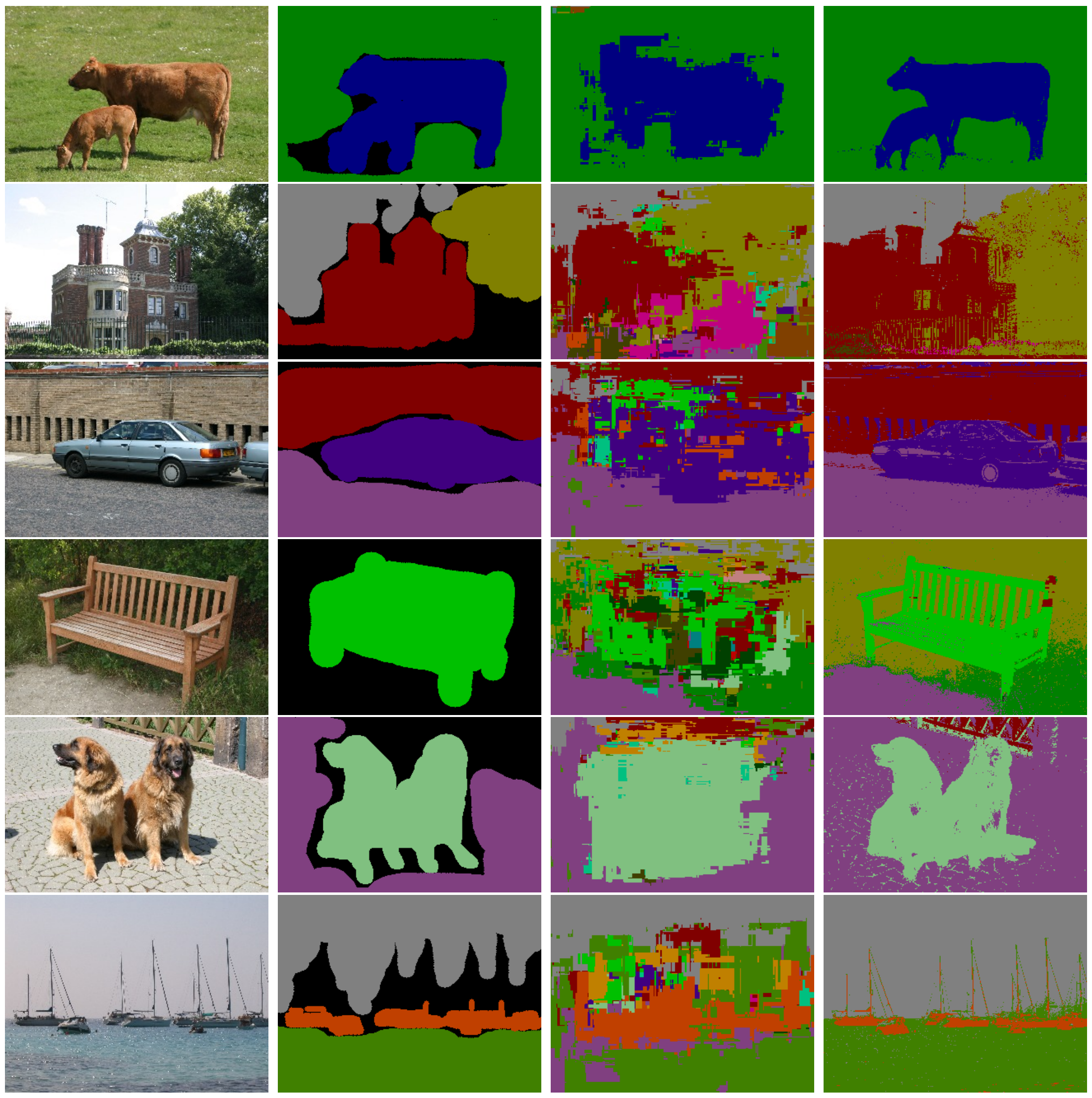

FIGURE 5.9: **Random Forests with Textons and FC-CRF**. Columns denote from left to right: original image, ground truth image, output labeling of Random Forest with Texton features, ouput labeling of fully-connected CRF method applied on top of the result of Random Forest with Texton features.

5.2.6 Discussion

The conducted experiments show that the more advanced classification approach always performs better in terms of accuracy. But on the other hand this comes at the cost of proportionally increasing the running time of the algorithm. For example, as it follows from Table 5.1 the method of Krähenbühl *et al.* [26] performs almost twice as better as the best accuracy results achieved by our implementation of Random Forest with Texton features and fully-connected CRF. But on the other hand the former method requires almost a day for training and testing of running on a very powerful machine, while for the latter it takes just a bit more than 10 minutes for training and less than 5 minutes for testing. And the main reason for better performance of Krähenbühl *et al.* [26] is that they use a much richer set of features. But we overcome this difference by using advanced features for road detection in the following experiments, while retaining low running time. We also choose our implementations, because they are based on Random Forests, which are better suitable for on-line learning than boosting, which is used in Krähenbühl *et al.* [26]

5.3 Experiments on the New Test Set

For notational convenience, from here on we use the word *old* when we refer to the training and testing datasets from [45], and *new* when we mean the introduced more challenging dataset. In all the following experiments we performed training on the old training set.

We performed a number of experiments on the dataset proposed in Chapter 4. First of all, we show that the new dataset is indeed more challenging not only from our personal point of view, but also for the state-of-the-art method for general purpose image segmentation of Krähenbühl *et al.* [26]. This shows that standard approaches which learn the class distribution from the training data can perform well on the data which shares the same distribution, but degrade in accuracy if the distribution changes in the testing data. Then we show that some of the standard approaches totally fail when the distribution changes. And also that features are very important to be chosen carefully for a particular application.

5.3.1 State-of-the-art Segmentation Algorithm

In order to show that the introduced data-set is indeed a more difficult challenge that the old one, we applied the current state-of-the-art segmentation algorithm [26] pipeline to both sets. The pipeline consists of two main steps: the initial segmentation is done using some unary classifier, then the fully-connected CRF algorithm as applied to it. The authors used their own implementation of TextonBoost [38] (the corresponding code can be found at the following location `http://graphics.stanford.edu/projects/densecrf/textonboost/`). We also used this code, and thanks to the fact, that the code is well-written and, though contains almost no comments, is easy to understand, we slightly changed the code to make it being able to work with our train and test datasets and, correspondingly, with our class-to-color mappings. After getting the unaries from TextonBoost classifier, we applied the code (2.4.1) for performing inference into fully-connected CRFs, which enhances the resulting segmentation results.

Test set	Unary error, %					FC-CRF error, %				
	Total	Average	Road	Background	Sky	Total	Average	Road	Background	Sky
Old	1.9	2.5	0.9	2.3	4.3	1.6	1.9	0.7	2.2	2.7
New	21.5	33	53.1	7.6	38.2	20.2	31.4	52.7	6.5	35

TABLE 5.2: Comparison of Krähenbühl *et al.* [26] semantic image segmentation algorithm on the old and the new test test. Training was performed on the old train set.

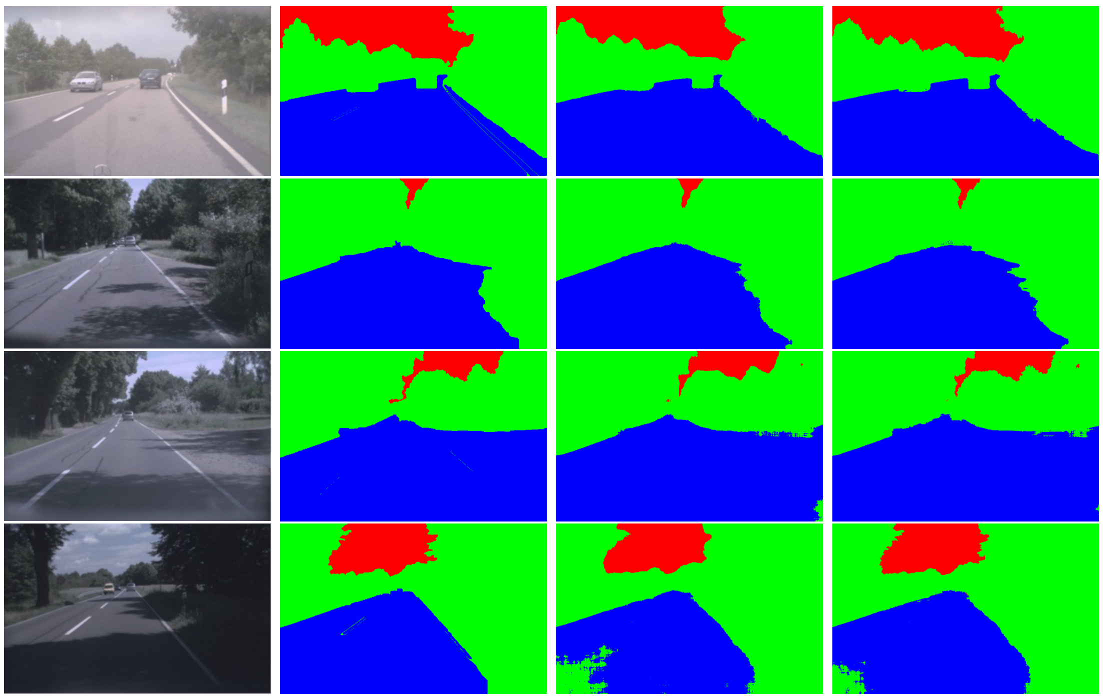

FIGURE 5.10: **State-of-the-art on the old test set.** Columns denote from left to right: original image, ground truth image, output of the unary classifier, enhanced segmentation after applying the FC-CRF algorithm.

