Graz University of Technology

Institute for Computer Graphics and Vision

Master's Thesis

A Transformation Invariant Learning Approach for Detection and Tracking using Weakly-related Videos

Samuel Schulter

Graz, Austria, Jan 2011

Thesis supervisor

Univ. Prof. DI Dr. Horst Bischof

Instructor

DI Dr. Christian Leistner

Abstract

For current computer vision systems, object detection and tracking are challenging tasks, whereas humans perform very well on both of them. This comes from the fact that these systems have to cope with all variations and transformations that occur in natural scenes, such as shape, appearance, different illuminations, and occlusions. In general, machine learning algorithms learn hypotheses based on labeled training data to separate correctly unseen test data according to their class, *i.e.*, positive or negative. In computer vision, this principle also holds for detection algorithms as well as for tracking approaches that are based on classifiers. Indeed, the amount of labeled training data available is often too small to capture all possible transformations and intra-class variations of the target object, what makes generalization a hard task. In contrast, unlabeled data exist in large amounts and are typically easy to collect. But the extraction of useful information from unlabeled data is difficult in practice, as they often stem from different distributions than the labeled data, *i.e.*, they are only weakly-related towards the target class.

In contrast to a heterogenous collection of single images, video sequences, as a source of unlabeled data, comprise an underlying structure given by real-world constraints, which is the space-time coherence of naturally moving objects. It is extremely unlikely that moving objects, which are captured prominently in a video, vanish from one frame to another. On the other hand, they are likely to show smooth local transformations from frame to frame. Exploiting this fact, one can extract these local transformations of moving objects in natural scenes and use them to improve a classifier. Furthermore, local transformations can often be shared over different target-classes, which allows to include only weakly-related data in the learning process.

The main intent of this Master's Thesis is to present a transformation invariant learning approach using unlabeled weakly-related videos and its embedding into a real-world detection and tracking system including data acquisition. Based on a Random Forest framework, we define an optimization problem, which also involves data containing local transformations of naturally moving objects extracted from video sequences. We gathered real-world videos from the web and applied a dense optical flow in order to extract useful motion information from video data. The evaluation of our methods showed that we can improve the generalization performance in object detection and tracking.

Keywords. Object detection, object tracking, video, Random Forest, cognitive science, invariant object representations, optical flow.

Kurzfassung

Objektdetektion und -verfolgung sind für aktuelle Computer Vision Systeme anspruchsvolle Aufgaben, für Menschen aber, sind sie einfach zu verrichten. Das hat den Grund, dass diese System mit allen möglichen Variationen und Transformationen zurecht kommen müssen, die in natürlichen Szenen auftreten, wie z.B. Forms-, Erscheinungs-, Beleuchtungsänderungen oder teilweiser Bedeckungen. Im Allgemeinen lernen Machine Learning Algorithmen Hypothesen, basierend auf annotierten Trainingsdaten, um ungesehene Daten korrekt bezüglich ihrer Klassen, also positiv oder negativ, zu trennen. In der Computer Vision gilt dieses Prinzip sowohl für Detektions- als auch für Verfolgungsalgorithmen, die auf Klassifizierern basieren. Jedoch ist die Anzahl der verfügbaren, annotierten Trainingsdaten oft zu klein um alle möglichen Objekttransformationen und intra-class Variationen abzudecken, was die Generalisierung zu einer schweren Aufgabe macht. Im Gegensatz dazu existieren nicht annotierte Daten in großen Mengen und sind typischerweise auch einfach zu sammeln. Aber der Gewinnn von sinnvollen Informationen aus nicht annotierten Daten ist in der Praxis schwierig, da sie oft von einer unterschiedlichen Verteilung stammen, als jene der annotierten Daten, d.h. sie hängen nur schwach mit der Zielklasse zusammen.

Im Gegensatz zu einer heterogenen Sammlung von Einzelbildern besitzen Videosequenzen, als Quelle von nicht annotierten Daten, eine von Natur gegebene, zugrundeliegende Struktur, und zwar den Raum-Zeit Zusammenhang von sich natürlich bewegten Objekten. Es ist höchst unwahrscheinlich, dass sich bewegte Objekte, die auffallend im Video gezeigt werden, von einem Bild auf das nächste verschwinden. Andererseits ist es wahrscheinlich, dass die Objekte leichten, lokalen Transformationen unterzogen wurden. Nützt man dieses Wissen aus, kann man diese lokalen Transformation von bewegten Objekten extrahieren und weiterverwenden um einen Klassifizierer zu verbessern. Außerdem können lokale Transformationen oft unter verschiedenen Zielklassen geteilt werden, was es erlaubt, Daten in den Lernprozess miteinzubeziehen, die nur schwachen Zusammenhang zur Zielklasse zeigen.

Der Hauptzweck dieser Masterarbeit ist es, nicht annotierte Videos, die nur schwach mit der Zieklasse verwandt sind, in einen transformationsinvarianten Lernansatz zu integrieren, und zu zeigen, wie dieser in einem vollständigen Detektions- und Verfolgungssystem verwendet werden kann. Basierend auf einem Random Forest - Gerüst, definieren wir ein Optimierungsproblem, welches zusätzlich Daten miteinbezieht, die lokale Transformationen von sich natürlich bewegten Objekten beinhaltet, und die aus Videosequenzen extrahiert wurden. Wir sammelten natürliche Videosequenzen aus dem Internet und errechneten einen dichten optischen Fluss, um sinnvolle Informationen aus den Videodaten zu extrahieren. Die Evaluierung unserer Methoden zeigte, dass wir die Generalisierungsleistung von Objektdetektion und -verfolgung verbessern können.

Schlüsselwörter. Objektdetektion, Objektverfolgung, Video, Random Forest, Kognitionswissenschaft, invariante Objektrepräsentation, optischer Fluss

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

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.

Graz, am …………………………… …………………………………………………..
(Unterschrift)

Englische Fassung:

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.

…………………………… …………………………………………………..
date (signature)

Contents

List of Figures v

1 Introduction **1**

1.1 Motivation 1

1.2 Video 5

1.3 Learning from Video 5

1.4 Human perception 8

1.5 Organisation of the Master's thesis 9

2 Related Work **11**

2.1 Short introduction to Machine Learning 12

2.1.1 Feature space 12

2.1.2 Unsupervised Learning 13

2.1.3 Supervised Learning 15

2.2 Semi-Supervised Learning 15

2.3 Transfer Learning 18

2.4 Learning from weakly-related data 21

2.5 Learning from video data 22

2.5.1 Human perception 24

2.6 Part-based approaches 25

2.7 Chapter Summary 26

3 Hough Forests with Motion Data **27**

3.1 Random Forests 28

3.2 Learning from motion data 30
3.3 Extending Hough Forests to learn from motion data 32
3.3.1 Hough Forests 33
3.3.2 Hough Forest Regularization 36
3.3.3 Training a Codebook 37
3.4 Chapter summary 39

4 Learning Framework and Motion Extraction 41
4.1 Processing pipeline 41
4.2 Extraction of motion data 42
4.2.1 Optical Flow 44
4.2.2 Motion-based Segmentation 46
4.3 Chapter Summary 49

5 Experiments 51
5.1 Video database 51
5.2 Object Detection 53
5.2.1 Datasets 54
5.2.2 Performance improvement 54
5.2.3 Influence of parameters 57
5.2.4 Influence of motion data 58
5.2.5 Discussion 62
5.3 Tracking 63
5.3.1 Class-specific tracking 64
5.3.2 Tracking of arbitrary objects 66
5.3.3 Discussion 67
5.4 Chapter Summary 68

6 Conclusion and Outlook 69
6.1 Conclusion 69
6.2 Outlook 70

Bibliography **73**

List of Figures

1.1 This figure gives a schematic overview of a common approach to object detection. It can be separated in two different phases, "Train" and "Test", where the first one learns a hypothesis based on image representations of given labeled images. In the "Test" phase, the hypothesis predicts target class instances in test images. 2

1.2 Some result images from a pedestrian detection system. (a) and (b) show two successful examples, *i.e.*, true positives. In (c), a missed detection (false negative) is illustrated and (d) shows a wrong prediction (false positive). 3

1.3 Some frames of a good working tracking algorithm which follows a soft toy. The image content is taken from [3]. 4

1.4 Some frames of a failing tracking algorithm. The image content is taken from [67]. 4

1.5 Some frames of a video sequence capturing a koala bear moving its body. The different parts of the body undergo several local transformations. . 6

1.6 It is very likely that object parts that look similar, i.e., have similar appearance, can undergo similar transformations. This knowledge can then be transferred between unknown categories. The figures (taken from [28]) show weakly-related objects, which all share a common part, the wheels. 7

1.7 A schematic illustration of the object detection and tracking procedures presented in this thesis. Both algorithms incorporate local transformations extracted from video data. 8

2.1 This figure shows an example of good and bad features. Blue and green points correspond to different distributions and ought to be separated easily, which is the case in (a), but not in (b). 13

2.2 This figure describes a typical clustering task. In (a), one can see the given dataset **X**, *i.e.*, some point clouds in the feature space. In (b), each of these feature vectors is assigned a cluster, indicated with the colors red, green and black. 14

2.3 This figure illustrates the supervised learning task. In (a), the training data is presented (blue points from class 1 and green points from class 2) and the trained classifier f. (b) shows the evaluation of f on the test dataset. 16

2.4 This figure illustrates a simple SSL scenario. (a) and (b) show the supervised case of learning a hypothesis from the given labeled dataset. In (c), the dataset is extended with unlabeled data points and in (d), the newly trained hypothesis is illustrated. 17

2.5 This figure from [59] illustrates the basic principle of transfer learning. In (a), one sees the traditional machine learning approach. Each task has its own learning system. (b) shows the approach of transfer learning, where knowledge is shared over different tasks. 19

2.6 This figure gives a detailed overview of the different categories of the transfer learning problem [59]. 20

3.1 In (a), one can see some training images used in the Hough Forest Framework. The red box is the bounding box of the target object, in this case a pedestrian from the TUD-pedestrian dataset [2]. Within the bounding box, small patches are extracted at random locations, which can be seen from the blue squares. All patches from positive training examples store an offset vector to the object centroid (green line). (b) shows the extracted 16x16 patches. 34

3.2 A 3D-scale-space of Hough images and its corresponding test images are shown in (a). An overview of the training procedure from [58] is illustrated in (b). 35

4.1 This figure gives a rough overview of the processing pipelines for both, detection (a) and tracking (b) tasks. 43

4.2 Two different video sequences, (a) and (b), where one can see a patch (blue square) describing the same local area of a moving object over all frames. 44

4.3 A schematic overview of the process of extracting pair-wise motion data out of video sequences using optical flow and a motion-based segmentation technique. 45

4.4 Here one can see some figures from the calculation of the optical flow. (a) and (b) show two consecutive frames from a video sequence including the border used for estimating the background motion. The optical flow calculated with the Huber-L^1 method [84] is illustrated in (c). Furthermore, (d) shows the final background subtracted optical flow. 47

4.5 Motion-based segmentation pipeline: (a) shows a background motion subtracted optical flow field, which serves as input for the segmentation pipeline. The mean subtracted flow is illustrated in (b). (c) shows the binary images after the segmentation using the Potts-model and (d) illustrates the original input image (first frame) with the corresponding bounding box around the moving object. 48

4.6 This figure illustrates the extraction of patches from two successive frames. (a) shows the binary image after segmentation. Within the white area, which describes the moving object, the patches can be positioned. (b) and (c) show two successive frames with the corresponding extracted patches. The little green squares denote the patches. 49

5.1 Here one can see several frames from the "moving-objects"-database in 3 different sequences, which contain a lion (a), a panda bear (b), and a train (c). 52

5.2 Figures (a), (b), and (c) illustrate some train and test images from the 3 datasets used for the object detection experiments. Training images are depicted on the left side, testing images on the right. In (b) one can see the reduced testing images from the TUD-pedestrian dataset. 55

5.3 PR curves for the three different datasets. The parameters used in the experiments can be seen in the figures. 56

5.4 In this figure, the influence of the parameter t^{Δ} is illustrated. (a), (b), and (c) show the same three pairs, but with different values for the parameter. That is, the first column is always the same, but the second one illustrates the different levels of transformation. 58

5.5 Results on the ETHZ-cars dataset for different values of λ in (a) and # pairs in (b). The influence of the parameter t^{Δ} was evaluated on the Weizmann-horses dataset (c). 59

5.6 Here one can see two examples of simulated motion by using virtually warped versions of the original image. 60

5.7 In (a), we show how different “degrees of weak-relation” of the motion data influences the performance . The results from the “virtual-samples” experiment are shown in (b). The motion data still supports the classifier, although the labeled data already includes some simulated local transformations. Figure (c) shows the results when extending the labeled dataset with data from the video sequences. 61

5.8 Some frames of the sequences used for the class-specific tracking experiment. In both sequences, one has to track a pedestrian. 65

5.9 Again, we depict some frames of the used tracking sequences. In (a), a face has to be tracked, and in (b), a bottle of liquor. 66

Chapter 1

Introduction

Contents

1.1 Motivation **1**
1.2 Video **5**
1.3 Learning from Video **5**
1.4 Human perception **8**
1.5 Organisation of the Master's thesis **9**

1.1 Motivation

Computer vision is a strong and fast growing research area, which achieved loads of impressive results and attracted wide interest in the last couple of years. But the performance in several different tasks still lacks behind that of the human visual system. Image segmentation, for instance, is an easy task for human beings, but is very demanding for a computer vision system. Many issues like illumination, pose, occlusion, and so on have to be considered to achieve good results. While computer vision offers a rich range of different tasks, in this thesis, we will focus on object detection and tracking.

The goal in object detection is to predict the correct locations of target objects in unseen static images. The first step in a common approach to object detection is to collect a set of labeled, static training images from the target class, *e.g.*, pedestrians or cars, and a set of random negative images or background images. In a second step, one tries to find a good image representation, *i.e.*, extracting informative features from training images. Third, a learning algorithm is provided with these features to find a good hypothesis, separating target object features from background features. The hypothesis is then evaluated on the feature vectors extracted from unseen test images. Figure 1.1 illustrates this scenario.

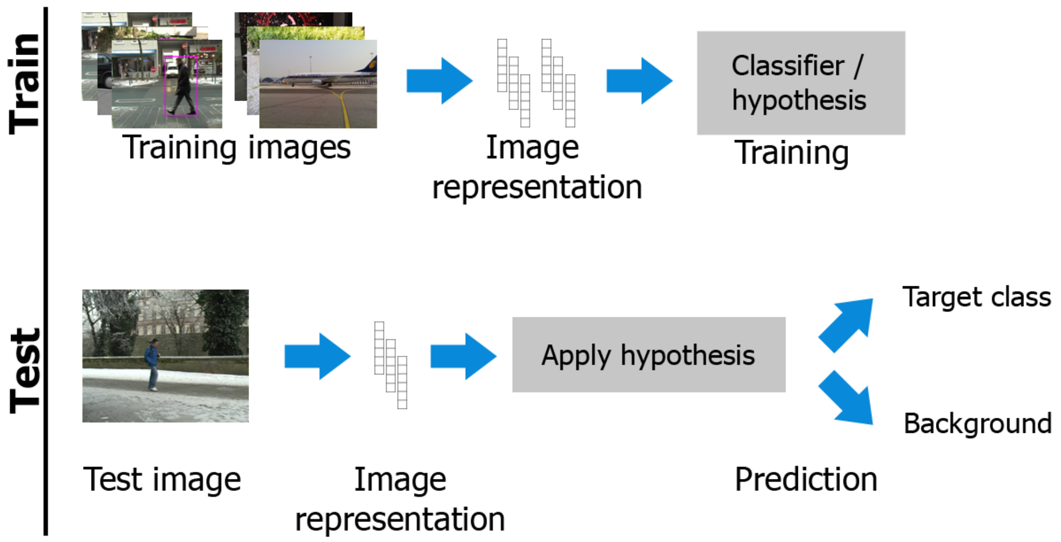
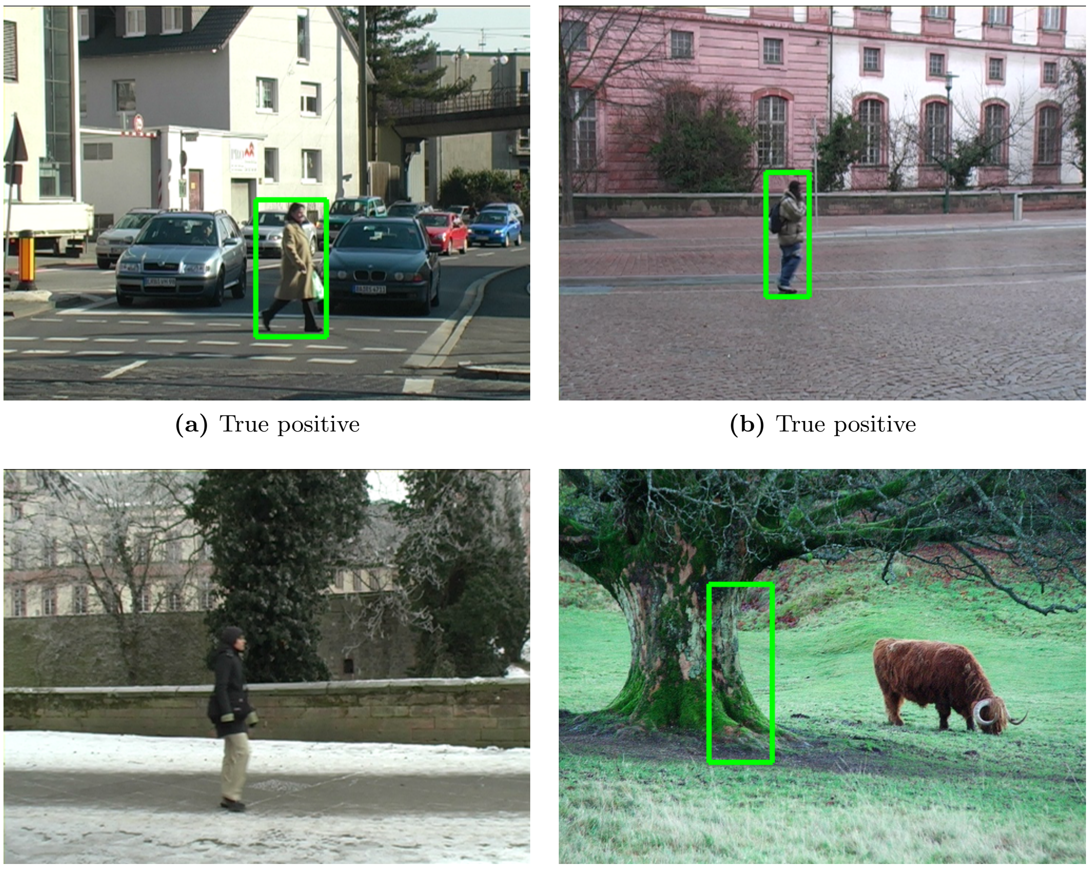


Figure 1.1: This figure gives a schematic overview of a common approach to object detection. It can be separated in two different phases, "Train" and "Test", where the first one learns a hypothesis based on image representations of given labeled images. In the "Test" phase, the hypothesis predicts target class instances in test images.

Prominent examples for such feature extraction methods are Histogram of Oriented Gradients (HOGs) [24], Haar-Features [80], or Local Binary Patterns (LBPs) [1], which all showed to perform well in object detection tasks. Boosting [32] or support vector machines (SVM) [79] are good examples for well-known learning algorithms in computer vision. They have been applied to many different tasks, like text-categorization [40–42, 69], speaker identification, face detection or handwriting recognition [20]. A current state-of-the-art object detection system, which is based on SVMs and HOGs, is described in [29].

Figure 1.2 shows result images from a pedestrian detection system, where one clearly sees that the system can correctly predict the location and size of a target-class object in some images but totally fails on other ones, *i.e.*, miss some pedestrians (Figure 1.2c) or predict some at wrong locations (Figure 1.2d).


(a) True positive

(b) True positive

(c) False negative

(d) False positive

Figure 1.2: Some result images from a pedestrian detection system. (a) and (b) show two successful examples, *i.e.*, true positives. In (c), a missed detection (false negative) is illustrated and (d) shows a wrong prediction (false positive).

One can also look up the current detection results (competition "comp3") of the tough VOC challenge [*]. The winning method for the category "person" achieves an average precision (AP) value of 47.5%, meaning that not even half of the objects in the testing images are detected correctly. Other categories show better results but many are even worse. The best winning AP value found over all categories is 58.4% and the worst is 13.0%. Although the VOC challenge is a very demanding object detection task, human beings ought to achieve AP values above at least 90.0% in each of the categories. Thus, there is enough space for improvement in object detection.

[*] http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2010/results/index.html

As mentioned above, this thesis will also focus on the task of object tracking. Contrary to the class-specific object detection task described in the previous paragraphs, here we are confronted with an instance-specific problem. The goal in object tracking is to find and track a specific instance of a class in a video sequence. Many different approaches to object tracking have been proposed, just to mention a few, [18] describes a mean shift based approach, [3] shows how to improve tracking using multiple-instance learning (MIL) in combination with boosting, and in [66] a Random Forest (RF) based method is presented. A good review of state-of-the-art tracking algorithms is given by Yilmaz *et al.* in [92]. Modern tracking algorithms often use the "tracking-by-detection" attempt, where one exploits an object detector for the tracking task. Some methods pre-train a classifier on a specific target class and update the model to the current class instance at the beginning of the video, with the argumentation that in many applications one has an a priori knowledge about the target object class [34]. Other methods are able to track arbitrary objects, learned from the first frame of a video sequence, *e.g.*, [3, 18, 66]. As an example, Figure 1.3 shows some frames of a good working tracking video.

