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)

Graz University of Technology

Institute for Computer Graphics and Vision

Master Thesis

Transfer Learninig with Random Forests

Lukas Daum

October 30, 2011

Thesis supervisors

Univ. Prof. DI Dr. Horst Bischof

DI Dr. Christian Leistner

Abstract

Random Forests are a machine learning method. They can be used for classification and regression tasks. Several object detection algorithms also use Random Forests as classifiers. These object detectors need to be trained on positive example images which show the objects of interest and on negative example images which do not show these objects. Positive and negative examples are drawn from the same domain which is called the target domain. This target domain also denotes the domain, which the learner should perform well at.

Transfer Learning methods have the goal to gain more information about a target domain by extracting relevant information from a source domain. The source domain denotes a domain which is similar to the target domain but not the same. When classifiers like Random Forests are trained on data from a target domain, Transfer Learning methods can be used to add additional knowledge from a source domain. Since preparation of target training data can be very expensive the ability to reuse data from a source domain is very beneficial.

This Master Thesis addresses combination of Random Forest based methods with methods of Transfer Learning. PCA, Sparse Coding and Transfer Boosting are combined with Random Forest based methods. We also propose a Transfer Learning method for Random Forests. Random Forests have a strong mechanism to find out how similar one example is to a batch of other examples. This mechanism is used to remove examples from the source domain data which are not similar to the data from the target domain. These removed examples are so called outliers. Remaining examples can then be transferred to the target domain.

The aim of this master thesis is to compare these Transfer Learning methods amongst each other and to other state of the art methods. Evaluation of these methods on two classification tasks and two object detection tasks show that Transfer Boosting in combination with Random Forests can improve the performance on a given task as well as the removing of outliers in the source data set. For PCA and Sparse Coding the experiments

did not show any performance improvement compared to Random Forest based methods without Transfer Learning.

Keywords: Transfer Learning, Machine Learning, Random Forests, Computer Vision, Hough Forest, Transfer Boosting, PCA, Sparse Coding, Pedestrian Detection, Face Detection

Kurzfassung

Random Forests sind machine learning Methoden. Sie können für Klassifizierung und Regression verwendet werden. Einige Objekt detektierungs Algorithmen basieren ebenfalls auf dem Prinzip von Random Forests. Diese Detektoren müssen auf positiven und negativen Beispielen trainiert werden um Objekte in einer Szene detektieren zu können. Positive Beispiele bezeichnen Bilder die Objekte von interesse beinhalten. Negative Beispiele bezeichen Bilder die keines dieser Objekte zeigen. Positive und Negative Beispiele stammen aus der selben Domäne, der so genannten target domain. Diese target domain bezeichnet auch die Domäne auf welcher der trainierte Detektor angewendet werden soll.

Transfer Learning Methoden verfolgen das Ziel, zusätzliche Informationen über diese target domain zu sammeln. Dazu werden Informationen aus einer so genannten source domain verwendet. Diese source domain ist eine Domäne welche ähnlich aber nicht identisch zur target domain ist. Neben den Daten aus der target domain können so zusätzliche Daten aus der source domain für den Trainingsprozess verwendet werden. Diese zusätzlichen Daten können die Genauigkeit eines Klassifizierers verbessern. Das Sammeln von Daten aus einer target domain kann ein sehr aufwendiger und zeitintensiver Prozess sein. Deshalb ist es vorteilhaft wenn bereits vorhandene Daten aus einer source domain wiederverwendet werden können.

Diese Masterarbeit behandelt die Kombination von Transfer Learning Methoden und Methoden die auf dem Prinzip von Random Forests basieren. PCA, Sparse Coding und Transfer Boosting werden mit Radom Forests kombiniert und eine spezielle Transfer Learning Methode für Random Forests wird vorgestellt. Random Forests bieten ein gutes Werkzeug um herauszufinden wie ähnlich ein bestimmtes Beispiel einer Menge von anderen Beispielen ist. Auf diese Weise können Beispiele aus den source domain Daten entfernt werden, die sich zu sehr von den Daten der target domain unterscheiden. Diese Beispiele werden als outliers bezeichnet. Beispiele, die keine outliers sind können als zusätliche Daten verwendet werden.

Das Ziel diese Masterarbeit ist es, diese Transfer Learning Methoden untereinander, und mit anderen modernen Methoden, zu vergleichen. Die Evaluierung dieser Methoden in zwei Klassifizierungsproblemen, sowie zwei Objekt Detektierungsproblemen zeigt, dass Random Forests mit Transfer Boosting bessere Resultate liefern können. Auch das Entfernen der outliers aus den source domain Daten kann die Resultate verbessern. PCA und Sparse Coding hingegen führten in den Experimenten zu keiner Verbesserung der Resultate im Vergleich zu Random Forest Methoden ohne Transfer Learning.

Keywords: Transfer Learning, Machine Learning, Random Forests, Computer Vision, Hough Forest, Transfer Boosting, PCA, Sparse Coding, Pedestrian Detection, Face Detection

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Motivation

In machine learning the performance of a learner on a specific task is always dependent on the quality and quantity of data it was trained on. Thus, there is the need to carefully handcraft as much data as possible to achieve a good performance. Since this is a time consuming process there is a high interest in speeding up this process. For this purpose transfer learning can be used. Transfer learning tries to take into account some additional data, which is already present, to increase a learners performance. D_T denotes the target domain, which the learner should perform well at. This is the domain of the training and test data. D_S denotes the source domain, which is similar to D_T but not the same. D_S provides additional data to increase the performance on D_T.

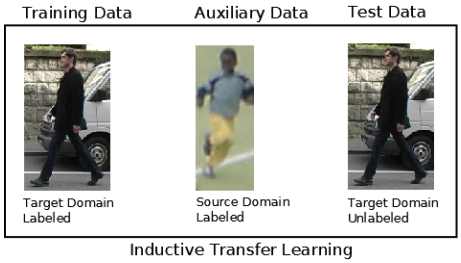
An example shows the problem and how Transfer Learning can help to improve a learned model. We assume that a company wants to install a camera inside a public building to measure the number of persons passing this camera. Therefore, the camera is connected to a computer which is running an object detector. This detector should use a machine learning method to distinguish persons from other objects or background. This

learner needs to be trained on data examples to learn a model. In this case, the target domain D_T is the domain of persons including the view angle of this specific camera, the light setting inside this building and camera related properties like color scheme or resolution. To obtain data for this target domain someone needs to take pictures with this camera and to label them. The labels need to be stored along with the corresponding images. This is a very time consuming process because a high number of labeled images is needed to yield good results in detection. The responsible technician decides to use Transfer Learning because he has access to many other object detection data sets which are labeled already. The source domain D_S is chosen as the domain of persons in an outdoor setting. The technician labels only a few images from D_T and adds the existing data set from D_S. During the process of Transfer Learning parts of D_S data which are relevant for D_T are used as additional examples for training. The data set for D_S includes a very high number of examples and it is still strongly related to D_T. Thus, the detector with the trained learner yields good detection results while the technician did only label a few examples himself.