Figure 5.10 shows several, so to say, "worst" examples for the setting when the training and testing was performed on the old dataset. Even these "worst" images prove the numbers given in Table 5.2. The segmentation results are almost ideal. Even shadows or a bit of dirt on the road does not create noticeable difficulties for the state-of-the-art classifier, even though this is a general purpose segmentation algorithm and is not tuned for dealing with road scenes. But this accuracy comes at the cost of running time which comprises almost a day on a very powerful machine. But all-in-all this experiment shows that the old test set is really not a problem for the state-of-the-art segmentation algorithms.

The second row of Table 5.2 shows the numerical results for the state-of-the-art when trained on the old and tested on the new dataset. Total and average errors prove that the new test set is around 10 times harder than the old one. Per class errors show that the main difficulty for the classifier is the “road” class: less than every second pixel belonging to this class gets labeled correctly. And this is quite expected, because the “road” in the new test set barely resemble the “road” samples which can be seen in the training data. And as the state-of-the-art method relies more on the appearance features it evidently has problems with correctly classifying “road” pixels. But, certainly, correct segmentation of the “road” is the prime aim for any classifier, so the new test set in fact introduces new challenges. “background” gets classified very well, but this can be explained by the fact that it is the most representative class in both train and test datasets so for any classifier it is always safer to predict “background”, because even such random choice has the highest probability of being correct just by chance compared to the other classes. It is interesting to notice that the sky also has quite a large test error, though its appearance shouldn’t be changing much when going from one dataset to another. So, putting “sky” into a separate class rather than into the “background” class, as it is done in other approaches (like [10]), makes the whole task of classification even more difficult.

In the following pages we show some particular examples of the segmentation results we get with the state-of-the-art trained on the old training set and tested on the new testing set. We used exactly the same parameters as those when we did the experiment on the old training and testing sets. The testing was carried out on the whole new test set, but just for the sake of increasing convenience we show good and fail cases for two parts of our new test set separately.

Figure 5.11 shows some results of the state-of-the-art segmentation algorithm applied to the “autumn” part of the new dataset. Despite the fact, that these images clearly differ very much from the training ones, the state-of-the-art manages to detect the road completely covered with yellow leaves, or the road which is not even covered with asphalt, or even some way (which cannot be even called a road) through forest. We think, that this successful cases are due the fact that the algorithm uses a reacher set of features (unlike in the original TextonBoost paper [38]), which also include location and HOG features, which must considerably help here.

On the contrary Figure 5.12 shows some particular examples from the “autumn” series (certainly, there are more fails, than shown here). While “sky” gets labeled more or less correctly in these images, but the algorithm totally fails in finding the road. But this is not a surprise, because the given visual conditions are totally different from the training ones.

The following Figure 5.13 shows some examples of relatively good segmentation results from the “winter” part. Surprisingly the state-of-the-art succeeds in finding the road fully covered with snow, which are for sure totally different from roads seen in the training set from the point of view of their appearance. Again, we think that this happens due to extended feature set, which includes not only appearance information. Figure 5.14 shows some failure cases of the state-of-the-art applied to the "winter" part of the new dataset. Same as with "autumn" failure cases, here "sky" also gets labeled somewhat correctly, but the road is not detected at all.

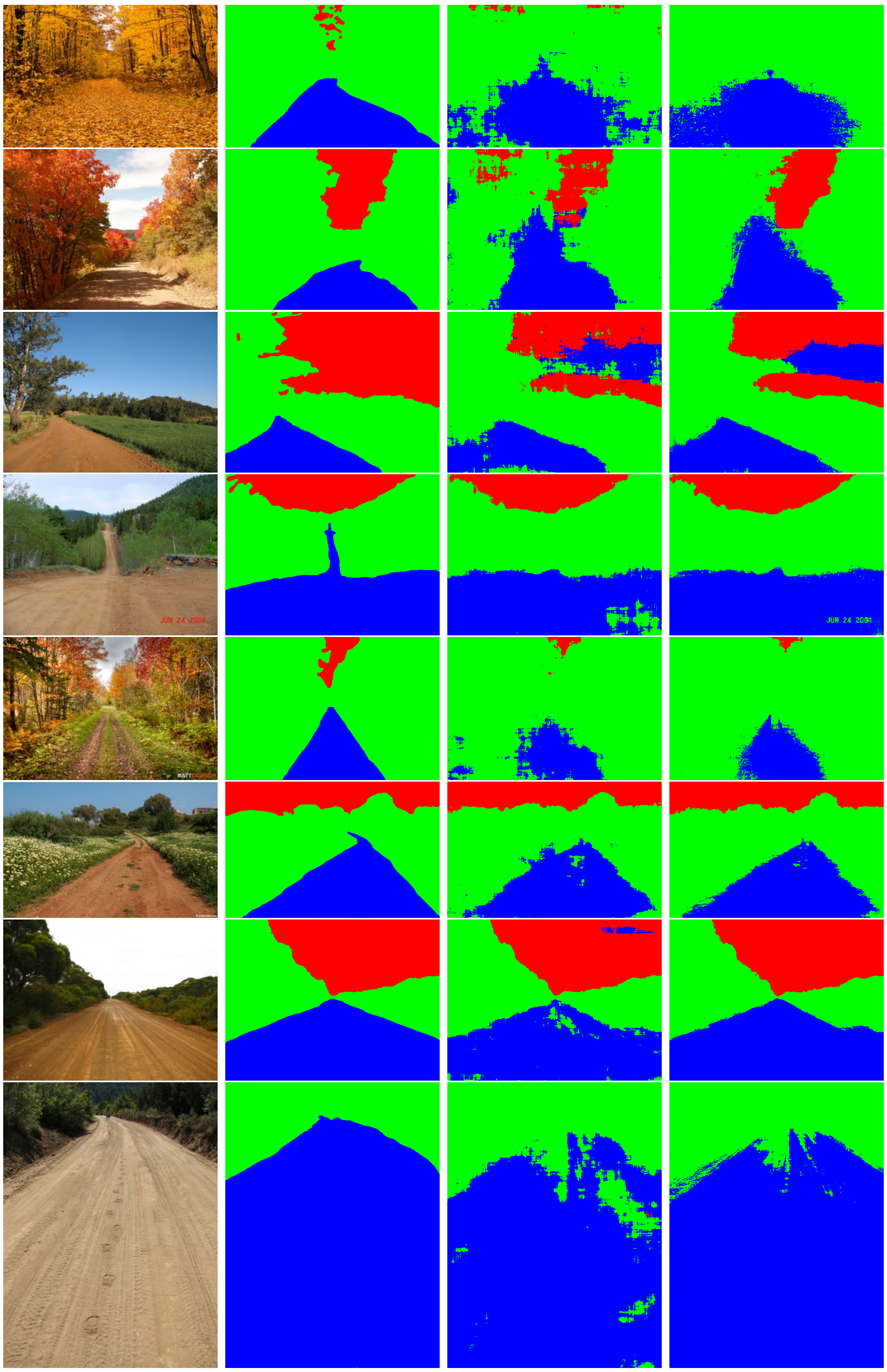

FIGURE 5.11: **State-of-the-art on the new test set: Relatively good cases of "autumn" part**. Columns denote from left to right: original image, ground truth image, output of the unary classifier, enhanced segmentation after applying the FC-CRF algorithm.