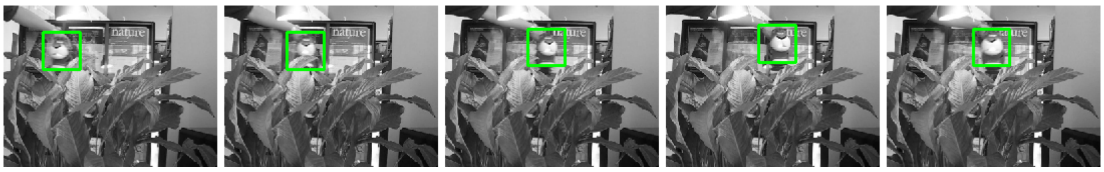


Figure 1.3: Some frames of a good working tracking algorithm which follows a soft toy. The image content is taken from [3].

Most problems in object tracking occur when the target object undergoes small local transformations due to articulation, pose or illumination changes. Another big problem in object tracking is the correct handling of occlussions. Some frames of a failing tracking task are shown in Figure 1.4, where one can clearly see that at some point the tracking algorithm cannot follow the target object anymore.

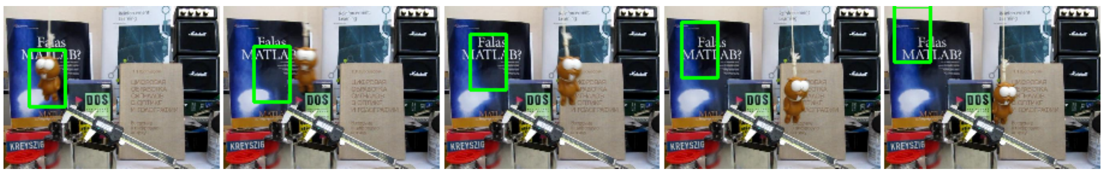


Figure 1.4: Some frames of a failing tracking algorithm. The image content is taken from [67].

To sum up, one can say that object detectors and trackers have to cope with a lot of tedious problems, which include, amongst others, intra-class variance, illumination, and pose changes. Therefore, it would be desirable to build detectors and trackers, which are insensitive to these effects. One source where we can find all of these effects was barely used in the recent research, and this is video. Videos can capture moving objects in real-world scenes, which actually perform natural pose and appearance changes. This information can be used to build transformation invariant detection and tracking algorithms.

1.2 Video

Video is a sequence of static images, which can capture motion from real-world scenes. Each image in a video is called a frame. The time between two consecutive frames is usually very short, *e.g.*, around 30 ms. "Frames per second" (FPS) is a common characteristic to describe this time interval. Due to this temporal coherence of the frames in a video, some assumptions about the captured real-world scene can be made. For instance, if an object is pictured prominently in frame t, it is very unlikely that the same object vanishes in the following frame $t+1$. But on the other hand, we can assume that it is very likely that the captured object has undergone some local transformations, *e.g.*, pose changes. Furthermore, considering the illumination in combination with these local transformations, the object might appear slightly different from frame to frame, especially in terms of color and intensity. For that reasons, video is a rich source of information that can be exploited to make object detectors and trackers insensitive to these transformations.

1.3 Learning from Video

In this thesis, we want to improve a learning algorithm for object detection and tracking tasks with the help of video data. As already mentioned above, video sequences comprise transformations of natural objects. These transformations include pose, appearance, or illumination changes, which all are effects, object detectors and trackers should be invariant to. Thus, it makes sense to extract these information from videos and integrate them to the learning procedure. Another motivation for building transformation invariant detectors and trackers by incorporating videos, which is detailed in the following section, comes from the cognitive science research and the human visual system.

It is important to note that we are solely interested in learning the transformations of moving objects in a video, not the object itself. To this end, we do not need any class labels like “person”, “dog”, or “car” in the video frames. This saves a lot of annotation work, which is a further motivation for this thesis. Nevertheless, the existence of any moving object in the video sequence is assumed. Some frames of reasonable video sequences used in our framework are shown in Figure 1.5. Here one can see a koala bear making a lot of local transformations by moving its body. When talking about unlabeled videos, one important problem arises: how can we extract informative parts from the video, *i.e.*, the foreground? The solution to this problem is the motion itself, as it separates the moving foreground from the static background. Of course, there are further problems like multiple moving objects captured in the image sequence or camera movement, but these problems can be handled and are discussed later in the thesis.

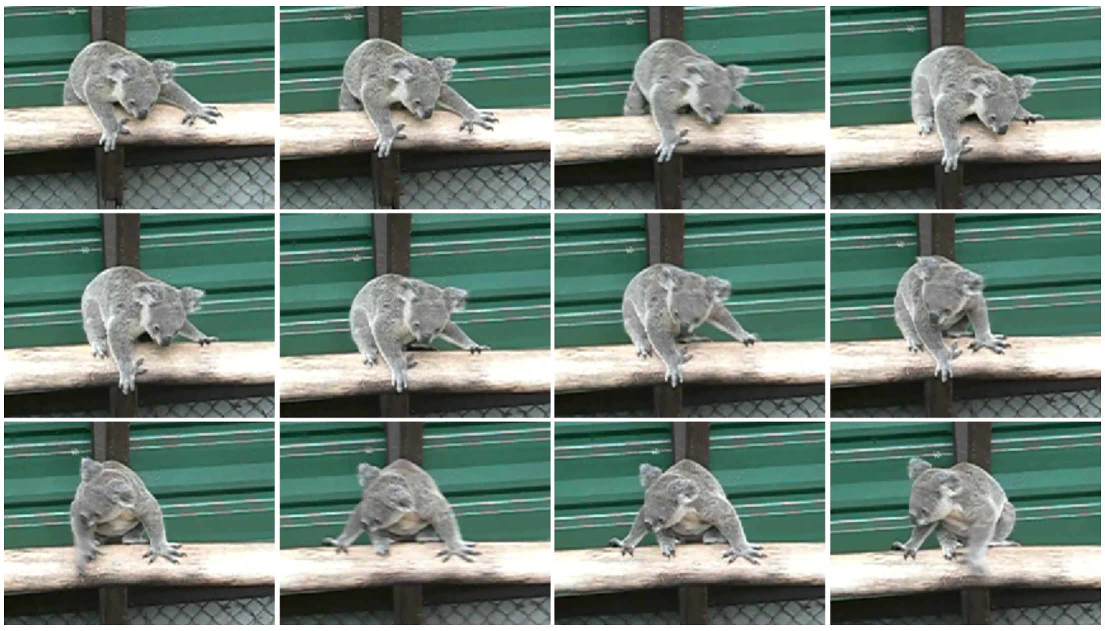
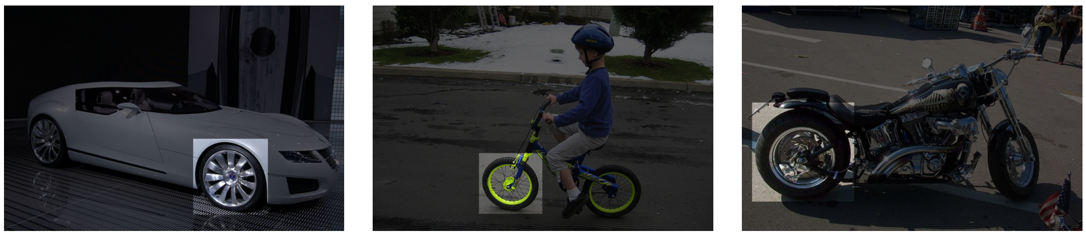


Figure 1.5: Some frames of a video sequence capturing a koala bear moving its body. The different parts of the body undergo several local transformations.

Another assumption, we make, is that local transformations of a specific object are not restricted to solely help a detection system for the same target-object. In other words, this means that we can extract local transformations from moving objects which are only weakly related towards the target class of the object detection or tracking task. Weakly-related objects can be understood as objects, which actually stem from different classes, but show some local appearances, which are similar. Figure 1.6 illustrates why this assumption makes sense. Here, one can see that local transformations of any object can also be associated with parts of any other object. This indicates that data, which describes those local object transformations, can be shared over more than one target class.



(a) Car **(b)** Bicycle **(c)** Motorbike

Figure 1.6: It is very likely that object parts that look similar, i.e., have similar appearance, can undergo similar transformations. This knowledge can then be transferred between unknown categories. The figures (taken from [28]) show weakly-related objects, which all share a common part, the wheels.

As these local object transformations can only be found at different parts of the moving object, we require a part-based learning method. We cannot share the transformations of the entire object over different target-objects, but parts of it. Another reason why it is desirable to use a part-based approach is the problem of the aspect ratio. Aspect ratios of objects in images or videos should be approximately equal to be useful for a common object detection algorithm, but this is often not the case when talking about weakly-related objects. As an example, consider the aspect ratios of trucks and motorbikes, which are obviously dissimilar, although the objects could share some local transformations, *e.g.*, the wheels.

Summing up, we present a part-based learning algorithm which incorporates local transformations of moving objects to improve the generalization power for both, detection and tracking tasks. A schematic overview of the procedure is illustrated in Figure 1.7.

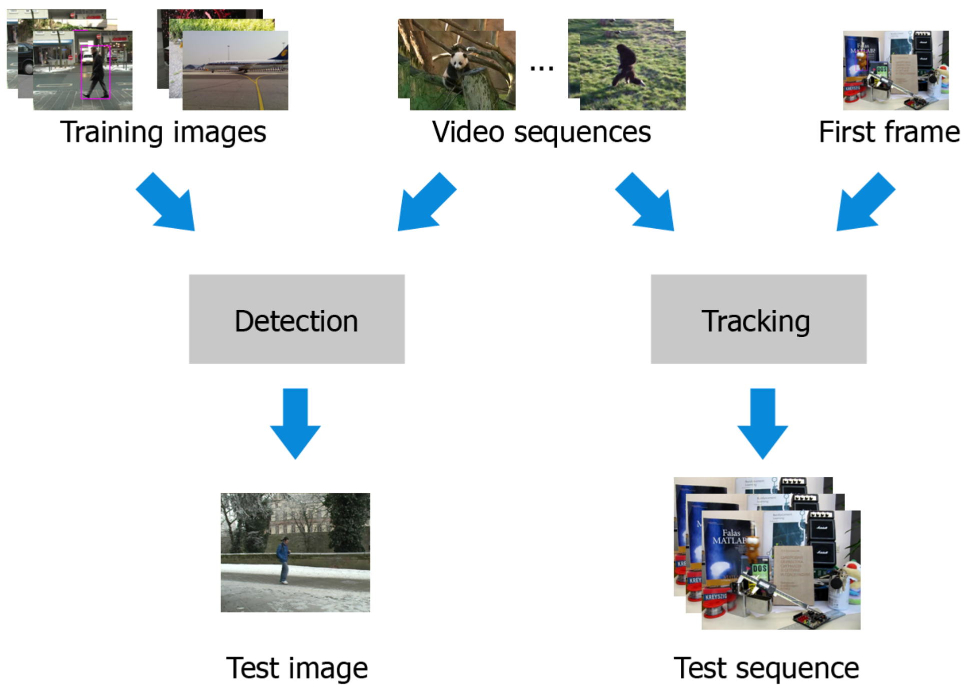


Figure 1.7: A schematic illustration of the object detection and tracking procedures presented in this thesis. Both algorithms incorporate local transformations extracted from video data.

1.4 Human perception

The idea of improving object detection and tracking tasks in computer vision systems by incorporating the information extracted from moving objects in videos, is a very natural one and inspired by the human visual system. Especially, when we think of how the input data for newborns and children look like. They see an "endless live stream" that contains unlabeled moving objects most of the time, and they can outperform current state-of-the-art object detection and tracking systems after a short "training phase". Therefore, it seems to make sense to use unlabeled videos as auxiliary data in a learning process to build transformation invariant detection and tracking systems.

Current research in cognitive science also investigates these problems and tries to find out how human beings accomplish detection and tracking tasks, see [5, 35, 82] for some examples. A more detailed discussion about the relation to the human visual system follows later in the thesis.

1.5 Organisation of the Master's thesis

The intent of this thesis is to improve object detection and tracking tasks by incorporating motion data capturing weakly-related objects. In other words, the generalization performance should be enhanced by learning local transformations of real-world objects, resulting in a transformation invariant object detector and tracker. Additionally, we want to present a procedure, which can extract these motion data, capturing local transformations, from unlabeled video sequences.

The remain of this thesis is organized as follows: Chapter 2 gives a short introduction to statistical machine learning and reviews recent learning approaches that include unlabeled and weakly-related data, see Sections 2.2, 2.3, and 2.4. In Section 2.5, we summarize some former works, which try to incorporate motion data from video sequences and point out the differences to our approach. Furthermore, Section 2.5.1 depicts the relation of the proposed methods to the human visual system.

In Chapter 3, our new learning algorithm is introduced and discussed in detail. Sections 3.3.2 and 3.3.3 show how it can be applied to learn an object detector and tracker, respectively.

In Chapter 4, we outline the processing pipeline which completes our framework to a real-world detection and tracking system. Furthermore, we introduce the notation of the motion data and describe the extraction process including optical flow and motion-based segmentation in Sections 4.2.1 and 4.2.2, respectively.

Chapter 5 illustrates the generated video database, the experimental setup, and the evaluation process. Furthermore, we present the results on real-world data for object detection and tracking tasks, respectively.

Finally, Chapter 6 summarizes the thesis and gives an outlook to future work.

Chapter 2

Related Work

Contents

2.1 Short introduction to Machine Learning 12

2.2 Semi-Supervised Learning 15

2.3 Transfer Learning 18

2.4 Learning from weakly-related data 21

2.5 Learning from video data 22

2.6 Part-based approaches 25

2.7 Chapter Summary 26

In many cases, object detection and tracking tasks can be reduced to a statistical machine learning problem, where one tries to find hypotheses which separate the target class from the background. Thus, we start this chapter by giving a short introduction to statistical machine learning in Section 2.1.

Apart from the standard techniques, machine learning offers more recent approaches that can improve the generalization performance with the help of auxiliary data. In particular, these methods are Semi Supervised Learning (SSL) and Transfer Learning (TL), which try to incorporate unlabeled or weakly-related data into the learning procedure. These methods and the differences to our approach are described in Sections 2.2 and 2.3, respectively. Some other approaches using weakly-related or video data are summarized in Sections 2.4 and 2.5, respectively. Again, we point out the differences to our method.

As already mentioned, we desire a part-based framework for incorporating local transformations from video data. Thus, we also describe some other object detection and tracking systems using a part-based approach in Section 2.6.

2.1 Short introduction to Machine Learning

Machine learning has a broad spectrum of different settings and applications, which is too large to be explained here in detail. As a starting point, we review the two most important settings, *unsupervised learning* and *supervised learning*. Both settings have to deal with input feature vectors drawn from a so-called feature space $\mathcal{X}$, which is a fact, they have in common. We review both settings in Sections 2.1.2 and 2.1.3, respectively. More detailed discussions about this topic can be found in [16, 54, 97], which also served as sources for the discussion in this thesis.

Before we review the learning methods, we give a deeper look on the feature space $\mathcal{X}$ in Section 2.1.1. Furthermore, we show some common examples of feature extraction methods in computer vision applications.

2.1.1 Feature space

A feature vector $\mathbf{x}_i$ describes an instance of any object and consists of d feature values $x_i^j \in \mathbb{R}$, each describing a property of the object. The d-dimensional space $\mathcal{X}$, which is spanned by the feature vectors, is called the feature space. Each feature vector can be represented as a point in this space and can be illustrated in a so-called scatter plot. As an example, let us consider the case of representing human beings in a 2-dimensional feature space. We choose two properties, weight and height, and map them to the x- and y-coordinates in a Cartesian coordinate system, respectively. Figure 2.1a illustrates this mapping. All points (blue and green) correspond to different objects, *i.e.*, human beings. As can be seen easily, there are two different distributions, one with low height and weight, and one with high height and weight.

To continue our example, we now consider the task of separating adults from children according to these two properties (a common scenario in supervised learning). Obviously, blue points in the scatter plot belong to adults and green points to children. In Figure 2.1a, the data can be separated easily, as we just have to draw a line between the distributions.

Figure 2.1b also represents human beings, but now, the two properties mapped to the coordinate system are eye color and hair color. These feature vectors are chosen badly, as we cannot find a simple hypothesis which separates the two distributions, *i.e.*, adults and children. Thus, there exist good and bad choices for feature vectors. The quality of feature vectors is the ability to separate the distributions, they stem from, easily. Features from the same distribution ought to have similar feature values and vice-versa. Figure 2.1 illustrates this difference.

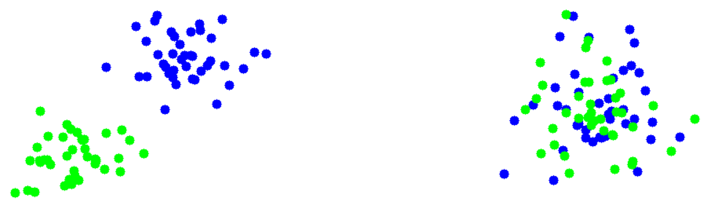

(a) Good features **(b)** Bad features

Figure 2.1: This figure shows an example of good and bad features. Blue and green points correspond to different distributions and ought to be separated easily, which is the case in (a), but not in (b).

In computer vision, we have to extract d-dimensional feature vectors from static images with the goal to find a representation of the target object, which has the ability to be separated easily from feature vectors comprising background clutter. There was an immense progress in feature extraction in the recent years and many different methods have been proposed. A very simple one is to concatenate all grayscale values of the defined target object. More mature methods also include some other image information, like gradients. Prominent examples are the SIFT descriptor [52], HOGs [24], or LBP patterns [1].

2.1.2 Unsupervised Learning

In unsupervised learning, one is given a dataset $\mathbf{X}$ containing N examples $\mathbf{x}_i$. In general, it is assumed that all examples are drawn independently and identically distributed (i.i.d.) from the space $\mathcal{X}$. One example of such a dataset is illustrated in Figure 2.2a. In this scenario, there is no supervision from a teacher who tells how the examples should be handled, hence the name unsupervised learning. This fact can also be noticed from the figure, as all points have the same color. No further information is given, except the feature values itself.

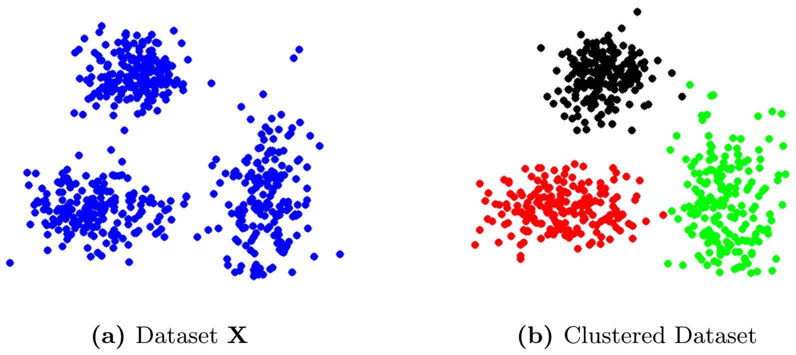

(a) Dataset $\mathbf{X}$ (b) Clustered Dataset

Figure 2.2: This figure describes a typical clustering task. In (a), one can see the given dataset $\mathbf{X}$, *i.e.*, some point clouds in the feature space. In (b), each of these feature vectors is assigned a cluster, indicated with the colors red, green and black.

The main goal in unsupervised learning is to estimate the marginal probability $p(\mathbf{x})$, where $\mathbf{x} \in \mathcal{X}$. But this is a very hard task. However, many other, simpler problems were investigated in the recent research. The most common tasks in unsupervised learning are clustering, outlier detection, or dimensionality reduction. Clustering tries to separate the data into different subsets, where each subset contains similar feature vectors. The similarity measure is often different and task-specific. Figure 2.2 illustrates an example of a clustering task. Well-known algorithms are k-means clustering [51] or mean shift [17]. Outlier detection tries to find those feature vectors, which do not belong to the majority of all other feature vectors in some sense [8]. The task of dimensionality reduction tries to find those dimensions in the feature space which are most informative. Subsequently, other dimensions can be neglected without significant loss in information, which can reduce computational effort in many machine learning applications. Principle Component Analysis (PCA) is the most prominent algorithm [60].

2.1.3 Supervised Learning

Supervised learning differs from unsupervised learning in the fact that, additionally to the feature vectors $\mathbf{x}_i$, its corresponding labels y_i are given. Thus, the learning algorithm is supervised in a way that it is provided with training data $\{x_i, y_i\}$, where $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. A standard requirement is that the training samples $\{x_i, y_i\}$ are drawn i.i.d. over $\mathcal{X} \times \mathcal{Y}$.

The goal in supervised learning is to find a mapping from the input space $\mathcal{X}$ to the output space $\mathcal{Y}$, namely $f : \mathcal{X} \rightarrow \mathcal{Y}$. $\mathcal{X}$ is the feature space described in Section 2.1.1. $\mathcal{Y}$ can be a discrete space, in which case the task is called classification and the function f is a classifier. Thus, the different values in $\mathcal{Y}$ are called classes. If $\mathcal{Y}$ is continuous, the problem is called regression and f is a regression function. The quality of f can easily be evaluated by its predictive power on test samples, which also stem from $\mathcal{X} \times \mathcal{Y}$.

Supervised learning can be separated into two families, generative and discriminative approaches. Generative learning tries to estimate the probability $p(\mathbf{x}|y)$, *i.e.*, it wants to know how the feature vectors $\mathbf{x}_i$ are generated. The second approach, discriminative learning, is not interested in this probability. Instead, it just models $p(y|\mathbf{x})$. Thus, it simply tries to discriminate the feature vectors $\mathbf{x}_i$ to its correct labels y_i.

Figure 2.3a shows an example of a learned function f for a given training set. The blue points belong to class 1 and the green points to class 2. This linear classifier separates the training data well. In Figure 2.3b, one can see the evaluation on a test set. Again, the classifier f can predict the classes of the given feature vectors with just a small error. Certainly, the complexity of the classifier can be increased to *e.g.*, non-linear, but for the purpose of explanation, we restrain the classifier to be linear.