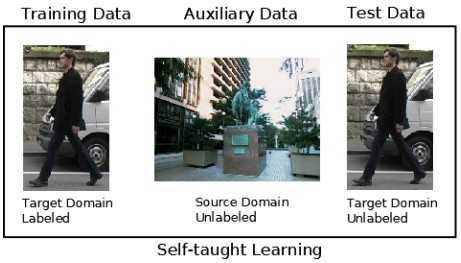
Especially in computer vision the collection and the labeling of data is very time consuming. For object detection tasks one has to collect images showing the object. Furthermore, each image has to be assigned a label. The form of these labels depends on the used machine learning algorithm. Some methods are able to use weakly labeled data. For images a weak label could be the count of objects inside the image or the information whether or not the image contains the desired object. A more precise label is the set of object bounding boxes for every image or even a complete segmentation for every object inside the image. The easiest solution of this problem is to use data that is already available and labeled. Many detection data sets are available at popular machine learning repositories. Some of them also contain different forms of labels and can therefore be used by many different learning algorithms. Since these data sets may not be of the same distribution as the target task, Transfer Learning is needed to make use of this data.

1.2 Transfer Learning

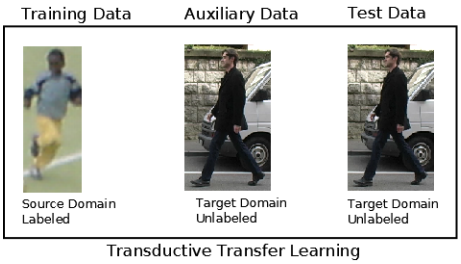
Transfer Learning (TL) is a sub-field of Machine Learning. The main focus of TL is the use of some auxiliary data X_s which is drawn from a so-called source domain D_S. This additional data should improve the performance of a learner or classifier which is trained on data X_t, drawn from D_T and evaluated on data X_u which is also drawn from target domain D_T.



(a) Setting for Inductive Transfer Learning



(b) Setting for Self-taught Learning



(c) Setting for Transductive Transfer Learning

Figure 1.1: Overview of different Transfer Learning Settings

Figure 1.1 illustrates the main settings of transfer learning. Subfigure 1.1(a) shows the setting of inductive transfer learning with training data which is drawn from the target domain D_T and auxiliary data from source domain D_S. In this case the auxiliary data contains labels which correspond to the class labels used in domain D_T. Thus, the samples contained in X_s can be used directly as additional samples for training.

Subfigure 1.1(b) shows the setting for self-taught learning which is a special form of inductive transfer learning where X_s does not contain any labels. Thus, the auxiliary data cannot be used as additional data for training but may help to find some common patterns which can be used to improve the quality of the trained learner.

Subfigure 1.1(c) shows the setting for transductive transfer learning. Here the labeled training data X_t is drawn from source domain D_S and thus its is hard to train a learner for D_T which yields a good performance. Therefore, some unlabeled examples from D_T are used to aquire some knowledge about the target domain. These samples in X_s can be used to shift the representation of samples in X_t so that they become more similar to the target domain samples. [39]

1.3 Object detection with Random Forests

Random Forests are learners or classifiers which are very popular because of the advantages they have over other kinds of machine learning algorithms. While they are very simple and can handle large amounts of data, they also yield very good results in classification and regression tasks compared to other state-of-the-art machine learning methods. A Random Forest is an ensemble of decision tree classifiers. Each tree is trained on the whole input data. Starting at the root node of each tree, the data is then split into two subsets according to the splitting rule defined for the tree. This process is repeated until every example inside the data set reaches a leaf node. Test examples will also reach leaf nodes when evaluated and will then be assigned the same label as the training examples which ended up in the same leaf. [2] [8] [17]

Figure 1.2: Example of object detection on TUD Pedestrian set

Object detection In object detection, the goal is to find one or more specified objects within a given scene. The result of this detection process is a number of detections. Each detection contains the objects position within the scene. Depending on the detector a detection may also contain information about the object size and orientation. To visualize a detection, a bounding box is drawn for each detection as shown in figure 1.2. Object detectors can use machine learning methods to learn object models. This means that

they learn to distinguish between different object classes. The learned model is contained inside the trained classifier as a mapping of input features to a specific object class output. The advantage of this approach is that the developer does not need to have that much knowledge about the detection task because the system can learn by examples. Therefore, a classifier is trained on some positive example images which contain an instance of the object and some negative example images which do not contain an object. Thus, the quality of such an object detector depends on the quality of the used learning algorithm.

Since Random Forests became very popular in machine learning many approaches have been developed which use them for object detection. They can be trained on small images showing only parts of the desired object. These small images are called image patches. When trained on these image patches they are able to detect different parts of the object seperately. The use of image patches makes them a very robust method for detection tasks because they are able to detect partially occluded or rotated objects.

1.4 Goal of this Master Thesis

The goal of this thesis is to compare different methods of Transfer Learning (TL) in combination with Random Forest based methods. We show how to combine PCA, Sparse Coding, Transfer Boosting with Random Forests and Hough Forests. We also propose a new method called transfer learning with outlier detection (TLOD). This method is based on the ability of Random Forests to find outlying examples within a certain object class. TLOD can also be used with Random Forests and Hough Forests.

These 4 TL methods will be evaluated on 4 different tasks. The first task is the classification of newsgroup entries which belong to different newsgroup categories. The second task is a classification task on handwritten character images. The third task is a pedestrian detection task and the fourth task is a face detection task. For each task the performance of our methods will be discussed. Our results will also be compared to the results of other methods.

Finally we want to identify cases where it is beneficial to use these TL methods and cases where they should not be used. Therefor we measure the impact of TL on the different tasks and compare these results to the results of Random Forest based methods without TL.