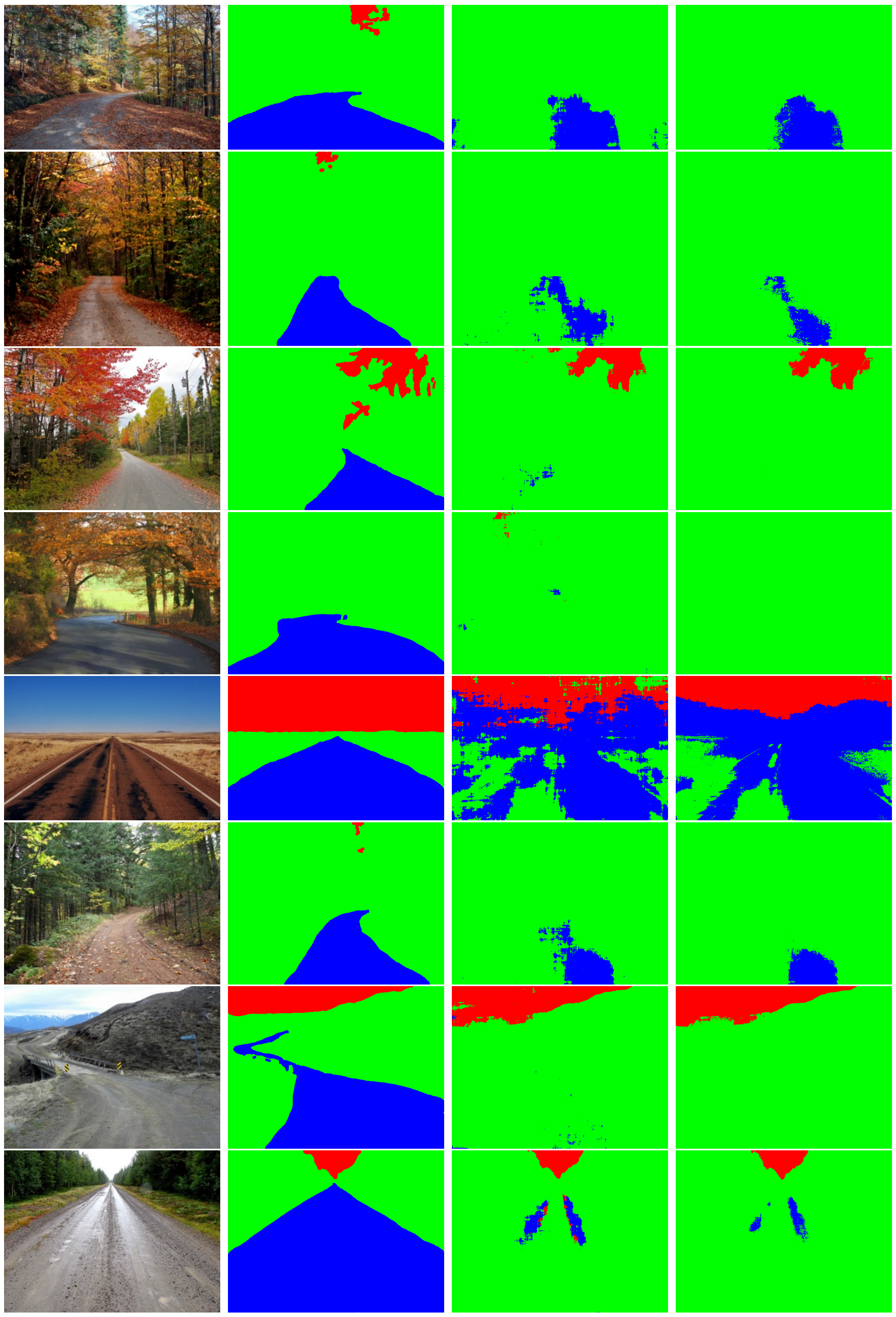

FIGURE 5.12: **State-of-the-art on the new test set: Fail cases on "autumn" part**. Columns denote from left to right: original image, ground truth image, output of the unary classifier, enhanced segmentation after applying the FC-CRF algorithm.

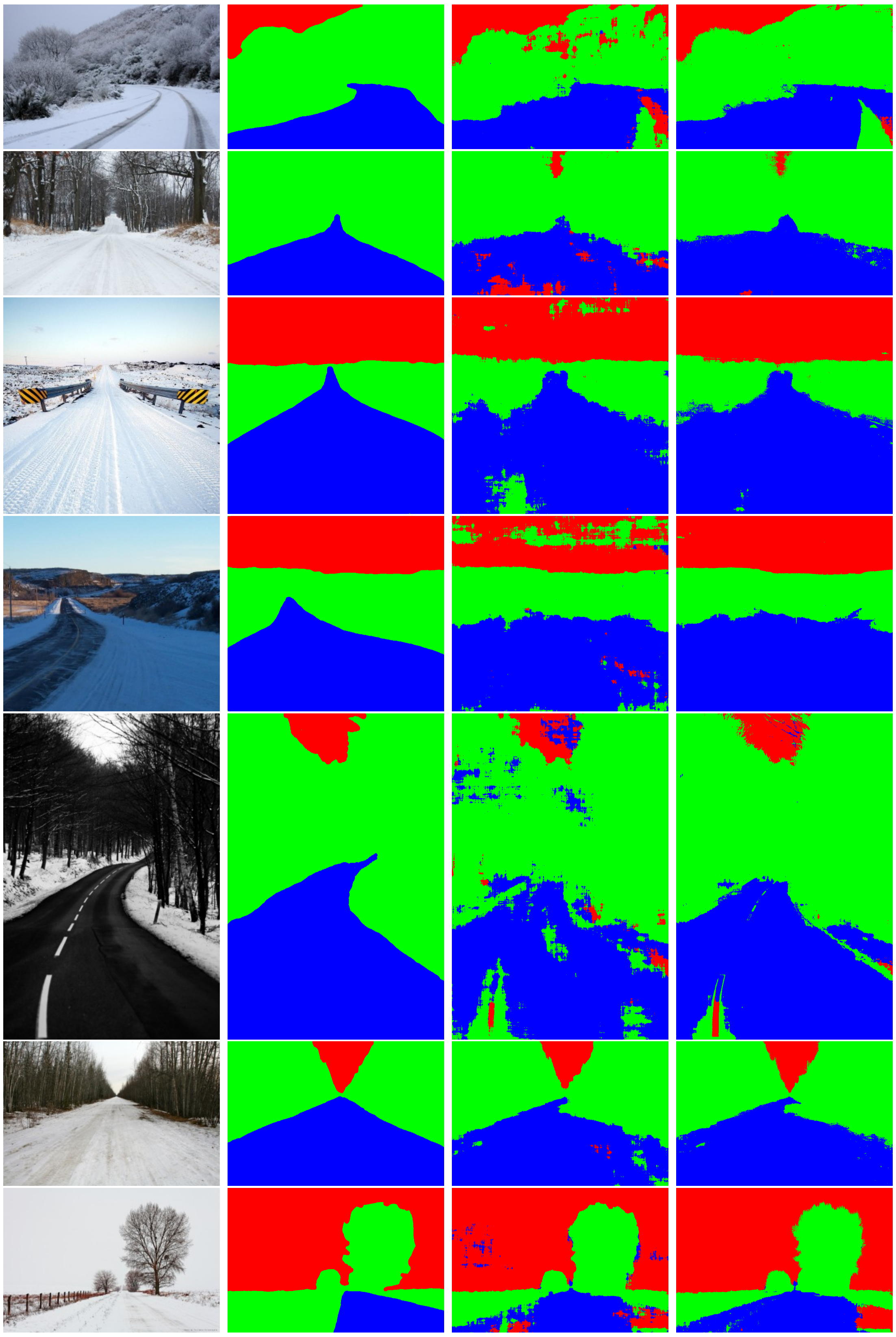

FIGURE 5.13: **State-of-the-art on the new test set: Relatively good cases on "winter" part**. Columns denote from left to right: original image, ground truth image, output of the unary classifier, enhanced segmentation after applying the FC-CRF algorithm.

