Samuel Schulter

Loss Minimization for Random Forests in Computer Vision

DOCTORAL THESIS

to achieve the university degree of
Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Prof. Dr. Horst Bischof
Institute for Computer Graphics and Vision

Prof. Dr. Fred A. Hamprecht
Interdisciplinary Center for Scientific Computing (IWR),
Heidelberg Collaboratory for Image Processing (HCI)

Graz, Austria, July 2015

To Elena

Machines take me by surprise with great frequency.

Alan Turing (1912 - 1954)

Abstract

Random forests are a well known machine learning technique that finds many practical applications in various fields of computer science. They show good generalization capabilities and low computational costs during both training and inference. Moreover, random forests are very flexible and can be applied to many tasks, including classification, regression, or density estimation. This machine learning model consists of an ensemble of binary trees, where each of them can make predictions individually. The final output is given as the average over the ensemble, though. In contrast to other learning algorithms like support vector machines, boosting, or neural networks, however, random forests optimize the training objective only locally. The objective is minimized for each tree individually rather than for the ensemble of trees, which ultimately makes the predictions. While this aspect allows for exploiting parallel computing architectures and, therefore, fast training, there is no control of the training procedure via a common loss function.

In this thesis, we address this problem and present a novel training scheme for random forests that explicitly minimizes a global loss function over the full ensemble. While a simple solution is to employ gradient boosting as a meta-learning algorithm, the attractive benefit of a fast training procedure would be lost. Even though we are inspired by gradient boosting, our approach of loss minimization is an inherent part of the tree growing process, which still allows for parallel training of the trees. Our loss minimization approach can be formulated as gradient descent in function space. Each gradient step grows the trees by one level of depth by finding splitting functions for all nodes in that level, which can still be done in parallel. Learning these splitting functions is influenced by the gradients that are computed for the given loss function. In this way, our formulation enables a proper minimization of any differentiable loss function for random forests. We present a formulation for classification and regression tasks and also provide some empirical analysis of the model on a set of basic machine learning benchmarks. We can observe better prediction accuracy compared to standard random forests while still having the advantage of a fast training time.

Furthermore, we present several computer vision applications where our novel formulation for training random forests could be applied successfully. We use our algorithm for two different object detection approaches, human head pose estimation from depth data, as well as single image super-resolution. These applications include both classification and regression tasks. Our results indicate that optimizing a common loss function over the full ensemble of trees instead of training them independently pays off. Our random forest formulation can be readily replaced with standard random forests while, at the same time, improving the results for both regression and classification tasks.

Kurzfassung

Random Forests sind eine weitbekannte Technik des Maschinellen Lernens und finden viele praktische Anwendungen in unterschiedlichen Disziplinen der Computerwissenschaften. Sie zeigen gute Generalisierungseigenschaften und geringen Rechenaufwand sowohl während des Trainings als auch während des Testens. Darüber hinaus sind Random Forests sehr flexibel und können für viele Aufgaben, einschließlich Klassifikation, Regression, oder auch Dichteschätzung, verwendet werden. Dieses Lernmodell besteht aus einer Menge von binären Bäumen, wobei jeder von ihnen einzeln ausgewertet werden kann um Vorhersagen zu treffen. Die finale Vorhersage wird jedoch als Mittelwert aus allen Bäumen errechnet. Im Gegensatz zu anderen Lernalgorithmen wie Support Vector Machines, Boosting oder neuronalen Netzen, optimieren Random Forests die Zielfunktion nur lokal, da jeder Baum unabhängig vom Rest und nicht gemeinsam trainiert wird. Während diese Eigenschaft es ermöglicht parallele Rechenarchitekturen auszunutzen, verhindert sie gleichzeitig die genaue Kontrolle über eine gemeinsame Zielfunktion und somit auch über das Training von Random Forests.

In dieser Arbeit beschäftigen wir uns mit diesem Problem und präsentieren einen neuartigen Trainingsalgorithmus für Random Forests, welcher explizit eine globale Zielfunktion über die gesamte Menge von Bäumen optimiert. Die triviale Lösung wäre es Gradient Boosting als Meta-Lernalgorithmus zu verwenden, wodurch aber der Vorteil des effizienten Trainings, also die Parallelisierung der Bäume, verloren gehen würde. Obwohl unser Ansatz auch von Gradient Boosting inspiriert ist, wird die Optimierung der gemeinsamen Zielfunktion direkt in das Wachsen der Bäume integriert, was ein paralleles Training von Random Forests wieder ermöglicht. Ähnlich zu Gradient Boosting kann unser Optimierungsansatz als Gradientenabstieg im Funktionenraum formuliert werden. Jeder Gradientenschritt lässt die Bäume um eine Ebene wachsen, was durch die Suche nach Splitting-Funktionen für alle Knoten in dieser Ebene geschieht. Dieser Prozess kann parallel ausgeführt werden. Die berechneten Gradienten, die von der gewählten Zielfunktion abhängen, beeinflussen das Lernen der Splitting-Funktionen in den Bäumen. Auf

diese Weise kann jegliche differenzierbare Zielfunktion für das Trainieren von Random Forests verwendet werden. Wir präsentieren in dieser Arbeit eine Formulierung für das Klassifikations- und Regressionsproblem, sowie eine empirische Analyse des Lernmodelles auf einer Reihe von grundlegenden Benchmarks des Maschinellen Lernens. Unsere Experimente zeigen, dass bei ähnlicher Trainingslaufzeit, der vorgestellte Lernalgorithmus bessere Vorhersagen treffen kann als normale Random Forests.

Darüber hinaus stellen wir verschiedene Computer Vision Anwendungen vor, wo unsere neue Formulierung für das Trainieren von Random Forests erfolgreich angewandt wurde. Wir verwenden den Algorithmus für zwei verschiedene Ansätze der Objekterkennung, für die Schätzung der Pose des menschlichen Kopfes von Tiefendaten, sowie für das Vergrößern von Einzelbildern. Diese Anwendungen beinhalten sowohl Klassifikations- als auch Regressionsprobleme. Unsere Ergebnisse zeigen, dass sich die Optimierung einer gemeinsamen Zielfunktion über alle Bäume hinweg, im Vergleich zum unabhängigen Trainieren einzelner Bäume, klar auszahlt. Unsere Formulierung des Trainings von Random Forests kann einfach in bereits existierende Anwendungen, die Random Forests verwenden, integriert werden und verbessert die Ergebnisse für Klassifikations- und Regressionsaufgaben.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

The text document uploaded to TUGRAZonline is identical to the presented doctoral thesis.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Dissertation identisch.

Ort	Datum	Unterschrift

Acknowledgments

This thesis is the result of a great collaboration with many people who all supported me in many different ways and deserve my deepest gratitude. First and foremost I want to thank my supervisor Horst Bischof who not only aroused my interest in the field of computer vision, but also gave me the change to pursue my PhD at the Institute for Computer Graphics and Vision (ICG). Horst is a remarkable advisor and his scientific management skills definitely build the basis for the friendly and professional working environment at ICG. I also thank Prof. Fred A. Hamprecht for agreeing to review my work and for his efforts in being my second advisor.

Great thanks go to all members of the learning, recognition, and surveillance group. I thank Peter M. Roth, the boss, for his seemingly endless support and advice for all my scientific activities and for proof-reading all my publications with greatest detail. Special thanks go to my current and former office mates and collaborators, Paul Wohlhart, Martin Köstinger, and Georg Poier. The research-related discussions we had were one of the most important input for my PhD studies. I also won't forget the delicious non-scientific conversations and the almost unbelievably high-class jokes that made this office such a great place to work at. I express my deepest gratitude to Christian Leistner, who motivated me to pursue a PhD in the first place and then advised me and collaborated with me throughout my whole PhD time. Thank you for all the great scientific and also non-scientific discussions and the ideas we have had together and hopefully continue to have in the future! The organizers of the reading group, Michael Donoser and Paul Wohlhart, also have my biggest appreciation. This group of people who discussed various topics in computer vision and challenged recently published work was such a big gain for me and allowed me to have a much broader look on the field. I also thank all other members of the ICG for so much fun when playing soccer or table-soccer, having one (or maybe two) beer at MoXX, and enjoying discussions during the sometimes questionable lunch. Working at ICG would be hard without the non-scientific staff. I thank Nicole, Christina, Karin, Renate, Andi, Alexander and Daniel for all the help with forms and hardware issues.

Finally, I want to thank my friends and family. It is needless to say that my friends in Graz and Fürstenfeld have my greatest appreciation for all the unforgettable moments we had together in the past and hopefully will still have in the future. I thank my family for supporting me throughout my whole life. I am endlessly grateful to my Mum and Dad for enabling me to enjoy such an education. I will never forget your love and support. I also thank my grand parents, my step father, my parents in law, my uncles and aunts, and my cousins. Last but not least, I want to express my deepest gratitude of all to my wife, Elena, who always supports me and makes my life better every day!

Contents

CHAPTER 1

Introduction

The ability to see is arguably a highly valuable sense of human beings. It enables us to perceive the three-dimensional world around us in a unique and effective way. The human eye captures incoming light and forwards the information to the brain, which processes the huge amount of constantly arriving data. Together with inputs from the other human senses, a long- and short-term memory (experience), the human brain can process and understand this huge amount of incoming data effectively. Giving a detailed description of the scene depicted in Figure 1.1a is no problem at all for human beings. Even children of early ages can easily describe the main concepts of the illustrated scene in their own words. Being able to describe this scene demonstrates the power of the human visual system (in combination with lots of experience) independently from other human senses like hearing or touching. Figure 1.1b gives another example of a natural scene. Even though the objects are captured in rather unusual poses, our visual system has no problem to identify and outline the main objects in the two-dimensional image.

The ability of humans to understand and interpret the visual input is not inborn. *It can be learned.* The inspiring and peerless 'Project Prakäsh' [165] was initiated by Prof. Pawan Sinha in 2003 and has the goal to treat curably blind children. As a side effect, the researchers could conduct several studies with children that gained sight late in their lives. These studies underpin that perceiving and understanding the visual world around us can be learned [20, 125, 165]. Humans have the ability to recognize a huge range of different objects at different levels of abstraction. This ranges from general shapes like circles or triangles to very specific instances like a tennis ball or traffic signs.

Humans build and train their visual system based on loads of images seen during their lives. Assuming the human eye captures around 15 frames per second, a typical person that is awake for 16 hours observes $8.64 \cdot 10^6$ images per day on average. Thus, at the age

(a) **(b)**

Figure 1.1: Two examples of a real world scene captured with a standard consumer camera. The scenes can be easily described by humans with a few sentences. While many different descriptions of the image are valid, one simple description for (a) could be 'A beautiful scenery with a snow-covered mountain in the back and green trees in the front'. For (b), a description might be 'Two low-rider cars in a weired pose at an exhibition'.

of 10, a person already observed around $3.15 \cdot 10^9$ images. Annotation for all these images is very sparse, though. Only a few of the perceived images are annotated in the sense that a description of objects, actions, attributes, etc. is given. Many of these annotated images are typically observed by humans during their early childhood. This is the time when children often get asked to name the objects they see, e.g., in a storybook. It is kind of natural to ask children these sort of questions and congratulate them if they know the correct answer. In case they do not know it, the 'teacher' provides the correct answer, i.e., the annotation. Children learn these associations very effectively and are able to distinguish many object categories early in their lives. Another example for annotations of images could be books, magazines, or newspapers, which illustrate pictures with a specific caption that describes the given scene.

The field of computer vision aims at building computational systems with similar or even better abilities than humans to automatically understand, interpret, and transform images without any human interaction. Examples of specific problems in the field include localizing faces, estimating the pose of objects, or understanding interactions between humans in an image. Beside these high-level examples, computer vision also deals with low-level problems like removing noise from images (e.g., camera sensor noise), estimating motion between two consecutive video frames, or super-resolving images (e.g., for print media). For almost all of these tasks, building models and algorithms that can automatically *learn* from data plays a critical role. Teaching computers to learn from data is the field of Machine Learning (ML), which is obviously an inherent part of computer vision.

1.1 Machine Learning in Computer Vision

The task of *ML* algorithms in computer vision is to learn a function $f(\cdot)$ that maps the representation of an input image to the desired output, which totally depends on the specific task at hand. When we consider high-level computer vision tasks like image classification, object detection and tracking, semantic image segmentation, or action recognition, *ML* techniques can be considered as one of its core building blocks to associate the given image with the desired output.

One example is image classification, where the main object in a given image has to be determined, see Figure 1.2a. The current state-of-the-art employs Neural Networks (NN) to learn the mapping f. *NN* are able to distinguish between 1000 different object categories on 10000 test images with misclassification rates of below 10% [172]. Another example is face detection, which is illustrated in Figure 1.2b. To localize each face in a given image, one approach is to evaluate the image content of a window at each location and scale. The learned function f can be designed with a Boosting (Boosting) algorithm [185] and aims at predicting a probability of whether the current window captures a face or not. The 2D coordinates with high probability define the final detections.

On the other hand, *ML* has been employed more and more for low-level vision tasks in recent years, which include image denoising [51, 84, 121], image deblurring [149, 150], or 3D reconstruction [74]. To give one particular example of a low-level vision application that builds upon *ML* techniques, we consider Single Image Super-Resolution (SISR), see Figure 1.2c. Given a low-resolution image the desired output is a visually pleasing high-resolution image that preserves sharp edges and thus high contrast. So-called dictionary learning methods have extensively been applied for this problem [177, 204] to learn a mapping $f(\cdot)$ from the low- to the high-resolution domain.

These are only three examples of *ML* techniques successfully employed in computer vision applications. While *NN*, *Boosting*, and dictionary learning are employed for these examples, many other *ML* algorithms exist and are used for computer vision tasks. Other prominent representatives are nearest neighbors, decision trees, or Support Vector Machine (SVM). In this thesis, however, we focus on Random Forests (RF), a highly flexible and effective learning algorithm that builds on decision trees and was extensively used in the computer vision community.

1.2 Random Forests for Computer Vision Applications

RF have been introduced in the late 1990's by several researchers [3, 4, 23, 78, 79]. Since then on, *RF* have been employed successfully in many different applications ranging from object detection [62] to human pose estimation [163]. *RF* mainly owe its popularity and success to its generalization power and its beneficial properties that are attractive for many computer vision applications [28]: First, this learning algorithm can deal with high-dimensional data spaces as it inherently performs feature selection. Each node in the

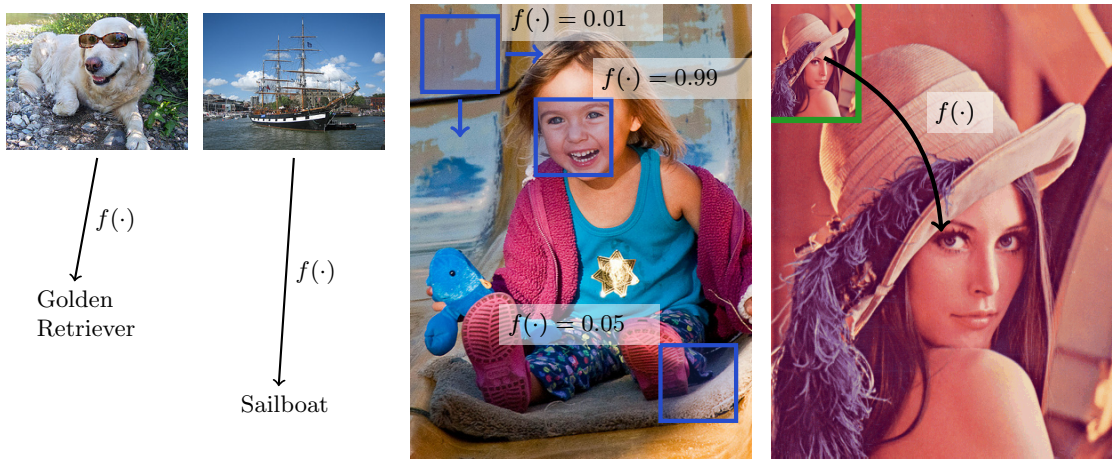


Figure 1.2: Three examples of classical computer vision tasks which all have *ML* algorithms at its core. (a) Current image classification methods can handle huge data sets like ImageNet [40] and achieve almost equal or even lower error rates than humans [92, 172]. (b) Face detection systems already achieved a level of performance in order to be successfully deployed in consumer cameras [185]. (c) In single image super-resolution a visually pleasing high-resolution image is produced given a low-resolution one by relying on a learned mapping between those domains.

trees of the *RF* operates only on a discriminative subset of the whole feature space. This property is very important for computer vision where data dimensionality often exceeds thousands or even hundred thousands of dimensions.

Another important advantage of using *RF* is the low computational cost for training and testing the model. *RF* consist of several trees that operate independently, allowing for straight-forward parallel training and inference on multi-core machines. Moreover, the recursive splitting of the data space turns a large prediction problem into a set of small ones, because the predictions made by the trees are always conditioned on previously evaluated splitting functions. Also testing in *RF* is fast as only a small fraction of the learned model has to be evaluated. In fact, only a single path from the root to the leaf node is considered for each tree and data sample, which is logarithmic in the model size.

Finally, *RF* can also deal with any kind of output space [43]. It can handle classification problems with multiple classes within the same model and even output probabilistic predictions. Other *ML* algorithms like *SVM* have to increase the model size and typically rely on some heuristics to deal with more than two classes. Also joint classification and regression problems can be naturally attacked with *RF* [50, 62]. *RF* can even be used for predicting structured outputs [43, 89].

To highlight the practical importance and the success of *RF* in the computer vision community, we briefly mention the human pose estimation system of Shotton et al. [163], which is implemented in the Microsoft® Kinect™ sensor. Given a depth image from this sensor, the task is to accurately infer the pose of a human being, see Figure 1.3. In the

Figure 1.3: The famous human pose estimation system proposed by Shotton et al. [163], integrated into the Microsoft® Kinect™ sensor. Given a depth image that captures a human body, this method infers a full skeleton in real-time. The image is taken from [163] and illustrates eight different depth images with the corresponding inferred human pose. Each color defines a different part of the human body.

original work, each pixel in the depth image is classified as one of several parts of the human body (e.g., head, torso, left arm, etc.) with a *RF* model. Based on this classification, a skeleton model is inferred and used to control computer games and other applications on the Microsoft® Xbox. Beside the human body pose estimation, *RF* have also been successfully employed for other applications, including facial fiducial detection [32, 37], head pose estimation [48–50], human pose estimation from RGB images [7, 38, 128], articulated hand pose estimation [173, 175], semantic segmentation [88, 89, 141, 164], or image denoising [51]. Figure 1.4 gives illustrative examples for each of these tasks. Note that this list of applications relying on *RF* is far from being complete. A good overview and a more thorough list can be found in [34].

1.3 Contribution and Outline

The content of this thesis is mainly based on the work presented in [152, 156–158]. These publications and this thesis is the result of a long-lasting and strong collaboration with my colleagues Christian Leistner, Paul Wohlhart, Peter M. Roth, and Horst Bischof.

The main part of the thesis starts in Chapter 2 with a review of the basic concepts of *ML*. We summarize the main learning paradigms, including supervised, unsupervised, and weakly-supervised learning. Popular *ML* algorithms for each of these concepts are also included. Then, in Section 2.2 we give an overview of ensemble methods, one specific group of *ML* algorithms. We start this overview with a detailed description of boosting algorithms in Section 2.2.2 and also present the famous AdaBoost algorithm [59]. Finally, we give a more thorough description of Gradient Boosting (GB) and *RF* in Sections 2.2.3 and 2.2.4, respectively, which build the basis for the main contribution of this thesis.

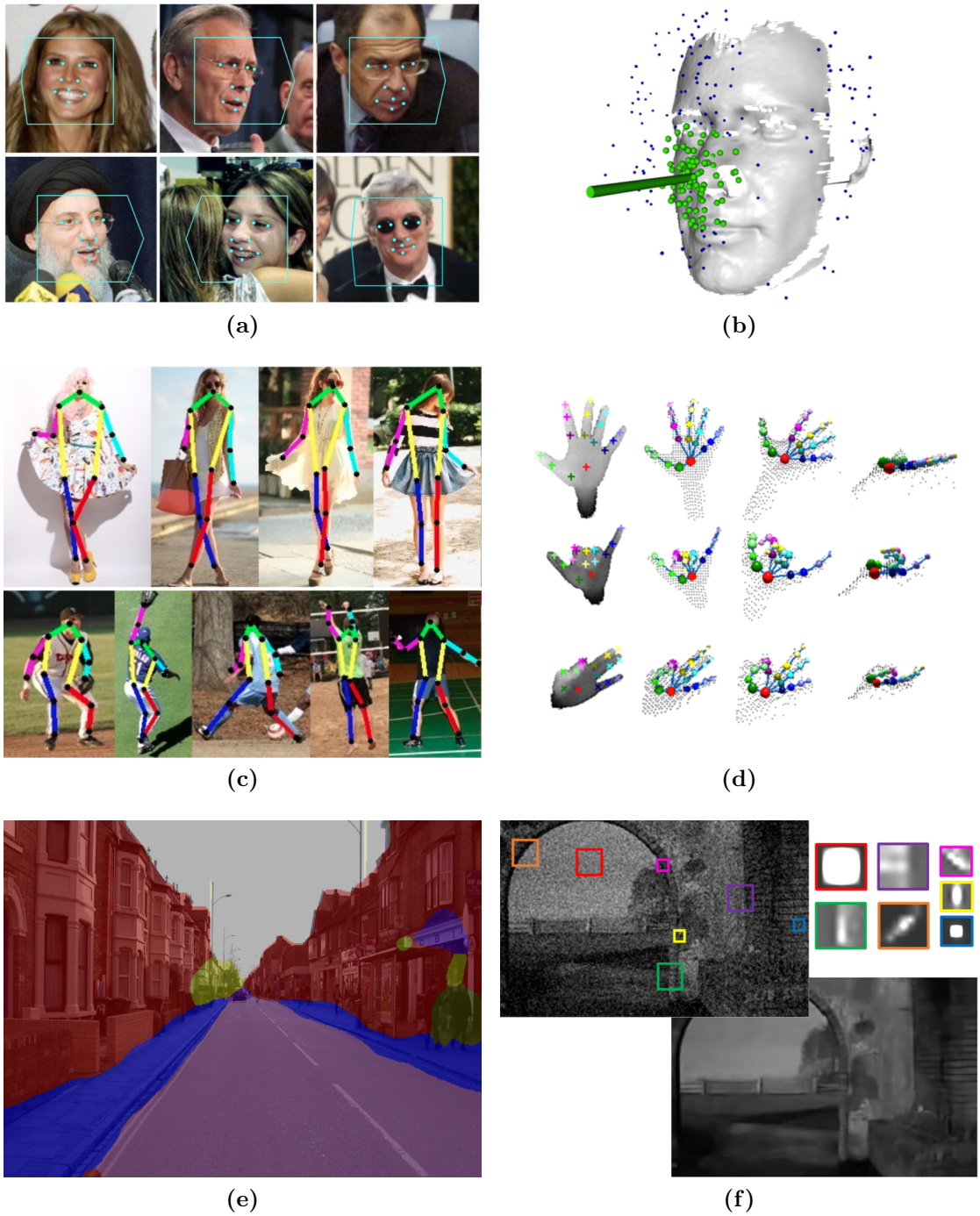


Figure 1.4: Some examples of successful applications building upon the *RF* algorithm: (a) facial landmark detection [37], (b) head pose estimation from depth images [49], (c) human pose estimation from RGB images [38], (d) articulated hand pose estimation [173], (e) semantic image segmentation [89], and (f) image denoising [51]. The images are taken and adapted from the corresponding citations.

RF are ensembles of binary trees that can be trained for various tasks, including classification and regression. While a single tree often gives poor results and tends to over-fit the data, combining several randomly or weakly trained trees via averaging into a single model performs significantly better [23]. In order for the averaging to make sense, it is essential that individual trees in the ensemble make different predictions, which is achieved by training the trees individually and injecting randomness into the training process. Individually training trees also comes with computational benefits as multiple cores in modern CPUs can be readily exploited. However, from a risk minimization point of view, there is no loss function that controls the training of the final model, i.e., the ensemble of all trees. Thus, the loss function optimized by plain *RF* only operates locally and for each tree individually, which can lead to suboptimal predictions.

In Chapter 3 we propose Alternating Decision and Regression Forests (ADRF) [156, 158], a novel training scheme for *RF* that optimizes a well-defined loss function over the full ensemble of trees and alleviates the above-mentioned problem. *ADRF* unites the loss minimization approach of *GB* with the flexibility and the computational benefits of *RF*. This training scheme incorporates the loss minimization directly into the tree growing process and enables to still train the trees in parallel. By doing so, we can incorporate any differentiable loss function into the training procedure of *RF*. As the name already suggests, we present formulations for both classification (Alternating Decision Forests (ADF) [158]) and regression tasks (Alternating Regression Forests (ARF) [156]). In the second part of Chapter 3, we also evaluate the proposed algorithm densely on standard *ML* benchmarks. We empirically investigate the properties of the algorithm and show that it compares favorably to both *RF* and *GB*.

In the remaining chapters of the thesis we show how to successfully apply *ADRF* for several computer vision applications. In Chapter 4, we use *ADRF* for object detection and integrate the proposed algorithm into two different frameworks. The first one is Hough Forests (HF) [62], which is used for object detection based on local evidence, see Section 4.1. We review the basic concepts of *HF*, show how to integrate *ADRF*, investigate related work, and present our experiments on standard object detection benchmarks. The second framework we are dealing with builds on a holistic object model [41]. These models typically rely on *Boosting* algorithms in a sliding window scheme in order to localize objects in unseen images and are restricted to a fixed-size bounding box that is predicted. We first show how *ADRF* (and thus also standard *RF*) can be readily integrated into this framework and, moreover, how the flexibility of *RF* and *ADRF* can be exploited in order to make more accurate bounding box predictions [157]. In Chapter 5, we use the concept of *HF* [62] and follow the work of [49, 50] to infer the pose (i.e., location and orientation) of a human head in 3D from depth images. Moreover, we present a novel holistic model to accurately infer the pose from a single depth patch capturing the human head. Again, we show how *ADRF* can be incorporated into both frameworks and that the prediction accuracy is improved over standard *RF* [156]. Chapter 6 deals with *SISR* and shows how *RF* and also *ARF* can be employed to achieve state-of-the-art results. Finally, we

conclude the thesis in Chapter 7, where we summarize and discuss the work and give an outlook of potential future work on this topic.

CHAPTER 2

Preliminaries and Related Work

Before we get to the main contribution of this thesis, we first define the notation and review the basic concepts of machine learning that we require for the later chapters. In the following section we start with a brief introduction to different aspects of machine learning, including different levels of supervision, e.g., fully-, weakly-, or unsupervised learning, as well as a review of popular learning algorithms. Then, we go into more detail on ensemble methods, a specific group of learning algorithms that combines several weaker models to a single stronger one. These methods have been shown to give extraordinary predictive accuracy and generalization capabilities and build the foundation of the algorithms proposed in this thesis.

2.1 Machine Learning

The field of Machine Learning (ML) is concerned with building computer algorithms that are able to learn from experience. According to Tom Mitchell [115], a general definition of *ML* can be stated as follows: *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

This definition can be best illustrated with an example. Assuming the task T is to recognize and classify handwritten digits in images, a natural performance measure P is the percentage of correctly classified digits. In this context, the experience E is defined as a large corpus of training images alongside with their correct classifications.

ML is a large field finding applications in many other scientific disciplines, e.g., natural language processing, medical diagnosis, stock market analysis, software engineering, recommender systems, and computer vision. While the current advances in this field are

still far away from the capabilities of how human beings can learn from experience, there already exist many success stories of *ML* algorithms that found their way into commercial products. A small excerpt of this list includes speech-recognition in mobile phones, face recognition in consumer cameras, or recommender systems in huge online stores.

In the following subsections, we provide a more formal definition of various *ML* problems, which are differentiated by the level of supervision, i.e., experience E, that is provided to the algorithm. As we deal with supervised learning algorithms in this thesis, we put more emphasis on this learning principle, which is explained in the following section.

Notation: Before we start with technical details and mathematical formulations in this thesis, we briefly outline the general style of notation. While scalars are not highlighted, we mark vectors bold-face, e.g., $\mathbf{x}$ or $\mathbf{y}$. Matrices and sets are marked bold-face and uppercase, e.g., $\mathbf{W}$. Running variables, indices, and counters (like the number of elements in a set) are marked typewriter, e.g., $\mathtt{j}$ or $\mathtt{N}$.

2.1.1 Supervised Learning

In a supervised *ML* problem we want to find a mapping function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X} = \mathbb{R}^{\mathtt{D}}$ is the $\mathtt{D}$ dimensional input or data space and $\mathcal{Y}$ is the output or label space that depends on the task at hand. Most *ML* problems can be formulated as classification or regression. For classification, $\mathcal{Y} = \{1, \dots, \mathtt{C}\}$, where $\mathtt{C}$ is the number of classes. For regression, $\mathcal{Y} = \mathbb{R}^{\mathtt{K}}$, where $\mathtt{K}$ defines the dimensionality of the output space. Please note that we want to keep the label space general and thus denote a single label $\mathbf{y} \in \mathcal{Y}$ as vector, i.e., bold-face, but it can also be a scalar, e.g., for classification.

In general, learning the mapping function f requires the definition of a loss function $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}} = f(\mathbf{x}))$, which measures the discrepancy between a ground truth label $\mathbf{y}$ and a prediction $\hat{\mathbf{y}}$. For classification, a common loss function that is used in theory is the 0-1 loss $\mathcal{L}_{01} = \mathbb{I}[\mathbf{y} \neq \hat{\mathbf{y}}]$. However, due to the discontinuity, approximations are used in practice. For binary classification, where $\mathtt{C} = 2$ and the label space is defined as $\mathcal{Y} = \{-1, +1\}$ for easier optimization, the exponential loss $\mathcal{L}_{\text{exp}} = \exp(-\mathbf{y} \cdot f(\mathbf{x}))$ or the hinge loss $\mathcal{L}_{\text{hinge}} = \max(0, 1 - \mathbf{y} \cdot f(\mathbf{x}))$ are often used. For regression, the squared loss $\mathcal{L}_{\ell_2^2} = \|\mathbf{y} - f(\mathbf{x})\|_2^2$ is commonly employed.

The goal in supervised *ML* is to find f such that the expected loss, which is also called the risk,

$$\mathcal{R}(f) = \mathbf{E}\left[\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})\right] = \int \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) \, \mathrm{d}p(\mathbf{x}, \mathbf{y}) \tag{2.1}$$

is minimized, i.e., $\arg\min_f \mathcal{R}(f)$. Unfortunately, the probability distribution $p(\mathbf{x}, \mathbf{y})$ is typically unknown, making sampling from it impossible. Hence, one has to rely on a finite training set

$$\mathbf{X} = \{\{\mathbf{x}_{\mathtt{n}}, \mathbf{y}_{\mathtt{n}}\} \in \mathcal{X} \times \mathcal{Y} \,|\, \mathtt{n} = 1, \dots, \mathtt{N}\} \tag{2.2}$$

consisting of $\mathtt{N}$ input-output pairs, where we call $\mathbf{x}$ a data sample and $\mathbf{y}$ a label. The data

is typically a specific feature representation of some other entity like text, audio, medical data, physical measurements, etc. For the computer vision applications we are addressing in this thesis, we extract features from images $\mathcal{I}$ such that $\mathbf{x} = \Phi(\mathcal{I})$, which we describe later in more detail. The ground-truth label $\mathbf{y}$ for each $\mathbf{x}$ typically has to be provided by a human annotator who evaluates $\mathbf{x}$ or the underlying entity, e.g., $\mathcal{I}$.

Due to the unknown probability distribution $p(\mathbf{x}, \mathbf{y})$ and the finite training set $\mathbf{X}$, one has to approximate (2.1) with the empirical risk

$$\mathcal{R}_{\text{emp}}(f) = \frac{1}{\mathtt{N}} \sum_{\mathtt{n}=1}^{\mathtt{N}} \mathcal{L}(\mathbf{y}_{\mathtt{n}}, \hat{\mathbf{y}}_{\mathtt{n}} = f(\mathbf{x}_{\mathtt{n}})) \;, \tag{2.3}$$

which again should be minimized, i.e., $\arg\min_f \mathcal{R}_{\text{emp}}(f)$. While this approximation might seem negligible in the first place, it can have a drastic impact on the final prediction performance on the learned function $f(\cdot)$. The reason is *over-fitting*. This phenomenon describes the situation when the learned function f over-fits the training data and performs poorly on unseen test data $\mathbf{X}_{\text{new}}$. Over-fitting typically occurs if the complexity of the chosen function f is too high. In this case, the learning algorithm often has no problems adjusting the parameters of f to achieve low empirical risk. At the same time, though, it is very likely that the learned function performs poorly on $\mathbf{X}_{\text{new}}$, i.e., the *generalization* error is high. There also exists the opposite of this phenomenon, namely under-fitting, which means that the chosen model is too simple to handle the given learning problem. The actual goal of supervised learning is to minimize the generalization error (or the expected error in general), i.e., how well does the model perform on data that is not seen during training. Thus, one has to select a suitable class of functions with a reasonable complexity for the problem at hand. Some basic strategies for this process are given in [46, 75]. It is also common practice to include a regularization term $\Gamma(f)$ on the model parameters of f, which also penalizes the complexity. In this case, learning f can be defined as minimizing

$$\mathcal{R}_{\text{emp}}(f) = \frac{1}{\mathtt{N}} \sum_{\mathtt{n}=1}^{\mathtt{N}} \mathcal{L}(\mathbf{y}_{\mathtt{n}}, \hat{\mathbf{y}}_{\mathtt{n}} = f(\mathbf{x}_{\mathtt{n}})) + \Gamma(f) \;. \tag{2.4}$$

This principle follows the intuition of Occam's Razor, which advocates to prefer simpler explanations over needlessly complicated ones [18, 115]. Computational learning theory provides different formulations to assess the generalization error (or the expected error). Examples include the Akaike information criterion (AIC), Bayesian information criterion (BIC), minimum description length (MDL), or the Vapnik-Chervonenkis (VC) dimensions. A good overview of these formulations can be found in [75]. However, all of them have some limitations, either that they are only valid for a restricted class of prediction and loss functions, or that they are simply hard to apply for certain classes of functions and only provide a rough upper bound on the generalization error. VC dimensions are the most general ones and Vapnik also proposed a structural risk minimization [182] approach

for Support Vector Machine (SVM), which considers the model complexity during the learning process. A more practical way to do model selection is cross-validation, which is well described in [75].

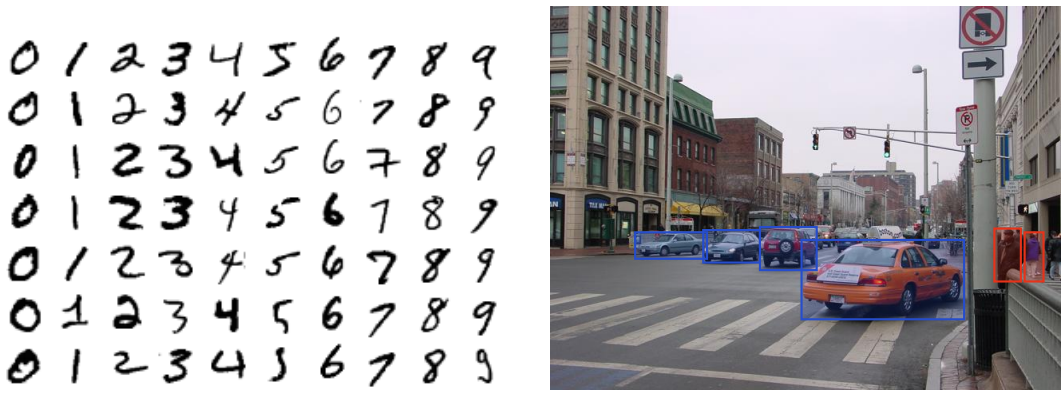
The problem of the generalization performance is also addressed in the bias-variance decomposition. One can define the expected prediction error $\bar{E}$ on a high-level as

$$\bar{E} = \sigma^2 + bias^2 + variance \tag{2.5}$$

The first term on the right hand side of Equation (2.5) defines the irreproducible error that stems from noise in the data generation process, i.e., in the unknown distribution $p(\mathbf{x}, \mathbf{y})$. The second term, the squared bias, describes the average difference between the predictions of a trained model and the true label $\mathbf{y}$. The last term is the variance of the trained model. Please note that the expectation (or the average) is defined over several randomly sampled training sets. One speaks about the bias-variance *dilemma* because when we make the model more complex, one typically can observe a lower bias but, at the same time, a higher variance of the model. Hastie et al. [75] provide an intuitive example with a k-Nearest Neighbor regression fit. A detailed description of this topic is given in [75] (Chapter 7). We will use this argumentation of bias and variance later for ensemble methods in Section 2.2 to describe some of their beneficial properties.

Supervised *ML* also differentiates between generative and discriminative models. As the name might already suggest, a generative model tries to directly learn the data generating probability distribution $p(\mathbf{x}, \mathbf{y})$ or $p(\mathbf{x}|\mathbf{y})$. This approach has the advantage that new data can be directly sampled from the model, which is a natural task. Consider the human brain, for instance: one can easily imagine a picture of a car or a train, i.e., we sample from this probability distribution. Dreaming is another example where the human brain generates new data. Discriminative models, on the other hand, only learn the discriminating function $f(\mathbf{x})$ between different classes or different target values, not the underlying data distribution. In most practical benchmarks and tasks, discriminative models typically outperform generative ones. A reason for this might be that quantitative evaluation is only concerned with the final predictions $\hat{\mathbf{y}} = f(\mathbf{x})$ and neglects how well new data can be generated from the model. While learning a generative model typically renders a harder task, discriminative models solely focus on learning f, giving them a crucial benefit on standard benchmarks. Here, we also focus on discriminative algorithms.

The *ML* literature presents an almost endless list of algorithms for all kinds of tasks. Reviewing all of them is far beyond the scope of this thesis and we refer the interested reader to [18, 46, 115]. Popular and effective *ML* algorithms include Fisher (or linear) discriminate analysis [54, 134], linear regression, logistic regression, *SVM* [182], classification and regression trees (CART) [24], neural networks [140], convolutional neural networks [95–97], or nearest neighbor. Another strand of *ML* algorithms are ensemble methods like Boosting (Boosting) and Random Forests (RF), which are the basis for this thesis and are devoted a separate section (see Section 2.2).



(a) (b)

Figure 2.1: Two examples of typical supervised *ML* tasks. (a) Handwritten digit classification on the popular `MNIST` data set. For each of the 10 classes (columns), the figure illustrates 7 training images (rows). (b) Object detection on a typical street-scene image. The figure illustrates the ground-truth annotation of objects that have to be detected, cars in blue and pedestrians in red. The annotation is given as bounding boxes around the objects. Please note that not all objects in the scene are annotated.

To conclude this section on supervised *ML*, we provide a set of example applications in order to better illustrate the problem. A very popular task is that of handwritten digit recognition. It is a classification problem with $\mathcal{X} \in \mathbb{R}^{28\times 28}$ and $\mathcal{Y} = \{1, \ldots, 10\}$. Examples for the given data can be seen in Figure 2.1a. Obviously, the task is to recognize the handwritten digit in an unseen 28×28 image. The task is relatively easy as the input data is clean and pre-processed. The current state-of-the-art already achieves recognition rates above 99.7% [189]. Another, more evolved problem is object detection in still images, which involves localizing and classifying semantic objects like pedestrians, cars, or animals. A typical object detection system requires more than just a learning algorithm but also other components like a localization scheme (e.g., a sliding window over a scale space pyramid) to find objects. An example is given in Figure 2.1b.

2.1.2 Unsupervised Learning

Contrary to supervised *ML*, the unsupervised learning paradigm has no access to any training signal $\mathbf{y} \in \mathcal{Y}$. Thus, as the name already suggests, no supervision is available, which immediately raises the question of the task to optimize for. As we have a strong focus on supervised algorithms in this thesis, we only provide a brief overview of different subtasks of unsupervised learning here. The goals can be diverse and the most popular tasks can summarized as follows.

Clustering: Given a set of data samples $\mathbf{X}_\mathcal{X} = \{\mathbf{x}_\mathtt{n}\}$ with $\mathtt{n} = 1, \ldots, \mathtt{N}$, the task is to find coherent groups of data within $\mathbf{X}_\mathcal{X}$ according to some similarity measure. Each

resulting group is also called a cluster. Each data sample $\mathbf{x}$ belongs to at least one cluster, either via hard assignment (one sample belongs to exactly one cluster) or via soft assignment (one sample belongs to all clusters with a certain weight or probability). Popular clustering algorithms are k-means [105], graph-based methods [60, 162], or discriminative clustering [39, 198].

Density estimation: Another example of unsupervised learning is density estimation. The task is to find an estimate of the density of $\mathcal{X}$, i.e., to find the underlying data-generating probability distribution $p(\mathbf{x})$ with the given data set $\mathbf{X}_{\mathcal{X}}$. Fitting a Gaussian mixture model with expectation-maximization [18] on a set of data is one algorithm that can be used to estimate the density of a given data set. *RF* [4, 23, 78] can also be used for density estimation [34]. One can already observe the close relation between density estimation and clustering as it can be understood as a softer version of the latter.

Dimensionality reduction: The task of dimensionality reduction is to find a function that maps high-dimensional data $\mathbf{x}_H \in \mathbb{R}^{\mathtt{D_H}}$ to low-dimensional data $\mathbf{x}_L \in \mathbb{R}^{\mathtt{D_L}}$ in such a way that most information about the data is preserved. The most popular algorithm is the principal component analysis (PCA) [80, 86]. While being a simple linear algorithm, it has shown to be very effective in practice. However, also non-linear alternatives exist, like the kernelized version of PCA (kPCA) [151] or autoencoders [77] (a special form of neural networks). Dimensionality reduction is often important for practical reasons as a pre-processing step for large systems that have to handle high-dimensional data.

Manifold learning: This task is highly related to dimensionality reduction but with the explicit goal of finding a lower dimensional manifold of the data. The term manifold learning is often interchangeably used with non-linear dimensionality reduction, although the latter one is only concerned with reducing the dimensionality and not with the explicit goal of finding the underlying structure. However, dimensionality reduction also often implicitly defines a low-dimensional manifold of high-dimensional data. Thus, these two disciplines are highly related and overlap in many works. Popular algorithms for manifold learning include self-organizing maps [87], autoencoders [77], kPCA [151], isomaps [176], or locally-linear embedding [143].

2.1.3 Weakly Supervised Learning

Weakly-supervised learning can be seen as a mixture between supervised and unsupervised *ML*. Many different formulations have been proposed and discussed in the *ML* literature in recent years. Most of them typically stem from some practical applications and problems that can be readily formulated in a weakly-supervised form. The two most popular examples of weak supervision are semi-supervised and multiple-instance learning, which we discuss in the following.

Semi-Supervised Learning: This concept assumes that a set of fully labeled samples $\mathbf{X}_\mathrm{L} = \{\mathbf{x}_\mathtt{n}, \mathbf{y}_\mathtt{n}\}_{\mathtt{n}=1}^{\mathtt{N}_\mathrm{L}}$ as well as a set of unlabeled samples $\mathbf{X}_\mathrm{U} = \{\mathbf{x}_\mathtt{n}\}_{\mathtt{n}=1}^{\mathtt{N}_\mathrm{U}}$ are given. The unlabeled set is typically much larger than the supervised one, i.e., $\mathtt{N}_\mathrm{L} \ll \mathtt{N}_\mathrm{U}$. The motivation for such a learning concept is clear: The often tedious and expensive task of manually labeling the data is reduced compared to supervised *ML*, while the learning task becomes much easier than in a fully unsupervised setup. However, one has to effectively make use of the unlabeled data set $\mathbf{X}_\mathrm{U}$. Many *ML* algorithms have been introduced or extended for semi-supervised learning, including *SVM* [15], *Boosting* [102, 109], and *RF* [103], among many others. A more detailed discussion of this topic can be found in [30, 205]. Applications making use of semi-supervised learning include text classification [169], image classification [102], or visual object tracking [72].

Multiple-Instance Learning: The multiple-instance learning scenario introduces the concept of bags of samples. A bag $\mathbf{B}_\mathtt{n}$ contains a set of samples $\mathbf{x}$ and has a (binary) label $\mathbf{y} \in \{-1, +1\}$, i.e., $\mathbf{B}_\mathtt{n} = \{\{\mathbf{x}_\mathtt{n}^\mathtt{j}\}, \mathbf{y}_\mathtt{n}\}$ with $\mathtt{n} = 1, \ldots, \mathtt{N}$ and $\mathtt{j} = 1, \ldots, \mathtt{N}_\mathtt{n}$. Multiple-instance learning is typically considered as a binary classification problem. Nevertheless, samples within the bags are only constrained on the label of its bags. Positive bags, i.e., $\mathbf{y} = +1$, contain at least one positive sample but the label of the remaining samples is unknown. For negative bags all samples within the bag can be assumed negative. This concept describes a different form of weak supervision than semi-supervised learning. Again, many learning algorithms have been proposed or extended for multiple-instance learning. These include *SVM* [6], *Boosting* [187], and *RF* [104]. Multiple-instance learning has been applied to different applications in computer vision like visual object tracking [104] or image classification [111].

Besides these two popular examples, other forms of weak supervision exist as well. This is especially true for the field of computer vision, where label information is often ambiguous and expensive to obtain. One example is an image-level annotation that only defines the existence of an object within the image, but not its exact location. A task could be to learn an object model or a mid-level representation from such ambiguous supervision, e.g., [116, 131, 154]. Another example concerns video data, where supervision can be given in the form of a video-level annotation. Again, only the existence of a certain object in a video is provided, but the exact temporal and spatial location is missing. Typical applications include action recognition [166, 167].

2.1.4 Online Learning

Up to now, we always assumed that all training data is readily available at all times during training a model. However, in many real-world scenarios only a single training example is given at a time. It can be used to update the prediction model, but is not available anymore after the update. We also mention that there exist slightly other forms than

strict online learning where, for instance, a small buffer of samples can be kept in memory. A larger deviation from the standard model is incremental learning where old samples can be stored but new ones are arriving one at a time.

Many algorithms can be considered online per se, because they build on an online optimization technique like stochastic gradient descent (e.g., Neural Networks (NN) or special formulations of *SVM* [160]). However, other learning algorithms have also been extended for the online scenario, like *Boosting* [71] and *RF* [145, 155].

The importance of such online applications becomes obvious when considering platforms with limited memory, closed systems with privacy issues, e.g., smart cameras, or the access to a huge amount of data that cannot be processed at once. Other examples include real-time systems that steadily have to adapt to the scene, e.g., in surveillance [130]. A popular example in the field of computer vision is object tracking: Given a video sequence capturing any kind of object and a bounding box annotation in the first frame, the task is to track the object throughout the whole sequence [71, 101, 145, 155]. Obviously, a model can be learned with the annotation of the first frame. However, the appearance of the object will change over time due to different illuminations, viewpoints, occlusions, etc., making the online adaptation of the object model necessary.

2.1.5 Transfer Learning

All *ML* principles discussed so far shared the fundamental assumption that training and testing data stem from the same probability distribution. In many cases, this assumption is valid. For instance, when collecting a data set for some computer vision task, one typically first collects images alongside with the ground-truth annotations and, then, splits the data into training and testing sets. In this case, one can assume the assumption on the underlying probability distribution of the data to be fulfilled.

On the other hand, there are many interesting scenarios where this basic assumption might be invalid. On a high level, consider the task of learning to speak Spanish. It is often said to be easier if you already know how to speak Italian. In this example, the task of transfer learning would be to *transfer* knowledge from one task (learning Italian) to another task (learning Spanish). In computer vision, transfer learning can also be seen as learning some predictor that can generalize from one data set to another for the same task, as each data set (or benchmark) has its own bias [179].

There are many different sub-categories in the field of transfer learning, which often vary in the amount of supervision that is given for each of the domains. A detailed review is out of scope of this thesis and we refer the interested reader to an excellent summary from Pan and Yang [126].

2.2 Ensemble Methods

Ensemble methods are a family of machine learning algorithms that combine several weaker models into a single stronger prediction function [75]. In order for ensemble methods to work, each weak model is required to perform better than random guessing and to be different to other weak models in terms of their predictions. Thus, the diversity of the weak models has to be given. Prominent examples of ensemble methods are *Boosting* and *RF*, which both build the basis for the algorithms presented in this thesis. In the following sections, we give a detailed description of both algorithms, *Boosting* in Sections 2.2.2 and 2.2.3 and *RF* in Section 2.2.4. Before that, we start with a more simple ensemble method named bootstrap aggregation. A good overview and a detailed summary of ensemble methods can be found in [46, 75].

2.2.1 Bootstrap Aggregation

Bootstrap aggregation can be considered one of the first ensemble methods [21] and as predecessor of *RF*. This method actually stems from a sampling technique called bootstrap that is used to compute the accuracy of a statistical parameter estimate. It is also often called 'bagging', and we will also use this abbreviation in the following.

The general procedure of bagging is as follows. Assume we have a data set of `N` samples. Then we draw `N` samples *with replacement* and equal probability from the data. This process is repeated `B` times resulting in `B` samples of size `N` from the original data, each one slightly different. Then, `B` predictors (may it be a classifier or a regressor) are trained on the corresponding `B` samples of the data set. The final prediction for a new data point $\mathbf{x}$ of a bagging model is the unweighted average of the predictions of the `B` individual models.

Bagging typically improves the results in scenarios where the individual predictors have low bias but high variance, as it is the case for regression trees, for instance. In this case, averaging can drastically reduce the variance while keeping the low bias [75], resulting in a better overall prediction. This effect can become more prominent when the individual prediction models make uncorrelated predictions. This idea follows the 'Wisdom of Crowds' principle [171], which states that the collection of knowledge from an independent and diverse set of people is typically better than the knowledge of a single one. However, in a standard bagging setup, the individual models are often not independent, as they are trained in the same way with the single difference of a slightly altered training set [75]. As we will see later, *RF* overcome this problem by using randomized decision or regression trees as their individual prediction models. The randomization of the tree growing procedures is beneficial to keep the correlation between individual models low.

2.2.2 Boosting

The general *Boosting* algorithm builds a prediction model by iteratively adding weak learners. Each weak learner is trained in a way to make up for the errors of the current state of the model, i.e., the sum of all weak learners added so far. While there exist an almost endless list of different *Boosting* variants, we first describe in Section 2.2.2.1 the most basic and most popular one, AdaBoost [59]. In Section 2.2.2.2, we describe some important properties of *Boosting* and review some interesting variants. Later, we will devote a more detailed section on Gradient Boosting (GB) (Section 2.2.3), which is a more generic variant of *Boosting* and also builds the basis for the main contribution of this thesis.

2.2.2.1 The AdaBoost Algorithm

AdaBoost is the most popular *Boosting* algorithm, which is the reason we want to briefly review the algorithm and some of its properties. Many works by both Freund and Schapire can be considered to contribute to this popular algorithm [58, 59, 147]. A very recent and broad summary of *Boosting*, including AdaBoost, can be found in [148].

AdaBoost is a supervised learning algorithm for binary classification tasks and requires a training set $\mathbf{X} = \{\mathbf{x}_\mathtt{n}, \mathbf{y}_\mathtt{n}\}_{\mathtt{n}=1}^{\mathtt{N}}$, where $\mathbf{y} = \{-1, +1\}$. The basic idea is to build a strong classifier $F(\mathbf{x})$ as a weighted sum of $\mathtt{T}$ weak learners $f(\mathbf{x})$, i.e.,

$$F(\mathbf{x}) = \sum_{\mathtt{t}=1}^{\mathtt{T}} \alpha_\mathtt{t} f_\mathtt{t}(\mathbf{x}) \ . \tag{2.6}$$

The final binary decision is then computed as $sign\left[F(\mathbf{x})\right]$, where $sign\left[\cdot\right]$ is the sign operator. Learning the weighted sum of weak learners is an iterative process. Each training sample $\mathbf{x}_\mathtt{n}$ is assigned a weight $w_\mathtt{n}$, which corresponds to the current 'importance' and is initialized with $w_\mathtt{n} = \frac{1}{\mathtt{N}}$ for all samples. The influence and meaning of this weight becomes clear later in this section. An overview of the whole learning process is given in Algorithm 1. As can be seen, each iteration $\mathtt{t}$ first finds a new weak learner $f_\mathtt{t}(\mathbf{x})$ that minimizes the weighted classification error in Equation (2.8). Leo Breiman observed in [22] that AdaBoost can also be formulated in a loss minimization framework and showed that the exponential loss

$$\mathcal{L}(F) = \sum_{\mathtt{n}=1}^{\mathtt{N}} e^{-\mathbf{y}_\mathtt{n} \cdot F(\mathbf{x}_\mathtt{n})} \tag{2.7}$$

is minimized. This interpretation decouples the algorithm from its objective and proved to be useful when developing new boosting variants.

Many different choices for $f_\mathtt{t}(\mathbf{x})$ are possible, as long as it makes predictions better than random guessing. A popular weak learner is a decision stump, i.e., a tree with a single splitting function and two leaf nodes. Having fixed a weak learner $f_\mathtt{t}$, the corresponding

error $\epsilon_{f_\mathtt{t}}$ is used to compute the influence $\alpha_\mathtt{t}$ of the weak learner in Equation (2.9). Finally, the weights of each data sample are updated via (2.10). The weight of samples that are correctly classified by the current weak learner, i.e., $\mathbf{y} f_\mathtt{t}(\mathbf{x}) > 0$, will be decreased and vice versa. This can be seen by noting that, by definition, $\epsilon_{f_\mathtt{t}} < \frac{1}{2}$, because we assumed each weak learner to be better than random guessing (otherwise $f_\mathtt{t}$ would be discarded). This implies $\alpha_\mathtt{t} > 0$. Hence, $\exp(-\alpha_\mathtt{t} \mathbf{y}_\mathtt{n} f_\mathtt{t}(\mathbf{x}_\mathtt{n})) < 1$ if $\mathbf{x}_\mathtt{n}$ is correctly classified. Obviously, the intuition of this update rule is that misclassified samples become more important (the weights become larger) for later iterations of the training process.

Algorithm 1: The training process of AdaBoost.

input : Labeled training set $\{\mathbf{x}_\mathtt{n}, \mathbf{y}_\mathtt{n}\}_{\mathtt{n}=1}^{\mathtt{N}} \in \mathcal{X} \times \mathcal{Y}$
input : Number of iterations $\mathtt{T}$
output: The set of weak learners $f_\mathtt{t}(\mathbf{x})$ with corresponding weights $\alpha_\mathtt{t}$
Initialize $w_\mathtt{n} = \frac{1}{\mathtt{N}}$ for $\mathtt{n} = 1, \ldots, \mathtt{N}$;
for $\mathtt{t}$ *from* 1 *to* $\mathtt{T}$ **do**
Find a weak learner

$$f_\mathtt{t} = \arg\min_f \epsilon_f = \frac{\sum_{\mathtt{n}=1}^{\mathtt{N}} w_\mathtt{n} \mathbb{I}\left[\mathbf{y}_\mathtt{n} \neq f(\mathbf{x}_\mathtt{n})\right]}{\sum_{\mathtt{n}=1}^{\mathtt{N}} w_\mathtt{n}} \tag{2.8}$$

minimizing the weighted miss-classification error ;
Compute the influence of the current weak learner via

$$\alpha_\mathtt{t} = \ln\left(\frac{1 - \epsilon_{f_\mathtt{t}}}{\epsilon_{f_\mathtt{t}}}\right) . \tag{2.9}$$

where $\epsilon_{f_\mathtt{t}}$ is the error on the training set of the current weak learner ;
Update weights of each data sample

$$w_\mathtt{n} = \frac{w_\mathtt{n} \exp(-\alpha_\mathtt{t} \mathbf{y}_\mathtt{n} f_\mathtt{t}(\mathbf{x}_\mathtt{n}))}{Z_\mathtt{t}} , \tag{2.10}$$

where $Z_\mathtt{t}$ is a normalization constant ;

2.2.2.2 Properties and Variants of Boosting

Boosting in general is a very strong and flexible classifier and has been studied extensively in recent years. It is beyond the scope of this thesis to give a broad review of all these *Boosting* variants and investigations. While the interested reader is referred to [148] for more details, we briefly mention the most important properties, variants, and applications of this learning principle.

The most obvious flaw of plain *Boosting* is the danger of over-fitting in the presence of noisy data. Consider, for instance, a mislabeled data point $\mathbf{x}_w$ in the training set $\mathbf{X}$, i.e.,

the annotation $\mathbf{y}_w$ is wrong. In general, the weak learners are found as to minimize the weighted misclassification error. Obviously, the larger portion of correctly labeled samples in $\mathbf{X}$ overrule $\mathbf{x}_w$ and a reasonable weak learner $f(\mathbf{x})$ will be found. This is because the weights w of all sample in the beginning of the learning process are more or less equal. If $f(\mathbf{x})$ is a good weak learner, it will also classify $\mathbf{x}_w$ correctly. However, as the annotation is wrong, its weight w_w starts to increase instead of decrease throughout the iterations. At some point of the training process, this weight will be so high that it strongly influences the search for the next weak learner. Hence, label noise can bias *Boosting* and lead to over-fitting. Nevertheless, also this issue is already addressed in the literature, see e.g., [103, 112, 113].

Another shortcoming of plain *Boosting* is that it is only formulated for the binary classification task. Fortunately, many different extensions for the multi-class case [58] as well as for regression [75] exist. *Boosting* has also been extended to weakly-supervised scenarios like semi-supervised learning [102, 109] or multiple-instance learning [187]. An online version of *Boosting* exists [71] as well as a version for structured output learning [161].

Also, the list of applications building upon *Boosting* is almost endless, even if only the field of computer vision is considered. One of the most popular examples is the face detection system of Viola and Jones [185]. They use *Boosting* in combination with Haar features for highly robust face detection. This work is the basis for the current state-of-the-art pedestrian detection systems [13, 41, 42]. *Boosting* also has a long and successful history in visual object tracking applications [10, 71, 72, 203].