1.5 Organisation of this Master Thesis

Chapter 2 will discuss related work in computer vision, Machine Learning and Transfer Learning and. Section will discuss image features used in computer vision. Section 3.3 will introduce the field of object detection. Section will discuss Decision Trees. Section 2.3.2 will introduce Random Forests [8] which are simple and powerfull classifiers and which is widely used in machine learning and computer vision. Section will discuss Hough Forests which are Random Forest based object detectors. Section will discuss the concept of Boosting algorithms. Section 2.4 will introduce Transfer Learning and discuss the main categories of TL which are Inductive Transfer Learning, Self-Taught Learning and Transductive Transfer Learning. Section 2.4.1.1 will review some methods of Inductive Transfer Learning, section 2.4.1.3 will review some methods of Self-Taught Learning and section 2.4.1.4 will describe methods of Transductive Transfer Learning.

Chapter 3 will present our methods to combine Transfer Learning with Random Forests to improve their performance. This chapter is divided into tree parts. The first part will discuss Transfer Learning for Random Forests in general machine learning scenarios. The second part will discuss Transfer Learning for Random Forests specialized for image classification tasks. The third part will discuss Transfer Learning for Hough Forests which are Random Forest based Object Detectors.

Chapter 4 will present experimental results for the used Transfer Learning methods and the Object Detector with TL. Sections 4.1, 4.2 and 4.3 will compare the used TL methods to some state-of-the-art methods. For this purpose, popular machine learning data sets will be used. Section 4.4 and section 4.5 will show the results for two object detection tasks, both for the Object Detector without TL and for our solution.

Chapter 5 will give a conclusion of the work presented in this thesis and an outlook to possible future work.

Chapter 2

Related Work

Contents

This chapter will discuss the concepts which are used for our work. The chapter is divided into four sections. The first section will discuss concepts of image features for computer vision. The second section will give a short introduction to object detection with a focus on Random Forest based methods. The third section will discuss machine learning methods like Random Forests [2] [8] [17], Hough Forests [15] and boosting [45]. The fourth section will give an overview of Transfer Learning concepts and will discuss the methods we use for our work.

2.1 Image Features

An image feature is one specific attribute of a digital image. Such a feature can represent attributes like pixel values at a specific image position, color, or more advanced properties like information about edges inside the image or even the occurance of a complex pattern inside the image. These are just a few examples and there are many more possibilities of what a feature can describe. For each feature and image a specific feature value for this image can be calculated. For instance, if the feature is defined as the color at a pixel position then the feature value may be red for one image but green for another one. Thus,

the feature defines what is observed and the feature value is a specific observation for a given image. To describe a single image usually more than one feature is needed. When a set of N features is defined it is called a feature space $F = \{f_0, f_1 \dots f_N\}$ where f_i is one single feature. The features contained in F can be used to describe a single image by calculating a feature value v_i for each feature f_i. The result is a vector of feature values $V = \{v_0, v_1 \dots v_N\}$ which represents the available information about the given image. A very simple example for an image feature is the raw pixel value. The raw pixel value is the color or grayscale value of a single pixel inside the image. When F is the set of all pixel positions inside an image then V contains exactly the same information as the image itself. In this section we will discuss two common types of features which are used for our work.

2.1.1 Pixel-Pair Features

A Pixel-Pair Feature is defined by two pixel locations $p(x, y)$ and $p(u, v)$ inside the image. The raw pixel values from both pixel positions are combined by an arithmetic operation. Usually $p(u, v)$ is subtracted from $p(x, y)$ as shown in equation 2.1.

$$v = p(x, y) - p(u, v) \tag{2.1}$$

Since the pixel pair feature observes two positions it provides information about differences inside the image.

2.1.2 Haar Features

Haar Features [50] are advanced features which are able to provide information about edges inside an image.

Figure 2.1 shows different kinds of Haar Features. Each feature consists of black regions and white regions. The sum of all pixel values inside the white regions are subtracted from the sum of all pixel values inside black regions.

$$v = \sum p(black) - \sum p(white) \tag{2.2}$$

The different kinds of Haar Features shown in figure 2.1 can be used to detect edges at different angles. Features A and C are able to detect vertical edges, Feature B is able to detect a horizontal edge, and D and E can be used to detect diagonal edges. For each Haar Feature the size of regions and the position inside the image window is variable.

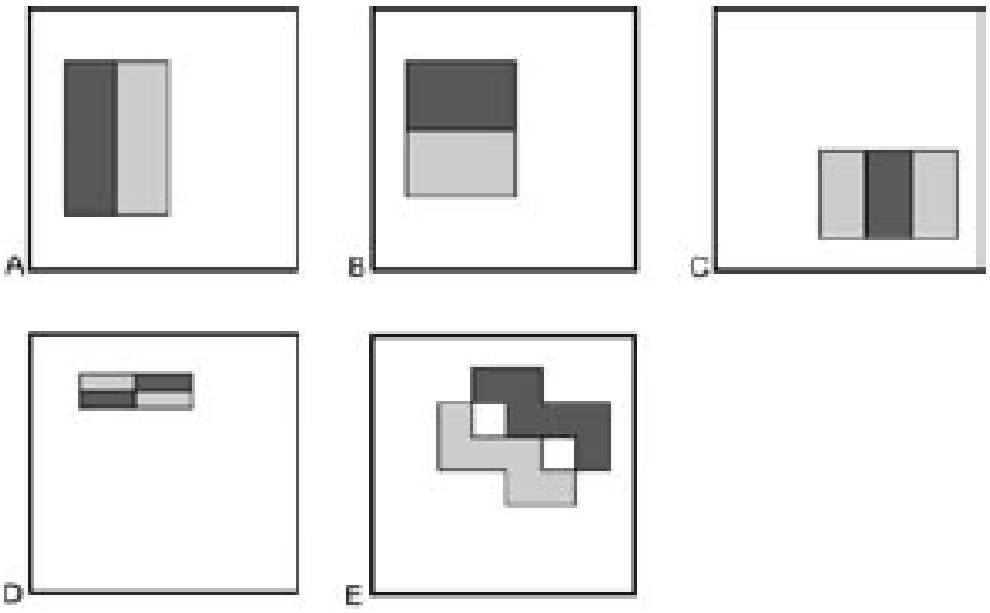


Figure 2.1: Examples of Haar Features, image taken from [24]

It is very time consuming to calculate many Haar Features at once because a high number of pixel values needs to be read out. Thus, an Integral Image [50] can be used to speed up this process. An Integral Image is a special representation of the original input image where each pixel value is the sum of all pixel values above and to the left. To calculate such an Integral Image every pixel needs to be read once. Every rectangular region of a Haar Feature can then be calculated as the difference of the integral value of the upper left corner and the bottom right corner [50].