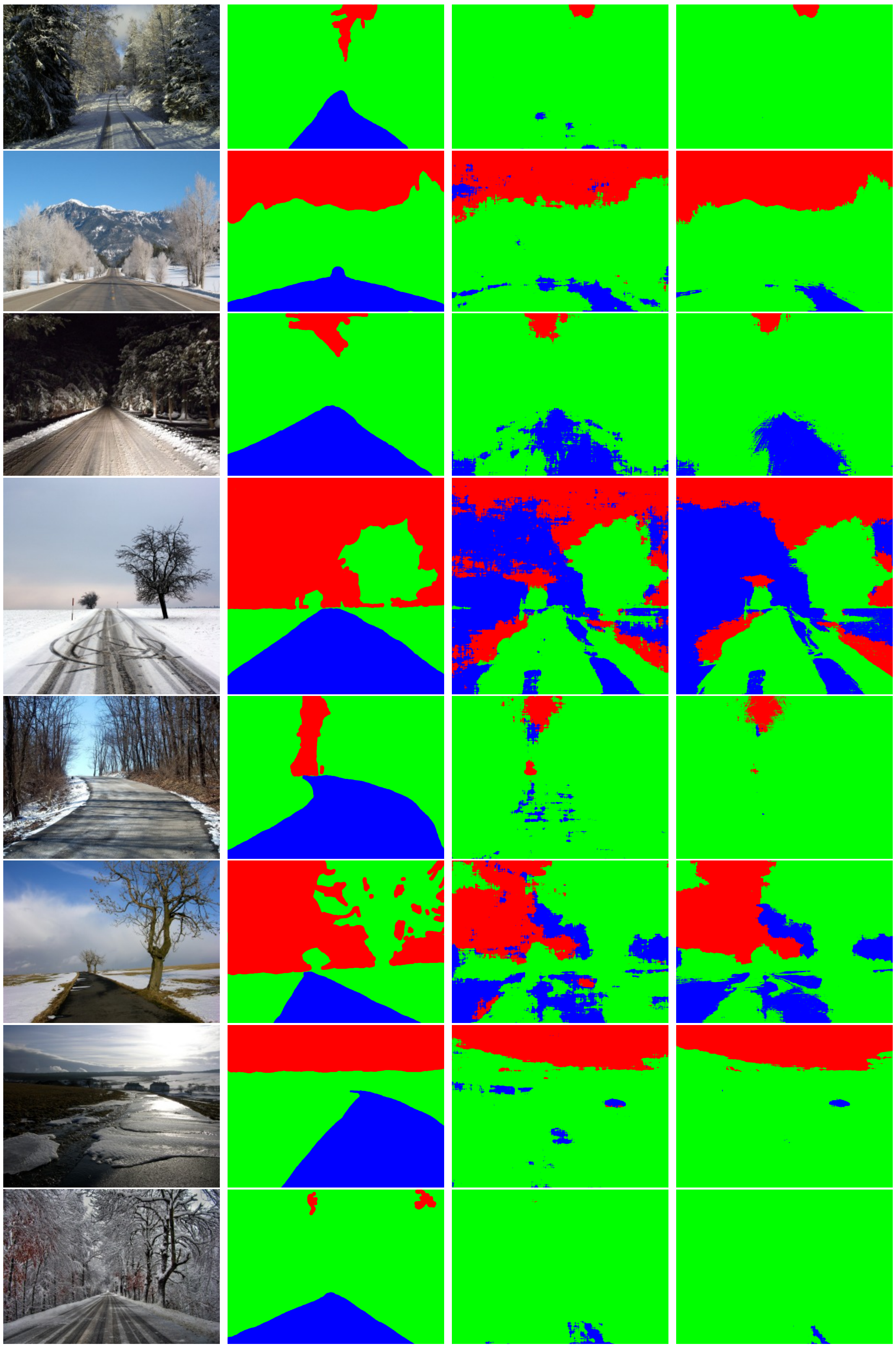

FIGURE 5.14: **State-of-the-art on the new test set: Fail cases on "winter" part**. Columns denote from left to right: original image, ground truth image, output of the unary classifier, enhanced segmentation after applying the FC-CRF algorithm.

Method	Unary error, %					FC-CRF error, %				
	Total	Average	Road	Background	Sky	Total	Average	Road	Background	Sky
Random Forest	35.9	61.5	90.1	6.9	87	34.7	63.7	95.2	**2.2**	93.8
RF/patches	30.6	49.5	91	6.8	50.6	30.7	50.7	92.1	5.8	54.1
RF+SCL	29.9	48.5	91.4	**6.2**	47.9	-	-	-	-	-
RF/Textons	69.2	61.3	96.7	72.7	**14.4**	69.7	61.4	97.4	73.3	**13.6**
Krähenbühl [26]	**21.5**	**33**	**53.1**	7.6	38.2	**20.2**	**31.4**	**52.7**	6.5	35

Table 5.3: Comparison of approaches described in Section 5.2. For a comparison results of Krähenbühl *et al.* [26] are also provided. Bold font highlights the best numbers.

5.3.2 Simple Features

We started our experiments on the new test set by first applying segmentation algorithms from part 5.2 using the same features and all the parameters for the corresponding approaches as there. The results can be seen in Table 5.3. There is not so much can be said about it. Obviously, the first three approaches cannot even slightly compete with the state-of-the-art, because they use simple CIELab values as features, and, as expected, appearance-based models cannot faithfully generalize over the visual conditions not present in the training set. Basically, the road test error numbers for this approach prove this argument, because it always stays extremely huge. “sky” gets labeled a bit better by Random Forest on feature patches and Structured Class-Labels approaches, but again all the three methods just assign “background” label when they don’t know what to do.

An interesting observation can be made regarding the approach based on Random Forest and Texton 2.2.2 features. This approach showed very good results on the MSRC dataset 5.1 and, in fact, performs extremely well (Figure 5.16) on the old testing set, showing total error numbers ∼6% for the unaries and ∼4% after applying the fully-connected CRF approach, but here it simply fails everything. Figure 5.15 shows some results for this approach applied to the new testing set. It is interesting that the algorithm performs quite good segmentation, but assigns totally wrong labels.

We think that this is due to the fact how Texton features work. Both Figures 5.15 and 5.16 show texton maps for each input image, where colors denote different textons and are the same for both figures. If we compare them, then texton maps of old test images show the expected uniform assignment of “road” and “sky” pixels, whereas texton maps for the new test images contain almost only noise. And also due to the fact that appearance of “road” changes dramatically when going from old train to new test data

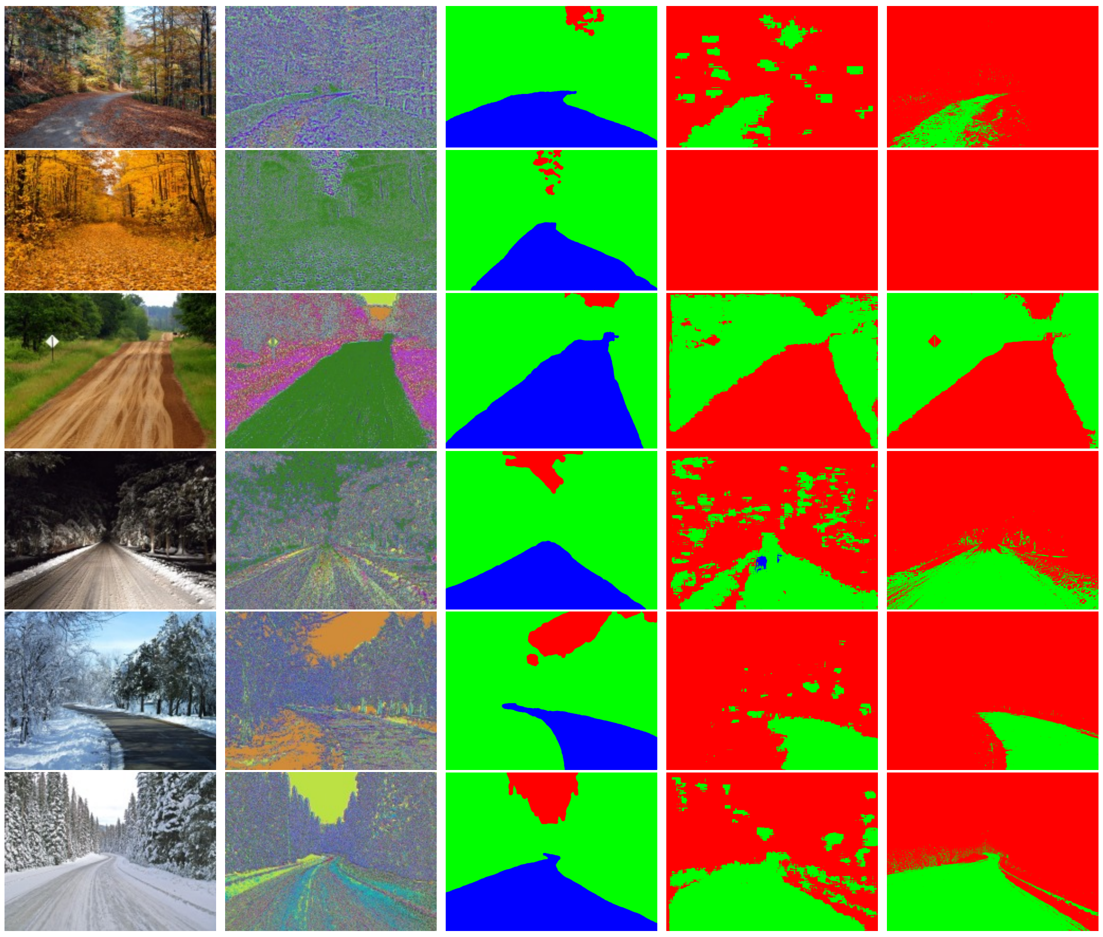

FIGURE 5.15: **Random Forest with Texton features and the new test set**. Though the algorithm segments out the "road" and "sky" classes quite well, it assigns wrong labels. Texton maps show particular problems of clusterization, because such assignments for road are totally different from the ones seen in the training data. Columns denote from left to right: original image, texton map for it, ground truth image, output of the unary classifier, enhanced segmentation after applying the FC-CRF algorithm. For texton maps colors denote different textons (cluster centers).

and, probably, whitening parameters and cluster centers computed on the old train set do not suit the test data, and pixels from the new test images are simply assigned to completely wrong cluster centers even after whitening. These textons were designed by the authors for the MSRC dataset, in which train and test sets share the same appearance and such clusterization works well for it, but does not work in our case.