2.2.3 Gradient Boosting

In this section, we review a generalization of *Boosting* that is called gradient boosting. In contrast to AdaBoost, which optimizes the exponential loss, *GB* can incorporate any differentiable loss function. This algorithm, together with *RF* (described later in this chapter), builds the basis for the main contribution of this thesis.

In [61], Friedman showed that *Boosting* can also be understood as performing gradient descent in function space, paving the way for *GB*. The training procedure of *GB* runs in $\mathtt{T}$ iterations, where in each of them a new weak classifier $f_\mathtt{t}(\mathbf{x})$ is trained, its contribution rate $\nu_\mathtt{t}$ determined, and both added to the strong model F. In more detail, given a labeled training set $\mathbf{X}$ with $\mathtt{N}$ samples, training the parameters of a single weak learner can be written as

$$\Theta^\mathtt{t} = \arg\min_{\Theta} \sum_{\mathtt{n}=1}^{\mathtt{N}} \mathcal{L}(\mathbf{y}_\mathtt{n}; F_0^{\mathtt{t}-1}(\mathbf{x}_\mathtt{n}) + f_\mathtt{t}(\mathbf{x}_\mathtt{n}; \Theta)) \; . \tag{2.11}$$

Here, $\mathcal{L}(\cdot)$ is a differentiable loss function, $F_0^{\mathtt{t}-1}(\mathbf{x}) = \sum_{j=0}^{\mathtt{t}-1} \nu_\mathtt{t} \cdot f_j(\mathbf{x}; \Theta^j)$ describes the already trained classifier and $f_\mathtt{t}(\mathbf{x}; \Theta^\mathtt{t})$ is the classifier in the current iteration $\mathtt{t}$; $\nu_\mathtt{t}$ is often set constant and called the shrinkage factor [75].

As shown by Friedman [61], solving (2.11) can be understood as steepest descent in

the N-dimensional input data space. Thus, one can compute the negative gradient

$$-\mathbf{g}_{\mathtt{t}}(\mathbf{x}_{\mathtt{n}}) = \left[\frac{\partial \mathcal{L}(\mathbf{y}_{\mathtt{n}}, F(\mathbf{x}))}{\partial F(\mathbf{x})}\right]_{F(\mathbf{x})=F_0^{\mathtt{t}-1}(\mathbf{x}_{\mathtt{n}})} \tag{2.12}$$

for each data sample $\mathbf{x}_{\mathtt{n}}$. While the gradients in (2.12) are only defined on the data samples $\mathbf{x}_{\mathtt{n}}$ and cannot be generalized to any $\mathbf{x} \in \mathcal{X}$, one can still find a weak learner $f_{\mathtt{t}}(\mathbf{x}, \Theta^{\mathtt{t}})$ that gives correlated predictions to $-\mathbf{g}_{\mathtt{t}}(\mathbf{x}_{\mathtt{n}})$. In other words, one has to train a new weak learner with an adapted training set $\mathbf{X}_{\mathbf{g}_{\mathtt{t}}} = \{\mathbf{x}_{\mathtt{n}}, -\mathbf{g}_{\mathtt{t}}(\mathbf{x}_{\mathtt{n}})\}_{\mathtt{n}=1}^{\mathtt{N}}$. While this can be easily done for regression or binary classification tasks (with continuous output values), for a multi-class problem with multi-class capable weak learners (e.g., classification trees), there exists an alternative way to train $f_{\mathtt{t}}(\mathbf{x}, \Theta^{\mathtt{t}})$.

Obviously, one can also interpret the norm of $-\mathbf{g}_{\mathtt{t}}(\mathbf{x}_{\mathtt{n}})$ as an importance factor for data sample $\mathbf{x}_{\mathtt{n}}$. As is done in ordinary *Boosting* [58, 59, 147], each data sample gets assigned a weight $w_{\mathtt{n}}^{\mathtt{t}}$ indicating how well this particular sample is already classified for each iteration t. Low values indicate a good classification and vice versa, which allows the model to subsequently put its emphasis on hard samples (i.e., those that have been misclassified). These weights get updated in each iteration t of the training based on the norm of the gradients, i.e.,

$$w_{\mathtt{n}}^{\mathtt{t}} = |-\mathbf{g}_{\mathtt{t}}(\mathbf{x}_{\mathtt{n}})|\,. \tag{2.13}$$

During training the weak learner $f_{\mathtt{t}}(\mathbf{x}, \Theta^{\mathtt{t}})$, these weights have to be taken into account.

Algorithm 2: Training procedure of *GB* for the regression case.

input : Labeled training set $\{\mathbf{x}_{\mathtt{n}}, \mathbf{y}_{\mathtt{n}}\}_{\mathtt{n}=1}^{\mathtt{N}} \in \mathcal{X} \times \mathcal{Y}$
input : Number of iterations T
input : Shrinkage factor ν
output: The set of weak learners $f_{\mathtt{t}}(\mathbf{x})$ with corresponding parameters $\Theta^{\mathtt{t}}$
Initialize $F_0^0 = const$;
for t *from* 1 *to* T **do**
  Compute all gradients $\mathbf{g}_{\mathtt{t}}(\mathbf{x}_{\mathtt{n}})$ with $F_0^{\mathtt{t}-1}(\mathbf{x})$ according to Equation (2.12) ;
  Generate adapted data set $\mathbf{X}_{\mathbf{g}_{\mathtt{t}}}$;
  Train weak learner $f_{\mathtt{t}}(\mathbf{x}; \Theta^{\mathtt{t}})$ with data set $\mathbf{X}_{\mathbf{g}_{\mathtt{t}}}$;
  Set

$$F_0^{\mathtt{t}}(\mathbf{x}) = F_0^{\mathtt{t}-1}(\mathbf{x}) + \nu \cdot f_{\mathtt{t}}(\mathbf{x}; \Theta^{\mathtt{t}}) \tag{2.14}$$

  to add new weak learner to the model ;

GB has the advantage that any differentiable loss function can be used. This even allows for incorporating non-convex loss functions, which have been shown to be more robust to label noise [113]. Algorithm 2 summarizes the training procedure of *GB* for the regression case. For more details we refer the interested reader to [75, 148] and the references therein. In Chapter 3, we show how to combine the ideas and benefits of *GB*

and *RF* (described in the following) into a common framework for both, classification and regression tasks.

2.2.4 Random Forests

Before we delve into the training and inference procedures of *RF*, we briefly recapitulate the history and the origination process of this popular *ML* algorithm, which is also nicely summarized in [34].

The fundamental base of *RF* are decision trees. The seminal book of Leo Breiman [24] on classification and regression trees (CART) initiated high interest in solving *ML* problems with trees. Ross Quinlan presented extensions to CART and introduced ID3 [132] and C4.5 [133]. During that time, researchers found how several weak learners can be combined into a stronger model. The work of Schapire [147] was one of the first and paved the way for many successful *Boosting* algorithms (already discussed above) and also for *RF*.

Amit and Geman soon discovered the effectiveness of combining randomized decision trees via ensemble methods for handwritten digit recognition [3, 4]. At the same time, Tin Kam Ho independently presented another form of combining randomized decision trees for the task of handwritten digit recognition [78, 79]. Finally, Breiman [23] formally introduced the term 'random forests' in 2001, including several variants of the algorithm for both classification and regression and a thorough analysis. One can find a very pleasing text on the history of *RF* in the foreword of [34] written by Yali Amit and Donald Geman.

Since then on, *RF* became highly popular and an almost endless list of works relying on this *ML* technique was presented. *RF* find applications in many disciplines in the computer vision community, e.g., object detection [62, 110, 157, 195, 197], semantic segmentation [26, 88, 89, 141, 164], pose estimation [38, 49, 50, 67, 163], tracking [70, 155], image denoising [51], etc. *RF* also led to one of the biggest success stories in computer vision, namely, the human pose estimation system implemented in the Microsoft® Kinect™ sensor based on randomized trees [163]. An excellent survey on *RF* including many examples of successful applications in the computer vision community can be found in [34].

In the following, we describe the technical details of *RF*. We start with the structure of the trees in Section 2.2.4.1. Then, we review the training procedure in Section 2.2.4.2 and describe the inference process in Section 2.2.4.3. Finally, we discuss some properties of *RF* in Section 2.2.4.4, which concludes the chapter on preliminaries.

2.2.4.1 The Model Structure of Random Forests

A *RF* consists of a set of $\mathtt{T}$ binary trees $\mathcal{T}_\mathtt{t}$, where each of them is typically trained independently from each other and their predictions are averaged. A binary tree $\mathcal{T}$ is a graph where each node $\mathtt{j}$ has a single parent node $\mathrm{Par}(\mathtt{j})$ and exactly two child nodes, a left one, $\mathrm{Child}_\mathrm{L}(\mathtt{j})$, and a right one, $\mathrm{Child}_\mathrm{R}(\mathtt{j})$. Exceptions are the root node, which has no parent, and all leaf nodes, which do not have any children. The leaf nodes in the

tree are also called terminal nodes in the literature. All remaining nodes are often called internal or splitting nodes.

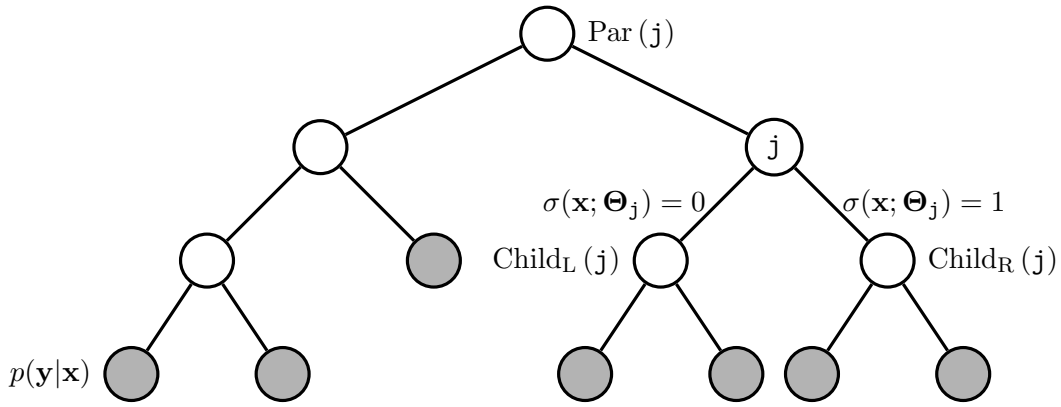


Figure 2.2: This figure gives an example of the structure of a binary tree $\mathcal{T}_\mathtt{t}$ in a random forest model. For a node $\mathtt{j}$, we illustrate the parent and two children alongside with the corresponding splitting function $\sigma(\cdot;\boldsymbol{\Theta}_\mathtt{j})$. Leaf nodes are highlighted in gray.

Each internal node $\mathtt{j}$ stores a splitting function $\sigma(\mathbf{x};\boldsymbol{\Theta}_\mathtt{j})$ that routes a data sample $\mathbf{x}$ to the left ($\sigma(\mathbf{x};\boldsymbol{\Theta}_\mathtt{j}) = 0$) or right subtree ($\sigma(\mathbf{x};\boldsymbol{\Theta}_\mathtt{j}) = 1$). The parameters $\boldsymbol{\Theta}_\mathtt{j}$ that define $\sigma(\cdot;\boldsymbol{\Theta}_\mathtt{j})$ have to be learned during the training phase that is described later. Thus, the collection of all splitting functions in the internal nodes defines the paths from the root node to a single leaf node. Formally, the process of routing sample $\mathbf{x}$ from any node $\mathtt{j}$ through the tree to a leaf node can be written as the recursive function

$$f(\mathbf{x},\mathtt{j}) = \begin{cases} \mathtt{j} & \text{if} & \mathtt{j} \text{ is a child} \\ f(\mathbf{x},\text{Child}_\text{L}(\mathtt{j})) & \text{else if} & \sigma(\mathbf{x};\boldsymbol{\Theta}_\mathtt{j}) = 0 \\ f(\mathbf{x},\text{Child}_\text{R}(\mathtt{j})) & \text{otherwise} & \end{cases} . \tag{2.15}$$

Each leaf node stores a prediction model $p(\mathbf{y}|\mathbf{x})$ for label $\mathbf{y}$ given data $\mathbf{x}$. The prediction model has to be defined for the task at hand, e.g., classification or regression. We will define both, the splitting functions and the prediction models, in the following section in more detail and give some examples for different applications. Figure 2.2 depicts an example of such a tree including splitting functions and prediction models.

We conclude this section with some properties and definitions of binary trees that will be used throughout this thesis. We can define the depth $\delta(\mathtt{j})$ of a node $\mathtt{j}$ as the length of the path from the node to the root of the tree. The path includes the node $\mathtt{j}$ itself. Thus, the root node has depth $\delta(\mathtt{j} = \text{root}) = 1$ and any direct child of the root has depth 2. The depth of the whole tree is defined as the maximum depth of any node, i.e., $\delta(\mathcal{T}) = \max_{\mathtt{j}\in\mathcal{T}} \delta(\mathtt{j})$. An important hyper-parameter (i.e., defined by the user) is the maximum tree depth $\mathtt{D}_\text{max}$, which constrains the tree $\mathcal{T}$ via $\delta(\mathcal{T}) \leq \mathtt{D}_\text{max}$. We also define a *fully-balanced* tree as a tree where all leaf nodes have the same depth $\delta(\cdot)$. Assuming a fully balanced tree of depth $\delta(\mathcal{T}) = \mathtt{D}_\text{max}$, the number of total nodes is $2^{\mathtt{D}_\text{max}} - 1$.

Then, the number of leaf nodes is $2^{\mathtt{D}_{\max}-1}$ and, hence, the number of internal nodes is $2^{\mathtt{D}_{\max}} - 1 - 2^{\mathtt{D}_{\max}-1} = 2^{\mathtt{D}_{\max}-1} - 1$. For this special case of a fully-balanced tree, the number of nodes doubles with each level of depth. Finally, we define the cardinality $\mathrm{card}\,(\mathtt{j})$ of a node $\mathtt{j}$ as the number of samples passing this node during the training time of the tree. For instance, the root node of tree $\mathtt{t}$ obviously has cardinality $\mathrm{card}\,(\mathtt{j} = \mathrm{root}) = \mathtt{N}_\mathtt{t}$, where $\mathtt{N}_\mathtt{t}$ is the number of training examples for this particular tree.

2.2.4.2 Training Random Forests

While *RF* is an ensemble method, the training procedure is quite different compared to the *Boosting* algorithm described earlier. Instead of training the weak learners (randomized trees in the case of *RF*) iteratively with the goal to correct mistakes of the current ensemble, *RF* treat each weak learner independently. This allows for training (and testing) the trees in parallel, which is a major benefit over *Boosting* regarding computational costs. Moreover, this training procedure yields decorrelated trees which is essential for the generalization performance of *RF*. For fully correlated or even identical trees, the ensemble cannot improve the prediction over a single weak learner, as we will also see later. The independence of the trees also eases the description of *RF* as we can reduce it to the training procedure of a single tree $\mathcal{T}_\mathtt{t} \in \mathcal{F}$, where $\mathtt{t} = 1, \ldots, \mathtt{T}$. In the following, we present a formulation for both classification and regression tasks.

Given is a set of training data $\mathbf{X} = \{\mathbf{x}_\mathtt{n}, \mathbf{y}_\mathtt{n}\}_{\mathtt{n}=1}^{\mathtt{N}}$, where $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$. The data space $\mathcal{X}$ can almost always be written as a $\mathtt{D}$-dimensional feature vector ($\mathcal{X} = \mathbb{R}^\mathtt{D}$). While for some cases, e.g., computer vision applications, a different form might be more convenient to use, here, we stick to the vectorized form. The feature vector is typically a representation computed from some other entity like a text, an audio sequence, or an image, as already mentioned before. The goal is to build a tree that represents a mapping from $\mathcal{X}$ to $\mathcal{Y}$ such that some error is minimized and good prediction about $\mathcal{Y}$ can be done. In the following, we first describe the training of the tree structure and then show how the leaf node models are computed.

Training the Tree Structure Training the tree structure involves recursively splitting the data into disjoint sets by finding splitting functions

$$\sigma(\mathbf{x}; \boldsymbol{\Theta}_\mathtt{j}) = \begin{cases} 0 & \text{if } \xi(\mathbf{x}; \boldsymbol{\Theta}_\mathtt{j}) < 0 \\ 1 & \text{otherwise} \end{cases}, \tag{2.16}$$

for all internal nodes $\mathtt{j}$, where $\xi(\cdot; \boldsymbol{\Theta})$ is the data response function and $\boldsymbol{\Theta}_\mathtt{j}$ are the parameters that define it. These parameters have to be learned. Each of these splitting functions separate the data in node $\mathtt{j}$ into two disjoint sets with the goal that the labels in these sets become more compact and better predictions can be made. In the following, we drop the subscript $\mathtt{j}$ that indicates the node index wherever it is clear from the context for a better readability.

The response function $\xi(\cdot;\cdot)$ totally depends on the form of the data space $\mathcal{X}$ for the particular application, e.g., unstructured machine learning data or structured image or text data. Here, we only give a few typically employed examples of response functions. Later chapters define specifically designed response functions. Given a standard feature vector $\mathbf{x} \in \mathcal{X} = \mathbb{R}^{\mathtt{D}}$, the most popular choice is to apply a thresholding operation on a single feature dimension. This is defined as

$$\xi_{\text{single}}(\mathbf{x};\boldsymbol{\Theta}) = \mathbf{x}[\boldsymbol{\Theta}_1] - \Theta_{\text{th}}\,, \tag{2.17}$$

where $\mathbf{x}[\boldsymbol{\Theta}_1]$ returns the value of $\mathbf{x}$ at dimension $\boldsymbol{\Theta}_1 \in \{1,\ldots,\mathtt{D}\} \subset \mathbb{Z}$, and $\Theta_{\text{th}} \in \mathbb{R}$ is a threshold. Another option that is also often employed for image data is a thresholding function on the difference of two dimensions. We thus define a 'pair-difference' response function as

$$\xi_{\text{pair}}(\mathbf{x};\boldsymbol{\Theta}) = \mathbf{x}[\boldsymbol{\Theta}_1] - \mathbf{x}[\boldsymbol{\Theta}_2] - \Theta_{\text{th}}\,, \tag{2.18}$$

where we additionally have $\boldsymbol{\Theta}_2 \in \{1,\ldots,\mathtt{D}\} \subset \mathbb{Z}$ and $\boldsymbol{\Theta}_1 \neq \boldsymbol{\Theta}_2$.

The typical procedure for finding good parameters $\boldsymbol{\Theta}$ for the splitting function $\sigma(\cdot;\boldsymbol{\Theta})$ is to sample a random set of parameter values $\boldsymbol{\Theta}_k$ and choosing the best one $\boldsymbol{\Theta}^*$ according to a quality measure. The parameters $\boldsymbol{\Theta}^*$ are thus found via a randomized grid search on the parameter space. In general, the quality measure for a splitting function $\sigma(\mathbf{x};\boldsymbol{\Theta})$ has the form

$$Q(\sigma(\cdot;\boldsymbol{\Theta}),\mathbf{X}) = E(\mathbf{X}) - \sum_{c\in\{Le,Ri\}} |\mathbf{X}^c| \cdot E(\mathbf{X}^c)\,, \tag{2.19}$$

where Le and Ri define the left and right child nodes, and $|\cdot|$ is the cardinality operator. We define $\mathbf{X}^{Le} = \{\mathbf{x} \in \mathbf{X} : \sigma(\mathbf{x};\boldsymbol{\Theta}) = 0\}$, $\mathbf{X}^{Re} = \{\mathbf{x} \in \mathbf{X} : \sigma(\mathbf{x};\boldsymbol{\Theta}) = 1\}$. Please note the implicit dependency of the second term in Equation (2.19) on the splitting function $\sigma(\cdot;\boldsymbol{\Theta})$. Also note that the first term in (2.19) is independent from $\sigma(\cdot;\boldsymbol{\Theta})$. The function $E(\mathbf{X})$ aims at measuring the compactness or the purity of the given data $\mathbf{X}$. In the following, we thus denote this function as compactness measure. The intuition is to have similar data samples falling into the same leaf nodes, thus, giving coherent predictions. Equation (2.19) resembles the information gain if $E(\mathbf{X})$ is the Shannon entropy. However, for finding $\boldsymbol{\Theta}^*$, maximizing (2.19) is equivalent to minimizing the cost function

$$C(\sigma(\cdot;\boldsymbol{\Theta}),\mathbf{X}) = \sum_{c\in\{Le,Ri\}} |\mathbf{X}^c| \cdot E(\mathbf{X}^c)\,, \tag{2.20}$$

as the first term in (2.19) is independent from $\boldsymbol{\Theta}$ and can be dropped.

The compactness measure $E(\mathbf{X})$ (and thus also $Q(\cdot)$ and $C(\cdot)$) typically operates only on the label space $\mathcal{Y}$, e.g., for classification or regression tasks. However, one can also imagine to include the data space into $E(\cdot)$ for clustering tasks or as an additional regression (see Chapter 6). As with the response function, modeling the compactness measure is task-dependent. Here, we review the basic measures for the classification and regression

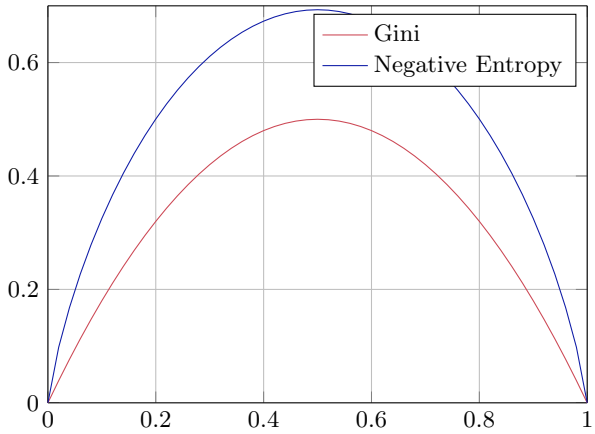


Figure 2.3: The figure illustrates the shape of two different compactness measures used to evaluate the quality of splitting functions in RF for the classification task. We illustrate the negative entropy, $E_{\text{Entr}}(\mathbf{X})$, and the Gini index, $E_{\text{Gini}}(\mathbf{X})$, for a 2-class problem.

tasks. The two most employed measures for classification tasks are the entropy

$$E_{\text{Entr}}(\mathbf{X}) = -\sum_{\mathtt{c}=1}^{\mathtt{C}} p(\mathbf{y} = \mathtt{c}|\mathbf{X}) \cdot \log(p(\mathbf{y} = \mathtt{c}|\mathbf{X}))\,, \tag{2.21}$$

where $\mathtt{C}$ is the number of classes, and the Gini index

$$E_{\text{Gini}}(\mathbf{X}) = \sum_{\mathtt{c}=1}^{\mathtt{C}} p(\mathbf{y} = \mathtt{c}|\mathbf{X}) \cdot (1 - p(\mathbf{y} = \mathtt{c}|\mathbf{X})) = 1 - \sum_{\mathtt{c}=1}^{\mathtt{C}} p(\mathbf{y} = \mathtt{c}|\mathbf{X})^2\,. \tag{2.22}$$

For both cases, the conditioned class probabilities are defined as

$$p(\mathbf{y} = \mathtt{c}|\mathbf{X}) = \frac{\sum_{\mathtt{n}=1}^{|\mathbf{X}|} \mathbb{I}\left[\mathbf{y}_\mathtt{n} = c\right]}{|\mathbf{X}|}\,, \tag{2.23}$$

where $\mathbb{I}\left[\cdot\right]$ is the indicator function. The shapes of the entropy and Gini-based measures are illustrated in Figure 2.3 for 2 classes. The Gini index is an approximation of the entropy that avoids computing the logarithm. For regression, many different measures can be defined. One of the most basic ones is the 'reduction in variance'

$$E(\mathbf{X}) = \sum_{\mathtt{n}=1}^{|\mathbf{X}|} \|\mathbf{y}_\mathtt{n} - \bar{\mathbf{y}}\|^2\,, \tag{2.24}$$

where

$$\bar{\mathbf{y}} = \frac{1}{|\mathbf{X}|} \sum_{\mathtt{n}=1}^{|\mathbf{X}|} \mathbf{y}_\mathtt{n} \tag{2.25}$$

is the empirical mean of the labels in $\mathbf{X}$. Another option is the differential entropy, i.e.,

the continuous pendant to the discrete Shannon entropy,

$$E(\mathbf{X}) = -\int_{\mathcal{Y}} p(\mathbf{y}|\mathbf{X}) \cdot \log\left(p(\mathbf{y}|\mathbf{X})\right) \mathrm{d}\mathbf{y} \ . \tag{2.26}$$

For classification, the probability density $p(\mathbf{y}|\mathbf{X})$ can be easily computed as the empirical distribution of labels in $\mathbf{X}$. For regression, parametric but also non-parametric models exist [120] to compute (2.26). For the case of a Gaussian model, i.e., $p(\mathbf{y}|\mathbf{X}) = \mathcal{N}(\mu(\mathbf{X}), \sigma^2(\mathbf{X}))$ with mean $\mu(\mathbf{X})$ and variance $\sigma^2(\mathbf{X})$ of the labels in $\mathbf{X}$, the integral can be solved in closed-form. This results in

$$E(\mathbf{X}) = \frac{\mathtt{K}}{2}(1 - \log(2\pi)) + \frac{1}{2}\log\left(\det\left(\Sigma_{\mathbf{X}}\right)\right) \tag{2.27}$$

making the computation tractable. Algorithm 3 summarizes the process of finding a splitting function via randomized grid search.

Algorithm 3: Finding a splitting function $\sigma(\cdot;\boldsymbol{\Theta})$ via randomized grid search in *RF*.

input : Labeled training set $\mathbf{X} = \{\mathbf{x}_\mathtt{n}, \mathbf{y}_\mathtt{n}\}_{\mathtt{n}=1}^{\mathtt{N}}$
input : Number k of splitting functions to sample
output: Parameters $\boldsymbol{\Theta}^*$ for a splitting function $\sigma(\cdot;\boldsymbol{\Theta})$
Initialize $C^* = \infty$;
for k_i *from* 1 *to* k **do**
  Randomly sample split function parameters $\boldsymbol{\Theta}_{k_i}$;
  Compute the splitting costs C_{k_i} of the sampled split parameters $\boldsymbol{\Theta}_{k_i}$ via Equation (2.20) ;
  if $C_{k_i} < C^*$ **then**
    Set $C^* = C_{k_i}$;
    Set $\boldsymbol{\Theta}^* = \boldsymbol{\Theta}_{k_i}$;

When the randomized grid search found good parameters $\boldsymbol{\Theta}^*$ for the splitting function, we fix them, split the data into two halves, and send it to the two newly created leaf nodes. Tree growing continues by finding splitting functions for these newly created leaf nodes. There exist different schemes of selecting the next node for further splitting in the tree. The two most popular ones are 'depth-first' and 'breadth-first', which are illustrated in Figure 2.4. The depth-first variant can be seen as a recursive function that splits nodes until a stopping criterion is reached and then continues with the next subtree. The breadth-first variant finds splitting functions for all nodes with the same depth before continuing with nodes deeper in the tree. Other options also exist where nodes are selected based on the current training error or training loss in decreasing order [90]. All these procedures start with the root node of a tree and continue in a greedy manner down the tree until one of the defined stopping criteria is reached.

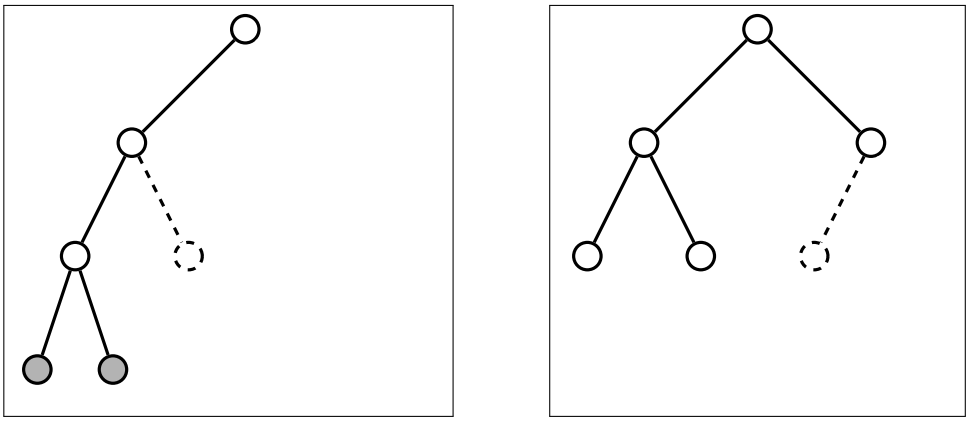


Figure 2.4: This figure illustrates two different tree growing schemes, (a) depth- and (b) breadth-first. For both cases, the tree growing process is in iteration 6, i.e., the sixth node (marked dashed) is under consideration for splitting. While already two final leaf nodes exist in (a) because some stopping criteria was already met ($\mathtt{D}_{\max} = 4$), there are no final leafs created yet in (b).

The stopping criteria for tree growing are typically the following: (i) a node reaches a maximum tree depth $\mathtt{D}_{\max}$; (ii) the cardinality of a node $\mathrm{card}\,(\mathtt{j})$ becomes smaller than some pre-defined threshold $\mathtt{N}_{\min}$; (iii) the set of labels $\mathbf{X}_{\mathcal{Y}} = \{\mathbf{y}_\mathtt{n}\}_{\mathtt{n}=1}^{|\mathbf{X}_{\mathcal{Y}}|}$ in node $\mathtt{j}$ becomes pure. While this is easy to define for classification tasks (all samples are from the same class), one would have to define a specific measure for regression tasks. In any of these cases, tree growing stops and a leaf node is instantiated. The whole tree growing process is summarized in Algorithm 4.

Algorithm 4: The tree growing process in *RF*.

input : Labeled training set $\mathbf{X} = \{\mathbf{x}_\mathtt{n}, \mathbf{y}_\mathtt{n}\}_{\mathtt{n}=1}^{\mathtt{N}}$
input : Parameters for stopping criteria ($\mathtt{D}_{\max}$ and $\mathtt{N}_{\min}$)
output: A trained tree $\mathcal{T}$
Initialize tree $\mathcal{T}$ with a single root node $\mathtt{j}$ = root ;
while *Nodes* $\mathtt{j}$ *are left for splitting* **do**
  Select a node $\mathtt{j}$ to be considered for splitting (depends on depth- or breadth-first scheme) ;
  if $\delta\,(\mathtt{j}) \geq \mathtt{D}_{max}$ *OR* $card\,(\mathtt{j}) < \mathtt{N}_{min}$ **then**
    Create a leaf node and prediction model $p(\mathbf{y}|\mathbf{x})$;
  else
    Create a splitting node and find parameters $\boldsymbol{\Theta}^*$ for the splitting function $\sigma(\cdot;\boldsymbol{\Theta}^*)$) via Algorithm 3 ;
    Split data of current node $\mathbf{X}_\mathtt{j}$ according to $\sigma(\cdot;\boldsymbol{\Theta}^*)$ into $\mathbf{X}_\mathtt{j}^{Le}$ and $\mathbf{X}_\mathtt{j}^{Re}$;
    Create two children and assign the corresponding data samples.

Prediction Models in the Leaf Nodes As soon as the tree structure is fixed, i.e., tree growing stopped, we are left with a set of nodes that all fulfill any of the stopping criteria. These are the leaf nodes and we have to compute a prediction model with the data falling into these nodes.

We now describe how to compute a prediction model for a leaf node $\mathtt{j}$ given a set of data $\mathbf{X}_\mathtt{j} \subseteq \mathbf{X}$ (in the rare and unreasonable case that the root node is the single leaf node, $\mathbf{X}_\mathtt{j} = \mathbf{X}$). The goal is to find a function predicting the correct label $\mathbf{y}$ for a given $\mathbf{x}$. The leaf model has to be simple in order to keep the computational costs low and to avoid over-fitting the data. Thus, constant models are often used. Sometimes, also linear models are employed (see [51] or Chapter 6).

Different tasks, e.g., classification or regression, obviously require different prediction models. For classification, the standard constant model is a multinomial distribution with support given by the class histogram of $\mathbf{X}_\mathtt{j}$, i.e.,

$$p(\mathbf{y} = \mathtt{c}|\mathbf{x}) = \frac{1}{|\mathbf{X}_\mathtt{j}|} \sum_{\mathtt{n}=1}^{|\mathbf{X}_\mathtt{j}|} \mathbb{I}\left[\mathbf{y}_\mathtt{n} = \mathtt{c}\right] \ . \tag{2.28}$$

Also for regression one can employ a constant prediction model by computing the empirical mean $\bar{\mathbf{y}}$ of the continuous target variables $\mathbf{y}$, i.e.,

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{|\mathbf{X}_\mathtt{j}|} \sum_{\mathtt{n}=1}^{|\mathbf{X}_\mathtt{j}|} \mathbf{y}_\mathtt{n} \ . \tag{2.29}$$

One can also employ a parametric approach and fit a probability distribution like a Gaussian by computing mean and variance of the data. Another possibility is a non-parametric approach (e.g., [62, 121]). Yet another option is a linear regression model

$$p(\mathbf{y}|\mathbf{x}) = \mathbf{W} \cdot \mathbf{x} \ , \tag{2.30}$$

where $\mathbf{W}$ is the solution of a least squares problem. Such a linear prediction model in the leaf nodes has recently also been used for image denoising [51] and single image super-resolution (see Chapter 6).

2.2.4.3 Making Predictions with Random Forests

After the training phase of *RF*, the tree structure as well as the prediction models in the leaf nodes are fixed. Computing a prediction for a newly arriving data sample $\mathbf{x}$ involves evaluating each tree in the forest independently and averaging the results. As with the training of the trees, evaluating them can also be done in parallel, which again contributes to the computational efficiency of *RF*.

Nevertheless, the main contribution to computational efficiency is the fast tree evaluation process. Computing a prediction $p(\mathbf{y}|\mathbf{x})$ of sample $\mathbf{x}$ for a single tree $\mathcal{T}_\mathtt{t}$ only requires

to route the sample from the root node to one leaf node by evaluating the corresponding splitting functions $\sigma(\cdot;\cdot)$ along this path, which is defined by the recursive function (2.15). The data sample $\mathbf{x}$ thus arrives at a single leaf node in tree $\mathcal{T}_\mathtt{t}$ which stores a prediction $p_\mathtt{t}(\mathbf{y}|\mathbf{x})$. As already described before, this prediction model is mostly constant, but also other forms (like a linear prediction model) exist.

Having computed the prediction from each individual tree, i.e., $p_\mathtt{t}(\mathbf{y}|\mathbf{x})$, one can easily compute the final prediction via averaging as

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{\mathtt{T}} \sum_{\mathtt{t}=1}^{\mathtt{T}} p_\mathtt{t}(\mathbf{y}|\mathbf{x}) \ . \tag{2.31}$$

Please note that this operation requires the definition of summation and division on the label space $\mathcal{Y}$. There can be special instances of $\mathcal{Y}$ that do not fulfill these requirements, especially for structured prediction tasks like semantic segmentation [89] or contour detection [43]. For these cases, only a similarity is defined on instances of $\mathcal{Y}$. While one solution to make predictions is to relax $\mathcal{Y}$ in order to fulfill the above mentioned requirements, another one could be to select only a single instance from all predictions of the tree based on the similarity measure, cf. [43].

The computational costs of inference in a single tree are only logarithmic in the number of nodes of the tree (assuming a balanced tree). In particular, routing a single sample $\mathbf{x}$ to the corresponding leaf node costs at most $\mathtt{D}_{\max} - 1$ times the costs of evaluating a splitting function $\sigma(\cdot;\cdot)$. The computational costs of the full prediction also includes the costs for evaluating the leaf nodes, which is often a constant factor.

2.2.4.4 Discussion and Properties of Random Forests

In this last section on *RF*, we briefly summarize some properties of this learning algorithm. We start with the basic benefit (among others) that makes *RF* a popular choice for many applications in the *ML* and computer vision communities: the efficient training and inference procedures. The efficiency stems from the independent training and evaluation of the trees, where parallel computing on multiple CPU cores can be easily exploited. Also the randomized grid search used for finding splitting functions in the training phase of *RF* (see Section 2.2.4.2) is efficient. Tuning the parameters of this grid search (e.g., the number of splitting functions to be evaluated) has a direct influence on the training time but often little effect on the accuracy of the resulting prediction model. As the diversity of the trees in *RF* is required for a good generalization, inducing a larger factor of randomization via a smaller search space in the grid search (and thus faster training) can even be beneficial [155]. Moreover, the tree structure of the model allows for a quick prediction of a new data sample $\mathbf{x}$. Only a small fraction of the learned splitting functions (those that define the way from the root to the leaf node) have to be evaluated. Evaluation of a splitting function is always conditioned on the outcome of previous splitting functions (except for the root node obviously). When assuming a balanced tree, one only needs to

compute as many splitting functions as levels of depth in the tree. Thus, the costs for accessing the prediction model in *RF* is only logarithmic in the number of nodes of a tree. Obviously, the leaf node prediction model itself also has to be evaluated. In most cases, though, this is a simply constant or linear (cf., Chapter 6) model.

Another interesting property of *RF* is the easy and effective handling of multi-class problems. The training algorithm separates the data space and learns local prediction models in all leaf nodes, which correspond to multinomial probability distributions. Thus, computing the probability of a test sample $\mathbf{x}$ for a certain class $\mathtt{c}$ only requires routing $\mathbf{x}$ to the corresponding leaf nodes and a look-up in the distribution. This stands in contrast to many other *ML* algorithms (*Boosting*, *SVM*) that either need a separate and often complicated formulation for the multi-class case [33, 207] or have to rely on heuristics like one-vs-one or one-vs-all. A notable exception is *NN*, which also builds upon a natural multi-class formulation. Nevertheless, all of the above approaches except *RF* have a linear dependence on the number of classes $\mathtt{C}$ regarding the inference costs.

As already mentioned in the beginning of this chapter, *generalizing* to unseen data is the ultimate goal of an *ML* algorithm. We can generally define the generalization error ϵ_{G} as the expectation of misclassification

$$\epsilon_{\mathrm{G}} = \mathbf{E}_{\mathcal{X}\times\mathcal{Y}}\left[\mathbb{I}\left[\max_{\hat{\mathbf{y}}} p(\hat{\mathbf{y}}|\mathbf{x}) \neq \mathbf{y}\right]\right] \ . \tag{2.32}$$

Interestingly, *RF* do not over-fit on the training data with an increasing number of trees (and thus a more complex model) but saturate at some point. Breiman provides a proof based on the strong law of large numbers in [23]. However, in practice one can observe over-fitting either due to too deep trees or high correlation between the prediction of the trees. While the tree depth is a hyper-parameter set by the user (typically based on experience and/or properties of the task at hand and the given training data), correlation between trees is a property of the randomized grid search and the training of *RF* (see Section 2.2.4.2). Breiman [23] also gives an upper bound on the generalization error as

$$\epsilon_{\mathrm{G}} \leq \bar{\rho} \cdot \frac{1 - s^2}{s^2} \ , \tag{2.33}$$

where s is the strength of the trees and $\bar{\rho}$ the average correlation between them. Defining these two quantities requires the margin function of *RF*

$$m_{\mathrm{rf}}(\mathbf{x}, \mathbf{y}) = \mathbf{E}_{\boldsymbol{\Theta}_{\mathrm{tree}}}\left[p(\mathbf{y}|\mathbf{x}, \boldsymbol{\Theta}_{\mathrm{tree}})\right] - \max_{\hat{\mathbf{y}}\neq\mathbf{y}} \mathbf{E}_{\boldsymbol{\Theta}_{\mathrm{tree}}}\left[p(\hat{\mathbf{y}}|\mathbf{x}, \boldsymbol{\Theta}_{\mathrm{tree}})\right] \ , \tag{2.34}$$

where $\boldsymbol{\Theta}_{\mathrm{tree}}$ is a random variable defining all parameters of a tree (splitting functions, etc.). Then, the strength and the average correlation can be defined as

$$s = \mathbf{E}_{\mathcal{X}\times\mathcal{Y}}\left[m_{\mathrm{rf}}(\mathbf{x}, \mathbf{y})\right] \tag{2.35}$$

and

$$\bar{\rho} = \frac{\mathbf{E}_{\boldsymbol{\Theta}_{\text{tree}} \times \boldsymbol{\Theta}'_{\text{tree}}} \left[\rho(\boldsymbol{\Theta}_{\text{tree}}, \boldsymbol{\Theta}'_{\text{tree}}) \cdot \text{std}(\boldsymbol{\Theta}_{\text{tree}}) \cdot \text{std}(\boldsymbol{\Theta}'_{\text{tree}}) \right]}{\mathbf{E}_{\boldsymbol{\Theta}_{\text{tree}} \times \boldsymbol{\Theta}'_{\text{tree}}} \left[\text{std}(\boldsymbol{\Theta}_{\text{tree}}) \cdot \text{std}(\boldsymbol{\Theta}'_{\text{tree}}) \right]} \,, \tag{2.36}$$

respectively, where $\text{std}(\cdot)$ is the standard deviation [23]. While an exact derivation of the upper bound ϵ_{G} in Equation (2.33) can be found in [23], the basic intuition behind it is the following: Minimizing the generalization error requires strong but uncorrelated trees.

The good generalization property of *RF* can also be argued from another point of view, namely the bias-variance dilemma already discussed in Section 2.1.1. We already discussed and explained the beneficial properties on the generalization performance of averaging weaker models for bagging in Section 2.2.1. The consensus of individual models with low bias but high variance can reduce the variance while keeping also the bias low. If the individual models are independent, the averaging typically pays off most. While bagging only achieved different individual models by using slightly different training data, *RF* additionally introduce randomization into the training process leading to more uncorrelated trees.

2.3 Summary

In Section 2.1 of this chapter, we first introduced important *ML* concepts that are required throughout the thesis. We described supervised learning for different tasks like classification or regression and define the basic tools like a loss function, the data space and the label space. We also briefly outlined other forms of *ML* that work with less supervision, e.g., semi-supervised learning.

We then recapitulated ensemble methods in Section 2.2, which builds the basis for the main contribution of this thesis as presented in the next chapter. Ensemble methods are a special case of *ML* algorithms that combine several weak learners into a single strong model. *Boosting* is a popular representative and is described in Section 2.2.2. While *Boosting* is limited in the loss function that is optimized during the training phase, *GB* can handle any differentiable loss. This learning algorithm interprets *Boosting* as functional gradient descent. Another ensemble method we need for our novel training algorithm, which is presented in the next chapter, is *RF*. This algorithm uses randomized trees as weak learners that are trained independently from each other, in contrast to *Boosting* approaches. In Section 2.2.4 we describe this learning algorithm and provide the required details for later chapters.

CHAPTER 3

Alternating Decision and Regression Forests

In the previous chapter we introduced Random Forests (RF) as a very flexible Machine Learning (ML) technique for different tasks like classification or regression. We also presented several computer vision applications that successfully employ *RF* and show its importance in this field. In this chapter, we propose a novel training scheme for *RF* that is able to optimize a global loss function over all trees, instead of training them independently. We denote this algorithm Alternating Decision and Regression Forests (ADRF), which was originally presented in [156] and [158]. After having motivated the novel training scheme in Section 3.1, we describe *ADRF* in detail and discuss its properties as well as related work in Sections 3.2 and 3.3. Finally, this chapter also contains a thorough evaluation of the proposed training scheme on several classification and regression benchmarks from the *ML* community (see Section 3.4). We also conduct several experiments to get a better understanding of the intrinsic properties of *ADRF*. Beside the *ML* benchmarks, *ADRF* can also be applied to different computer vision applications, which we demonstrate in the remaining chapters of this theses (Chapters 4 to 6). These applications include object detection, human head pose estimation from depth data, as well as single image super-resolution.

3.1 Introduction

Typical machine learning algorithms like logistic regression, Support Vector Machine (SVM), Boosting (Boosting), or Neural Networks (NN) optimize a well-defined objective function for the task at hand. As described in Chapter 2, the objective function measures how well the training examples are already predicted by the current model and typically also includes some form of regularization on the model complexity. Multi-class logistic

regression (or softmax regression), for instance, minimizes the negative log likelihood

$$\Theta^* = \arg\min_{\Theta} \frac{1}{\mathtt{N}} \sum_{\mathtt{n}=1}^{\mathtt{N}} \mathcal{L}(\mathbf{y}_\mathtt{n}, f_{\mathrm{LR}}) + \lambda \cdot \Gamma(\Theta) \ , \tag{3.1}$$

with

$$\mathcal{L}(\mathbf{y}_\mathtt{n}, f_{\mathrm{LR}}) = -\sum_{c=1}^{\mathtt{C}} [\mathbf{y}_\mathtt{n} = c] \cdot \log \left(f_{\mathrm{LR}}(c; \mathbf{x}_\mathtt{n}, \Theta) \right) \tag{3.2}$$

and

$$\Gamma(\Theta) = \sum_{c}^{\mathtt{C}} \|\Theta_c\|^2 \ . \tag{3.3}$$

The operator $[\cdot]$ in Equation (3.2) is the Iverson bracket, $\Theta_c \in \mathbb{R}^\mathtt{D}$ are the model parameters for class c, and the prediction of the logistic regression model for class c and data sample $\mathbf{x} \in \mathbb{R}^\mathtt{D}$ is defined as

$$f_{\mathrm{LR}}(c; \mathbf{x}, \Theta) = \frac{e^{\Theta_c^\top \mathbf{x}}}{\sum_{\hat{c}=1}^{\mathtt{C}} e^{\Theta_{\hat{c}}^\top \mathbf{x}}} \ . \tag{3.4}$$

Another example is the binary *SVM* that also optimizes Equation (3.1) to learn the model parameters. In contrast to logistic regression, though, the *SVM* employs the hinge loss

$$\mathcal{L}(\mathbf{y}_\mathtt{n}, f_{\mathrm{SVM}}) = \max \left[0, 1 - \mathbf{y}_\mathtt{n} \cdot f_{\mathrm{SVM}}(\mathbf{x}_\mathtt{n}, \Theta) \right] \ , \tag{3.5}$$

where $f_{\mathrm{SVM}} = \Theta^\top \mathbf{x}$, and $\mathbf{y} \in \{-1, +1\}$. *NN*, *Boosting*, and other learning algorithms have similar objective functions. Importantly, all of them include the prediction of the current state of the full model during the training phase in order to minimize a given loss.

For *RF* the training scheme is different. *RF* consist of a set of binary trees, where each of them is trained independently from each other. One benefit of independently training the trees is clearly the easy parallelization and thus efficient training on modern machines with many CPU cores. Independent training of the trees also leads to diversity in the ensemble, which is essential for the success of random forests [23]. As described in Section 2.2.4.4, independence between the trees and the amount of randomization induced in each single tree control the diversity. However, the right amount of diversity is typically unknown and depends on several, task-dependent factors including the choice of splitting functions, the type of bagging, etc. On one side of the spectrum, we find extremely randomized forests [65], where randomization in the splitting functions is relatively high. The formulations of Rota Bulò and Kontschieder [141] or Yao et al. [202] operate on the other side of this spectrum. These works incorporate *NN* or *SVM* as splitting functions, however, both include a form of regularization to keep randomization and, thus, diversity between the trees.

Still, while the trees in *RF* are trained independently, the predictions on new test data are made based on the combination of all trees, e.g., via averaging the results. Thus,

the prediction of the final model is not considered during training, which stands in clear contrast to the above mentioned learning algorithms. One obvious solution to incorporate a well-defined loss function into *RF* is to use *Boosting* to train all trees. As described in Section 2.2.2, *Boosting* can be seen as a meta-learning algorithm for an ensemble of learners, where each single (weak) learner is trained after the other and the objective function is updated intermediately. However, by doing so, *RF* lose their attractive property of parallel, and thus efficient, training.

In this chapter, we tackle this issue and present *ADRF* that optimizes a well-defined loss function over all trees in the ensemble, while keeping the computational benefits of *RF* (see Section 3.2). The weak learners, i.e., the trees, are not trained one after the other as in *Boosting*, but the optimization of the loss function is directly integrated into the tree growing process. The trees are trained in a breadth-first manner, enabling the ensemble to make reasonable predictions on the training data after growing each level of depth. This, in turn, allows for using a well-defined loss function that takes the predictions of all the trees into account, as in other learning algorithms. The proposed algorithm thus alternates between optimizing the loss function and growing the trees by one level of depth, which is inspired by Gradient Boosting (GB). We discuss the relation of *ADRF* and *GB* in Section 3.2.4. The incorporation of a global loss function over all the trees also bears the potential risk of inducing dependence between the trees and, thus, lowering their diversity. However, we empirically show in Sections 3.4.1.5 and 3.4.2.5 that actually the opposite is true for *ADRF*.

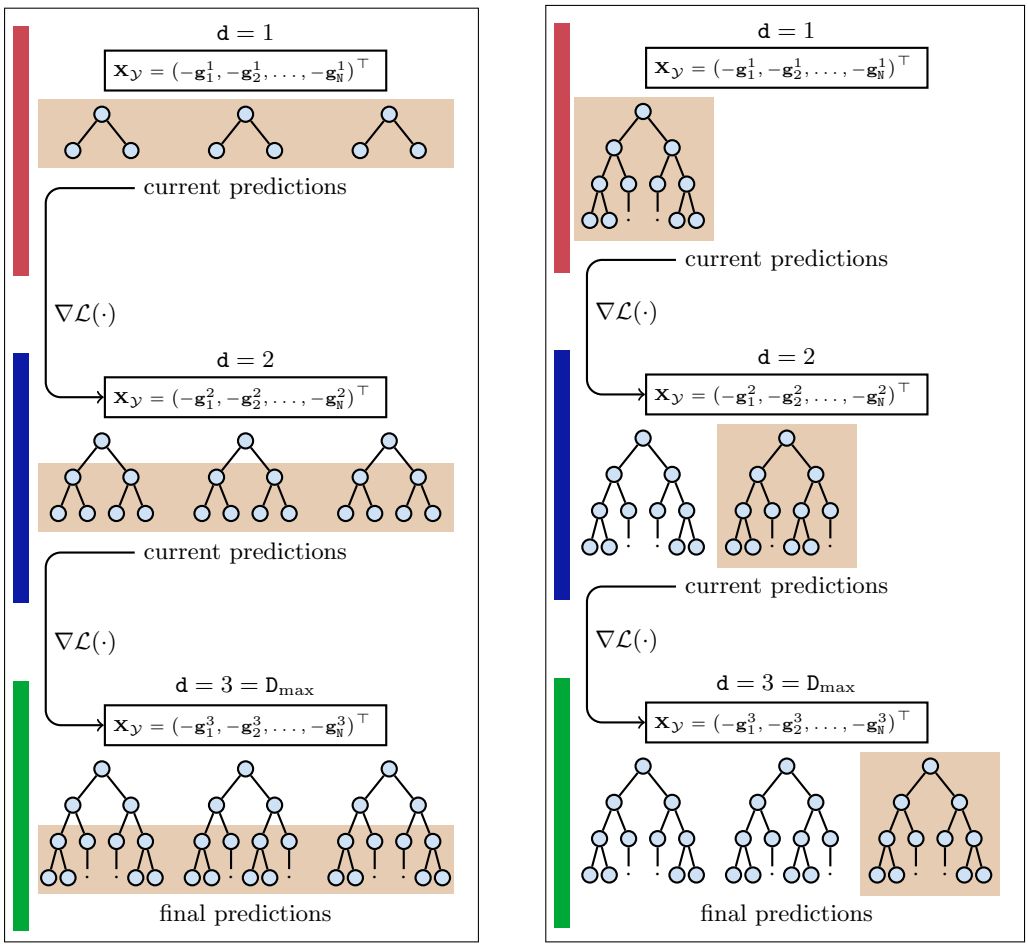
Also note that the presented approach only affects the training of *RF* and does not touch the inference, thus inheriting all the beneficial properties of *RF* during inference. *ADRF* can be applied for classification as well as regression tasks, which we describe in Sections 3.2.3 and 3.2.2, respectively. The novel training scheme can thus be easily employed in many applications relying on *RF*, which we will see in the later chapters of this thesis.

3.2 Alternating Decision and Regression Forests

We now present the main contribution of this thesis, which we originally proposed in [156] and [158]. After having described the general training scheme of *ADRF* in the following section, we present the specializations for regression and classification in Sections 3.2.2 and 3.2.3, respectively. Finally, in Section 3.2.4 we discuss the proposed learning scheme in more detail.

3.2.1 A Stage-Wise Random Forest Model

In order to optimize a global loss $\mathcal{L}(\cdot)$ over all trees, we could easily train boosted trees as mentioned before. However, the goal of *ADRF* is to explicitly integrate the global loss into the tree growing scheme, thus preserving the parallel training of standard *RF*. For



(a) Training scheme of *ADRF* **(b)** Training scheme of *GB*

Figure 3.1: Comparison between the training schemes of *ADRF* and *GB*. In both cases, (a) and (b), we illustrate the iterative training process vertically. Each stage is marked with a colored bar on the left (red, blue, and green). This example has only three training iterations, $d = 1, \ldots, 3 = \mathtt{D}_{\max}$. For *ADRF* (a), this corresponds to the level of depths that are trained. For *GB* (b), this corresponds to the number of trees in the model. In both cases, the light orange area indicates that part of the model that is currently learned. To end up with a model having the same complexity, we use three trees for *ADRF* and the same tree depth for *GB* as in *ADRF*. The difference between the training schemes is how the ensemble of trees is build up: (a) In *ADRF*, the gradient computation is directly integrated into the tree growing process. All trees are initialized with their root nodes. After finding splitting functions for the current level of depth, the gradients are updated and the trees continue to grow. (b) In *GB*, the trees are added sequentially into the model. Each iteration consists of gradient computation based on the existing set of trees in the model and full training of a new tree.

that purpose, we need a stage-wise tree growing scheme during which we can evaluate the loss $\mathcal{L}(\cdot)$ of the current model and update the training set correspondingly, akin to *GB* (see Section 2.2.3).

To get a stage-wise classifier, we let our trees $\mathcal{T}_\mathtt{t}$ grow in an iterative, breadth-first manner, contrary to a typical depth-first scheme in standard *RF*. We start with an initial model $\mathcal{F}^0(\mathbf{x})$, which consists of the $\mathtt{T}$ root nodes, and let the trees grow stage by stage. Each stage in *ADRF* corresponds to a single depth of the forest, i.e., all nodes $\mathtt{j}$ having the same depth $\delta(\mathtt{j}) = \mathtt{d}$. The iterations are thus indexed with $\mathtt{d} = 1, \ldots, \mathtt{D}_{\max}$, where $\mathtt{D}_{\max}$ is the maximum depth of the trees. Note that we can also use other stage-wise training schemes, e.g., a fixed number of nodes to be grown iteratively in a breadth-first manner. In any case, after training a single iteration $\mathtt{d}$, we always have a ready-to-use prediction model $\mathcal{F}^\mathtt{d}(\mathbf{x}_\mathtt{n}) = \frac{1}{\mathtt{T}} \sum_{\mathtt{t}=1}^{\mathtt{T}} p_\mathtt{t}^\mathtt{d}(\mathbf{y}_\mathtt{n}|\mathbf{x}_\mathtt{n})$, which is also true for $\mathcal{F}^0(\mathbf{x})$. Here, $p_\mathtt{t}^\mathtt{d}(\mathbf{y}_\mathtt{n}|\mathbf{x}_\mathtt{n})$ is the estimated label prediction of sample $\mathbf{x}_\mathtt{n}$ returned by tree $\mathcal{T}_\mathtt{t}^\mathtt{d}$, which denotes tree $\mathcal{T}_\mathtt{t}$ grown up to iteration $\mathtt{d}$.

We can now use this strong classifier $\mathcal{F}^\mathtt{d}(\mathbf{x})$ and a given loss function $\mathcal{L}(\cdot)$ to compute the negative gradients $-\mathbf{g}_{\mathtt{d}+1}(\mathbf{x}_\mathtt{n})$, cf., Equation (2.12), which are used for training the next stage $\mathtt{d}+1$. This is done by converting all leaf nodes in the current iteration into split nodes and finding appropriate parameters Θ for each of them by using the computed negative gradients. This process of *alternating* between training a single stage $\mathtt{d}$ of the forest and updating the negative gradients $-\mathbf{g}(\mathbf{x})$ for the next stage is repeated until the same stopping criteria as in standard *RF* are reached (see Section 2.2.4). Hence, we name this learning method *Alternating Decision and Regression Forests*. We give an illustrative overview of this scheme in Figure 3.1, which also includes a comparison to *GB*. We detail the algorithm for classification and regression in the following two sections, respectively.

Finally, we note that inference in *ADRF* is exactly the same as in *RF*, i.e., *ADRF* also inherits the properties of low computational costs during the testing phase. Thus, *ADRF* can incorporate well-defined losses into the training of *RF*, however, without violating their most important benefits and characteristics.

3.2.2 The Regression Case

For regression, the label space $\mathcal{Y}$ is continuous and $\mathtt{K}$-dimensional. As described in Sec. 2.2.4, the prediction in the leafs of *RF* can take different forms, e.g., constant, linear, etc. We can thus directly use the same formulation as in *GB* and compute the intermediate training data set $\mathbf{X}_{\mathbf{g}_\mathtt{d}} = \{\mathbf{x}_\mathtt{n}, -\mathbf{g}_\mathtt{d}(\mathbf{x}_\mathtt{n})\}_{\mathtt{n}=1}^{\mathtt{N}}$ for training stage $\mathtt{d}$ of the trees. We call the labels of this intermediate training set, i.e., $-\mathbf{g}_\mathtt{d}(\mathbf{x}_\mathtt{n})$, 'pseudo targets'. Figure 3.2 gives an illustration of the intermediate training set. Please note that newly created leaf nodes (after splitting) will give predictions about these 'pseudo targets' and already existing internal nodes also give predictions.