2.2 Object detection

In this section the basic concept of object detection will be explained and how machine learning methods can be used for Object Detectors. Hough Forests [15] which are used for our work will be discussed in detail.

2.2.1 Introduction to object detection

An Object Detector is a method to detect objects of interest within one or more images. These objects can be things like cars, different animals, persons and many more. Depending on the detection algorithm an Object Detector is able to give a hypothesis about

object properties like position inside the image and object dimensions. Two quantities determine the quality of an Object Detector. The first one is the rate of true positive detections which is the rate of correct detected objects. The second one is the rate of false positives which are false detections or duplicate detections of the same object. While it is possible to fully programm an Object Detector it is a common approach to use machine learning algorithms to learn object representations. The benefit of this approach is that the Detector can be trained on examples which reduces the complexity of the system from a developers view.

Examples for Object Detectors using machine learning methods are the Viola-Jones Detector [50] [24], object detection via Codebooks [29] [28], Hough Forests [15] and Random Subwindows for Robust Image Classification [34]. The latter three methods use Random Forests to learn object properties.

2.2.2 Performance Measurement for Object Detectors

To measure the performance of an Object Detector the detection hypothesis needs to be compared to the original labels of the used evaluation data set. The PASCAL * project provides rules for performance measurement of detection algorithms which are based on the detection of object bounding boxes. For each test image I_k a set of ground truth bounding box rectangles $G_k = \{g_1 \dots g_n\}$ needs to be available, where n is the number of objects contained in image I_k. These rectangles are compared to detections made by an Object Detector $D_k = \{d_1 \dots d_n\}$. The area of overlap A_o represents the quality of a single detection and is calculated as shown in equation 2.3.

$$A_o = \frac{size(d_i \cap g_i)}{size(d_i \cup g_i)} \tag{2.3}$$

Here, $size(d_i \cap g_i)$ denotes the size of the overlap area and $size(d_i \cup g_i)$ denotes the sum of area size for d_i and the area size for g_i. If $A_o > 0.5$, the detected rectangle is treated as a positive detection. Otherwise the rectangle is regarded as a false positive otherwise. Multiple detections of the same object are also counted as false positives. Thus, only the most accurate detection is taken.

Based on the number of true positives and false positives, recall, precision and f-measure of the object detector can be calculated as described in [1]. Equations 2.4, 2.5 and 2.6 show the calculation of recall, precision and f-measure values.

*http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2010/htmldoc/

$$Recall = \frac{TP}{nP} \tag{2.4}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.5}$$

$$F - measure = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision} \tag{2.6}$$

TP is the number of true positive detections made by the detector. FP is the number of false positive detections and nP is the real number of positives. These calculations are done for multiple threshold values to obtain a recall-precision curve as a final output and performance visualization.

2.3 Machine Learning

This section will explain the concept of decision trees [49] [41] [42] and how they are used to create Random Forests [2]. Another topic within this section is boosting [45] which is a method to build ensemble classifiers.

2.3.1 Decision Trees

Decision trees [49] [41] [42] have a long history in machine learning since they have several advantages over other classification algorithms. They can handle high dimensional data sets in an efficient way with low computational effort and without the need to reduce the dimensionality. [8]

The algorithm to grow a random decision tree is shown in algorithm 1. This is the decision tree model we use for our work. There are various different decision tree algorithms that are similar to this algorithm but have different splitting rules, stopping criteria and feature selection rules.

Algorithm 1 Random decision tree growing algorithm [8]

- **Require:** Training set X with N samples and M features.
- **Require:** Number of random drawn features for splitting $F \leq M$
- **Require:** Splitting rule S to measure the quality of a split
- **Require:** Stopping criterion C. e.g. given data Y contains only one class

1. Create decision tree, provide training set X to it's *Root Node*
2. Recursively grow nodes starting at the *Root Node*
3. If(C is met) Stopping Criterion
   - Make the node a *Leaf Node* and store the class label.
4. Else
   - Take F random features out of M to split X into two sub sets $X(l)$ and $X(r)$
   - Find the best split according to splitting rule S
   - Store the best split's feature index and threshold for this node
   - Create a left child node, providing $X(l)$ instead of X
   - Create a right child node, providing $X(r)$ instead of X

Breiman [8] proposes the use of *Classification and Regression Trees (CART)* [49] as decision tree implementation. They use the *Gini Criterion* as splitting rule S, and the stopping criterion C is met when one of the statements below is true for the observed node.

- All samples have the same value for the chosen split feature.
- All samples belong to the same class.
- Tree depth has reached a defined maximum value.
- Number of samples is less than a defined minimum.
- Number of samples in one or both child nodes is less than a defined minimum.
- The split improvement according to S is less than a defined minimum.

The C4.5 algorithm [41] [42] is similar to CART but uses *Information Gain* as splitting rule S, which is related to the entropy of the given data set.

Another group of decision tree algorithms is introduced by the FACT algorithm [33]. It was invented to reduce the feature selection bias of traditional decision tree algorithms like CART and C4.5. Well known successors of the FACT algorithm are QUEST [32], CRUISE [25] [26] and GUIDE [31]. These algorithms use different mathematical models to find optimal splits instead of choosing the best out of a random set.

2.3.2 Random Forests

Random Forests [2] [8] [17] are ensemble classifiers that consist of multiple decision trees. Each tree is trained seperately and has an internal structure which is different from all other trees in the forest. This happens because every node split inside a decision tree is chosen from a few random splits. For the classification of one sample every tree in the forest will vote for a class, and the most popular class is the classification output for the forest classifier. Thus, a false classification by one tree can be corrected by other trees voting for the correct class. The percentage of votes for the most popular class is also called the confidence. Random Forests offer several benefits which makes them a viable choice over other classifiers. Every tree within the ensemble is based on randomly selected features, the output hypothesis for one example is averaged over the whole forest. Thus overfitting can be avoided very well. This means that the ability to generalize on new data is improved. Random Forests are also well suited to use Bagging [6] which is another method to reduce vaiance and overfitting. Another advantage of Random Forests is, that they are computationally fast and that they can handle a high number of data examples.