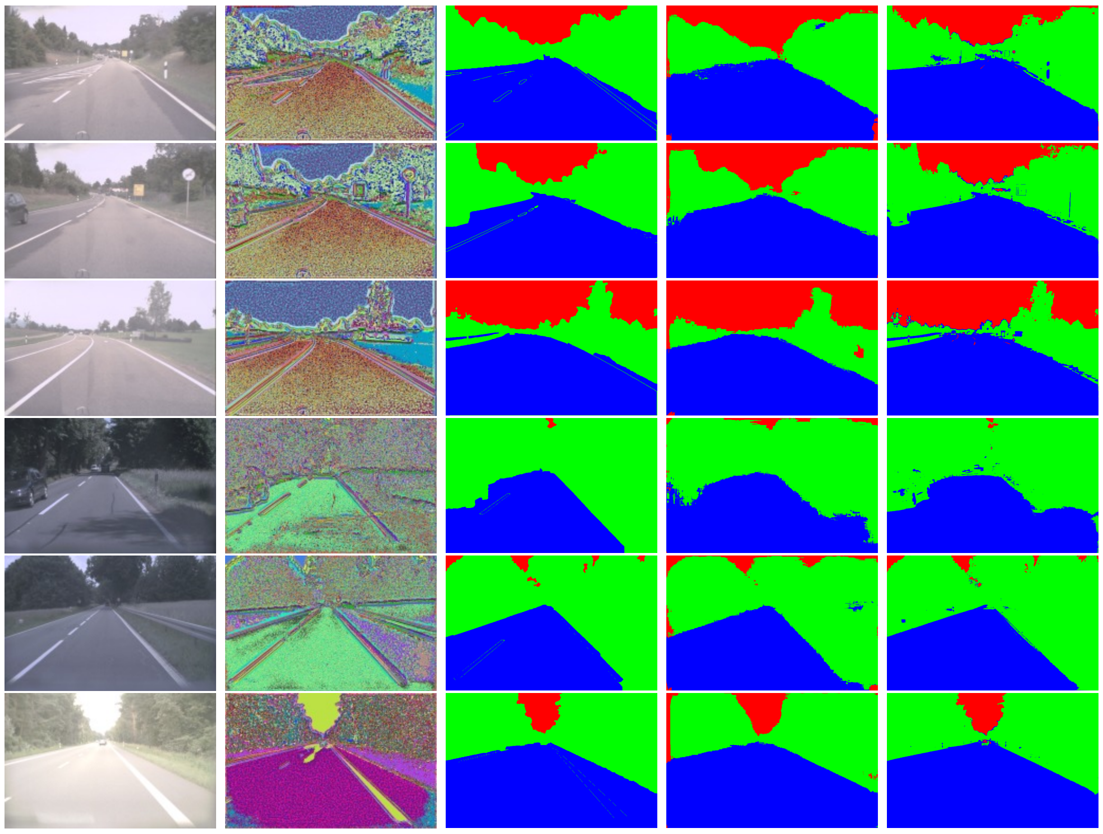

FIGURE 5.16: **Random Forest with Texton features and the new test set**. The segmentation results on the old testing set are almost perfect. Texton maps show almost uniform assignment. Columns denote from left to right: original image, texton map for it, ground truth image, output of the unary classifier, enhanced segmentation after applying the FC-CRF algorithm. For texton maps colors denote different textons (cluster centers).

5.3.3 Advanced Features for Road Detection

Then we conducted a number of experiments on the new testing set using the above described classifiers but this time with the advanced features for road detection from original paper by Christian Wojek and Bernt Schiele [45] (described in 2.2.3). Then numerical results are given in Table 5.4. As it follows from the table, even a simple ordinary Random Forest (with the help of fully-connected CRFs), but which uses advance features, can successfully compete with very sophisticated algorithms, and the original method of Wojek *et al.* [45]. This experiment shows that feature engineering is an essential part of the designing process of a good classifier.

Method	Error, %				
	Total	Average	Road	Background	Sky
RF	20.1	27.7	43.8	10.5	29.1
SCL	19.9	27.6	38.6	11.1	33.1
RF+FC-CRF	**18.8**	24.7	40.8	11.0	**22.2**
Krähenbühl *et al.* [26]	20.2	31.4	52.7	**6.5**	35
Wojek *et al.* [45]	19.2	**23.9**	**34.2**	13.2	24.5

TABLE 5.4: Comparison of different approaches based on Random Forest with advanced features for road detection, and Krähenbühl *et al.* [26] approach. Bold font highlights the best numbers.

5.3.4 Discussion

The results presented in Table 5.2 show that our proposed dataset is almost 10 times harder than the older one from [45]. This introduces new challenges into the segmentation community.

In Section 5.3.2 we showed that the Random Forest classifier with Texton features, which showed one of the best results on the MSRC dataset (Table 5.1), totally fails when the underlying class distribution changes with time or with datasets. This is a particular example of the fact that adaptation is an essential technique for handling such changes.

Section 5.3.3 shows the importance of careful feature selection for a particular task. Based on the obtained results in Table 5.4 we decided to take pure Random Forest together with the advanced features, because this setting shows accuracy comparable (and even better for some classes) with Krähenbühl *et al.* [26]. But Random Forest has extremely little training and testing time. And using pure Random Forest allows to perform adding of new samples in on-line learning much easier and faster.

5.4 On-line Learning

On-line learning is a method of incorporating arriving information at the time of testing. It is based on the fact that we put new samples, which have just been classified by the algorithm, together with the predicted label straight into the classifier and re-train it. It was shown in [33] that on-line learning converges to off-line one, so pure on-line method can be seen as just a speeding up technique. Instead we use, as we call it, batched learning, where we take a number of consequent images (in all experiments we fixed the size of a batch to 10 images) and use process them by the algorithm.

We also combine off-line learning together with on-line learning by first training a classifier on the old training set (off-line part) and then refine it by adding new samples at the testing time (on-line part). Correspondingly, we call the old training set, which is used for training in off-line manner, as *off-line set*, and the new testing set, which is used for on-line learning, as *on-line set*. And the classifiers, which was trained only on the off-line set or on both sets as *off-line* and, correspondingly, *on-line* classifier.

For all the following experiments we randomly permuted the on-line set and saved the permuted indexes, in order to imitate a random input to all the methods and to be sure that they all get the same input in order to be fully comparable.

5.4.1 Naive Approach

The naive approach (Section 3.1) is based on performing segmentation of a new batch of images, then selecting a number of pixels of each class and adding them into the classifier, which is later re-trained on the old and new new samples.

Figure 5.17 shows the behavior of test errors for each class, as well as total and average, depending on how many of the on-line images the algorithm has seen so far. In this experiment we carried out accuracy evaluation on the whole on-line dataset after every other lately processed batch of on-line files. The very first values correspond to error rates of an off-line classifier (*i.e.* the one, which was trained solely on the off-line data). As it follows from the chart even the most simple on-line learning approach helps to considerably decrease test error for the "road" class, as well as "sky". "background" error increases slightly, but the Average and Total error decrease in general.

In order to evaluate the real on-line accuracy of an on-line classifier we propose to conduct an experiment, which simulates the real on-line scenario of new data arrival and testing. This method implies computing error rates not on the whole on-line set, but exactly on the part of it, which has just arrived. It exactly corresponds to the

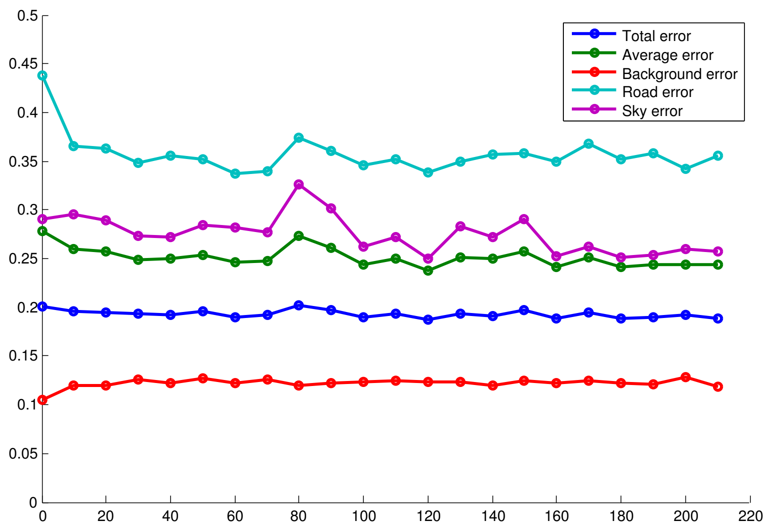

FIGURE 5.17: **Naive approach for on-line learning**. X-axis is the number of new images that has been seen up to the current moment, Y-axis is the error rates (averaged over 3 runs) computed on the whole new testing set after processing the corresponding number of new images.

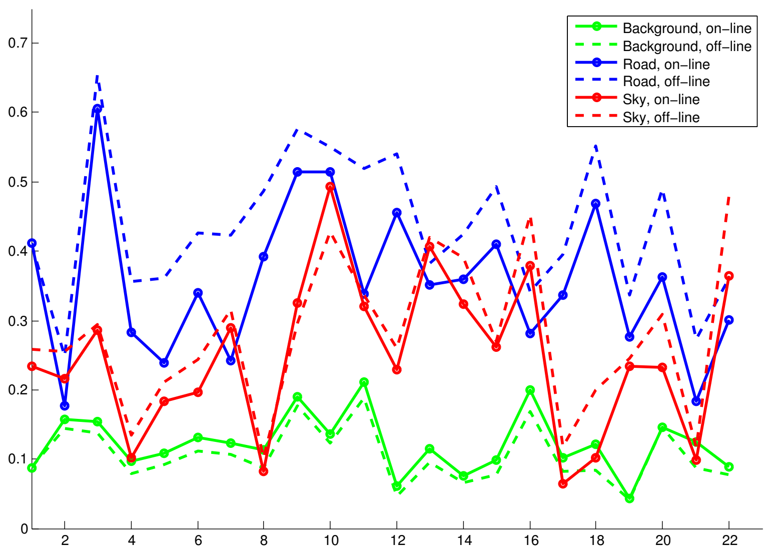

FIGURE 5.18: **Real on-line evaluation of the naive approach**. Solid lines denote on-line classifier and dashed lines its off-line counterpart.

real-world application, when a new image arrives and we want to be sure to carry out accurate segmentation particularly on this image, rather than improving segmentation of the images which are already “the past” regarding current moment. Figure 5.18 shows results for such evaluation. As it follows from the plot, on-line classifier indeed performs better, than its off-line counterpart, especially for the “road” class: the “road” on-line curve is always (and sometimes considerably) lower than the off-line one.