2.2 Semi-Supervised Learning

Semi-Supervised learning (SSL) is somewhere midway between supervised and unsupervised learning. In this case, the learning algorithm is given both, labeled and unlabeled data. We define the labeled dataset as $\mathbf{X}_l$ which contains l labeled examples $\{\mathbf{x}_i, y_i\} \in \mathcal{X} \times \mathcal{Y}$. Additionally, the algorithm is given an unlabeled dataset $\mathbf{X}_u$ containing u feature vectors $\mathbf{x}_i \in \mathcal{X}$. Thus, only a subset of all feature vectors has supervision in form of labels y_i.

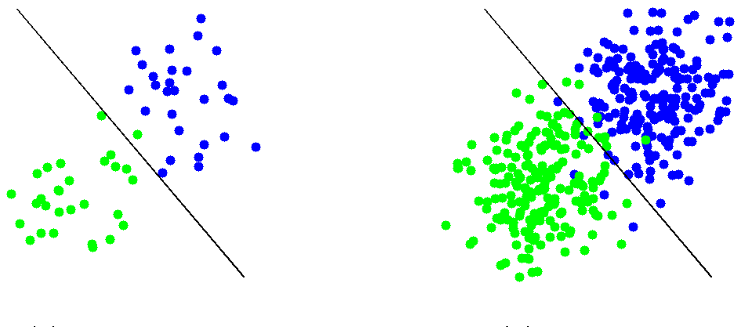


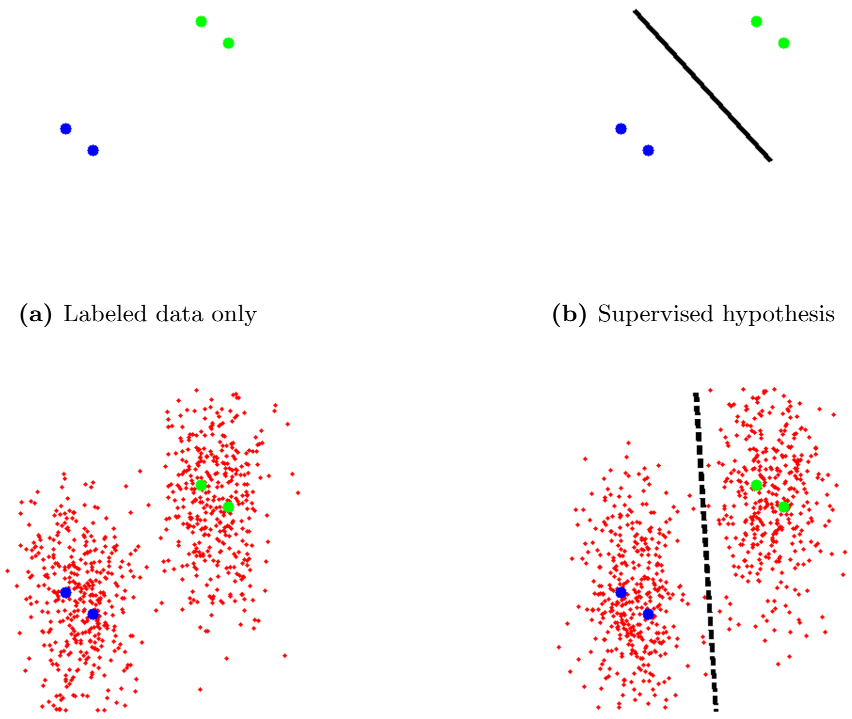
Figure 2.3: This figure illustrates the supervised learning task. In (a), the training data is presented (blue points from class 1 and green points from class 2) and the trained classifier f. (b) shows the evaluation of f on the test dataset.

SSL offers many different tasks, amongst others, these are semi-supervised classification, constrained clustering, regression, or dimensionality reduction. But only the first setting is the most suitable for this thesis and, thus, is discussed in more detail.

In Semi-supervised classification, SSL is considered as an extension to supervised learning, where additional unlabeled data is available. Typically, it is assumed that $l \ll u$, *i.e.*, the amount of unlabeled data is much larger than that of labeled data. Similar to supervised learning, the goal is to train a classifier $f : \mathcal{X} \to \mathcal{Y}$, but now from both, labeled and unlabeled data. Actually, in this scenario, one can differ between two slightly different settings, inductive and transductive SSL. The goal in the first setting is to achieve good generalization performance on future data, *i.e.*, beyond $\mathbf{X}_l \cup \mathbf{X}_u$. The latter one tries to find that function f, which has good predictive power on the unlabeled data $\mathbf{X}_u$ only. Of course, this setting is simpler, as f is just defined on the given training set, *i.e.*, $f : \mathcal{X}^{l+u} \to \mathcal{Y}^{l+u}$.

Unlabeled data can vastly improve the generalization performance of a classifier f, if used correctly. Otherwise, a classifier including unlabeled data can yield worse results than a classifier, which is trained solely on labeled data. The key to the success of SSL are the assumptions one makes about the link between the marginal distribution $p(\mathbf{x})$ and the target label y. Many SSL settings have slightly different assumptions about this link. A good summary can be found in [97], which also served as a base for this discussion. The book from Chapelle *et al.* [16] also gives a good introduction to Semi-supervised learning.

As an example for semi-supervised classification, consider the scenario depicted in Figure 2.4. The upper row of images illustrates the supervised learning scenario, where only two datapoints are available for each class, respectively. A hypothesis is found, which seems to separate the data well (see Figure 2.4b). In the lower row of images, unlabeled data (red points) are added. In Figure 2.4c, one can easily see both distributions of the two different classes and a low density region between them. Thus, one can find a much better hypothesis by considering the unlabeled data points, which is depicted in Figure 2.4d. This example demonstrates one problem of supervised learning. The labeled training dataset does not always reflect the properties of the distribution they stem from, especially, if the number of labeled examples is small.



(a) Labeled data only **(b)** Supervised hypothesis

(c) Labeled and unlabeled data **(d)** Semi-supervised hypothesis

Figure 2.4: This figure illustrates a simple SSL scenario. (a) and (b) show the supervised case of learning a hypothesis from the given labeled dataset. In (c), the dataset is extended with unlabeled data points and in (d), the newly trained hypothesis is illustrated.

SSL has a lot of different applications, where using unlabeled data helps to improve the performance. Amongst others, these applications include speech recognition [93], natural language processing [11, 44], spam filtering [9, 89], protein 3D structure prediction [85, 86], or website classification [73, 96]. Thus, one can find many SSL algorithms in the recent literature. For instance, the transductive SVM (TSVM) [41, 43] extends the well-known SVM to incorporate unlabeled data. Similar, a semi-supervised Random Forest was presented by Leistner *et al.* [49]. Apart from improving the performance, SSL has another big advantage. A lot of time and cost can be saved for annotating training data, as unlabeled data is often much easier to collect than labeled data.

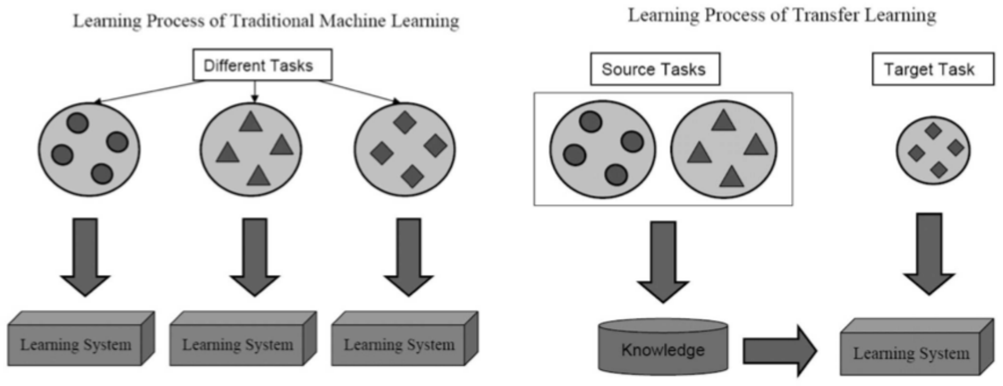
It is also worth noting that SSL makes a different assumption than we do in this thesis. Although both approaches include unlabeled data, we do not make such a strong assumption between the marginal distribution $p(\mathbf{x})$ and the target label y. SSL methods typically assume that all data points are drawn from the same distribution, but we also try to learn from weakly-related data, *i.e.*, data from different distributions.

2.3 Transfer Learning

Supervised and semi-supervised learning both assume that the training and test data are drawn from the same distribution. Thus, a classifier has to be build from scratch with newly collected training data, once the target domain of the task changes. This is expensive and time consuming in most of the cases. It would be desirable to transfer some knowledge from the old task to the new one, *i.e.*, to reuse previously learned knowledge. Transfer learning (TL) tackles this problem. As a practical example, consider the case of learning to speak Spanish, which is often said to be easier, if you already know how to speak Italien. A good summary about this learning principle can be found in [59], which also served as a basis for this review here.

The research on transfer learning was fundamentally motivated by the works of Schmidhuber [70] and Thrun & Mitchell [78]. TL can be applied to different machine learning settings, *e.g.*, classification, regression, or clustering. Figure 2.5 shows the basic principle of transfer learning compared to traditional machine learning.

The notation in transfer learning defines a “domain” and a “task”, which is similar to the ordinary machine learning notation. A domain $\mathcal{D}$ consists of a feature space



(a) Traditional machine learning **(b)** Transfer learning

Figure 2.5: This figure from [59] illustrates the basic principle of transfer learning. In (a), one sees the traditional machine learning approach. Each task has its own learning system. (b) shows the approach of transfer learning, where knowledge is shared over different tasks.

$\mathcal{X}$ and its marginal probability distribution $p(\mathbf{x}_i)$, where $\mathbf{x}_i$ is a feature vector in the space. A task $\mathcal{T}$ consists of the label space $\mathcal{Y}$ and a predictive function $f(\cdot)$, which has to be learned from the training data [59].

Transfer learning is categorized in three different settings: inductive, transductive and unsupervised transfer learning. This division is based on the different relations between source and target domain or source and target task. An overview is illustrated in Figure 2.6. Inductive transfer learning assumes different source and target tasks, independent from the domains, *i.e.*, source and target domains can be equal or different. In transductive transfer learning, source and target tasks are the same, but the domains are different. That is, either the feature space or the marginal probability distribution from source and target domains is different. Unsupervised transfer learning is similar to the inductive transfer learning setting, in a sense that source and target tasks are not the same, but they are related. However, this category concentrates on solving unsupervised target tasks, like clustering or density estimation. No labeled data is available at all.

A lot of different applications and algorithms have been proposed in the recent years in the context of transfer learning. Dai *et al.* [23] present an extension to the boosting framework, which can effectively transfer knowledge from old data to

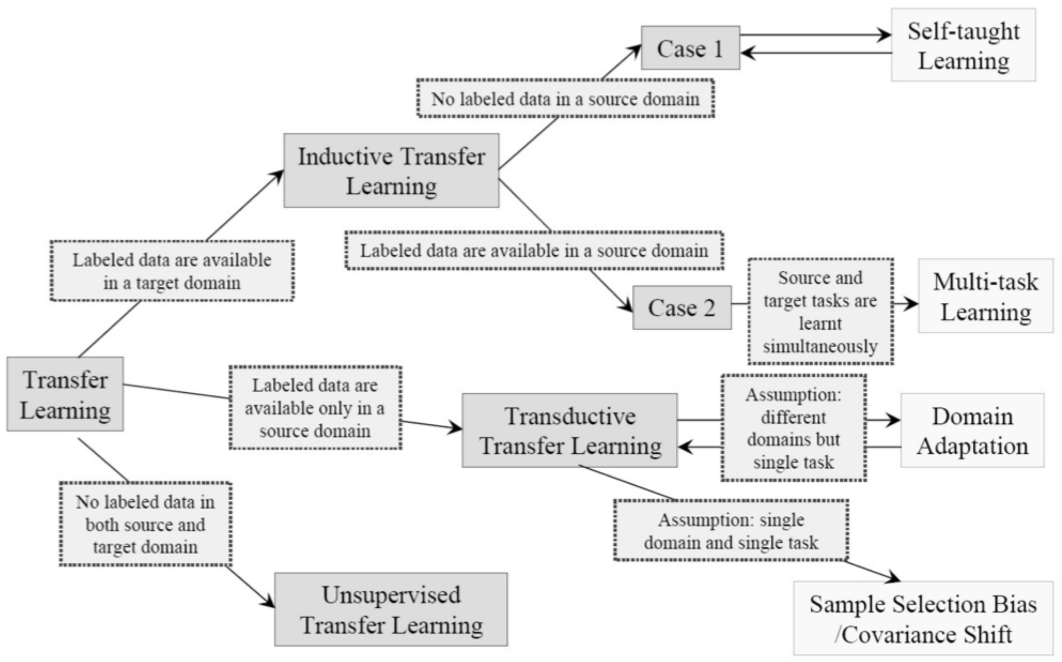

Transfer Learning
Labeled data are available in a target domain
Inductive Transfer Learning
No labeled data in a source domain
Case 1
Self-taught Learning
Labeled data are available in a source domain
Case 2
Source and target tasks are learnt simultaneously
Multi-task Learning
Labeled data are available only in a source domain
Transductive Transfer Learning
Assumption: different domains but single task
Domain Adaptation
Assumption: single domain and single task
Sample Selection Bias /Covariance Shift
No labeled data in both source and target domain
Unsupervised Transfer Learning

Figure 2.6: This figure gives a detailed overview of the different categories of the transfer learning problem [59].

a new target domain. They also give a theoretical and empirical analysis to their algorithm. Text classification is one of the applications, transfer learning has been applied to successfully [21, 22, 64]. Furthermore, this learning principle has been used in computer vision tasks. Stark *et al.* [74] present a shape-based model and show how to transfer knowledge for other target-classes. Three different levels of knowledge are transferable between object models [74]. These are the appearance of different parts of the model, local symmetries, *e.g.*, limbs of human beings, and the topology of parts. In [37], Heitz *et al.* use a landmark-based model to transfer knowledge from cartoon images to natural ones for object classification. An extended SVM-based framework using auxiliary data is presented in [88] and applied to the task of image classification.

Although some aspects of transfer learning are similar to our approach, we differ from this principle as we concentrate on learning motion transformations from video sequences. That is, in our method, the unlabeled auxiliary data has a different form than the labeled data.

2.4 Learning from weakly-related data

This section reviews some works about learning from weakly-related data. Although they have a strong relation to transfer learning, which was described in the previous section, they do not explicitly exert this learning principle.

Self-taught learning, introduced by Raina *et al.* in [63], proposes a new learning method, which uses the help of unlabeled data to improve a classifier. Crucial, the unlabeled data may stem from a different distribution or domain than that of the labeled data. The main idea in [63] is to use sparse coding methods to learn basis features from the unlabeled, weakly-related dataset. Then, the labeled images (and also the test images) are represented using a linear combination of these basis features. Consequently, the linear combinations are assigned a label and can be forwarded to a simple supervised learning algorithm, like an SVM or boosting.

Yang & Jin [90] propose an SSL approach to learn an SVM from both labeled and unlabeled data, where the unlabeled dataset is only weakly-related to the labeled one, *i.e.*, the domains of the datasets are different. The main application in [90] is text-categorization. This approach is similar to that of self-taught learning, but, in contrast to Raina *et al.* [63], here the data representation is combined with the training of the classifier. In particular, Yang & Jin [90] use both labeled and unlabeled data to find a feature representation.

In contrast to these methods, where unlabeled, static images are used as auxiliary data to increase the amount of available objects in the dataset, we learn from video data and try to exploit the motion of objects.

In [46], attribute-based classification is introduced, which is a method that shows how to learn a classifier without any training examples of the target class. Objects are described with a high-level representation based on arbitrary semantic attributes, like shape, color or, even geographic information. Thus, a classifier can be pre-trained on a database that is unrelated to the current target class, and new classes can be detected without another training phase. As an example, the attribute *yellow-colored* corresponds to bees, tigers, canary birds, and many more. Thus, one can introduce an intermediate layer, consisting of attributes, between low-level features and high-level object classes. A classifier can be trained to extract object attributes from images, and due to the attributes, one can detect new classes, *e.g.*, an elephant, which could be described with the attributes: gray, big, long trunk, living in warm regions. A disadvantage of this method is that one has to label the intermediate layer, *i.e.*, the attributes. In contrast, we do not need to label any attributes in the auxiliary data, like “hand”, “head”, or “limb”. We do not even know the label of the weakly-related objects.

2.5 Learning from video data

In this section, we provide a review of papers which use the help of sequential data to receive transformation invariant representations of target objects. Many of the presented methods explicitly try to learn motion invariant feature vectors while the remain integrates the invariance in the learning phase of the classifier.

In [25], features are presented which combine HOGs with orientation histograms of either optical flow or spatio-temporal derivatives. That is, these features are extracted out of two successive frames taken from a video sequence. The evaluation of these spatio-temporal features is done with an SVM on pair-wise test images.

Another object representation, which is based on a combination of static image features and motion-based features, is shown in [81]. The system integrates intensity with motion information. Viola *et al.* [81] propose an efficient method to extract motion from two successive frames. The generated features are used to train an AdaBoost classifier and the framework is applied to the task of pedestrian detection. The evaluation is done on pair-wise images extracted from video sequences.

LeCun *et al.* [77] describe a convolutional architecture based on Restricted Bolzmann Machines (RBM) to learn spatio-temporal features from successive frames in video sequences. The extracted motion-sensitive features correspond to the transformations between the two frames. [77] show on synthetic data that their method can represent flow-like features when the type of transformation is restricted. This approach is also used to extract useful features for human activity recognition. Again, the testing process is done on pair-wise data.

Slow feature analysis (SFA) is another method, introduced by Wiskott & Sejnowski in [87], to learn invariant object representations for classification tasks in an unsupervised fashion from temporal input sequences. Based on a nonlinear expansion of the input signal and a principle component analysis (PCA), it learns a large number of decorrelated and invariant features. A hierarchical network of SFA modules can learn translation, size, rotation, and contrast invariance for one-dimensional objects. However, the performance degrades if the network is trained to learn multiple invariances at a time. The principle learning goal is to find a function $g(\cdot)$ which is applied on the sequential input signal $x(t)$ and results in a slowly varying output function $y(t) := g(x(t))$. Of course, the trivial constant output signal has to be avoided.

All works described in the previous few paragraphs try to construct motion-invariant features to learn object representations, unlike our method, presented in this thesis. We try to incorporate the motion data in the training phase of a classifier and use ordinary object representations, like HOGs. Furthermore, despite SFA, all the methods described above, can only be applied to sequential data during training and evaluation. In contrast, our method is able to learn from motion data but is evaluated on single images.

Stavens & Thrun [75] describe how to exploit unlabeled real-world sequential data via optical flow to tune the parameters of prominent object representations, namely, SIFTs and HOGs. The goal is to learn domain-optimized versions of SIFT and HOG features in an unsupervised way. They also argue that optimizing the parameters manually is too time consuming, and a more automated process, using warping images to simulate motion, tends to be synthetic and does not include all variations from the dataset. This argumentation is very similar to ours in this thesis, but we try to improve the learning algorithm itself, via incorporating real-world videos, not the feature representation. [75] estimate the correspondences between frames using Harris corner features and an optical flow. Each corner feature is tracked and assumed to capture naturally transformed salient points in the videos. In a second step, an efficient and effective search procedure is used to find optimal values for the feature parameters of SIFT and HOG. Of course, the goal is to tune the parameters in a sense that the object representation shows similar values for the same real-world objects, but distinctive values for different ones. As this method is completely unsupervised and, although the performance is very good in the experiments, there is a limitation [75]. Due to multiple occurrences of the same object in the videos, the algorithm occasionally tries to find distinctive representations for the same objects.

Continuous transformation (CT) learning is introduced in [76] and strongly motivated by the human visual system. They use a neural network (NN) with some input and output cells. The goal is to train this network in a sense that tiny shifts or transformations of the input stimuli always yield the activation of the same output cells. This procedure relies on the fact that shifted or transformed versions of the input stimuli have a lot of overlapping input cells. In [61], this CT learning approach, based on a hierarchical network model, is used to build translation invariant object representations. Again, they motivate their work by the human visual system, where many papers show that neurons in the visual cortex respond to objects invariantly with respect to translation, size, contrast, lighting, spatial frequency, and view.

Mobahi *et al.* [55] also exploit the temporal coherence in unlabeled video data to improve a supervised object detection task, in a similar way to ours. Using a deep architecture of convolutional networks, they try to become more invariant to object transformations and to yield better results in detection tasks. They propose a training objective, which is regularized, based on temporal coherence data terms. This is a very similar approach to ours in the task of object detection, but they use a self-created video database, which contains artificial 3D objects without any background clutter. In contrast, we use real-world data and a more light-weight part-based approach, which is more flexible for practical usage.

2.5.1 Human perception

As already mentioned in Section 1.4, we argue that there is a strong relation between the human visual system and the proposed learning method in this thesis. We believe that infants learn object representations in the first few months of life, supported by a huge amount of unlabeled video sequences with a minimal amount of supervision. In more detail, the motion itself or the temporal coherence of moving objects in the real world gives the supervision needed to learn robust object representations, which are very discriminative between different object classes, while being robust against intra-class variations. In the recent years, the cognitive science research investigated a lot of work in the question how infants are able to learn robust object representations in such a short time, and also supports our reasoning.

Gerhardstein *et al.* [35] states that one source of the robustness and performance of the human visual system is the prior, or top-down knowledge, which helps to better understand the real-world. This statement is widely accepted in cognitive science and easily comprehensible. But they further concentrate on the other source, which does not assume any prior experience. Crucial, they argue that motion cues play a very critical and important role when no prior knowledge is available, *e.g.*, for infants. Children are attracted to motion, while in contrast, it is more difficult for them to use static information, like color. Motion information is not as prone to ambiguous interpretation, due to the temporal coherence in real world scenes.

Wallis & Bülthoff [82] support the argumentation that the temporal coherence of moving objects is important to object recognition or to the task of finding good and invariant object representations. Some experimental results in [82] suggest that spatiotemporally smooth sequences improve the process of learning more invariant object representations, which allow some intra-class variation.