2.3.2.1 Bagging

Bagging or *Out Of Bag Error Estimation* [7] is a technique to get an unbiased error estimation during the training phase. When training a single tree C_t for the forest, an *In Bag Set* B_t needs to be generated. This set is a bootstrap sample filled by randomly choosing samples from the original training set. Samples which are already in B_t are replaced to avoid duplicates. The remaining samples form the *Out Of Bag Set* OB_t. The tree is trained using B_t and afterwards used to classify OB_t. After all trees have been trained the classifications of their *Out Of Bag Sets* are combined. For each sample the most popular class is the hypothesis and the percentage of votes for other classes is the *Out Of Bag Error*. When *Bagging* is used there is no need to estimate the test error by more time consuming methods like *k-Fold Cross Validation*. Algorithm 2 shows how Bagging is used [8]. The use of Bagging has several benefits. No tree within the ensemble is trained

Algorithm 2 Random Forest growing algorithm with Bagging [8]

- **Require:** Training set X with N samples and M features
- **Require:** Tree count T
- For($t = 0$ to T),
  1. Create a unique *In Bag Set* B_t by random subsampling from X with replacement
  2. The corresponding *Out Of Bag Set* $OB_t = X - B_t$
  3. Train a new decision tree C_t on B_t
  4. Classify examples in OB_t using tree C_t
  5. Store votes in a global voting matrix V
  6. Add tree C_t to the forest
- The *Out Of Bag Error* for each sample x in X is the percentage of false votes stored in V

on the whole data set. Thus Bagging reduces variance and the possibility of overfitting on the training set. This may increase the accuracy of the forest because it improves its ability to generalize on new data examples. Another benefit is, that there is no need for a seperate evaluation data set. In [6] the concept of Bagging is described more in detail.

2.3.2.2 Proximities

For a data set with N samples and M features that is used on a *Random Forest*, one can compute a *Proximity Matrix*. Equation 2.7 shows the *Proximity Matrix* P which is a $N \times N$ matrix where p_{xy} and p_{yx} represent the proximity of samples x and y.

$$P = \begin{pmatrix} p(1,1) & p(1,2) & \cdots & p(1,N) \\ p(2,1) & p(2,2) & \cdots & p(2,N) \\ \vdots & \vdots & \ddots & \vdots \\ p(N,1) & p(N,2) & \cdots & p(N,N) \end{pmatrix} \tag{2.7}$$

Every time a data set is shown to the tree each sample will end up in a specific leaf node which is its terminal node. When two samples x and y share the same terminal node, the value of p_{xy} and p_{yx} is increased by one. In a Random Forest the proximities of all trees are summed up to get the proximity matrix for the forest. Samples with high proximity values are more similar to other parts of the data set than samples with low

proximity [8].

Algorithm 3 Proximity calculation [8]

- **Require:** Data set X with N samples and M features
- **Require:** Set of Trees T
- **Require:** Proximity matrix P of size $N \times N$
- Set all elements in P to 0
- For(each tree t in T),
  1. Put X down tree t, perform splits as defined in t
  2. When one sample x reaches a leaf node n:
     - For(each other sample y in n),
       - Set proximity $P(x, y) = P(x, y) + 1$
       - Set proximity $P(y, x) = P(y, x) + 1$

Algorithm 3 shows how the proximity matrix P is calculated for a given data set X. This proximity matrix contains the proximities for all trees within the forest. Thus proximities between two samples are only high if they share leaf nodes in many trees. P shows general proximities between all samples which means that the class labels are not taken into account. The average proximity $a(i)$ for a sample i cam be calculated using P. The average proximity $a(i)$ is the proximity of i to all other samples with the same class label. Thus it shows how similar a sample i is to the rest of its class.

$$a(i) = \sum_{j=1}^{n} \left(p^2(i, j) \cdot sameclass(i, j) \right) \tag{2.8}$$

Equation 2.8 shows how the average proximity $a(i)$ for a specific sample i can be calculated. Here $sameclass(i, j) = 1$ if i and j share the same class label and $sameclass(i, j) = 0$ otherwise. This means the average proximity $a(i)$ is the sum of proximities between i and all other samples j which belong to the same class as i. The average proximity provides information about similarities within a certain class. Samples with a high average proximity are more similar to the other samples in their class than samples with low average proximity [8]. The average proximities can be further used to find outliers which are samples with low similarity to the rest of its class.

2.3.2.3 Outliers

Random Forests provide a simple way to identify outliers in given data sets. Therefore, a data set with N samples and M features is shown to the forest. Proximitiy matrix P and the corresponding average proximities $a(i)$ for all samples i are calculated as described in section 2.3.2.2. Using the average proximity $a(i)$ the *Raw Outlier Measure* can be calculated for sample i.

$$r(i) = N/a(i) \tag{2.9}$$

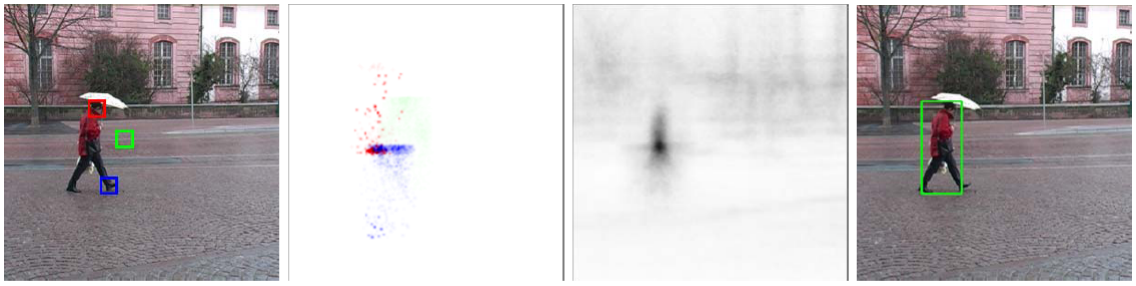
Equation 2.9 shows how the raw outlier measure $r(i)$ is calculated where N is the total number of samples and $a(i)$ is the average proximity of sample i. Using these raw outlier measures one can calculate the median value of all *Raw Outlier Measures* within a certain class c.

$$o(i) = \frac{r(i) - m(c)}{d(i)} \tag{2.10}$$

Equation 2.10 shows how the final *Outlier Measure* $o(i)$ is calculated for a specific sample i. Here $r(i)$ is the *Raw Outlier Measure* of sample i, $m(c)$ is the median value of *Raw Outlier Measures* for all samples with same class label as i and $d(i)$ is the absolute deviation of $r(i)$ from median $m(c)$. The final *Outlier Measure* for sample i is 0 if $r(i)$ lies on the median $m(c)$, -1 if $r(i)$ is below the median $m(c)$ and 1 if $r(i)$ lies above the median $m(c)$. Thus, samples with $o(i) = 1$ are possible candidates for outlying examples within their class. This means, that the outlier has a low similarity to the other samples within its class. This can help to find samples which are misslabeled, of bad quality or harmfull to the learned model. These outlier measures are also used in a Transfer Learning method which is part of our work.