5.4.2 Adding Constraint in the Form Prior Labeling

An extension of the naive approach would be to use some prior information, which can be computed on the off-line set. [9] propose to compute class-histogram on the off-line data, then normalize it per-pixel and use it as a prior distribution to weigh the output probability distribution of the classifier at testing time, which should help to decrease the number of false positives being accepted.

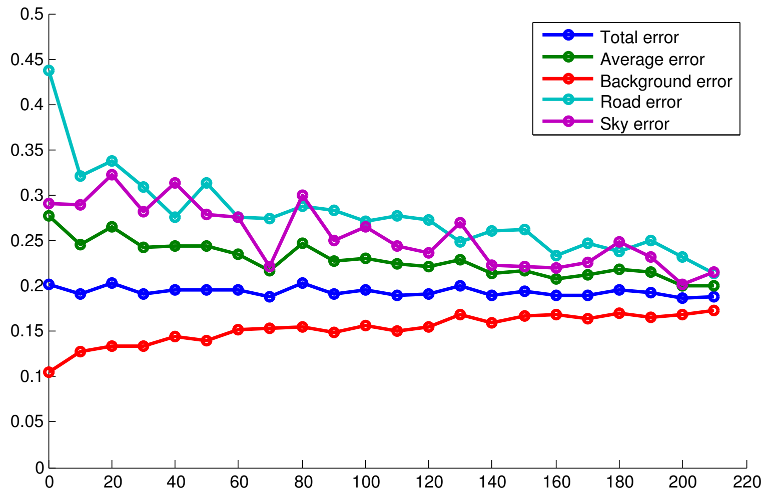

FIGURE 5.19: **On-line learning with the prior labeling constraint**. X-axis is the number of new images that has been seen up to the current moment, Y-axis is the error rates (averaged over 3 runs) computed on the whole new testing set after processing the corresponding number of new images.

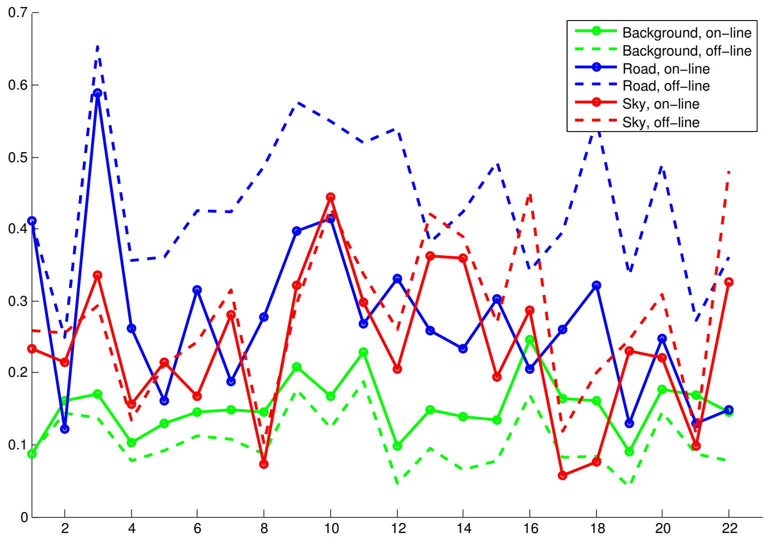

FIGURE 5.20: **Real on-line evaluation of the prior-constrained approach**. Solid lines denote on-line classifier and dashed lines its off-line counterpart.

Figure 5.19 shows error rates depending on how many of the on-line images the algorithm has seen so far. This approach considerably decreases "road" error making it almost 2 times less. But on the other hand, “background” error in this case considerably increases, although the overall Total and Average errors go down, as well as “sky” error. The reason for such behavior can be that this algorithm tries to make any segmentation look close to the prior labeling (Figure 3.1) with some deviation, but this deviation may not be enough for some particularly difficult cases, which we have a lot in our on-line dataset. For a real-life application the segmentation algorithm should not only correctly find the “road”, but also its boundaries which is the “background” class. Otherwise, it would be easy just to mark all pixels in the lower part of an image as “road”, resulting in low error rate for this class, but making such systems useless in practice.

As in the previous section, Figure 5.20 shows results for the real on-line evaluation of the approach. The accuracy of “road” classification improves great, but it comes at the cost of considerable increase in the “background” error.

5.4.3 Bayesian Model Update for Scene Segmentation under Structured Scene Prior

In Section 3.3 we proposed a new way of incorporating unlabeled data, based on Bayesian tracking formulations ([21]). But in contrast to previous work, we track an evolving model over time and use a scene segmentation prior in order to avoid model drift.

In this, experiment we used 5 particles each of which being a Random Forest classifier with advanced features for road detection, but, in general, any other classifier can be used as a particle. Bayesian formulation of the approach gives mathematical ensures that increasing the number of particles to infinity will guarantee that the tracking algorithm will not miss the target distribution. But increasing the number of particles makes the whole system very computationally demanding.

Figure 5.21 shows error rates in dependence on how many of the on-line images the algorithm has seen so far. As it follows from the plot, the tracking converges quite quickly and stays at the same level almost without any variation. Only “sky” error is constantly going down during the whole observation period.

Figure 5.22 shows results for the real on-line evaluation of the approach. It is remarkable, that the error rate of the “road” class stays always considerably below the off-line reference. Unlike for the naive on-line approach (Figure 5.18) the error of the “sky” class stays always below the off-line reference. The “background” curves almost coincide for the on-line and off-line approaches. And this totally corresponds to the original designing aim: to improve the detection of the road without sacrificing other classes.

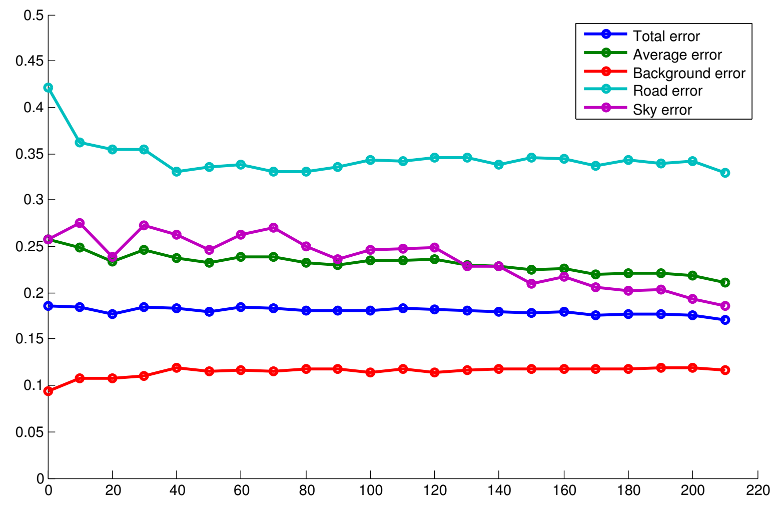

FIGURE 5.21: **Bayesian Model Update with Structured Scene Prior**. X-axis is the number of new images that has been seen up to the current moment, Y-axis is the error rates (averaged over 3 runs) computed on the whole new testing set after processing the corresponding number of new images.

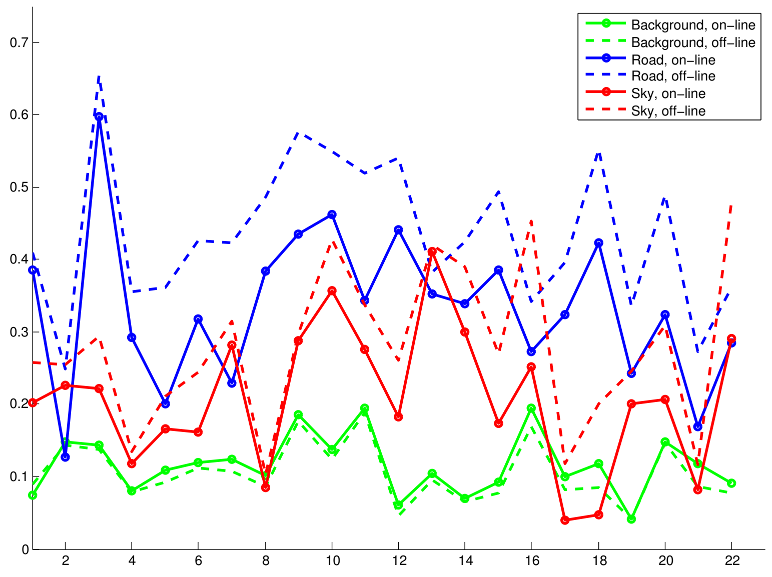

FIGURE 5.22: **Real on-line evaluation of the Bayesian Model Update with Structured Scene Prior approach**. Solid lines denote on-line classifier and dashed lines its off-line counterpart.