Furthermore, Balas & Sinha [5] also describe that dynamic input for visual systems, both computational and biological, shows a performance-enhancing impact. Assisted by two different experiments, they show that temporal coherent input data increases both the generalization of object identity and the sensitivity to appearance changes in static images.

In [50], Li argues that the responses from the inferior temporal cortex (IT) are discriminative to different objects but also very invariant to changes in object position, scale, and pose. To learn these invariances in the object representation, they propose a solution based on the temporal coherence of the visual experience of human beings. Unsupervised temporal slowness learning (UTL) is presented, which could be the mechanism how humans find tolerant object representations.

2.6 Part-based approaches

As we also make use of a part-based object detection and tracking approach, this section reviews some recent works on this topic. Although many different part-based approaches have been proposed [19, 26, 27], we limit ourself to review two works in more detail.

Felzenszwalb *et al.* [29] present a deformable part-based model which rest upon discriminatively trained support vector machines. The object model consists of a root filter, *i.e.*, an SVM trained on the representation of the whole target object, and several part filters covering different parts of the objects. Quadratic cost functions score all deformations of the model, which controls the flexibility of the part filter positions.

This model from [29] achieved brilliant results on the PASCAL challenge [28] in the recent years and outperformed many other participants.

Another part-based approach is presented by Wang *et al.* in [83]. A combination of HOG and LBP features served as feature representation and the algorithm was applied to the task of human detection. Additionally, the partial occlusion problem is handled in this work. The method is based on separate detectors, a global and several part detectors, which is similar to [29]. Wang *et al.* [83] use a scanning window approach in the detection phase. Ambiguous windows are segmented, based on an occlusion likelihood map, to separate occluded from non-occluded regions. The part detectors are then applied to occlusion-free regions, yielding the final classifier.

Furthermore, the proposed method in this thesis also stems from a part-based object detection framework, which is reviewed later in Section 3.3.1.

2.7 Chapter Summary

In this chapter, we reviewed the basic notation, settings, and problems in machine learning and gave some examples. Then, we introduced some variations of the ordinary machine learning setting, which include auxiliary data in the learning process. Namely, these are Semi-Supervised learning in Section 2.2 and Transfer learning in Section 2.3. Furthermore, we reviewed some works in the recent research about learning from weakly-related data and learning from video. In Section 2.5.1, we made a short excursion to the human visual system and pointed out some similarities and relations to our approach, presented in the thesis. Finally, we reviewed some former works on part-based approaches.

Chapter 3

Hough Forests with Motion Data

Contents

3.1 Random Forests 28

3.2 Learning from motion data 30

3.3 Extending Hough Forests to learn from motion data 32

3.4 Chapter summary 39

As already stated in the previous sections, to perform object detection and tracking while incorporating motion data, we need a learning algorithm, which is able to handle a part-based approach. In addition, we require the learning algorithm to be online, which is necessary for object tracking. A current state-of-the-art detection system using a part-based approach was introduced by Gall & Lempitsky in [33], the Hough Forests. They have been shown to perform well in different object detection tasks. Furthermore, [34] shows how this framework can be updated online and presents good results in different tracking tasks. As the implementation is also publicly available, this framework seemed very suitable as a base for adopting our methods, which are presented in this chapter.

Random Forests [15] are the basis of the Hough Forest framework and will be summarized in the following section and extended to learn from motion data in Section 3.2. Afterwards, we show how we modified the Hough Forest framework to incorporate our methods. In Section 3.3.2, we describe the integration of motion data via regularization to accomplish the detection task and Section 3.3.3 shows the training of a general codebook, which is used for object tracking.

3.1 Random Forests

A Random Forest, introduced by Breiman [15], is an ensemble method and consists of randomized decision trees, which operate independently during training and evaluation time. For each tree a subset of the training data is generated by subsampling with replacement from the original dataset, *i.e.*, bagging. The remaining examples, which are not included in the training subset, are called Out-of-bag (OOB) examples and can be used to compute an Out-of-bag error (OOBE) [15].

We denote a tree in the RF as $f_i(\mathbf{x};\Theta)) : \mathcal{X} \rightarrow \mathcal{Y}$, where $i = 1 \dots T$ and T is the number of trees. Θ describes the various stochastic properties of the tree, like the random training subset or the splitting rules of each node, and $\mathbf{x}$ is a feature vector. The whole random forest can be written as $F = (f_1, \dots, f_T)$. During training, each decision node generates a predefined number of random tests and calculates response values for each training example. There exist several possibilities to define such a random test, the easiest one is to randomly choose one feature value of the d-dimensional feature vector $\mathbf{x}$ and use this as response value. While extremely randomized forests [36] also use a predefined number of random threshold values in a range corresponding to the response values, conventional RFs sweep the threshold values in that range to split the training data into two subsets. We denote such a splitting test as follows

$$\xi(\mathbf{x};\hat{\Theta},\tau) = \begin{cases} 0, & \text{if } \mathbf{x}(\hat{\Theta}) < \tau \\ 1, & \text{otherwise} \end{cases} , \tag{3.1}$$

where $\hat{\Theta}$ are the stochastic elements needed to calculate a response value from the training data $\mathbf{x}$ in the current node and τ is a threshold value. Thus, a training example $\mathbf{x}$ will be forwarded to the left or right child node of the current node, according to this splitting rule 3.1. Over all these possible splits, the node chooses that combination of $\hat{\Theta}$ and τ which optimizes the information gain

$$\Delta H = -\frac{|I_l|}{|I_l| + |I_r|} \cdot H(I_l) - \frac{|I_r|}{|I_l| + |I_r|} \cdot H(I_r) \,, \tag{3.2}$$

where I_l and I_r are the subsets of training data, which are forwarded to the left or right child node, respectively. The node score $H(I)$ is often calculated using the entropy $H(I) = -\sum_{k=1}^{K} p_k^j \cdot log(p_k^j)$ or the Gini Index $H(I) = -\sum_{k=1}^{K} p_k^j \cdot (1 - p_k^j)$, where K is the number of classes. p_k^j defines the probability of node j belonging to class k. In other words, p_k^j is the fraction of training examples belonging to class k which fall into node j to all training examples falling in this node.

After this random test is fixed for the current node, the training examples are split according to Equation 3.1 and forwarded to the left or right child nodes. Then, the same procedure starts again for both child nodes. That is, the trees are grown in a recursive manner. If a decision node is provided with too few or pure training data (only training examples from one class are available), or the depth of the tree reached its maximum, a leaf node is created. The training examples, which fall into this leaf, are used to build the density function about the class labels, which is further used to do prediction.

The prediction probability of class k in the RF can thus be computed as

$$p(k|\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} p_t(k|\mathbf{x}) \; , \tag{3.3}$$

where $p_t(k|\mathbf{x})$ is the estimate of class k in the leaf of the t^{th} tree, where the data sample $\mathbf{x}$ fell into. This structure of holding class probabilities in the leaf nodes makes the RF inherently multi-class. Hence, according to Breiman [15], one can define a multi-class decision function as

$$C(\mathbf{x}) = \arg\max_{k \in \mathcal{Y}} p(k|\mathbf{x}) \; , \tag{3.4}$$

and, furthermore, a classification margin like

$$m_l(\mathbf{x}, y) = p(y|\mathbf{x}) - \max_{\substack{k \in \mathcal{Y} \\ k \neq y}} p(k|\mathbf{x}) \; . \tag{3.5}$$

Note, predicting the correct class label y for example $\mathbf{x}$, requires the margin to be bigger than zero. Therefore, one can define the generalization error like

$$GE = E_{(\mathcal{X},\mathcal{Y})}(m_l(\mathbf{x}, y) < 0) \; . \tag{3.6}$$

Breiman [15] showed that the generalization error GE has an upper bound given by

$$GE \leq \bar{\rho}\frac{1 - s^2}{s^2} \; , \tag{3.7}$$

where $\bar{\rho}$ defines the mean correlation between two pairs of trees, *i.e.*, $\bar{\rho}$ measures the similarity between the predictions of two trees. The strength s of the set of trees is defined as $s = E_{(\mathcal{X},\mathcal{Y})}(m_l(\mathbf{x}, y))$. That is, a high strength value and high independency of the trees increase the generalization performance. Clearly, $\bar{\rho} > 0$ has to hold. A good summary about ensemble methods and the Random Forest framework can be found in [48], which also served as a template for this review.

Random Forests showed to perform better or at least equal to state-of-the-art methods in a various number of computer vision tasks. For instance, RFs are applied to image classification [13] or clustering tasks [56]. Furthermore, they were also used in the task of image segmentation [68, 71].

RFs have become very popular, not least because they own some advantages to other classifiers, which are attractive for computer vision applications. Especially, both training and prediction phases can be performed fast, as RFs are parallelized easily due to the independence of the trees. Multi-Core CPU and GPU implementations [72] have been shown to improve training as well as detection speed heavily. Another advantage of RFs is its inherent capability of making multi-class predictions without having to build several binary classifiers. Furthermore, [15] also states that RFs are more robust to label noise than many other classifiers. This characteristic also suits very well for our task, as we produce a certain amount of noise due to the unlabeled videos, where we extract our motion data from.

3.2 Learning from motion data

In order to incorporate motion data in the Random Forest framework, we need a representation for motion and a suitable splitting criterion. The goal is to train a Random Forest based on motion data, while the evaluation phase handles non-sequential data.

We propose to represent the motion data in form of pairs $\Phi_i^j = (\mathbf{x}_i^t, \mathbf{x}_i^{t+\Delta}, y_i)$, where $\mathbf{x}_i^t$ is a feature vector at frame t and $\mathbf{x}_i^{t+\Delta}$ at frame $t + \Delta$. Note, we do not consider the part-based approach at this point, thus $\mathbf{x}_i^t$ is a general feature vector describing a moving object. Δ is a parameter, which is used to tune the time (in # frames) between the two extracted features and inherently the level of transformation of the moving object. The training pairs are gathered over different video sequences $\mathcal{V}_j$, $j = 1 \ldots \#$ videos, and are collected in a pool of pairs Φ_i^j, $i = 1 \ldots \#$ pairs. $y_i \in \{same, different\}$ is the label of a pair Φ_i^j and simply describes if both feature vectors are extracted from the same video sequence containing the same moving object ($y_i = same$), or not ($y_i = different$). That is, “different” pairs are created artificially.

After describing the representation of the motion data, we need to define a splitting criterion which is capable of handling motion data in form of pairs Φ_i^j. In fact, the RF has to split the pairs based on their feature vectors $\mathbf{x}_i^t$, which is important for the evaluation phase of non-sequential data. Thus, the actual splitting of feature vectors into left and right child node remains equal to the standard Random Forest, but the quality measure for these splits in the training phase is different, in a way that it also integrates the pair-wise motion data.

As we want the RF to become invariant to local object transformations, we try to find splits which keep “same” pairs together and split “different” pairs apart in each node of the tree during training. To this end, we define a so-called pair error ε_{pair} which can be written as

$$\begin{aligned} \varepsilon_{pair} = & \frac{1}{2 \cdot |I_{same}|} \sum_{I_{same}} I(\xi(\mathbf{x}_i^t; \hat{\Theta}, \tau) \neq \xi(\mathbf{x}_i^{t+\Delta}; \hat{\Theta}, \tau)) \\ & + \frac{1}{2 \cdot |I_{different}|} \sum_{I_{different}} I(\xi(\mathbf{x}_i^t; \hat{\Theta}, \tau) = \xi(\mathbf{x}_i^{t+\Delta}; \hat{\Theta}, \tau)) \,, \end{aligned} \tag{3.8}$$

where $I(\cdot)$ is the indicator function and ξ denotes the random test for the current node, as described in Section 3.1. Each node in the tree tries to find that random splitting test, which minimizes Equation 3.8. Thus, the optimizer is penalized for (a) a random test $\hat{\Theta}$ and threshold τ, which forwards the instances from a pair Φ_i^j which is labeled as “same” to different child nodes, and (b) a random test $\hat{\Theta}$ and threshold τ, which forwards both instances of a pair labeled as “different” to the same child node. Of course, pairs that are split apart in any node of the tree have to be discarded from the training set and cannot be forwarded to any child node, which is desirable in the case of “different” pairs, but is penalized for “same” pairs.

Although we train the RF on pair-wise data Φ_i^j, prediction is done on single instances $\mathbf{x}_i^t$. Despite the fact, that we do not have any class-specific information in the leaves of the RF at this point, the classifier is now applicable for detection and tracking tasks.

Discussion: Till now, we briefly described the main properties of this splitting criterion. The RF handles pair-wise motion data during training and non-sequential data during evaluation. It tries to split "different" pairs apart and keeps "same" pairs together. Due to the fact that this rule tries to keep "same" pairs together, it learns a more transformation insensitive representation of objects. In more detail, the algorithm searches for a split, which chooses those features from the feature vectors that are most invariant to local transformations, while still separating "different" pairs.

In contrast to the standard Random Forest, a split with one empty child node is a reasonable solution and might even be the optimal one. The optimal solution of this pair-wise splitting criterion simply is to keep all "same" pairs together and split all "different" pairs apart, independent from the direction they are forwarded to. This can end in one empty child node. The trivial solution, *i.e.*, a threshold τ below or above all responses, which would keep all "same" pairs together but also the "different" pairs, is inherently prevented due to the error on the set of "different" pairs. Such a threshold τ would forward all pairs to either the left or right child node and make a minimal error on the "same" pairs but a maximal one on the "different" pairs. Thus, the combination of the error on the set of "same" pairs and "different" pairs keeps the balance of the splitting criterion.

3.3 Extending Hough Forests to learn from motion data

A Hough Forest, as introduced in [33], is based on the standard Random Forest framework and uses a part-based approach. Hough Forests have been shown to perform well in detection tasks and, furthermore, [34] described how to use them in the task of tracking. These facts make them attractive for our purposes and, as the code is also publicly available, we chose this framework as a basis for our ideas.

The following section summarizes the Hough Forest framework. Section 3.3.2 shows how we can extend it to simultaneously learn from labeled data and unlabeled motion data for detection. In Section 3.3.3, we review how the framework can be updated online [34] and how we modified it to perform tracking with assistance of motion data.

3.3.1 Hough Forests

Hough Forests [33] use a part-based approach to perform object detection. Therefore, the training data consists of small 16x16 patches $P_i = (\mathbf{I}_i, y_i, \mathbf{d}_i)$, where $\mathbf{I}_i$ describes the appearance of the patch, y_i is the class label and $\mathbf{d}_i$ is a vector pointing to the center of the target object. The appearance $\mathbf{I}_i$ of a patch P_i consists of C channels, each of them itself a 16x16 image describing different properties of the extracted image, *e.g.*, the gradients. Thus, the appearance of a patch can be written as $\mathbf{I}_i = (I_i^1, \ldots, I_i^C)$. In case of object detection, positive patches are collected from training images containing the target class and negative patches from background images. For negative patches, $\mathbf{d}_i$ is undefined. Figure 3.1 shows some training images from a pedestrian dataset, where the red box denotes the bounding box of the target object. Patches P_i are extracted at random locations within the bounding box and can be seen in Figure 3.1b. Before the patches are extracted, the training images are resized to a suitable common size, in a sense that 16x16 patches contain useful information.

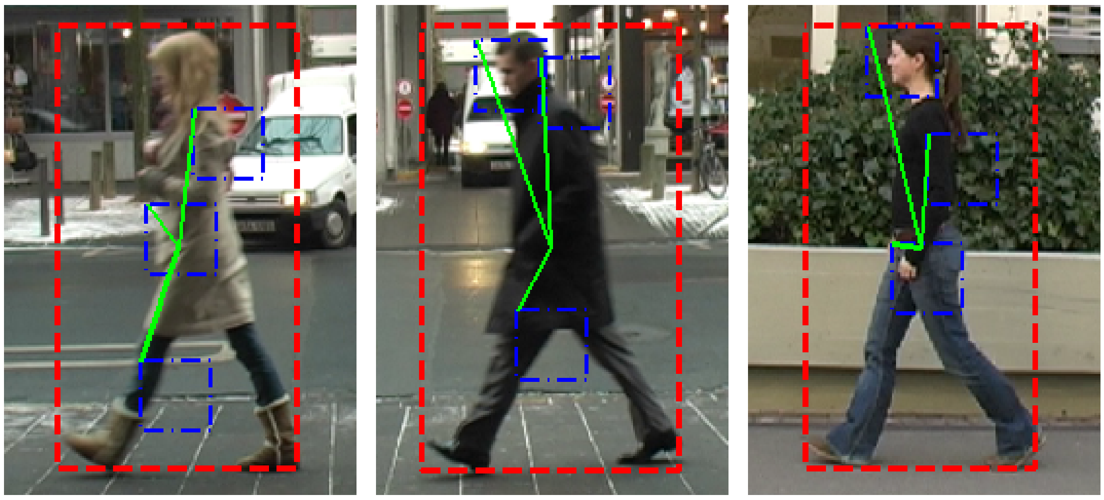
In this framework, a binary test can be applied to the appearance $\mathbf{I}_i$ of a sample and is defined by a channel $a \in (1, ..., C)$, two locations (p, q) and (r, s) in the 16x16 image, and a threshold value τ. The binary test can be written as

$$\xi(\mathbf{I}; a, p, q, r, s, \tau) = \begin{cases} 0, & \text{if } \mathbf{I}^a(p, q) < \mathbf{I}^a(r, s) + \tau \\ 1, & \text{otherwise} \end{cases} . \tag{3.9}$$

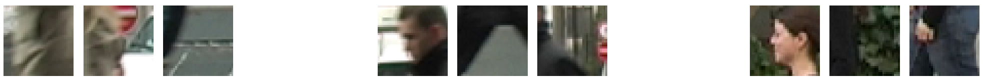
While training the Hough Forest, all nodes in each tree try to find the best binary test among a random subset of all binary tests. The trees are growing by following the same recursive procedure from the standard RF [15]. The main characteristic of Hough Forests is the way the quality of the binary tests is evaluated. On the one hand, Gall & Lempitsky [33] try to minimize the "class-label uncertainty", which is the same as in the standard Random Forest and follows equation 3.2. For purposes of notation, we rewrite this equation to $U_1(I_l, I_r) = -\Delta H$. On the other hand, a node in the Hough Forest also tries to minimize the "offset-uncertainty", which corresponds to the variance of the offset vectors $\mathbf{d}_i$ of positive patches and is defined as

$$U_2(I_l, I_r) = \sum_{\substack{i:c_i=1 \\ i \in I_l}} (\mathbf{d}_i - \mathbf{d}_{I_l})^2 + \sum_{\substack{i:c_i=1 \\ i \in I_r}} (\mathbf{d}_i - \mathbf{d}_{I_r})^2 , \tag{3.10}$$

where $\mathbf{d}_{I_l}$ and $\mathbf{d}_{I_r}$ are the mean offset vectors in the subsets I_l and I_r, respectively.



(a) Labeled images



(b) Extracted patches

Figure 3.1: In (a), one can see some training images used in the Hough Forest Framework. The red box is the bounding box of the target object, in this case a pedestrian from the TUD-pedestrian dataset [2]. Within the bounding box, small patches are extracted at random locations, which can be seen from the blue squares. All patches from positive training examples store an offset vector to the object centroid (green line). (b) shows the extracted 16x16 patches.

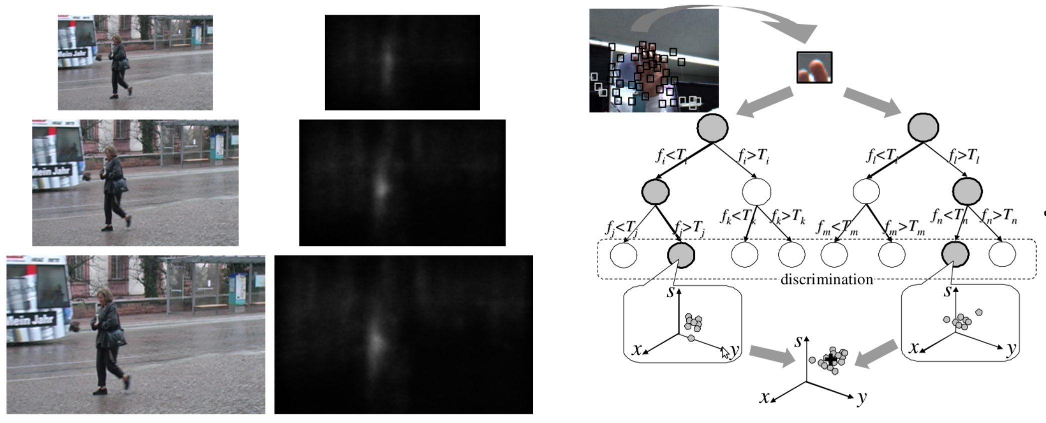
The decision, whether to minimize the “class-label uncertainty” or the “offset uncertainty”, is done randomly. The resulting optimization problem can be summarized to the following:

$$\underset{k}{argmin}\, U_* \left(\left\{ p_i | \xi^k(I_i) = 0 \right\}, \left\{ p_i | \xi^k(I_i) = 1 \right\} \right) , \tag{3.11}$$

where * is either 1 or 2, according to the random choice.

If a tree in the Hough Forest reaches a maximum depth or the number of training samples is too small, a leaf node is created, which stores the proportion of positive to negative patches P_i reaching this node. Additionally, a list of all offset vectors $\mathbf{d}_i$ from the positive patches is stored. Figure 3.2b illustrates the general training procedure. Please note, that the scale information in the leaf nodes, depicted in this figure, is not included in the Hough Forest framework. Scale information is introduced in the detection phase with a sliding window approach.