2.3.3 Hough Forests

Invented by Gall and Lempitsky [15], Hough Forests are object detectors based on the Random Forest framework. A single Hough Forest is trained on positive and negative example patches in order to detect the object contained in the positive examples. These sample patches are extracted randomly from positive or negative training images. In addition to the usual training process for Random Forests, Hough Forests also learn center offsets. A center offset defines a two dimensional vector which denotes the offset from a certain patch to a possible center of an object. Once trained, the Hough Forest will produce



(a) – Original image with three sample patches emphasized

(b) – Votes assigned to these patches by the Hough forest

(c) – Hough image aggregating votes from all patches

(d) – The detection hypothesis corresponding to the peak in *(c)*

Figure 2.2: Detection Process of a Hough Forest, images taken from [15]

a hypothesis of object center positions when given a test image or image sequence. Figure 2.2 shows how Hough Forests detect an object. Image **(a)** shows three sample patches and image **(b)** shows their votes for the center where each vote has the same color as its corresponding patch. Image **(c)** shows the output image after all votes were made. The pixel with the highest density is the detected center. Image **(d)** shows the detection rectangle based on the output image from **(c)**.

2.3.3.1 Training Process

For training of a Hough Forest, small image patches p_i are extracted out of all training images. When given a positive example, N patches are taken at random positions inside the image and these patches are labeled as positive examples or foreground examples. Furthermore, the offset from the patch center to the center of the object from where it has been extracted needs to be stored as additional information. This offset o_i is the core element for the voting process of the detection phase and defines the Hough Forest. For images which contain multiple objects at once, N patches are sampled from each object using only random positions inside the bounding box of the corresponding object. Since the positive patches are sampled based on a bounding box instead of an accurate segmentation, there will be many positive patches which do not show parts of the object but parts of the background. While these samples are false positive training examples and thus lower the quality of training data, the underlying Random Forest can cope with this number of wrong labeled examples. When given a negative example, patches are drawn randomly from any position inside the image and are labeled as negative examples or background examples. For negative examples there is no need to store additional information like an offset.

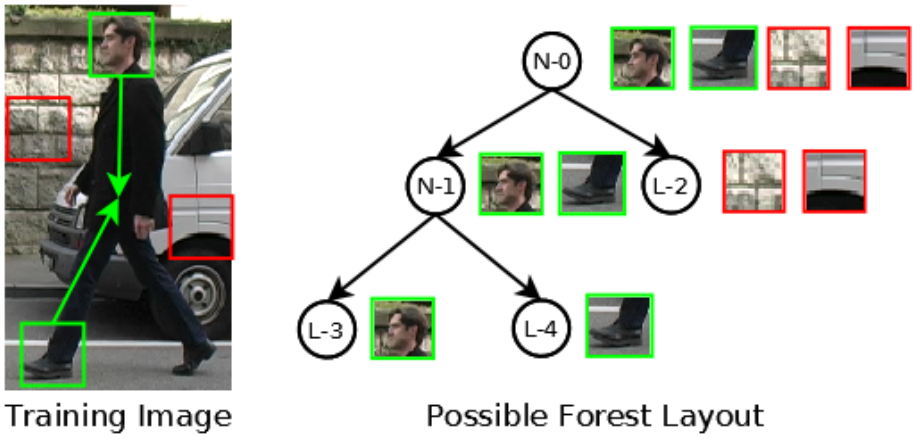


Figure 2.3: Illustration of Hough Forest training

Growing the trees: All decision trees inside the Hough Forest are grown, based on the information of all sample patches $P = \{p_1, p_2 \dots p_M\}$ where M is the total amount of positive and negative patches. Starting at the root node of a tree the patches P_n entering node n are split into two subsets. This split should decrease the uncertainty of either class distribution or offset direction. Since it may not be possible to decrease the class uncertainty and the offset uncertainty at once, only one of them is observed at once. A random choice decides whether the class uncertainty or the offset uncertainty is optimized. An exception of this random choice occurs when only positive examples are left. In this case always the offset uncertainty is decreased. Figure 2.3 schematically shows the process of growing a single tree. The red rectangles denote background patches and the green rectangles denote foreground patches including their offsets o_i visualized as green arrows. Node `N-0` is the root node which is trained on the complete set of patches. The split in `N-0` is chosen to decrease the class uncertainty and seperates the two background patches from the two foreground patches. The background patches end up in leaf node `L-2` while the foreground examples can be further seperated according to their offset to decrease offset uncertainty. `L-3` will now also store the offset of the head-patch whereas `L-4` will store the offset of the foot-patch.

Node Splits: Every time a set of patches P_n enters a new node, a specified amount of possible splits is generated randomly. Each split is defined by a pixel pair feature. According to the values of the chosen pixel pair feature a patch is passed to the left child node or to the right child node.

$$PixelPair(p(x,y),p(u,v)) < \tau \tag{2.11}$$

Equation 2.11 shows the decision criterion for passing a patch to the left child node. $PixelPair(p(x,y),p(u,v))$ denotes the value of a pixel pair feature and τ is a handicap value which is also chosen randomly for each possible split. If this criterion is not met, the patch is passed to the right child node. As mentioned before a split can either try to minimize the class uncertainty or to minimize the offset uncertainty. If a node should split the set of patches with respect to the class uncertainty, the split is chosen which minimizes $CLU(P_n)$ as shown in equation 2.12.

$$CLU(P_n) = |P_n| \cdot Entropy(class(P_n)) \tag{2.12}$$

In equation 2.12 $Entropy(class(P_n))$ denotes the standard entropy over the class labels for all examples in P_n. When the node should decrease the offset uncertainty, the split is chosen which minimizes $OU(P_n)$ as shown in equation 2.13.

$$OU(P_n) = \sum_{i=1}^{m} (o(p_i) - o(P_n))^2 \cdot f(p_i) \tag{2.13}$$

In equation 2.13 $o(p_i)$ is the offset vector for example p_i, $o(P_n)$ is the mean offset vector over all m examples in P_n, $f(p_i) = 1$ if p_i is a foreground patch and $f(p_i) = 0$ otherwise. Once a set of patches cannot be split any more, the node containing these patches is defined as a leaf node and stores the offsets of all foreground patches within this node. The reason to form a leaf node depends on standard decision tree stopping criteria as defined in section 2.3.1.