Unlike in Boosting (with trees or decision stumps as weak learners), where the path of any sample $\mathbf{x}$ is unknown beforehand, in *ADRF*, this path is always fixed as we have a hierarchical classifier structure. Thus, we can immediately add the target distribution $p_{\text{pa}}(\mathbf{y}|\mathbf{x})$ of the parent node, i.e., the new split node in depth $\mathtt{d}-1$, to the current target

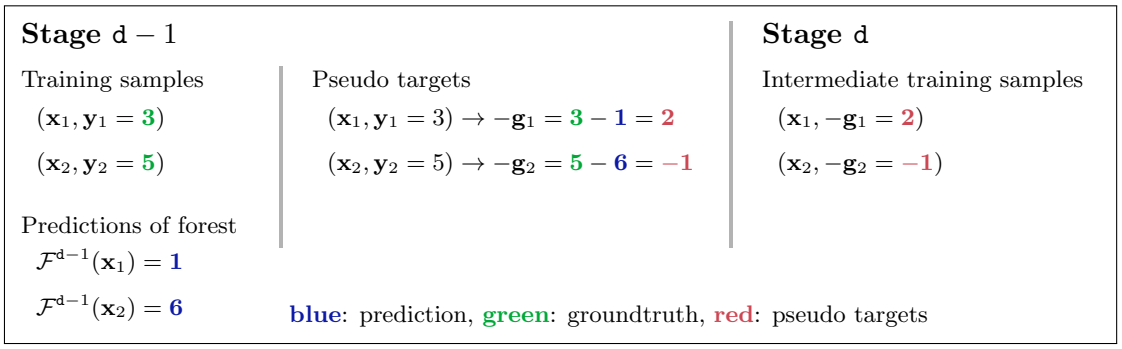


Figure 3.2: Exemplary illustration of the 'pseudo targets' in Alternating Regression Forests (ARF) used to train the intermediate stages. Here, we have a 1-dimensional regression problem and we inspect two training examples, $(\mathbf{x}_1, \mathbf{y}_1 = 3)$ and $(\mathbf{x}_2, \mathbf{y}_2 = 5)$. Green values indicate the ground truth values of these examples. In stage $\mathtt{d}-1$, the forest predicts 1 and 6 for these examples, respectively (indicated in blue). Assuming a squared loss, whose derivative is simply the difference between ground truth and prediction, we compute the 'pseudo targets' marked in red. This results in $-\mathbf{g}_1 = 2$ and $-\mathbf{g}_2 = -1$. In stage $\mathtt{d}$, these 'pseudo targets' are used in an intermediate training set to search for new splitting functions.

distributions $p_{\text{ch}}(\mathbf{y}|\mathbf{x})$ of the new child nodes in stage $\mathtt{d}$ as

$$p_{\text{ch}}(\mathbf{y}|\mathbf{x}) = \nu \cdot p_{\text{ch}}(\mathbf{y}|\mathbf{x}) + p_{\text{pa}}(\mathbf{y}|\mathbf{x}) \;, \tag{3.6}$$

where ν is again the shrinkage factor as already described for *GB* in Section 2.2.3. This update equation above can be seen as an equivalent to Equation (2.14) in *GB*. The shrinkage factor ν corresponds to a learning rate that weights the gradients $-\mathbf{g}$. Please note that we also could have created the intermediate training set as $\mathbf{X}_{\mathbf{g}_\mathtt{d}} = \{\mathbf{x}_\mathtt{n}, \nu \cdot -\mathbf{g}_\mathtt{d}(\mathbf{x}_\mathtt{n})\}_{\mathtt{n}=1}^{\mathtt{N}}$ instead of applying ν in Equation (3.6). While the shrinkage factor for *GB* is often set $\nu = 0.1$, for *ADRF* the default setting is $\nu = 1$. The reason is the small number of training iterations in *ADRF* compared to *GB*, which is limited by the maximum tree depth $\mathtt{D}_{\text{max}}$. In any case, after having updated the prediction in the new child nodes as defined above, we set $p_{\text{pa}} = 0$ and continue the training of *ADRF* with iteration $\mathtt{d}+1$. The intermediate classifier $\mathcal{F}^\mathtt{d}(\mathbf{x})$ can thus be used like a standard *RF* for making predictions, see Figure 3.3. We call this instance of our approach Alternating Regression Forests (ARF) and summarize the complete training procedure in Algorithm 5.

Typical loss functions for the regression case include the squared ($\mathcal{L}^{\text{R}}_{\ell_2}(\mathbf{r}) = \|\mathbf{r}\|^2$), the absolute ($\mathcal{L}^{\text{R}}_{\ell_1}(\mathbf{r}) = |\mathbf{r}|$), and the Huber loss (well known from robust statistics [75])

$$\mathcal{L}^{\text{R}}_{\text{Huber},\delta}(\mathbf{r}) = \begin{cases} \frac{1}{2}\mathbf{r}^2 & \text{for } |\mathbf{r}| \leq \delta \\ \delta(|\mathbf{r}| - \frac{1}{2}\delta) & \text{otherwise} \end{cases} \;, \tag{3.7}$$

where $\mathbf{r} = \mathbf{y} - \mathcal{F}(\mathbf{x})$ defines the residual. The squared loss is the most restrictive one, giving relatively high penalty to samples that are incorrectly regressed, compared to, e.g.,

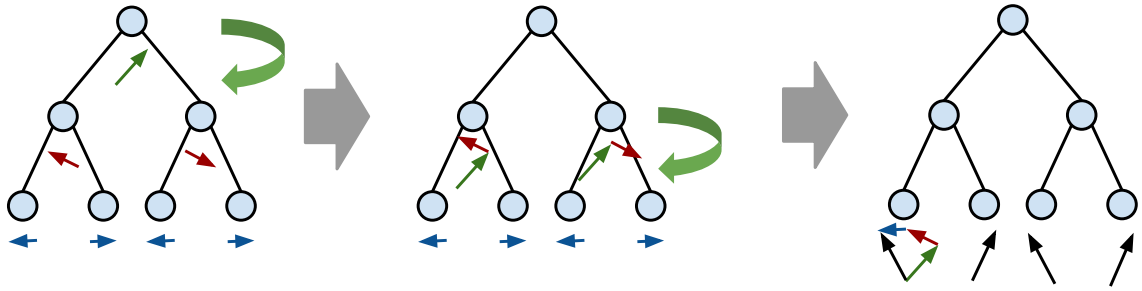


Figure 3.3: Transforming the stage-wise additive model used in *ADRF* to minimize a common loss function over all trees into a standard *RF* model. The original additive model stores predictions in each node, also in internal nodes. This is indicated with the two dimensional regression targets (arrows) below each node in the left part of the figure. As the path from the root to each single leaf node is fixed after training, the predictions can be propagated from the root node to the leaf nodes by adding them. This is done by adding the predictions from the previous level to the current ones until the leaf nodes are reached (left to right in the figure). Predictions from different levels are highlighted with different colors. The final predictions (right) are black.

the absolute loss. The Huber loss [75] can be adapted with the parameter δ, resulting in different shapes of the loss. It behaves like a squared loss for residuals below δ and like an absolute loss for residuals above δ. We visualize these loss functions for the one-dimensional regression case in Figure 3.4a. However, note that any differentiable function could be employed in theory.

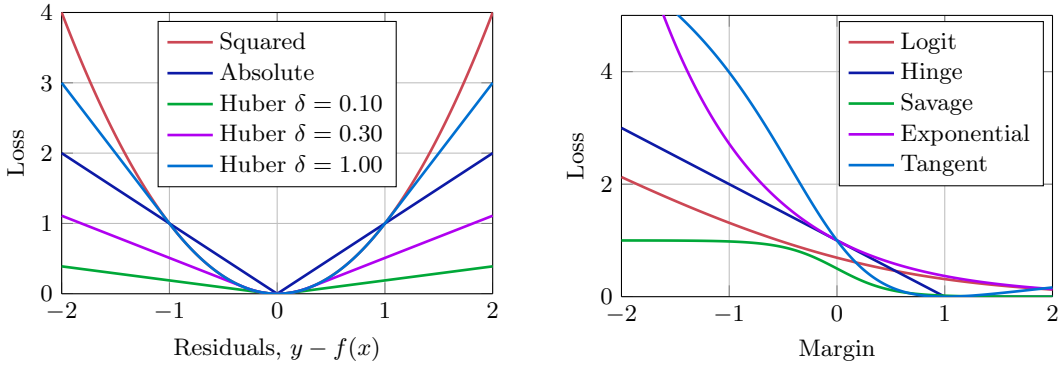
Algorithm 5: Training of *Alternating Regression Forests*

input : Labeled training set $\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^{N} \in \mathcal{X} \times \mathcal{Y}$
output: Trained forest $\mathcal{F}^{D_{max}}$

Init $\mathcal{F}^0$ as T root nodes with $p_t^0(\mathbf{y}|\mathbf{x})$;
for d *from* 1 *to* D_{max} **do**
  Check stopping criteria for all nodes in depth d ;
  Calculate predictions $\mathcal{F}^{d-1}(\mathbf{x}_n, \Theta)$ of all samples ;
  Calculate 'pseudo targets' $-\mathbf{g}_d(\mathbf{x}_n)$, Equation (2.12) ;
  Find splitting functions for stage d with $\mathbf{X}_{\mathbf{g}_d}$;
  Calculate $p_{\text{ch}}(\mathbf{y}|\mathbf{x})$ in all (intermediate) leaf nodes ;
  Add $p_{\text{pa}}(\mathbf{y}|\mathbf{x})$ to corresponding $p_{\text{ch}}(\mathbf{y}|\mathbf{x})$, Equation (3.6) ;
  Set $p_{\text{pa}}(\mathbf{y}|\mathbf{x}) = 0$;
  Set $\mathcal{F}^d(\mathbf{x}) = \mathcal{F}^{d-1}(\mathbf{x}) + f^d(\mathbf{x})$;

3.2.3 The Classification Case

For the classification case, where we term our method Alternating Decision Forests (ADF), one cannot directly use the gradients from (2.12) because of the discrete label space $\mathcal{Y}$. As already described in Sec. 2.2.3 and also employed by other boosting



(a) **(b)**

Figure 3.4: (a) Different loss functions for regression problems. The squared loss gives the highest penalty to data samples with high (absolute) residuals. The absolute loss has a linear relation between residuals and penalty and results in a constant gradient. The Huber loss can be adapted with the parameter δ and shows a squared form near 0 and becomes linear for larger (absolute) residuals. (b) Different loss functions for classification problems. The logit, hinge, and exponential loss are convex functions and thus give high penalty for misclassified data samples. On the other hand, the savage and tangent losses are non-convex and flatten with increasingly negative margin. This property makes them more robust to label noise than other loss functions.

approaches [58, 59, 147], we interpret the norm of the gradient as importance weight for each training sample, see Equation (2.13).

For *ADF*, we can also easily compute the weights $w_{\mathtt{n}}^{\mathtt{d}}$ with the classifier from the previous stage $\mathcal{F}^{\mathtt{d}-1}(\mathbf{x})$. However, to find splitting functions for all nodes in stage $\mathtt{d}$, we have to take these weights into account. To do so, we need to adapt the compactness measure $E(\mathbf{X})$, which is either the Shannon entropy (Equation (2.21)) or the Gini index (Equation (2.22)) for the classification case. For both cases, the weights need to be integrated into the computation of the class distributions $p(c|\mathbf{X}_\mathcal{Y})$ for the set $\mathbf{X}_\mathcal{Y}$. This can easily be done as

$$p(c|\mathbf{X}_\mathcal{Y}) = \frac{\sum_{\mathtt{n}=1}^{|\mathbf{X}_\mathcal{Y}|} \mathbb{I}\left[\mathbf{y}_\mathtt{n} = c\right] \cdot w_\mathtt{n}^\mathtt{d}}{\sum_{\mathtt{n}=1}^{|\mathbf{X}_\mathcal{Y}|} w_\mathtt{n}^\mathtt{d}} \,. \tag{3.8}$$

Here, $\mathbb{I}\left[\mathbf{y}_\mathtt{n} = c\right]$ is the indicator function returning 1 if the label $\mathbf{y}_\mathtt{n} \in \mathbf{X}_\mathcal{Y}$ is equal to c, and 0 otherwise. Note that we never create an intermediate training set for *ADF*, which means that adding up predictions from previous stages is not required as in *ARF*, cf., Equation (3.6). Thus, also the shrinkage factor ν is irrelevant for *ADF*.

Typical loss functions for the classification case include the log, the hinge, or the exponential loss. While these loss functions are typically less robust to noise, the Savage [113] and Tangent [112] losses define two robust, non-convex functions. We illustrate these choices in Figure 3.4b. Please note that all these loss functions are defined over the

classification margin

$$m(\mathbf{y}_{\mathrm{GT}}, \mathbf{x}) = \mathcal{F}(\mathbf{y}_{\mathrm{GT}}|\mathbf{x}) - \max_{\mathbf{y} \neq \mathbf{y}_{\mathrm{GT}}} \mathcal{F}(\mathbf{y}|\mathbf{x})\,, \tag{3.9}$$

which is the difference between the prediction of the model for the ground-truth label $\mathbf{y}_{\mathrm{GT}}$ and the highest prediction for any other class. Thus, the loss can be written as

$$\mathcal{L}(\mathbf{y}_{\mathrm{GT}}, \mathcal{F}(\mathbf{x})) = \mathcal{L}(m(\mathbf{y}_{\mathrm{GT}}, \mathbf{x})) \tag{3.10}$$

and the corresponding derivative with respect to the classifier output $\mathcal{F}(\mathbf{x})$ can be computed via the chain rule

$$\frac{\partial \mathcal{L}(\mathbf{y}_{\mathrm{GT}}, \mathcal{F}(\mathbf{x}))}{\partial \mathcal{F}(\mathbf{x})} = \frac{\partial \mathcal{L}(m(\mathbf{y}_{\mathrm{GT}}, \mathbf{x}))}{\partial m(\mathbf{y}_{\mathrm{GT}}, \mathbf{x})} \cdot \frac{\partial m(\mathbf{y}_{\mathrm{GT}}, \mathbf{x})}{\partial \mathcal{F}(\mathbf{x})}\,. \tag{3.11}$$

as described in [144] (Appendix A). Algorithm 6 summarizes the classification case.

Algorithm 6: Training of *Alternating Decision Forests*

input : Labeled training set $\{\mathbf{x}_\mathtt{n}, \mathbf{y}_\mathtt{n}\}_{\mathtt{n}=1}^{\mathtt{N}} \in \mathcal{X} \times \mathcal{Y}$
input : Maximum tree depth $\mathtt{D}_{\max}$
Init $\mathcal{F}^0$ as $\mathtt{T}$ root nodes with $p_\mathtt{t}^0(\mathbf{y}|\mathbf{x})$;
Init weights $w_\mathtt{n}^1 = \frac{1}{\mathtt{N}}$;
for $\mathtt{d}$ *from* 1 *to* $\mathtt{D}_{max}$ **do**
  Check stopping criteria for all nodes in depth $\mathtt{d}$;
  Split nodes in depth $\mathtt{d}$ by considering $w_\mathtt{n}^\mathtt{d}$, Equation (3.8) ;
  Update weights $w_\mathtt{n}^{\mathtt{d}+1}$, Equation (2.13) ;

3.2.4 Discussion

In the following, we briefly discuss the properties of the proposed *ADRF* and compare it with related approaches like boosted trees [75] and regular random forests [4, 23].

The alternating tree growing scheme in *ADRF* can be interpreted as a guided training of each single tree, but also as an explicit collaboration between all trees in the model. All nodes in the trees concentrate on hard-to-predict training samples having high gradients $\mathbf{g}_\mathtt{d}(\mathbf{x}_\mathtt{n})$ and, thus, do not waste effort on samples that are already learned relatively well by the entire model. This property is obvious for the classification case with the explicit computation of weights depending on the gradients. It is also clear for the regression case as larger 'pseudo targets', i.e., gradients, will have more influence on the objective for finding splitting functions. However, unlike boosted trees, this global loss is now an inherent part of *RF*. As we show in our experimental evaluations, this property of *ADRF* typically leads to more compact models in terms of tree depth.

When interrelating the training process of the individual trees in *RF*, one has to take

care of the generalization error ϵ_{G}, which we defined in Equation (2.32). It can also be upper bounded based on the strength of the trees s and the average correlation between the trees $\bar{\rho}$, see [23] and Equation 2.33. A low generalization error requires strong but uncorrelated trees. While *ADRF* explicitly enforces the collaboration of all trees in the forest, the correlation ρ can be kept low as in standard *RF*, because (i) the splitting functions $\sigma(\mathbf{x}; \boldsymbol{\Theta})$ are still drawn completely random and (ii) the set of samples falling in common nodes is different for each tree. In fact, our experimental evaluations on strength and correlation of trees in Sections 3.4.1.5 and 3.4.2.5 for classification and regression, respectively, show an interesting and probably unexpected behavior. Trees in *ADRF* are even less correlated than standard *RF*, while, at the same time, individual trees in *ADRF* perform worse than those of *RF*. Giving more thoughts on this aspect, we realize that this behavior is actually encouraged by the common loss function over the ensemble. The performance of individual trees in the ensemble becomes less important when the training objective is evaluated on the average of all trees. The full ensembles gains center stage.

Continuing the thought about strength and correlation of trees in *RF* and *ADRF*, the effect of the degree of randomization and the number of trees in the ensemble becomes interesting. In Section 3.4.2.6, we conduct such an experiment on regression benchmarks and compare the behavior of *RF* and *ADRF*. As expected, it turns out that *ADRF* better handles the interplay between the amount of randomization and the model size (i.e., number of trees `T`).

We further want to discuss the relations between *ADRF* and boosting with stumps/trees as weak learners, i.e., boosted trees. Alternating between training a full tree and updating the gradients would correspond to boosted trees. Although *ADRF* can be equivalently formulated, there are two main differences. First, boosted trees define a weak learner as a full tree, i.e., all splitting functions in the tree, which is a hierarchical model and hard to parallelize. In contrast, *ADRF* regards the weak learners as one level of nodes over a set of trees, i.e., a subset of the splitting functions, which can be better parallelized. Thus, boosting pools from the same space of weak learners in all iterations, i.e., the number of splitting functions is the same. In contrast, for *ADRF*, this space increases over time as the number of split functions to be optimized typically increases in each iteration. Second, boosted trees train a weak learner from scratch in each iteration, e.g., a new classification or regression tree. Contrary, in *ADRF*, a weak learner corresponds to all splitting functions in a single iteration `d`. This implies that for $\mathtt{d} > 0$ the training samples are conditioned on the structure of the trees up to depth $\mathtt{d} - 1$, which eases the task for the weak learner in the current iteration.

Finally, we also discuss the computational costs of training *ADRF*, boosting, and *RF*. In boosting, the overall classifier corresponds to a flat additive combination of the weak learners, e.g., decision or regression stumps/trees. Thus, each weak learner has to be trained consecutively, which may take a long time to get a desired model complexity for the task at hand. In contrast, *ADRF* regard each depth of the forest as a single weak leaner, i.e., several different splitting functions $\sigma(\mathbf{x}; \boldsymbol{\Theta})$, thus being able to parallelize them.

Standard *RF* can also be trained easily in parallel, which also makes them fast during both training and testing, however, without optimizing a global loss.

3.3 Related Work on ADRF

Freund and Mason [57] presented an algorithm to represent AdaBoost [59] as a *single* decision tree, which is one of the works inspiring *ADRF*. Although the method is termed Alternating Decision Tree or AD Tree, it is quite different to the algorithm presented in this work. In more detail, an AD Tree consists of two different types of nodes, a split node and a predictor node, which are alternated during training. Several split nodes can be attached to a single predictor node, thus allowing for multiple paths of a single example to traverse down the tree. This entity makes both training and inference rather slow. In contrast, *ADRF* sticks with the basic binary tree structure as in standard decision trees and *RF*, but still defines the loss over the full ensemble of trees.

In *ADRF*, the individual trees become interdependent or entangled during training, contrary to *RF*, where the trees are trained independently. This is similar to the works like [117] and [90] that train decision trees breadth-first or according to a priority queue, in order to incorporate contextual features into the learning process. However, both works only consider the predictions within a single decision tree and use those predictions as additional features for the splitting functions in the subsequent stages. In contrast, *ADRF* collects the information from the whole classifier, i.e., all trees, and uses the predictions in order to minimize a global loss function. Extending *ADRF* to incorporate such contextual information as features for splitting functions is straightforward.

Another work that integrates a differentiable loss function into the tree growing process is that of Jancsary et al. [83], where the leaves of the trees store parameters for a Gaussian conditional random field with different interaction types. Each factor type is only associated with a single tree but they are connected implicitly via the random field. In contrast, *ADRF* trains a set of trees, which are connected via averaging over the predictions, i.e., a generic *RF*.

We also would like to point out the differences of *ADRF* to boosted trees [75], i.e., standard boosting with trees as weak learners. As mentioned before, in boosted trees, the weak learners are trained sequentially and the gradients are updated after training each single tree. This makes the whole training phase much slower than *ADRF*, as no parallelization is possible. Furthermore, during training a single weak learner in boosted trees, the gradients $-\mathbf{g}_{\mathtt{d}}(\mathbf{x}_{\mathtt{n}})$ are not updated. Contrary, in *ADRF*, it is exactly this property that allows for learning more compact models as the growing of each tree is guided by the gradient updates from the previous stage in the trees.

3.4 Experimental Evaluation on Machine Learning Data

In this section, we start with our experimental evaluation of *ADRF* for both classification and regression tasks on standard machine learning benchmarks. We also analyze various parameters of *ADRF* on these benchmarks and investigate some properties compared to regular *RF* and *GB*.

3.4.1 Classification Experiments

Our experiments on classification benchmarks give a detailed analysis of the proposed classifier, i.e., *ADF*. After outlining the experimental setup in Section 3.4.1.1, we first compare *ADF* with the most related competing methods, i.e., *RF* and *GB* on 7 different data sets (see Section 3.4.1.2). We also investigate different choices of the loss function. Then, we evaluate the influence of several important parameters common to *ADF*, *RF*, and *GB* on the overall classification performance in Section 3.4.1.3. Finally, we analyze the proposed algorithm in more detail. We investigate the evolution of the gradients in Section 3.4.1.4 and the strength and correlation of the trees in Section 3.4.1.5.

3.4.1.1 Experimental Setup and Data Sets

For a fair comparison between all three classifiers we set the common parameters to the same values. We set the number of trees $\mathtt{T} = 100$ (for *GB*, this is equivalent to the number of weak learners), the maximum tree depth $\mathtt{D}_{\max} = 15$, the number of random splits evaluated per node to $\sqrt{\mathtt{D}}$ [23], the number of random thresholds per split function to 10, and, finally, the minimum number of samples for further splitting to 5. These are standard settings that yield good overall results on all data sets. Keep in mind that tuning these parameters can boost the performance on a single data set, but, at the same time, also worsen it on another one.

For all results, we report the mean and standard deviation of the classification error over 5 independent runs as all classifiers are non-deterministic. In order to evaluate the statistical significance of our results, we perform a t-test with a significance level of $\alpha = 0.1\%$ on the classification error. In particular, we apply a Welch's t-test [193] as we assume unequal variances for the different methods evaluated. For each method under investigation, our null hypothesis is that the classification error of this method is equal or worse than *RF*. For both, *ADF* and *GB*, we expect the null hypothesis to be rejected. Even though the t-test assumes a Gaussian distributed measured variable (the classification error), which is not necessarily the case here, we empirically find that it is a good approximation. We successfully applied an Anderson-Darling test [5] and also show a normal probability plot for both *RF* and *ADF* for one data set in Figure 3.5.

To compare *ADF* with related approaches and to investigate different parameter settings, we use 7 standard machine learning data sets. These are the data sets `Letter` (`Let`), `Mnist`, `PenDigits` (`PD`), and `Protein` (`Pt`) from [55], the `USPS` data set [82], the `Char74k`

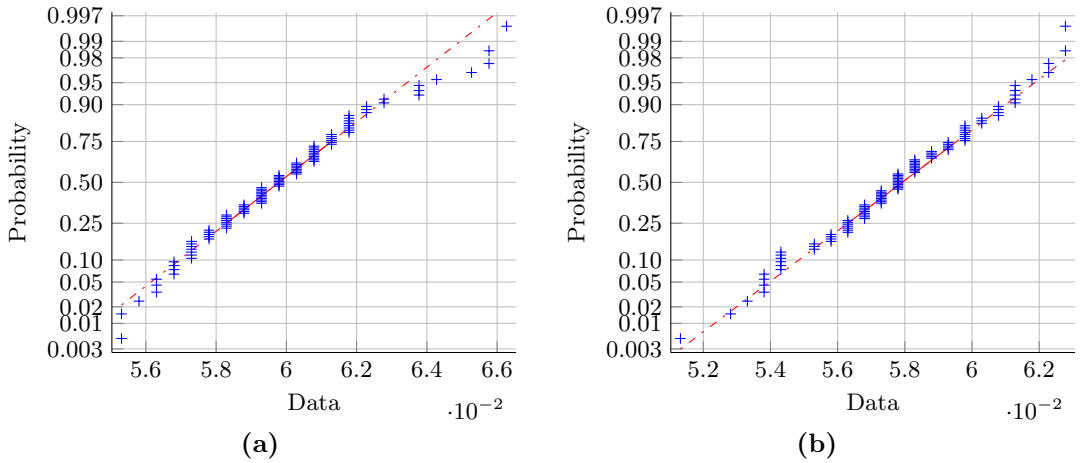


Figure 3.5: Normal probability plots for the classification error of (a) *RF* and (b) *ADF*. We used 100 independent runs on the `USPS` data set. The more the blue pluses build a linear shape (red dash-dotted line), the more their underlying probability distribution gets closer to a normal distribution. For both cases, the distribution of the classification error is almost normal distributed and the data also passes an Anderson-Darling test [5] (see text for more details).

Properties	`Let`	`USPS`	`Mnist`	`C74k`	`OF`	`PD`	`Pt`
#Train	16k	7291	60k	66.7k	320	7494	14.9k
#Test	4000	2007	10k	7400	80	3498	6621
#Features	16	256	784	64	4096	16	357
#Classes	26	10	10	62	40	10	3

Table 3.1: Properties of the 7 machine learning data sets (columns) used in our evaluation for the classification task. Each row defines a different quantity: (i) number of training examples, (ii) number of test examples, (iii) feature dimensionality `D`, and (iv) number of classes `C`.

(`C74k`) data set [27], and the `OlivettiFaces` (`OF`) data set [146]. The properties of these benchmarks are summarized in Table 3.1.

3.4.1.2 Comparison of *ADF* with Other Classifiers

In this section, we compare all methods against each other on all 7 data sets. As the common parameters of *ADF*, *RF*, and *GB* are set equally, as mentioned before, we directly compare the way the tree structure is built. For *ADF* and *GB* we evaluate 5 different loss functions: Logit, Hinge, Exponential, Savage [113] and Tangent [112], see Figure 3.4b. Table 3.2 depicts our results. One can observe that *ADF* and *GB* with the two robust loss functions (Savage or Tangent) share the top performing ranks. *ADF* wins on 4 and *GB* on 3 data sets (highlighted in green). Second and third ranks (highlighted in blue and red, respectively) are also typically shared by *ADF* and *GB* with one of these loss functions.

Method	Loss	Let	USPS	Mnist	C74k	OF	PD	Pt
RF	-	3.41 ± 0.13	5.81 ± 0.18	2.84 ± 0.05	17.81 ± 0.22	3.70 ± 0.84	3.11 ± 0.10	30.76 ± 0.33
GB	Hinge	3.27 ± 0.13	5.79 ± 0.19	2.82 ± 0.04	17.62 ± 0.10	3.65 ± 0.88	3.02 ± 0.12	30.90 ± 0.22
	Logit	3.28 ± 0.15	5.75 ± 0.16	2.81 ± 0.04	17.45 ± 0.19	2.95 ± 0.80	3.11 ± 0.11	30.73 ± 0.33
	Exp	3.22 ± 0.11	5.73 ± 0.19	2.77 ± 0.04	17.10 ± 0.10	3.00 ± 0.72	3.15 ± 0.13	30.77 ± 0.37
	Savage	3.02 ± 0.11	5.67 ± 0.19	2.68 ± 0.09	16.67 ± 0.17	2.90 ± 0.70	3.06 ± 0.09	30.66 ± 0.37
	Tangent	2.90 ± 0.14	5.64 ± 0.14	2.66 ± 0.08	16.27 ± 0.32	2.80 ± 0.54	2.86 ± 0.12	30.82 ± 0.36
ADF	Hinge	3.45 ± 0.15	5.77 ± 0.14	2.82 ± 0.04	17.75 ± 0.28	3.65 ± 1.14	3.14 ± 0.10	30.86 ± 0.34
	Logit	3.24 ± 0.14	5.69 ± 0.16	2.75 ± 0.05	17.44 ± 0.12	3.45 ± 0.83	3.03 ± 0.11	30.68 ± 0.34
	Exp	3.16 ± 0.14	5.68 ± 0.14	2.61 ± 0.06	17.22 ± 0.26	3.25 ± 0.81	2.99 ± 0.14	30.61 ± 0.39
	Savage	3.03 ± 0.09	5.61 ± 0.18	2.57 ± 0.06	17.06 ± 0.10	3.30 ± 0.95	2.91 ± 0.15	30.72 ± 0.34
	Tangent	2.95 ± 0.12	5.55 ± 0.17	2.52 ± 0.09	16.78 ± 0.09	3.10 ± 0.89	2.87 ± 0.12	30.87 ± 0.29

Table 3.2: *ADF* compared with the two main competitors, *RF* and *GB*, on 7 data sets. The best performing methods are marked green, 2nd best blue, and 3rd best red. We indicate statistical significance of the results **bold-face**. As can be seen, *ADF* and *GB* clearly outperform the standard *RF* algorithm on all data sets. Note that each method has the same model complexity in terms of number of trees $\mathtt{T}$ and maximum tree depth $\mathtt{D}_{\max}$.

Method	Let	USPS	Mnist	C74k	OF	PD	Pt	Average
ADF	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
RF	0.9	0.8	0.7	0.8	0.9	0.6	1.1	0.8
GB	2.9	5.2	6.7	3.7	3.8	3.1	9.1	4.9

Table 3.3: Comparison of the training time between the three main competitors on 7 data sets. All values are relative to *GB*. One can clearly see that *GB* is slower to train than both *ADF* and *RF*. On Mnist, *GB* even takes 6 times longer to train compared to *ADF*. *RF* is slightly faster to train than *ADF* (on average over the data sets) as no synchronization between the trees is required.

While the classification error of both *ADF* and *GB* are more or less equal, Table 3.3 reveals one of the main advantages of *ADF* over *GB*, namely the training time. We provide a relative comparison of *RF* and *GB* to *ADF* in Table 3.3. As expected, *GB* is much slower to train (up to 8 times), while *ADF* and *RF* have similar training time. *ADF* is typically slightly slower to train, which is obviously due to the additional synchronization of the trees in order to compute the intermediate gradients, as well as the gradient computation itself. However, for some data sets, *ADF* is even faster to train, which can be explained by a faster convergence of the trees, i.e., the creation of pure leaf nodes earlier in the tree due to the guided training. Thus, *ADF* in combination with a robust loss function inherits the beneficial computational costs from *RF*, while consistently outperforming *RF* on all data sets.

As mentioned before, we also include a statistical significance test in our results. In Table 3.2, we thus mark those results that are better than *RF* with statistical significance as bold-face. For all but one data set (*USPS*), *ADF* and *GB* achieve this goal and reject the null hypothesis.

It is also worthwhile to investigate the influence of the different loss functions. The Logit, Hinge and Exponential losses are typically outperformed by the Savage and Tangent losses for both methods, *ADF* and *GB*. This indicates that the non-convexity of the loss, and thus the robustness to outliers (see Section 3.2.3 and Figure 3.4b), plays a crucial role for the weight updates and thus the tree growing. For our further experiments, we fix the loss function to be the Tangent loss, as we can expect the best overall performance.

3.4.1.3 Parameter Evaluation

In this section, we analyze several parameters that influence the behavior of the three learning algorithms, *RF*, *GB*, and *ADF*.

First, we evaluate the influence of two important parameters common to all evaluated classifiers on the Letter and Pendigits data sets. We investigate the number of trees T and the maximum tree depth $\mathtt{D}_{\max}$, which we vary in the ranges $[1, 500]$ and $[5, 25]$, respectively. As mentioned before, we choose the Tangent loss for both *ADF* and *GB* due to the good overall performance in the previous experiment.

It is obvious that *ADF* and *GB* only improve over *RF* for larger number of trees (see Figures 3.6a and 3.6b). If the number of trees is small, both *ADF* and *GB* do not have the chance to extract reliable information about the current state of the classifier. Thus, the performance is more or less the same as *RF* in this regime. However, as soon as the number of trees increases, also the performance of *ADF* and *GB* increases and outperforms *RF*. The behavior of the second parameter, i.e., the maximum tree depth $\mathtt{D}_{\max}$, is different. Here, *ADF* and *GB* consistently outperform *RF* (see Figures 3.6c and 3.6d). This can be explained by the fact that the weight updates can rely on more stable predictions, as a larger number of trees is available, regardless of the current depth of the trees. This experiment also reflects the guidance of the tree growing mentioned in Sec. 3.2.4, as *RF* need deeper trees to reach the performance of *ADF* and *GB*. When using deeper trees, the performance of *RF* closely approaches *ADF* and *GB* on these two data sets.

Finally, we investigate a scaling of the margin defined in Equation (3.9) that is used to evaluate the loss function. We define the margin scaling as $\alpha \cdot m_{\text{rf}}(\mathbf{x})$, where we let α vary in $[0.2, 1.8]$. The reason to explore α is that the loss functions are defined in $[-\infty, +\infty]$ but the margin $m(\mathbf{x})$ can only take values in $[-1, +1]$ because *RF* and *GB* only predict values in $[0, 1]$. We present the result in Figures 3.6e and 3.6f, again for 2 data sets. Obviously, this parameter only influences *ADF* and *GB*. We get best results with $\alpha = 1.4$ for *ADF* and $\alpha = 1$ for *GB* on the `Letter` data set. For `PenDigits`, the best results are achieved with $\alpha = 0.6$ for *ADF* and $\alpha = 1.0$ for *GB*.

3.4.1.4 Progress of the Gradients

In this section, we analyze the progress of the norm of the gradients $|-\mathbf{g}_{\mathtt{d}}(\mathbf{x}_{\mathtt{n}})|$ (see Equation (2.13)) throughout the training stages $\mathtt{d} = 1, \ldots, \mathtt{D}_{\max}$ of *ADF*. The intuition is to get a better understanding of the internals of *ADF*.

In Figure 3.7, we visualize this progress of the norm (on the y-axis) for a randomly selected subset of the training examples (x-axis) from the `MNIST` data set. The initial norms of the gradients are proportional to the class weights of the data set, i.e., lower norm for samples from more frequent classes. While these initial norms of the samples take different values, the variation is too small to be visible in the figure. The reason is that the norms either converge to a low final value (for easy to classify samples) or diverge (for hard to classify samples), making the variation of the gradient norms higher during the final training iterations.

We analyze this behavior in more detail with some examples in Figure 3.8. We select the top 6 samples from the data set that have high and low weights after training, respectively. To be more robust, we average over the last 3 iterations and choose the highest and lowest values. The first two rows in Figure 3.8 illustrate some converging progresses of the gradient norm. These samples are easy to classify and typically belong to the same class. Only for a better variation in the visualization, we include the first 6 samples with low final gradient norm out of the first 600 with equal distance. The second two rows in

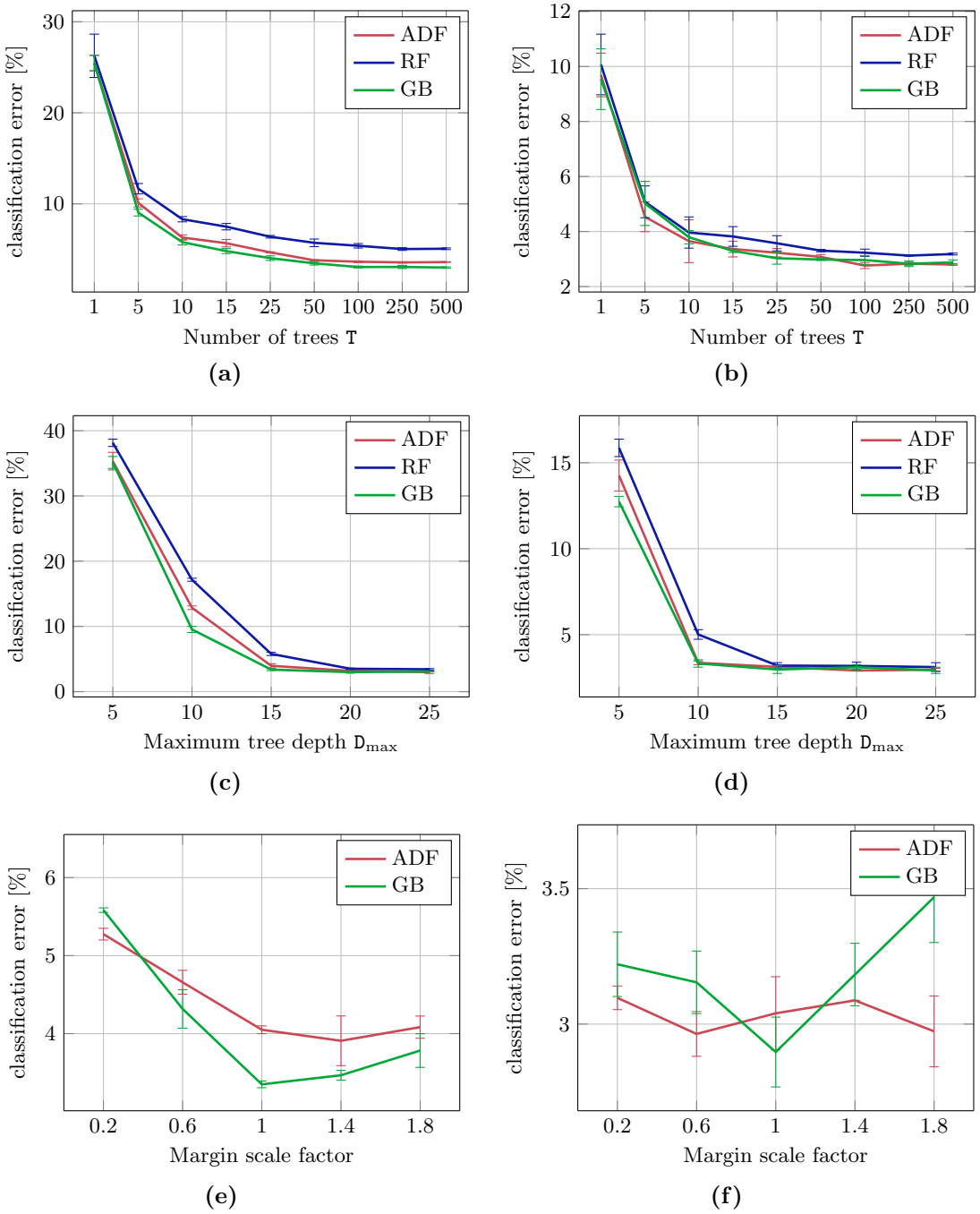


Figure 3.6: Three different parameters are evaluated for three different methods (*RF*, *GB*, and *ADF*) on two data sets. Each row shows the result of one particular parameter on two different data sets, `Letter` (left) and `PenDigits` (right). The parameters under consideration are: (a-b) the number of trees `T`, (c-d) the maximum tree depth D_{max}, and (e-f) the scaling factor of the margin (which has no influence on *RF*).

Figure 3.7: Progress of the gradient norm on the `Mnist` data set for a randomly selected subset of training examples. Each pixel on the x-axis corresponds to one out of 1000 different training examples. The y-axis shows the corresponding evolution of the gradient norm during the training phase. Please note that we scaled the y-axis for better visualization. The original scale would be 15 pixel as we used a maximum tree depth of $\mathtt{D}_{\max} = 15$ for this experiment. Dark values indicate high norm and vice versa (excluding the border). See text for more details.

Figure 3.8 illustrate some hard to classify examples along with their progress of $|-\mathbf{g}_t(\mathbf{x}_n)|$. One can observe that these samples are indeed hard to classify, as their appearance can easily be confused with other classes.

3.4.1.5 Strength and Correlation of Trees

The strength and correlation of trees in *RF* are important factors for achieving a low generalization error, as discussed in Section 2.2.4.4 (see Equation (2.33)). When training a *RF* model, one aims at a good trade-off between high strength of the trees and low correlation between them. Obviously, if there would exist a single tree that gives perfect predictions, the correlation between several trees is irrelevant and can be at a maximum. In fact, one would then only need this single tree, but this is an unrealistic scenario. Low correlation means that all the trees should be different and provide different predictions, which, when averaged, give good overall results. Roughly speaking, this is the case when the errors made by a single tree are corrected by other trees.

In *RF*, each tree is trained independently from each other, which helps to provide low correlation. On the other hand, the proposed learning scheme, *ADF*, exploits knowledge from the full model, i.e., the ensemble, to train the trees. The trees are thus interrelated in some form, which might increase correlation as already discussed before. We argue in Section 3.2.4 that our model still randomly samples splitting functions and minimizing a common loss does not increase the correlation. In this section, we thus provide an empirical analysis of this issue and compare strength and correlation of trees for *RF* and *ADF*. In fact and contrary to our prior belief, the outcome of the experiments in this section show an interesting behavior, namely, that the correlation actually becomes lower when training with the *ADF* scheme.

We used Equation (2.35) to compute the strength of the trees, where we replaced the expectation over the data- and labelspace with the empirical mean over a test set of a given data set. Computing the strength also requires the margin from Equation (2.34), where we again replaced the expectation with the empirical mean over the trees. To compute the average correlation as suggested by Breiman [23] (and defined in Equation (2.36)), the learned parameters of the *RF* are directly compared. However, this might be complicated

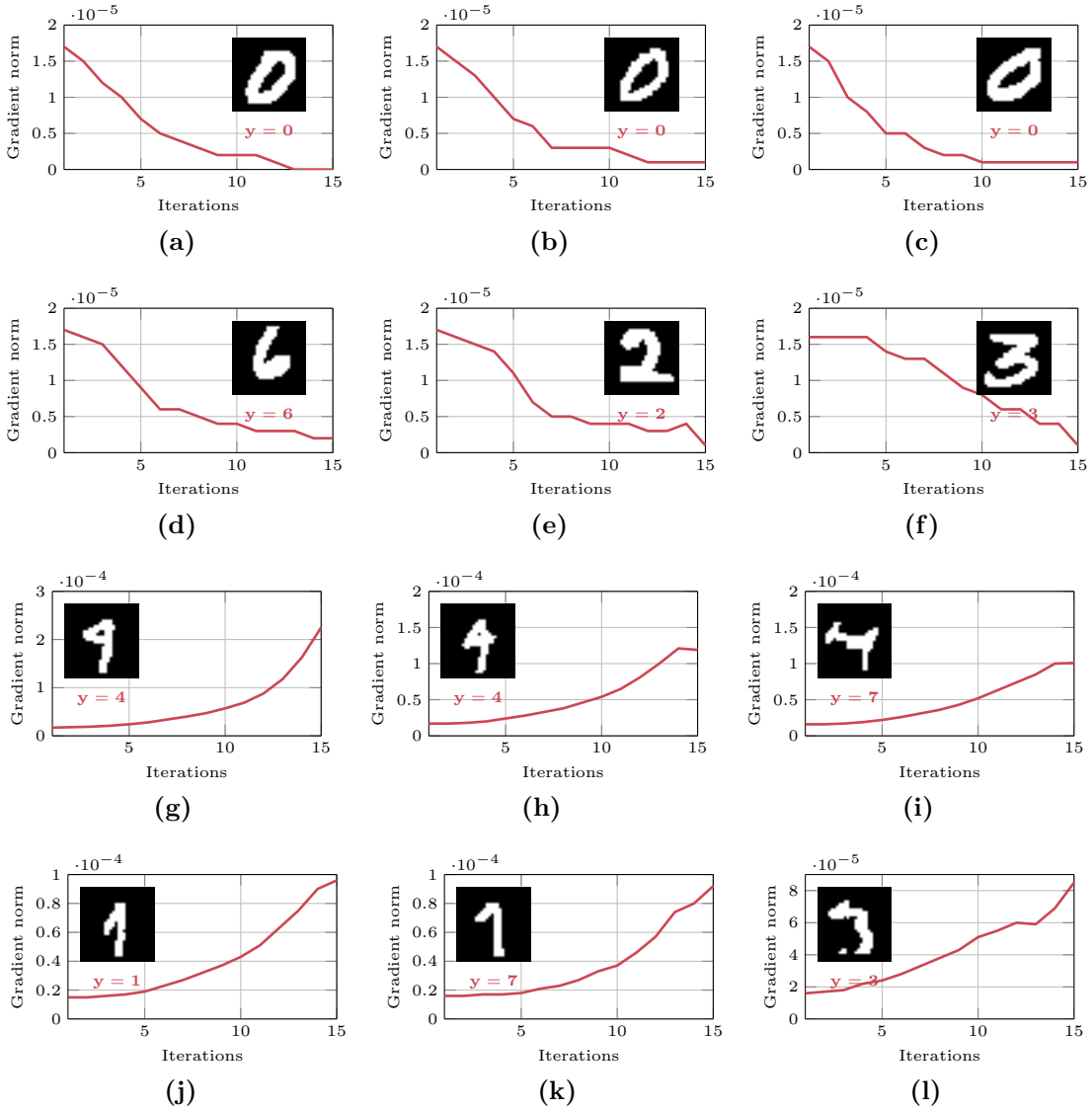


Figure 3.8: These figures illustrate the evolution of the gradient norms in *ADF* for different training examples. The first two rows, (a) to (f), show easy to classify training examples. The norm of the gradients decreases with increasing number of training iterations. The second two rows, (g) to (l), show hard training examples. In this case, the gradient norm increases, i.e., the loss is high for these examples. Each figure also visualizes the corresponding training examples $\mathbf{x}$ from the `Mnist` data set alongside with its ground truth label $\mathbf{y}$. Some of the hard to classify training samples are also visually ambiguous.

to compute as we do not require the trees to have the same size. Moreover, directly comparing the learned parameters can be suboptimal as different models can still yield the exact same predictions for all data points. Consider for instance two copies of the same tree, where we add an additional dummy split function on top of the root node

which forwards all data points to one side. Beside the fact that the tree models now have different size, this little change has no effect on the predictions but would yield a lower correlation between the trees. Here, we thus evaluate the correlation of the predictions of the individual trees. In particular, we compute the correlation of the predictions between each of the single trees, which results in a $\mathtt{T} \times \mathtt{T}$ matrix, where $\mathtt{T}$ is the number of trees. To get a final estimate of the correlation, we average the upper triangle of this symmetric matrix.

We compute these two quantities, i.e., strength and correlation for *ADF* and *RF* on three different data sets (`Letter`, `USPS`, and `Mnist`). Moreover, we include the classification error of the full ensemble as well as the mean classification error of each single tree in the ensemble. The latter quantity can be considered an alternative way to measure the strength of each individual tree. The results are depicted in Figure 3.9. First of all, we can again see in Figure 3.9a that *ADF* achieves lower classification error than *RF* on all three data sets. Then, we can make several interesting observations in Figures 3.9b, 3.9c, and 3.9d, which show the strength, the correlation, and the mean classification error of the individual trees, respectively. *ADF* seems to have slightly lower strength than *RF* (Figure 3.9b) but also less correlation between the trees (Figure 3.9c). Moreover, Figure 3.9d shows that the individual trees of *ADF* yield poorer performance than those of *RF*. Interestingly, it seems that the *ADF* training scheme builds a set of trees that give good performance when averaged. However, when evaluated individually, they perform worse than individually trained trees in *RF*. Actually, this is the behavior we want to achieve. We do not care about the performance of single trees, but rather about the combination of all of them. In Section 3.4.2.5, we perform the same experiments for the regression case with a similar outcome.

3.4.2 Regression Experiments

Similar to the experiments for classification, we also analyze the performance and properties of *ARF* for the regression task. After a detailed description of the experimental setup and the data sets, we compare *ARF* with *RF* and *GB* on 22 standard regression benchmarks. Furthermore, we also investigate the influence of different parameters of *ARF*.

3.4.2.1 Experimental Setup and Data Sets

We use a set of 22 machine learning benchmarks from different sources (UCI, StatLib, Delve)[1] to evaluate the difference between *RF*, *GB*, and *ARF*. For *GB* and *ARF*, we evaluate the loss functions presented in Section 3.2.2 (Squared, Absolute, and Huber).

As most benchmarks do not provide specific train-test splits, we split the given data into 60% training and 40% testing data. To provide statistically fair results, we repeat the

[1] A collection of the data sets can be found at `http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html`

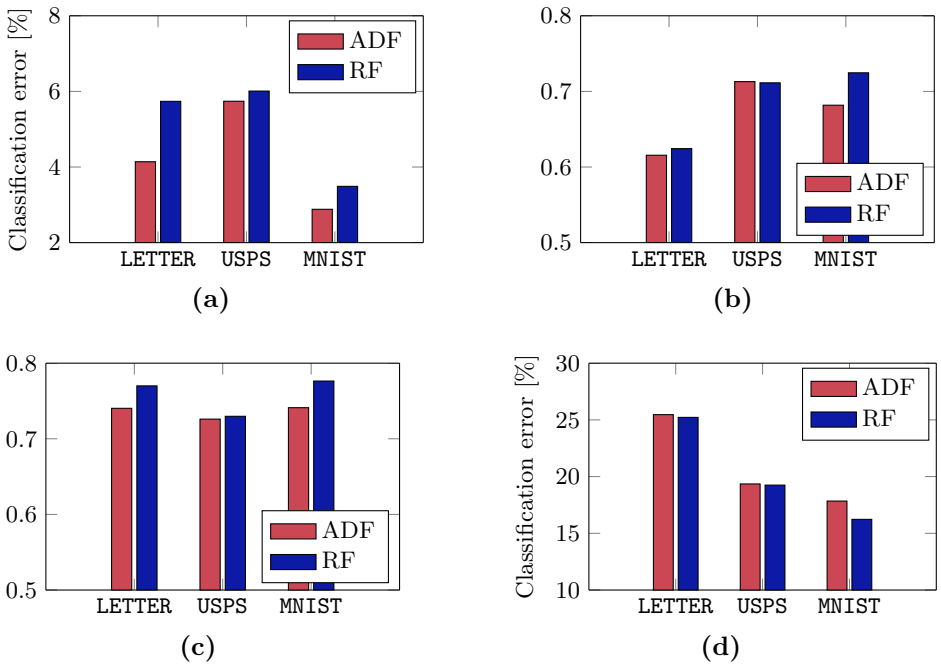


Figure 3.9: This figure shows in (a) the classification error, in (b) the strength, in (c) the correlation, and in (d) the mean performance of single trees of the *ADF* and *RF* models. Each figure depicts these quantities for three different data sets: `Let`, `USPS`, and `Mnist`. It can be observed in (a) that *ADF* outperforms *RF* in terms of classification error. One can also see in (b) and (c) that *ADF* has slightly less strong trees but also shows less correlation between them, which makes the overall ensemble stronger. Finally, (d) shows that the average performance of the single trees is worse for *ADF* than for *RF*. Please see the text for more details.

following procedure 5 times and average the results: We first build a random train-test split (unless an explicit split is given) with the above defined ratio. Then, for each split, we train and test all methods 4 times in order to further decrease statistical uncertainties due to the random tree growing scheme involved in all methods. This procedure results in a total of 20 averaging runs per method and data set. As for classification, we apply a Welch's t-test to gain insight into the statistical significance of our results. We measure the performance as the Root Mean Squared Error (RMSE).

The parameters of all trees for the different methods (*RF*, *GB*, and *ARF*) are set equally: We used 50 trees with a maximum depth of 15. Tree growing also stops if the sample size in a node becomes lower than 10. As for classification, we use $\sqrt{\mathtt{D}}$ random tests [23] (`D` being the input feature dimensionality) and 20 randomly chosen thresholds. For *GB*, the number of trees again corresponds to the number of iterations/weak learners, thus having the same model complexity as the other methods.

3.4.2.2 Comparison of *ARF* with Other Regressors

In Table 3.4, we compare *ARF* with *RF* and *GB* and present our results as mean and standard deviation of RMSE values. Again, we mark the top three performing methods with different colors (green, blue, and red, respectively). Bold-faced values indicate statistically significant better results compared to *RF* according to the same Welch's t-test as it was used for the classification case.

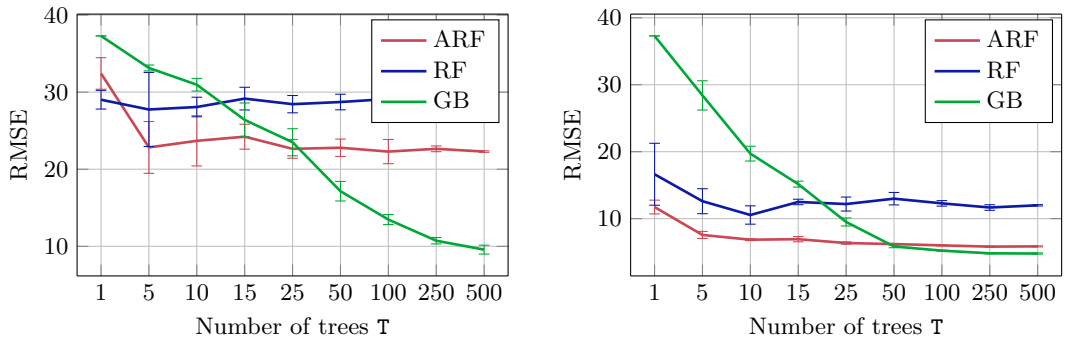
As can be seen from the table, *ARF* and *GB* win in all but 1 data set, in most cases with statistical significance. *ARF* wins 14 and *GB* 7 data sets. The performance difference between the loss functions for both *ARF* and *GB* is, however, rather minor. Throughout all experiments we fix the parameter δ for the Huber loss to 1.0. Furthermore, we also evaluate the overall training and testing time. Averaged over all data sets, *ARF* and *RF* have more or less the same computational costs, while *GB* takes approximately 5 times longer, which is similar to the classification case. Please note that these values correspond to 50 weak learners and could be even more distinctive if the number of weak learners is increased.

3.4.2.3 Comparison of Common Parameters

While the goal in the previous experiment was to evaluate all methods with the same model complexity (i.e., same number of trees and maximum depth), we further compare the performance for different complexities in this second experiment. We choose the `Pol` data set, fixed the loss for *GB* and *ARF* to be the squared loss, and varied the number of trees, i.e., weak learners, between 10 and 500 for two fixed maximum depth values, 5 and 15. Again, we averaged the results for each parameter combination over several train-test splits. The results are illustrated in Figure 3.10. We make three observations: First, a larger amount of weak learners is important for *GB* (both plots), which, however, also implicates a longer training time compared to *RF* and *ARF*, as no parallelization is possible. Note that we used a fixed shrinkage value for *GB* for all values of `T`. Increasing this parameter for less trees in the model would yield better results for *GB*. Second, *GB* can handle shallow trees as weak learners much better than *RF* or *ARF* (see Figure 3.10a). Finally, the performance of both *GB* and *ARF* is similar with the appropriate parameter settings, while *RF*, which do not optimize a global loss function, lags behind (see Figure 3.10b).

Data set	Scale	RF	GB			ARF		
			Absolute	Squared	Huber	Absolute	Squared	Huber
Abalone		1.92 ± 0.02	1.96 ± 0.04	1.96 ± 0.02	1.94 ± 0.04	1.90 ± 0.02	1.90 ± 0.02	1.90 ± 0.02
Ailerons	10^{-4}	1.51 ± 0.01	1.32 ± 0.01	1.32 ± 0.01	1.32 ± 0.01	1.26 ± 0.01	1.26 ± 0.01	1.26 ± 0.01
AutoMPG		1.99 ± 0.03	1.98 ± 0.06	**1.94 ± 0.04**	1.96 ± 0.05	**1.96 ± 0.03**	1.96 ± 0.04	1.97 ± 0.03
AutoPrice		1596 ± 60	1550 ± 71	1530 ± 74	1569 ± 63	**1518 ± 62**	**1500 ± 49**	**1535 ± 33**
BreastCancer		30.2 ± 0.2	31.4 ± 0.5	31.5 ± 0.4	31.5 ± 0.5	30.1 ± 0.1	30.1 ± 0.2	30.0 ± 0.2
CalifornaHousing	10^{2}	428.9 ± 4.1	**365.7 ± 3.2**	**365.7 ± 3.3**	**365.5 ± 2.3**	**352.0 ± 2.6**	**350.5 ± 2.3**	**351.7 ± 3.0**
CartDelve		$.892 \pm .009$	**$.863 \pm .001$**	**$.863 \pm .002$**	**$.863 \pm .002$**	**$.818 \pm .001$**	**$.819 \pm .001$**	**$.819 \pm .001$**
CPUAct		1.99 ± 0.01	**1.79 ± 0.02**	**1.79 ± 0.02**	**1.79 ± 0.02**	**1.80 ± 0.02**	**1.80 ± 0.02**	**1.80 ± 0.02**
CPUSmall		2.27 ± 0.01	**2.09 ± 0.01**	**2.09 ± 0.01**	**2.09 ± 0.02**	**2.09 ± 0.02**	**2.07 ± 0.01**	**2.08 ± 0.02**
DeltaAilerons	10^{-4}	1.15 ± 0.00	1.17 ± 0.01	1.17 ± 0.00	1.17 ± 0.01	1.14 ± 0.00	1.14 ± 0.00	1.14 ± 0.00
DeltaElevators	10^{-3}	1.11 ± 0.00	1.12 ± 0.00	1.12 ± 0.00	1.12 ± 0.00	1.12 ± 0.00	1.12 ± 0.00	1.12 ± 0.00
Diabetes		$.569 \pm .013$	$.593 \pm .027$	$.593 \pm .022$	$.605 \pm .031$	$.558 \pm .013$	$.560 \pm .011$	$.556 \pm .012$
Elevators	10^{-3}	2.43 ± 0.01	2.06 ± 0.02	2.06 ± 0.02	2.07 ± 0.02	1.98 ± 0.02	1.98 ± 0.01	1.99 ± 0.02
FriedmanDelve		1.27 ± 0.01	**0.98 ± 0.00**	**0.98 ± 0.01**	**0.98 ± 0.00**	**0.87 ± 0.00**	**0.87 ± 0.00**	**0.87 ± 0.00**
Housing		2.40 ± 0.05	**2.24 ± 0.05**	**2.23 ± 0.06**	**2.24 ± 0.07**	**2.27 ± 0.07**	**2.24 ± 0.05**	**2.26 ± 0.05**
Kinematics		$.126 \pm .001$	**$.108 \pm .001$**	**$.109 \pm .002$**	**$.108 \pm .001$**	**$.092 \pm .001$**	**$.092 \pm .001$**	**$.092 \pm .001$**
Machine		33.5 ± 0.8	33.6 ± 1.2	32.9 ± 1.0	33.7 ± 1.1	33.1 ± 1.1	33.2 ± 0.9	32.7 ± 1.2
Pol		11.8 ± 0.8	**5.8 ± 0.2**	**5.8 ± 0.2**	**5.8 ± 0.2**	**6.2 ± 0.2**	**6.2 ± 0.1**	**6.2 ± 0.1**
Pyrimidinis		$.067 \pm .002$	$.063 \pm .002$	$.061 \pm .003$	$.061 \pm .004$	$.060 \pm .002$	$.060 \pm .002$	$.060 \pm .002$
Servo		$.427 \pm .022$	**$.281 \pm .016$**	**$.289 \pm .014$**	**$.285 \pm .016$**	**$.357 \pm .017$**	**$.359 \pm .020$**	**$.361 \pm .016$**
StockAirplane		$.708 \pm .014$	**$.597 \pm .023$**	**$.610 \pm .015$**	**$.594 \pm .011$**	$.715 \pm .017$	$.720 \pm .021$	$.709 \pm .027$
Triazines		$.100 \pm .001$	$.097 \pm .002$	$.096 \pm .004$	$.096 \pm .003$	$.098 \pm .002$	$.098 \pm .002$	$.098 \pm .002$

Table 3.4: *ARF* compared with the two main competitors, *RF* and *GB*, on 22 data sets. Best performing methods are marked green, 2nd best blue, and 3rd best red. Statistical significance is indicated **bold-face**. The results are presented as RMSE values averaged over several runs (mean and standard deviation is given).