The detection procedure from [33] is based on the generalized hough transformation from Ballard [6]. In the first step of this process, 16x16 pixel patches $\mathbf{I}_i$ are extracted at each location $\mathbf{y}$ in the test image and provided to the Hough Forest, which returns leaf node statistics about the class label probability C_L and, in case $C_L > 0$, some offset vectors $\mathbf{d}_i$ stored in the current leaf node. These offset vectors point to possible object centers $\mathbf{x}$, relative to the current location $\mathbf{y}$. In the so-called Hough image, which has the same size as the test image, a score for the current patch $\mathbf{I}_i$ is added at location $\mathbf{x}$. The value of this score depends on the class label probability C_L, higher probabilities correspond to higher score values. In general one can say that each patch $\mathbf{I}_i$ casts a probabilistic vote for some specific location in the Hough image, where possible object centers can occur. These votes are accumulated and object centers can be estimated using a maximum search in the Hough image. Bounding boxes are assumed to be fixed in width W and height H. In practice, one can estimate the bounding box size of objects by taking the mean over the training examples. Scale invariance is introduced at prediction time by applying the trained classifier to scaled versions of the test images, yielding several hough images, one for each scale. The maximum search is then performed in the 3D-scale-space. Figure 3.2a illustrates such a 3D-scale-space of the hough images and its corresponding scaled images.



(a) Labeled images **(b)** Extracted patches

Figure 3.2: A 3D-scale-space of Hough images and its corresponding test images are shown in (a). An overview of the training procedure from [58] is illustrated in (b).

A more detailed discussion about the Hough Forest framework including experimental results can be found in the original paper from Gall & Lempitsky [33]. We have to mention, that there is a very similar work from Okada [58] which was simultaneously published with [33]. One difference to the Hough Forest framework is the optimization problem in a decision node. While Gall & Lempitsky [33] always minimize the "class-label uncertainty" and the "offset uncertainty" in different nodes, Okada [58] merges the optimization problems and steers the influence of the "offset uncertainty" with an adaptive parameter depending on the current depth of the tree.

3.3.2 Hough Forest Regularization

In this section, we want to show how to incorporate the motion data in the Hough Forest framework in order to perform object detection. As a first step, we have to provide the motion data in form of patches to be suitable for the Hough Forest framework. We denote a pair of patches as follows

$$\Phi_i = (\mathbf{I}_i^t, \mathbf{I}_i^{t+\Delta}, y_i) \, . \tag{3.12}$$

$\mathbf{I}_i^t$ and $\mathbf{I}_i^{t+\Delta}$ are the appearances of the same part of a moving object at frame t and $t+\Delta$, respectively. $\mathbf{I}_i^t$ is defined in the same way as described in section 3.3.1. To take advantage of the motion data, we regularize the standard Hough Forest optimization problem, denoted in Equation 3.11, with the pair error ε_{pair} from Equation 3.8. Note that we only consider the appearance of the motion data and, therefore, we only regularize those nodes which optimize "class-label uncertainty". The new "class-label uncertainty" optimization problem is given by

$$\underset{k}{argmin}\, U_1\left(\left\{p_i | \xi^k(I_i) = 0\right\}, \left\{p_i | \xi^k(I_i) = 1\right\}\right) + \lambda \cdot \varepsilon_{pair} \tag{3.13}$$

where ε_{pair} is the error on the pair-wise motion data as defined in Equation 3.8 and λ is a parameter to control the influence of the motion data regularization.

To sum up, this regularization penalizes the randomly chosen binary test ξ^k in a node of the Hough Forest, if it performs bad on the pair-wise motion data, *i.e.*, if it splits up "same" labeled pairs or keeps "different" labeled pairs together, as described in Section 3.2. As a result, we expect better generalization performance in object detection tasks, as we now inherently learn how to cope with local object transformations and illumination changes.

3.3.3 Training a Codebook

The previous section described how to regularize the Hough Forest for the object detection task. Now, we want to present our methods used for object tracking. As already mentioned, Gall *et al.* [34] showed how to online update the Hough Forest, which is necessary for tracking. The basic idea there is to train a codebook with a standard Hough Forest on class-specific training data in an offline manner, *i.e.*, in the same way as it is done for the object detection task. Like shown in [34], one can calculate the probability of an object from class c at position $\mathbf{x}$, given the leaf statistics for a patch extracted at location $\mathbf{y}$ in the image, with

$$p(E(\mathbf{x})|L(\mathbf{y})) = \frac{1}{|P_{L(\mathbf{y})}|}\left(\sum_{P_i \in P_{L(\mathbf{y})}} p(c=1|L(\mathbf{y})) \cdot \delta_{\mathbf{d}_i}(\mathbf{y}-\mathbf{x})\right) . \tag{3.14}$$

When it comes to the task of tracking a single instance of a class c, *e.g.*, one specific person from the class "pedestrian", one is interested in the probability of the evidence of a given instance I. This can be written as

$$\begin{aligned} &p(E_I(\mathbf{x})|L(\mathbf{y})) = \\ &\quad \frac{1}{|P_{L(\mathbf{y})}|}\left(\sum_{P_i \in P_{L(\mathbf{y})}} p(P_i \in I|c=1, L(\mathbf{y})) \cdot p(c=1|L(\mathbf{y})) \cdot \delta_{\mathbf{d}_i}(\mathbf{y}-\mathbf{x})\right) . \end{aligned} \tag{3.15}$$

In order to make the codebook instance-specific, the probability $p(P_i \in I|c=1, L(\mathbf{y}))$ has to be estimated. This can be done by counting how often a patch P_i votes for the target object $\{\mathbf{y}|P_i \in I \cap P_{L(\mathbf{y})}\}$ and how often for other objects $\{\mathbf{y}|P_i \notin I \cap P_{L(\mathbf{y})}\}$. Based on a clustering of the Hough space, which finds confident regions for the target object, the number of votes for and against the target object can be counted. In other words, they search patches P_i voting for the target object and patches voting for background. These patches are used for updating the codebook and estimating the desired probability, given as

$$p(P_i \in I|c=1, L(\mathbf{y})) = \begin{cases} 0.5, & \text{if } \left|\{\mathbf{y}|P_i \in I \cap P_{L(\mathbf{y})}\}\right| + \\ & \quad \left|\{\mathbf{y}|P_i \notin I \cap P_{L(\mathbf{y})}\}\right| = 0 \\[2ex] \frac{\left|\{\mathbf{y}|P_i \in I \cap P_{L(\mathbf{y})}\}\right|}{\left|\{\mathbf{y}|P_i \in I \cap P_{L(\mathbf{y})}\}\right| + \left|\{\mathbf{y}|P_i \notin I \cap P_{L(\mathbf{y})}\}\right|}, & \text{otherwise} \end{cases} . \tag{3.16}$$

If a patch P_i was never used for voting, the probability for the patch belonging to the instance I is set to 0.5. Otherwise, it is the fraction between the number of votes for the instance and all votes given by this patch. Note, this update procedure neither adds new patches to the leaves of the trees, nor updates the offset votes $\mathbf{d}$ of the leaf statistics. It just updates the weights of the votes for the instance I. A more detailed discussion about this update process and state-of-the-art results are presented in [34]. Nevertheless, the pre-trained forest is limited to a specific object class, which might be a problem in many applications.

To get rid of this limitation, we propose a new method by training a general, non-class-specific codebook solely on motion data. This procedure is exactly adapted from section 3.2 to the Hough Forest framework, where the only difference in notation is the form of the input data, which is now patch-based. The reformulation was already stated in the previous section. To review, we try to minimize the pair error ε_{pair} on the motion data and leaves are created similar to RFs, *i.e.*, if a maximum depth is reached or a node is pure. Note that in this scenario, pure means that only pairs are left which are labeled "same" or "different", respectively. No leaf statistics about any target classes are available after training the codebook, which makes our tracking approach capable of tracking arbitrary objects in any given video sequences, as opposed to the tracking framework presented in [34].

As we are lacking any leaf node statistics, we have to update the trees with instance-specific information from the target object. To this end, we initialize the forest from the first frame of a given tracking sequence with positive patches including offset vectors and random negative patches. Additionally, we generate a set of virtually warped patches to increase the amount of data from the first frame. These patches fall into leaves of the general codebook, where we can build an instance-specific statistic and store the offset vectors $\mathbf{d}_i$. Note that this differs from [34], as we now have to update patches P_i and offset object votes $\mathbf{d}_i$. Furthermore, we do not need the update procedure to yield an instance specific codebook, as described above (see Equation 3.16), because we do not have to separate between the object class and the instance class. We do not assume any prior object class and, thus, we are inherently instance-specific as soon as we update the general codebook. During tracking, we further collect patches in frames with confident predictions to update the trees for handling illumination and pose changes of the target object.

3.4 Chapter summary

In this chapter, we first reviewed the Random Forest framework from [15] as a base to incorporate motion data for the tasks of object detection and tracking. In Section 3.2, we defined the pair error, which can measure the quality of a split in a node from a Random Forest when it is given pair-wise motion data. Section 3.3 describes how we can extend a state-of-the-art detection framework [33, 58] with our ideas. First, we show how to regularize the “class-label uncertainty” optimization with pair-wise motion data for object detection and, second, we depict how to train a general codebook for object tracking solely on motion data.

Chapter 4

Learning Framework and Motion Extraction

Contents

4.1 Processing pipeline 41

4.2 Extraction of motion data 42

4.3 Chapter Summary 49

In the last chapter, we showed how we can use motion data to train a classifier, which is more invariant to local object transformations and illumination changes. Therefore, we used pair-wise motion data in form of patches that are suitable for the Hough Forest framework [33, 58]. This chapter gives an overview of the whole detection and tracking framework, including data acquisition, training a classifier, and evaluation phase, see Section 4.1. Furthermore, we show how we extract the pair-wise motion data patches from video sequences using optical flow and a motion-based segmentation in Section 4.2.

4.1 Processing pipeline

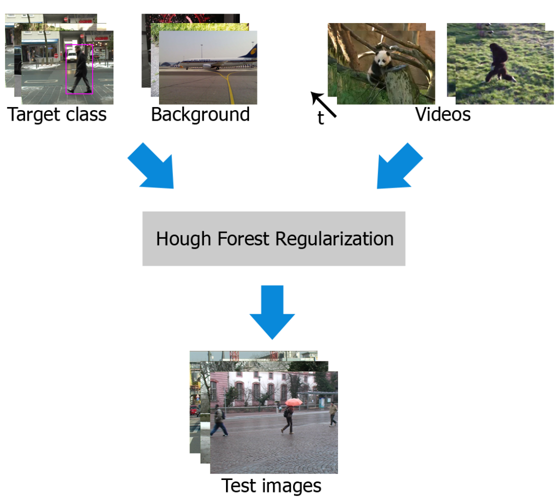
In this section, we describe the framework for both detection and tracking tasks in more detail. A pipeline starting from data acquisition and ending at the evaluation process is illustrated in Figure 4.1. We used the publicly available code from [33] and extended it to incorporate our methods. In the following, we detail our procedures for both tasks.

Detection: For the task of object detection, we obviously need labeled training images, comprising labeled target-class objects, and background images. Additionally, we now need pair-wise motion data extracted from video sequences to regularize the Hough Forest framework, like described in Chapter 3. The acquisition of the motion data is described exactly in Section 4.2. In a second step, the classifier can be trained using all the training images, as described in Section 3.3.2. When training is finished, the classifier is evaluated on test images, which are typically picked from the same source as the labeled training data. We present the test images to our classifier and evaluate the results. The evaluation process is described in more detail in Chapter 5. An overview of the detection pipeline is illustrated in Figure 4.1a.

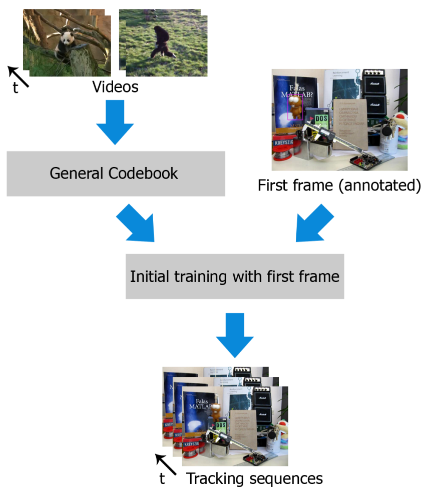
Tracking: For the tracking task, no labeled training data is required in the first place. Our approach trains a general codebook solely on unlabeled motion data, as described in Section 3.3.3. After training the codebook, which lacks any leaf statistics about a target instance, it is updated online from a video sequence and evaluated at the same time. In other words, the tracking algorithm is initialized with patches from the first frame of the video sequence, containing the labeled target instance. Afterwards, the classifier is updated online and evaluated with the ground truth during the tracking phase. Figure 4.1b shows a simple overview of this procedure. Again, the evaluation process is described in more detail in Chapter 5.

4.2 Extraction of motion data

In this section, we describe the process of extracting pair-wise motion data, which are suitable for our framework, out of video sequences $\mathcal{V}_j$. As already mentioned before, the objects captured from the video sequences does not have to coincide with the target class of the object detector or tracker, as we only extract parts of the moving objects, which can be shared over different target classes. But in order to extract helpful information from the sequences, we have to set a constraint that moving objects are shown prominently in each frame of the sequences.



(a) Detection

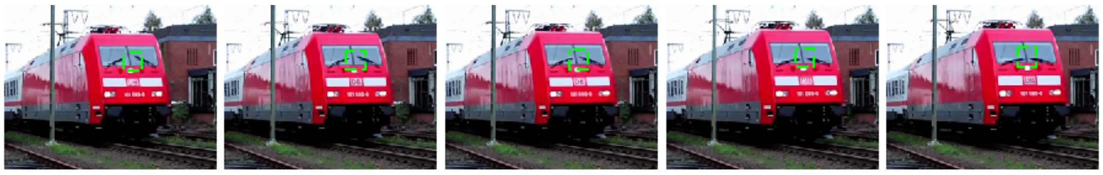


(b) Tracking

Figure 4.1: This figure gives a rough overview of the processing pipelines for both, detection (a) and tracking (b) tasks.

Videos, as a source for collecting object transformations without any annotation work, have a very important advantage: the motion itself. Moving objects can be separated from the static background due to the motion. Furthermore, we are only interested in local transformations of natural objects, which can be found especially on moving objects. Thus, extracting motion data from video sequences fits exactly our needs.

In the following, we define the goal of this processing step in our framework, *i.e.*, the extraction of motion data. Assume, we are given a set of N video sequences $\mathcal{V}_j$, each consisting of N_j frames F_j^t. Now, we have to build a database of pairs Φ_i^j containing two corresponding 16x16 image patch appearances $\mathbf{I}_i^t$ and $\mathbf{I}_i^{t+\Delta}$, where each patch appearance has the same dimension as the Hough Forest input data. Like described in Section 3.2, a pair Φ_i^j is labeled as "same" if its two patches $\mathbf{I}_i^{\{t,t+\Delta\}}$ characterize the same local area of a moving object in a video sequence $\mathcal{V}_j$ at different time instances t and $t+\Delta$, respectively. Pairs labeled as "different" are build artificially by combining two patches extracted from two video sequence $\mathcal{V}_k$ and $\mathcal{V}_l$ $(k \neq l)$ containing different objects. A subset of frames from two different video sequences is depicted in Figure 4.2, where each frame contains a patch describing the same local area of the moving object (green square).


(a) Train

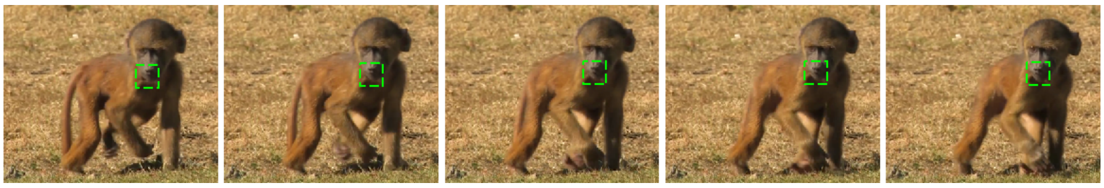

(b) Monkey

Figure 4.2: Two different video sequences, (a) and (b), where one can see a patch (blue square) describing the same local area of a moving object over all frames.

Building such a database of pairs Φ_i^j requires to extract motion from the video sequences $\mathcal{V}_j$, which can be achieved using a dense optical flow, described in the following section. Furthermore, with a motion-based segmentation approach, we can find the boundaries of the most prominent moving object in the video frames, see Section 4.2.2. An overview of the processing pipeline for extracting motion data is illustrated in Figure 4.3.

4.2.1 Optical Flow

The first step for gathering useful information of moving objects from a video sequence $\mathcal{V}_i$, is to estimate the motion between two subsequent frames F_i^t and F_i^{t+1}. Note that we do not have any a priori knowledge about the moving object, like location, pose, or class label. Thus, we use a dense optical flow to extract the motion.

Optical flow estimation between two temporal coherent images is one of the key problems in computer vision and was encouraged by two famous papers from Lucas & Kanade [53] and Horn & Schunk [38]. There exist a vast amount of approaches to optical flow estimation, including global and local methods, gradient-based, phase-based, or energy-based methods. We refer to the surveys from [4, 7, 31] as a review of many different approaches. We chose the anisotropic Huber-L^1 optical flow from Werlberger *et al.* [84], as the implementation is publicly available and the method gives good and robust results. Furthermore, the performance in speed is also impressive as it is a GPU

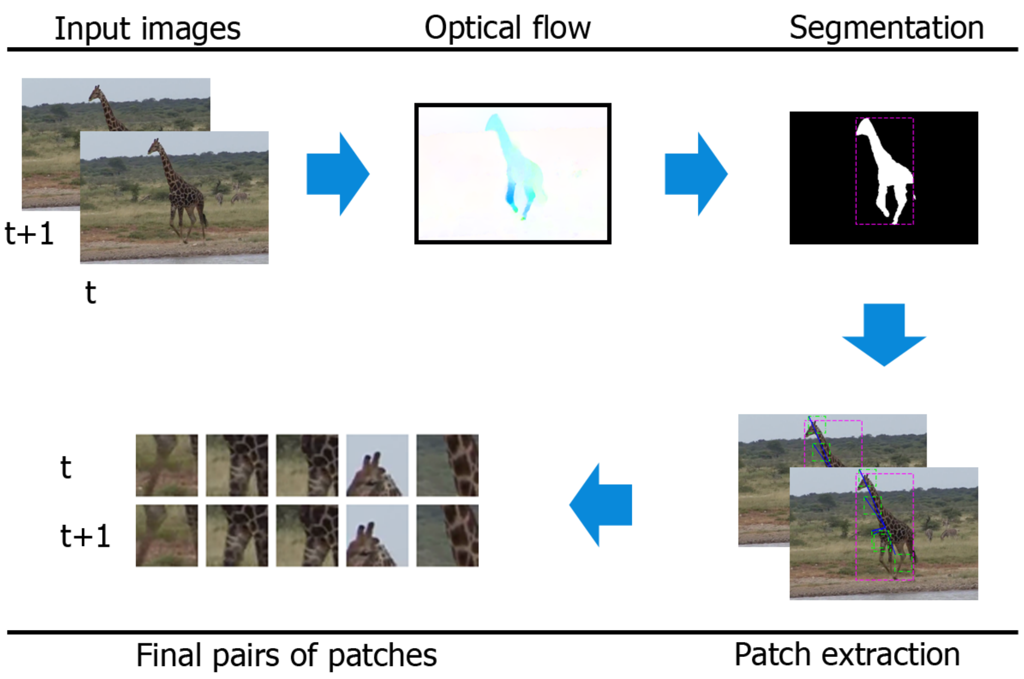


Figure 4.3: A schematic overview of the process of extracting pair-wise motion data out of video sequences using optical flow and a motion-based segmentation technique.

implementation. It takes about 4 sec to estimate the optical flow on a *NVIDIA GeForce 320M* mobile GPU. Please note, using a high-end GPU the time for calculation would decrease vastly.

The optical flow from Werlberger *et al.* [84] is a global method and extends an approach based on an L^1-norm data term and an isotropic Total Variation (TV) regularization, like described in [94]. Werlberger *et al.* [84] replace the TV-regularization with an anisotropic (image-driven) Huber regularization, which can better cope with the so-called “stair-casing” problem, *i.e.*, sparsely textured areas yielding piecewise constant solutions in the optical flow estimation.

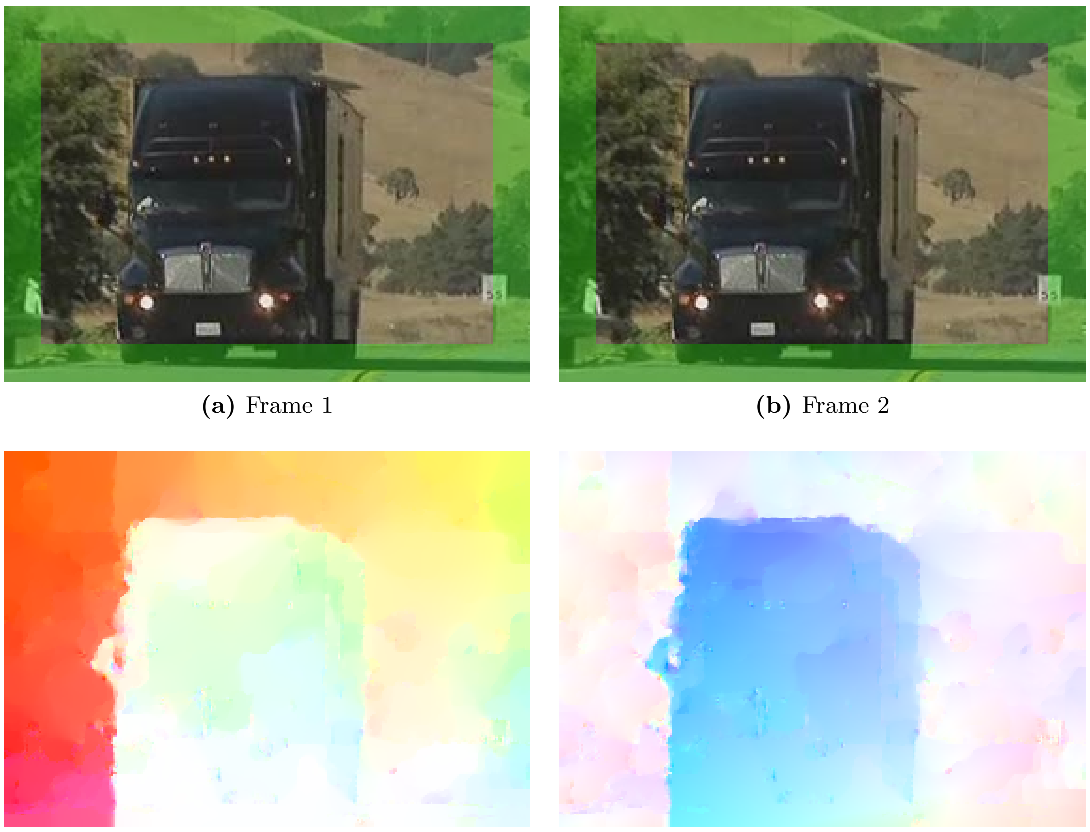
Figure 4.4c shows an example of the optical flow between two consecutive frames. The 2D-velocity vectors of the optical flow are color-coded, where the color itself describes the direction and the intensity the length of the vectors.