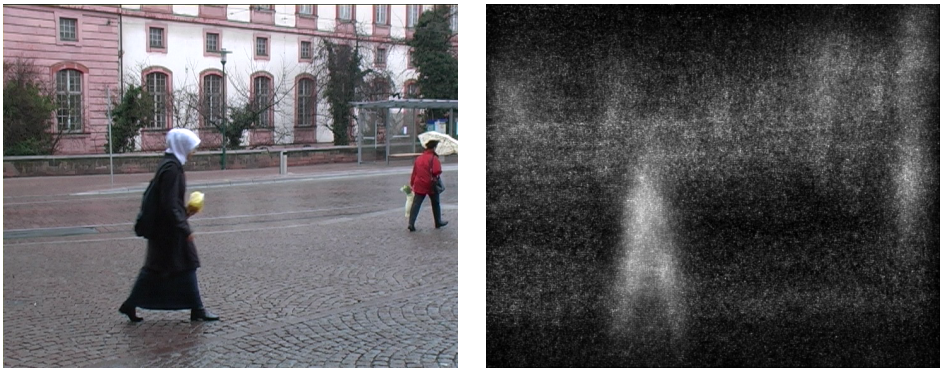
2.3.3.2 Detection of Objects

Once the Hough Forest is trained, it can be used to detect objects in one or more test images. Each test image is divided into a set of image patches. One patch is created for every pixel inside the image to create a patch set $P^t = \{p_1^t, p_2^t \dots p_l^t\}$ where l is the total number of patches. Again, the patch set is put down the forest and at every node the patches are further split according to the learned pixel pair tests. When one or more patches enter a leaf node, the offsets learned in the training phase are used to update the hypothesis H^t. Therefore, the position of a possible object centroid Pos_n^C is calculated for every offset.

$$Pos_n^C = coords(p_i^t) + o_n \tag{2.14}$$

In equation 2.14 $coords(p_i^t)$ is the center position of the image patch p_i^t and o_n is the offset with index n. For a certain position Pos_n^C, the pixel value at this position $H^t(Pos_n^C)$ is recalculated as shown in equation 2.15.

$$H^t(Pos_n^C) = H^t(Pos_n^C) + (\frac{w(o_n)}{\sum_i w(o_i)} \cdot N) \tag{2.15}$$



(a) Test Image from TUD-Pedestrian data set

(b) Corresponding Hough Image

Figure 2.4: A Hough Image Example

In equation 2.15 $\sum_i w(o_i)$ is the total sum of weights of all offsets which are present at this leaf node. N is the number of offsets in this node. Usually the weight of a single offset is one. Note that offsets will only be present inside leafs which contain at least one positive examples. Thus, pure background leafs will not update H^t at all. The final hypothesis H^t which is called a hough image looks similar to the example shown in figure 2.3.3.2 where the left image 2.4(a) shows the input image and 2.4(b) the corresponding hough image.

Pyramid Detection: Sample images used to train the Hough Forest do not have high variations in size since they can be scaled to a uniform size. In comparison test images may contain objects at different scales. Thus, the detector may fail in detecting an object when it is too small or too big.

Figure 2.5 illustrates the problem that occurs at different scales. The coloured rectan-

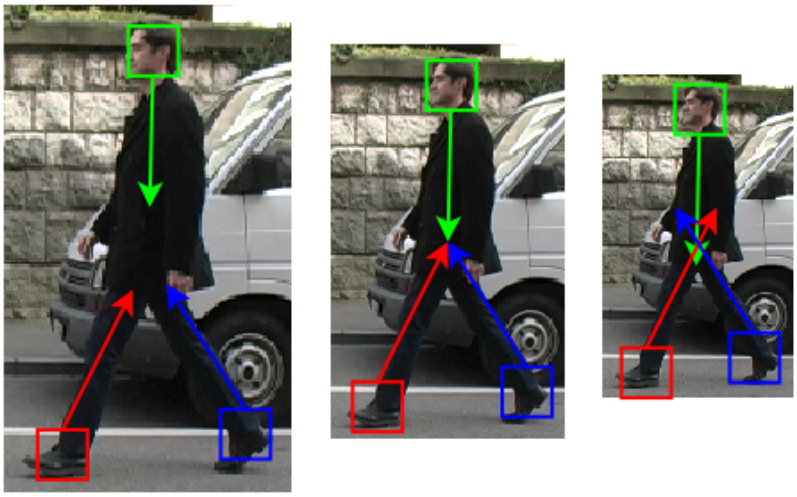


Figure 2.5: Detection diffusion at different object scales

gles denote sample patches and the arrows represent center offsets used for voting. The center image shows an optimal voting where the image is exactly the same scale as the images used during training phase. The hough votes will more likely produce an area of high density in the hough image. The left image is at a higher scale. Thus, the center offsets are too short to reach the center of the object which leads to a large area with intensities higher than background intensities but will not produce a local maximum at the object center. For smaller scale objects the hough votings will also have troubles to hit the center. In addition there will be less patches which can be classified as foreground patches.