Figure 5.23 presents some examples of image labelings' evolution in the process of adaptation. The labeling gets more consistent with time: segmentation "holes" in the "road" are getting smaller or eliminated at all, while the shape of the "road" is getting closer to the ground truth image. And this happens without using ground truth images in the process of adaptation, but only by tracking the evolving class distribution which allows to adapt to new severe conditions on-the-fly.

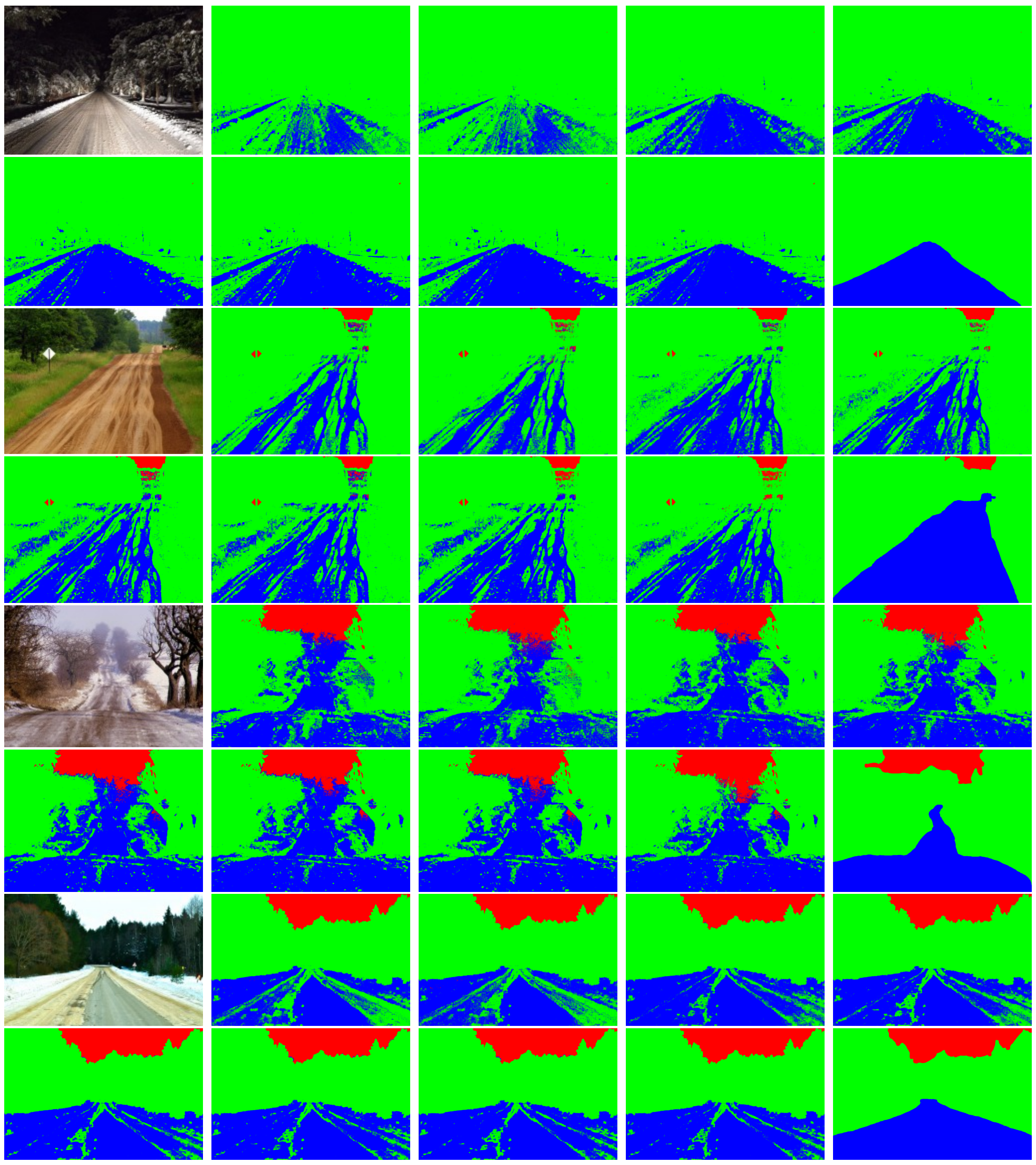

FIGURE 5.23: **An example of image labeling evolution in on-line learning**. First image presents the original image from the new dataset. Next four images show the segmentation after the first four iterations, and next four - after the last fout iterations of our proposed Bayesian Model Update algorithm. The labeling becomes better and gets closer to the ground truth (the last image) with time.

5.4.4 Discussion

Our experiments with on-line learning showed that adaptation to new conditions is a very helpful tool, which allows to improve stability in situations when the underlying class distribution changes with time. Table 5.5 summarizes the error rates for the converged on-line methods and an off-line one, which was just a Random Forest with advanced features for road detection trained only on the off-line set.

Method	Error, %				
	Total	Average	Road	Background	Sky
Naive	18.8	24.4	35.6	11.9	25.7
Prior-constrained	18.8	**20.1**	**21.3**	17.4	21.5
Bayesian	**17.1**	21.0	32.9	11.7	**18.5**
Off-line (RF)	20.1	27.7	43.8	**10.5**	29.1

Table 5.5: Comparison of different approaches of on-line methods evaluated on the whole testing set after converging. For comparison results for the employed off-line method is provided. Bold font highlights the best numbers.

The numbers show, that even a naive approach brings more than 1% decrease in total error. For comparison: getting the same effect requires application of the fully-connected CRF method (Table 5.4). Prior-constrained on-line method results also in around 1% decrease of total error, but it just sacrifices the error rate of "background" in favor of "road", which is very very debatable whether it is useful in practice.

The proposed Bayesian Model Update method shows the best improve -3% for total error, and results in more than 10% decrease of "road" and "sky" errors, while increasing the "background" error in just 1%. Considering the difficultiness of the on-line testing set, we consider the obtained numbers to be a considerable improvement over other approaches.

Chapter 6

Conclusions and Future Work

Summary: Image segmentation is a very old field of Computer Vision and has been studied for many years already. But despite of all the achieved progress, there are still some particular fields which require further researching. One such fields can be Image segmentation in changing visual conditions. We introduced a new road scenes dataset in this work, which shows that in situations when the testing sets exhibits high extent of visual appearance variation, which considerably differs from the conditions contained in the training set, standard approaches, which rely on the assumption of the constant class distribution over sets, show considerable degrade in accuracy or simply fail. For example, the evaluation of the current state-of-the-art method for general purpose scene segmentation by Krähenbühl *et al.* [26] on the proposed dataset and the old dataset from [45] shows at least ten times worse accuracy on the proposed set. And the method based on Random Forest with Texton features from [38] fails on the proposed dataset showing totally unacceptable results. But using advanced features for road detection allows even a simple Random Forest classifier to show good accuracy on the new testing set.

We showed in this work that on-line learning can be used as a powerful tool for performing adaptation to changing visual conditions giving the possibility for tracking the correspondingly changing underlying class distribution. Even the naive approach, when new samples are added just based on their confidence scores, already shows improvement of more than 1% in comparison with the off-line classifier, the one which didn't use test set for refining its predicting abilities. We experimented with the Prior-constrained method from [9], which considerably decreases error for one class ("road"), but at the same time increases error for other (like "background"). As a result the total error improve is not better than for the naive on-line approach.

To get better results for on-line learning based approaches and get better adaptation to changing conditions, we proposed Bayesian Model Update under Structured Scene Prior for tracking the evolving class distribution. This method allowed to get a considerable improvement of 3% for the total error in comparison with the off-line classifier, and more than 10% improvement for the error rates of “road” and “sky”.

Future work: Based on our experiments several directions for the future work can be outlined. First direction is further improvement of the new dataset. Current dataset exhibits considerable amount of visual variations, but its main disadvantage is that the images it contains are not captured by a car-mounted camera, and therefore show no temporal consistency between frames. Though such conditions suit the purposes of evaluation and comparisons of different methods (same as conventional MSRC or VOC datasets), but it will be interesting to have a dataset captured in real-life conditions and which will have temporal coherency between adjacent frames. This will also allow to ensure following application of adaptive algorithms into real-world and industry, because we think that it is exactly automobile industry that will benefit most of all from application of such systems.

Additional research is necessary for improving Bayesian Model Update approach. First of all, batched learning should be substituted by a real on-line learning with should considerably improve the overall running time of the system. This will also allow to increase the number of particles, which represent the distribution over models being tracked in our proposed Bayesian Model Update approach, which in turn can ensure that the tracking algorithm will not miss the target distribution due to lack of particles' representation. Another important concern is the problem of numerics, because computing likelihoods at the weights update stage of the Bayesian Model Update algorithm requires multiplication of a lot of probabilities which may result in underflow errors.

We used only a pure Random Forest as a particle in our proposed Bayesian Model Update approach, but this is a general framework which allows to incorporate any other classifier. It can be Boosting, Support Vector Machine, or anything else. If the number of particles is set to a sufficiently large number, all this classifiers can be made work together in a unified framework, which will ensure accurate tracking of the evolving target distribution.