(a) $\mathtt{D}_{\max} = 5$ **(b)** $\mathtt{D}_{\max} = 15$

Figure 3.10: Parameter evaluation of *RF*, *GB* and *ARF* conducted on the `Pol` data set. We vary the number of trees `T` for two choices of the maximum tree depth, (a) $\mathtt{D}_{\max} = 5$ and (b) $\mathtt{D}_{\max} = 15$. In (a), one can observe that *GB* can best handle a limited tree depth. However, deeper trees yield better results (b) and, in this setup, *GB* and *ARF* achieve similar performance. Note that *GB* takes much longer to train. Overall, both *GB* and *ARF* outperform standard *RF*.

3.4.2.4 Influence of the Learning Rate on *ARF*

Further, we examine the influence of the learning rate (or shrinkage factor) in *ARF*, which is defined in Equation (3.6). As in *GB*, the learning rate is directly related to the size of the gradient descent step in the training procedure. The shrinkage factor is typically set relatively low (e.g., 0.1) for *GB* because many gradient descent steps are performed. However, this is different for *ARF* where the number of gradient descent steps is limited by the maximum tree depth $\mathtt{D}_{\max}$. We thus investigate the influence of this parameter on *ARF* for different data sets and vary the parameter in the range $[0.1, 2.0]$. The results for 6 randomly selected data sets are shown in Figure 3.11. We can observe very different behavior for the different data sets. This indicates that the performance of *ARF* could be further improved with a separate choice of the learning rate for each data set (e.g., via cross validation). For future work, it could be interesting to make this parameter adaptive in each gradient descent step, i.e., perform a line search, instead of using a fixed constant value.

3.4.2.5 Strength and Correlation of Trees

As for the classification case (see Section 3.4.1.5), we perform an analysis on the trade-off between strength and correlation of the trees in the proposed *ARF* model and plain *RF*. We investigate the regression performance as RMSE, the strength of the trees, the correlation between individual trees, and the mean performance of single trees on three different data sets (`CaliforniaHousing`, `Kinematics`, and `Pol`). The correlation is computed in the same way as described in Section 3.4.1.5 with the difference that the output of the trees is not a probability distribution but raw regression values. One difference to the

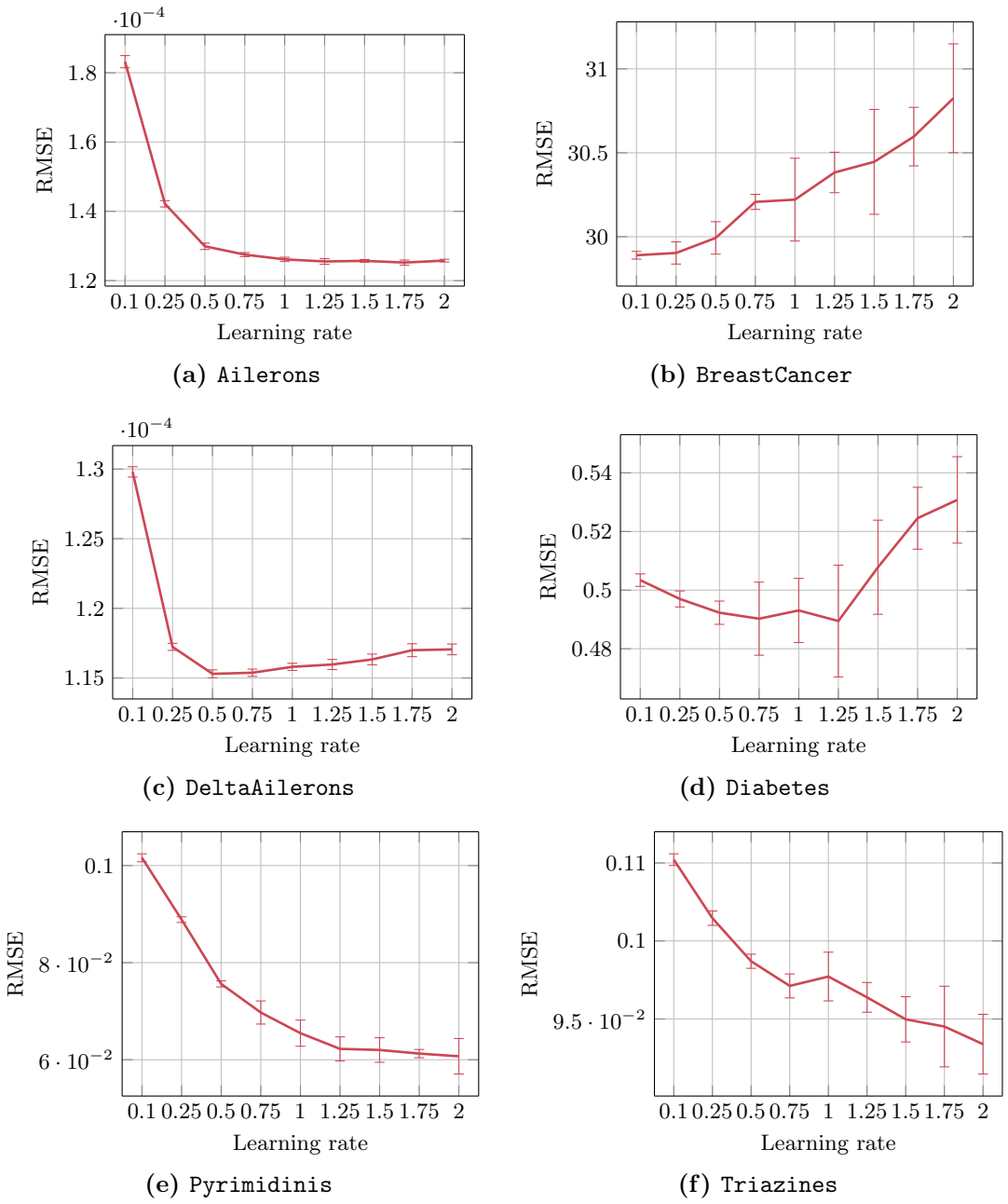


Figure 3.11: Influence of the learning rate of *ARF* on the regression performance for 6 different data sets. We vary the parameter in the range $[0.1, 2.0]$ and can observe different behavior for each data set.

classification case is the computation of the strength as we cannot compute a margin here. We thus, use the inverse of the regression performance, which is not optimal but reasonable in this case. Nevertheless, we still have the mean performance of individual trees as an alternative measure of the strength, which is more meaningful for the regression case.

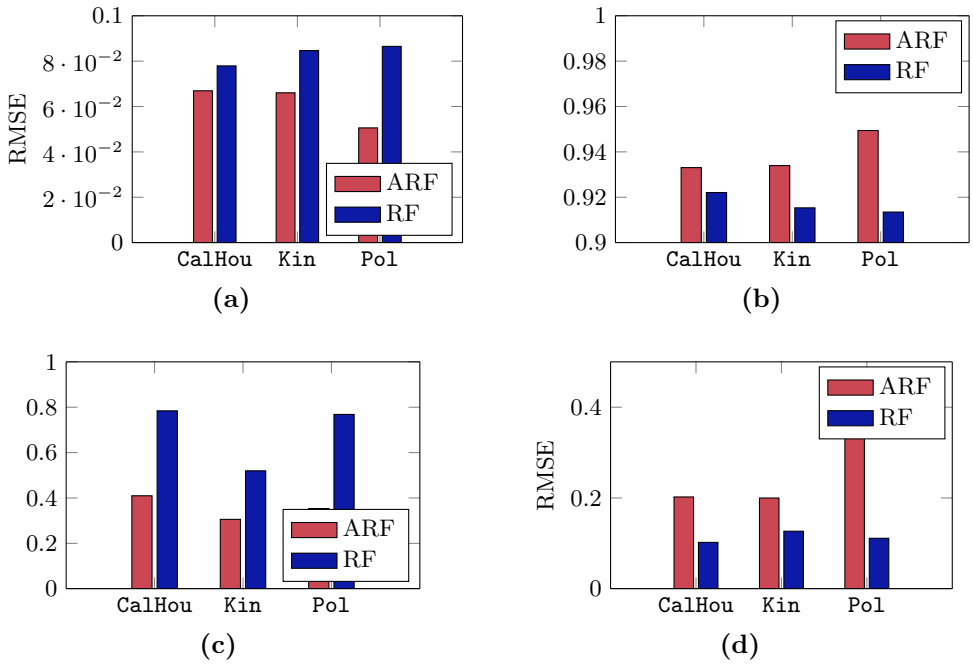


Figure 3.12: This figure shows in (a) the regression error (RMSE), in (b) the strength, in (c) the correlation, and in (d) the mean performance of single trees of the *ARF* and *RF* models. Each figure depicts these quantities for three different data sets, `CaliforniaHousing` (abbreviated with `CalHou`), `Kinematics` (abbreviated with `Kin`), and `Pol`. It can be observed in (a) that *ARF* outperforms *RF* in terms of regression error. One can also see in (c) that *ARF* shows less correlation between the trees, which makes the overall ensemble stronger. Finally, (d) shows that the average performance of the single trees is worse for *ARF* than for *RF*. Please see the text for more details.

Please note that we normalize all the data (i.e., the target values) to be within the range $[0, 1]$ in order to compare values across different data sets.

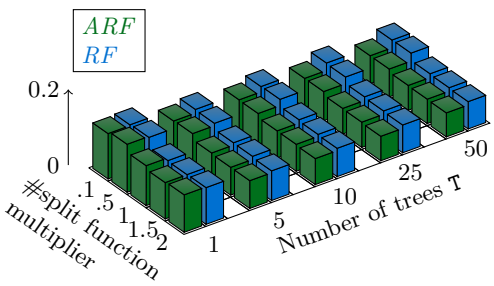
Figure 3.12 provides the results on three different data sets. The outcome indicates a similar interpretation as for the classification case. We see in Figure 3.12a that the overall regression error is smaller for *ARF* compared to plain *RF*. The strength of *ARF* is higher than that of *RF* (see Figure 3.12b), which is different to the classification case, but is most likely attributed to the suboptimal measure. However, when looking at the mean regression error of the individual trees in Figure 3.12d, we observe that individual trees trained via *ARF* are significantly worse compared with those from *RF*. The correlation between the trees, on the other hand, is significantly less for *ARF*. This leads to the same conclusion as for the classification case in Section 3.4.1.5. The proposed training scheme improves the overall ensemble, i.e., the actual model that is used to make predictions, instead of individual trees.

3.4.2.6 The Effect of Randomization and Model Size

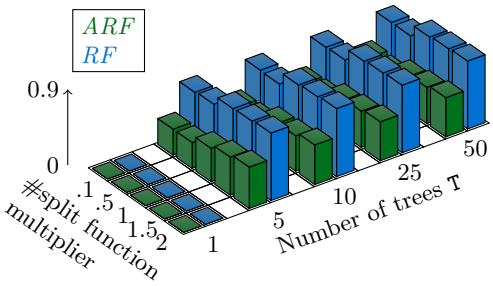
The strength and correlation investigated in the previous experiment motivate an interesting analysis of the internal behavior of *RF* and *ADRF*. The upper bound on the generalization error (Equation (2.33)) tells us that a good trade-off between strength and correlation is essential for achieving good performance with *RF*. Although other parameters are also involved, there is one specific combination of two aspects of *RF* that well reflects this trade-off: the amount of randomization of single trees and the number of trees to average (i.e., the model size). The more randomization is injected into the individual trees, the worse their individual performance. This, on the other hand, leads to lower correlation between the trees, which can be exploited to compensate the performance loss by using a larger ensemble. In this experiment, we want to analyze this trade-off and its effect on the predictive power of both *RF* and *ARF*.

To simulate the randomization of individual trees, we varied the number of splitting functions that are evaluated per node. As already stated in Section 3.4.1.1, a good rule of thumb is to evaluate $\sqrt{\mathtt{D}}$ randomly sampled splitting functions, where `D` is the dimensionality of the input data $\mathcal{X}$ [23]. We modify this value with a multiplier and round to the next integer value to fix the number of splitting functions evaluated per node. We use 5 values for the multiplier, $[0.1, 0.5, 1, 1.5, 2]$. The model size is easily simulated with the number of trees in the ensemble, where we evaluate $\mathtt{T} = [1, 5, 10, 25, 50]$. We use three data sets, `CaliforniaHousing`, `Kinematics`, and `Pol` and measured the overall performance of the ensemble as RMSE, the correlation between the trees, and the average performance of individual trees (again as RMSE). The outcome of these experiments can be seen in Figures 3.13, 3.14, and 3.15 for the three data sets, respectively.

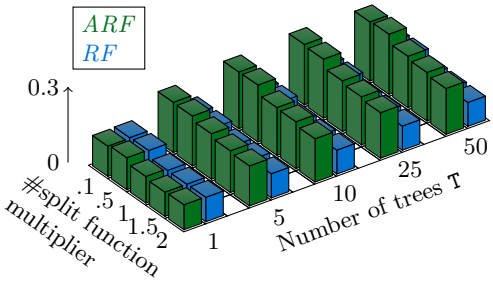
All three data sets show a similar outcome of the experiment, where the effect of randomization is most clearly seen for the `Pol` data set. One can clearly see that the overall error decreases with less randomization, i.e., more splitting functions evaluated. Interestingly, one would expect that the error actually starts to increase again if the randomization gets too low of individual trees. However, this effect cannot be observed for these experiments, which indicates that there is still enough randomness in the trees. The correlation between trees steadily increases as the randomization gets less. It can be noted in Figure 3.15b that the correlation increases much stronger for *RF* than *ARF*, which also might explain the better overall performance in terms of RMSE, cf., Figure 3.15a (not clearly seen due to scaling) and Table 3.4. As in the previous section, we also show the average performance of individual trees, which is another indicator for the strength of the trees. Obviously, the strength of individual trees increases (i.e., lower RMSE) as soon as the randomization during split function optimization gets less, which can be observed in Figure 3.15c. Interestingly, though, one can also see that this measure of strength does not vary too much for *ADRF*, which is an effect of the global loss optimization as already discussed in the previous section. Finally, we want to note the special case when the model consists of only a single tree. In this case, we set the correlation to 0 for a



(a) Error in RMSE



(b) Correlation



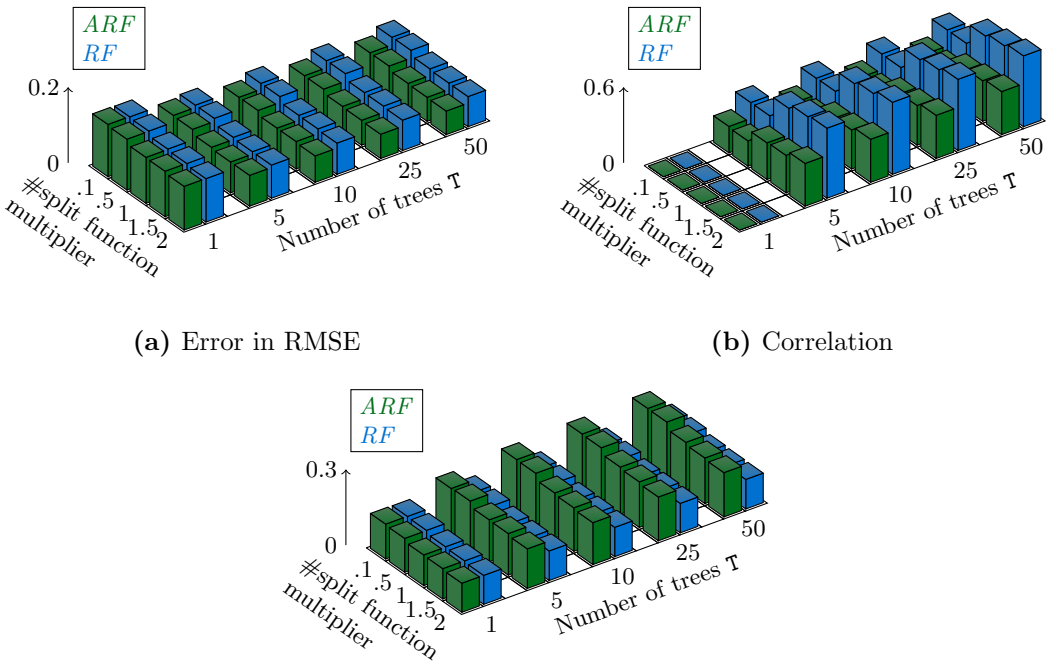
(c) Average tree performance in RMSE

Figure 3.13: We evaluate the effect of the trade-off between randomization (the multiplier of the number of splitting functions) and model size in the ensemble of trees. We measure in (a) the overall performance as RMSE, in (b) the correlation between trees, and in (c) the average performance of individual trees on the `CaliforniaHousing` data set for *RF* (blue) and *ARF* (green). See text for more details.

better visualization. Also, observe the different behavior of the average performance of individual trees for *ARF* when a single or multiple trees are used. For a single tree, the average performance also increases (i.e., lower RMSE) similar to *RF*. On the other hand, when multiple trees are used in the ensemble, the behavior of *RF* and *ARF* is different.

3.5 Summary

In this chapter, we presented a novel training scheme for *RF*, *Alternating Decision and Regression Forests*, which is the main contribution of this thesis. In contrast to standard *RF*, *ADRF* enable the minimization of a global loss function defined over the full model, i.e., all trees in the ensemble. We formulate the learning algorithm as a loss minimization problem and borrow ideas from *GB* to do gradient descent in functional space, see Section 3.2. While plain *GB* can also be used to minimize a loss over a set of randomized trees, such a formulation has the clear drawback of a long training time as the trees have to be trained iteratively. In contrast, *ADRF* directly integrates the loss minimization into



(a) Error in RMSE

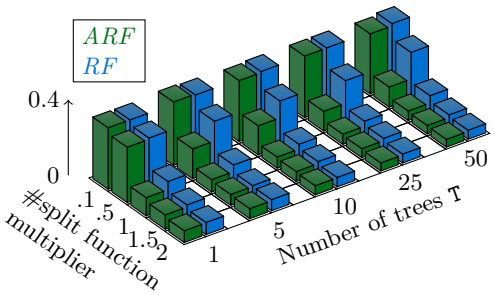
(b) Correlation

(c) Average tree performance in RMSE

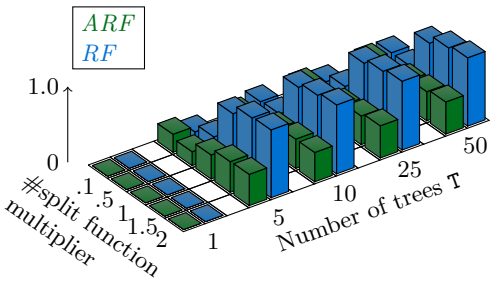
Figure 3.14: We evaluate the effect of the trade-off between randomization (the multiplier of the number of splitting functions) and model size in the ensemble of trees. We measure in (a) the overall performance as RMSE, in (b) the correlation between trees, and in (c) the average performance of individual trees on the `Kinematics` data set for *RF* (blue) and *ARF* (green). See text for more details.

the tree growing process and allows for parallelizing the training over the trees. Thus, our proposed formulation inherits the computational benefits of standard *RF*, which made this learning algorithm popular for so many computer vision applications. In order to tackle a wide spectrum of *ML* problems, we present both, a regression and classification formulation in Sections 3.2.2 and 3.2.3, respectively.

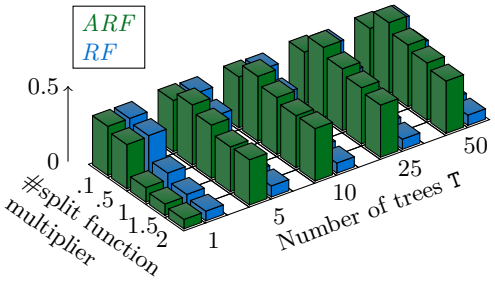
The main outcome of the proposed learning scheme is a *RF* variant that optimizes a global loss over the full ensemble, which results in clearly better predictions and generalization capabilities compared with standard *RF*. The results of *ADRF* are similar to *GB* (with decision or regression trees as weak learners) while, at the same time, being much faster during training. The last part of this chapter underlines the effectiveness of the proposed formulation with empirical evaluations on a set of standard classification and regression benchmarks in Section 3.4. We also provide some empirical analysis of *ADRF* in order to better understand the intrinsic behavior of this learning algorithm. *ADRF* can also be effectively employed for a large set of computer vision applications, which will be demonstrated in the following chapters.



(a) Error in RMSE



(b) Correlation



(c) Average tree performance in RMSE

Figure 3.15: We evaluate the effect of the trade-off between randomization (the multiplier of the number of splitting functions) and model size in the ensemble of trees. We measure in (a) the overall performance as RMSE, in (b) the correlation between trees, and in (c) the average performance of individual trees on the `Pol` data set for *RF* (blue) and *ARF* (green). See text for more details.

CHAPTER 4

Object Detection with Alternating Decision and Regression Forests

Object detection is one of the most important tasks in computer vision. The goal is to accurately localize all instances of a semantic category, e.g., a person or a car, in an unseen image and to outline their boundaries. The outline is typically a bounding box around the object. Modern object detectors have to be both accurate and fast as they are often employed for time-critical tasks in the automotive or robotic industry. Moreover, they also act as a building block in various other applications like semantic segmentation [93, 184] or scene recognition [127].

In recent years, the progress in this field has been tremendous. Popular detection approaches can be roughly subdivided into four different strands: First, the works based on the rigid template detector of Dalal and Triggs [36] using HOG features and SVMs. In particular, the Deformable Parts Model [53] extends the rigid detector with a part-based multi-component model and held the state-of-the-art in standard object detection benchmarks over many years. Second, object detectors building on the work of Viola and Jones [186] using Boosting and various feature channels [14, 41, 42]. These rigid detectors are less flexible in terms of object outlines, but achieve state-of-the-art results on pedestrian detection benchmarks and typically run in real-time, e.g., [41]. Third, detection models operating over small patches that vote for object centers with the generalized Hough transform [62, 100]. These detectors are very flexible and powerful in detecting body parts [163] or fiducial points in faces [37] but are less successful on common object detection benchmarks. Finally, a very recent trend in object detection builds on the combination of category-independent object proposals [2, 31, 180, 206], strong image representations and powerful machine learning methods. Girshick et al. [66] currently holds the state-of-the-art in this field. The method uses generic object proposals from [180] and a deep convolutional neural network pre-trained on ImageNet [92].

In this chapter, we show how Alternating Decision and Regression Forests (ADRF) can be employed for two different object detection approaches that rely on Random Forests (RF) as their base learning algorithm. We start with a method based on local evidence [62, 100] in Section 4.1, which belongs to the third group mentioned above. We present the details of this object detection approach, show how *ADRF* can be easily integrated into this framework, and review related work on this particular topic. In Section 4.2, we show how *ADRF* can also be successfully employed for the second of the above summarized groups. Moreover, we illustrate how the general *RF* concept can be utilized in order to predict flexible aspect ratios of the bounding boxes in order to outline objects more accurately.

4.1 Object Detection based on Local Evidence

The main idea behind object detection based on local evidence is that an object is described by a large set of small parts (not necessarily with semantic meaning) that are connected in a star-shaped model. These small parts of an object are often called patches. The seminal work on the Implicit Shape Model (ISM) presented by Leibe et al. [100] popularized this form of object detection.

The benefits of this approach are clear: The highly local model allows for a much richer representation of the object compared to a single template (e.g., [36]) or a multi-component model (e.g., [53]). Furthermore, the local patches that are only connected by by a star-shaped model can also handle articulations better than a rigid template. Finally, occlusions are also naturally handled.

Five years after the *ISM* was presented, Gall and Lempitsky [62] revived the research of patch-based object detection with the introduction of the popular Hough Forests (HF) framework. *HF* is a special instance of the *ISM* that yields more accurate results and is considerably faster during both training and inference. We will review these object detection principles in the following section in more detail. An exhaustive investigation of this topic can be found in [195].

4.1.1 The Implicit Shape Model and Hough Forests

In this section, we give a detailed description of the *HF* object detection framework and show how the idea of *ADRF* can be easily incorporated. We will focus on a general review of *HF* as it can be considered a modern version of *ISM*. Nevertheless, we still point out the explicit differences between these two models when appropriate.

4.1.1.1 The Object Model

As in standard object detection tasks, we are given a set of images $\mathcal{I}$ with bounding box annotation. We always consider detecting a single object category at once. Multiple categories would simply require multiple detection models, although other possibilities

exist as well [135]. Thus, each instance of an object category is annotated with a bounding box $\mathcal{B} = [t, l, h, w]$ in the training set, where t and l define the top-left corner of $\mathcal{B}$ and h and w the height and width, respectively, cf., Figure 4.1. These bounding boxes capturing instances of the desired object category are often called positive training examples.

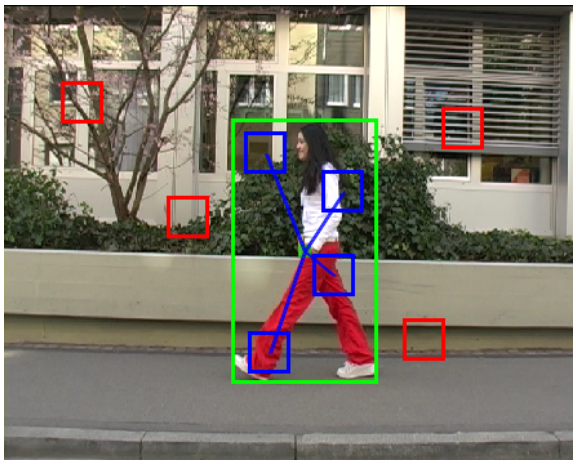


Figure 4.1: This figure illustrates the star-shaped model of *HF*. Patches within the green ground truth bounding box of the object define the object model. These patches are colored blue and are connected to a common reference point, which is the center of the bounding box in this case. Together with patches extracted from the background (red), they forms the training data of *HF*. Foreground patches store the offset vector from the patch to the reference point. Note that negative patches do not have any additional information.

To acquire a training set, all positive instances are extracted from the training images with the given bounding boxes. The training samples are all resized to a common width or height in order to ensure a fixed-size model. For pedestrian detection, the height of the training samples is typically fixed to $h = 100$ pixel. As the aspect ratio of the bounding boxes, i.e., $\frac{w}{h}$, is typically kept constant, the width varies for each training instance. Additionally, a negative training set that does not contain the desired object category is collected from the images.

At this point, the locality of the patch-based detection paradigm comes into play. From each of the training instances a set of small patches are extracted alongside with an offset vector to a reference point of the object (only for positive instances), e.g., the center of the bounding box $\mathcal{B}$. Thus, all patches are connected to a single reference point, which defines the star-shaped object model of *ISM* and *HF*, see Figure 4.1. For patches from negative training instances, no offset vector is defined. The size of the patches depends on the model size as they should capture a reasonable amount of information from the object. Thus, for a model height of $h = 100$ pixel the size of the patches is often set around 16×16 pixel [62]. Formally, we define a training patch as $\mathbf{P_n} = \{\mathbf{x}, \mathbf{y}, \mathbf{t}\}$, where $\mathtt{j} = 1, \ldots, \mathtt{N}$ and $\mathtt{N}$ is the total number of patches extracted from the training images. The appearance, i.e., the feature representation, of $\mathbf{P}$ is denoted $\mathbf{x} \in \mathbb{R}^{p_h \times p_w \times c}$, where p_h, p_w, and c denote

the height, the width, and the number of computed feature channels, respectively. These feature channels include the LUV color channels, first and second order derivatives of the gray-scale image and oriented gradients (similar to HOG [36]). Each patch is extracted from either the positive ($\mathbf{y} = 1$) or the negative ($\mathbf{y} = 0$) training instances, i.e., $\mathbf{y} \in \{0, 1\}$. Finally, all positive patches are equipped with an offset vector $\mathbf{t} \in \mathbb{Z}^2$ pointing from the center of the patch to the reference point of the object category. As mentioned before, $\mathbf{t}$ is undefined for negative patches. While for *ISM* the patches are extracted at sparse keypoint locations (e.g., keypoints returned by the Harris detector [73]), for *HF* the patches are extracted randomly with a uniform distribution. The process of generating the training data for this object detection model is illustrated in Figure 4.1.

This set of patches can already be used as an object detection model. Given an unseen test image $\mathcal{I}$, all patches are extracted and matched to all training patches to find the nearest neighbors. Then, each patch can 'vote' for an object reference point with the offset vector associated with the matched patch from the training set. However, this set of training patches is typically too large for efficient nearest neighbor matching. Therefore, a compact codebook of patch-prototypes is learned, similar to a bag-of-words representation [168], which then defines the final object model.

4.1.1.2 Learning the Codebook

Learning the codebook is one of the main differences between *ISM* [100] and *HF* [62]. *ISM* learns a flat codebook via k-means clustering, which only considers the appearance $\mathbf{x}$ of the patches $\mathbf{P}$. *HF* on the other hand builds the codebook with a discriminatively trained *RF*, which considers both the appearance $\mathbf{x}$ and the offset vectors $\mathbf{t}$, i.e., the structure of the star-shaped model. In general, *HF* employ the standard *RF* framework to discriminatively train the codebook, where each leaf node then corresponds to a codebook entry. However, *HF* shows a few subtle differences compared with the standard training protocol of *RF* described in Section 2.2.4.2.

First and most importantly, *HF* deals with a classification problem (positive and negative patches) as well as a regression problem (offset vectors $\mathbf{t}$ for positive patches). Thus, the quality function for finding good splits in the trees is formulated accordingly as a joint classification and regression task via

$$Q_{HF}(\sigma(\mathbf{P};\boldsymbol{\Theta}),\mathbf{X}) = \lambda(\gamma)\cdot Q_C(\sigma(\mathbf{P};\boldsymbol{\Theta}),\mathbf{X}) + (1-\lambda(\gamma))\cdot Q_R(\sigma(\mathbf{P};\boldsymbol{\Theta}),\mathbf{X})\ . \tag{4.1}$$

The classification objective Q_C employs the Shannon entropy from Equation (2.21) as compactness measure. For the regression objective Q_R, Gall and Lempitsky [62] employ the reduction-in-variance compactness measure from Equation (2.24). The variable $\lambda(\gamma)$ controls the influence of the different tasks and depends on a user-specified parameter γ. In [62], $\lambda(\gamma)$ is defined as a binary random variable that is sampled from a uniform distribution for each single node separately. That is, each node randomly selects either the classification or the regression objective, which has computational benefits as only a

single objective has to be evaluated per node. The last few levels of the trees are treated in a special way because $\lambda(\gamma)$ is fixed to 0, thus focusing solely on the regression part. For a node $\mathtt{j}$ in *HF* [62], this steering parameter can thus be formally defined as

$$\lambda(\gamma) = \begin{cases} 0 & \text{if } \delta(\mathtt{j}) \geq \gamma \\ \mathcal{U}\{0,1\} & \text{otherwise} \end{cases} , \tag{4.2}$$

where γ defines a certain depth of the tree, $\delta(\mathtt{j})$ is the depth of node $\mathtt{j}$, and $\mathcal{U}\{0,1\}$ defines the discrete uniform distribution for the two values 0 and 1. At this point, we also mention the work of Okada [122] that appeared simultaneously with [62] and presents an almost identical approach compared to *HF*. One difference is the variable $\lambda(\gamma)$ that is allowed to take values between 0 and 1 and is made dependent on the class purity of the current node.

In contrast to standard *RF*, *HF* also differs in the response function $\xi(\cdot;\boldsymbol{\Theta})$ (see Section 2.2.4.2) that is adapted to the specific feature representation used for object detection. Gall and Lempitsky [62] employ pixel pair differences on a single feature channel. The response function is thus defined as

$$\xi_{HF}(\mathbf{x};\boldsymbol{\Theta}) = \mathbf{x}_{\Theta_c}[\Theta_1] - \mathbf{x}_{\Theta_c}[\Theta_2] - \Theta_{\text{th}} , \tag{4.3}$$

where $\Theta_c \in \{1,\ldots,c\}$ selects a single feature channel and $\Theta_{\{1,2\}} \in \mathbb{Z}^2$ are two pixel locations.

The last main difference to standard *RF* are the leaf node models, i.e., the codebook entries. The ratio of positive examples as well as all offset vectors $\mathbf{t}$ that fall into this leaf during training are stored. This can be seen as a non-parametric regression model in the leaf nodes.

4.1.1.3 Incorporating Alternating Random Forests into Hough Forests

As in standard *RF*, also *HF* optimize the trees independently from each other. That is, the entropy (for classification) and the reduction in variance (for regression) are optimized only on the node level, i.e., locally, without regarding a global loss function. This results in the same disadvantages as for standard *RF* mentioned in Chapter 3. We thus apply the learning principle of *ADRF* to *HF*.

HF optimizes two objectives, classification and regression. Each node randomly selects one of the two objective functions that should be optimized to find a good splitting function for this particular node in the tree. Thus, only a fraction of the nodes operate in the classification mode while the other part optimizes the regression objective. Fortunately, this fact does not influence the *ADRF* training principle. We simply combine Alternating Decision Forests (ADF) and Alternating Regression Forests (ARF) into a single model.

For classification, each patch $\mathbf{P}_{\mathtt{n}}$ is thus assigned a weight $w_{\mathtt{n}}$. This weight is always updated after training a single stage of the classifier according to a given global loss

function $\mathcal{L}(\cdot)$, as described in Sec. 3.2. The possibility that all nodes in one stage do not optimize the classification but rather the regression objective is not optimal in the sense that gradients (or weights) do not influence the splitting functions. For the classification objective, this would somehow be a lost stage during tree growing. However, we can still evaluate the predictions about the class labels in each single stage of the algorithm and continue the *ADF* training.

For regression, we have to take care of the non-parametric leaf node model that is used in *HF*, i.e., all voting vectors from the training set are stored in the leafs. In order to evaluate the loss for the regression and compute the gradients, we have to have a single prediction for each offset $\mathbf{t_n}$ from the training set. In each leaf (or intermediate leaf during training), we have to compute a mode of the offset vectors falling into the corresponding leaf. A simple way to do this is to compute the mean offset in each leaf node and also the mean over several trees, as successfully demonstrated by Girshick et al. [67] for pose estimation. In [156], we followed such an approach and replaced the non-parametric model with a simple mean. To handle possibly occurring multi-modal distributions in the leaf nodes, we simply trained the trees a few levels deeper [156]. However, this leads to longer training times and decreased the performance in some preliminary experiments compared to a non-parametric distribution of voting vectors. In this thesis, we thus follow a different approach of integrating *ARF* into *HF*. We keep the non-parametric distribution of voting vectors in the final leaf nodes of the model, but simultaneously compute the mean of this distribution. This approximation, i.e., the mean, is then used to evaluate the loss function and to compute gradients. For the regression case, we also found a regularization of the *ARF* training scheme to work particularly well. We used a slightly different variant of Equation (3.6) to sum up the intermediate predictions over the levels of depth, which is defined as

$$p_{\text{ch}}^{\text{HF}}(\mathbf{y}|\mathbf{x}) = p_{\text{ch}}(\mathbf{y}|\mathbf{x}) + \xi \cdot p_{\text{pa}}(\mathbf{y}|\mathbf{x}) \; . \tag{4.4}$$

In contrast to Equation (3.6), we now have included the regularizing factor ξ to the second term, the prediction of the parent node. Throughout all our experiments, we use $\xi = 0.1$, which proved to work well for different data sets. One can regard this regularization similar to the shrinkage factor in Gradient Boosting (GB) and standard *ARF*.

The inference process of *HF* and both of these versions of integrating *ARF* are equal and follow the generalized Hough voting scheme. Each patch in a test image votes for tentative object centers in a Hough image, where local maxima indicate detected objects [62]. The only difference can be that leaf nodes only store a single offset vector (the mean of the training data) instead of having a non-parametric model, which is only the case for our first variant, cf., [156].

4.1.1.4 Detecting Objects via Generalized Hough Voting

The above described *RF* model can be used for localizing instances of an object category by applying the generalized Hough transform. In the following, we describe this process

for a single object category, i.e., $\mathtt{C} = 2$, and a single image scale. Detecting instances at multiple scales is easily done by apply the detector in a scale-space pyramid and searching for maximal scoring detections along this additional scale dimension (beside the spatial dimensions). A detailed derivation for the multi-class case can be found in [64].

During training, we kept one dimension of the bounding box at a fixed scale for each of the training image. For testing, we assume the bounding box to be fixed with size $\mathtt{W} \times \mathtt{H}$. Many object detection approaches make this assumption of a fixed bounding box size, e.g., [36, 41, 42]. Given an image $\mathcal{I}$, a ready-to-use *HF*, and the above assumption, the goal is to find the centroids of the bounding boxes $\mathcal{B}$ of each object instance. Ultimately, we are interested in the random event $E(\mathbf{p})$ for each pixel $\mathbf{p} \in \mathcal{I}$ of being the centroid of an object instance. Patches $\mathbf{P}(\mathbf{q}) = \{\mathbf{x}(\mathbf{q}), \mathbf{y}(\mathbf{q}), \mathbf{t}(\mathbf{q})\}$ extracted at each each location $\mathbf{q}$ provide the local evidence for objects centered at pixel $\mathbf{p}$. Note that $\mathbf{y}(\mathbf{q})$ and $\mathbf{t}(\mathbf{q})$ are latent variables and only $\mathbf{x}(\mathbf{q})$ is observed during testing. Thus, we define the probability $p(E(\mathbf{p})|\mathbf{x}(\mathbf{q}))$, which models the contribution of the local evidence at pixel $\mathbf{q}$ to the existence of an object $E(\mathbf{p})$ at pixel $\mathbf{p}$ based on the star-shaped model underlying *HF* and *ISM*.

As in [62], we only consider the contribution from patches $\mathbf{P}(\mathbf{q})$ located within the bounding box $\mathcal{B}$ of size $\mathtt{W} \times \mathtt{H}$ centered at location $\mathbf{p}$. This implies $\mathbf{y}(\mathbf{q}) = 1$ and results in

$$
\begin{aligned}
p(E(\mathbf{p})|\mathbf{x}(\mathbf{q})) &= p(E(\mathbf{p}), \mathbf{y}(\mathbf{q}) = 1|\mathbf{x}(\mathbf{q})) && (4.5)\\
&= p(E(\mathbf{p})|\mathbf{y}(\mathbf{q}) = 1, \mathbf{x}(\mathbf{q})) \cdot p(\mathbf{y}(\mathbf{q}) = 1|\mathbf{x}(\mathbf{q})) && (4.6)\\
&= p(\mathbf{t}(\mathbf{q}) = \mathbf{q} - \mathbf{p}|\mathbf{y}(\mathbf{q}) = 1, \mathbf{x}(\mathbf{q})) \cdot p(\mathbf{y}(\mathbf{q}) = 1|\mathbf{x}(\mathbf{q})) \; . && (4.7)
\end{aligned}
$$

As also mentioned in [62], the assumption $\mathbf{y}(\mathbf{q}) = 1$ restricts long-term interactions which could influence the existence of an object instance $E(\mathbf{p})$. For instance, a sidewalk can give local evidence for the existence of pedestrians in a typical street scene.

Nevertheless, we can now estimate both factors from Equation (4.7) by evaluating the trained *HF*, which we denote $\mathcal{F} = \{\mathcal{T}_\mathtt{t}\}_{\mathtt{t}=1}^{\mathtt{T}}$. Let us assume that the local evidence $\mathbf{x}(\mathbf{q})$ ends up in leaf $\mathtt{j}$ of tree $\mathcal{T}_\mathtt{t}$ of the *HF*. Then, the latter term in Equation (4.7) is simply the class probability for the positive class stored in this particular leaf node $\mathtt{j}$. The former term corresponds to the regression part of *HF* and depends on the chosen leaf node models. *HF* employs a non-parametric prediction model in each leaf node $\mathtt{j}$ and store all offset vectors $T_\mathtt{j} = \{\mathbf{t}_i^\mathtt{j}\}$ falling into this leaf during training. Thus, [62] employs a Parzen window estimate with Gaussian kernels based on $T_\mathtt{j}$ to compute $p(\mathbf{t}(\mathbf{q}) = \mathbf{q} - \mathbf{p}|\mathbf{y}(\mathbf{q}) = 1, \mathbf{x}(\mathbf{q}))$. We can now write Equation (4.5) for a single tree $\mathcal{T}_\mathtt{t}$ as

$$
\begin{aligned}
p_\mathtt{t}(E(\mathbf{p})|\mathbf{x}(\mathbf{q})) &= p(\mathbf{t}(\mathbf{q}) = \mathbf{q} - \mathbf{p}|\mathbf{y}(\mathbf{q}) = 1, \mathbf{x}(\mathbf{q}); \mathcal{T}_\mathtt{t}) \cdot p(\mathbf{y}(\mathbf{q}) = 1|\mathbf{x}(\mathbf{q}); \mathcal{T}_\mathtt{t}) && (4.8)\\
&= \left[\frac{1}{|T_\mathtt{j}|} \sum_{\mathbf{t}_i^\mathtt{j} \in T_\mathtt{j}} \frac{1}{2\pi\sigma^2} \exp\left(-\frac{\|(\mathbf{q} - \mathbf{p}) - \mathbf{t}_i^\mathtt{j}\|^2}{2\sigma^2} \right) \right] \cdot p(\mathbf{y}(\mathbf{q}) = 1|\mathbf{x}(\mathbf{q}); \mathcal{T}_\mathtt{t}) \; , && (4.9)
\end{aligned}
$$

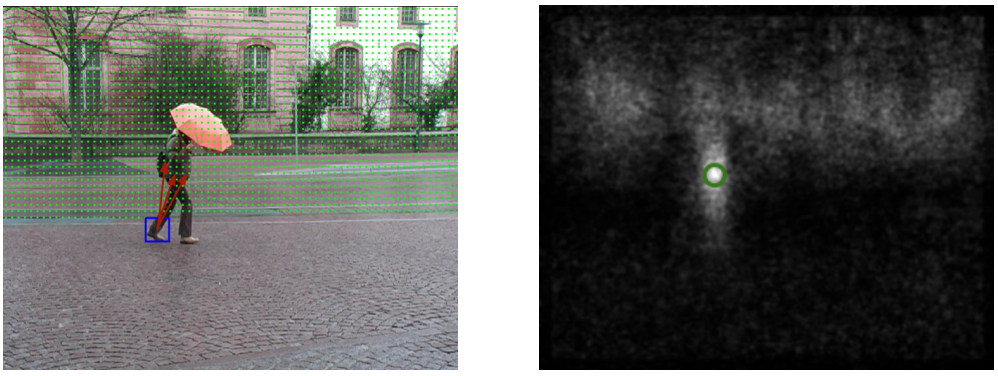
where $\mathbf{t}_i^{\mathtt{j}} \in T_{\mathtt{j}}$ define the means and $\sigma^2 \cdot \mathtt{I}$ the covariance of each 2 dimensional Gaussian kernel. The parameter σ^2 is defined by the user. To compute the final probability of $E(\mathbf{p})$ for a pixel $\mathbf{p}$ based on a single local evidence $\mathbf{x}(\mathbf{q})$ at pixel $\mathbf{q}$, we simply average (4.8) over all trees

$$p(E(\mathbf{p})|\mathbf{x}(\mathbf{q})) = \frac{1}{\mathtt{T}} \sum_{\mathtt{t}=1}^{\mathtt{T}} p_{\mathtt{t}}(E(\mathbf{p})|\mathbf{x}(\mathbf{q})) \ . \tag{4.10}$$

Following [62], we can finally integrate the contribution from all pixels $\mathbf{q} \in \mathcal{B}(\mathbf{p})$ within the bounding box centered at $\mathbf{p}$ to get the final non-probabilistic score

$$S(\mathbf{p}) = \sum_{\mathbf{q} \in \mathcal{B}(\mathbf{p})} p(E(\mathbf{p})|\mathbf{x}(\mathbf{q})) \tag{4.11}$$

at pixel $\mathbf{p}$. The higher the score, the higher the evidence for an object instance to be located at pixel $\mathbf{p} \in \mathcal{I}$.



(a) **(b)**

Figure 4.2: This figure illustrates the inference process of *HF* to localize objects in a given image. (a) A sliding window approach is used to evaluate all patches in the image. The green dots illustrate the location of patches that were already considered for voting. Please note that this is typically done for each pixel in practice. The blue rectangle shows the current patch under consideration and the red arrows are returned by the *HF* and used to vote for object centers. In this example, a patch of a foot is detected and the *HF* returns voting vectors that point upwards. (b) The resulting Hough map has local peaks that should correspond to object detections. In particular it should correspond to the location of the reference point of an object, i.e., the center of a bounding box. In this case, the detection is accurate, which is indicated with the green circle around the highest peak in the image.

Computing the score (4.11) via the above described derivation is quite inefficient because the local evidence for many pixels $\mathbf{q}$ have to be re-computed. A more efficient variant can be implemented as follows. First, the score map is initialized as: $\forall \mathbf{p} \in \mathcal{I} : S(\mathbf{p}) = 0$. Then, each tree $\mathcal{T}_{\mathtt{t}} \in \mathcal{F}$ is evaluated for each pixel $\mathbf{p} \in \mathcal{I}$ returning a class probability $p(\mathbf{y}(\mathbf{p}) = 1|\mathbf{x}(\mathbf{p}); \mathcal{T}_{\mathtt{t}})$ and a set of offset vectors $T_{\mathtt{j}}^{\mathtt{t}} = \{\mathbf{t}_i^{\mathtt{j}}\}$. Given this information the

score map can be updated as

$$\forall \mathbf{p} \in \mathcal{I},\ \ \forall \mathcal{T}_\mathtt{t} \in \mathcal{F},\ \ \forall \mathbf{t}_i^\mathtt{j} \in T_\mathtt{j}^\mathtt{t} :\ \ S(\mathbf{p}+\mathbf{t}_i^\mathtt{j}) = S(\mathbf{p}+\mathbf{t}_i^\mathtt{j}) + \frac{1}{\mathtt{T}} p(\mathbf{y}(\mathbf{p})=1|\mathbf{x}(\mathbf{p}); \mathcal{T}_\mathtt{t}) \ . \quad (4.12)$$

Finally, the score map $S(\cdot)$ is smoothed with a Gaussian. Figure 4.2 illustrates the voting process of *HF*.

4.1.2 Related Work

Before evaluating the *HF* framework and the presented extensions on object detection benchmarks, we briefly recapitulate the impact of *HF* in the field of computer vision.

The original paper by Gall and Lempitsky [62] presents *HF* for the task of object detection. Many extensions have been presented for this particular task based on *HF*. Razavi et al. [135] presents a multi-class detection approach that directly exploits the inherent multi-class capabilities of *RF*. This approach scales sublinearly with the number of classes compared to a standard 1-vs-all or 1-vs-1 approach.

As already mentioned previously, the codebook in *HF* is trained discriminatively. However, the score of a single object detection is the sum of all its part detections that vote to a common point in the 2D image plane. The part detections, i.e., the votes from the codebook, are obtained independently from each other. Thus, the final score for an object detection system is not trained discriminatively. Maji and Malik [107] as well as Wohlhart et al. [197] attack this issue. [107] builds upon the older *ISM* and learns discriminative weights only for each codebook entry but not for each single voting vector. Recall that a codebook entry typically contains more than a single voting vector from the training data. Furthermore, the learned weights are constrained to be positive. On the other hand, Wohlhart et al. [197] builds upon the *HF* framework. Discriminative weights are learned for each voting vector from the training set and can also become negative. The resulting Hough maps are cleaner and the number of false-positive detections is reduced.

Lehmann et al. [98] also learns discriminative weights for each codebook entry in *ISM*. However, the weights are again constrained to be positive. On the other hand, the authors also show the interesting equivalence of the patch-based voting principles (*ISM*, *HF*) and linear holistic models, e.g., [36].

Another obvious and interesting application that can also be addressed with *HF* is visual object tracking in a tracking-by-detection setting. Gall et al. [63] were the first to extend *HF* for the task of tracking. They used an offline pre-trained *HF* for the general object category of interest. That is, the object category has to be known a priori. Then, the *HF* is online adapted to specific instances of the same object category by updating the statistics in the leaf nodes.

In contrast, we presented an online tracking approach that is able to track a priori unknown instances [155]. The target has to be annotated with a bounding box only in the first frame of the video sequence. We combined the *HF* framework with online

random forests [145] for the task of tracking. Godec et al. [70] presented another tracking method for non-rigid objects based on *HF*. Although they do not build upon the online random forest [145], they present an online adaptable tracker that can also be trained from a single annotation without knowing the object category beforehand. [70] builds the trees completely random to their full size and just updates the leaf node statistics during tracking. Furthermore, this extension includes back-projecting the voting elements [136] to get a rough segmentation of the tracked object based on GrabCut [142]. The segmentation is then used to improve the sequential updates of the tracking algorithm [70].

In general, *HF* can also be considered as a joint classification and regression model that operates on smaller local patches that independently vote for a common prediction. Realizing this fact enables many other computer vision applications to exploit the general *HF* framework. In the following, we review a brief excerpt of recent literature.

A very successful application building upon *HF* is pose estimation of humans, human heads, and hands from either RGB or depth images. Girshick et al. [67] extends the seminal human pose estimation approach of Shotton et al. [163]. Based on the *HF* principle, an accurate regression approach for a set of pre-defined human joints (e.g., torso, elbows, hands, knees, etc.) is employed. While [67] uses depth images as input, Dantone et al. [38] present a similar pose estimation approach from RGB images. Fanelli et al. [48–50] present a pose estimation method for the human head. Given a single depth image, the task is to identify the location of the head in 3D as well as the orientation. This is rendered as a 6 dimensional pose estimation problem again with the *HF* framework. In this thesis, we also work on this particular pose estimation problem and delve into this topic in Chapter 5.

All these pose estimation approaches are highly useful for human computer interaction applications, which become more and more important. The 'smart home', i.e., controlling the lights, the heating, or the television automatically or with some simple gestures, is the current focus of many companies. These gestures can often be recognized with computer vision technology. The human hand is the most obvious part of the body to perform such gestures. Thus, many computer vision approaches for this particular task have been presented in recent years, with quite some success. For instance, the work of Tang et al. [175] presents an accurate human hand pose estimation approach that also builds upon a voting-based approach like *HF*.

Dantone et al. [37] and Cootes et al. [32] use *HF* for facial landmark detection. Given an image capturing a human face from an arbitrary viewpoint (frontal to profile), the task is to accurately localize a set of pre-defined facial landmark points (e.g., mouth and eye corners, etc.). In general, both approaches extend the regression part of the *HF* framework from a 2 dimensional (vote for a single reference point of the object) to a $k \cdot 2$ dimensional one (k is the number of facial landmarks). Furthermore, Dantone et al. [37] condition *HF* on the viewpoint of the face in order to make better predictions for the landmarks. On the other hand, Cootes et al. [32] combine the predictions given by *HF* with a statistical shape model, yielding a more accurate and physically valid localization of the facial landmarks.

	TUD-pedestrian	TUD-campus	TUD-crossing	ETHZ-cars
# train images	400 (3200)	-	-	420
# test images	250	71	201	175

Table 4.1: The number of train and test images for four different data sets we use in our evaluation of object detectors based on local evidence. TUD-campus and TUD-crossing data sets use the training images from TUD-pedestrian. For the TUD-pedestrian data set, we augment the training set with slightly jittered versions, resulting in 3200 images, as done in [195].

4.1.3 Experiments

In this section, we evaluate the influence of our proposed *ADF* and *ARF* training schemes on the *HF* framework for object detection. We split these evaluations into two separate parts due to the fact that *ARF* requires special handling of the *HF* framework as described in Section 4.1.1.3. To recap, the non-parametric distribution of voting vectors in the leaf nodes has to be handled. The first part thus only evaluates different variants of *ADF* and compares with *HF*. These evaluations are similar to the ones presented in [158]. In the second part, we evaluate *ARF*, where we adapt the framework accordingly (see Section 4.1.1.3). Before we come to the main experiments, we first describe the experimental setup.

4.1.3.1 Experimental Setup

For both experiments, which we describe in the following two sections, we use four different data sets: the TUD-pedestrian, TUD-campus, TUD-crossing data sets from [8], and the ETHZ-cars data set from [99]. These are three pedestrian and one car detection data sets, respectively. The TUD-campus and TUD-crossing data sets only contain images for evaluation and the detection models are supposed to be trained with the training images of the TUD-pedestrian data set. The number of train and test images for each data set can be seen in Table 4.1. The set of training images is actually doubled as we add 'negative' images not containing the target object. All the provided training images from the respective data set contain the target object. Figure 4.3 gives examples of the test images from each data set.

We use the same general random forests settings for all our experiments whenever possible. We employ 10 trees, a maximum tree depth of 15, 20000 randomly sampled splitting functions (pixel-pair tests) per node, and 5000 randomly selected training samples per node for split optimization. For the evaluation we plot precision-recall curves and show the area-under-curve (AUC) values. As we operate with randomly trained detectors as well as randomly selected training patches (cf., Section 4.1.1.2), we average the results of several independent runs. We thus average over two independent sets of training patches. For each training set, we again average over two independent training phases of the *RF* models, resulting in four averaging runs in total. As precision-recall curves are complicated to average, we present our results in two different ways: First, we show the mean and

Figure 4.3: Examples of test images from the four data sets we use in our experimental evaluation of object detectors based on local evidence. Each column illustrates three different test images from the `TUD-pedestrian`, `TUD-campus`, `TUD-crossing`, and `ETHZ-cars` data sets, respectively.

standard deviation of the AUC values for each data set and evaluated method. Second, we plot one of the four precision-recall curves for each method and data set that is closest to the mean AUC value. While the first variant gives more accurate estimates of the final performance of each method over several independent runs, the second one (the precision-recall curve) shows the behavior of the learned object detectors in more detail. In the following experiments we always compare with *HF*, which serves as our baseline. In our papers [156, 158], where we conducted similar experiments with slightly different settings and implementations, we also compared with a boosted *HF* implementation, which performs similar to our *ADF* and *ARF* variants but takes significantly more time to train. We omit these results here due to the long training procedure and because preliminary results indicated the same outcome and interpretation of results as in the papers.

4.1.3.2 Evaluation of Alternating Decision Forests

In this section, we evaluate the performance of the *ADF* training scheme in the *HF* framework. While *HF* solve a joint classification and regression problem, we only investigate the impact of the modification of the classification part with *ADF*. As in [62], we evaluate the regression objective exclusively starting with depth 13. We compare plain *HF* to *ADF* employing three different loss functions: savage, exponential, and tangent.

The resulting AUC values are depicted as mean and standard deviation in Figure 4.4 for all four data sets. The figure shows that by choosing the right loss function, *ADF* can outperform the baseline for all data sets. While the performance difference is not significant for the `TUD-crossing` data set, it still is for the other three. The performance boost is most pronounced for the `ETHZ-cars` data set, where all loss functions outperform the baseline. Figure 4.5 shows the corresponding precision-recall curves for all data sets. As mentioned above, we plot that curve that has the closest AUC value to the corresponding mean value for each method. These curves still give a more detailed insight into the behavior of the different methods. For instance, we clearly see in Figure 4.5a that all variants of *ADF* achieve higher precision and that *HF* gets higher recall. For the `TUD-campus` data set, we see in Figure 4.5c that the *ADF* variants achieve higher recall.

4.1.3.3 Evaluation of Alternating Regression Forests

In this section, we investigate the performance of the *ARF* training scheme in this object detection setup. As already described in Section 4.1.1.3, *ARF* requires a different handling in the *HF* framework. The reason is that we need a single prediction (i.e., voting vector) per leaf node in order to be able to evaluate the loss function and compute gradients. The standard *HF* framework uses a non-parametric distribution of voting vectors in the leaf nodes, i.e., all vectors are used for voting. In this thesis, we simultaneously store the mean voting vector for *ARF* but still keep the non-parametric distribution in the final leaf nodes. For this experiment, we compare the *HF* and *ADF* baselines with *ARF* and *ADRF* (i.e., *ADF* and *ARF* combined).