One problem, which often occurs in videos that capture moving objects, is the camera movement. On the one hand it provides a good framing of the dominant object, but on the other hand, without treating this effect, the optical flow calculation often fails. Figure 4.4c shows that the actual object, which should be extracted, shows less motion than the background, as can be seen in the intensity of the colors. Thus, one has to subtract the background motion, similar to [91]. The idea there is to use a small border of the input frames (*e.g.*, 10% of the image height and width, respectively) to estimate the background motion and subtract it from the actual optical flow. These border regions are highlighted green in Figures 4.4a and 4.4b. In contrast to [91] we estimate a fully-affine camera model using a RANSAC approach, instead of reducing the camera motion to translational movements. The resulting, background subtracted optical flow is presented in Figure 4.4d, where one can clearly see that the moving object has the largest motion, *i.e.*, the highest intensity. Note, due to the background motion subtraction, we added a further constraint on the video sequences $\mathcal{V}_i$, and that is the presence of a thin background border around the moving object in each frame.

4.2.2 Motion-based Segmentation

The optical flow image (including background motion subtraction) described in the last section, see Figure 4.4d or 4.5a for an example, serves as the input for the next unit in our processing pipeline, the motion-based segmentation of the moving object. A lot of research was investigated in the task of segmenting moving objects in videos and a lot of different methods were proposed. Zhang & Lu [95] give a good overview of several existing approaches. More examples of motion-based segmentation can be found in [39] or [45].

Our method first clusters a random subset of the optical flow field into 2 parts using k-Means. The resulting cluster centers describe the two mean motion vectors in the optical flow field, optimally corresponding to fore- and background motion. As a preprocessing step, these mean motion vectors are subtracted from the input optical flow field, yielding a better partitioning between fore- and background. An example image after this preprocessing step is depicted in Figure 4.5b. Then, we applied a simple Potts-based model [62] to the preprocessed optical flow in order to segment the image. The Potts-model was introduced in computer vision tasks as a special case of the Mumford-Shah model [57] with the goal to partition the image in N regions. An energy functional, consisting of a regularization term and a data term, is minimized. The regularization term is responsible for spatially coherent regions and the data term



(a) Frame 1 **(b)** Frame 2

(c) Optical flow **(d)** Background subtracted optical flow

Figure 4.4: Here one can see some figures from the calculation of the optical flow. (a) and (b) show two consecutive frames from a video sequence including the border used for estimating the background motion. The optical flow calculated with the Huber-L^1 method [84] is illustrated in (c). Furthermore, (d) shows the final background subtracted optical flow.

for data fidelity. The segmented optical flow image can then be thresholded yielding a binary image, which contains a separation between fore- and background, see Figure 4.5c. The largest connected region in the image is assumed to be the moving object and all other regions are neglected for further processing. Thus, one can draw a bounding box around the moving object according to the largest region in the segmented binary image. All segmentation steps are summarized in Figure 4.5.

After segmenting the moving object, we can extract small patches, appropriate for the input format of our extended Hough Forest framework. Of course, all segmented objects have to be resized to a common size which fits the framework, *i.e.*, a common height or width which is appropriate to extract useful 16x16 patches. Typically, we

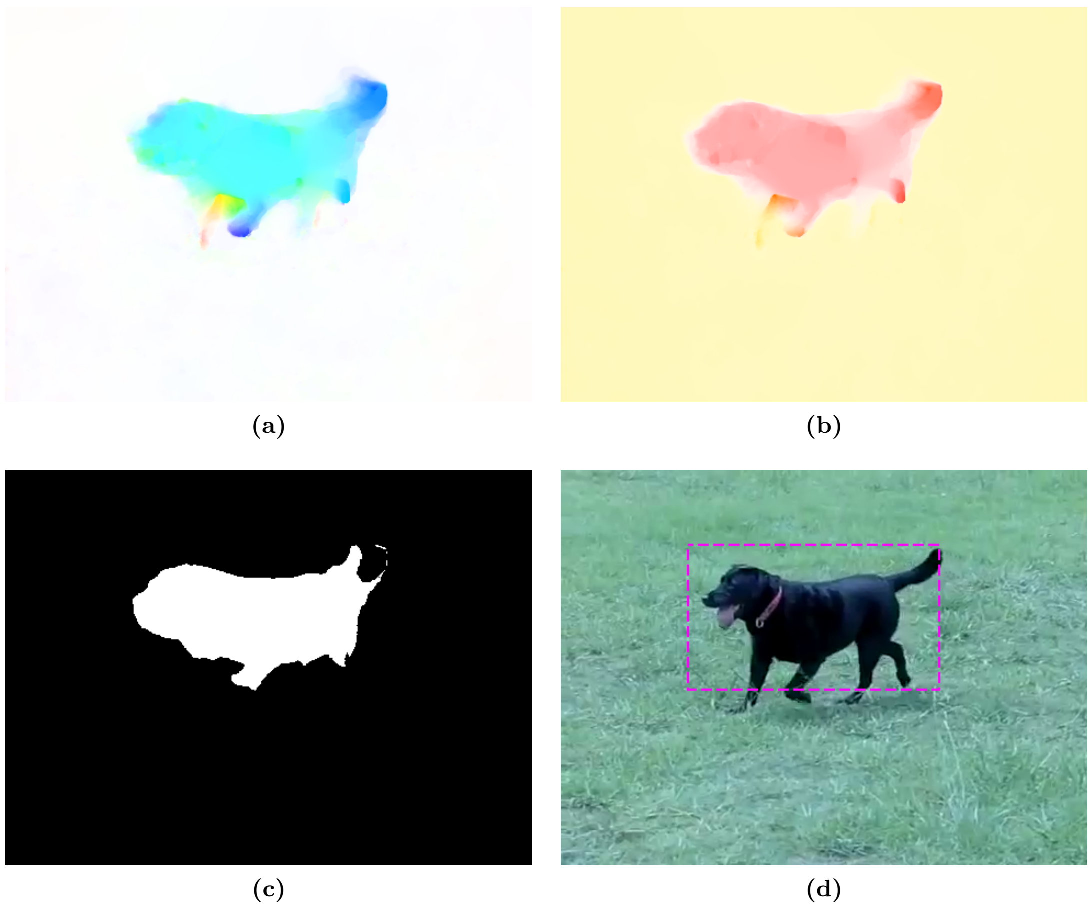
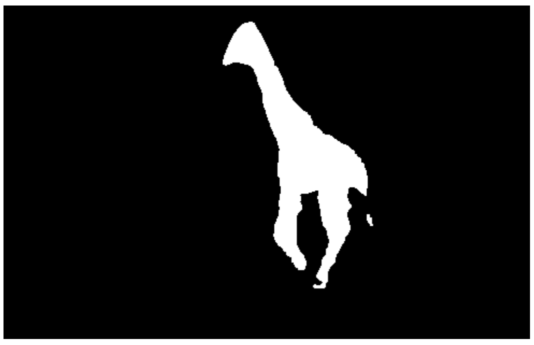
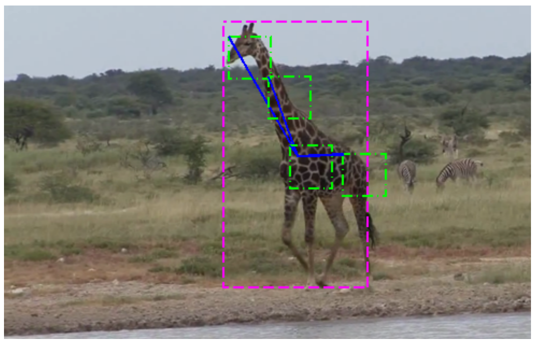


Figure 4.5: Motion-based segmentation pipeline: (a) shows a background motion subtracted optical flow field, which serves as input for the segmentation pipeline. The mean subtracted flow is illustrated in (b). (c) shows the binary images after the segmentation using the Potts-model and (d) illustrates the original input image (first frame) with the corresponding bounding box around the moving object.

chose a height or width of 100 px for the objects. Thanks to the segmentation process, we have the boundaries of the moving target object and, thus, we can locate all patches exactly on the object, which reduces the amount of noise produced during this phase. In order to build pairs of patches Φ_i^j, we have to track them from frame to frame using the calculated optical flow. After t^Δ frames we extract the corresponding second patch of the pair at the tracked location. Figure 4.6 illustrates the extraction of some patches.


(a) Binary image


(b) Frame 1

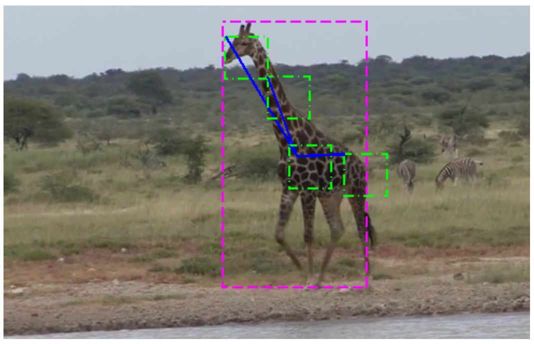

(c) Frame 2

Figure 4.6: This figure illustrates the extraction of patches from two successive frames. (a) shows the binary image after segmentation. Within the white area, which describes the moving object, the patches can be positioned. (b) and (c) show two successive frames with the corresponding extracted patches. The little green squares denote the patches.

4.3 Chapter Summary

This chapter first summarizes our processing pipeline for a detection and tracking framework in Section 4.1. Furthermore, Section 4.2 describes exactly how we extract the pair-wise motion data used for the Hough Forest regularization and the training of a general codebook. This procedure includes a dense optical flow, see Section 4.2.1, and a motion-based segmentation method, which is reviewed in Section 4.2.2.

Chapter 5

Experiments

Contents

5.1 Video database **51**

5.2 Object Detection **53**

5.3 Tracking **63**

5.4 Chapter Summary **68**

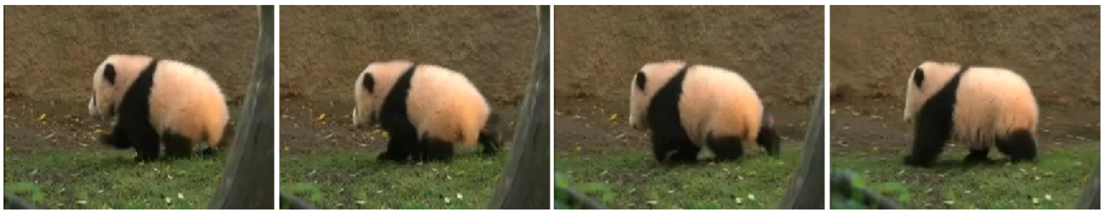
In this chapter, we describe all experiments, which were done to evaluate the performance of our proposed methods for both detection and tracking tasks. As already explained in Chapter 3, we extended the Hough Forest framework for our purposes, which is the reason we also used the publicly available code from [33] for implementation. As a first step in the experimental part of this thesis, we needed a database of videos for extracting motion data. This collection is detailed in the following section. Thereafter, we discuss the experimental setups and results of the detection task in Section 5.2 and of the tracking task in Section 5.3, respectively.

5.1 Video database

For the purpose of learning transformation invariant object classifiers, we need a database of video sequences containing moving objects, which show several natural variations and transformations. Amongst others, these transformations may include shape, appearance, or illumination changes. Unfortunately, we did not find a suitable database satisfying all our requirements in the current literature. To this end, we used a web-based search engine to find publicly available video sequences to build our own motion database. The video sequences contain a lot of different animals, *e.g.*, panda bears, tigers, or dogs. Furthermore, we collected video sequences capturing trains, cars, and trucks. Figure 5.1 illustrates a subset of the database.

For the practical use in our framework, pair-wise motion patches, containing local transformations of moving objects, had to be extracted from the sequences. This can be achieved by using a segmentation technique in combination with optical flow, which is described in Section 4.2. Remember, this procedure constrains the video sequences in a way that they have to capture the moving object prominently, while still holding a certain background clutter in the boundaries of all frames in the sequences. The frames depicted in Figure 5.1, fulfill these two constraints.

(a) Lion sequence


(b) Panda bear sequence

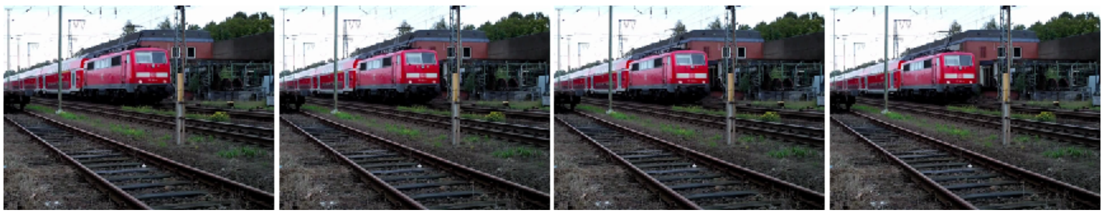

(c) Train sequence

Figure 5.1: Here one can see several frames from the “moving-objects”-database in 3 different sequences, which contain a lion (a), a panda bear (b), and a train (c).

5.2 Object Detection

To get experimental results in the task of object detection, we evaluated our methods proposed in Chapter 3 on three different datasets, which are described in Section 5.2.1. As a base object detector, we always used the standard Hough Forest framework [33] and we tried to show improvements by incorporating motion data. The number of trees in the forest was limited to 10 and the maximum depth of each tree was set to 15. We always used around 20000 labeled training patches, equally partitioned to the positive and negative classes. To obtain a set of labeled patches with approximately the same size for each dataset, the number of patches extracted from one training image was chosen according to the number of available images in each dataset. The results presented here are always an average over 5 independent runs per experiment.

We used the PASCAL overlap criteria [28] for measuring the performance of the trained classifiers on the test images. All results are presented using a precision-recall (PR) curve and the estimated average precision (AP) value. Each predicted bounding box B_p is compared with the ground truth bounding boxes B_{gt} on the test dataset by calculating the overlap ratio a_o, which is given as

$$a_o = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \; . \tag{5.1}$$

If this overlap ratio a_o exceeds a certain threshold, 0.5 in our case, the detected bounding box is defined as *true positive*. Furthermore, *false positive* and *false negative* detections are counted for the calculation of the precision and recall values, which are defined as follows

$$precision = \frac{|true\ positive|}{|true\ positive| + |false\ positive|} \tag{5.2}$$

$$recall = \frac{|true\ positive|}{|true\ positive| + |false\ negative|} \; . \tag{5.3}$$

The AP value is calculated according to [28] and is given by

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} p_{interp}(r) \; , \tag{5.4}$$

where $p_{interp}(r) = \max_{\tilde{r}:\tilde{r} \geq r} p(\tilde{r})$ and $p(\tilde{r})$ is the measured precision at recall $\tilde{r}$.

In the following section, we describe the three datasets used for the evaluation and, thereafter, we present our results on various experiments in Sections 5.2.2 to 5.2.4.

5.2.1 Datasets

As mentioned above, for evaluating our methods we used three publicly available datasets, a car dataset from [47], a pedestrian dataset from [2] and, finally, a horse dataset from [12]. In the following, we briefly discuss these datasets.

ETHZ-cars: This multi-view car dataset from [47] contains images capturing cars viewed from 7 different angles. The dataset was split up to build a training set of 700 cropped samples and 175 test samples. The image collection involves a lot of illumination changes and slight occlusions.

TUD-pedestrian: The dataset consists of 400 positive labeled training images showing pedestrians walking on the street from a side view. 250 test images are provided showing pedestrians from almost side-views with a certain amount of background clutter. The test images were cropped to reduce the time of the testing phase. But we left enough background clutter, whereby the task of object detection, *i.e.*, classification and localization of objects, did not simply reduce to classification only. In Figure 5.2b, we show some images of the original testset including the highlighted border, which was cut away.

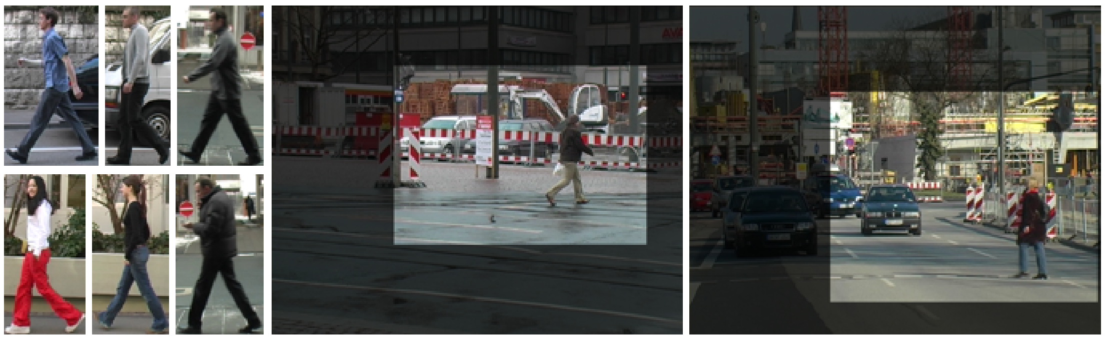
Weizmann-horses: This dataset contains 328 images from different types of horses, all looking leftwards. In order to make the detection task a bit more difficult, we randomly flipped some images, making the horses also look to the right. Furthermore, we split up the dataset in a training set with 200 cropped images and a testset containing the remaining 128 examples.

Figure 5.2 shows a collection of training and testing images from the three different datasets.

5.2.2 Performance improvement

Before going into more detail about all different parameters in the framework and the properties of the motion data, we want to show the general improvement on all

(a) ETHZ-cars [47]


(b) TUD-pedestrian [2]

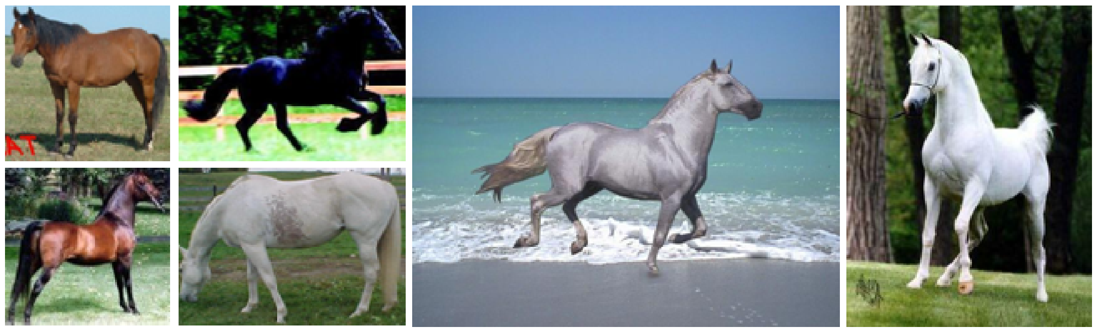
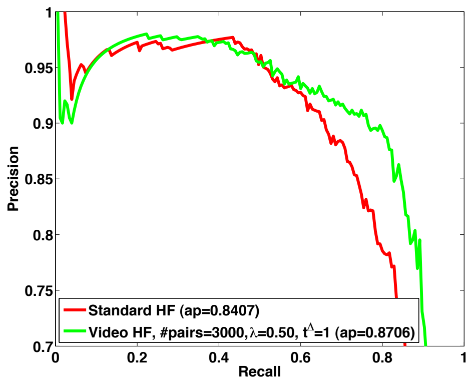
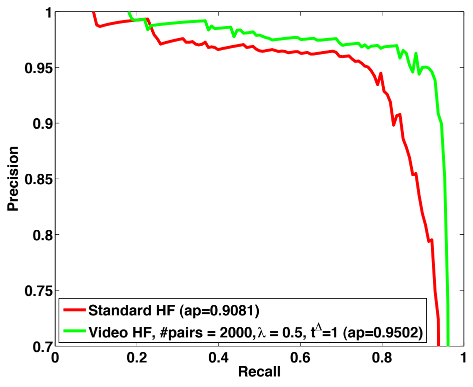

(c) Weizmann-horses [12]

Figure 5.2: Figures (a), (b), and (c) illustrate some train and test images from the 3 datasets used for the object detection experiments. Training images are depicted on the left side, testing images on the right. In (b) one can see the reduced testing images from the TUD-pedestrian dataset.

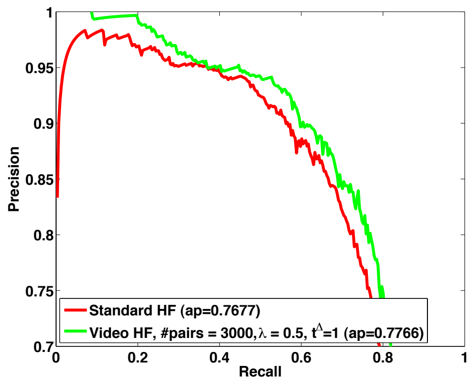
three datasets in this section. Thus, we compared the standard Hough Forest with our proposed method. The number of motion pairs used and the parameter λ were set differently for each dataset to yield good performances. Figure 5.3 shows the results on all three datasets including the actual values of the parameters for each dataset, respectively. The red curve shows the base approach, *i.e.*, the standard Hough Forest, and our proposed method is depicted with the green curve. As can be seen, we outperform the standard Hough Forest in each of the datasets. Especially, the recall values can be improved with our method, but also the AP values reflect the performance boost. As motion data, we used sequences from animals for the TUD-pedestrian and the Weizmann-horses datasets. Sequences capturing trains were used for the ETHZ-cars datasets.