Bibliography

[1] The cambridge-driving labeled video database. http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/.

[2] Fastest fourier transform in the west (fftw). http://www.fftw.org/.

[3] Intel threading building blocks. http://threadingbuildingblocks.org/.

[4] Microsoft research cambridge database. http://research.microsoft.com/en-us/projects/objectclassrecognition/.

[5] Usps dataset. http://www.cs.nyu.edu/~roweis/data.html.

[6] Visual object classes challenge 2012. http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2012/.

[7] Andrew Adams, Jongmin Baek, and Myers Abraham Davis. Fast high-dimensional filtering using the permutohedral lattice. In *EUROGRAPHICS*, 2010.

[8] Yaniv Alon, Andras Ferencz, and Amnon Shashua. Off-road path following using region classification and geometric projection constraints. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.

[9] José Manuel Álvarez, Theo Gevers, Yann LeCun, and Antonio M. López:. Road scene segmentation from a single image. In *European Conference on Computer Vision (ECCV)*, 2012.

[10] José Manuel Álvarez and Antonio M. López:. Road detection based on illuminant invariance. In *IEEE Trans. Intelligent Transportation Systems (ITS)*, 2010.

[11] Leo Breiman. Random forests. 2001.

[12] Gabriel J. Brostow, Jamie Shotton, Julien Fauqueur, and Roberto Cipolla. Segmentation and recognition using structure from motion point clouds. In *European Conference on Computer Vision (ECCV)*, 2008.

[13] Frank Dellaert, Wolfram Burgard, Dieter Fox, and Sebastian Thrun. Using the condensation algorithm for robust, vision-based mobile robot localization. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 1999.

[14] Charles Elkan. Using the triangle inequality to accelerate k-means. In *International Conference on Machine Learning (ICML)*, 2003.

[15] Yoav Freund. Boosting a weak learning algorithm by majority. In *Information and Computation*, 1995.

[16] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning (ICML)*, 1996.

[17] Brian Fulkerson, Andrea Vedaldi, and Stefano Soatto. Class segmentation and object localization with superpixel neighborhoods. In *International Conference on Computer Vision (ICCV)*, 2009.

[18] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. 2006.

[19] Stephen Gould, Jim Rodgers, David Cohen, Gal Elidan, and Daphne Koller. Multi-class segmentation with relative location prior. In *International Journal of Computer Vision (IJCV)*, 2008.

[20] Yacov Hel-Or and Hagit Hel-Or. Real-time pattern matching using projection kernels. In *Pattern Analysis and Machine Intelligence (PAMI)*, 2005.

[21] Michael Isard and Andrew Blake. Condensation - conditional density propagation for visual tracking. In *International Journal of Computer Vision (IJCV)*, 1998.

[22] David G. Jones and Jitendra Malik. A computational framework for determining stereo correspondence from a set of linear spatial filters. In *European Conference on Computer Vision (ECCV)*, 1992.

[23] George David Forney Jr. The viterbi algorithm: A personal history. 2005.

[24] Béla Julesz. Textons, the elements of texture perception, and their interactions. In *Nature*, 1981.

[25] Peter Kontschieder, Samuel Rota Bulò, Horst Bischof, and Marcello Pelillo. Structured class-labels in random forests for semantic image labelling. In *International Conference on Computer Vision (ICCV)*, 2011.

[26] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

[27] Brian Kulis, Kate Saenko, and Trevor Darrell. What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[28] Lubor Ladický, Chris Russell, and Pushmeet Kohli. Associative hierarchical crfs for object class image segmentation. In *International Conference on Computer Vision (ICCV)*, 2009.

[29] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning (ICML)*, 2001.

[30] Jitendra Malik, Serge Belongie, Thomas Leung, and Jianbo Shi. Contour and texture analysis for image segmentation. In *International Journal of Computer Vision (IJCV)*, 2001.

[31] Nikunj C. Osa and Stuart Russell. Online bagging and boosting. In *Artificial Intelligence and Statistics*, 2001.

[32] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *European Conference on Computer Vision (ECCV)*, 2010.

[33] Amir Saffari, Christian Leistner, Jakob Santner, Martin Godec, and Horst Bischof. On-line random forests. In *International Conference on Computer Vision (ICCV)*.

[34] Robert E. Schapire. The strength of weak learnability. In *Machine Learning*, 1990.

[35] Toby Sharp. Implementing decision trees and forests on a gpu. In *European Conference on Computer Vision (ECCV)*, 2008.

[36] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-time human pose recognition in parts from single depth images. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[37] Jamie Shotton, Matthiew Johnson, and Roberto Cipolla. Semantic texton forests for image categorization and segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[38] Jamie Shotton, John Winn, Carsten Rother, and Antonio Criminisi. Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. In *International Journal of Computer Vision (IJCV)*, 2009.

[39] Manik Varma and Andrew Zisserman. Classifying images of materials: Achieving viewpoint and illumination independence. In *European Conference on Computer Vision (ECCV)*, 2002.

[40] Manik Varma and Andrew Zisserman. A statistical approach to texture classication from single images. In *International Journal of Computer Vision (IJCV)*, 2005.

[41] Andrea Vedaldi and Brian Fulkerson. VLFeat: An open and portable library of computer vision algorithms. `http://www.vlfeat.org/`, 2008.

[42] Jackob J. Verbeek and Bill Triggs. Scene segmentation with crfs learned from partially labeled images. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

[43] Paul Viola and Michael J. Jones. Rapid object detection using a boosted cascade of simple features. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001.

[44] John Winn, Antonio Criminisi, and Thomas Minka. Object categorization by learned universal visual dictionary. In *International Conference on Computer Vision (ICCV)*, 2005.

[45] Christian Wojek and Bernt Schiele. A dynamic conditional random field model for joint labeling of object and scene classes. In *European Conference on Computer Vision (ECCV)*, 2008.

Appendix A

Detailed Random Forest Algorithm

Here is the full algorithm for building Random Forests. **TrainRandomForest** (Algorithm 4) function gets the whole bunch of training samples $(x_i, y_i)_{i=1}^{N}$ and builds M trees each of depth D. The main problem here is that the whole sample should be allocated in the memory at once which may require sometimes considerable amount of memory. But on the other hand each tree can be build independently in a separate thread allowing for easy parallelization.

BuildTree (Algorithm 6) is a function which builds each Random tree independently in a recursive manner. This algorithm can be altered easily to better reflect the specific needs of the particular application of the Random Forest. This is a general algorithm.

GetDistribution (Algorithm 5) is a function which takes a particular sample and traverses it through a tree until terminating in a leaf node, then it returns the distribution stored in the leaf node. Again, this is a general algorithm which can be altered to store anything apart from distribution, *e.g.* label patches or just a single label.

```
Data: a set of training points $(x_i, y_i)_{i=1}^{N}$, maximum depth $D$, number of trees $M$,
      number of random tests per node $K$,
Result: final classifier $\mathcal{F} = \{f_1, f_2, \ldots, f_M\}$
for $m \leftarrow 1$ to $M$ do
    1. compute per class weights $w$
    2. Subsample the set of training points to $S = (x_i, y_i)_{i=1}^{N'}$, so that $N' < N$
    3. $f_m \leftarrow$ BuildTree$(S, D, K, w)$
end
```

Algorithm 4: TrainRandomForest function

```
Data: test sample $x$
Result: distribution $p(c|x), c \in \mathcal{Y}$
while $node.is\_leaf \neq true$ do
    perform feature test $g(x)$;
    if $g(x) < \theta$ then proceed to the left subtree;
    else proceed to the right subtree;
end
```

Algorithm 5: GetDistribution function

```
Data: a set of training points $S = (x_i, y_i)_{i=1}^{N'}$, current depth level $level$, number of
      random tests per node $K$, per class weights $w$
Result: random tree $f(x)$
$distribution \leftarrow$ ComputeDistribution$(S, w)$;
$node\_impurity \leftarrow E(S, w)$;
if $node\_impurity < threshold$ or $|S| < minN$ or $level = 0$ then
    $node.distribution \leftarrow distribution$;
    terminate;
end
$level \leftarrow level - 1$;
$score' \leftarrow +\mathrm{inf}$;
for $k \leftarrow 1$ to $K$ do
    generate random parameters for the split;
    choose random feature $d$;
    get responses as $S' \leftarrow \{x_d | (x, y) \in S\}$;
    find $a = \min S'$ and $b = \max S'$ values;
    randomly uniformly sample $c$ from $[a, b]$;
    separate $S$ into $L = \{(x, y) | x_d \leq c, (x, y) \in S\}$ and $R = \{(x, y) | x_d > c, (x, y) \in S\}$;
    compute score as $score \leftarrow \frac{|R|}{|S|} E(R, w) + \frac{|L|}{|S|} E(L, w)$;
    if $score < score'$ then
        $score' \leftarrow score$;
        save the splitting parameters into node;
        remember best split as $L' \leftarrow L$ and $R' \leftarrow R$;
    end
end
if $|L'| > 0$ then
    BuildTree$(L', level, K, w)$;
end
if $|R'| > 0$ then
    BuildTree$(R', level, K, w)$;
end
```

Algorithm 6: BuildTree function