As in the previous experiment, we show the mean and standard deviation of the AUC scores from the corresponding precision-recall curves. Recap that we did four independent runs of these experiments. Figure 4.6 illustrates our results. We can observe that *ARF* and *ADRF* significantly boost the performance for two of the data sets, `TUD-crossing` and `TUD-campus`. Compared with the setup in [156], this variant of integrating *ARF* into *HF* cannot boost the performance on the `TUD-pedestrian` data set but the overall performance is higher. We can also see that the combination of *ADF* and *ARF*, i.e., *ADRF* does not provide an additional performance boost, except for the `ETHZ-cars` data set. Figure 4.7 shows the corresponding precision-recall curves. One can clearly observe the performance boost that is due to the *ARF* training scheme.

4.1.4 Discussion

To close the topic on object detection based on local evidence, we briefly want to discuss the outcome of the experiments from Section 4.1.3. We did two separate experiments in Sections 4.1.3.2 and 4.1.3.3, respectively, to investigate the integration of *ADF* and *ARF* into *HF*. In the first setup, it clearly pays off to integrate the *ADF* training scheme into the *HF* framework. For the second setup, special treatment of *HF* is required to integrate *ARF*. None of the two variants proposed in Section 4.1.1.3 is optimal. Both make an

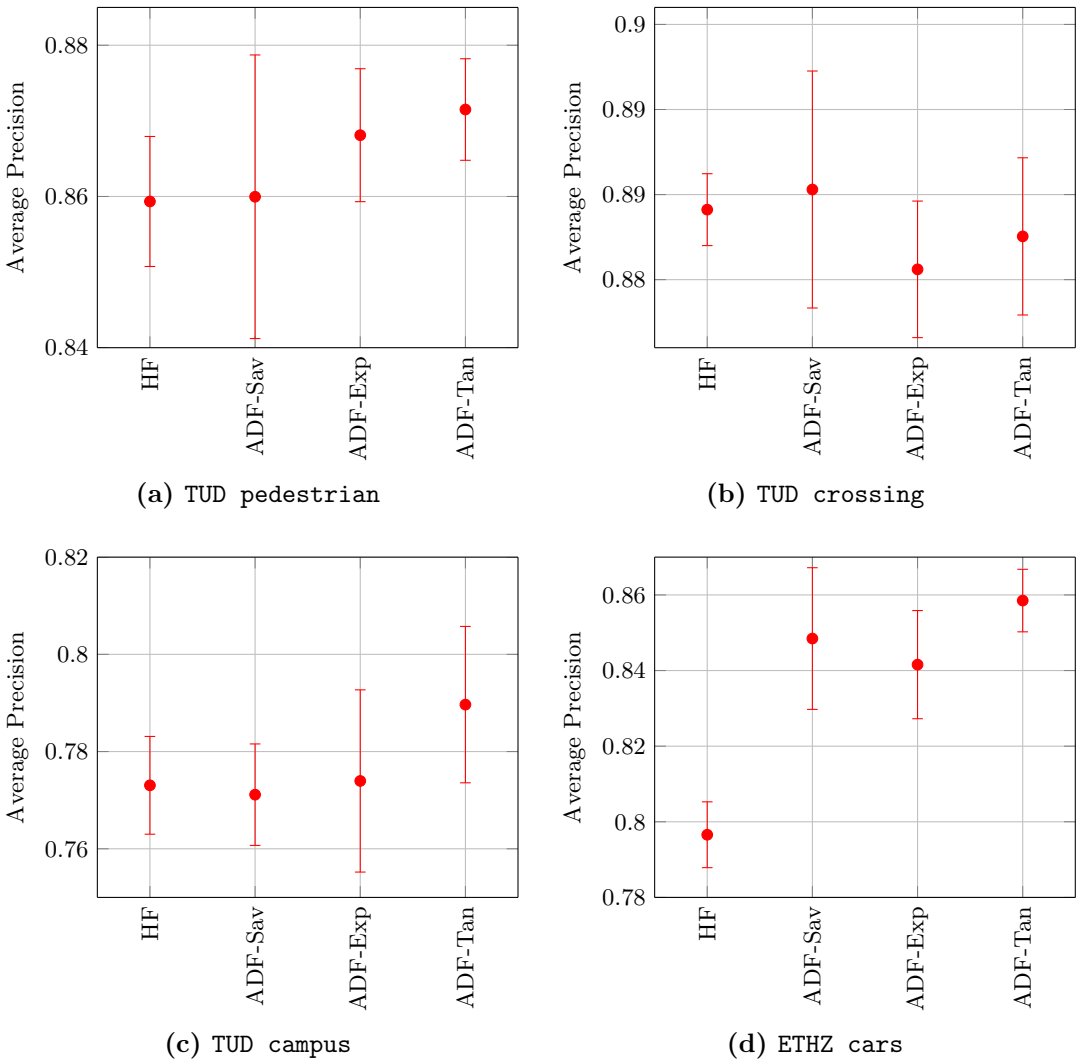
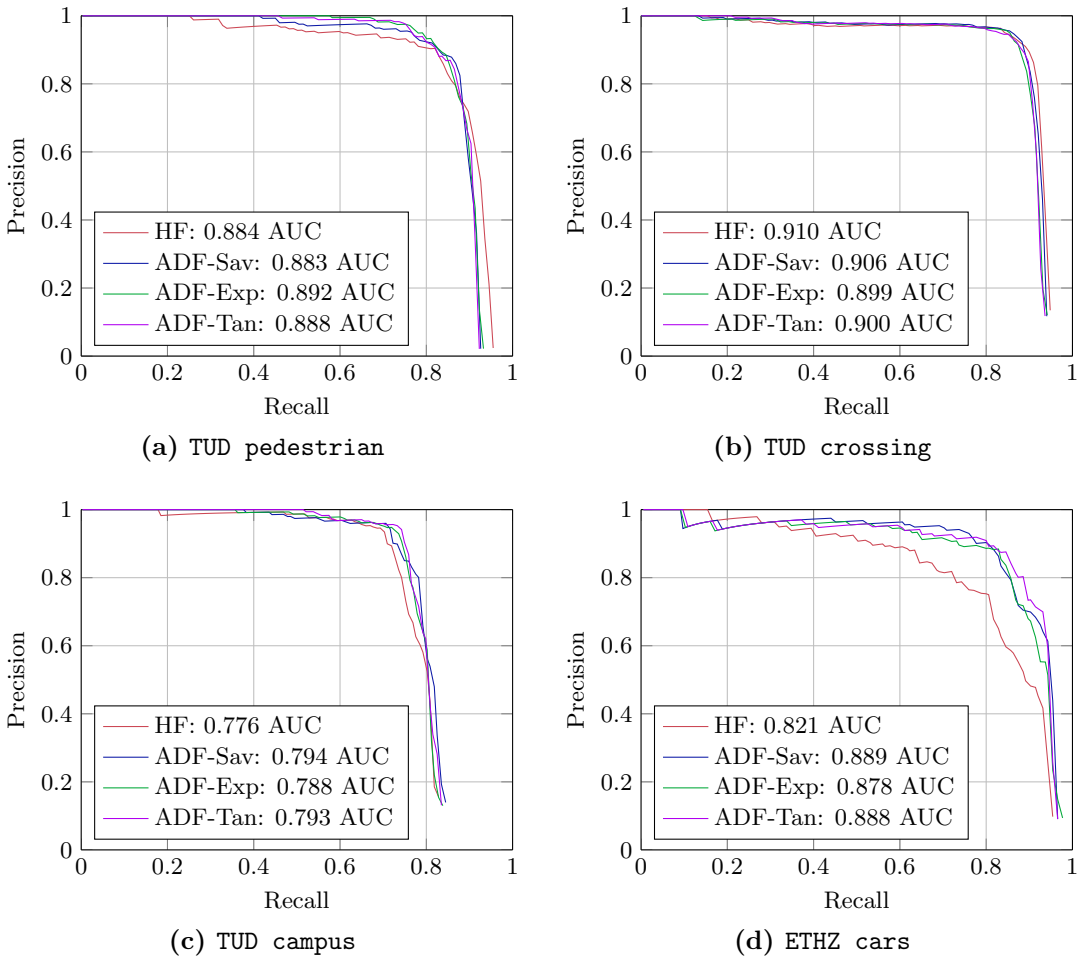


Figure 4.4: This figure illustrates the mean and standard deviation of AUC values (from the corresponding precision-recall curves) for four different data sets. We compare three variants of our *ADF* (different loss functions are employed, savage (Sav), exponential (Exp), and tangent (Tan)) to the *HF* baseline.

approximation of the non-parametric voting distribution, one only at the intermediate nodes and the other one additionally at the final leaf node level. In any case, the full potential of *ARF* cannot be exploited. A better way of training would definitely be to integrate the full Hough voting process into the loss evaluation, which was successfully done in a very recent work of Redondo-Cabrera and Lopez-Sastre [137].

Compared to template-based [36, 41, 42, 53, 157] or proposal-based [66, 180, 191] detection approaches, one has to challenge the whole object detection paradigm based on local evidence, anyway. In the second part of this chapter, we already see that a simple

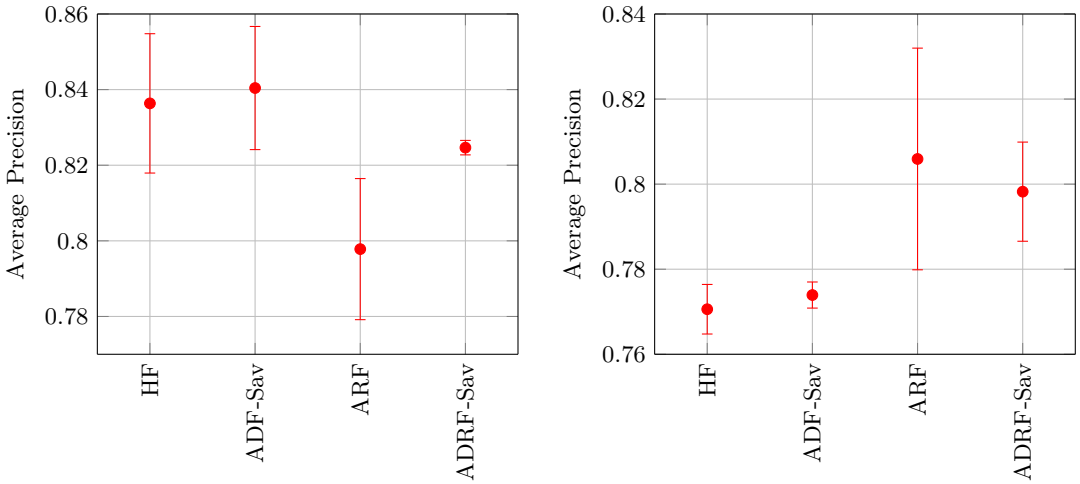


(a) `TUD pedestrian` **(b)** `TUD crossing`

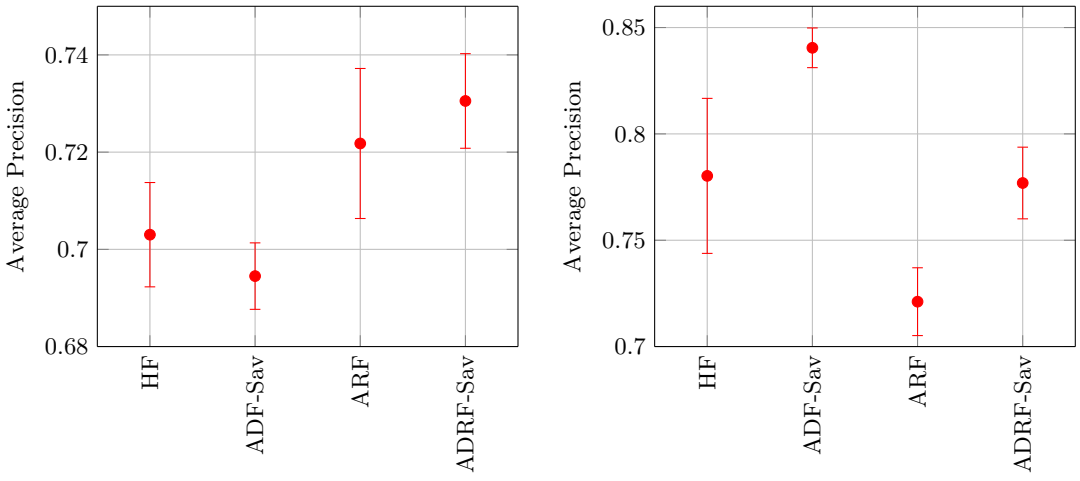
(c) `TUD campus` **(d)** `ETHZ cars`

Figure 4.5: This figure illustrates precision-recall curves of *HF* and three variants of *ADF* (different loss functions are employed, savage (Sav), exponential (Exp), and tangent (Tan)) for four data sets.

template-based approach [157] can easily outperform the *HF* framework. The concept of *ADF* can also be integrated into this template-based detection approach, as we will see in the following sections. However, *HF*-like approaches still show their justification in many other computer vision applications like pose estimation [67, 129, 175].

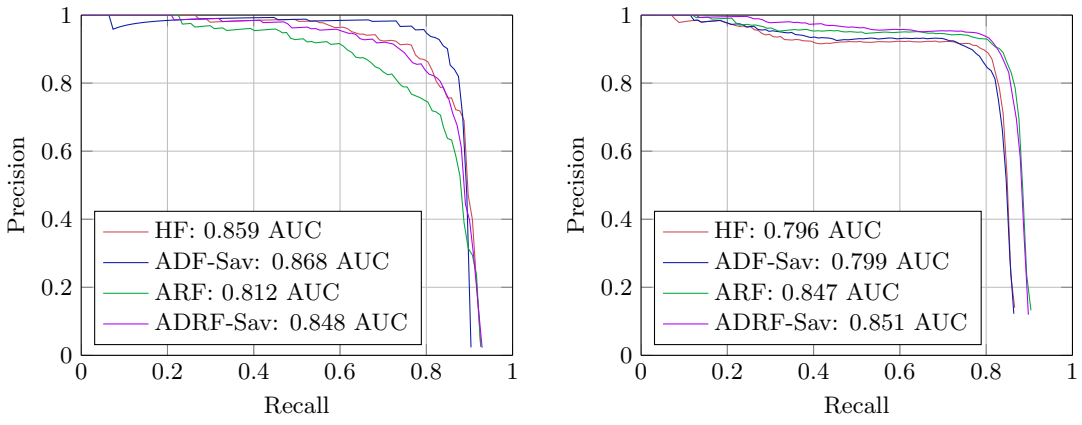


(a) `TUD pedestrian` **(b)** `TUD crossing`



(c) `TUD campus` **(d)** `ETHZ cars`

Figure 4.6: This figure illustrates the mean and standard deviation of AUC values (from the corresponding precision-recall curves) for four different data sets. We compare the baseline (*HF*) with *ADF*, *ARF*, and *ADRF*. For *ADF* and *ADRF*, we use the savage loss. In contrast to [156], all the models here employ the standard non parametric leaf node model of *HF* [62].



(a) `TUD pedestrian` **(b)** `TUD crossing`

(c) `TUD campus` **(d)** `ETHZ cars`

Figure 4.7: This figure depicts precision-recall curves of *HF*, *ADF*, *ARF*, and *ADRF* for four data sets.

4.2 Holistic Object Detection with a Rigid Model

The second part of this chapter deals with holistic object detection where the model describes an instance of the desired object category as a rigid template. The general detection paradigm is rather simple: During the training phase, the model learns from positive and negative instances in order to discriminate the desired object category from background. During the testing phase, a sliding window on the full scale-space of the image is applied in order to locate object instances. Popular representatives for this category of object detectors are [14, 36, 41, 42, 185], which are typically designed and evaluated for pedestrian or face detection. These two semantic categories show a rather rigid shape, can be well modeled with such template-based approaches, and are two of the most important categories in many computer vision systems.

Most detection frameworks in this category build upon a gradient-based image representation, sometimes in combination with color features. The image representation is typically structured and takes the form of the object template. On top of this representation, a learning algorithm tries to discriminate the object from the background. Support Vector Machine (SVM) and Boosting (Boosting) are the two predominant machine learning approaches employed in this field. *SVM* is typically used with a linear kernel in order to ensure low computational costs during inference. On the other hand, *Boosting* offers a non-linear decision boundary and can be more expressive, while being competitive with *SVM* regarding computational costs.

In this thesis, we build on the work of Dollár et al. [41], which we review in more detail in Section 4.2.2. Instead of using *Boosting* as in [41], we show how *RF* and also *ADF* can be successfully employed. Moreover, we can exploit the flexibility of the general *RF* framework to make any kind of structured prediction [43, 62, 89], allowing for addressing a critical shortcoming common to most holistic object detection approaches: the fixed aspect ratio of the detections. We present a method capable of predicting more accurate bounding boxes with a joint classification and regression *RF* formulation similar to [62, 69]. The label space for object detection gets augmented with the bounding box size, which we exploit during both training and testing. In this way, we cannot only predict the foreground probability of a detection but can also regress the extent of the object with a single model, alleviating the need for learning many mixture models [53]. The detection pipeline and the models we describe here have already been published in our paper [157].

In our experiments, we demonstrate that this object detection approach yields state-of-the-art results on several data sets. We compare it with related approaches like *HF* [62] (see Section 4.1), the Deformable Parts Model (DPM) [53], and a Boosting-based rigid template approach [14, 41, 42]. We also evaluate the difference between plain *RF* and our *ADF* training scheme for this task. The superior results of *ADF* again confirm the effectiveness of this training algorithm. More importantly, we also show that our approach can accurately regress the bounding box aspect ratio of objects in unseen test images. To illustrate this in our experiments, we investigate the typical quantitative evaluation criteria

of object detection systems, which focus on the amount of true and false positive detections in a certain data set. True positives are commonly characterized by a predefined overlap (50% in most benchmarks) of the detection with the ground truth bounding box. However, the question arises if a criterion based on such an arbitrarily chosen threshold actually reflects the quality of an object detector. Consider for instance the blue bounding box in Figure 4.8, a detection given by a template-based detector similar to [14, 41, 42]. It is a true positive but has an overlap of only 59% with the green ground truth bounding box. In contrast, the red bounding box is the output of our detection algorithm, which has a much higher overlap of 89% with the ground truth, thus identifying the extent of the object more accurately. Our experiments show that the detection performance of most typical detectors breaks down when increasing this overlap criterion for true positives, while our approach still gives comparably good results.

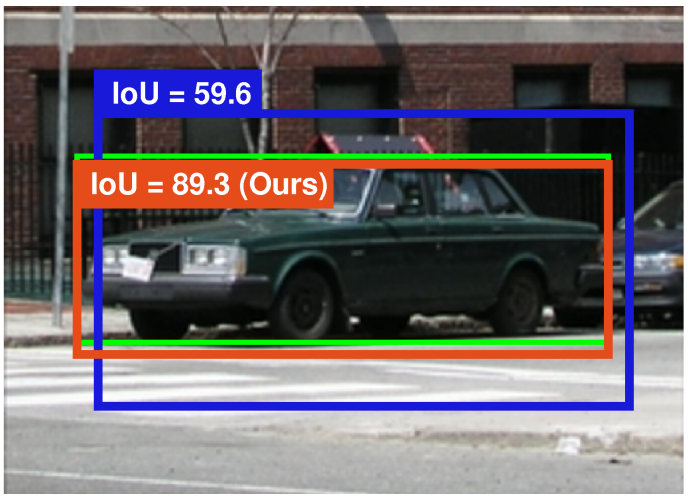


Figure 4.8: Illustrative output of the proposed detector (red) and a state-of-the-art method (blue). The given values show the overlap with the green ground truth bounding box, where our detector achieves a much higher overlap because it regresses the aspect ratio of the object.

4.2.1 Related Work

The most related approaches to ours are the works of Dollár et al. [41, 42] and Benenson et al. [14], who build on a similar detection pipeline and employ the same features. The influential Integral Channel Features (ICF) [42] compute several feature channels including color, gradient magnitude, and orientation quantized gradients, which is similar to [14]. Both works rely on an efficient *Boosting* framework for learning the object models. This line of work almost exclusively focuses on pedestrian detection and there are some extensions that make this framework extremely fast [13] or exploit application-specific knowledge [12]. However, only fixed size bounding boxes are predicted, which is reasonable for pedestrians but is a limitation if dealing with other objects.

The *DPM* [53] extends the rigid HOG template and *SVM* approach of [36] and includes deformable parts and multiple components. This mixture model captures the intra-class variability by separating the training data according to the aspect ratio (newer versions also include appearance), thus enabling the prediction of a discrete set of aspect ratios. Furthermore, a linear regression on the inferred part locations refines the aspect ratio prediction. Nevertheless, the *DPM* has some disadvantages because (i) different models have to be trained for each component and (ii) the aspect ratio prediction per component is limited on a linear model solely based on the part locations. In particular, the first issue limits the *DPM* in two ways: First, the model becomes slower during both training and testing, and second, the more components are employed, the less training data is available for each of them. In contrast, we have a single model that can exploit all the training data and can predict a continuous aspect ratio.

Blaschko and Lampert [19] formulate the detection as a regression problem and train a structured output *SVM* for learning. While yielding accurate results, a fast localization method is necessary to have a reasonable running time during both training and detection. They employ Efficient Subwindow Search [94], which requires computing an upper bound on the detection score, thus limiting the choice of features and learning method. Our approach is more flexible in the choice of features, faster during training and also has a reasonable runtime within a sliding window scheme.

We also note that *RF*, in general, have rarely been employed for object detection. One exception is the *HF* framework [62], which describes an object as a set of small patches that are connected to a reference point, typically the center of the object (see Section 4.1.1). However, this patch-based approach is relatively slow compared to other detection models. To overcome this issue, [174] proposes a two-level approach for speeding up the detection process. Nevertheless, it still relies on the Hough voting scheme for the final prediction, where the non maximum suppression is a delicate task, cf. [196]. While *HF* [62] typically predict a fixed bounding box, it can also handle variable aspect ratios: either via back-projecting the voting elements, which then define the bounding box, or via voting in a third dimension in Hough space. However, employing the back-projection is rather slow or memory intensive, while increasing the Hough space dimensionality hampers the maximum search.

Recently, [110] showed a holistic *RF* model that trains local experts (*SVM*) in each node. However, this model also builds on a fixed bounding box prediction and was only evaluated on pedestrian detection benchmarks.

4.2.2 The Object Detection Framework

Our object detection system builds on the framework of [41]. The training data is given as a set of rigid templates $\mathcal{P}$ with size $\bar{h} \times \bar{w}$. This size is the mean of the bounding box sizes of all the objects in the training set, normalized to 100 pixel width or height, depending on what is larger. For the purpose of a clean description, we will use the task of car detection

as an example and always refer to a fixed-width model throughout the rest of the chapter. As in [41], the size of $\mathcal{P}$ also includes a padding of 20% in order to capture some context around the objects. We center such a template at each of the annotated, properly rescaled training objects and on randomly sampled bounding boxes from negative images. For each of the cropped training examples, we calculate 10 different feature channels [41, 42]: We use the 3 LUV color channels, the gradient magnitude, and 6 gradient channels, quantized into equally sized orientation bins, similar to [14].

The training data $\mathbf{X}$, i.e., positive and negative samples, is thus given as a set of pairs $\{\mathbf{x}_\mathtt{n}, \mathbf{y}_\mathtt{n}\}_{\mathtt{n}=1}^{\mathtt{N}}$, where $\mathbf{x}_\mathtt{n} \in \mathbb{R}^{\bar{h}\times\bar{w}\times 10}$ and $\mathbf{y}_\mathtt{n} \in \{0, 1\}$. While [41, 42] employ *Boosting*, we train a *RF* $\mathcal{F} = \{\mathcal{T}_\mathtt{t}\}_{\mathtt{t}=1}^{\mathtt{T}}$ as described in Section 2.2.4.2 using the training set $\mathbf{X}$. We use the standard training objective given in Equation (2.20) and employ pixel-pair tests as splitting functions, given in Equation (4.3). Please note that in the following sections, we will extend the label space and also provide a different objective function for training the model.

The training phase also includes three rounds of bootstrapping after the initial training of the *RF*. In each round, a set of hard negative windows is identified by applying the current model on the negative images, which are then added to the pool of negative data [41]. In each round, we re-train the *RF* from scratch.

4.2.3 Predicting the Aspect Ratio

We now describe how the flexibility of *RF* can be utilized in order to make more accurate bounding box predictions with a joint classification and regression formulation. Before we present the training procedure in Section 4.2.3.2, we first show how the label space of the training problem is augmented with a regression target in the following section. Finally, in Section 4.2.3.3 we present the inference process where the *RF* is used to localize objects and to accurately predict their extent.

4.2.3.1 Augmenting the Label Space

As described in the previous section, each training example is cropped and scaled such that it fits in the template $\mathcal{P}$ of size $\bar{h} \times \bar{w}$. The scaling factor is defined by the fixed model width. Thus, the actual height of the objects captured in the training images most likely varies with the viewpoint of the object. See Figure 4.9 for some examples. In order to give predictions about the correct height, and thus the aspect ratio of an object, we additionally store the actual height z for each of the training examples.

Therefore, we augment the label space with the ground truth height of each positive training example, which extends the label space to $\mathcal{Y} = \{0, 1\} \times \mathbb{R}$. Our training set now becomes a set of triplets $\{\mathbf{x}_\mathtt{n}, \mathbf{y}_\mathtt{n}, z_\mathtt{n}\}_{\mathtt{n}=1}^{\mathtt{N}}$, where $\mathbf{y}_\mathtt{n} \in \{0, 1\}$ still corresponds to the positive and negative label, and $z_\mathtt{n} \in \mathbb{R}$ is the correct bounding box height. Please note that $z_\mathtt{n}$ corresponds to the object height only for the example of a fixed-width model. It would

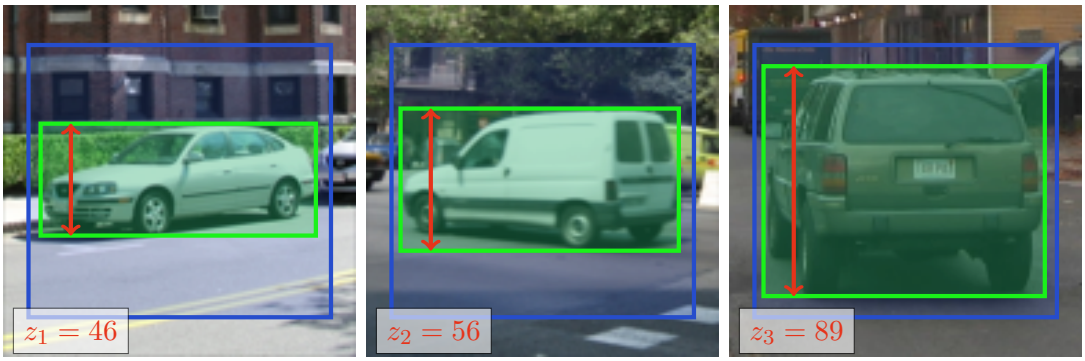


Figure 4.9: Three training examples capturing objects with different height and a common fixed width. The object representations (features) are computed from the whole template (blue box), but the regression targets z_n are different for each example. The green box indicates the ground truth annotation and also defines the regression target z_n (red arrow).

correspond to the width for a fixed-height model. Also note that for background training samples, i.e., $\mathbf{y}^n = 0$, z_n is undefined.

4.2.3.2 Training the Random Forest

For training our joint classification and regression *RF* formulation, we have two objectives. First, we want to separate positive and negative classes and, second, we want to regress the bounding box height for positive samples. Similar to *HF* [62], we use two separate split node evaluation criteria that optimize the different objective functions. The first evaluation criterion targets (binary) classification and the second one regression.

We can directly make use of the *HF* framework for this joint classification and regression problem. The reader is thus referred to Section 4.1.1.2 for a detailed review on the training principle of *HF*. As described in Section 4.1.1.3, we can also easily integrate the training principle of *ADF* into this framework. We will see in the experiments that *ADF* outperforms plain *RF* also in this holistic object detection setup.

In general, a single splitting node decides randomly with equal probability which of the two objectives are being optimized (see Section 4.1.1.2). As in *HF* [62], we also assign certain levels of depth in the tree a fixed type of evaluation objective that has to be optimized with the variable $\lambda(\gamma)$. In this setup, we slightly extend the parameter γ from Section 4.1.1.2, which now has different interpretations: While setting $\gamma = 0$ ignores the regression objective at all, setting $\gamma > 0$ indicates that starting with depth γ, only regression nodes are evaluated in all trees, similar to [62]. Here, we additionally allow setting $\gamma < 0$, where the first levels up to depth $|\gamma|$ are fixed to optimize the regression objective. The remaining levels of the tree are again randomly selected with a discrete uniform distribution.

Tree growing stops as in standard *RF* when either the maximum tree depth is reached or not enough training examples are available for further splitting. In contrast to standard

RF, where tree growing also stops if a certain node becomes pure in terms of class labels, in this setting, we continue splitting nodes containing only positives but fix the splitting objective to be regression. These are the same stopping criteria as for *HF* described in Section 4.1.1.2. The resulting leaf nodes then calculate (i) the class histogram based on the training data falling into that leaf and (ii) the mean of the regression targets z of all positive training examples. As each tree can thus return two kinds of outputs, we denote the classification output of tree $\mathtt{t}$ as $f_{\mathtt{t}}^{C}(\mathbf{x})$ and the regression output as $f_{\mathtt{t}}^{R}(\mathbf{x})$.

4.2.3.3 Detection with Aspect Ratio Regression

For detecting objects in unseen test images, we employ a standard sliding window approach over the scale-space. The score of a window $\mathcal{W}$ at location (x, y) in the image is given by the classification output $s = \mathcal{F}^{C}(\mathbf{x}) = \frac{1}{\mathtt{T}} \sum_{\mathtt{t}=1}^{\mathtt{T}} f_{\mathtt{t}}^{C}(\mathbf{x})$ of the *RF*. As we use an ensemble method which consists of several independent weak classifiers (randomized trees in this case), we could parallelize their evaluation to achieve a higher detection speed. However, given a certain detection threshold τ below which detections are discarded, we can also iteratively evaluate the trees and employ an early-stopping scheme. We can still benefit from parallel processing, e.g., at the scale space pyramid or for the simultaneous evaluation of multiple images.

In our early stopping approach, we examine whether or not the trees in the *RF* not evaluated up to now can theoretically achieve such a high foreground probability that the total score for that window $\mathcal{W}$ exceeds the detection threshold τ. Assume that we already evaluated the trees up to index $\mathtt{t} < \mathtt{T}$, the current unnormalized score thus is $s_{i \leq \mathtt{t}} = \sum_{i=1}^{\mathtt{t}} f_i^{C}(\mathbf{x})$. The upper bound of the score of the remaining trees is $\bar{s}_{i > \mathtt{t}} = \sum_{i=\mathtt{t}+1}^{\mathtt{T}} 1.0 = \mathtt{T} - \mathtt{t}$. Therefore, if

$$s_{i \leq \mathtt{t}} + \bar{s}_{i > \mathtt{t}} < \tau \cdot \mathtt{T} \tag{4.13}$$

we can already stop evaluating the feature vector $\mathbf{x}$ in the current window $\mathcal{W}$. For instance, having $\mathtt{T} = 10$ and a detection threshold $\tau = 0.95$, the evaluation can already stop if the first tree has a foreground probability $f_1^{C}(\mathbf{x}) < 0.5$. Using this approach, we can reject clear negative windows during the evaluation process very quickly without reducing detection performance.

For each window with $s \geq \tau$ we evaluate $f_{\mathtt{t}}^{R}(\mathbf{x})$ for all trees in the *RF* to return the prediction of the regression target, i.e., the height of the object captured in the current window. The final estimate z of the object height is given as the average over all independent trees:

$$z = \mathcal{F}^{R}(\mathbf{x}) = \frac{1}{\mathtt{T}} \sum_{\mathtt{t}=1}^{\mathtt{T}} f_{\mathtt{t}}^{R}(\mathbf{x}) \ . \tag{4.14}$$

Please note that a mode seeking approach like mean shift could also be employed, but averaging turned out to be a good choice in our setting. The resulting detection window

including the detection score in the original scale of the test image is then given by $\mathcal{D} = (\frac{x}{\kappa}, \frac{y}{\kappa}, \frac{w=100}{\kappa}, \frac{z}{\kappa}, s)$, where κ is the scale of the detection.

After having identified a set of potential detections for a test image, we apply a standard greedy non-maximum suppression approach that removes detections having an overlap greater than 50% with a higher scoring detection [53].

4.2.4 Experiments

In this section, we demonstrate the performance of our general object detection approach and evaluate the difference between plain *RF* and *ADF* in more detail. First, we compare with state-of-the-art methods on three different data sets with a standard object detection evaluation criterion. Second, we investigate the ability of the joint classification and regression *RF* formulation for making accurate bounding box predictions. We thus evaluate the detection performance of all methods when the evaluation criterion (i.e., the bounding box overlap with ground-truth) is tightened. Finally, we analyze our trained model and the most relevant parameters.

4.2.4.1 Overall Performance Evaluation

We first evaluate the overall detection performance of the proposed approach on three standard benchmarks. We investigate three different variants of our approach: *StdRF* implements a standard *RF* disregarding the regression information at all. *StdRF-Regr* trains a *RF* and includes the regression information during both training and testing. *ADF-Regr* employs the *ADF* training scheme for classification nodes, see Sections 4.1.1.3 and 3.2.3.

In addition, we give a comparison to state-of-the-art detection approaches. First, we evaluate Aggregate Channel Features (ACF) [41], a *Boosting*-based approach similar to [14, 42] that builds on the same detection pipeline (i.e., the same features and bootstrapping scheme) as our detector. Second, we compare with *HF* [62] that also rely on a *RF* framework for learning the object model, however, it works on the patch-level and employs the generalized Hough voting scheme for detection. Please note that both approaches only predict a single bounding box aspect ratio, which is averaged over the training data. Finally, we also compare with the *DPM* [53] (*DPMfull*), where we additionally evaluate a version that only uses the root filter (*DPMroot*) to have a fair comparison with the other approaches building on a rigid template. However, more important for our scenario are the multiple components included in *DPM*, which are defined by clustering the aspect ratio of the training bounding boxes. To denote the different versions of [53] we add the number of components (1, 2, or 4) as postfix to *DPMroot* or *DPMfull*, respectively. For all approaches building on randomization steps, i.e., *HF* [62] and our variants, we average the results over three independent runs. As the standard deviation is rather small in our experiments, we only report the average results for a cleaner presentation.

Data sets: We use three different data sets for evaluation purpose, namely the `ETHZcars` [99], the `TUDpedestrian` [8], and the `MITStreetsceneCars` [17]. For the `ETHZcars` data set, we use 420 training and 175 testing images that capture cars from different viewing angles. The test set defines a rather easy detection task because it shows cars prominently with little background. Nevertheless, it exactly fits our needs for evaluating the quality of the bounding box predictions as the aspect ratio of the ground truth bounding boxes strongly varies. The `TUDpedestrian` data set contains 400 training images and 250 test images. Also in this scenario, the aspect ratio varies due to different articulations. Finally, the `MITStreetsceneCars` is a larger data set capturing cars in different street scene scenarios and under different illumination. We split this data into 2/3 training and 1/3 testing images, resulting in 2909 and 1020 images, respectively.

Evaluation Criterion: The evaluation criterion in this experiment is the commonly used Pascal overlap [47]. For each detection $\mathcal{D}$ it calculates the overlap with a ground truth bounding box $\mathcal{G}$ as the intersection over union:

$$\mathrm{IoU}(\mathcal{D}, \mathcal{G}) = \frac{\mathcal{D} \cap \mathcal{G}}{\mathcal{D} \cup \mathcal{G}} \,. \tag{4.15}$$

The outcome of the function $\mathrm{IoU}(\mathcal{D}, \mathcal{G})$ separates all detections $\mathcal{D}$ into true ($\mathrm{IoU} \geq \xi$) or false ($\mathrm{IoU} < \xi$) positives, which are used for drawing precision-recall curves and calculating the Area Under Curve (AUC) measure. The parameter ξ is the success threshold that is typically (and also in this experiment) set to 0.5. Note that in the following section we evaluate the detection results when setting $\xi > 0.5$.

Results: We depict our results as precision-recall curves for all data sets in Figure 4.10 and report the *AUC* values in the corresponding legend. As can be seen, the proposed *ADF-Regr* is always en par with the best performing approach and clearly wins on one of the data sets. We also note that *ADF-Regr* is typically better than *ACF*, which is the most related detection approach. One exception is the `ETHZcars` data set where the difference between the two methods is insignificant and all results are rather saturated. On the `TUDpedestrian` and the `MITStreetsceneCars` data sets our approach is 11.7% and 7.7% better than *ACF*, respectively. Furthermore, our approach performs significantly better than *HF*, the only other *RF* based method. Finally, we also note that *ADF-Regr* outperforms all versions of *DPM* except for *DPMfull2* and *DPMfull4* on the `ETHZcars` data set. Moreover, including the parts in the *DPM* does not always improve the results on these data sets.

When comparing the different proposed variants, i.e., *StdRF*, *StdRF-Regr* and *ADF-Regr*, we see that including the regression output typically gives significantly better performance and that the *ADF* learning scheme (see Section 3.2.3) further improves the results.

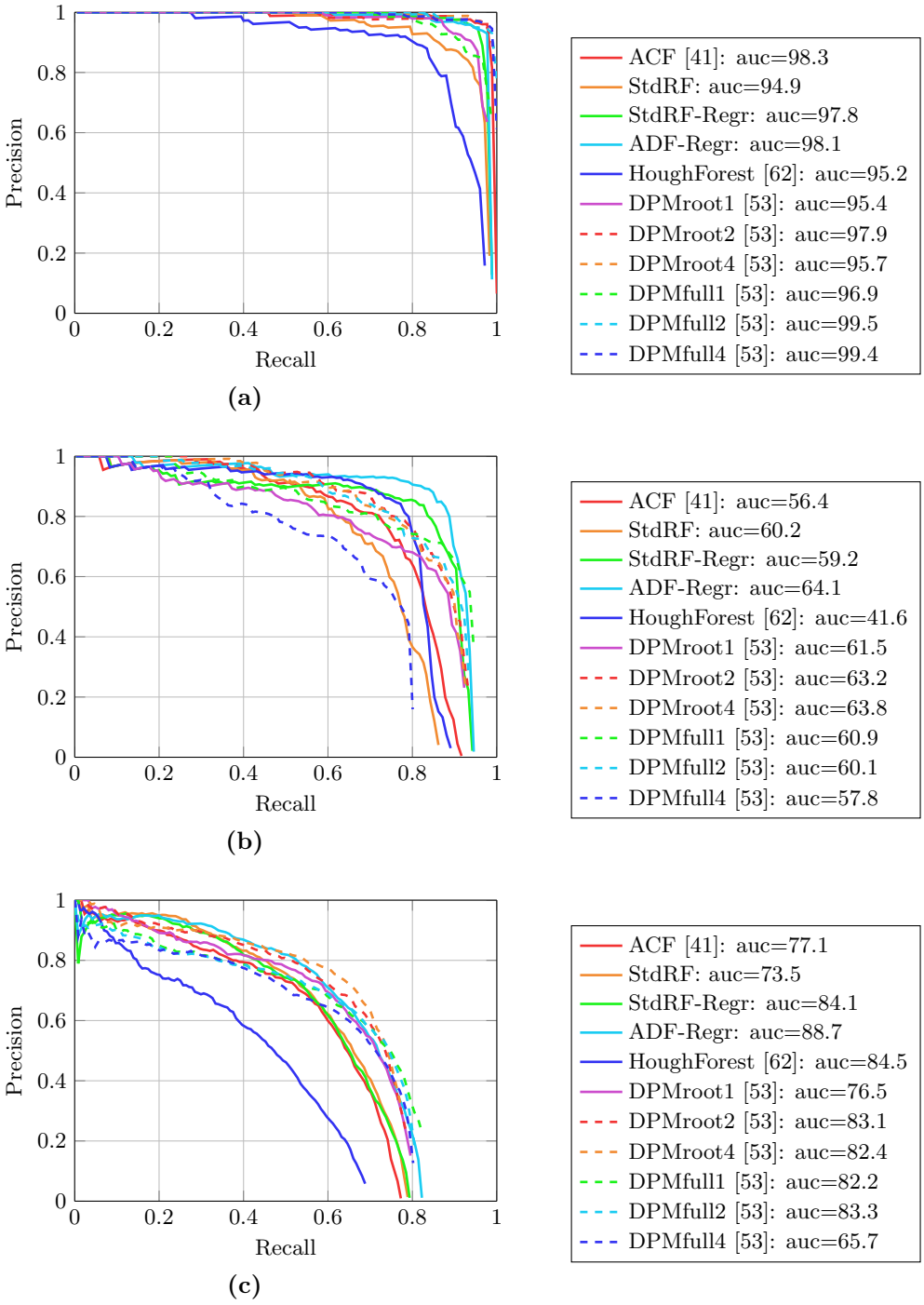


Figure 4.10: Precision recall curves for all evaluated approaches on three different data sets: (a) `ETHZcars`, (b) `TUDpedestrian`, and (c) `MITStreetsceneCars`.

4.2.4.2 Tightening the Pascal Overlap Criterion

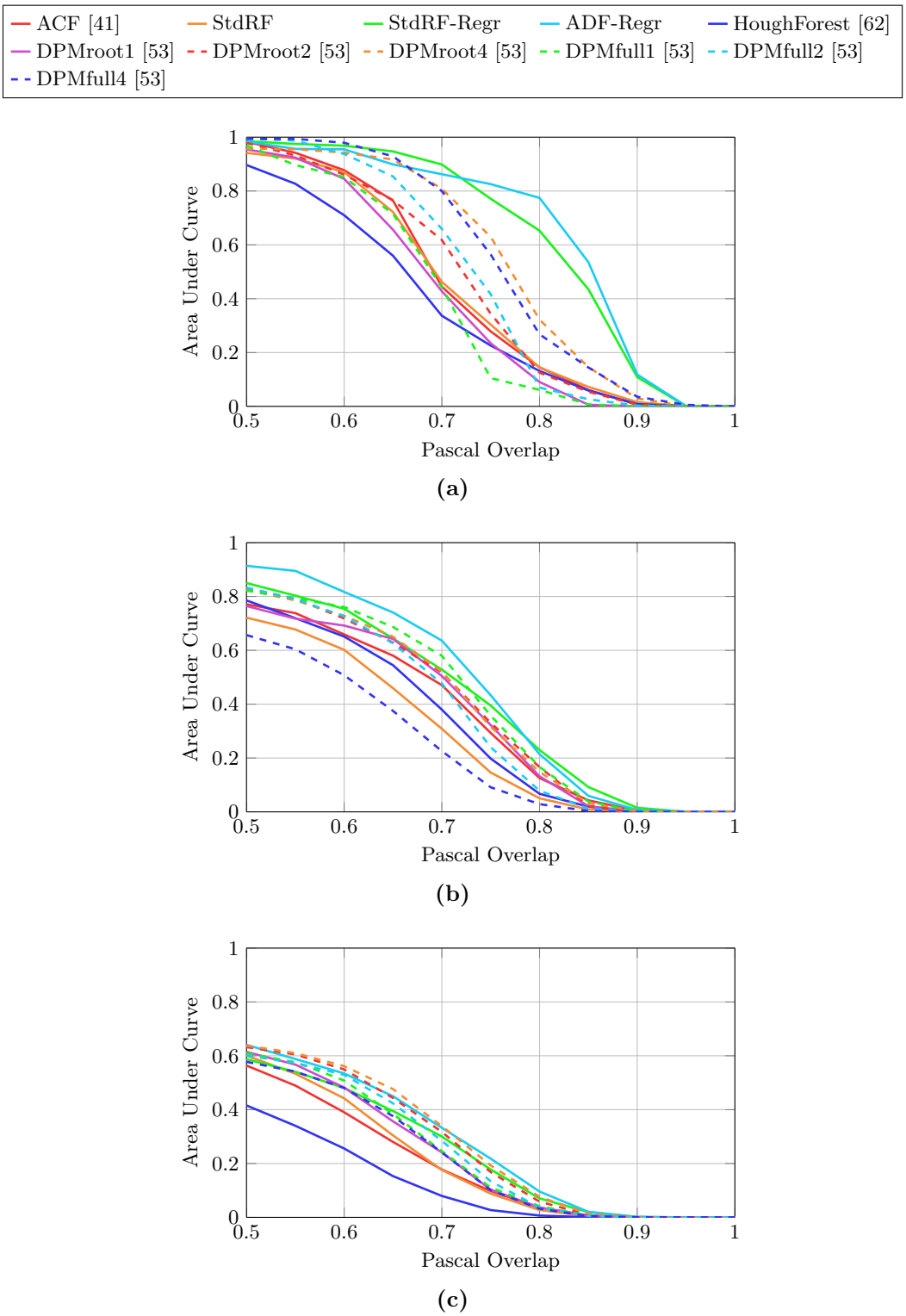
In this experiment, we directly evaluate the quality of the bounding box predictions of the different methods and investigate the performance of our joint classification-regression prediction in more detail. To do so, we use a different type of evaluation curve. Following the standard Pascal criterion, we draw precision-recall curves and measure the AUC. However, we vary the success threshold ξ for a true positive detection, see Equation 4.15. We then plot the AUC value over ξ in the range between 0.5 to 1.0. Apart from the evaluation criterion, the experimental setup is the same as in the previous experiment.

Results: We illustrate our results for all three data sets in Figure 4.11. As can be seen, the proposed *ADF-Regr* gives the best results on two benchmarks and is en par with *DPM* (2 or 4 components) on one benchmark. The performance difference between all methods is most pronounced on the `ETHZcars` data set (Figure 4.11a), which shows most aspect ratio variations. We can see that *HF* [62], *ACF* [41], *StdRF*, *DPMroot1* and also *DPMfull1* drastically lose performance if the success threshold ξ gets increased. For instance, when setting $\xi = 0.7$ none of these approaches achieve higher AUC than 50%. Using *DPM* [53] with 2 or 4 components (whether or not parts are included) increases the performance to 62% and 80%, respectively. This result is intuitive as increasing the number of components also increases the number of aspect ratios that can be predicted. However, further adding components will likely decrease the performance as less training examples will be available per component (can be observed on `TUDped` and `MITStreetsceneCars`). In contrast, our *RF* formulations, *StdRF-Regr* and *ADF-Regr*, can predict the bounding boxes even more accurately, improving over *DPMroot4* by 6% for $\xi = 0.7$ and by 41% for $\xi = 0.8$. Our best variant, *ADF-Regr* achieves an AUC value of 73% at the very tight success threshold of $\xi = 0.8$, while all methods predicting a fixed bounding box do not exceed an AUC value of 20%. We illustrate some qualitative results in Figure 4.12.

4.2.4.3 Analysis of the Random Forest Model

In this section, we analyze the most relevant parameters of our *RF* framework. We evaluate the number of trees `T`, as well as the parameter γ that regulates the amount of regression nodes evaluated during training (for *StdRF-Regr* and *ADF-Regr*). Furthermore, we also investigate the feature selection process of *RF* when including the regression objective during the training phase.

Number of trees: First, we evaluate the number of trees `T` when fixing the maximum tree depth to $\mathtt{D}_{\max} = 12$ on the `TUDpedestrian` data set. In this setting, the additional regression is also turned on and the parameter γ is set to -2. The results are depicted in Figure 4.13a. As expected, we can see a clear trend of increasing performance with the number of trees `T` up to a certain limit $\mathtt{T} = 100$. For more trees, the results are saturated.



(a)

(b)

(c)

Figure 4.11: Area under the Curve (AUC) for increasingly harder Intersection over Union success thresholds for different data sets: (a) `ETHZcars`, (b) `TUDpedestrian`, and (c) `MITStreetsceneCars`.

Figure 4.12: Example results when comparing a fixed bounding box prediction and a flexible one. The green box is the ground truth, the blue one is the fixed size bounding box, i.e., the mean over the training examples, and the red one is the flexible bounding box predicted with our joint classification and regression *RF*. The three right images in the bottom row show bad examples.

Regression-Only Parameter: We further analyze the parameter γ. For this evaluation, we use the *ETHZcars* data set, which provides the most variation in aspect ratios. In Figure 4.13b, we see the mean and standard deviation of 3 independent runs for different values of γ. We can observe that using too many or too few regression nodes is unfavorable. Best performance can be observed by setting γ either to -2 or 9 (when using trees with maximum depth of $\mathtt{D}_{\max} = 12$).

Feature Selection: We also visualize the spatial feature selection frequency for *StdRF-Regr* and *StdRF*, i.e., with or without including the regression objective, on the car model of the `ETHZcars` data set. To do so, we count how often a certain location in the rigid template was selected for performing a split in the *RF*, regardless of the feature channel. Again, we set $\gamma = -2$ for *StdRF-Regr*.

Figure 4.14 illustrates the behavior for both training schemes when summing up all selected feature locations. In this setting, we used 100 trees with a maximum tree depth of 12 over 2 independent runs. Interestingly, we can observe that the feature selection is very different for the two learning schemes. *StdRF* concentrates on positions on the left and right side of the cars. This behavior seems clear as the width of the model is

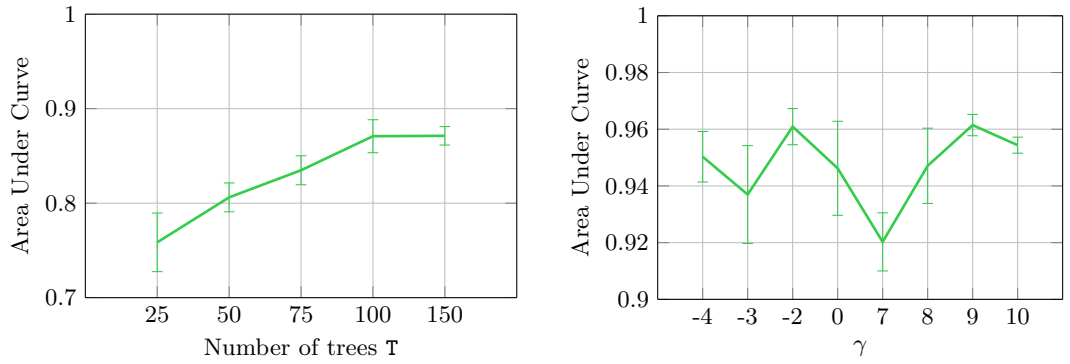


Figure 4.13: (a) Evaluation of the number of trees `T` on the `TUD-pedestrian` data set. (b) Evaluation of the parameter γ that influences the behavior of the regression objective in the *RF*. We plot the accuracy as *AUC* for different values of γ for the `ETHZcars` data set.

fixed and, thus, the gradients at these locations are present in all positive training images. Furthermore, for this training scheme, also the bottom of the templates are frequently selected. This can be explained by the fact that cars typically stand on the street and also have gradients on the bottom, which is typically not the case in negative images.

On the other hand, *StdRF-Regr* evaluates a much more diverse set of locations as it also tries to separate the different viewing angles, e.g., frontal-view from side-view cars. Thus, it selects the features at different y-coordinates in the center of the x-dimension. Of course, this training scheme also evaluates classification nodes. We can thus observe similarly selected locations as for *StdRF*.

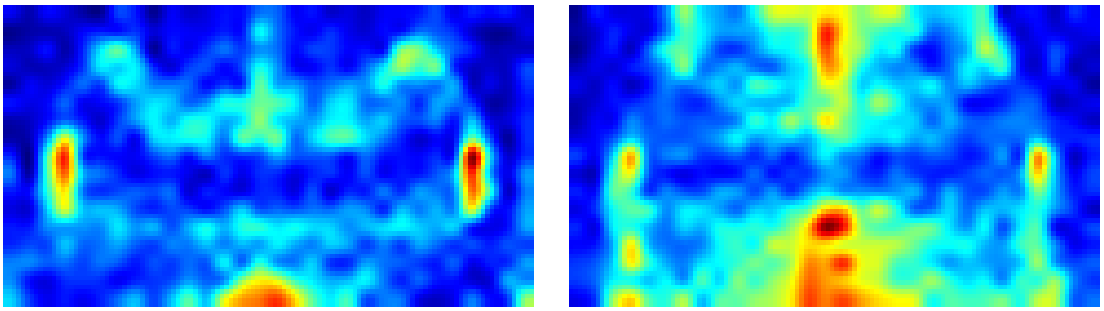


Figure 4.14: Spatial feature selection of both training schemes: (a) *StdRF*, where only the classification splitting criterion is active and (b) *StdRF-Regr*, which also includes the regression objective. Both figures show the spatial distribution of selected features of the car model for the `ETHZcars` data set. Red and blue colors indicate high and low frequencies of the feature selection, respectively.

Classification model: Finally, we evaluate if the regression objective during training also improves the classification performance of the *RF*. That is, we train two models: one including the regression objective ($\gamma = -2$) and one only evaluating classification nodes ($\gamma = 0$). During testing, however, we only use the fixed mean bounding box in order to turn off the effect of regressing the bounding box height on the final *AUC* performance. We observe that including regression during training improves the performance regardless if the regression output is actually used during detection. On the `TUDpedestrian` data set, we get $78.3 \pm 1.2\%$ *AUC* without using the regression objective ($\gamma = 0$) and $83.3 \pm 3.1\%$ when including it ($\gamma = -2$). Including the regression output during the testing phase for predicting variable bounding box aspect ratios further improves the results, cf., Section 4.2.4.1.

4.3 Summary

In this chapter, we applied the proposed *RF* training scheme from Section 3.2 for the task of generic object detection. We focused on two different schemes, a local patch-based framework as well as a holistic template-based approach. For both scenarios, the proposed *ADRF* could be successfully integrated.

The first part of this chapter deals with object detection based on local evidence, see Section 4.1. We review the ideas of *ISM* [100] and *HF* [62], where we provide details on the object detection model and how *ADRF* can be readily integrated. After describing the inference process of *HF* with the generalized Hough transform, we evaluate the effect of the newly integrated *RF* training scheme on the object detection performance. We can show that both the classification (see Section 3.2.3) and regression part (see Section 3.2.2) of our *ADRF* training scheme improves the results compared to the original *HF* formulation on standard benchmarks data sets.

The second part of this chapter deals with holistic object detection with a rigid template, see Section 4.2. Beside the integration of *ADF* into this framework, we propose a general *RF* based object detection model that is capable of predicting variable bounding box aspect ratios. Compared to the *HF* framework investigated in the first part of the chapter and several other state-of-the-art detection approaches, our method handles continuously varying aspect ratios in a single *RF* model. We augmented the standard binary label space accordingly with a regression target for predicting the bounding box aspect ratio. Our joint classification and regression formulation successfully exploits the additional label information during both training and testing. Our results on common object detection benchmarks showed that our proposed model is better or en par with related state-of-the-art approaches for standard evaluation criteria. Furthermore, our experiments revealed that most commonly used object detectors that predict a fixed bounding box size break down as soon as the evaluation criterion becomes tighter in terms of overlap with the ground truth bounding box. In contrast, our formulation can efficiently deal with variable aspect ratios in a single model and achieves good results even if the overlap criterion gets

harder. Finally, the *ADF* framework provides better scores compared with the plain *RF* training scheme also in the holistic object detection setup.

CHAPTER 5

Human Head Pose Estimation from Depth Data

Another computer vision application we address in this thesis is human head pose estimation from depth images. Given a depth image $\mathcal{D}$ capturing a human head, the task is to automatically infer the position in 3D and the orientation. The position is simply defined by the x, y, and z values (in millimeters) in the world coordinate system, assuming known intrinsic parameters of the camera. The orientation is modeled via Euler angles (yaw, pitch, and roll). The full pose of the head is thus defined as a 6 dimensional vector $\mathbf{h} = [x, y, z, \alpha, \beta, \gamma]^\top$ that has to be inferred for a given depth image. An illustration of the head orientation is illustrated in Figure 5.1 and some example depth images with ground truth annotation from [50] are shown in Figure 5.2.

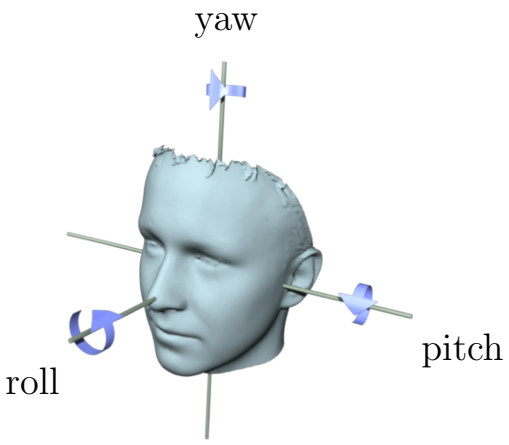


Figure 5.1: This figure illustrates the 3D annotation of the human head orientation with Euler angles, i.e., yaw, pitch, and roll. These three angles correspond to the last three values in the pose vector **h**. Image taken from [91]

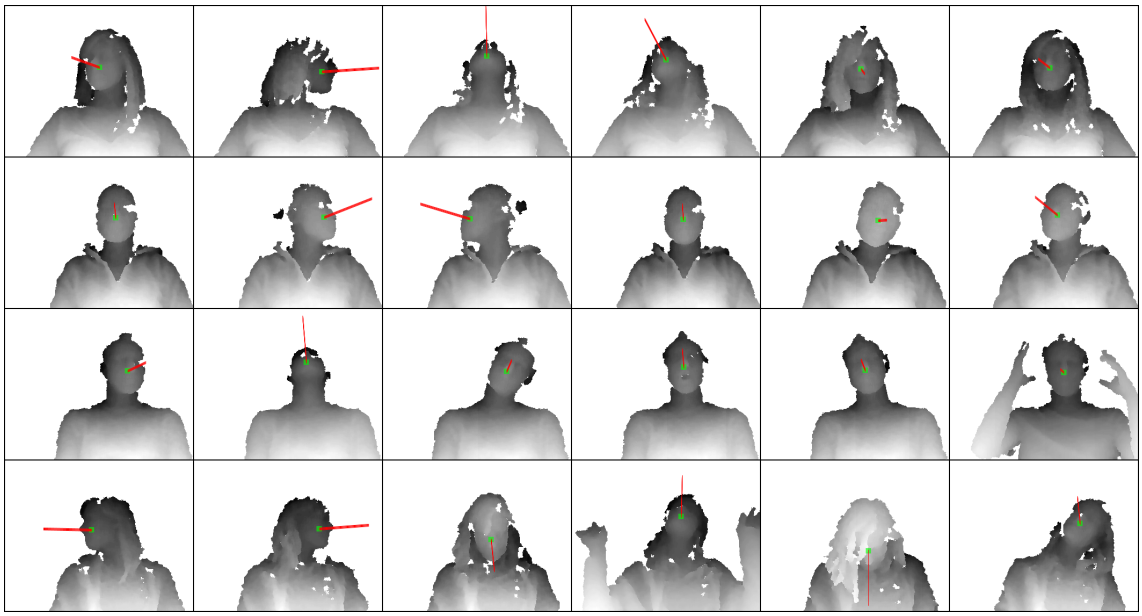


Figure 5.2: Exemplary depth images with corresponding head poses from the Biwi Kinect Head Pose Database [50]. Each row corresponds to one sequence capturing a single person. Each column shows different frames within that sequence. The green dot illustrates the head center location, which is projected to the 2D image space and overlayed on the depth image. The red lines indicates the viewing direction of the head.