(a) ETHZ-car



(b) Weizmann-horse



(c) TUD-pedestrian

Figure 5.3: PR curves for the three different datasets. The parameters used in the experiments can be seen in the figures.

5.2.3 Influence of parameters

In this section, we want to describe the influence of some parameters in our proposed method. First of all, we explored the effect of the parameter λ, which steers the importance between labeled data and unlabeled, pair-wise motion data, see Equation 3.13. Furthermore, we investigated the influence of different amounts of pair-wise motion data. Finally, a third experiment examined the effect of the parameter t^{Δ}, which is the number of frames between two patches in a pair of the motion database and inherently controls the level of transformation or variation, a moving object has undergone.

For the first experiment, we evaluated our approach for 5 different values of $\lambda \in \{0.1, 0.25, 0.5, 0.75, 0.9\}$ on the ETHZ-cars dataset. The results are depicted in Figure 5.5a. Again, the base approach is drawn with a red curve and the other colors correspond to different values of λ. We regularized our forest with 1500 pairs extracted from sequences containing trains. As can be seen in the figure, we receive higher recall values if we put more emphasis on the motion data, *i.e.*, for higher values of λ. On the other hand, we can perceive a drop in precision at low recall values for a high λ parameter. That is, when emphasizing the motion data in the regularization, we train a more invariant classifier which can detect more cars in the test images. Thus, we get high recall values. But this comes with the price of additional false positives with high confidence, as we, apparently, confuse the classifier in a sense (drop in precision at low recall values). This drop does not seem to influence the AP values, as they have their maximum at the highest value of λ. But this stems from the calculation of the AP, which tolerates such sinks, as it just takes the maximum value of the precision in a given interval, see Equation 5.4.

In a second experiment, we showed the influence of the amount of pair-wise motion data used for regularization. To this end, we took subsets of the motion data, *i.e.*, $\#_{pairs} \in \{1500, 3000, 4000, 5000\}$, and regularized the Hough Forest. The resulting curves are shown in Figure 5.5b. As can be seen, the amount of pair-wise motion data has to be selected well. Too less or too much data is counterproductive.

The third experiment was about the influence of the parameter t^{Δ}, *i.e.*, the number of frames between two patches in a pair, or, in a more practical way, the level of transformation between two corresponding patches. As an example, we depict some pairs with different values of the parameter t^{Δ} in Figure 5.4. For this experiment, we used the Weizmann-horses dataset. Figure 5.5c illustrates the result, which indicates that the level of transformation of the pair-wise motion data has to be kept low. But one has to keep in mind that this parameter does not exactly controls the level of transformation uniformly over all moving objects. It just tells how many frames were left out when extracting two corresponding patches in a video sequence. But videos can have different FPS values and, especially, objects can move fast or slow.

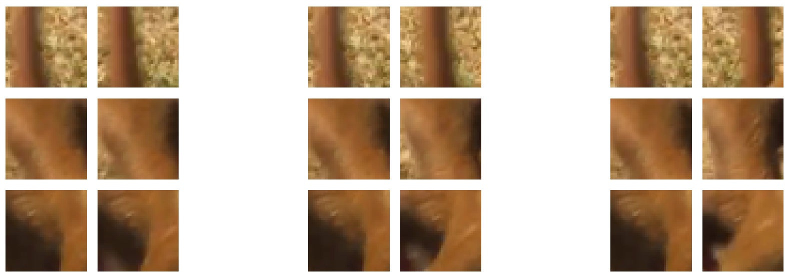
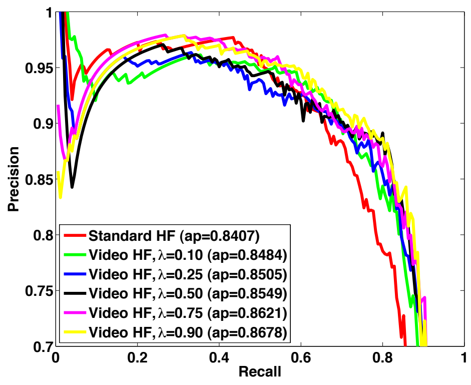

(a) $t^{\Delta} = 1$ **(b)** $t^{\Delta} = 2$ **(c)** $t^{\Delta} = 3$

Figure 5.4: In this figure, the influence of the parameter t^{Δ} is illustrated. (a), (b), and (c) show the same three pairs, but with different values for the parameter. That is, the first column is always the same, but the second one illustrates the different levels of transformation.

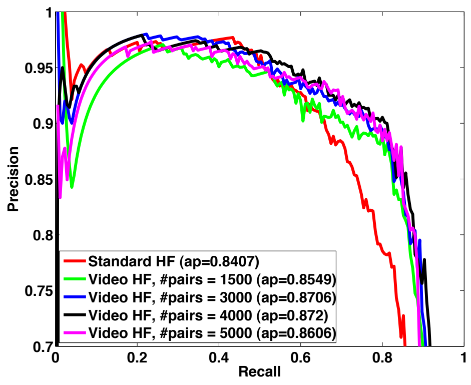
5.2.4 Influence of motion data

In this section, we want to show some experiments about the content of the motion data. First, we investigate the degree of weak-relation of the pair-wise motion data, which can be used to regularize the Hough Forest. Then, we examine if the motion data extracted from real-world videos can be replaced by virtually warped images, *i.e.*, simulated motion. In a last experiment, we use the motion data as additional labeled data and explore its influence on the results, especially, when using only weakly-related data.

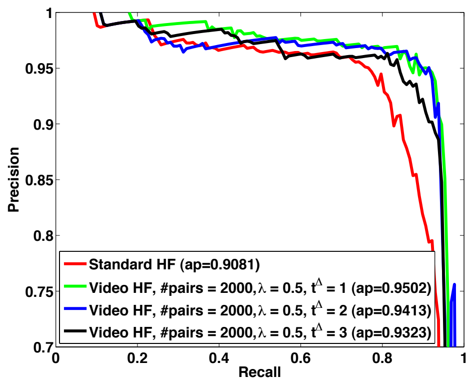
For the first experiment, we built three different sets of motion data. The first two were extracted from videos capturing trains and animals, respectively. The third one was created by using random, unlabeled background images, where we cropped some 16x16 patches and warped them slightly to simulate motion, *i.e.*, a local transformation to build "same" pairs. An example of such an unlabeled random image and its warped version can be seen in Figures 5.6a and 5.6b. "Different" pairs were created by randomly combining two patches. Using the ETHZ-cars dataset, we compared the standard



(a) Influence of λ



(b) Influence of # pairs



(c) Influence of t^{Δ}

Figure 5.5: Results on the ETHZ-cars dataset for different values of λ in (a) and # pairs in (b). The influence of the parameter t^{Δ} was evaluated on the Weizmann-horses dataset (c).

Hough Forest with three version of our method, each regularized with one of these sets of motion data. Of course, the parameters of the framework were fixed during all these experiments. The resulting PR-curves are illustrated in Figure 5.7a. We simply used a few motion pairs to speedup the training and testing phases, thus, the AP values do not show a big improvement here. Nevertheless, one can clearly see that the content of motion data matters when regularizing the Hough Forest. The more the motion data is related to the target class, the better results can be achieved. Simulated random motion even deteriorates the performance compared to the standard Hough Forest.

Another experiment showed the support from pair-wise motion data, when the labeled training dataset was increased by virtually warped positive labeled data. Figures

5.6c and 5.6d show an example of a positive labeled image and its virtually warped version. In this case, the classifier should yield better results, as we now use more training data, which simulate motion in a way. Thus, the classifier gets more invariant to local transformations due to the extended labeled dataset. For this experiment, we also used the ETHZ-cars dataset. Now, we want to know if the additional regularization with pair-wise motion data from natural objects (in this case trains) still supports the classifier. To this end, we trained a standard Hough Forest with the increased training set. Additionally, we provided the same extended labeled dataset to our method and added some motion pairs from videos containing trains for regularization. The results are depicted in Figure 5.7b and they show that we can still improve the generalization power by adding motion data, although we already extended the labeled dataset by virtually warped images.

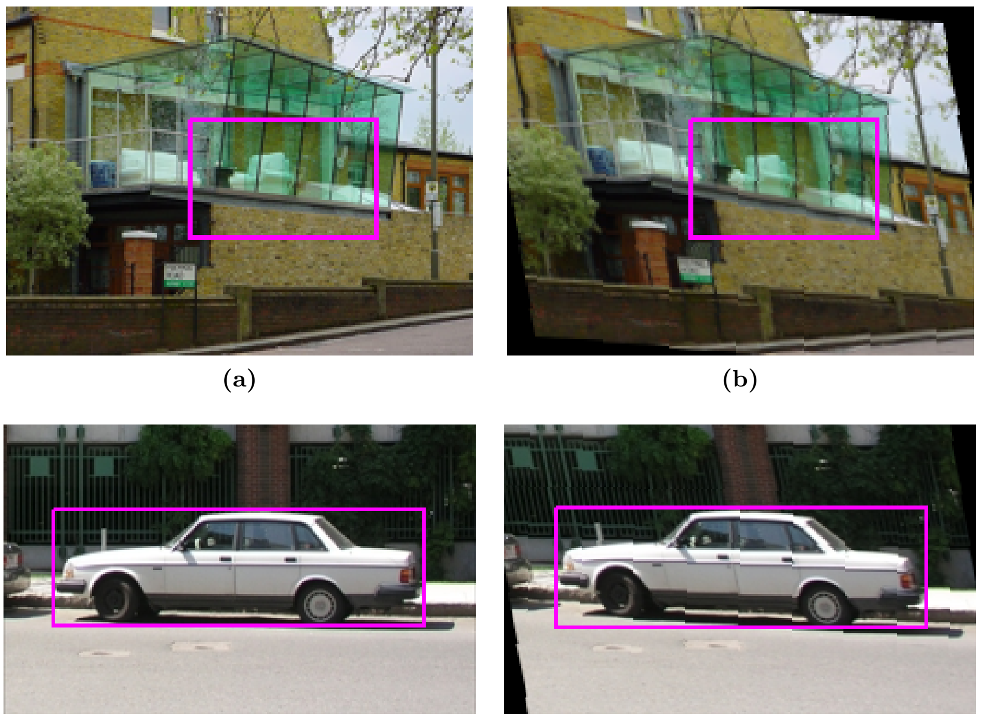
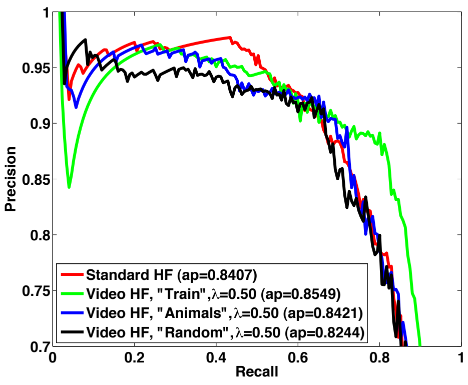
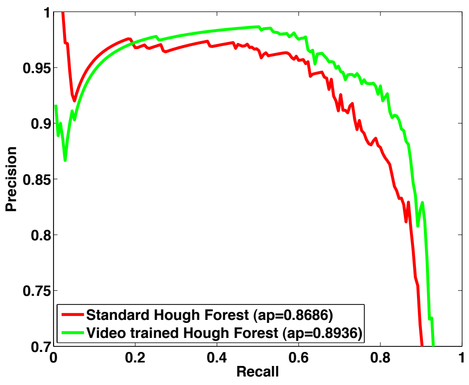


Figure 5.6: Here one can see two examples of simulated motion by using virtually warped versions of the original image.

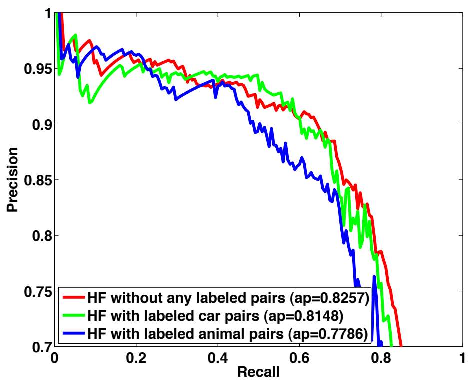
Since we can estimate the bounding boxes around moving objects in the video sequences, like described in Section 4.2.2, one can also ask why these objects are not added to the labeled dataset. The reason is the weak relation to the target-class.



(a) Weak-relation



(b) Virtual samples



(c) Labeled pairs

Figure 5.7: In (a), we show how different "degrees of weak-relation" of the motion data influences the performance . The results from the "virtual-samples" experiment are shown in (b). The motion data still supports the classifier, although the labeled data already includes some simulated local transformations. Figure (c) shows the results when extending the labeled dataset with data from the video sequences.

To illustrate this problem, we made another experiment, where we incorporated the patches extracted from different videos as labeled data to the framework. The red curve in Figure 5.7c shows the result of the standard Hough Forest without any additional labeled pairs on the ETHZ-cars dataset. Furthermore, we can see that the the PR curve gets slightly worse, if we add additional labeled pairs extracted from videos capturing cars (green curve). At first glance, this sounds controversial, but when we consider two facts, this effect can be explained easily. First, the car videos and the ETHZ-cars dataset do not stem from exactly the same data distribution and, second, we introduce a certain amount of label noise, as the additional labeled pairs are extracted from unlabeled videos and, thus, contain some background clutter. But the quintessence of this experiment can be seen in the blue PR curve. If we extend the labeled dataset with weakly-related data from video sequences, in our case from videos containing animals, the classifier gets confused and the generalization performance declines.

5.2.5 Discussion

The experiments about the object detection task all showed that we can improve the generalization performance towards the standard Hough Forest with our method (described in Section 3.3.2). We investigated the influence of the different parameters in the framework and showed that one has to choose the weighting and the amount of pair-wise motion data correctly to yield good results. Furthermore, we showed that the parameter t^{Δ} also influences the performance. But as already mentioned, this parameter is kind of ambiguous, when we consider some properties of the video sequences itself and its contents. Due to different FPS values and slow or fast moving objects, the level of transformation is not constant over all different video contents. Thus, we produce a certain amount of noise in the pair-wise motion data. A possibility to circumvent this problem in future work could be the estimation of velocity out of the optical flow and normalize it over the sequences. This would give a better control about the level of transformation of the objects.

Furthermore, our experiments reveal that the content of the motion data is important for the detection task. Of course, if the labeled dataset and the motion data are similar, the algorithm can benefit from this relation, which can be seen in Figure 5.7a. It also seems important to use motion data from naturally moving objects instead of virtually warped images, which only simulate motion (see Figure 5.7b).

We have to mention that we also tried to compare our method with a Hough Forest, which integrates the motion data as simple unlabeled data. That is, we adapted the Hough Forest to apply SSL by using a deterministic annealing approach, similar to Semi-Supervised Random Forests [49]. We actually had the intention to illustrate the limitation of an SSL approach, which is the assumption that labeled and unlabeled data have to be related, *i.e.*, stem from the same class distribution, to yield good performance. But as we could not improve the generalization performance with the additional unlabeled, but related data with the SSL approach in the first place, we omitted this experiment. In contrast to SSL, our method does not make such an assumption, as we can also improve the generalization performance with the use of weakly-related data, which is confirmed by the experiments described in this section.

5.3 Tracking

In order to perform the object tracking task, we followed Section 3.3.3 for building general codebooks, solely trained on pair-wise motion data. We always used 10000 pairs, 5000 labeled as "same" and 5000 labeled as "different". We generated two different codebooks, one trained on sequences capturing animals, and a second one using the simulated motion pairs, described in Section 5.2.4, from random unlabeled pictures. Both forests consist of 5 trees with a maximum depth of 8.

After training a class-invariant codebook, the forest totally lacks any leaf statistics. Thus, we have to update the leaves from the annotated object in the first frame of the tracking sequence. To increase the available amount of training data for the initial update of the forest, we created 5 perspectively transformed images of the first frame. Then, we cropped 100 positive and 500 negative patches from each virtual image and used them to update the general codebook to an instance-specific one. To account for illumination and pose changes over time, we updated the leaf statistics during tracking in each frame with 10 positive and 50 negative patches, if the confidence of the current tracked object is good. Otherwise no updates are done. The number of center votes in each leaf is limited to 30, which forces the tracker to concentrate on up-to-date information. This limitation was accomplished with a simple first-in-first-out (FIFO) structure in the leaf nodes.

The evaluation process is similar to that in the object detection tasks. We used the PASCAL overlap criterion [28], which was already reviewed in Section 5.2. In each frame, the predicted bounding box B_p is compared with the ground truth B_{gt}, yielding an overlap ratio a_o. Now, there are two possibilities to measure the tracking performance. First, one simply averages this overlap ratio a_o over all frames. The other way is to check if $a_o > 0.5$ holds for each frame. If yes, the current frame is denoted as "correct", otherwise as "wrong". Then, the percentage of correctly predicted frames can be calculated. In the following, we always state, which method was used for comparison. All results depicted in the following two sections are an average over three independent runs. As our tracking method does not consider scaling, we always resized the ground truth to the initial bounding box from the first frame of the sequences.

Our main goal in these tracking experiments is to show that we can yield state-of-the-art tracking results with the use of a motion-based class-invariant codebook. In other words, we want to show that a codebook, which is trained on transformations of moving objects, can exploit this knowledge especially in the task of object tracking to compete with other algorithms. Furthermore, we want to show that we can circumvent the limitation of the standard Hough Forest tracking framework [34], which is the class-specific codebook. In Section 3.3.3, the training procedure of a general codebook with the help of pair-wise motion data is described in detail.

We present two different tracking experiments in the following two sections. The first experiment considers the case where one has an a priori knowledge about the target object. This gives the standard Hough Forest tracking framework an advantage, as the codebook can be trained on target-class data in advance. However, in the second experiment, the task is to track arbitrary objects.

5.3.1 Class-specific tracking

In this section, we compare our tracking method with the online adaptable version of the standard Hough Forest (oaHF) on a pedestrian tracking task. As already described above, we used two different general codebooks, one trained with animal sequences (aVHF) and the other with virtually warped random data (rVHF). The tracking sequences were taken from [14] and [30], where pedestrians are shown when walking on a public place and in a subway station, respectively. Some frames from both sequences are illustrated in Figure 5.8.

The results of the experiment are presented in Table 5.1, where the values of the oaHF are adopted form [34]. In this experiment, we used the first evaluation method, *i.e.*, the mean over the overlap ratio a_o without any thresholding, as the results from [34] are also presented in this way. But we further provide the percentage of correctly predicted frames in brackets for our methods. As can be seen from the table, aVHF can compete with oaHF and even outperform it on two sequences. But on average, oaHF gives better results, which can be explained with the fact that this method is already pre-trained with pedestrian data. oaHF even yields reasonable results on the tough *i-Lids hard* sequence. On the other hand, our methods are more flexible and adaptable,

(a) i-Lids medium

(b) PETS09

Figure 5.8: Some frames of the sequences used for the class-specific tracking experiment. In both sequences, one has to track a pedestrian.

Sequence	aVHF	rVHF	oaHF [34]
i-Lids easy	34.7 ± 34.0 (39.7)	<u>50.3</u> ± 29.0 (62.4)	**69.2**
i-Lids medium	**77.6** ± 27.0 (87.0)	49.3 ± 28.3 (57.5)	<u>65.4</u>
i-Lids hard	21.7 ± 32.4 (24.7)	<u>41.8</u> ± 27.4 (50.3)	**65.9**
PETS09	**70.7** ± 27.3 (87.3)	48.2 ± 32.4 (59.4)	<u>60.3</u>
Average	<u>51.2</u>	47.4	**65.2**

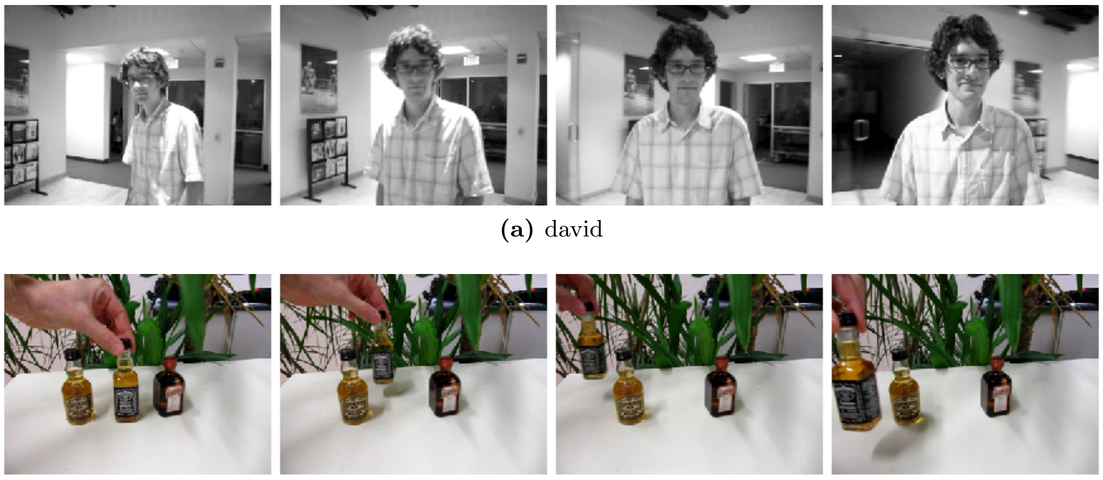
Table 5.1: The results on four pedestrian tracking sequences are presented as the mean of a_o. Additionally, for our proposed methods (aVHF & rVHF), the standard deviation is given and the percentage of correctly predicted frames is shown in brackets. Best performing methods are marked bold, second best methods are underlined.

as we do not assume any prior knowledge. This gives an advantage on easier tracking sequences, although aVHF yields bad results on the *i-Lids easy* sequence. But this comes from the fact that there is a period in the video, when the person to be tracked is partly occluded and covered in shadows. Thus, aVHF often totally loses the target object.

The table also indicates that aVHF outperforms rVHF. That is, motion data extracted from sequences of animals suit better for the task of pedestrian tracking than simulated motion data. On the other hand, rVHF seems to be more flexible and adaptable is it outperforms aVHF on the more difficult tracking sequences.