Human head pose estimation is an essential component for applications like human behavior analysis or human computer interaction. These applications require an accurate estimate of the head pose in order to operate reliably. The advent of cheap depth sensors like the Microsoft® Kinect™ or other Time of Flight (ToF) sensors allow systems to rely on depth information in practical scenarios. Using 3D information resolves many ambiguities compared to a standard 2D image sensor. We review related work on human head pose estimation, including 2D and 3D attempts, in Section 5.1.

In this thesis, we evaluate two different approaches for human head pose estimation. First, the local patch-based approach of Fanelli et al. [49, 50], which extends the Hough Forests (HF) framework [62] for this particular task. As this model again builds upon the basic Random Forests (RF) framework, we can easily integrate the ideas of Alternating Decision and Regression Forests (ADRF) to learn better models that give more accurate predictions, as we already showed in [156]. In Section 5.2, we describe this framework presented by Fanelli et al. [49, 50] in more detail as well as our extensions to integrate *ADRF*. Second, we evaluate a new holistic approach for this task. Inspired by the better performance of a template-based object detection algorithm compared with *HF* (see Chapter 4), we also investigate a similar pipeline for head pose estimation, which we describe in Section 5.3 in more detail. The experimental evaluation of both approaches is summarized in Section 5.4.

5.1 Related Work on Human Head Pose Estimation

An accurate estimation of the human head pose is an important building block for many high-level applications, for instance, in the field of human-computer interactions. Researchers have worked on this topic for many years, used different sensing techniques, and applied many different methods. Thus, the body of literature is large for this particular topic. Murphy and Trivedi provide a thorough survey on head pose estimation in computer vision [119], which summarizes and compares 90 different works up to the year of 2009. Instead of building a functional taxonomy that would distinguish the methods by its operating domain, e.g., 2D or 3D methods, the authors choose to organize them based on their underlying fundamental approach, e.g., nonlinear regression, manifold embedding, or tracking-based approaches.

A lot has changed since 2009 and many new works have been presented. Recent years showed a strong trend towards 3D methods due to the increasing availability of cheap consumer 3D sensing technology like the Microsoft® Kinect™ or other *ToF* cameras. In this thesis, we give an up-to-date review of related work on human head pose estimation, which is based on Fanelli et al. [48]. We also build upon a functional taxonomy and separate related work into 2D and 3D approaches in the following two subsections, respectively.

5.1.1 2D Approaches

Approaches in this category only rely on 2D information from a single, standard camera. In contrast to 3D approaches, which we summarize in the following section, 2D approaches typically benefit from a cheap and easy to use sensing device. However, the advent of consumer depth cameras with acceptable prices and comfortable installation makes this benefit rather small. Nevertheless, according to [48], we also subdivide this category of 2D approaches further into appearance-based (holistic) and feature-based (local) approaches.

Appearance-based: Jones and Viola [85] discretize the space of head poses into n sets and build on a two-stage approach to estimate the head pose. First, a learned decision tree initially estimates the pose by predicting one of the n discrete poses for each window in the image. Then, to verify the detection, a pose-specific face detector is applied, which is based on their famous work on face detection with boosted classifiers [185]. The authors also try to first apply all pose-specific detectors and then select the best one to estimate the pose. While this gives slightly better results, the computational costs are obviously higher.

Other works use manifold learning and rely on the assumption that the head pose builds a low-dimensional manifold of the high-dimensional face images. Unlike previous approaches that did not exploit the pose labels for finding the embedding, Balasubramanian et al. [11] use this supervision to bias the embedding. They call this 'biased manifold embedding'. Osadchy et al. [124] define a parameterized low-dimensional head pose man-

ifold by hand. A convolutional neural network is trained to both detect faces and project the high-dimensional face image onto the manifold, where the head pose can be extracted.

Feature-based: This category of head pose estimation systems rely on the detection of facial features like the nose tip, eyes, or mouth corners. Whitehill and Movellan [194] present a head pose estimation system that relies on a fixed set of 4 facial features, the nose tip, the mouth center, and the two eye centers. After having detected the face and the facial features, the pose is first roughly classified with a set of detectors that are trained for certain ranges of the head pose. Finally, a linear regression takes the locations of the detected features as well as the output of the rough pose classification as input and computes the final pose estimate.

Instead of relying on a global set of facial features, Vatahska et al. [183] define subsets of features for different head poses. First, a face image is classified into a rough pose (frontal, left or right profile), which defines the currently active set of facial features. Then, AdaBoost [59] is applied to detect the features in the active set. Finally, a neural network is fed with the positions of the detected features to estimate the final pose.

Compared with appearance-based approaches, systems relying on the detection of facial features can be more robust to general occlusions and illumination changes, but as soon as a single feature cannot be recognized the systems easily break down.

5.1.2 3D Approaches

The most often occurring issues with approaches only relying on 2D information are partial occlusions, change of illumination, and the lack of features in rather textureless regions of the face. Furthermore, the manual annotation of 2D face images with an accurate pose is a hard task on its own. The annotation, however, is required for learning and evaluation. Fortunately, the increasing availability and the decreasing costs of depth sensors make it easier to include 3D information into the process of human head pose estimation. Most recent work thus focus on 3D-based approaches, either in combination with 2D information or with depth alone.

Seemann et al. [159] does not employ a direct 3D sensor, but extract 3D information from a stereo camera setup. The proposed method first detects and crops the face in the image. Then, after normalizing the extracted gray-scale and disparity patch, a neural network estimates the head pose. The system can estimate pan and tilt rotation from $-90°$ to $+90°$ and runs at 10 FPS on a 320×240 pixel image on a standard consumer PC from that time.

The approach presented by Breitenstein et al. [25] estimates the 3D pose of the human head from a single range image captured with a stereo-enhanced structured light method [192]. The system first detects the nose with a newly developed shape signature. Based on the estimate of the nose, head pose hypotheses are generated and a set of templates is rendered from a generic face model. The template with the minimal alignment

error to the observed input data is selected and defines the final pose estimate. This approach is real-time capable due to parallel computing on the GPU. It is also robust to large pose variations ($\pm 90°$ yaw, $\pm 45°$ pitch, and $\pm 30°$ roll rotations), facial expressions, and partial occlusions. However, the nose has to be visible in order to make a prediction, which is also the case for [108].

In contrast, the approach presented by Fanelli et al. [49] does not depend on the visibility of any facial feature and still can handle large variations of the head pose. The method employs a random regression forest to learn a mapping from a high-quality depth image patch to the head pose (6 degrees of freedom) in the spirit of *HF* [62]. Each patch is extracted from a depth image and can cast a vote for the 6 pose parameters (3D position and yaw, pitch, roll rotations). All votes are aggregated to make a final estimate. In contrast to [25], this approach reaches real-time performance without using a GPU.

Both [25] and [49] assume the head to be the only object present in the depth image. The follow-up work of Fanelli et al. [50] relaxes this assumption and integrates the face detection process into the random forest framework of [49] making it even more similar to [62]. Thus, this approach can handle situations where other parts of the body or even other objects are present in the captured scene. Furthermore, it does not rely on high-quality range images, but can deal with rather low-quality depth images from a consumer sensor like the Microsoft® Kinect™ . In the following section, we review [50] in more detail as it builds the basis for one of our attempts to human head pose estimation. We show how to integrate the learning principle of *ADRF* and that it again gives more accurate predictions compared to the baseline.

Finally, we mention a very recent work of Riegler et al. [139] that combines ideas of *HF* [62] with convolutional neural networks to infer the head pose. Thus, this method learns the mapping from depth image patches to head pose in an end-to-end fashion. This also includes a feature representation for depth images, which is typically hard to design by hand [48–50].

5.2 Head Pose Estimation with the Hough Forests Model

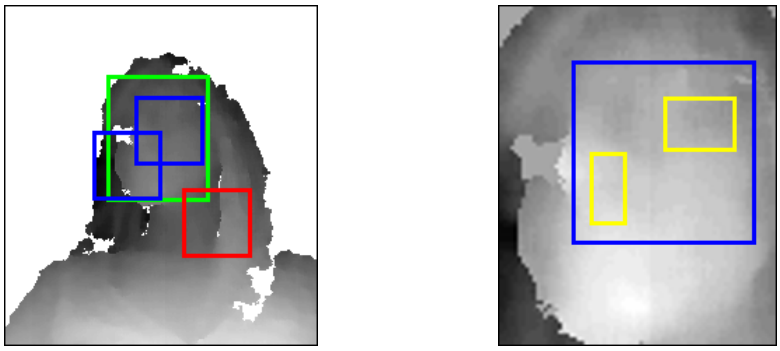
Our first attempt to human head pose estimation follows the framework of Fanelli et al. [50] that is based on *HF* [62]. We already described *HF* for object detection in Section 4.1.1 and how to extend it with the learning principle of *ADRF*. In this section, we only review the modifications of this framework for human head pose estimation from depth data [48, 50]. We briefly described the general task of head pose estimation already in the beginning of this chapter.

5.2.1 Data Modality and Labelspace

The first notable difference to object detection on 2D images is the underlying data itself, i.e., the dataspace $\mathcal{X}$ and the labelspace $\mathcal{Y}$. The data is given as depth images from a

Microsoft® Kinect™ sensor. See Figure 5.2 for some example input images of the Biwi Kinect Head Pose Database [50], which we also use in our experimental evaluation. As with standard *HF*, the data is represented with small patches $\mathbf{P} \in \mathbb{R}^{100\times100}$ of the depth image $\mathcal{D}$, where the size of the patches for this application is fixed to 100×100 pixels. While other works, e.g., [49], compute surface normals as additional features for depth data, [50] only relies on the raw depth images. The reason is that the depth data from the Kinect™ sensor is relatively noisy and an estimate of the normals is unreliable.

The label space of the *HF* approach for this application is $\mathcal{Y} = \{0, 1\} \times \mathbb{R}^6$ because each patch is attached a discrete class label $\mathtt{c}$ and a 6 dimensional pose vector $\mathbf{h}$. The class label defines whether or not the center of a patch lies within a bounding box $\mathcal{B}$ around the human head (see Figure 5.3 for an example). As already stated before, $\mathbf{h} = [x, y, z, \alpha, \beta, \gamma]^\top$, where x, y, z define the position of the head center in 3D space and α, β, γ define the head orientation as yaw, pitch, and roll, i.e., Euler angles.



(a) **(b)**

Figure 5.3: This figure illustrates the extraction of training data from depth images as well as the form of the splitting functions used for this application. (a) The green bounding box outlines the extent of the face of the human head. The blue and red squares illustrate three 100×100 training patches used to train the *RF*. Patches having their center pixel within the green bounding box belong to the foreground class (blue). All other patches belong to the background class (red). Only foreground patches are additionally associated with the corresponding pose vector $\mathbf{h}$. (b) This figure shows a zoomed-in version of the green bounding box including the upper foreground patch (blue). Here, we illustrate the form of the splitting function defined in Equation (5.1). The depth values inside the yellow boxes are averaged and subtracted from each other, yielding the final response for this patch $\mathbf{P}$ and splitting function parameters $\boldsymbol{\Theta}$.

5.2.2 Adaptations to the Hough Forest Model

The general learning framework is the same as described in Section 4.1.1. However, some notable modifications have to be applied in order to make *HF* work for this particular task. Due to the different modality of the data, i.e., noisy depth images, the response function $\xi(\mathbf{P}; \boldsymbol{\Theta})$ for a data patch $\mathbf{P}$ and parameters $\boldsymbol{\Theta}$ has to be adapted [50]. In contrast to simple pixel-pair tests [62], noisy depth data requires a more robust estimate of the

local evidence at a certain pixel location. Thus, [50] extends pixel-pair tests to regions in the depth image as

$$\xi_{\text{HF-depth}}(\mathbf{P};\boldsymbol{\Theta}) = \frac{1}{|\boldsymbol{\Theta}_1|}\sum_{\mathbf{p}\in\boldsymbol{\Theta}_1}\mathbf{P}[\mathbf{p}] - \frac{1}{|\boldsymbol{\Theta}_2|}\sum_{\mathbf{p}\in\boldsymbol{\Theta}_2}\mathbf{P}[\mathbf{p}] - \Theta_{\text{th}}\ , \tag{5.1}$$

where $|\cdot|$ is the cardinality operator. For this response function, $\boldsymbol{\Theta}_1$ and $\boldsymbol{\Theta}_2$ define sets of pixels $\mathbf{p}$ from two rectangles. These rectangles are constrained such that their center location is within the boundaries of the data patch $\mathbf{P}$ (see Figure 5.3). One has to take care of invalid pixels (undefined depth values or background), which should be excluded from the summations in Equation (5.1) as described in [50]. The summations in (5.1) can be efficiently evaluated via integral images.

While [50] also employs the general quality function of *HF*, as defined in Equation (4.1), the regression part $Q_{\text{R}}(\sigma(\mathbf{x};\boldsymbol{\Theta}),\mathbf{X})$ is defined differently. Instead of employing reduction-in-variance for the compactness measure $E(\mathbf{X})$ in (2.20), the differential entropy (2.26) with probabilities modeled as realizations of a Gaussian distribution $\mathcal{N}(\mathbf{h};\bar{\mathbf{h}},\boldsymbol{\Sigma}_{\mathbf{h}})$ is used. The parameters of the distribution are the mean $\bar{\mathbf{h}}$ and the covariance matrix $\boldsymbol{\Sigma}_{\mathbf{h}}$. The Gaussian distribution allows for solving (2.26) in closed form, which results in

$$E(\mathbf{X}) = -\log(|\boldsymbol{\Sigma}_{\mathbf{h}}|)\ , \tag{5.2}$$

where we omit constant terms. As presented in [49], also [50] approximates (5.2) with a block-diagonal covariance matrix

$$\hat{\boldsymbol{\Sigma}_{\mathbf{h}}} = \begin{bmatrix} \boldsymbol{\Sigma}_{\mathbf{h}}^{\text{Loc}} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Sigma}_{\mathbf{h}}^{\text{Rot}} \end{bmatrix} \tag{5.3}$$

where $\boldsymbol{\Sigma}_{\mathbf{h}}^{\text{Loc}}$ is the covariance matrix of the first three elements in $\mathbf{h}$, which are the location variables. The second half of $\mathbf{h}$ are the Euler angles defining the rotation of the head and give $\boldsymbol{\Sigma}_{\mathbf{h}}^{\text{Rot}}$. The block-diagonal covariance matrix turns (5.2) into

$$E(\mathbf{X}) = -\log(|\boldsymbol{\Sigma}_{\mathbf{h}}^{\text{Loc}}| + |\boldsymbol{\Sigma}_{\mathbf{h}}^{\text{Rot}}|)\ . \tag{5.4}$$

Minimizing (5.4) reduces the regression uncertainty. Furthermore, [50] makes the parameter γ that steers the influence between the classification and regression objective in (4.1) continuous and dependent on the current depth of the tree. Finally, the leaf node models in [50] have a parametric form, in contrast to the non-parametric version in standard *HF* [62], where all votes from the training set are stored that fall in a leaf. As during growing the trees, the leafs store a Gaussian distribution $\mathcal{N}(\mathbf{h};\bar{\mathbf{h}},\boldsymbol{\Sigma}_{\mathbf{h}})$ for the pose $\mathbf{h}$.

We can extend the training procedure of [50] with the learning principle of *ADRF* in the same way as we did for standard *HF* (see Section 4.1.1.3), as the modifications described above do not directly affect the *ADRF* training scheme.

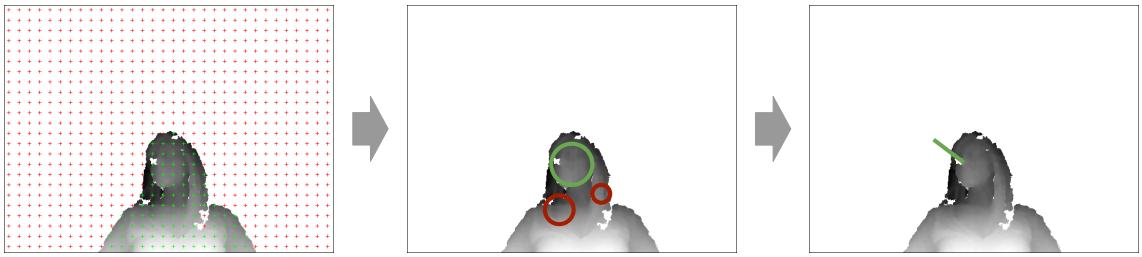


Figure 5.4: This figure illustrates the process of estimating the pose of a human head from a given depth image. (Left) The given depth image overlayed with the sampling grid of patches that are evaluated by the *RF* model. Only patches with valid depth information are allowed to vote for the pose (green dots). All remaining patches on the grid are neglected (red). As can be seen, the depth image not only contains the human head, but also the torso. (Middle) Thus, the first step in the inference process is to vote for potential locations of the head (first three dimensions in the pose vector $\mathbf{h}$). Mean-shift is used to aggregate the votes. The green and red circles indicate potential detections, where the radius is proportional to the number of supporting votes. Detections with too little support are discarded (red circles). (Right) Finally, the orientation of the head (green line) is estimated for all remaining detections.

5.2.3 Inferring the Head Pose with Hough Forests

To localize the head and infer its pose on an unseen test depth image $\mathcal{D}$, patches $\mathbf{P}$ are first extracted on a regular grid. A stride parameter defines the density of sampling patches and directly influences the trade-off between accuracy and speed.

Each extracted patch $\mathbf{P} \in \mathcal{D}$ is routed through all trees in the forest, resulting in $\mathtt{T}$ leaf nodes giving estimates of the head pose $\mathbf{h}$. In order to make the overall estimate more robust, [50] only considers leaf nodes that (i) have $p(\mathtt{c} = \text{head}|\mathbf{P}) = 1$ and (ii) $\text{tr}\,(\mathbf{\Sigma_h}) < \kappa$, where $\text{tr}\,(\cdot)$ defines the trace of a matrix. In words, leaf nodes having assigned a single training patch from the background are discarded from voting for the head pose. Moreover, leaf nodes where the sum of the variances of its prediction is above a certain threshold κ are discarded as well.

All remaining predictions are considered for voting. Finding the final estimate of the head pose involves multiple steps. First, only the 3 dimensions of the location of the head, i.e., x, y, and z, are considered. Then, similar to *HF* [62], a mode seeking approach (mean-shift in 3D for this case) is employed to find a set of potential predictions of the head. The set of predictions are grouped to allow multiple heads to be detected. Groups with a small number of supporting votes are discarded. In a second step, the head orientation, i.e., α, β, and γ, is computed as the mean of the remaining votes for each group separately. Figure 5.4 illustrates this inference process. A more detailed description can be found in [48–50].

5.3 Holistic Head Pose Model

In this section, we present our second attempt to human head pose estimation from depth images. Motivated by the experimental outcome of the previous chapter on object detection, we also evaluate a holistic approach for this task. The operating pipeline is very simple: First, a basic face detector identifies the center of the head. While this is typically simple given this type of data, the detection is not very accurate and the depth of the head center is still unknown. As a second step, we thus apply a *RF* to predict the remaining offset between the 3D location of the head detection (projected from 2D with the given depth image) and the true head location. Finally, another *RF* predicts the orientation from a single patch capturing the full human head. In the following, we briefly describe each of these steps in more detail.

Detecting the face: Depending on the provided data, one can apply different face detection algorithms. The Microsoft® Kinect™ sensor, for instance, provides an RGB image alongside with the depth data. Thus, standard face detection algorithms can be employed, which already work extraordinary well [114]. Also note that the additional depth data can constrain the search area significantly, cf., Figure 5.2, which typically further eases the task. If only depth data is available, one can also train a basic boosting cascade [41] or again use *HF* [50].

Estimating the head position: Assuming that the face detection algorithm provides a 2D location of the face, $(\tilde{x}, \tilde{y})$, one can transform this point into the 3D coordinate system given the depth estimate of the sensor at that 2D location. Note that this location typically does not coincide with the center of the head in 3D, which becomes clear when thinking of the depth value (head center versus the surface of the head). Given the true 3D annotation of a certain training data set and the outcome of a face detector, one can generate a new training set to learn the offset between these two locations. The training data are patches capturing the human head cropped from the depth images, i.e., the output of the detector. This corresponds to a holistic description of the head. The training labels are the above described offsets between 3D location of the detection and the true center of the head. We use a standard *RF* and also evaluate the Alternating Regression Forests (ARF) training scheme for this task.

Estimating the head orientation: The only thing missing for a full head pose estimation system is the orientation of the head. Given annotation in the form of yaw, pitch, and roll, as well as the same depth patches described above, we train another *RF* to estimate the orientation. Again, we can easily evaluate the *ARF* training scheme for this task.

After applying the three above-described steps, one ends up with a full estimate of the head pose $\mathbf{h}$, i.e., the location and the orientation in 3D.

5.4 Experiments

In this section, we demonstrate the performance of the above described human head pose estimation approaches. We are mainly interested in the difference between plain *HF* and our *RF* training schemes proposed in Chapter 3 (Alternating Decision Forests (ADF) and *ARF*), which we incorporated into both, a local (see Section 5.2) and a holistic estimation pipeline (see Section 5.3). Before presenting our main results, we briefly outline the experimental setup in the following section.

5.4.1 Experimental Setup

For all experiments we use the data set from [50], which is publicly available and contains more than 15K frames of depth images capturing human faces. The data set itself is split into 24 sequences, each capturing one person sitting around 1 meter away from the sensor. The ground truth annotation is given as the head center position in 3D and the pose in Euler angles for each frame. This ground truth was acquired off-line with a template-based tracker, which relies on a personalized model of each individual [50]. According to Fanelli et al. [50], this annotation is not perfect but the error is very low, translational and rotational errors of around 1mm and 1°, respectively. The data set covers rotation angles between ±75° for yaw, ±60° for pitch, and ±50° for roll. Figure 5.2 provides some examples.

Following [50], we split the data into 22 training sequences and 2 testing sequences. The separation is done based on individuals. Two persons (with ids 1 and 12) are selected for the test sequences [48]. All remaining persons (18 individuals in 22 sequences) are used for training.

The data set shows an uneven distribution of head poses, i.e., most of them are frontal facing. To counteract this issue, one has to balance the training data by binning the Euler angles (yaw, pitch, and roll) and balancing the number of frames extracted from each bin. Among some other details, this process is not well explained in [48, 50], which most probably is the reason for slightly different accuracy values reported in [48, 50] and this thesis. Here, we use the following scheme to balance the training data set. We create a 2-dimensional histogram over yaw and pitch, where each dimension is divided into 7 bins. Then we drop a random selection of training frames per bin such that we have a maximum of 1000 frames per bin. This reduces the number of training images from 14447 to 12909.

The main focus of the following set of experiments is to compare the plain *HF* training scheme (modified for this regression task [50]), *ADF*, and *ARF*. We integrate *ADF* and *ARF* into two different pose estimation pipelines (local and holistic), as described before. For the local approach, we endow all *RF* variants with the same parameters from [50], i.e., we use 7 trees, a maximum depth of 15 and 20000 random tests per node. For the holistic approach, we use 32 trees with a maximum tree depth of 20. The reason for using a stronger model (i.e., more trees) is simply the fact that the holistic approach makes

a prediction of the pose based on a single (but larger) input patch. The local approach operates on small patches that together vote for a final prediction.

5.4.2 Results

Following Fanelli et al. [50], we also present our results as the percentage of correctly predicted frames over different success thresholds for both, position and orientation of the head, respectively. The success threshold defines whether a particular frame is correctly predicted or not. For the position of the head, a ball in 3D is defined around the ground truth location, where the radius defines the success threshold and varies between 10 and 30 mm. For the rotation of the head, the average difference between ground truth and predicted angles is thresholded. Figure 5.5 illustrates our results.

For a fair comparison of all *HF* based approaches, we use our own implementation of all variants (same code basis). However, as some details of the experimental setup in [50] (training data generation, etc.) are not fully specified, we get slightly different results compared to those reported in [50]. We can still see from the plots that both approaches optimizing a global loss (HF-*ADF* and HF-*ARF*) consistently improve over the *HF* baseline. Although this pose estimation problem is evaluated as a regression task, we can observe that HF-*ADF* (classification nodes) significantly improve over *HF*. A reason for this might be that HF-*ADF* produces cleaner leaf nodes (i.e., more leafs having $p(\mathbf{y} = 1|\mathbf{P})$), which leads to more effective voting nodes that improve the overall result. In this experiment we also evaluate the performance of the combination of *ADF* and *ARF*, i.e., HF-*ADRF*. We can observe that HF-*ADRF* gives the best results for the more important tighter success thresholds (i.e., 10 mm) in the head position regression (see Figure 5.5a), and also gives good results for the orientation estimation (see Figure 5.5b).

Figure 5.5 also illustrates the results of our holistic pose estimation approach described in Section 5.3 (Hol-*RF* and Hol-*ARF*). One can see that the general accuracy for both position and orientation is better than with the local patch-based approach, except for the head position in the tighter error range. For the estimation of the 3D position, the Hol-*ARF* variant also outperforms the Hol-*RF* baseline, which can also be seen in Table 5.1. Unfortunately, this is not the case for the orientation. In any case, these results of the holistic approach have to be taken with a grain of salt. The reason is that we only simulate the face detection step for these experiments. We use the ground truth 2D location of the head (defined by the center of a binary segmentation mask provided with the data set) and randomly added an offset vector, which mimics the uncertainty of the face detection. This offset is modeled with an isotropic 2D Gaussian having zero mean and a standard deviation of 10 pixels in both dimensions. However, we again mention that the face detection task for this data set is not a tough one and such a standard deviation is reasonable.

To get a better insight in the accuracies of all methods, we also give the raw errors for all 6 variables and the aggregated head position and orientation errors in Table 5.1. Again, we observe that all methods optimizing a global loss consistently outperform their baseline

([50] for patch-based and Hol-RF for the holistic approaches) in this task. As already mentioned above, one exception is Hol-ARF, which performs worse than the baseline for the head orientation.

Again it is interesting to see that the simple holistic approach is highly competitive with the advanced patch-based approaches. This is especially true for the orientation, where the holistic approach can significantly outperform the local one. When looking at Table 5.1, a combination of these paradigms might give even better results on this data set.

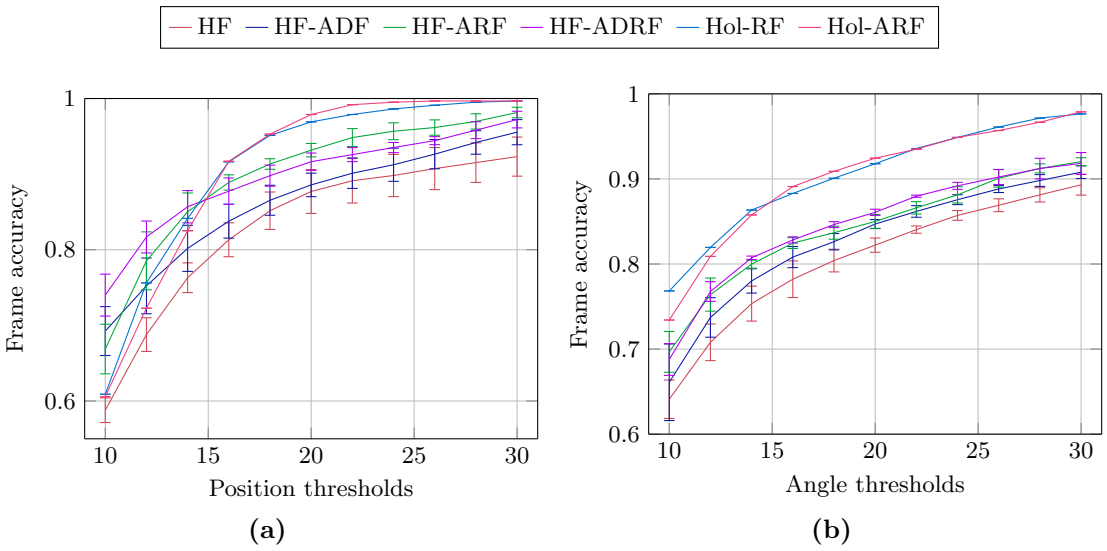


Figure 5.5: Frame accuracy of the competing methods (HF, HF-*ADF*, HF-*ARF*, HF-*ADRF*, Hol-*RF*, and Hol-*ARF*) for different success thresholds of (a) the head position in mm and (b) the head orientation in degree.

5.5 Summary

Human head pose estimation from depth data is an interesting and useful task that finds many potential applications in human-computer interaction systems. In this chapter, we first discussed related work in this field and then reviewed a recent framework based on *HF* that showed state-of-the-art performance on a standard head pose estimation benchmark. As we already presented in Chapter 4 for object detection in 2D images, we can again incorporate our *ADRF* learning principle into *HF* in the same way for human head pose estimation. Moreover, we described and evaluated a novel holistic approach for human head pose estimation, which also builds upon *RF*. We thus also investigated our *ARF* training scheme for this holistic attempt. The experimental evaluation on the BIWI Kinect Head Pose Database [50] revealed that our global loss optimization algorithms for *RF* improve the results over standard *HF* for this particular task. The results also show that

	HF [50]	HF-ARF	HF-ADF	HF-ADRF	Hol-RF	Hol-ARF
X	6.03 ± 2.20	5.84 ± 1.67	4.77 ± 1.34	4.64 ± 1.36	$\mathbf{4.56 \pm 0.00}$	4.86 ± 0.00
Y	8.63 ± 5.32	4.93 ± 2.31	5.88 ± 1.96	4.91 ± 2.02	4.74 ± 0.00	$\mathbf{4.58 \pm 0.00}$
Z	4.39 ± 2.71	4.28 ± 1.48	$\mathbf{3.53 \pm 1.16}$	4.00 ± 1.27	5.25 ± 0.00	4.61 ± 0.00
Position	12.99 ± 5.33	9.84 ± 2.26	9.50 ± 2.09	$\mathbf{8.68 \pm 1.92}$	9.87 ± 0.00	9.30 ± 0.00
Yaw	3.79 ± 0.80	3.67 ± 0.65	3.54 ± 0.49	3.52 ± 0.55	$\mathbf{3.18 \pm 0.00}$	3.41 ± 0.00
Pitch	9.27 ± 3.05	9.17 ± 2.57	7.87 ± 1.83	8.18 ± 2.08	$\mathbf{5.70 \pm 0.00}$	6.26 ± 0.00
Roll	6.62 ± 3.24	4.83 ± 1.70	5.39 ± 1.51	4.77 ± 1.45	$\mathbf{2.64 \pm 0.00}$	2.83 ± 0.00
Angle	13.48 ± 3.46	12.14 ± 2.23	11.48 ± 1.70	11.17 ± 1.77	$\mathbf{7.77 \pm 0.00}$	8.46 ± 0.00

Table 5.1: Raw regression errors (mean and standard deviation) of the Hough-voting based approaches *HF* [50], HF-*ARF*, HF-*ADF*, HF-*ADRF*, as well as the holistic approaches *Hol-RF* and *Hol-ARF*. The values are given in mm for X, Y, Z, and Position and in degree for Yaw, Pitch, Roll and Angle.

the simple holistic pipeline can be highly competitive and even better than the patch-based baseline.

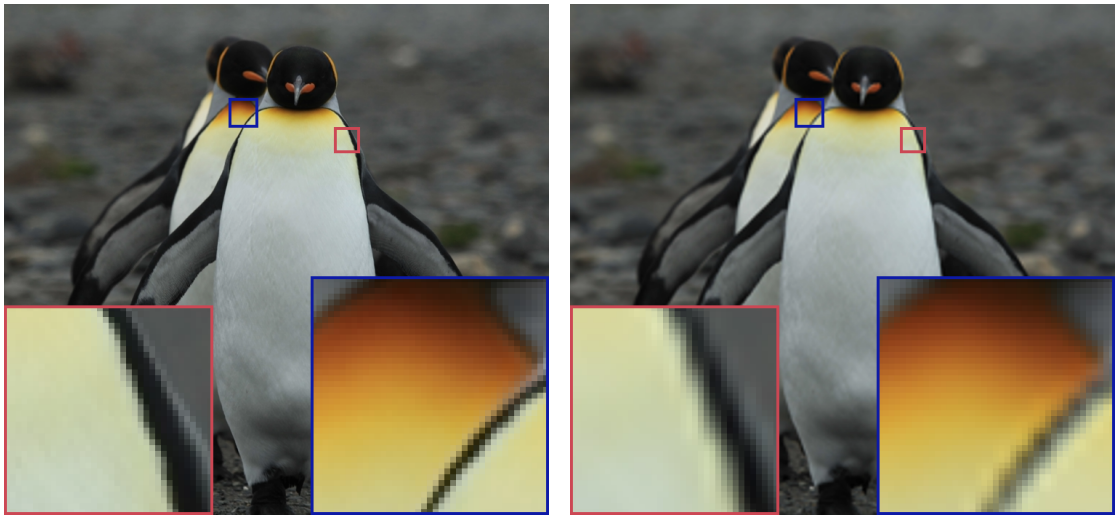
CHAPTER 6

Single Image Super-Resolution with Random Forests

Single Image Super-Resolution (SISR) [56, 68] is a classical and important computer vision problem with many interesting applications, ranging from medical and astronomical imaging to law enforcement. The task in *SISR* is to generate a visually pleasing high-resolution output from a single low-resolution input image. Although the problem is inherently ambiguous and ill-posed, simple linear, bicubic or Lanzcos interpolations [45] are often used to reconstruct the high-resolution image. These methods are extremely fast but typically yield poor results as they rely on simple smoothness assumptions that are rarely fulfilled in real images.

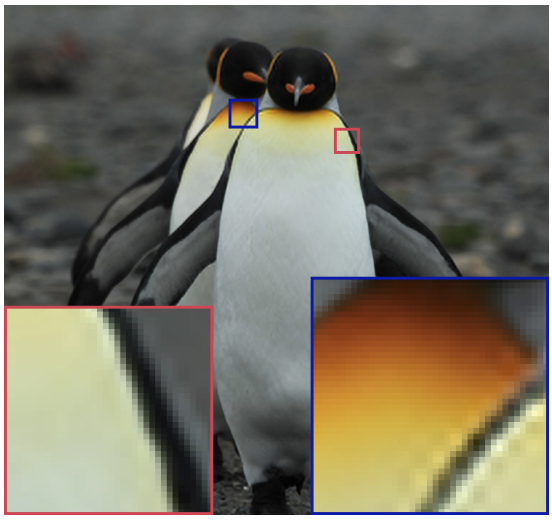
More powerful methods rely on statistical image priors [52, 56] or use sophisticated machine learning techniques [44, 200] to learn a mapping from low- to high-resolution patches. Among the best performing algorithms are sparse-coding or dictionary learning, which assume that natural patches can be represented using sparse activations of dictionary atoms. In particular, coupled dictionary learning approaches [190, 201, 204] achieved state-of-the-art results for *SISR*. Recently, Timofte et al. [178] highlighted the computational bottlenecks of these methods and proposed to replace the single dictionary with many smaller ones, thus avoiding the costly sparse-coding step during inference. This leads to a vast computational speed-up while keeping the same accuracy as previous methods.

In this chapter, we show that the efficient formulation from [178] can be naturally casted as a locally linear multivariate regression problem and that Random Forests (RF) as described in Section 2.2.4 nicely fit into this framework. We propose a novel regularized objective function optimized during tree growing that operates not only on the output label domain, but also on the input data domain. The goal of the objective is to cluster data samples that have high similarity in both domains. This eases the task for the locally

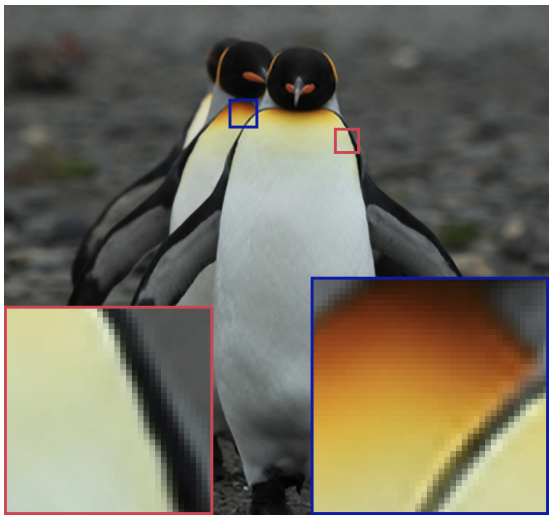


(a) Original

(b) Bicubic: PSNR = 38.33db, IFC = 4.61



(c) BPJDL [76]: PSNR = 40.79db, IFC = 4.57



(d) RFL: PSNR = 41.55db, IFC = 5.77

Figure 6.1: Exemplary results of our super-resolution approach (RFL) compared with related work. As can be seen, RFL compares favorably with basic bicubic upsampling and BPJDL [76]. The blue and red rectangles show zoomed-in areas to better underline the differences between the methods. In contrast to the other approaches, RFL produces much sharper edges without strong artificial artifacts. These results are computed for an upscaling factor of 3.

linear regressors that are learned in the leaf nodes of the trees and yields better results for *SISR* (see Figure 6.1). Furthermore, we demonstrate that Alternating Regression Forests (ARF) can be integrated into this low-level computer vision application without great effort and gives significantly better results than plain *RF*. Inference with the proposed model is also fast as patches can be routed to leaf nodes using only a few simple feature look-ups plus evaluating a linear regressor.

Our experiments demonstrate the effectiveness of our *RF* based approach on different benchmarks, where we present state-of-the-art results. Besides comparing with previous *SISR* methods including dictionary-based as well as other direct regression-based approaches, we investigate different variants of our *RF* model and validate the benefits of the novel regularized objective. We also demonstrate the effectiveness of the *ARF* training scheme from Section 3.2.2 for this task. Finally, we also evaluate the influence of several important parameters of our approach.

6.1 Related Work

The task of upscaling images, i.e., image super-resolution, has a long history in the computer vision community. While many approaches assume to have multiple views of the scene, either from a stereo setup or via temporal aggregation, this work focuses on single image super-resolution (*SISR*). Many different ideas have been explored to attempt the *SISR* problem, including dictionary learning [190, 201, 204], direct regression [44, 152, 177], or exemplar-based methods (either from the same image [68, 81] or from a large corpus of external data [170]). Here, we limit the related work to recent dictionary learning and regression-based literature as it builds the basis for our super-resolution approach. The interested reader is referred to a comprehensive survey paper [181].

Dictionary learning approaches for *SISR* typically build upon sparse coding [123]. Yang et al. [201] were one of the first who used a sparse coding formulation to learn dictionaries for the low- and high-resolution domain that are coupled via a common encoding. Zeyde et al. [204] improved upon [201] by using a stronger image representation. Furthermore, [204] employed kSVD and orthogonal matching pursuit for sparse coding [1], as well as a direct regression of the high-resolution dictionary for faster training and inference. While both approaches assume strictly equal encodings in the low- and high-resolution domains, Wang et al. [190] relaxed this constraint to a linear dependence, which they term semi-coupled. However, all these approaches are still quite slow during both training as well as inference because a sparse encoding is required.

Very recently, two different approaches came up to approximate sparse coding, aiming at much faster inference for different applications [106, 178]. The consent in both works is to replace the single large dictionary and ℓ_1-norm regularization with many smaller sub-dictionaries with ℓ_2-norm regularization. Thus, finding encoding vectors or reconstruction vectors is a quadratic problem for each sub-dictionary and can be solved in closed-form, leading to fast inference. The main effort during inference is shifted to finding the best

sub-dictionary. While [106] looks for a sub-dictionary with a small enough reconstruction error, Timofte et al. [178] selects the sub-dictionary with highest correlation to the input signal. We provide more details on [178] in Section 6.3.

As mentioned in the introduction, a different approach to *SISR* is to directly learn a mapping from the low- to the high-resolution domain via regression. Yang and Yang [200] propose to tackle the complex regression problem by splitting the input data space and learning simple regressors for each data cell. Dai et al. [35] follows a similar approach but jointly learns the separation into cells and the regressors. While the basic idea is similar to our work, both approaches have a key problem: One has to manually define the number of clusters and thus regressors, which directly influences the trade-off between upscaling quality and inference time. However, finding the appropriate number of clusters is typically data-dependent. By using *RF* on the other hand, (i) the granularity of the data cells in the input space is found automatically and (ii) the effective size of small linear regressors is typically much larger without increasing inference time.

Another recent work builds on convolutional neural networks to learn the complex mapping function [44], which is termed SRCNN. Our experiments show that our *RF* approach can learn a more accurate mapping and outperforms SRCNN. Moreover, our models can be trained within minutes on a single CPU core, while SRCNN takes 3 days to train even with the aid of GPUs. Because we base our approach on *RF* (see Section 2.2.4), we also have to mention that the training procedure can be easily parallelized on multiple CPU cores, which reduces the training time even further. The highly efficient training (and also inference) is one of the key advantages of this model.

6.2 General Super-Resolution Pipeline

Before we present our locally linear regression approach to *SISR* in Sections 6.3 and 6.4, we first outline the general computing pipeline. This pipeline is illustrated in Figure 6.2 and builds the basis for most of the methods we compare with in our experiments and also for the proposed approach.

Given a low-resolution image $\mathcal{I}_{\text{low}}$, the task is to recover the original high-resolution image $\mathcal{I}_{\text{high}}$. The first step is to apply bicubic upsampling with the desired scaling factor, which provides an intermediate image $\mathcal{I}_{\text{lowFreq}}$ that already has the desired output size but misses high frequency components. The reason is that the bicubic upsampling is a simple interpolation method and only assumes a smoothness prior resulting in blurred edges. At this point, the sophisticated super-resolution method comes into play. The goal is to recover the missing high frequency image $\mathcal{I}_{\text{highFreq}}$ from the low frequency image $\mathcal{I}_{\text{lowFreq}}$, which is illustrated with the formula `y = f(x)` in Figure 6.2. The high frequency image is defined as $\mathcal{I}_{\text{highFreq}} = \mathcal{I}_{\text{high}} - \mathcal{I}_{\text{lowFreq}}$, which is obviously not available during inference as $\mathcal{I}_{\text{high}}$ is unknown. Adding low and high frequency images gives the final output image $\mathcal{I}_{\text{final}} = \mathcal{I}_{\text{lowFreq}} + \mathcal{I}_{\text{highFreq}}$. Please note that this procedure is the same for many of the recent attempts on *SISR*, e.g., [152, 177, 178, 201, 204]. Thus, the actual mapping that is

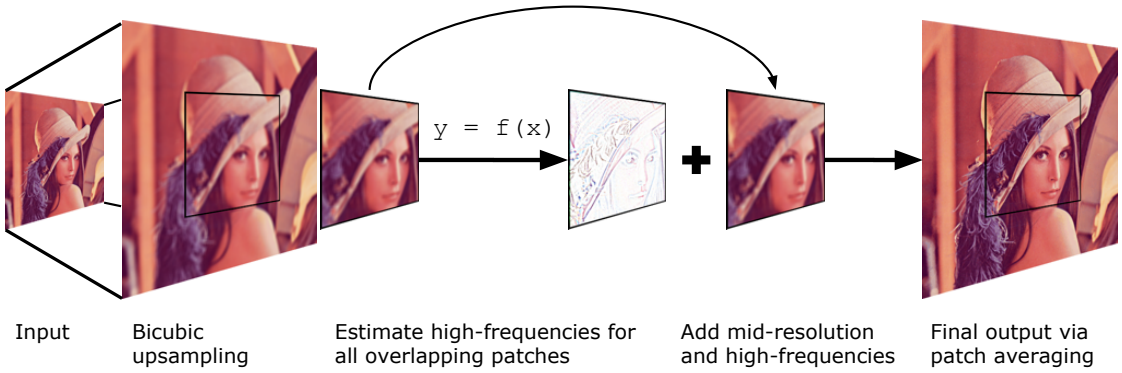


Figure 6.2: This figure gives a general overview of the *SISR* pipeline that we employ, which basically follows [178, 204]. From left to right: The low-resolution image is upsampled with a bicubic kernel. All patches are extracted and a learned mapping function $f(\cdot)$ is applied to estimate the high-frequency components that are missing in the bicubic upsampled version. The combination of these two gives the final output, where overlapping patches are simply averaged. Please note that we use RGB images here only for illustration. In practice, the pipeline operates on the luminance channel of the YCbCr color space.

need to be learned is from the low-frequency to the high-frequency domain.

As in many other super-resolution works [29, 68, 178, 204], this framework also applies regular bicubic upsampling on the color components and the sophisticated upsampling models only on the luminance component of an image. The reason is the human visual perception that is much more sensitive to high frequency changes in intensity than in color. In particular, this framework uses the YCbCr color space and applies the learned mapping on the Y channel. Note that Figure 6.2 is only an illustration where the RGB color space is used.

Another important aspect of this super-resolution pipeline is the fact that it operates on small patches of the image to apply the mapping, which is also roughly indicated in Figure 6.2. In practice, the low frequency image $\mathcal{I}_{\text{lowFreq}}$ is divided into a set of overlapping patches. Then, some basic features are computed (pre-defined linear filters) and a principal component analysis is applied for having a lower dimensionality. This defines the input to the mapping function that has to be learned. The output is the corresponding high frequency patch from $\mathcal{I}_{\text{highFreq}}$. Also note that the size of these patches varies with the upscaling factor: The larger the upscaling factor, the larger the patches. This consideration is intuitive as the real or effective image information of $\mathcal{I}_{\text{low}}$ that is available in the low frequency image $\mathcal{I}_{\text{lowFreq}}$ (the bicubic upsampled version of $\mathcal{I}_{\text{low}}$) is more spread out if the upscaling factor increases.

The final reconstructed image $\mathcal{I}_{\text{final}}$ is computed as the average of all overlapping output patches, which are computed with the super-resolution model. While this averaging approach is rather simple it still produces visually pleasing output images. Other attempts could integrate some form of image prior. A basic constraint would be that the downscaled

version of the final output image equals the original low resolution input image.

In summary, the super-resolution model has to learn a function that maps patches from $\mathcal{I}_{\text{lowFreq}}$ to the corresponding patches from $\mathcal{I}_{\text{highFreq}}$. The training data for learning this mapping can be automatically created without the need of any manual annotation. One simply collects a set of images that define the high resolution images and downscales them to generate the corresponding low resolution images. Thus, a huge number of training data can be easily generated.

In the following, we will use the terms 'low resolution' and 'low frequency', as well as 'high resolution' and 'high frequency' interchangeably in order to avoid any confusion. In any case, we always mean the input and output data that is used to learn the mapping function.

6.3 Coupled Dictionary Learning

In recent years, the predominant approaches for dictionary-based *SISR* were based on coupled dictionary learning. The first and most generic formulation was proposed by Yang et al. [201]. We denote the set of $\mathtt{N}$ samples from the low-resolution domain as $\mathbf{X}_{\text{L}} \in \mathbb{R}^{\mathtt{D}_{\text{L}} \times \mathtt{N}}$ and from the high-resolution domain as $\mathbf{X}_{\text{H}} \in \mathbb{R}^{\mathtt{D}_{\text{H}} \times \mathtt{N}}$, where each column corresponds to one sample $\mathbf{x}_{\text{L}}$ and $\mathbf{x}_{\text{H}}$, respectively. In the following, we use the notation $\mathbf{X}_{\text{L}}$ and $\mathbf{X}_{\text{H}}$ for both sets and data matrices. Then, the coupled dictionary learning problem is defined as

$$\min_{\mathbf{D}_{\text{L}}, \mathbf{D}_{\text{H}}, \mathbf{E}} = \frac{1}{\mathtt{D}_{\text{L}}} \|\mathbf{X}_{\text{L}} - \mathbf{D}_{\text{L}}\mathbf{E}\|_2^2 + \frac{1}{\mathtt{D}_{\text{H}}} \|\mathbf{X}_{\text{H}} - \mathbf{D}_{\text{H}}\mathbf{E}\|_2^2 + \Gamma(\mathbf{E}), \tag{6.1}$$

where $\mathbf{D}_{\text{L}} \in \mathbb{R}^{\mathtt{D}_{\text{L}} \times B}$ and $\mathbf{D}_{\text{H}} \in \mathbb{R}^{\mathtt{D}_{\text{H}} \times B}$ are the low- and high-resolution dictionaries, respectively. The common encoding $\mathbf{E} \in \mathbb{R}^{B \times \mathtt{N}}$ couples both domains. The regularization $\Gamma(\mathbf{E})$ is typically a sparsity inducing norm on the columns of $\mathbf{E}$, e.g., the ℓ_0 or ℓ_1 norm.

As mentioned in Section 6.1, Zeyde et al. [204] improved upon [201] with some tweaks on the learning scheme. One main difference to [201] is that they first learn the low-resolution dictionary $\mathbf{D}_{\text{L}}$ via sparse coding and compute the encoding $\mathbf{E}$. Then, they fix $\mathbf{D}_{\text{L}}$ and $\mathbf{E}$ in (6.1) and directly learn the high-resolution dictionary $\mathbf{D}_{\text{H}}$, which is a simple least squares problem. This combination of low- and high-resolution dictionaries is the basis and the first step for anchored neighborhood regression (ANR) [178] and A+ [177].

The key observation in [178] is that one can replace the sparse coding during inference, which is time-consuming, by pre-selecting so-called anchored points $\bar{\mathbf{D}}_{\text{L}}^i \in \mathbf{D}_{\text{L}}$ for each dictionary atom $\mathbf{d}^i$ in $\mathbf{D}_{\text{L}}$ already during training. During inference, a new data sample $\mathbf{x}$ selects the closest dictionary atom $\mathbf{d}^*$ in $\mathbf{D}_{\text{L}}$ according to the angular distance and uses only the pre-defined anchor points $\bar{\mathbf{D}}_{\text{L}}^*$ for computing the encoding $\mathbf{e}$. Thus, no sparse coding with ℓ_1-norm regularizer is required as the dictionary atoms are pre-selected. The problem of finding an encoding $\mathbf{e}$ for sample $\mathbf{x}$ reduces to a least squares problem and can

be computed in closed-form

$$\mathbf{e} = \arg\min_{\mathbf{e}} \|\mathbf{x} - \bar{\mathbf{D}}_{\mathrm{L}}^{*}\mathbf{e}\|_2^2 = (\bar{\mathbf{D}}_{\mathrm{L}}^{*\top}\bar{\mathbf{D}}_{\mathrm{L}}^{*})^{-1}\,\bar{\mathbf{D}}_{\mathrm{L}}^{*\top}\mathbf{x} = \mathbf{P}^{*}\mathbf{x}\;. \tag{6.2}$$

All matrices $\mathbf{P}^{i}$ for each dictionary atom can be pre-computed in the training phase. Furthermore, using the corresponding high-resolution anchored points $\bar{\mathbf{D}}_{\mathrm{H}}^{*}$, the reconstructed high-resolution sample $\hat{\mathbf{x}}_{\mathrm{H}}$ can be computed as

$$\hat{\mathbf{x}}_{\mathrm{H}} = \bar{\mathbf{D}}_{\mathrm{H}}^{*} \cdot \mathbf{e} = \bar{\mathbf{D}}_{\mathrm{H}}^{*} \cdot \mathbf{P}^{*} \cdot \mathbf{x}_{\mathrm{L}} = \mathbf{W}^{*} \cdot \mathbf{x}_{\mathrm{L}}\;. \tag{6.3}$$

As can be seen, the problem reduces to selecting an atom from $\mathbf{D}_{\mathrm{L}}$ and applying the pre-computed mapping $\mathbf{W}^{*}$. While ANR [178] is limited to selecting close anchor points (angular distance) from the learned dictionary $\mathbf{D}_{\mathrm{L}}$, A+ [177] shows that using a much larger corpus from the full training set yields more accurate predictors and better results.

Both works [177, 178] still make a detour via sparse-coded dictionaries. However, one can also see this problem more directly as non-linear regression. In this work, we thus reformulate the problem of learning the mapping from the low- to the high-resolution domain via locally linear regression. To do so, we realize that the mapping function $\mathbf{W}$ in (6.3) depends on the data $\mathbf{x}_{\mathrm{L}}$, which yields

$$\hat{\mathbf{x}}_{\mathrm{H}} = \mathbf{W}(\mathbf{x}_{\mathrm{L}}) \cdot \mathbf{x}_{\mathrm{L}}\;. \tag{6.4}$$

Now, training only requires to find the function $\mathbf{W}(\mathbf{x}_{\mathrm{L}})$. Two recent works [44, 200], already mentioned in Section 6.1, also attack the super-resolution task with a direct regression formulation to learn $\mathbf{W}(\mathbf{x}_{\mathrm{L}})$. In the next section, we detail our approach and show how *RF* can be used as an effective algorithm to learn this mapping, which has some critical benefits over previous works.

6.4 Random Forests for Super-Resolution

In this section, we investigate the problem of learning the data-dependent function $\mathbf{W}(\mathbf{x}_{\mathrm{L}})$ defined in Equation (6.4). Assuming a regular squared loss, the learning problem can be formulated as

$$\arg\min_{\mathbf{W}(\mathbf{x}_{\mathrm{L}})} \sum_{n=1}^{\mathtt{N}} \|\mathbf{x}_{\mathrm{H}}^{n} - \mathbf{W}(\mathbf{x}_{\mathrm{L}}^{n}) \cdot \mathbf{x}_{\mathrm{L}}^{n}\|_2^2\;. \tag{6.5}$$

We can generalize this model with different basis functions $\phi(\mathbf{x})$ resulting in

$$\arg\min_{\mathbf{W}_j(\mathbf{x}_{\mathrm{L}})\;\forall j} \sum_{n=1}^{\mathtt{N}} \|\mathbf{x}_{\mathrm{H}}^{n} - \sum_{j=0}^{\gamma} \mathbf{W}_j(\mathbf{x}_{\mathrm{L}}^{n}) \cdot \phi_j(\mathbf{x}_{\mathrm{L}}^{n})\|_2^2\;, \tag{6.6}$$

where the goal is to find the data-dependent regression matrices $\mathbf{W}_j(\mathbf{x}_\mathrm{L})$ for each of the $\gamma + 1$ basis functions. While the standard choice for $\phi_j(\mathbf{x})$ is the linear basis function, i.e., $\phi_j(\mathbf{x}) = \mathbf{x}$, one can also opt for polynomial ($\phi_j(\mathbf{x}) = \mathbf{x}^{[j]}$, where $^{[\cdot]}$ denotes the point-wise power operator) or radial basis functions ($\phi_j(\mathbf{x}) = \exp\left(\frac{\|\mathbf{x}-\mu_j\|^2}{\sigma_j}\right)$, where μ_j and σ_j are parameters of the basis function) [18]. We evaluate different choices of $\phi_j(\mathbf{x})$ in our experiments. Either way, the objective stays linear in the parameters to be learned, if the data-dependence is neglected for a moment. The question remaining is how to define the data-dependence of $\mathbf{W}$ on $\mathbf{x}_\mathrm{L}$.

Here, we employ *RF* to create the data dependence. For this task, i.e., multi-variate regression with $\mathtt{D}_\mathrm{H}$ dimensions, the goal is to learn a mapping from the data input space $\mathcal{X} = \mathbb{R}^{\mathtt{D}_\mathrm{L}}$ to the target output space $\mathcal{Y} = \mathbb{R}^{\mathtt{D}_\mathrm{H}}$. At this point, we assume that the structure of each single tree already exists and we postpone the description of the training procedure for this particular task to Section 6.4.1.

Each tree $\mathcal{T}_t$ independently separates the data space into disjoint cells, which correspond to the leaf nodes. Using more than one tree, i.e., forests, leads to overlapping cells. This separation defines our data dependence for $\mathbf{W}(\mathbf{x}_\mathrm{L})$ and each leaf l can learn a linear model

$$m_l(\mathbf{x}_\mathrm{L}) = \sum_{j=0}^{\gamma} \mathbf{W}_j^l \cdot \phi_j(\mathbf{x}_\mathrm{L}) \; . \tag{6.7}$$

Finding all $\mathbf{W}_j^l$ requires solving the least squares problem which can be done in closed-form as $\mathbf{W}^{l\top} = (\Phi(\mathbf{X}_\mathrm{L})^\top \Phi(\mathbf{X}_\mathrm{L}) + \lambda \mathtt{I})^{-1} \Phi(\mathbf{X}_\mathrm{L})^\top \cdot \mathbf{X}_\mathrm{H}$, where we stacked all $\mathbf{W}_j^l$ and the data into matrices $\mathbf{W}^l$, $\Phi(\mathbf{X}_\mathrm{L})$, and $\mathbf{X}_\mathrm{H}$. The regularization parameter λ has to be specified by the user and avoids singular matrices. Now, we can easily see the relation to the locally linear regression model from Equation (6.4). Because we average predictions over all $\mathtt{T}$ trees during inference, the data-dependent mapping matrix $\mathbf{W}(\mathbf{x}_\mathrm{L})$ is modeled as

$$\hat{\mathbf{x}}_\mathrm{H} = m(\mathbf{x}_\mathrm{L}) = \mathbf{W}(\mathbf{x}_\mathrm{L}) \cdot \mathbf{x}_\mathrm{L} = \frac{1}{\mathtt{T}} \sum_{t=1}^{\mathtt{T}} m_{l(t)}(\mathbf{x}_\mathrm{L}) \; , \tag{6.8}$$

where $l(t)$ is the leaf in tree t the sample $\mathbf{x}_\mathrm{L}$ is routed to.

We also note that the recently presented filter forests [51] have similar leaf node models $m_l(\mathbf{x}_\mathrm{L})$. They use the linear basis function and learn the model for a single output dimension within the context of image denoising. Our leaf node model from (6.7) can be considered as a more general formulation. Another option for $m_l(\mathbf{x}_\mathrm{L})$, which is typically used in random regression forests, is a constant model, i.e., $m_l(\mathbf{x}_\mathrm{L}) = \hat{m} = \text{const}$. The constant prediction $\hat{m}$ is often the empirical mean over the labels $\mathbf{x}_\mathrm{H}$ of the training samples falling into that leaf. We evaluate different versions of $m_l(\mathbf{x}_\mathrm{L})$ in our experiments.

6.4.1 Learning the Tree Structure

All trees in our *RF* are trained independently from each other with a set of $\mathtt{N}$ training samples $\{\mathbf{x}_{\mathrm{L}}^n, \mathbf{x}_{\mathrm{H}}^n\} \in \mathcal{X} \times \mathcal{Y}$. To briefly recap Section 2.2.4.2, training a single random tree involves recursively splitting the training data into disjoint subsets by finding splitting functions

$$\sigma(\mathbf{x}_{\mathrm{L}}; \boldsymbol{\Theta}) = \begin{cases} 0 & \text{if } \xi(\mathbf{x}_{\mathrm{L}}; \Theta) < 0 \\ 1 & \text{otherwise} \end{cases} , \tag{6.9}$$

for all internal nodes in $\mathcal{T}_t$. Splitting starts at the root node and continues in a greedy manner down the tree until a maximum depth $\mathtt{D}_{\max}$ is reached and a leaf is created. The values of $\boldsymbol{\Theta}$ define the response function $\xi(\mathbf{x}_{\mathrm{L}}; \boldsymbol{\Theta})$. For this particular application, we build on pair-wise differences for the response function, i.e., $\xi(\mathbf{x}_{\mathrm{L}}; \boldsymbol{\Theta}) = \mathbf{x}_{\mathrm{L}}[\Theta_1] - \mathbf{x}_{\mathrm{L}}[\Theta_2] - \Theta_{\mathrm{th}}$, where the operator $[\cdot]$ selects one dimension of $\mathbf{x}_{\mathrm{L}}$, $\Theta_1, \Theta_2 \in \{1, \ldots, \mathtt{D}_{\mathrm{L}}\} \subset \mathbb{Z}$, and $\Theta_{\mathrm{th}} \in \mathbb{R}$ is a threshold.

The typical procedure for finding good parameters $\boldsymbol{\Theta}$ for the splitting function $\sigma(\cdot; \boldsymbol{\Theta})$ is to sample a random set of parameter values $\boldsymbol{\Theta}_k$ and choosing the best one $\boldsymbol{\Theta}^*$ according to a quality measure. The quality for a specific splitting function $\sigma(\mathbf{x}_{\mathrm{L}}; \boldsymbol{\Theta})$ is computed as

$$Q(\sigma(\cdot; \boldsymbol{\Theta}), \mathbf{X}_{\mathrm{H}}, \mathbf{X}_{\mathrm{L}}) = E(\mathbf{X}_{\mathrm{H}}, \mathbf{X}_{\mathrm{L}}) - \sum_{c \in \{Le, Ri\}} |\mathbf{X}^c| \cdot E(\mathbf{X}_{\mathrm{H}}^c, \mathbf{X}_{\mathrm{L}}^c) , \tag{6.10}$$

where Le and Ri define the left and right child nodes and $|\cdot|$ is the cardinality operator. With a slight abuse of notation, we define for both domains $\mathbf{X}_{\{\mathrm{H,L}\}}^{Le} = \{\mathbf{x}_{\{\mathrm{H,L}\}} : \sigma(\mathbf{x}_{\mathrm{L}}; \boldsymbol{\Theta}) = 0\}$, $\mathbf{X}_{\{\mathrm{H,L}\}}^{Re} = \{\mathbf{x}_{\{\mathrm{H,L}\}} : \sigma(\mathbf{x}_{\mathrm{L}}; \boldsymbol{\Theta}) = 1\}$. The function $E(\mathbf{X}_{\mathrm{H}}, \mathbf{X}_{\mathrm{L}})$ aims at measuring the compactness or the purity of the data, cf., Section 2.2.4.2. The intuition is to have similar data samples falling into the same leaf nodes, thus, giving coherent predictions.

For our task, we define a novel regularized compactness measure $E(\mathbf{X}_{\mathrm{H}}, \mathbf{X}_{\mathrm{L}})$ that not only operates on the label space $\mathcal{Y}$ but also on the input space $\mathcal{X}$. We define it as

$$E(\mathbf{X}_{\mathrm{H}}, \mathbf{X}_{\mathrm{L}}) = \frac{1}{|\mathbf{X}|} \sum_{n=1}^{|\mathbf{X}|} \left(\|\mathbf{x}_{\mathrm{H}}^n - m(\mathbf{x}_{\mathrm{L}}^n)\|_2^2 + \kappa \cdot \|\mathbf{x}_{\mathrm{L}}^n - \bar{\mathbf{x}}_{\mathrm{L}}\|_2^2 \right) , \tag{6.11}$$

where $m(\mathbf{x}_{\mathrm{L}}^n)$ again is the prediction for the sample $\mathbf{x}_{\mathrm{L}}^n$, $\bar{\mathbf{x}}_{\mathrm{L}}$ is the mean over the samples $\mathbf{x}_{\mathrm{L}}^n$, and κ is a hyper-parameter.

The first term in (6.11) operates on the label space and we get different variants depending on the choice of $m(\mathbf{x}_{\mathrm{L}})$. If $m(\mathbf{x}_{\mathrm{L}})$ is the average over all samples in $\mathbf{X}_{\mathrm{H}}$, i.e., a constant model, we have the reduction-in-variance objective that is employed in many works, e.g., in Hough Forests (HF) [62]. Assuming a linear model $m(\mathbf{x}_{\mathrm{L}})$, we have a reconstruction-based objective as in [51], which is typically more time consuming than the reduction-in-variance option. The model chosen during growing the trees and for the final leaf nodes can be different. For instance, one could assume a constant model during growing the trees, but a linear one for the final leaf nodes.

The second term in (6.11) operates on the data space and can be considered as a clustering objective that is steered with κ. The intuition of this regularization is that we not only require to put samples into a common leaf node that have similar labels $\mathbf{x}_\mathrm{H}$, but we also want the samples themselves (i.e., $\mathbf{x}_\mathrm{L}$) to be similar. This potentially eases the task for the linear regression model $m_l(\mathbf{x}_\mathrm{L})$ in the leaf.

After fixing the best split $\sigma(\cdot)$ according to (6.10), the data in the current node is split and forwarded to the left and right children, respectively. As in standard *RF*, growing continues until one of the stopping criteria are met and the final leaf nodes are created.

6.4.2 Integrating Alternating Regression Forests

In the previous sections, we showed how *RF* can be used for learning a locally linear mapping function between low- and high-resolution images, alongside with some modifications of the splitting criterion. While the first contribution of this chapter is the connection between dictionary-based *SISR* approaches with locally linear regression and that *RF* are well suited for this task, the second contribution is the integration and use of the *ARF* training scheme. We already discussed the general framework of *ARF* in Sections 3.2 and 3.2.2. Again, the integration is straightforward for this particular application.

Fortunately, the generic formulation from Section 3.2.2 does not change. The reason is that only the predictions of the whole *RF* are required to compute the gradients, but not how these predictions are computed. While we always employed a constant leaf node model for *ARF* in previous chapters, for this task, we use a multivariate linear model to compute the predictions of all trees. As already said, this does not change the generic *ARF* algorithm, but it has an impact on the training time. While the leaf nodes for standard *RF* only have to be computed when any of the stopping criteria is met, the prediction models have to be computed also for intermediate nodes in *ARF* in order to evaluate the common loss function, see Section 3.2.1. Computing a constant prediction model is obviously less time-consuming than a multivariate linear model as it involves solving a linear system of equations. Nevertheless, the integration is easy and the increased training time pays off, which is confirmed in our experimental evaluation in Section 6.5.3.

6.5 Experiments on Single Image Super-Resolution

In this section we assess the performance of our direct regression-based approach for *SISR* on different data sets as well as 3 upscaling factors. After outlining our experimental setup including details on data sets and evaluation metrics, we compare with different *RF* variants as well as several state-of-the-art *SISR* approaches. To provide more insights into the proposed method, we finally investigate several properties and parameter choices in more detail.

6.5.1 Experimental Setup and Benchmarks

The publicly available framework of Timofte et al. [178] is a perfect base for evaluating *SISR* approaches. It includes state-of-the-art results of different dictionary learning [178, 201, 204] as well as neighborhood embedding [16, 29] approaches allowing for a fair and direct comparison on the same code basis. As in many other super-resolution works [29, 68, 204], this framework also follows the general procedure described in Section 6.2.

For our evaluations, we use three different sets of test images, `Set5` from [16], `Set14` from [204], and `BSDS` from [9], consisting of 5, 14, and 200 images, respectively. For `Set5` and `Set14`, the training set consists of 91 images that are provided with the framework of Timofte et al. [178] and were also used in previous works. For `BSDS`, we use the provided 200 training images. As described in Section 6.2, the actual input to the super-resolution method are features computed on patches that are extracted from the bicubic upsampled low resolution image. We sample patches from those images with a stride that corresponds to the upscaling factor and use a patch size of 3×3 multiplied with the upscaling factor. Regarding the features, the most basic choice is the patch itself. However, better results can be achieved by using first- and second-order derivatives of the patch followed by a basic dimensionality reduction (PCA), as was used in [178, 204]. Note that we use the exact same features as in previous work, allowing for a direct comparison of the upsampling method itself. Unless otherwise noted, the default settings for our *RF* model are: $\mathtt{T} = 15$, $\mathtt{D}_{\max} = 15$, $\lambda = 0.01$, and $\kappa = 1$.

6.5.2 Random Regression Forest Variants

Before comparing with state-of-the-art *SISR* approaches, we evaluate different variants of our *RF* based method. In Section 6.4, we formulated our approach as a locally linear regressor, where locality is defined via a *RF*. Thus, the *RF* holds a linear prediction model in each leaf node, akin to filter forests [51]. For the linear model, we can choose between different basis functions $\phi(\mathbf{x})$. We evaluate the linear (RFL) and the polynomial (RFP) cases. In contrast, one can also store a constant model (RFC) in each leaf node by finding a mode in the continuous output space of the incoming data, e.g., by taking the average (see Section 6.4). Another choice is using structured forests from Dollár and Zitnick [43], which learn a mapping from input to arbitrary structured outputs. However, this option makes little sense for this task because the output space is continuous and a quality measure can be defined easily also for high-dimensional output spaces, see Section 6.4.1.

For the comparison between these different *RF* choices, we set the number of trees to $\mathtt{T} = 8$ and keep the remaining standard settings. For the constant leaf node model (RFC), we fix the minimum number of samples per leaf node to 5. For the two linear models (RFL and RFP), this parameter is set higher (64) as a linear regression model has to be learned. We evaluate on `Set5` and `Set14` for two different magnification factors. All results are presented in Table 6.1. We report the upscaling quality (PSNR in dB) as well

Set	Set5				Set14			
Factor	x2		x3		x2		x3	
RFC	35.3	0.6	31.3	0.3	31.4	1.2	28.2	0.6
RFL	36.4	0.7	32.1	0.4	32.2	1.5	28.9	0.8
RFP	36.4	0.9	32.2	0.5	32.2	1.7	28.9	1.0

Table 6.1: Comparison of different leaf node prediction models on two data sets and for different upscaling factors. We compare a constant leaf node model (RFC), a linear one (RFL), and a polynomial one (RFP). For each setting, the left column presents the PSNR in dB and the right column the upscaling time in seconds.

as the upscaling time (in seconds), respectively. While the performance gap between RFL and RFP is almost zero, one can clearly observe the inferior results of the constant leaf node model.

6.5.3 Comparison with State-of-the-art

We split our comparison with related work into two parts. First, we compare with methods that build upon the exact same framework as [178, 204] in order to have a dedicated comparison of our regression model. Besides standard bicubic upsampling, we compare with a sparse coding approach [204], ANR [178], as well as neighborhood embedding [29]. We use the same 91 training images as in [178] for `Set5` and `Set14`. For `BSDS`, we use the provided 200 training images. Second, we compare with state-of-the-art methods in *SISR* that eventually build upon a different framework and use different sets of training images. We compare with A+ [178], SRCNN [44], and BPJDL [76]. Regarding the proposed approaches, we evaluate the standard *RF* model (RFL) and also *ARF*, here denoted as ARFL. As described in Section 6.4.2, ARFL optimizes the squared loss, which is the natural choice for this task and the PSNR metric. Additionally, we add two variants (RFL+ and ARFL+) which use an augmented training set consisting of the union of the 200 BSDS and the 91 images from [178].

data set	factor	Bicubic PSNR/IFC/Time	Zeyde [204] PSNR/IFC/Time	ANR [178] PSNR/IFC/Time	NE+LLE PSNR/IFC/Time	RFL PSNR/IFC/Time	ARFL PSNR/IFC/Time
Set5	x2	33.66/6.08/0.0	35.77/7.86/3.9	35.82/8.07/0.7	35.76/7.86/5.4	36.55/8.52/1.2	**36.70**/**8.56**/1.2
	x3	30.39/3.58/0.0	31.91/4.51/1.8	31.91/4.60/0.4	31.84/4.51/2.5	32.46/4.92/0.9	**32.58**/**4.92**/1.0
	x4	28.42/2.33/0.0	29.73/2.96/1.1	29.73/3.03/0.3	29.62/2.96/1.4	30.15/**3.22**/0.8	**30.21**/3.19/0.7
Set14	x2	30.23/6.07/0.0	31.81/7.64/8.0	31.78/7.81/1.3	31.75/7.65/11.1	32.26/8.14/2.2	**32.37**/**8.14**/2.1
	x3	27.54/3.47/0.0	28.68/4.23/3.6	28.64/4.31/0.7	28.60/4.24/5.2	29.05/**4.54**/1.7	**29.13**/4.53/1.7
	x4	26.00/2.24/0.0	26.91/2.75/2.3	26.87/2.81/0.5	26.82/2.76/3.0	27.24/**2.95**/1.3	**27.30**/2.92/1.3
BSDS	x2	29.70/5.68/0.0	30.99/7.10/19.5	30.94/7.24/0.9	30.91/7.08/10.8	31.50/**7.58**/2.3	**31.62**/7.56/2.3
	x3	27.26/3.21/0.0	28.05/3.87/5.8	27.99/3.91/0.5	27.97/3.85/4.4	28.39/**4.15**/2.2	**28.46**/4.14/2.3
	x4	25.97/2.04/0.0	26.60/2.50/3.6	26.56/2.53/0.3	26.53/2.49/2.7	26.86/**2.67**/2.1	**26.90**/2.64/2.1

Table 6.2: Results of the proposed method compared within the framework of [178] on 3 data sets for 3 upscaling factors. The results of RFL are averaged over 3 independent runs.We omit the standard deviation in the table as it was negligibly low (0.0059 and 0.0058 on average for PSNR and IFC, respectively).

data set	factor	A+ [177] PSNR/IFC/Time	SRCNN [44] PSNR/IFC/Time	BPJDL [76] PSNR/IFC/Time	RFL PSNR/IFC/Time	RFL+ PSNR/IFC/Time	ARFL+ PSNR/IFC/Time
Set5	x2	36.55/8.47/0.8	36.34/7.52/3.0	36.41/7.77/129.8	36.55/8.52/1.1	36.73/8.66/2.0	**36.89**/**8.66**/2.1
	x3	32.59/4.93/0.5	32.39/4.31/3.0	32.10/4.50/110.0	32.46/4.92/1.0	32.63/**5.00**/1.6	**32.72**/4.99/1.7
	x4	30.28/3.25/0.3	30.09/2.84/3.2	-	30.15/3.22/0.8	30.29/**3.27**/1.5	**30.35**/3.24/1.5
Set14	x2	32.28/8.10/1.5	32.18/7.23/4.9	32.17/7.60/243.8	32.26/8.14/2.3	32.41/**8.28**/3.9	**32.52**/8.25/3.9
	x3	29.13/4.53/0.9	29.00/4.03/5.0	28.76/4.17/217.7	29.05/4.54/1.8	29.17/**4.60**/2.5	**29.23**/4.57/2.5
	x4	27.32/2.96/0.6	27.20/2.61/5.2	-	27.24/2.95/1.3	27.35/**2.98**/2.1	**27.41**/2.96/2.1
BSDS	x2	31.44/7.46/1.2	31.38/6.77/3.4	31.35/6.92/144.1	31.50/7.58/2.5	31.52/**7.61**/2.8	**31.66**/7.60/3.1
	x3	28.36/4.09/0.6	28.28/3.69/3.4	28.10/3.72/137.6	28.39/**4.15**/2.3	28.38/4.15/2.0	**28.45**/4.13/2.0
	x4	26.83/2.63/0.4	26.73/2.38/3.5	-	26.86/**2.67**/2.1	26.85/2.66/1.7	**26.89**/2.63/1.7

Table 6.3: Results of the proposed method compared with state-of-the-art works on 3 data sets for 3 upscaling factors.

Tables 6.2 and 6.3 show the quantitative results for both parts of our evaluation on different upscaling factors and image sets, respectively. We report both PSNR (in dB) and the IFC score, which was shown to have higher correlation with the human perception compared to other metrics [199]. As can be seen in Table 6.2, our super-resolution forests with linear leaf node models, RFL and ARFL, achieve better results than all dictionary-based and neighborhood embedding approaches. Importantly, ARFL again clearly outperforms the standard model, i.e., RFL. Furthermore, all our models compare favorably with state-of-the-art methods, see Table 6.3. We can even outperform the complex CNN, although our approach can be trained within minutes while [44] takes 3 days on a GPU. Note however that the training time for the models relying on the *ARF* scheme, ARFL and ARFL+, is longer. In this setting, the difference of the training times between standard *RF* and *ARF* becomes larger compared to, e.g., Section 3.4.1. The reason is that the intermediate predictions, which have to be computed for each node, require more computational costs as a linear system of equations has to be solved, instead of a simple averaging operation. The inference times of SRCNN (in the table) differ from the ones reported in [44] as only the slower Matlab implementation is publicly available. Qualitative results can be found in Figure 6.1 and Section 6.5.4.

In Figure 6.3 we visualize the trade-off between upscaling quality and inference time for different methods. We include two additional neighborhood embedding approaches [16, 29] (NE+LS, NE+NNLS) as well as different variants of our *RF* (with different number of trees $\mathtt{T} = \{1, 5, 10, 15\}$). The figure shows the average results from `Set5` for an upscaling factor of 2. One can see that RFL provides a good trade-off between accuracy and inference time. Already the variant with a single tree (RFL-1) gives better results than many related methods. Using 5 trees improves the results significantly with only a slight increase in inference time. SRCNN [44] is clearly the fastest method because no explicit feature computation is required (timings directly taken from [44]), which could potentially be sped-up in the framework of Timofte [178].

Finally, we compare different dictionary sizes for dictionary-based approaches with our *RF* in Figure 6.4a. For our model, we again include four variants with different number of trees $\mathtt{T} = \{1, 5, 10, 15\}$. As can be seen, our weakest model ($\mathtt{T} = 1$) already outperforms dictionary based models up to a dictionary size of 2048 while being almost as fast as GR [178] (Figure 6.4b). When using more trees, RFL outperforms even larger dictionaries. Figure 6.4c shows that our *RF* model (trained from 91 images) takes less time to train than ANR [178], even though we do not train our trees in parallel yet.

6.5.4 Qualitative Results

Beside the quantitative results presented in the previous section, we also provide qualitative results comparing the presented *RF*based super-resolution approach with many related methods. Here, we include the results of 4 different images for upscaling factors 3 (Figures 6.8 to 6.11) and 4 (Figures 6.12 to 6.15), respectively. These figures are attached

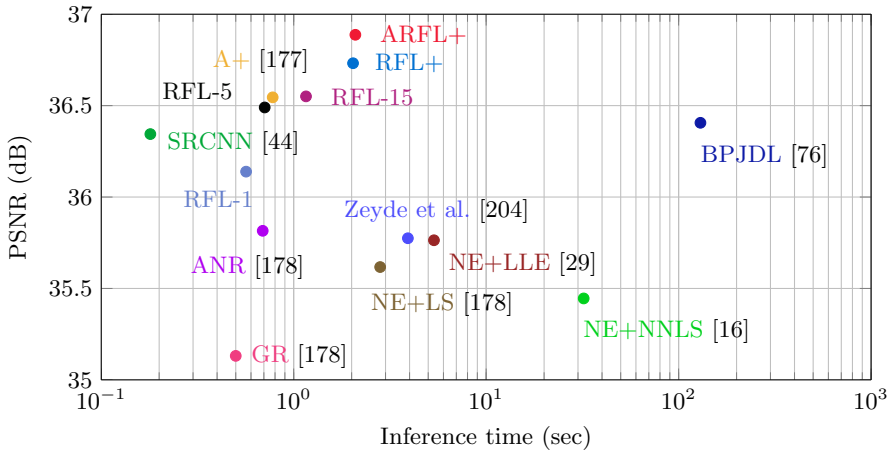


Figure 6.3: Visualization of the trade-off between accuracy and inference time for different methods. The results are the average over the images from `Set5`.

at the end of this chapter to keep a smooth and easy-to-read layout for the remaining sections. More qualitative results can be found in [153]. A good example of the qualitative differences between the evaluated models can be seen in Figure 6.8. The skiing stick shows severe artifacts for many upscaling methods like [44, 76, 178, 204]. A+ [177] and our approaches, which all rely on a locally linear regression, show relatively good results without artifacts. ARFL+ achieves the best results in terms of both PSNR and IFC scores. Sharper edges can also be observed at the skiing helmet for these methods. These differences also become clear when looking at the striped structures in Figures 6.9, 6.10, and 6.11. Similar observations can also be made for an upscaling factor of 4. Please note that we did not include the qualitative results of [76] as we did not had the model for this upscaling factor. As can be clearly seen, the proposed RFL produces much sharper edges with little artifacts.

6.5.5 Influence of the Tree Structure

The main factor influencing the tree structure of RF, beside the inherent randomness induced, is the objective function used to evaluate potential splitting functions. We investigate the influence of six different choices. First, a fully random selection of the splitting function (Ra), i.e., extremely randomized trees [65]. Second, an objective that prefers balanced trees (Ba), which we define as

$$\mathrm{Q}(\sigma, \Theta, \mathbf{X}) = -\left(|\mathbf{X}^{Le}| - |\mathbf{X}^{Ri}|\right)^2 , \tag{6.12}$$

where $\mathbf{X}^{Le}$ and $\mathbf{X}^{Ri}$ are the samples falling into left and right child nodes according σ. The remaining options are reduction-in-variance with $\kappa = 0$ (Va) and $\kappa = 1$ (VaF) and the reconstruction-based objective, again with $\kappa = 0$ (Re) and $\kappa = 1$ (ReF), respectively.

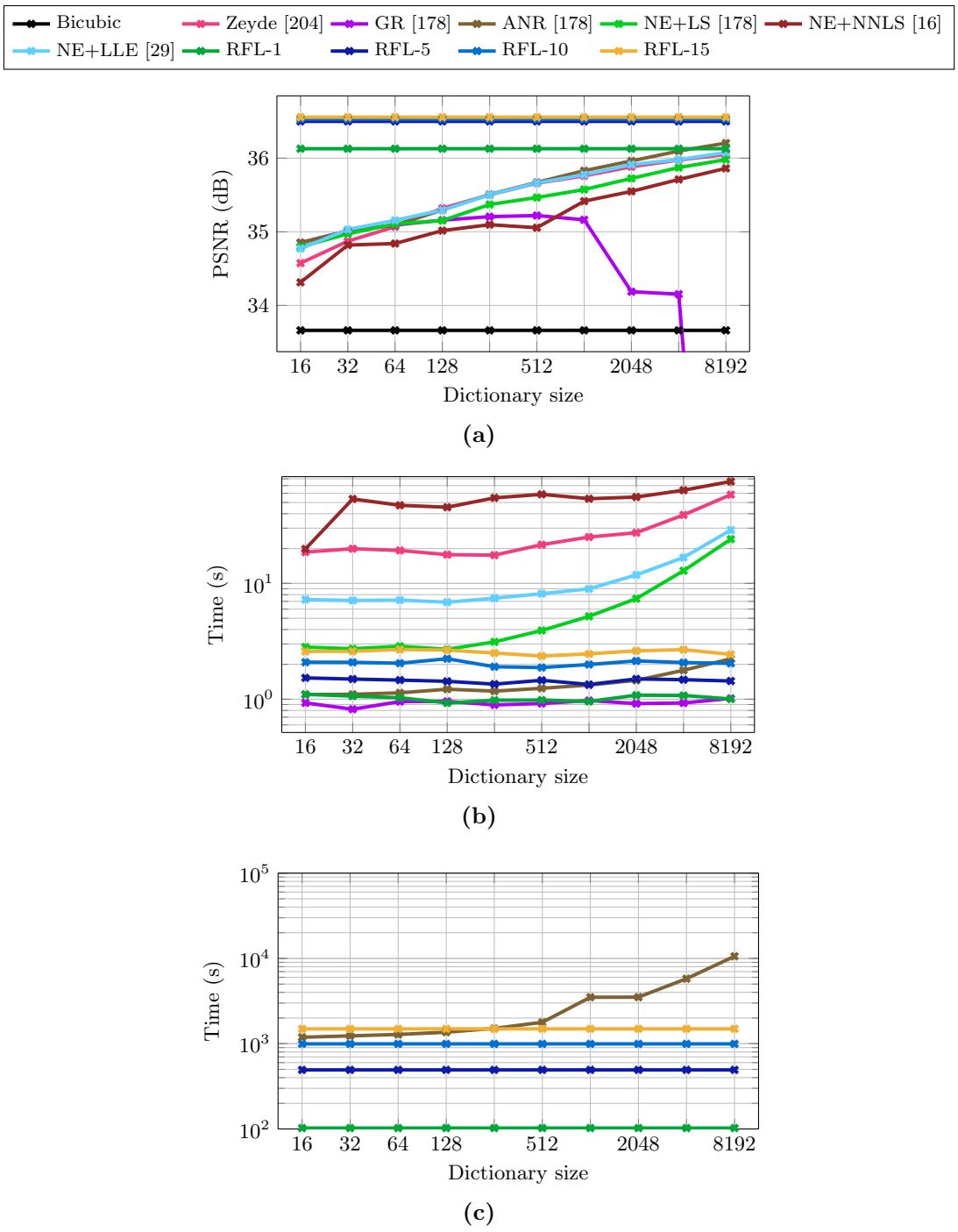
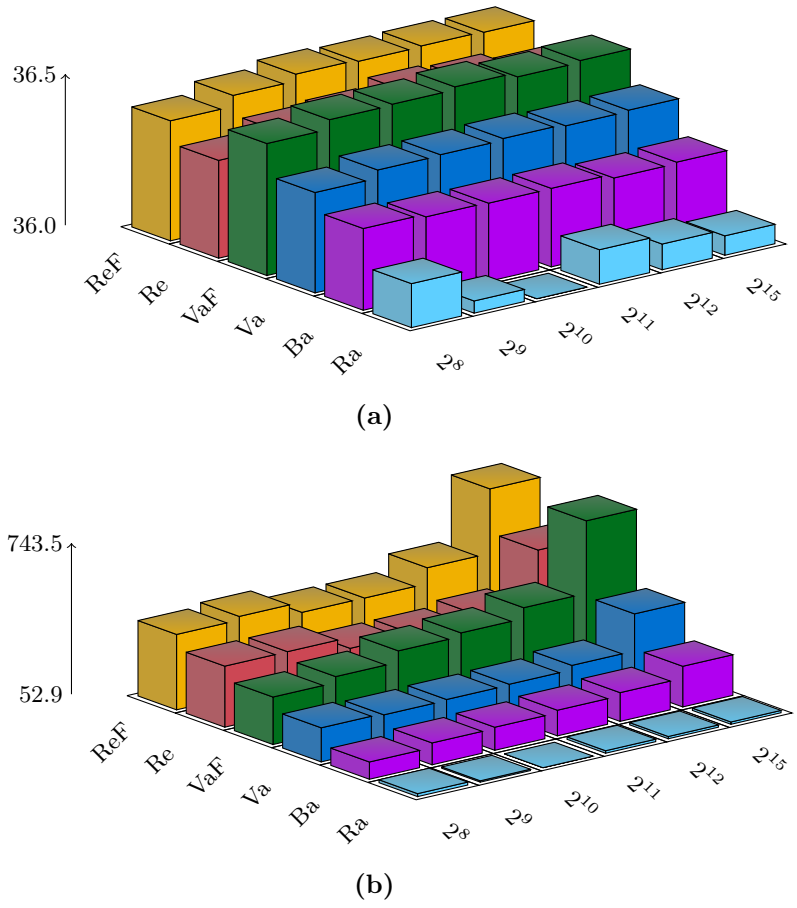


Figure 6.4: Our *RF*model with different number of tress (1, 5, 10, and 15) compared to dictionary learning based approaches with different sizes of the dictionary $\mathbf{D}$. We present the upscaling quality (a), the inference time (b), and the training time for ANR [178] (c) compared to our approach.

Another parameter in our implementation is the number of samples $\hat{N}$ considered for finding a splitting function σ in each node. We use reservoir sampling [188] to shrink the data $\mathbf{X}$ to $\min(|\mathbf{X}|, \hat{N})$ samples and use those to find a splitting function σ, which significantly reduces the training time without sacrificing quality, as we also demonstrate for other applications [155]. After fixing σ, all the data is forwarded to the left and right children for further growing the tree.

We present our results in Figure 6.5. The upscaling scores (Figure 6.5a) reveal that Va and Re are similarly good, while VaF and ReF give better results, confirming the importance of the regularization in the objective function (6.11). While Ba and Ra are inferior, it is worth mentioning that the simple balanced objective function (Ba) (6.12) gives relatively good results and being faster during training, cf., Figure 6.5b. On the other hand, the parameter $\hat{N}$ (evaluated for $2^{\{8,9,10,11,12,15\}}$) has little effect on the scores.



(a)

(b)

Figure 6.5: Influence of two parameters of the training scheme for the tree structure (splitting objective and subsample size $\hat{N}$) of our super-resolution forests on (a) the upscaling quality and (b) the training time of the trees. See text for more details.

6.5.6 Influence of the Number of Training Examples

The basic framework of Timofte et al. [178] used 91 images to extract the training patches for all methods, including GR, ANR, Zeyde et al. [204], and the neighborhood embedding methods. The generation of training data is described in Section 6.2. The number of patches extracted from the 91 training images is around $1.25 \cdot 10^6$, which is already a moderately sized training set for *RF*. However, the very recent follow-up work on ANR, A+ [177], as well as Dong et al. [44] showed improved results with larger training sets. In Section 6.5.3, we evaluated our *RF* methods with more training images, denoted RFL+ and ARFL+, which also yields improved results. Here, we provide an additional experiment that investigates this behavior in more detail.

We augmented the regular training set (91 images) with images from the BSDS500 data set [9], such that the total amount of images is 200, where around $5 \cdot 10^6$ patches are extracted, similar to [177]. In Section 6.5.3, we actually trained RFL+ with 291 images. However, here, we investigate the behavior of the training set size with a maximum of 200 images. The resulting scores and training times for RFL on `Set5` for a magnification factor of 2 are depicted in Figure 6.6. We observe that the upscaling quality steadily increases with the number of training samples, resulting in a final score of 36.68 dB with $5 \cdot 10^6$ training samples. In comparison, the usual setting of [178] has around $1.25 \cdot 10^6$ training samples, which corresponds to the fraction 0.25 and 36.55 dB in the figure.

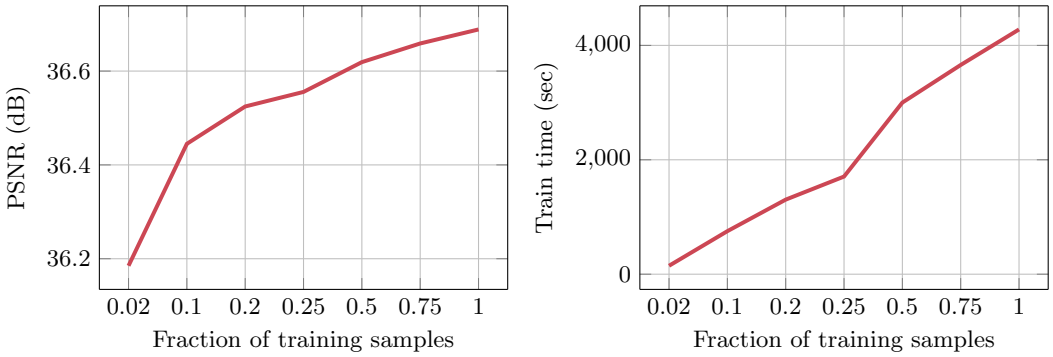
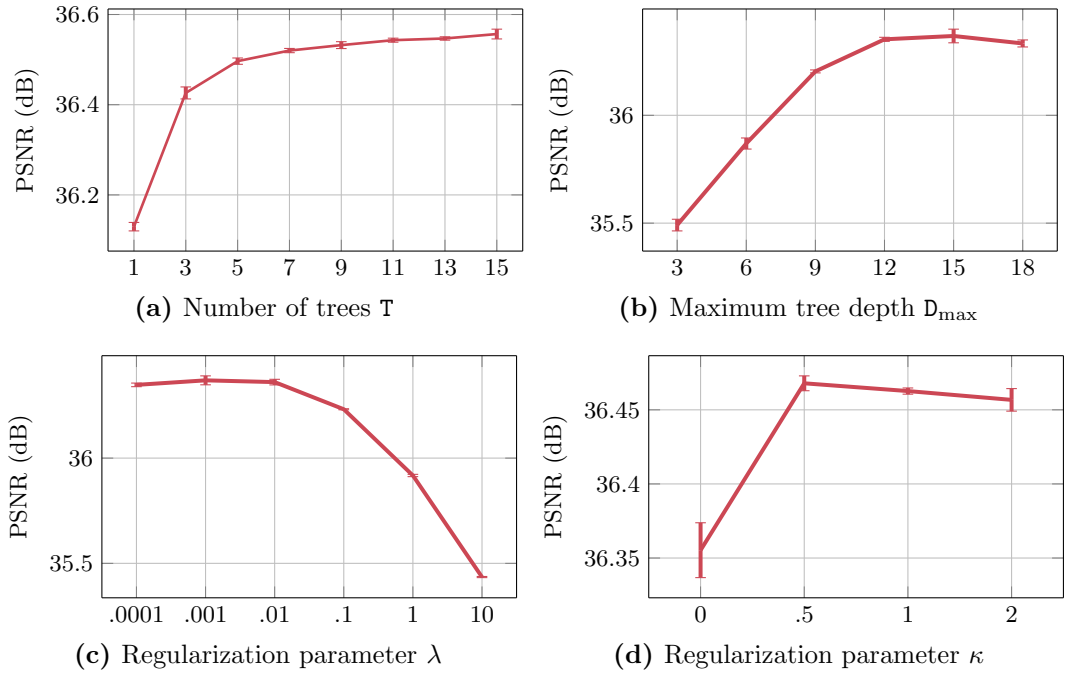


Figure 6.6: Influence of the number of training samples. For this experiment, the total number was around $5 \cdot 10^6$. We monitor the behavior of the upscaling quality in (a) and the training time in (b) for different fractions of the training set size.

6.5.7 Important Random Forest Parameters

Our final set of experiments on *SISR* investigate several important parameters of our method (beside those that have already been investigated). These parameters include the number of trees $\mathtt{T}$ in the ensemble, the maximum tree depth $\mathtt{D}_{\max}$, the regularization parameter λ for the linear regression in the leaf nodes, and, finally, the regularization

parameter κ for the splitting objective. Figure 6.7a shows the expected behavior of the parameter $\mathtt{T}$ for the *RF* approach. The performance steadily increases with increasing $\mathtt{T}$ until saturation, which is at around $\mathtt{T} = 13$ for this particular application. The second parameter in our evaluation is the maximum tree depth $\mathtt{D}_{\max}$, which has strong influence on training and inference times. From Figure 6.7b, we can see a saturation of the accuracy at depth $\mathtt{D}_{\max} = 12$ and even a slight drop in performance with too deep trees. Figure 6.7c indicates to use a rather low regularization parameter λ. In Figure 6.7d we can again see that the regularization in Equation (6.11) is important and should be activated.



(a) Number of trees $\mathtt{T}$ **(b)** Maximum tree depth $\mathtt{D}_{\max}$

(c) Regularization parameter λ **(d)** Regularization parameter κ

Figure 6.7: *RF* parameter evaluation on `Set5`: (a) number of trees $\mathtt{T}$, (b) maximum tree depth $\mathtt{D}_{\max}$, regularization parameters (c) λ, and (d) κ.

6.5.8 Computation Costs

We also investigate the computational costs of our method and of the framework from Timofte et al. [178] in more detail. In Table 6.4 we list the measured timings (in seconds) of the different computational stages for inference. These include feature computation, i.e., first and second order derivatives, (*CompFeat*), dimensionality reduction (*PCA*), *RF* evaluation (*RFL*), adding the bicubic upsampled image and the predicted high-frequency component (*Add Low+High*), and finally, merging all predicted patches via averaging (*OverlapAdd*). The results for RFL with different number of trees (1,5,10, and 15) are averaged over the images from `Set5` for an upscaling factor of 2. We can first observe that the *RF* inference (column *RFL*) scales linearly with the number of trees, when we

Variant	CompFeat	PCA	RFL	Add-Low+High	OverlapAdd
RFL-1	0.05 ± 0.04	0.00 ± 0.00	0.06 ± 0.03	0.02 ± 0.01	0.34 ± 0.25
RFL-5	0.05 ± 0.03	0.00 ± 0.00	0.25 ± 0.10	0.02 ± 0.01	0.34 ± 0.25
RFL-10	0.05 ± 0.04	0.00 ± 0.00	0.51 ± 0.20	0.02 ± 0.01	0.34 ± 0.25
RFL-15	0.05 ± 0.03	0.00 ± 0.00	0.77 ± 0.29	0.02 ± 0.01	0.34 ± 0.26

Table 6.4: Computational costs for different steps of the inference with the framework of [178] with a single CPU core measured in seconds.

Variant	CompFeat	PCA	RFL	Add-Low+High	OverlapAdd
RFL-1	0.05 ± 0.03	0.00 ± 0.00	0.06 ± 0.04	0.02 ± 0.01	0.34 ± 0.25
RFL-5	0.05 ± 0.04	0.00 ± 0.00	0.17 ± 0.07	0.02 ± 0.01	0.34 ± 0.25
RFL-10	0.05 ± 0.04	0.01 ± 0.00	0.31 ± 0.13	0.02 ± 0.01	0.34 ± 0.25
RFL-15	0.05 ± 0.04	0.01 ± 0.00	0.44 ± 0.18	0.01 ± 0.01	0.34 ± 0.25

Table 6.5: Computational costs for different steps of the inference with the framework of [178] with two CPU cores measured in seconds.

use a single CPU core. Further, we can see that feature computation, dimensionality reduction and summation of bicubic and high-frequency components take relatively little time. However, the implementation of the patch averaging (column *OverlapAdd*) seems to be suboptimal as it takes relatively long. With an optimized implementation of these parts, one could potentially reach similar efficiency as the SRCNN approach [44].

RF can be easily parallelized on multiple CPU cores. When we also exploit this fact, which we did not do for all previous experiments, we can decrease the computational costs of our super-resolution approach. In Table 6.5 we present the same results as above (cf., Table 6.4) but with access to two CPU cores. Intuitively, we can observe faster inference for the *RF* inference (column *RFL*). However, one has to be aware of the fact that using too much cores could also lead to increased inference times due to computational overhead. This can happen when the images to be upscaled are relatively small and only a few patches get extracted. Obviously, the larger the images, the more benefit you get by using more CPU cores.

6.6 Summary

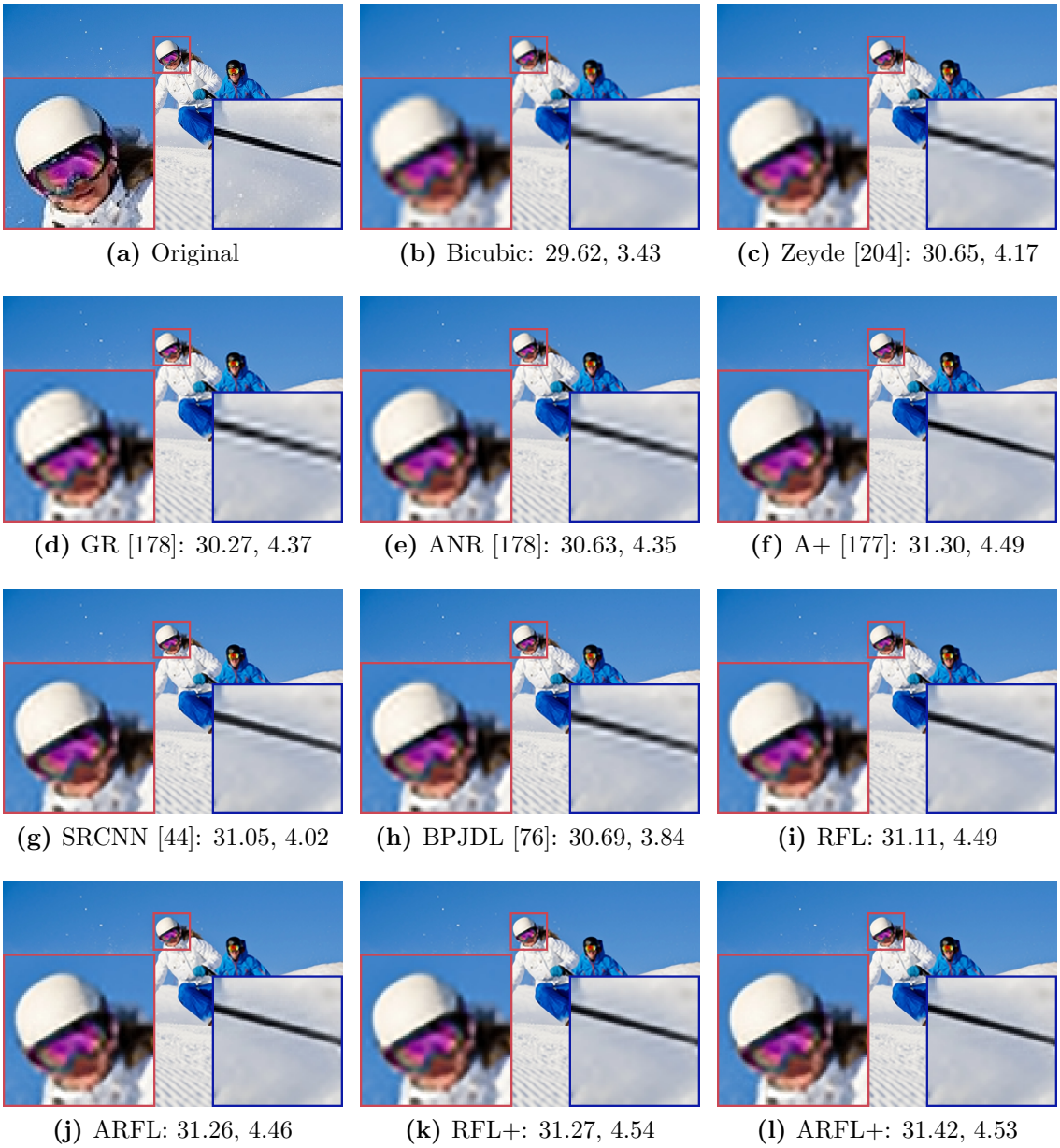
In this chapter, we present a new approach for single image super-resolution via *RF*. We show the close relation between recent sparse coding based approaches and locally linear regression. We exploit this connection and avoid the detour of using a sparse-coded dictionary to learn the mapping from low- to high-resolution images. Instead, we follow a more direct approach with a random regression forest formulation. Our *RF* approach to *SISR* builds on linear prediction models in the leaf nodes instead of typically used constant models. Additionally, it employs a new regularization on the splitting objective function which operates on the output as well as the input domain of the data.

The use of *RF* for this particular application proved to be highly effective and shows some critical benefits over previous dictionary and direct regression approaches. Beside the fact that the *RF* formulation naturally fits into the locally linear regression framework, the hierarchical structure of this model gives some nice properties. It enables the use of a much larger effective codebook without decreasing the access time compared to a flat structure used in previous work. Each leaf node of the trees can be considered as one entry in a codebook. Accessing the codebook entries happens in logarithmic time complexity, compared to a linear one for flat codebooks. Moreover, the ensemble of several trees allows for an even larger codebook when considering all the possible combinations of leaf nodes. This considerably larger codebook is most probably the reason our approach yields better performance on the benchmarks.

However, we also have to mention the downside of the large codebook, which obviously leads to a large model size, in terms of memory usage. This can be become a problem when deploying such a system to mobile devices, where super-resolution is definitely an interesting task. Fortunately, there already exists attempts to train the *RF* in a way to get more compact models without losing its predictive power [138].

Another benefit of the proposed approach is the relatively fast training time. Compared with previous works that have to create a dictionary with sparse coding as a building block, e.g., [177, 178, 204], *RF* can be trained relatively fast and can be easily parallelized over multiple CPU cores. Especially when the training size increases, which we showed to be important for good results, fast training comes in handy.

Finally, we showed that our proposed training scheme, *ARF*, again improves over standard *RF* for this application. We observed a considerable boost in the final super-resolution quality by only exchanging the *RF* training scheme with our proposed one. Our results confirm the effectiveness of this general super-resolution approach with *RF* on different benchmarks, where we outperform the current state-of-the-art. The inference of our *RF* model is among the fastest methods and the training time is typically one or several orders of magnitude less than related approaches.



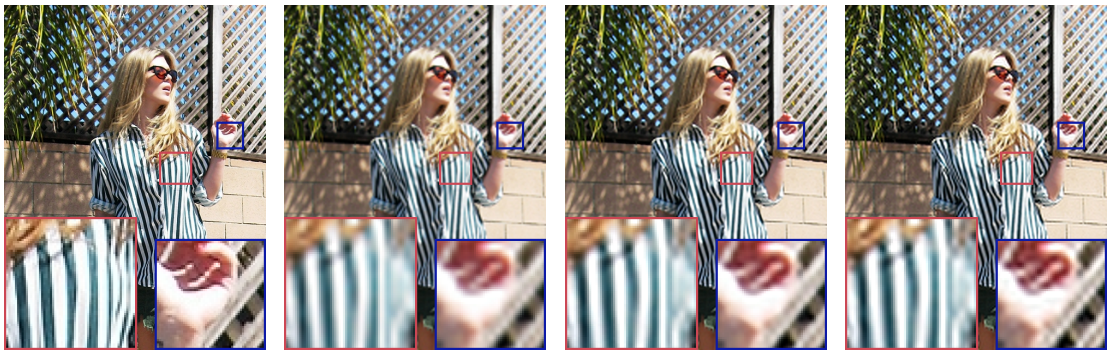
(a) Original **(b)** Bicubic: 29.62, 3.43 **(c)** Zeyde [204]: 30.65, 4.17

(d) GR [178]: 30.27, 4.37 **(e)** ANR [178]: 30.63, 4.35 **(f)** A+ [177]: 31.30, 4.49

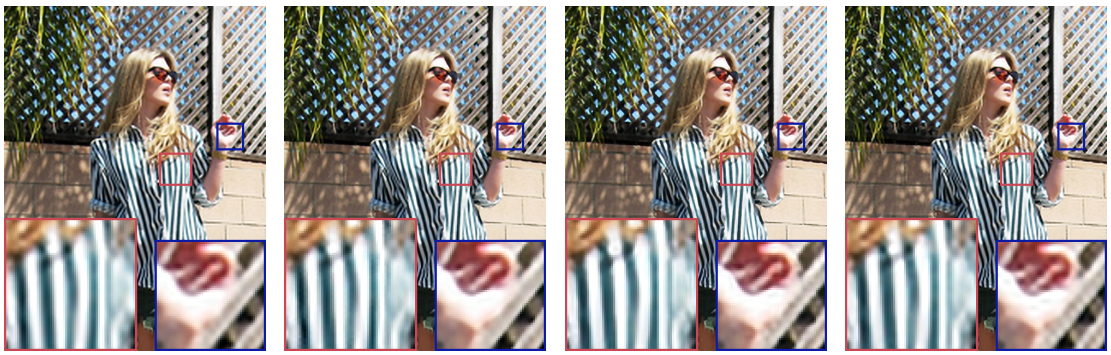
(g) SRCNN [44]: 31.05, 4.02 **(h)** BPJDL [76]: 30.69, 3.84 **(i)** RFL: 31.11, 4.49

(j) ARFL: 31.26, 4.46 **(k)** RFL+: 31.27, 4.54 **(l)** ARFL+: 31.42, 4.53

Figure 6.8: Qualitative results of state-of-the-art methods for upscaling factor x3 on image `skiing`. The numbers in the subcaptions refer to PSNR and IFC scores, respectively. Best viewed in color and digital zoom.



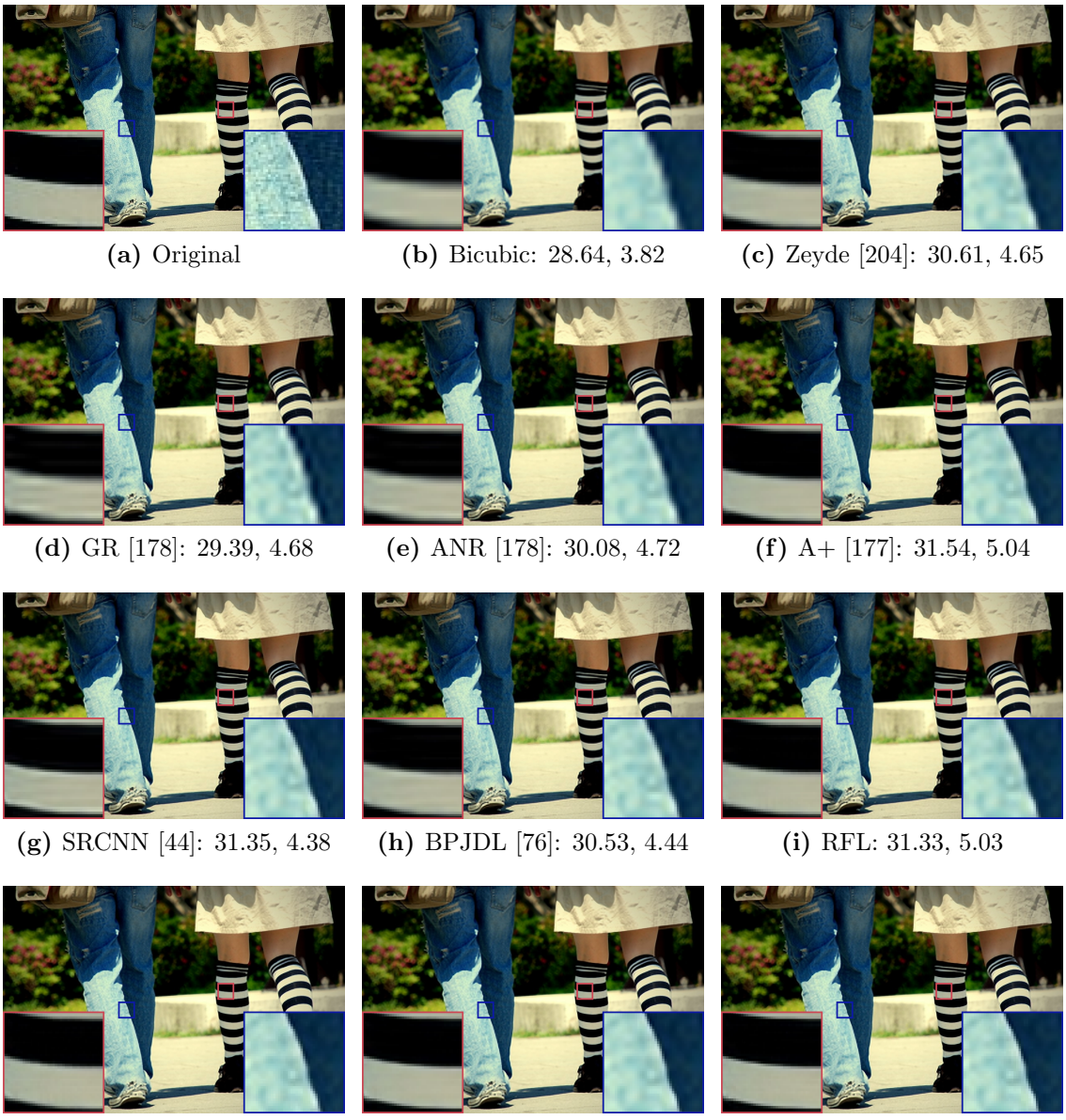
(a) Original **(b)** Bicubic: 21.79, 3.73 **(c)** Zeyde [204]: 22.75, 4.47 **(d)** GR [178]: 22.51, 4.60



(e) ANR [178]: 22.73, 4.56 **(f)** A+ [177]: 23.15, 4.77 **(g)** SRCNN [44]: 23.36, 4.34 **(h)** BPJDL [76]: 22.95, 4.63

(i) RFL: 23.19, 4.80 **(j)** ARFL: 23.32, 4.78 **(k)** RFL+: 23.34, 4.85 **(l)** ARFL+: 23.48, 4.81

Figure 6.9: Qualitative results of state-of-the-art methods for upscaling factor x3 on image `striped_girl_2`. The numbers in the subcaptions refer to PSNR and IFC scores, respectively. Best viewed in color and digital zoom.



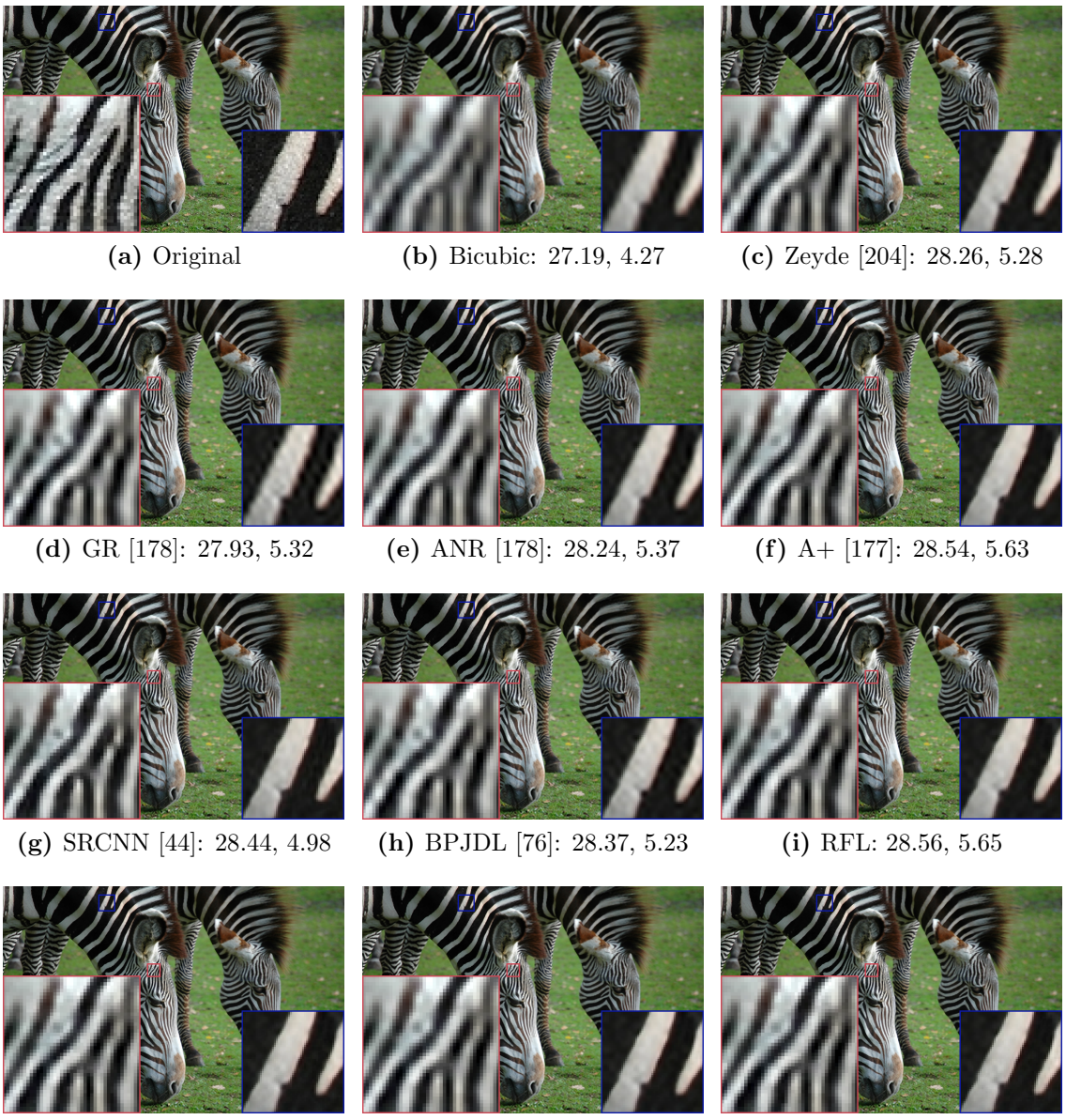
(a) Original **(b)** Bicubic: 28.64, 3.82 **(c)** Zeyde [204]: 30.61, 4.65

(d) GR [178]: 29.39, 4.68 **(e)** ANR [178]: 30.08, 4.72 **(f)** A+ [177]: 31.54, 5.04

(g) SRCNN [44]: 31.35, 4.38 **(h)** BPJDL [76]: 30.53, 4.44 **(i)** RFL: 31.33, 5.03

(j) ARFL: 31.60, 5.05 **(k)** RFL+: 31.82, 5.18 **(l)** ARFL+: 32.02, 5.17

Figure 6.10: Qualitative results of state-of-the-art methods for upscaling factor x3 on image `legs`. The numbers in the subcaptions refer to PSNR and IFC scores, respectively. Best viewed in color and digital zoom.