5.3.2 Tracking of arbitrary objects

Now, we want to focus on tracking arbitrary objects, *i.e.*, the target class to be tracked is not known in advance. We used five different sequences from [67], [3], [10], and [65]. Figure 5.9 depicts a subset of these frames.



(a) david

(b) liquor

Figure 5.9: Again, we depict some frames of the used tracking sequences. In (a), a face has to be tracked, and in (b), a bottle of liquor.

As the oaHF assumes a priori knowledge about the tracked object, we do not know which dataset would suit best for the pre-training of the trees and, thus, yield good performance in the tracking tasks. A comparison to oaHF would be inappropriate. Hence, we compare our two pre-trained general codebooks (aVHF and rVHF) with three state-of-the-art tracking algorithms, which do not require a priori knowledge about the target object. Namely, these are On-line Random Forests (ORF) [66], Multiple-Instance-Learning Boosting (MILB) [3], and PROST [67]. The results for all five different sequences are listed in Table 5.2. As an evaluation measurement for this experiment, we used the percentage of correctly predicted frames, *i.e.*, the second method presented above. For completeness, we also show the mean and standard deviation of the overlap ratio a_0 in brackets.

The resulting values in the table indicate, that our method can compete with the other tracking algorithms and even outperform them on some sequences. Furthermore, our method has the best resulting average value over all five sequences. That is, our proposed general codebook constantly gives good results in all tracking sequences used

Sequence	aVHF	rVHF	ORF [66]	MILB [3]	PROST [67]
david	65.6 (60.4 ± 22.3)	**91.8** (77.9 ± 23.1)	72.0	60.2	80.0
faceocc2	83.4 (75.9 ± 21.0)	**87.9** (78.3 ± 19.9)	65.0	81.1	82.0
lemming	79.2 (65.8 ± 28.0)	**79.5** (62.1 ± 27.3)	17.2	54.9	70.5
girl	63.0 (61.4 ± 22.8)	82.8 (70.6 ± 21.5)	**96.0**	56.6	89.0
liquor	**86.7** (75.7 ± 28.9)	82.4 (70.9 ± 32.4)	53.6	20.1	85.4
Average	75.6	**84.9**	60.8	54.6	81.4

Table 5.2: This table presents the results on five different tracking sequences as the percentage of correctly predicted frames. Mean and standard deviation of the PASCAL score are depicted in brackets. Additionally, the last row depicts the average values over all sequences. Again, best performing methods are marked bold, second best methods are underlined.

in this experiment. Please note, that we could not show that motion data from naturally moving objects outperform simulated motion from background images. Both, rVHF and aVHF show nearly the same performance for the two sequences *lemming* and *liquor*. Only these two sequences have color images, the other three have only grayscale images in its sequences. This could be a possible reason for the loss in performance of the aVHF method on these three videos. But of course, the different datasets used for pre-training the general codebook sometimes suit better, sometimes worse for a given target object, to be tracked. Nonetheless, both methods (rVHF and aVHF) can compete with other state-of-the-art tracking methods.

5.3.3 Discussion

With two different tracking experiments, we showed that our proposed tracking method can compete with other state-of-the-art algorithms. The first experiment about tracking class-specific objects, *i.e.*, the target object is known in advance, demonstrated a drawback of our method. If the target object is lost while tracking, and maybe another object is even adapted, it is very hard to retrieve the actual target object. In contrast, oaHF has the advantage of a class-specific pre-trained codebook. That is, if the tracked object is lost at any time, the retrieval is much easier. Of course, this advantage comes with the price of having less flexibility and adaptability.

The second experiment emphasized exactly these two properties of our proposed methods (aVHF & rVHF). On five different tracking sequences, we showed that rVHF can outperform some of them and can even yield the best result on average. As already mentioned above, the codebook trained with random simulated motion data shows better results than that trained with sequences of animals. It seems that rVHF is more adaptable than aVHF, which was already indicated in the first experiment. Furthermore, we assume that some datasets for pre-training the general codebook suit better, some worse for the different tracking sequences.

5.4 Chapter Summary

In this chapter, we presented our video database, which was used to regularize the Hough Forest for object detection tasks and to train a general codebook for tracking tasks. Furthermore, we presented our experimental results.

In the task of object detection, we investigated the influence of different parameters of the framework on the generalization performance. We showed that we can outperform the standard Hough Forest framework in three different datasets, when incorporating motion data.

In the task of object tracking, we presented two different experiments. We showed that we cannot outperform the standard Hough Forest tracking algorithm on the task of pedestrian tracking, as our method does not assume any prior knowledge about the target object. But we also showed that our method can yield state-of-the-art results on tracking arbitrary objects and even outperforms the other methods on some sequences and on average over all sequences.

Chapter 6

Conclusion and Outlook

Contents

6.1 Conclusion 69

6.2 Outlook 70

6.1 Conclusion

In this Master's Thesis, we presented a learning approach, which incorporates motion-based data from weakly-related moving objects with the goal to improve object detection and tracking tasks. Chapter 1 motivates the topic of improving learning tasks in computer vision with the help of unlabeled data, as it is much more practical and cost-efficient as opposed to large amounts of labeled data. Especially, the use of videos, as a source for unlabeled data, is very reasonable because they comprise an underlying structure given by real-world constraints, which is the time-space coherence of naturally moving objects. This fact is also exploited in this thesis to extract useful information from unlabeled video sequences, which was described in Chapter 4.

In Chapter 2, we reviewed the task of machine learning with all its different settings, like supervised, unsupervised or semi-supervised learning. Furthermore, we showed some approaches proposed in the current literature, which are similar to ours. Additionally, we established a link to the human visual system as we argue that human beings also learn object representations from unlabeled video data.

The learning approach, which integrates pair-wise motion data capturing local transformations from moving objects, was detailed in Chapter 3. As a foundation for our ideas, we reviewed the Random Forest framework [15] and depicted an extension to learn from local transformations. Furthermore, this extension was transferred to a state-of-the-art object detection and tracking framework, namely Hough Forests [33, 34, 58]. In order to improve the object detection task, we regularized the optimization problem with unlabeled motion data. For the tracking task, we trained a class-invariant codebook solely with motion data, which can be refined to track a specific instance of any object.

In chapter 5, we presented the experimental results for several detection and tracking tasks on different datasets. We showed that detection and tracking algorithms can be improved, when including local transformations of moving objects in the learning process. That is, a learning system can become more insensitive to these variations and transformations, which occur in natural scenes, by explicitly learning them. The experimental results for both detection and tracking tasks also showed that learning these transformations makes sense, as we can outperform the base approach and compete with state-of-the-art algorithms. Furthermore, we investigated the influence of some parameters in the framework and different content in the video sequences. Finally, we discussed and justified the outcome of all experiments.

6.2 Outlook

Several parts of the presented framework in this Master's Thesis offer space for improvements. First of all, one can investigate other forms of the splitting criterion for pair-wise data to yield more balanced subsets of pairs for the child nodes.

Another item is the improvement of the patch extraction method, as it fails in certain video sequences and does not handle some effects which often occur. One example are suddenly appearing letterings, which confuse the optical flow estimation, or more specifically, the subsequent segmentation procedure. Furthermore, the video database described in Section 5.1 can be optimized in a sense that it only contains video sequences which properly suit for the patch extraction process, in order to reduce noise.

As this thesis presents a general learning method, *i.e.*, learning local object transformations based on video data, it is not limited to a certain machine learning algorithm, like the Random Forests or Hough Forests, which are used here. Other algorithms could be proposed using such a regularization method to improve generalization performance in both detection and tracking tasks.

Finally, the influence of different image or patch representations can be explored, as we use many different data sources. Other feature descriptors may be more powerful for this task.

Bibliography

[1] Ahonen, T., Hadid, A., and Pietikäinen, M. (2006). Face description with local binary patterns: application to face recognition. *IEEE transactions on pattern analysis and machine intelligence*, 28(12):2037–41.

[2] Andriluka, M., Roth, S., and Schiele, B. (2008). People-tracking-by-detection and people-detection-by-tracking. *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

[3] Babenko, B., Yang, M., and Belongie, S. (2009). Visual tracking with online multiple instance learning. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 983–990. IEEE.

[4] Baker, S., Roth, S., Scharstein, D., Black, M. J., Lewis, J., and Szeliski, R. (2007). A Database and Evaluation Methodology for Optical Flow. *2007 IEEE 11th International Conference on Computer Vision*, (December):1–8.

[5] Balas, B. and Sinha, P. (2008). Observing object motion induces increased generalization and sensitivity. *Perception*, 37(8):1160–1174.

[6] Ballard, D. (1981). Generalizing the Hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2).

[7] Barron, J. L., Fleet, D. J., and Beauchemin, S. S. (1994). Performance of optical flow techniques.

[8] Ben-Gall, I. (2005). *Outlier Detection*, chapter Outlier De. Springer.

[9] Benczúr, A., Csalogány, K., and Lukács, L. (2007). Semi-Supervised Learning: A Comparative Study for Web Spam and Telephone User Churn. *In Graph Labeling*, pages 1–8.

[10] Birchfield, S. (1998). Elliptical Head Tracking Using Intensity Gradients and Color Histograms. In *1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1998. Proceedings*, number June, pages 232–237.

[11] Blitzer, J. (2008). Semi-supervised learning for natural language processing. *Computational Linguistics on Human Language*, (June):3–3.

[12] Borenstein, E. and Ullman, S. (2002). Class-specific, top-down segmentation. *Computer Vision—ECCV 2002*, pages 639–641.

[13] Bosch, A., Zisserman, A., and Munoz, X. (2007). Image Classification using Random Forests and Ferns. *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8.

[14] Branch, H. O. S. D. (2007). Imagery library for intelligent detec- tion systems i-lids.

[15] Breiman, L. (2001). Random forests. *Machine learning*, pages 1–33.

[16] Chapelle, O., Schölkopf, B., and Zien, A. (2006). *Semi-Supervised Learning*. MIT Press, Cambridge, MA.

[17] Cheng, Y. (1995). Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799.

[18] Comaniciu, D., Ramesh, V., and Meer, P. (2003). Real-time tracking of non-rigid objects using mean shift.

[19] Crandall, D., Felzenszwalb, P., and Huttenlocher, D. (2005). Spatial Priors for Part-Based Recognition Using Statistical Models. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 1:10–17.

[20] Cristianini, N. and Shawe-Taylor, J. (2000). *An introduction to support vector machines : and other kernel-based learning methods*. Cambridge University Press.

[21] Dai, W., Xue, G., and Yang, Q. (2007a). Transferring naive bayes classifiers for text classification. *PROCEEDINGS OF THE NATIONAL*, pages 540–545.

[22] Dai, W., Xue, G.-R., Yang, Q., and Yu, Y. (2007b). Co-clustering based classification for out-of-domain documents. *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '07*, page 210.

[23] Dai, W., Yang, Q., and Xue, G. (2007c). Boosting for transfer learning. *conference on Machine learning*.

[24] Dalal, N. and Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, pages 886–893.

[25] Dalal, N., Triggs, B., and Schmid, C. (2006). Human detection using oriented histograms of flow and appearance. *Computer Vision–ECCV 2006*, pages 428–441.

[26] Davis, L. (2009). Multiple instance feature for robust part-based object detection. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 405–412.

[27] Epshtein, B. and Ullman, S. (2007). Semantic Hierarchies for Recognizing Objects and Parts. *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

[28] Everingham, M., Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2009). The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 88(2):303–338.

[29] Felzenszwalb, P., McAllester, D., and Ramanan, D. (2008). A discriminatively trained, multiscale, deformable part model. *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

[30] Ferryman, J., Crowley, J. L., and Shahrokni, A. (2009). Pets 2009 benchmark data.

[31] Fleet, D. J. and Weiss, Y. (2005). Optical flow Estimation. *Handbook of Mathematical Models*, 19:239–258.

[32] Freund, Y., Schapire, R., and Abe, N. (1999). A short introduction to boosting. *JOURNAL-JAPANESE SOCIETY FOR ARTIFICIAL INTELLIGENCE*, 14(5):771–780.

[33] Gall, J. and Lempitsky, V. (2009). Class-specific Hough forests for object detection. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1022–1029.

[34] Gall, J., Razavi, N., and Gool, L, V. (2010). On-line Adaption of Class-specific Codebooks for Instance Tracking. *Citeseer*, pages 1–12.

[35] Gerhardstein, P., Shroff, G., and Dickerson, K. (2009). THE DEVELOPMENT OF OBJECT RECOGNITION THROUGH INFANCY. *New directions in*.

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

[37] Heitz, G. and Elidan, G. (2005). Transfer learning of object classes: From cartoons to photographs. *Workshop on Inductive Transfer*, pages 2–5.

[38] Horn, B. K. and Schunck, B. G. (1981). Determining optical flow. *Artificial Intelligence*, 17(1-3):185–203.

[39] Jepson, A., Fleet, D., and Black, M. (2002). A layered motion representation with occlusion and compact spatial support. *Computer Vision—ECCV 2002*, 1:692–706.

[40] Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. *Machine Learning: ECML-98*, pages 137–142.

[41] Joachims, T. (1999). Transductive Inference for Text Classification using Support Vector Machines. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 200–209. Citeseer.

[42] Joachims, T. and Others (1998). Estimating the generalization performance of an SVM efficiently. *constraints*, 1.

[43] Kasabov, N. (2004). Transductive support vector machines and applications in bioinfomatics for promoter recognition. *International Conference on Neural Networks and Signal Processing, 2003. Proceedings of the 2003*, 3(2):1–6.

[44] Kate, R. J. and Mooney, R. J. (2007). Semi-supervised learning for semantic parsing using support vector machines. *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers on XX - NAACL '07*, (April):81–84.

[45] Kühne, G., Richter, S., and Beier, M. (2001). Motion-based segmentation and contour-based classification of video objects. *Proceedings of the ninth ACM international conference on Multimedia - MULTIMEDIA '01*, page 41.

[46] Lampert, C., Nickisch, H., and Harmeling, S. (2009). Learning to detect unseen object classes by between-class attribute transfer. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 951–958.

[47] Leibe, B., Cornelis, N., Cornelis, K., and Van Gool, L. (2007). Dynamic 3D Scene Analysis from a Moving Vehicle. *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

[48] Leistner, C. (2010). *Semi-Supervised Ensemble Methods for Computer Vision.* PhD thesis.

[49] Leistner, C., Saffari, A., Santner, J., and Bischof, H. (2010). Semi-supervised random forests. In *Computer Vision, 2009 IEEE 12th International Conference on*, volume 4, pages 506–513. IEEE.

[50] Li, N. (2008). Unsupervised natural experience rapidly alters invariant object representation in visual cortex. *Science*, 321(September):1502–1507.

[51] Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137.

[52] Lowe, D. (1999). Object recognition from local scale-invariant features. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, pages 1150–1157 vol.2.

[53] Lucas, B. and Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. In *International joint conference on artificial intelligence*, volume 3, pages 674–679. Citeseer.

[54] Mitchell, T. M. (1997). *Machine Learning.* McGraw-Hill, New York.

[55] Mobahi, H., Collobert, R., and Weston, J. (2009). Deep learning from temporal coherence in video. *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, pages 1–8.

[56] Moosmann, F., Triggs, B., and Jurie, F. (2007). Fast discriminative visual codebooks using randomized clustering forests. *NIPS*, 19:985.

[57] Mumford, D. and Shah, J. (1989). Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on Pure and Applied Mathematics*, 42(5):577–685.

[58] Okada, R. (2009). Discriminative Generalized Hough Transform for Object Detection. *Entropy*, (Iccv).

[59] Pan, S. J. and Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

[60] Pearson, K. (1901). LIII. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 6*, 2(11):559–572.

[61] Perry, G., Rolls, E. T., and Stringer, S. M. (2010). Continuous transformation learning of translation invariant representations. *Experimental brain research. Experimentelle Hirnforschung. Expérimentation cérébrale*, 204(2):255–70.

[62] Potts, R. B. (1952). Some generalized order-disorder transformations. *Some generalized order-disorder transformations. Mathematical Proceedings of the Cambridge Philosophical Society*, 48:106–109.

[63] Raina, R., Battle, A., Lee, H., and Packer, B. (2007). Self-taught learning: Transfer learning from unlabeled data. *on Machine learning*.

[64] Raina, R., Ng, A. Y., and Koller, D. (2006). Constructing informative priors using transfer learning. *Proceedings of the 23rd international conference on Machine learning - ICML '06*, pages 713–720.

[65] Ross, D. a., Lim, J., Lin, R.-S., and Yang, M.-H. (2007). Incremental Learning for Robust Visual Tracking. *International Journal of Computer Vision*, 77(1-3):125–141.

[66] Saffari, A., Leistner, C., Santner, J., Godec, M., and Bischof, H. (2009). On-line Random Forests. *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1393–1400.

[67] Santner, J., Leistner, C., Saffari, A., Pock, T., and Bischof, H. (2010). PROST: Parallel robust online simple tracking. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, number 813396. IEEE.

[68] Santner, J., Unger, M., Pock, T., Leistner, C., Saffari, A., and Bischof, H. (2009). Interactive texture segmentation using random forests and total variation. In *Proceedings of the British Machine Vision Conference (BMVC), London, UK*, pages 1–12.

[69] Schapire, R. and Singer, Y. (2000). BoosTexter: A boosting-based system for text categorization. *Machine learning*, 39(2):135–168.

[70] Schmidhuber, J. (1995). On Learning How To Learn Learning Strategies.

[71] Schroff, F., Criminisi, A., and Zisserman, A. (2008). Object class segmentation using random forests. In *Proceedings of the British Machine Vision Conference*. Citeseer.

[72] Sharp, T. (2008). Implementing decision trees and forests on a gpu. *Computer Vision–ECCV 2008*, pages 595–608.

[73] Soonthornphisaj, N. (2005). Combining ILP with Semi-supervised Learning for Web Page Categorization. *International Journal of Computational*.

[74] Stark, M., Goesele, M., and Schiele, B. (2009). A shape-based object class model for knowledge transfer. *2009 IEEE 12th International Conference on Computer Vision*, pages 373–380.

[75] Stavens, D. and Thrun, S. (2010). Unsupervised learning of invariant features using video. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1649–1656. IEEE.

[76] Stringer, S. M., Perry, G., Rolls, E. T., and Proske, J. H. (2006). Learning invariant object recognition in the visual system with continuous transformations. *Biological cybernetics*, 94(2):128–42.

[77] Taylor, G., Fergus, R., LeCun, Y., and Bregler, C. (2010). Convolutional Learning of Spatio-temporal Features. *Computer Vision–ECCV 2010*, pages 140–153.

[78] Thrun, S., Mitchell, T., and SCIENCE., C.-M. U. P. P. D. O. C. (1994). *Learning one more thing*. Number September. Citeseer.

[79] Vapnik, V. N. (1998). *Statistical Learning Theory*. Wiley-Interscience.

[80] Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, pages I–511–I–518.

[81] Viola, P., Jones, M. J., and Snow, D. (2003). Detecting pedestrians using patterns of motion and appearance. *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 734–741 vol.2.

[82] Wallis, G. and Bülthoff, H. H. (2001). Effects of temporal association on recognition memory. *Proceedings of the National Academy of Sciences of the United States of America*, 98(8):4800–4.

[83] Wang, X. and Han, T. (2010). An HOG-LBP human detector with partial occlusion handling. *Computer Vision, 2009 IEEE 12th.*

[84] Werlberger, M., Trobin, W., Pock, T., Wedel, A., Cremers, D., and Bischof, H. (2009). Anisotropic Huber-L1 optical flow. In *Proceedings of the British Machine Vision Conference*, pages 1–11.

[85] Weston, J., Kuang, R., Leslie, C., and Noble, W. S. (2006). Protein ranking by semi-supervised network propagation. *BMC bioinformatics*, 7 Suppl 1:S10.

[86] Weston, J., Leslie, C., Ie, E., Zhou, D., Elisseeff, A., and Noble, W. S. (2005). Semi-supervised protein classification using cluster kernels. *Bioinformatics (Oxford, England)*, 21(15):3241–7.

[87] Wiskott, L. and Sejnowski, T. J. (2002). Slow feature analysis: unsupervised learning of invariances. *Neural computation*, 14(4):715–70.

[88] Wu, P. and Dietterich, T. G. (2004). Improving SVM accuracy by training on auxiliary data sources. *Twenty-first international conference on Machine learning - ICML '04*, page 110.

[89] Xu, J., Fumera, G., and Roli, F. (2009). Training SpamAssassin with Active Semi-supervised Learning. *Proceedings of the 6th.*

[90] Yang, L. and Jin, R. (2008). Semi-supervised learning with weakly-related unlabeled data: Towards better text categorization. *Twenty-Second Annual Conference on*, pages 1–8.

[91] Yao, A., Uebersax, D., Gall, J., and Van Gool, L. (2010). Tracking People in Broadcast Sports. *Pattern Recognition*, pages 151–161.

[92] Yilmaz, A., Javed, O., and Shah, M. (2006). Object tracking. *ACM Computing Surveys*, 38(4):13–es.

[93] Yu, D., Varadarajan, B., Deng, L., and Acero, A. (2010). Active learning and semi-supervised learning for speech recognition: A unified framework using the global

entropy reduction maximization criterion. *Computer Speech & Language*, 24(3):433–444.

[94] Zach, C., Pock, T., and Bischof, H. (2007). A duality based approach for realtime TV-L 1 optical flow. In *Proceedings of the 29th DAGM conference on Pattern recognition*, volume 1, pages 214–223. Springer-Verlag.

[95] Zhang, D. and Lu, G. (2001). Segmentation of moving objects in image sequence: A review. *Circuits, Systems, and Signal Processing*, 20(2):143–183.

[96] Zhou, D. and Schölkopf, B. (2005). Semi-supervised learning on directed graphs. *Advances in neural information*.

[97] Zhu, X. and Goldberg, A. B. (2009). *Introduction to Semi-Supervised Learning*. Morgan and Claypool Publishers.