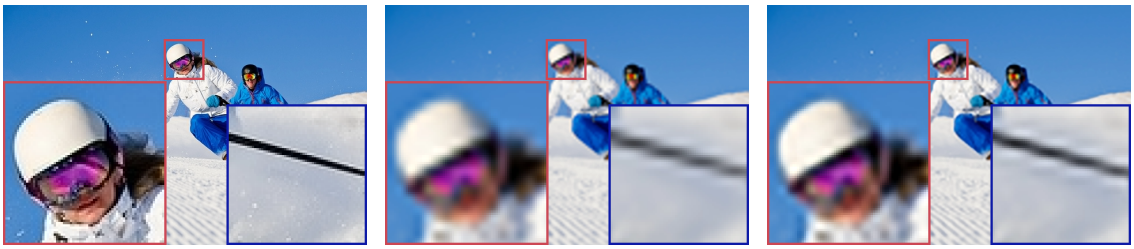
(a) Original **(b)** Bicubic: 27.19, 4.27 **(c)** Zeyde [204]: 28.26, 5.28

(d) GR [178]: 27.93, 5.32 **(e)** ANR [178]: 28.24, 5.37 **(f)** A+ [177]: 28.54, 5.63

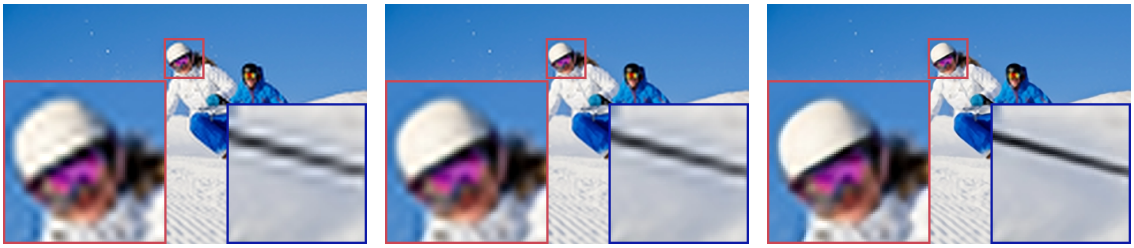
(g) SRCNN [44]: 28.44, 4.98 **(h)** BPJDL [76]: 28.37, 5.23 **(i)** RFL: 28.56, 5.65

(j) ARFL: 28.60, 5.63 **(k)** RFL+: 28.63, 5.73 **(l)** ARFL+: 28.67, 5.68

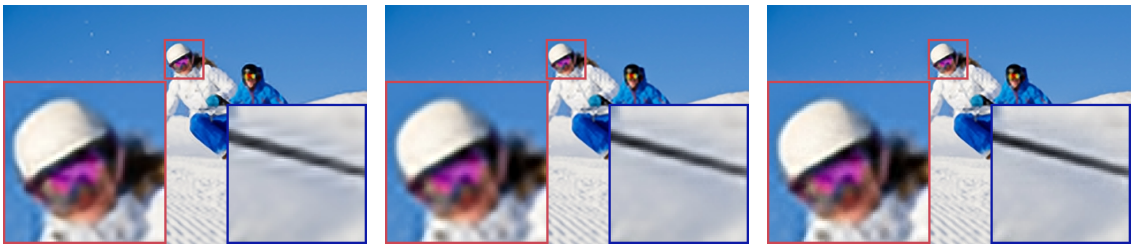
Figure 6.11: Qualitative results of state-of-the-art methods for upscaling factor x3 on image `zebras`. The numbers in the subcaptions refer to PSNR and IFC scores, respectively. Best viewed in color and digital zoom.



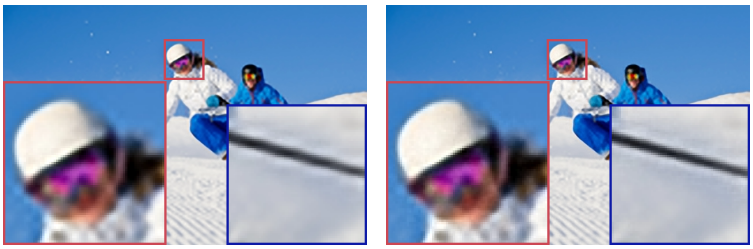
(a) Original **(b)** Bicubic: 28.15, 2.16 **(c)** Zeyde [204]: 29.05, 2.66



(d) GR [178]: 28.76, 2.74 **(e)** ANR [178]: 29.07, 2.75 **(f)** A+ [177]: 29.59, 2.84

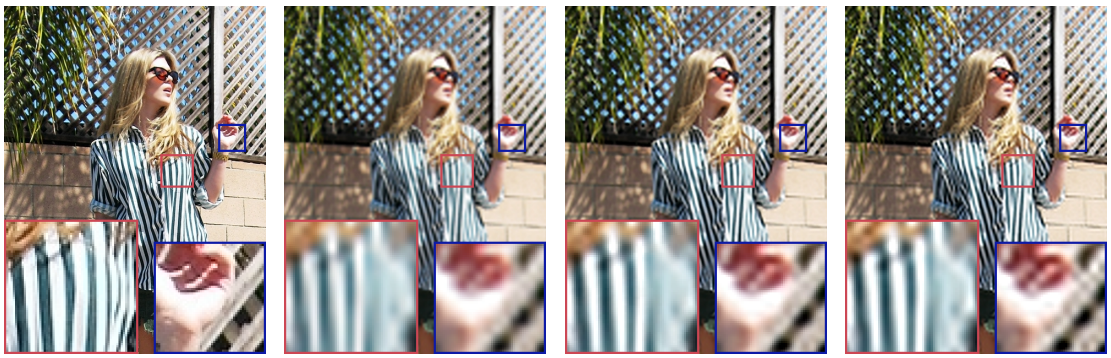


(g) SRCNN [44]: 29.29, 2.50 **(h)** RFL: 29.41, 2.81 **(i)** ARFL: 29.52, 2.77



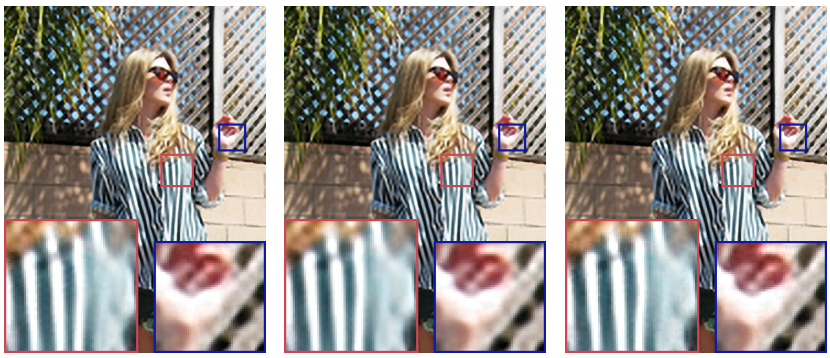
(j) RFL+: 29.49, 2.84 **(k)** ARFL+: 29.62, 2.82

Figure 6.12: Qualitative results of state-of-the-art methods for upscaling factor x4 on image `skiing`. The numbers in the subcaptions refer to PSNR and IFC scores, respectively. Best viewed in color and digital zoom.



(a) Original **(b)** Bicubic: 20.21, 2.39 **(c)** Zeyde [204]: 21.03, 2.88 **(d)** GR [178]: 20.93, 2.95

(e) ANR [178]: 21.09, 2.96 **(f)** A+ [177]: 21.21, 3.03 **(g)** SRCNN [44]: 21.31, 2.79 **(h)** RFL: 21.32, 3.06



(i) ARFL: 21.34, 3.02 **(j)** RFL+: 21.37, 3.07 **(k)** ARFL+: 21.42, 3.02

Figure 6.13: Qualitative results of state-of-the-art methods for upscaling factor x4 on image `striped_girl_2`. The numbers in the subcaptions refer to PSNR and IFC scores, respectively. Best viewed in color and digital zoom.

(a) Original **(b)** Bicubic: 26.99, 2.55 **(c)** Zeyde [204]: 28.61, 3.15

(d) GR [178]: 27.62, 3.12 **(e)** ANR [178]: 28.22, 3.18 **(f)** A+ [177]: 29.44, 3.38

(g) SRCNN [44]: 29.17, 2.90 **(h)** RFL: 29.22, 3.32 **(i)** ARFL: 29.43, 3.28

(j) RFL+: 29.70, 3.45 **(k)** ARFL+: 29.86, 3.42

Figure 6.14: Qualitative results of state-of-the-art methods for upscaling factor x4 on image `legs`. The numbers in the subcaptions refer to PSNR and IFC scores, respectively. Best viewed in color and digital zoom.

(a) Original **(b)** Bicubic: 25.56, 2.85 **(c)** Zeyde [204]: 26.50, 3.55

(d) GR [178]: 26.22, 3.56 **(e)** ANR [178]: 26.49, 3.62 **(f)** A+ [177]: 26.77, 3.80

(g) SRCNN [44]: 26.61, 3.37 **(h)** RFL: 26.79, 3.79 **(i)** ARFL: 26.82, 3.74

(j) RFL+: 26.86, 3.85 **(k)** ARFL+: 26.90, 3.80

Figure 6.15: Qualitative results of state-of-the-art methods for upscaling factor x4 on image `zebras`. The numbers in the subcaptions refer to PSNR and IFC scores, respectively. Best viewed in color and digital zoom.

CHAPTER 7

Conclusion and Outlook

In this thesis we addressed novel training algorithms for Random Forests (RF) and their applications to different computer vision problems. We focused on classification and regression tasks for high-level applications like object detection and pose estimation, but also for low-level applications like Single Image Super-Resolution (SISR). The motivation for investigating this flexible machine learning algorithm was the great success in the field of computer vision. As already discussed in Chapter 2, *RF* possess many benefits in terms of computational complexity and generalization capabilities compared to other machine learning algorithms, often making them the first choice for several applications. Arguably, the most prominent one is the human pose estimation system [163] in the Microsoft® Kinect™ that enables the estimation of the human skeleton in real-time from a single depth sensor. Beside the good generalization capabilities and the easy implementation, *RF* are also often chosen due to the fast training and inference times. All trees operate in isolation which allows for exploiting multiple CPU cores in an easy way. Inference in a tree is also very efficient as only a single path from the root node to a leaf node has to be processed, i.e., conditional computation. However, the training procedure of *RF* is rather uncommon compared to other state-of-the-art machine learning algorithms and bears some potential drawbacks.

Most other machine learning algorithms like Support Vector Machine (SVM), Neural Networks (NN), or Boosting (Boosting) minimize a loss (or risk) function in order to find the parameters of the full model. In *RF*, however, each tree is trained in isolation without knowing anything about the rest of the model. While this is often attributed as the main advantage and essential ingredient of *RF*, in Chapter 3 we argue that there is still room for improvement by exploiting information of the full model, i.e., all trees.

7.1 Summary

The main result of this thesis is a novel *RF* training algorithm that optimizes a global loss function over all the trees while maintaining the computational benefits during both training and inference. To do so, we combine ideas from both Gradient Boosting (GB) and *RF* and propose Alternating Decision and Regression Forests (ADRF), which thoroughly describe in Chapter 3. We present a regression and a classification formulation and conduct a dense evaluation on standard machine learning benchmarks. These experiments include a comparison with baseline machine learning algorithms, a detailed evaluation of several parameters, and an analysis of the behavior of some intrinsics of the novel algorithm.

The remaining chapters of the thesis contain several applications in the field of computer vision where the proposed algorithm can be easily applied and yields promising results. We start with generic object detection in Chapter 4 where we present two different strategies of attacking this task. The first one is based on local evidence of an object (see Section 4.1) where we build on the popular Hough Forests (HF) framework [62] to integrate our ideas. After reviewing the basic concepts of the object detection model, we show how to integrate the *ADRF* learning scheme, followed by a detailed evaluation on typically used detection benchmarks. The second object detection strategy (see Section 4.2) is based on a rigid object template this is often combined with a *Boosting* learning algorithm. Here, we not only show that *RF* can also be employed as learner and that Alternating Decision Forests (ADF) yields better results, but also that the flexibility of the general *RF* framework can be exploited to more accurately localize the objects in the images. *RF* can be easily used for joint classification and regression tasks, see *HF* [62]. In Section 4.2, we thus present a way to overcome the problem of predicting a fixed-size bounding box, an issue of many detection approaches that is inherent by design. Our formulation can predict the probability for an image window to either belonging to fore- or background, but also estimate the aspect ratio of the bounding box that enframes the object.

In Chapter 5, we present results of *ADRF* on human head pose estimation from depth images. We tackle this problem from two different views, a local and a holistic one, both relying on *RF*. First, we follow Fanelli et al. [50] to adapt the *HF* framework for this particular task. Then, we also propose a new holistic pose estimation method, which estimates the position and orientation of the head from a single image patch capturing the whole object. We again successfully integrate the *ADRF* training scheme into both approaches. The results on the standard benchmark data set confirm the effectiveness of the proposed algorithm when we compare with state-of-the-art. Finally, in Chapter 6 we switch from high-level to low-level computer vision tasks and propose a *RF* based approach to *SISR*. We present a locally-linear regression formulation with *RF* that achieves state-of-the-art performance compared to other dictionary learning and direct regression methods. Moreover, the use of Alternating Regression Forests (ARF) significantly boosts the performance.

7.2 Discussion

A question that naturally arises about a novel extension of an existing algorithm is: Why and when should it be applied? The obvious answer is that our proposed algorithm seems to outperform plain *RF* on several tasks for both classification and regression problems without significant increase in computational costs. We also want to mention that we employed *ADRF* for other tasks as well. While we did not always achieved significantly better results, we almost never encountered worse performance, which is an important insight to be considered. Nevertheless, for many applications *ADRF* outperforms plain *RF* as can be seen in Chapters 3 to 6. Another answer to the question above is that the novel algorithm can be readily replaced with *RF* in any application, which has several reasons. First, the resulting model and, thus, also the inference phase is exactly the same as in *RF*. Second, the input to the proposed training algorithm is also the same as in *RF*, i.e., no additional data has to be provided. One disadvantage that might arise is the additional training time (often negligible), due to the synchronization between the trees and the fact that prediction models have to be computed for each node (also intermediate ones). However, training *ADRF* is still clearly faster than *GB*.

Another delicate issue to be discussed is the topic of tree correlation that can deteriorate the generalization performance of *RF* (or ensemble methods in general), see Section 2.2.4.4. In the extreme case, all trees are exactly the same making the ensemble and the effect of averaging obsolete. *ADRF* exploits information from all the trees in the ensemble to find splitting functions in a single tree, which obviously influences the training. Nevertheless, only the training labels are changing (either via pseudo targets for regression or a weighting for classification), which is the same as in *GB* and puts a bias on the objective for finding splitting functions. However, the search for splitting functions in the trees stays the same. It is still a randomized grid search, which loosens the bias. We empirically investigate this issue for both, the classification and the regression case, with an interesting outcome. Contrary to our prior belief, the correlation between the trees becomes even lower compared to plain *RF*. The reason might be that the integration of the common loss function makes the model aware of the other trees in the ensemble, which also make predictions. Given this knowledge, the goal of a single tree is not to make perfect predictions on the current task, but to act as a good component in the ensemble such that the overall results get better. Still, we do not have any theoretical results showing that the generalization error is not affected by the novel training scheme. However, these empirical results are encouraging and indicate that *ADRF* does not hurt generalization but rather improves it.

7.3 Outlook

When looking at future avenues to continue research on this topic, one definitely has to investigate different choices of the loss function that can be integrated into the *ADRF*

framework. In general, any differentiable loss function (also non-convex) can be used in the proposed framework. The flexibility to use almost any kind of loss function also paves the way for integrating semi-supervised or multiple-instance learning loss functions directly into the tree growing process, which are highly interesting research directions due to the ever increasing amount of data available. Another, potentially even more interesting direction is to integrate a model-based inference algorithm into the training of *ADRF* which could also be enabled via the loss function, cf. [83, 84, 121]. One example might be a hand model that is fit to the data for articulated hand pose estimation [173, 175] or a low-level image model for single image super-resolution. This might find many applications in the field of computer vision. For object detection with the *HF* framework, Redondo-Cabrera and Lopez-Sastre [137] already showed that integrating the full inference process (including Hough voting and non-maxima suppression) into the loss function of *ADRF* significantly boosts the performance.

As we worked on supervised machine learning and proposed a new *RF* based algorithm, we must not forget about the current trend and success of deep learning [92], i.e., (convolutional) *NN*. *NN* have a linear model at the very last layer that makes the final prediction, e.g., a logistic regression or support vector machine layer. However, they build on a fully trained (end-to-end) representation that is an inherent part of the classifier and the input to this last classification layer. On the other hand, *RF* rely on a pre-computed feature representation and is not concerned with learning this representation, although there exist attempts to learn it, e.g., [118, 164]. There also exist attempts to introduce the concept of feature learning via neural networks into *RF* with good results on semantic image labeling [141]. Still, a limiting factor with integrating a learned representation into the *RF* is that computing gradients with respect to the input representation is almost impossible without delicate approximations, because of the hierarchical, hard splitting functions. On the other hand, *RF* possess computational benefits during both training and inference phases compared to *NN*. Of course, *NN* can benefit from highly tuned implementations and graphical processing units, but the number of computations is significantly less with *RF*. Moreover, besides the obviously easy deployment on parallel architectures, *RF* (or trees in general) can rely on conditional computations, i.e., data is processed only by subparts of the full model which is conditioned on previous computations. A promising direction for future research definitely is to unify the advantages of both worlds into a common principled learning algorithm.

APPENDIX A

List of Acronyms

ACF	Aggregate Channel Features
ADF	Alternating Decision Forests
ADRF	Alternating Decision and Regression Forests
ARF	Alternating Regression Forests
AUC	Area Under Curve
Boosting	Boosting
DPM	Deformable Parts Model
GB	Gradient Boosting
HF	Hough Forests
ISM	Implicit Shape Model
ML	Machine Learning
NN	Neural Networks
RF	Random Forests
SISR	Single Image Super-Resolution
SVM	Support Vector Machine
ToF	Time of Flight

APPENDIX B

List of Publications

My work at the Institute for Computer Graphics and Vision led to the following peer-reviewed publications. For the sake of completeness of this thesis, they are listed in chronological order.

B.1 2011

Improving Classifiers with Unlabeled Weakly-Related Videos

Christian Leistner, Martin Godec, Samuel Schulter, Amir Saffari, Manuel Werlberger, and Horst Bischof
In: *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*
June 2011, Colorado Springs, USA
(Accepted for poster presentation)

On-line Hough Forests

Samuel Schulter, Christian Leistner, Peter M. Roth, Luc Van Gool, and Horst Bischof
In: *Proceedings of British Machine Vision Conference (BMVC)*
September 2011, Dundee, United Kingdom
(Accepted for poster presentation)

B.2 2012

Discriminative Hough Forests for Object Detection

Paul Wohlhart, Samuel Schulter, Martin Köstinger, Peter M. Roth, and Horst Bischof
In: *Proceedings of British Machine Vision Conference (BMVC)*
September 2012, Guildford, United Kingdom
(Accepted for poster presentation)

B.3 2013

Alternating Decision Forests

Samuel Schulter, Paul Wohlhart, Christian Leistner, Amir Saffari, Peter M. Roth, and Horst Bischof
In: *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*
June 2013, Portland, USA
(Accepted for poster presentation)

Ordinal Random Forests for Object Detection

Samuel Schulter, Peter M. Roth, and Horst Bischof
In: *Proceedings of German Conference on Pattern Recognition (GCPR)*
September 2013, Saarbrücken, Germany
(Accepted for oral presentation)

Unsupervised Object Discovery and Segmentation in Videos

Samuel Schulter, Christian Leistner, Peter M. Roth, and Horst Bischof
In: *Proceedings of British Machine Vision Conference (BMVC)*
September 2013, Bristol, United Kingdom
(Accepted for oral presentation)

Alternating Regression Forests for Object Detection and Pose Estimation

Samuel Schulter, Christian Leistner, Paul Wohlhart, Peter M. Roth, and Horst Bischof
In: *Proceedings of International Conference on Computer Vision (ICCV)*
December 2013, Sydney, Australia
(Accepted for poster presentation)

B.4 2014

Accurate Object Detection with Joint Classification-Regression Random Forests

Samuel Schulter, Christian Leistner, Paul Wohlhart, Peter M. Roth, and Horst Bischof
In: *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*
June 2014, Columbus, USA
(Accepted for poster presentation)

Hough Forests Revisited: An Approach to Multiple Instance Tracking from Multiple Cameras

Georg Poier, Samuel Schulter, Sabine Sternig, Peter M. Roth, and Horst Bischof
In: *Proceedings of German Conference on Pattern Recognition (GCPR)*
September 2014, Münster, Germany
(Accepted for poster presentation)

B.5 2015

Fast and Accurate Image Upscaling with Super-Resolution Forests

Samuel Schulter, Christian Leistner and Horst Bischof
In: *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*
June 2015, Boston, USA
(Accepted for poster presentation)

Hybrid One-Shot 3D Hand Pose Estimation by Exploiting Uncertainties

Georg Poier, Konstantinos Roditakis, Samuel Schulter, Damien Michel, Horst Bischof and Antonis A. Argyros
In: *Proceedings of British Machine Vision Conference (BMVC)*
September 2015, Swansea, United Kingdom
(Accepted for oral presentation)

Bibliography

[1] Aharon, M., Elad, M., and Bruckstein, A. (2006). K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation. *Transactions on Signal Processing*, 54(11):4311–4322. (page 111)

[2] Alexe, B., Deselaers, T., and Ferrari, V. (2010). What is an Object? In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 63)

[3] Amit, Y. and Geman, D. (1994). Randomized Inquiries About Shape; An Application to Handwritten Digit Recognition. Technical Report 401, Department of Statistics, University of Chicago. (page 3, 22)

[4] Amit, Y. and Geman, D. (1997). Shape Quantization and Recognition with Randomized Trees. *Neural Computation*, 9(7):1545–1588. (page 3, 14, 22, 41)

[5] Anderson, T. W. and Darling, D. A. (1952). Asymptotic Theory of Certain "Goodness-of-Fit" Criteria Based on Stochastic Processes. *Annals of Mathematical Statistics*, 23(2):193–212. (page 44, 45)

[6] Andrews, S., Tsochantaridis, I., and Hofmann, T. (2002). Support Vector Machines for Multiple-Instance Learning. In *Advances Conference on Neural Information Processing Systems*. (page 15)

[7] Andriluka, M., Pishchulin, L., Gehler, P., and Schiele, B. (2014). 2D Human Pose Estimation: New Benchmark and State of the Art Analysis. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 5)

[8] Andriluka, M., Roth, S., and Schiele, B. (2008). People-Tracking-by-Detection and People-Detection-by-Tracking. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 73, 87)

[9] Arbelaez, P., Maire, M., Fowlkes, C., and Malik, J. (2011). Contour Detection and Hierarchical Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):898–916. (page 119, 126)

[10] Babenko, B., Yang, M.-H., and Belongie, S. (2009). Visual Tracking with Online Multiple Instance Learning. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 20)

[11] Balasubramanian, V. N., Ye, J., and Panchanathan, S. (2007). Biased Manifold Embedding: A Framework for Person-Independent Head Pose Estimation. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 97)

[12] Benenson, R., Mathias, M., Timofte, R., and Gool, L. v. (2012a). Fast stixel computation for fast pedestrian detection. In *ECCV Workshops*. (page 81)

[13] Benenson, R., Mathias, M., Timofte, R., and Gool, L. v. (2012b). Pedestrian detection at 100 frames per second. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 20, 81)

[14] Benenson, R., Mathias, M., Tuytelaars, T., and Gool, L. v. (2013). Seeking the strongest rigid detector. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 63, 80, 81, 83, 86)

[15] Bennett, K. P. and Demiriz, A. (1998). Semi-Supervised Support Vector Machines. In *Advances Conference on Neural Information Processing Systems*. (page 15)

[16] Bevilacqua, M., Roumy, A., Guillemot, C., and Alberi Morel, M.-L. (2012). Low-Complexity Single-Image Super-Resolution based on Nonnegative Neighbor Embedding. In *Proc. British Machine Vision Conference*. (page 119, 122, 123, 124)

[17] Bileschi, S. M. (2006). *StreetScenes: Towards Scene Understanding in Still Images*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science. (page 87)

[18] Bishop, C. M. (2007). *Pattern Recogntion and Machine Learning*. Springer. (page 11, 12, 14, 116)

[19] Blaschko, M. B. and Lampert, C. H. (2008). Learning to Localize Objects with Structured Output Regression. In *Proc. European Conference on Computer Vision*. (page 82)

[20] Bouvrie, J. V. and Sinha, P. (2007). Visual object concept discovery: Observations in congenitally blind children, and a computational approach. *Neural Computation*, 70(13–15):2218–2233. (page 1)

[21] Breiman, L. (1996). Bagging Predictors. *Machine Learning*, 24(2):123–140. (page 17)

[22] Breiman, L. (1999). Prediction Games and Arcing Algorithms. *Neural Computation*, 11(7):1493–1517. (page 18)

[23] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1):5–32. (page 3, 7, 14, 22, 31, 32, 34, 41, 42, 44, 50, 53, 59)

[24] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification And Regression Trees*. Chapman & Hall/CRC. (page 12, 22)

[25] Breitenstein, M. D., Kuettel, D., Weise, T., and van Gool, L. (2008). Real-Time Face Pose Estimation from Single Range Images. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 98, 99)

[26] Brostow, G. J., Shotton, J., Fauqueur, J., and Cipolla, R. (2008). Segmentation and Recognition using Structure from Motion Point Clouds. In *Proc. European Conference on Computer Vision*. (page 22)

[27] Campo, T. E. d., Babu, B. R., and Varma, M. (2009). Character Recognition in Natural Images. In *Proc. International Conference on Computer Vision Theory and Applications*. (page 45)

[28] Caruana, R., Karampatziakis, N., and Yessenalina, A. (2008). An empirical Evaluation of Supervised Learning in High Dimensions. In *International Conference on Machine Learning*. (page 3)

[29] Chang, H., Yeung, D.-Y., and Xiong, Y. (2004). Super-Resolution Through Neighbor Embedding. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 113, 119, 120, 122, 123, 124)

[30] Chapelle, O., Schölkopf, B., and Zien, A. (2010). *Semi-Supervised Learning*. MIT Press. (page 15)

[31] Cheng, M.-M., Zhang, Z., Lin, W.-Y., and Torr, P. (2014). BING: Binarized Normed Gradients for Objectness Estimation at 300fps. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 63)

[32] Cootes, T. F., Ionita, M., Lindner, C., and Sauer, P. (2012). Robust and Accurate Shape Model Fitting using Random Forest Regression Voting. In *Proc. European Conference on Computer Vision*. (page 5, 72)

[33] Crammer, K. and Singer, Y. (2001). On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. *Journal of Machine Learning Research*, 2(12):265–292. (page 31)

[34] Criminisi, A. and Shotton, J. (2013). *Decision Forests for Computer Vision and Medical Image Analysis*. Springer. (page 5, 14, 22)

[35] Dai, D., Timofte, R., and van Gool, L. (2015). Jointly Optimized Regressors for Image Super-resolution. In *Eurographs*. (page 112)

[36] Dalal, N. and Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 63, 64, 66, 69, 71, 76, 80, 82)

[37] Dantone, M., Gall, J., Fanelli, G., and Gool, L. v. (2012). Real-time Facial Feature Detection using Conditional Regression Forests. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 5, 6, 63, 72)

[38] Dantone, M., Gall, J., Leistner, C., and van Gool, L. (2013). Human Pose Estimation using Body Parts Dependent Joint Regressors. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 5, 6, 22, 72)

[39] De La Torre, F. and Kanade, T. (2006). Discriminative Cluster Analysis. In *International Conference on Machine Learning*. (page 14)

[40] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 4)

[41] Dollár, P., Appel, R., Belongie, S., and Perona, P. (2014). Fast Feature Pyramids for Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. (page 7, 20, 63, 69, 76, 80, 81, 82, 83, 86, 88, 89, 90, 103)

[42] Dollár, P., Tu, Z., Perona, P., and Belongie, S. (2009). Integral Channel Features. In *Proc. British Machine Vision Conference*. (page 20, 63, 69, 76, 80, 81, 83, 86)

[43] Dollár, P. and Zitnick, L. (2013). Structured Forests for Fast Edge Detection. In *Proc. International Conference on Computer Vision*. (page 4, 30, 80, 119)

[44] Dong, C., Change Loy, C., He, K., and Tang, X. (2014). Learning a deep convolutional network for image super-resolution. In *Proc. European Conference on Computer Vision*. (page 109, 111, 112, 115, 120, 121, 122, 123, 126, 128, 130, 131, 132, 133, 134, 135, 136, 137)

[45] Duchon, C. E. (1979). Lanczos Filtering in One and Two Dimensions. *Journal of Applied Meteorology*, 18(8):1016–1022. (page 109)

[46] Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification*. Wiley. (page 11, 12, 17)

[47] Everingham, M., Gool, L. v., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 88(2):303–338. (page 87)

[48] Fanelli, G., Dantone, M., Gall, J., Fossati, A., and Gool, L. v. (2013). Random Forests for Real Time 3D Face Analysis. *International Journal of Computer Vision*, 101(3):437–458. (page 5, 72, 97, 99, 102, 104)

[49] Fanelli, G., Gall, J., and Gool, L. v. (2011a). Real Time Head Pose Estimation with Random Regression Forests. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 6, 7, 22, 96, 99, 100, 101)

[50] Fanelli, G., Weise, T., Gall, J., and Gool, L. v. (2011b). Real Time Head Pose Estimation from Consumer Depth Cameras. In *Symposium of the German Association*

for Pattern Recognition. (page 4, 5, 7, 22, 72, 95, 96, 99, 100, 101, 102, 103, 104, 105, 106, 107, 140)

[51] Fanello, S., Keskin, C., Kohli, P., Izadi, Shahram Shotton, J., Criminisi, A., Pattacini, U., and Paek, T. (2014). Filter Forests for Learning Data-Dependent Convolutional Kernels. In *Proc. Conference on Computer Vision and Pattern Recognition.* (page 3, 5, 6, 22, 29, 116, 117, 119)

[52] Fattal, R. (2007). Upsampling via Imposed Edges Statistics. *ACM Transactions on Graphics*, 26(3):95. (page 109)

[53] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2010). Object Detection with Discriminatively Trained Part Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645. (page 63, 64, 76, 80, 82, 86, 88, 89, 90)

[54] Fisher, R. A. (1936). The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, 7(2):179–188. (page 12)

[55] Frank, A. and Asuncion, A. (2010). UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences. Available online at http://archive.ics.uci.edu/ml. (page 44)

[56] Freeman, W. T., Jones, T. R., and Pasztor, E. C. (2002). Example-Based Super-Resolution. *Computer Graphics and Applications*, 22(2):56–65. (page 109)

[57] Freund, Y. and Mason, L. (1999). The Alternating Decision Tree Learning Algorithm. In *International Conference on Machine Learning.* (page 43)

[58] Freund, Y. and Shapire, R. E. (1996). Experiments with a New Boosting Algorithm. In *International Conference on Machine Learning.* (page 18, 20, 21, 40)

[59] Freund, Y. and Shapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139. (page 5, 18, 21, 40, 43, 98)

[60] Frey, B. J. and Dueck, D. (2007). Clustering by Passing Messages Between Data Points. *Science*, 315(5814):972–976. (page 14)

[61] Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29(5):1189–1232. (page 20)

[62] Gall, J. and Lempitsky, V. (2009). Class-Specific Hough Forests for Object Detection. In *Proc. Conference on Computer Vision and Pattern Recognition.* (page 3, 4, 7, 22, 29, 63, 64, 65, 66, 67, 68, 69, 70, 71, 74, 78, 80, 82, 84, 86, 88, 89, 90, 93, 96, 99, 100, 101, 102, 117, 140)

[63] Gall, J., Razavi, N., and Van Gool, L. (2010). On-line Adaption of Class-specific Codebooks for Instance Tracking. In *Proc. British Machine Vision Conference*. (page 71)

[64] Gall, J., Yao, A., Razavi, N., Gool, L. v., and Lempitsky, V. (2011). Hough Forests for Object Detection, Tracking, and Action Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(11):2188–2202. (page 69)

[65] Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63(1):3–42. (page 34, 123)

[66] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 63, 76)

[67] Girshick, R., Shotton, J., Kohli, P., Criminisi, A., and Fitzgibbon, A. (2011). Efficient Regression of General-Activity Human Poses from Depth Images. In *Proc. International Conference on Computer Vision*. (page 22, 68, 72, 77)

[68] Glasner, Daniel, Bagon, Shai and Irani, Michal (2009). Super-Resolution From a Single Image. In *Proc. International Conference on Computer Vision*. (page 109, 111, 113, 119)

[69] Glocker, B., Pauly, O., Konukoglu, E., and Criminisi, A. (2012). Joint Classification-Regression Forests for Spatially Structured Multi-Object Segmentation. In *Proc. European Conference on Computer Vision*. (page 80)

[70] Godec, M., Roth, P. M., and Bischof, H. (2011). Hough-based Tracking of Non-rigid Objects. In *Proc. International Conference on Computer Vision*. (page 22, 72)

[71] Grabner, H. and Bischof, H. (2006). On-line Boosting and Vision. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 16, 20)

[72] Grabner, H., Leistner, C., and Bischof, H. (2008). Semi-Supervised On-line Boosting for Robust Tracking. In *Proc. European Conference on Computer Vision*. (page 15, 20)

[73] Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *Alvey Vision Conference*. (page 66)

[74] Hartmann, W., Havlena, M., and Schindler, K. (2014). Predicting Matchability. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 3)

[75] Hastie, T., Tibshirani, R., and Friedman, J. H. (2009). *The Elements of Statistical Learning*. Springer. (page 11, 12, 17, 20, 21, 38, 39, 41, 43)

[76] He, L., Qi, H., and Zaretzki, R. (2013). Beta Process Joint Dictionary Learning for Coupled Feature Spaces with Application to Single Image Super-Resolution. In *Proc.*

Conference on Computer Vision and Pattern Recognition. (page 110, 120, 121, 123, 130, 131, 132, 133)

[77] Hinton, G. E. and Zemel, R. S. (1994). Autoencoders, Minimum Description Length and Helmholtz Free Energy. In *Advances Conference on Neural Information Processing Systems.* (page 14)

[78] Ho, T. K. (1995). Random Decision Forests. In *International Conference on Document Analysis and Recognition.* (page 3, 14, 22)

[79] Ho, T. K. (1998). The Random Subspace Method for Constructing Decision Forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844. (page 3, 22)

[80] Hotelling, H. (1933). Analysis of a Complex of Statistical Variables into Principal Components. *Journal of Educational Psychology*, 24(7):498–520. (page 14)

[81] Huang, J.-B., Singh, A., and Ahuja, N. (2015). Single Image Super-Resolution using Transformed Self-Exemplars. In *Proc. Conference on Computer Vision and Pattern Recognition.* (page 111)

[82] Hull, J. J. (1994). A Database for Handwritten Text Recognition Research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554. (page 44)

[83] Jancsary, J., Nowozin, S., and Rother, C. (2012a). Loss-Specific Training of Non-Parametric Image Restoration Models: A New State of the Art. In *Proc. European Conference on Computer Vision.* (page 43, 142)

[84] Jancsary, J., Nowozin, S., Sharp, T., and Rother, C. (2012b). Regression Tree Fields. In *Proc. Conference on Computer Vision and Pattern Recognition.* (page 3, 142)

[85] Jones, M. and Viola, P. (2003). Fast multi-view face detection. Technical Report 96, Mitsubishi Electric Research Laboratories. (page 97)

[86] Karl, P. (1901). On Lines and Planes of Closest Fit to Systems of Points in Space. *Philosophical Magazine*, 2(6):559–572. (page 14)

[87] Kohonen, T. (1982). Self-Organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics*, 43(1):59–69. (page 14)

[88] Kontschieder, P., Kohli, P., Shotton, J., and Criminisi, A. (2013). GeoF: Geodesic Forests for Learning Coupled Predictors. In *Proc. Conference on Computer Vision and Pattern Recognition.* (page 5, 22)

[89] Kontschieder, P., Rota Bulò, S., Bischof, H., and Pelillo, M. (2011). Structured Class-Labels in Random Forests for Semantic Image Labelling. In *Proc. International Conference on Computer Vision.* (page 4, 5, 6, 22, 30, 80)

[90] Kontschieder, P., Rota Bulò, S., Criminisi, A., Kohli, P., Pelillo, M., and Bischof, H. (2012). Context-Sensitive Decision Forests for Object Detection. In *Advances Conference on Neural Information Processing Systems*. (page 27, 43)

[91] Köstinger, M., Wohlhart, P., Roth, P. M., and Bischof, H. (2011). Annotated Facial Landmarks in the Wild: A Large-scale, Real-world Database for Facial Landmark Localization. In *First IEEE International Workshop on Benchmarking Facial Image Analysis Technologies*. (page 95)

[92] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances Conference on Neural Information Processing Systems*. (page 4, 63, 142)

[93] Ladicky, L., Sturgess, P., Alahari, K., Russell, C., and Torr, P. H. (2010). What,Where & How Many? Combining Object Detectors and CRFs. In *Proc. European Conference on Computer Vision*. (page 63)

[94] Lampert, C. H., Blaschko, M. B., and Hofmann, T. (2008). Beyond Sliding Windows: Object Localization by Efficient Subwindow Search. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 82)

[95] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551. (page 12)

[96] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proc. of IEEE*, 86(11):2278–2324. (page)

[97] LeCun, Y. and Yoshua, B. (1995). *Convolutional Networks for Images, Speech, and Time Series*. The Handbook of Brain Theory and Neural Networks, MIT Press. (page 12)

[98] Lehmann, A., Leibe, B., and van Gool, L. (2011). Fast PRISM: Branch and Bound Hough Transfrom for Object Class Detection. *International Journal of Computer Vision*, 94(2):175–197. (page 71)

[99] Leibe, B., Cornelis, N., Cornelis, K., and Van Gool, L. (2007). Dynamic 3D Scene Analysis from a Moving Vehicle. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 73, 87)

[100] Leibe, B., Leonardis, A., and Schiele, B. (2004). Combined Object Categorization and Segmentation with an Implicit Shape Model. In *ECCV Workshops*. (page 63, 64, 66, 93)

[101] Leistner, C., Godec, M., Saffari, A., and Bischof, H. (2010a). On-line Multi-View Forests for Tracking. In *Symposium of the German Association for Pattern Recognition*. (page 16)

[102] Leistner, C., Grabner, H., and Bischof, H. (2008). Semi-Supervised Boosting using Visual Similarity Learning. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 15, 20)

[103] Leistner, C., Saffari, A., and Bischof, H. (2009). Semi-Supervised Random Forests. In *Proc. International Conference on Computer Vision*. (page 15, 20)

[104] Leistner, C., Saffari, A., and Bischof, H. (2010b). MIForests: Multiple-Instance Learning with Randomized Trees. In *Proc. European Conference on Computer Vision*. (page 15)

[105] Lloyd, S. P. (1982). Least Squares Quantization in PCM. *Transactions on Information Theory*, 28(2):129–137. (page 14)

[106] Lu, C., Shi, J., and Jia, J. (2013). Abnormal Event Detection at 150 FPS in MATLAB. In *Proc. International Conference on Computer Vision*. (page 111, 112)

[107] Maji, S. and Malik, J. (2009). Object detection using a max-margin hough transform. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 71)

[108] Malassiotis, S. and Strintzis, M. G. (2005). Robust real-time 3D head pose estimation from range data. *Pattern Recognition*, 38(8):1153–1165. (page 99)

[109] Mallapragada, P. K., Jin, R., Jain, A. K., and Liu, Y. (2008). SemiBoost: Boosting for Semi-supervised Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(11):2000–2014. (page 15, 20)

[110] Marín, J., Vázquez, D., López, A. M., Amores, J., and Leibe, B. (2013). Random Forests of Local Experts for Pedestrian Detection. In *Proc. International Conference on Computer Vision*. (page 22, 82)

[111] Maron, O. and Ratan, A. L. (1998). Multiple-Instance Learning for Natural Scene Classification. In *International Conference on Machine Learning*. (page 15)

[112] Masnadi-Shirazi, H., Mahadevan, V., and Vasconcelos, N. (2010). On the design of robust classifiers for computer vision. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 20, 40, 45)

[113] Masnadi-Shirazi, H. and Vasconcelos, N. (2008). On the Design of Loss Functions for Classification: theory, robustness to outliers, and SavageBoost. In *Advances Conference on Neural Information Processing Systems*. (page 20, 21, 40, 45)

[114] Mathias, M., Benenson, R., Pedersoli, M., and Van Gool, L. (2014). Face detection without bells and whistles. In *Proc. European Conference on Computer Vision*. (page 103)

[115] Mitchell, T. (1997). *Machine Learning*. Mcgraw-Hill Higher Education. (page 9, 11, 12)

[116] Mittelman, R., Lee, H., Kuipers, B., and Savarese, S. (2013). Weakly Supervised Learning of Mid-Level Features with Beta-Bernoulli Process Restricted Boltzmann Machines. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 15)

[117] Montillo, A., Shotton, J., Winn, J., Iglesias, J. E., Metaxas, D., and Criminisi, A. (2011). Entangled Decision Forests and their Application for Semantic Segmentation of CT Images. In *Proc. International Conference on Information Processing in Medical Imaging*. (page 43)

[118] Moosmann, F., Triggs, B., and Jurie, F. (2006). Fast Discriminative Visual Codebooks using Randomized Clustering Forests. In *Advances Conference on Neural Information Processing Systems*. (page 142)

[119] Murphy-Chutorian, E. and Trivedi, M. M. (2008). Head Pose Estimation in Computer Vision: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(4):607–626. (page 97)

[120] Nowozin, S. (2012). Improved Information Gain Estimates for Decision Tree Induction. In *International Conference on Machine Learning*. (page 27)

[121] Nowozin, S., Rother, C., Bagon, S., Sharp, T., Yao, B., and Kohli, P. (2011). Decision Tree Fields. In *Proc. International Conference on Computer Vision*. (page 3, 29, 142)

[122] Okada, R. (2009). Discriminative Generalized Hough Transform for Object Dectection. In *Proc. International Conference on Computer Vision*. (page 67)

[123] Olshausen, B. A. and Field, D. J. (1997). Sparse Coding with an Overcomplete Basis Set: A Strategy Employed by V1? *Vision Research*, 37(23):3311–3325. (page 111)

[124] Osadchy, M., Miller, M. L., and LeCun, Y. (2004). Synergistic Face Detection and Pose Estimation with Energy-Based Models. In *Advances Conference on Neural Information Processing Systems*. (page 97)

[125] Ostrovsky, Y., Meyers, E., Ganesh, S., Mathur, U., and Sinha, P. (2009). Visual Parsing After Recovery From Blindness. *Psychological Science*, 20(12):1484–1491. (page 1)

[126] Pan, S. J. and Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359. (page 16)

[127] Pandey, M. and Lazebnik, S. (2011). Scene Recognition and Weakly Supervised Object Localization with Deformable Part-Based Models. In *Proc. International Conference on Computer Vision*. (page 63)

[128] Pishchulin, L., Andriluka, M., Gehler, P., and Schiele, B. (2013). Strong Appearance and Expressive Spatial Models for Human Pose Estimation. In *Proc. International Conference on Computer Vision*. (page 5)

[129] Poier, G., Roditakis, K., Schulter, S., Michel, D., Bischof, H., and Argyros, A. A. (2015). Hybrid One-Shot 3D Hand Pose Estimation by Exploiting Uncertainties. In *Proc. British Machine Vision Conference*. (page 77)

[130] Poier, G., Schulter, S., Sternig, S., Roth, P. M., and Bischof, H. (2014). Hough Forests Revisited: An Approach to Multiple Instance Tracking from Multiple Cameras. In *German Conference on Pattern Recognition*. (page 16)

[131] Prest, A., Leistner, C., Civera, J., Schmid, C., and Ferrari, V. (2012). Learning Object Class Detectors from Weakly Annotated Video. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 15)

[132] Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning*, 1(1):81–106. (page 22)

[133] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann. (page 22)

[134] Rao, C. R. (1973). *Linear Statistical Inference and its Applications*. Wiley. (page 12)

[135] Razavi, N., Gall, J., and Gool, L. v. (2011). Scalable Multi-class Object Detection. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 65, 71)

[136] Razavi, N., Gall, J., and Van Gool, L. (2010). Backprojection Revisited: Scalable Multi-view Object Detection and Similarity Metrics for Detections. In *Proc. European Conference on Computer Vision*. (page 72)

[137] Redondo-Cabrera, C. and Lopez-Sastre, R. (2015). Because better detections are still possible: Multi-aspect object detection with Boosted Hough Forest. In *Proc. British Machine Vision Conference*. (page 76, 142)

[138] Ren, S., Cao, X., Wei, Y., and Sun, J. (2015). Global Refinement of Random Forest. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 129)

[139] Riegler, G., Ferstl, D., Rüther, M., and Bischof, H. (2014). Hough Networks for Head Pose Estimation and Facial Feature Localization. In *Proc. British Machine Vision Conference*. (page 99)

[140] Rosenblatt, F. (1958). The Perceptron : A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Reviews*, 65(6):386–408. (page 12)

[141] Rota Bulò, S. and Kontschieder, P. (2014). Neural Decision Forest for Semantic Image Labelling. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 5, 22, 34, 142)

[142] Rother, C., Kolmogorov, V., and Blake, A. (2004). GrabCut - Interactive Foreground Extraction using Iterated Graph Cuts. *ACM Transactions on Graphics*, 23(3):309–314. (page 72)

[143] Roweis, S. T. and Saul, L. K. (2000). Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, 290(5500):2323–2326. (page 14)

[144] Saffari, A. (2011). *Multi-Class Semi-Supervised and Online Boosting*. PhD thesis, Graz University of Technology. (page 41)

[145] Saffari, A., Leistner, C., Santner, J., Godec, M., and Bischof, H. (2009). On-line Random Forests. In *Proc. ICCV Workshop on On-line Learning for Computer Vision*. (page 16, 72)

[146] Samaria, F. S. and Harter, A. C. (1994). Parameterisation of a Stochastic Model for Human Face Identification. In *IEEE Workshop on Applications of Computer Vision*. (page 45)

[147] Schapire, R. E. (1990). The Strength of Weak Learnability. *Machine Learning*, 5(2):197–227. (page 18, 21, 22, 40)

[148] Schapire, R. E. and Freund, Y. (2012). *Boosting: Foundations and Algorithms*. MIT Press. (page 18, 19, 21)

[149] Schmidt, U. and Roth, S. (2014). Shrinkage Fields for Effective Image Restoration. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 3)

[150] Schmidt, U., Rother, C., Nowozin, S., Jancsary, J., and Roth, S. (2013). Discriminative Non-blind Deblurring. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 3)

[151] Schölkopf, B., Smola, A., and Müller, K.-R. (1998). Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation*, 10(5):1299–1319. (page 14)

[152] Schulter, S., Leistner, C., and Bischof, H. (2015a). Fast and Accurate Image Upscaling with Super-Resolution Forests. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 5, 111, 112)

[153] Schulter, S., Leistner, C., and Bischof, H. (2015b). Supplementary Material: Fast and Accurate Image Upscaling with Super-Resolution Forests. CVPR 2015 Supplementary Material. (page 123)

[154] Schulter, S., Leistner, C., Roth, P. M., and Bischof, H. (2013a). Unsupervised Object Discovery and Segmentation in Videos. In *Proc. British Machine Vision Conference*. (page 15)

[155] Schulter, S., Leistner, C., Roth, P. M., Van Gool, L., and Bischof, H. (2011). On-line Hough Forests. In *Proc. British Machine Vision Conference*. (page 16, 22, 30, 71, 125)

[156] Schulter, S., Leistner, C., Wohlhart, P., Roth, P. M., and Bischof, H. (2013b). Alternating Regression Forests for Object Detection and Pose Estimation. In *Proc. International Conference on Computer Vision*. (page 5, 7, 33, 35, 68, 74, 75, 78, 96)

[157] Schulter, S., Leistner, C., Wohlhart, P., Roth, P. M., and Bischof, H. (2014). Accurate Object Detection with Joint Classification-Regression Random Forests. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 7, 22, 76, 77, 80)

[158] Schulter, S., Wohlhart, P., Leistner, C., Saffari, A., Roth, P. M., and Bischof, H. (2013c). Alternating Decision Forests. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 5, 7, 33, 35, 73, 74)

[159] Seemann, E., Nickel, K., and Stiefelhagen, R. (2004). Head Pose Estimation Using Stereo Vision For Human-Robot Interaction. In *International Conference on Automatic Face & Gesture Recognition*. (page 98)

[160] Shalev-Schwartz, S., Singer, Y., and Srebro, N. (2007). Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *International Conference on Machine Learning*. (page 16)

[161] Shen, C., Lin, G., and van den Hengel, A. (2014). StructBoost: Boosting Methods For Predicting Structured Output Variables. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(10):2089–2103. (page 20)

[162] Shi, J. and Malik, J. (2000). Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905. (page 14)

[163] Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. (2011). Real-Time Human Pose Recognition in Parts from a Single Depth Image. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 3, 4, 5, 22, 63, 72, 139)

[164] Shotton, J., Johnson, M., and Cipolla, R. (2008). Semantic Texton Forests for Image Categorization and Segmentation. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 5, 22, 142)

[165] Sinha, P. (2004). Project Prakash: Face Classification Following Extended Visual Deprivation. In *International Conference on Development and Learning*. (page 1)

[166] Siva, P., Russell, C., and Xiang, T. (2012). In Defence of Negative Mining for Annotating Weakly Labelled Data. In *Proc. European Conference on Computer Vision*. (page 15)

[167] Siva, P. and Xiang, T. (2011). Weakly Supervised Action Detection. In *Proc. British Machine Vision Conference*. (page 15)

[168] Sivic, J. and Zisserman, A. (2003). Video Google: A Text Retrieval Approach to Object Matching in Videos. In *Proc. International Conference on Computer Vision*. (page 66)

[169] Su, J., Shirab, J. S., and Matwin, S. (2011). Large Scale Text Classification using Semi-supervised Multinomial Naive Bayes. In *International Conference on Machine Learning*. (page 15)

[170] Sun, L. and Hays, J. (2012). Super-resolution from Internet-scale Scene Matching. In *Proc. International Conference on Computational Photography*. (page 111)

[171] Surowiecki, J. (2004). *The Wisdom of Crowds: Why the Many Are Smarter Than the Few and How Collective Wisdom Shapes Business, Economies, Societies and Nations*. Doubleday. (page 17)

[172] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going Deeper with Convolutions. *CoRR*, abs/1409.4842. (page 3, 4)

[173] Tang, D., Chang, H. J., Tejani, A., and Kim, T.-K. (2014). Latent Regression Forest: Structured Estimation of 3D Articulated Hand Posture. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 5, 6, 142)

[174] Tang, D., Liu, Y., and Kim, T.-K. (2012). Fast Pedestrian Detection by Cascaded Random Forest with Dominant Orientation Templates. In *Proc. British Machine Vision Conference*. (page 82)

[175] Tang, D., Yu, T.-H., and Kim, T.-K. (2013). Real-time Articulated Hand Pose Estimation using Semi-supervised Transductive Regression Forests. In *Proc. International Conference on Computer Vision*. (page 5, 72, 77, 142)

[176] Tenenbaum, J. B., de Silva, V., and Langford, J. C. (2000). A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323. (page 14)

[177] Timofte, R., De Smet, V., , and Van Gool, L. (2014). A+: Adjusted Anchored Neighborhood Regression for Fast Super-Resolution. In *Proc. Asian Conference on Computer Vision*. (page 3, 111, 112, 114, 115, 121, 123, 126, 129, 130, 131, 132, 133, 134, 135, 136, 137)

[178] Timofte, R., De Smet, V., and Van Gool, L. (2013). Anchored Neighborhood Regression for Fast Example-Based Super-Resolution. In *Proc. International Conference on Computer Vision*. (page 109, 111, 112, 113, 114, 115, 119, 120, 121, 122, 123, 124, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137)

[179] Torralba, A. and Efros, A. (2011). An Unbiased Look at Dataset Bias. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 16)

[180] van de Sande, K. E. A., Uijlings, J. R. R., Gevers, T., and Smeulders, A. W. M. (2011). Segmentation as selective search for object recognition. In *Proc. International Conference on Computer Vision*. (page 63, 76)

[181] van Ouwerkerk, J. (2006). Image super-resolution survey. *Image and Vision Computing*, 24(10):1039–1052. (page 111)

[182] Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer. (page 11, 12)

[183] Vatahska, T., Bennewitz, M., and Behnke, S. (2007). Feature-based Head Pose Estimation from Images. In *International Conference on Humanoid Robots*. (page 98)

[184] Vineet, V., Warrell, J., Ladicky, L., and Torr, P. H. S. (2011). Human Instance Segmentation from Video using Detector-based Conditional Random Fields. In *Proc. British Machine Vision Conference*. (page 63)

[185] Viola, P. and Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 3, 4, 20, 80, 97)

[186] Viola, P. and Jones, M. (2004). Robust Real-time Object Detection. *International Journal of Computer Vision*, 57(2):137–154. (page 63)

[187] Viola, P., Platt, J. C., and Zhang, C. (2006). Multiple Instance Boosting for Object Detection. In *Advances Conference on Neural Information Processing Systems*. (page 15, 20)

[188] Vitter, J. S. (1985). Random Sampling with a Reservoir. *Transactions on Mathematical Software*, 11(1):37–57. (page 125)

[189] Wan, L., Zeiler, M., Zhang, S., LeCun, Y., and Fergus, R. (2013). Regularization of Neural Network using DropConnect. In *International Conference on Machine Learning*. (page 13)

[190] Wang, S., Zhang, L., Liang, Y., and Pan, Q. (2012). Semi-Coupled Dictionary Learning with Applications to Image Super-Resolution and Photo-Sketch Synthesis. In *Proc. Conference on Computer Vision and Pattern Recognition.* (page 109, 111)

[191] Wang, X., Yang, M., Zhu, S., and Lin, Y. (2013). Regionlets for Generic Object Detection. In *Proc. International Conference on Computer Vision.* (page 76)

[192] Weise, T., Leibe, B., and van Gool, L. (2007). Fast 3D Scanning with Automatic Motion Compensation. In *Proc. Conference on Computer Vision and Pattern Recognition.* (page 98)

[193] Welch, B. L. (1947). The generalization of "Student's" problem when several different population variances are involved. *Biometrika*, 34(2):28–35. (page 44)

[194] Whitehill, J. and Movellan, J. R. (2008). A Discriminative Approach to Frame-by-Frame Head Pose Tracking. In *International Conference on Automatic Face & Gesture Recognition.* (page 98)

[195] Wohlhart, P. (2014). *Object Detection Based on Local Evidence.* PhD thesis, Graz University of Technology. (page 22, 64, 73)

[196] Wohlhart, P., Donoser, M., Roth, P. M., and Bischof, H. (2012a). Detecting Partially Occluded Objects with an Implicit Shape Model Random Field. In *Proc. Asian Conference on Computer Vision.* (page 82)

[197] Wohlhart, P., Schulter, S., Köstinger, M., Roth, P. M., and Bischof, H. (2012b). Discriminative hough Forests for object Detection. In *Proc. British Machine Vision Conference.* (page 22, 71)

[198] Xu, L., Neufeld, J., Larson, B., and Schuurmans, D. (2004). Maximum Margin Clustering. In *Advances Conference on Neural Information Processing Systems.* (page 14)

[199] Yang, C.-Y., Ma, C., and Yang, M.-H. (2014). Single-Image Super-Resolution: A Benchmark. In *Proc. European Conference on Computer Vision.* (page 122)

[200] Yang, C.-Y. and Yang, M.-H. (2013). Fast Direct Super-Resolution by Simple Functions. In *Proc. International Conference on Computer Vision.* (page 109, 112, 115)

[201] Yang, J., Wright, J., Huang, T., and Ma, Y. (2010). Image Super-Resolution Via Sparse Representation. *Transactions on Image Processing*, 19(11):2861–2873. (page 109, 111, 112, 114, 119)

[202] Yao, B., Aditya, K., and Fei-Fei, L. (2011). Combining Randomization and Discrimination for Fine-Grained Image Categorization. In *Proc. Conference on Computer Vision and Pattern Recognition.* (page 34)

[203] Zeisl, B., Leistner, C., Saffari, A., and Bischof, H. (2010). Online Semi-Supervised Multiple-Instance Boosting. In *Proc. Conference on Computer Vision and Pattern Recognition*. (page 20)

[204] Zeyde, R., Elad, M., and Protter, M. (2010). On Single Image Scale-Up using Sparse-Representations. In *Curves and Surfaces*. (page 3, 109, 111, 112, 113, 114, 119, 120, 121, 123, 124, 126, 129, 130, 131, 132, 133, 134, 135, 136, 137)

[205] Zhu, X. and Goldberg, A. B. (2009). *Introduction to Semi-Supervised Learning*. Morgan & Claypool. (page 15)

[206] Zitnick, L. C. and Dollár, P. (2014). Edge Boxes: Locating Object Proposals from Edges. In *Proc. European Conference on Computer Vision*. (page 63)

[207] Zou, H., Zhu, J., and Hastie, T. (2008). New multicategory boosting algorithms based on multicategory Fisher-consistent losses. *Annals of Applied Statistics*, 2(4):1290–1306. (page 31)