A common way to solve this problem is to build an image pyramid where the original test image S is available at different scales S_i and to run the Hough Forest on every scale seperately. When this is done a single hypothesis will be available for every scale. According to figure 2.5 a hypothesis H_i for scale image S_i will show small center votes of high intensity when the detected object has the same scale as the objects used for training s_T. The more the scale of the detected object deviates from s_T the lower the intensity of the center vote will be. When multiple hypotheses H_i with different scales are available, an object will produce the most accurate center vote in the hypothesis where the object is shown at a scale which is closest to s_T. When this hypothesis is scaled so that the scale of the object is equal to s_T, this hypothesis will show the highest local maximum intensity at the objects center. In reality not every possible object scale can be covered because this would lead to very high calculation costs. Thus, only a few scales are used to detect very

small objects or very big objects relative to the training scale s_T. If the object we want to detect is half the size of the training object we need to doule the size of the test image. If the object we want to detect is twice the size of the training object we need to shrink the test image to half its original size. An important fact to consider is that a higher scale for a test image results in more test patches to vote for, whereas a smaller scale results in less patches. Thus, the output hypothesis H_i of every scale needs to be normalized according to its input scale.

$$H_i(x, y) = H_i(x, y) \cdot \frac{N \cdot M}{n \cdot m} \tag{2.16}$$

In equation 2.16 $N \times M$ denotes the pixel size of original input image S and $n \times m$ denotes the pixel size of the scaled image S_i. After all scale images S_i are normalized they can be further postprocessed to find a bounding box based detection hypothesis.

Postprocessing As mentioned before, the output of a Hough Forest with paramid detection are N hypotheses H_i in hough image form. Every hypothesis H_i represents one of N different scales which are chosen by the user. A postprocessing step is needed to calculate object bounding boxes for given hough images. These bounding boxes are the final output of a Hough Forest detector and show the positions and size for each detection.

Algorithm 4 Postprocessing method as defined by [15]

1. Calculate N Hough images $H = \{h_1 \dots h_N\}$ for N given scales.
2. Rescale hough images H so that all images are the same size.
3. Find the maximum pixel value $v_{max} \geq t$, where t is a defined threshold value, for all images $h_i \in H$ and add a detection rectangle d_t with centered at this point.
4. Set the detection rectangle size according to the original scale of the image which contained the maximum pixel value v_{max}.
5. Suppress voting elements for detection d_t by setting all pixels inside the detection rectangle to zero for all scale images.
6. Repeat these steps until no detection can be found any more.

Algorithm 4 schematically shows how detection rectangles can be found from a pyramid of hough images. The peak values inside the hough images define the centers of object detections. For each peak the size of the detection rectangle is defined by the original scale of its corresponding hough image. Detections with high peak values are found before

detections with lower ones. Thus, strong detections can occlude weaker ones which may make it hard to detect objects which are occluded by more than 50%.

2.3.4 Boosting

Boosting [45] is a method to create ensemble classifiers from a set of so-called weak classifiers. One single weak classifier may have very low predictive power but an ensemble of such classifiers has much more predictive power. Thus, the ensemble forms a strong classifier by combination of many weak classifiers. This strong classifier has higher accuracy than just one single weak classifier. For Boosting many weak classifiers are created. In an iterative process the best weak classifiers are selected according to their percentage of correct classifications. The final ensemble classifier combines the output of all selected weak classifiers to get a hypothesis.

Algorithm 5 shows the AdaBoost algorithm which is the boosting algorithm we used in our work. First a labeled training data set X needs to be available as well as a set of weak classifiers C. The weak classifiers may be constructed by a person or may be generated by another program. It is also possible to use a set of random weak classifiers. Since the quality of the final ensemble classifier depends on the quality of weak classifiers it is beneficial to have C containing as many weak classifiers as possible. During the boosting process all samples x in X are assigned a weight which specifies their importance. Initially all samples within a class are assigned the same weight. In each iteration the weak classifier is chosen which is able to minimize the classification error on X. This weak classifier C_i is added to the ensemble. After that the weights of all samples are updated. Those samples which were assigned the wrong class label by C_i are increased. In the next iteration step these samples will yield a higher error if they are assigned the wrong label. Thus, weak classifiers chosen at early stages are more general and should classify many samples correctly. Weak classifiers chosen at higher iteration steps are more specialized in classifying some few but difficult examples.

Algorithm 5 AdaBoost Algorithm [45]

- **Require:** A labeled training set X with m samples
- **Require:** Vector of true labels for X, $L = \{l_0 \dots l_N\}$ where labels are drawn from $Y = \{0, 1\}$.
- **Require:** A positive number of boosting iterations N
- **Require:** A set of weak classifiers C
- Assign each sample x in X a weight, forming weight vector $w^0 = \{w_1 \dots w_N\}$
- Initalize weight vector w to $w_i = \frac{1}{2k}$. k is the number of samples within the class of x
- **For** ($t = 0$ to N)

  1. Normalize all weights w_i to form a probability distibution by dividing through the total sum of weights

  $$w_i^t = \frac{w_i^t}{\sum_{i=1}^{m} w_i^t}$$

  2. For each weak classifier C_i, classify all samples in X and calculate the error $e(C_i) = \sum_{i=1}^{m} w_i^t \cdot e(x_i, C_i)$ where $e(x_i, C_i)$ is 0 when the samples is classified correctly and 1 otherwise
  3. Add the weak classifier with the lowest error $e(C_i)$ to the ensemble
  4. Calculate reweighting factor $\beta_t = \frac{\min e(C_i)}{1 - \min e(C_i)}$ where $\min e(C_i)$ is the error for the weak classifier, chosen in this iteration step
  5. Update the sample weights

  $$w_i^{t+1} = w_i^t \cdot \beta_t^{\epsilon_i}$$

  where ϵ_i is 0 if x_i is classified correctly and ϵ_i is 1 otherwise

- The final ensemble hypothesis $h(x)$ for example x is calculated as

$$h(x) = 1 \; if \; \sum_{j=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{j=1}^{T} \alpha_t \; , \; 0 \; otherwise$$

where $\alpha_t = log \frac{1}{\beta_t}$ and h_t is the hypothesis of the weak classifier added in iteration t

2.4 Transfer Learning

This section will discuss the main concepts of Transfer Learning and its different subcategories. Example methods for the different kinds of Transfer Learning will be mentioned and the Transfer Learning methods which are used for our work will be described in detail.

2.4.1 Introduction to Transfer Learning

Machine learning is a large field in computer science. There are many powerful algorithms which could be of great value in everyday life. The majority of these methods have one aspect in common. They need to be trained on some data to fulfill a given task. The accuracy of a machine learning method is highly dependent of the quality and quantity of this training data. Thus, one wants to use as many examples of high quality as possible. In this context, quality means that the sample improves the accuracy of the learner on new examples. These new examples are drawn from a so-called target domain D_T. Training samples should also be drawn from this domain to ensure good results. In most cases these data has to be carefully handcrafted to ensure high quality samples. This process is very time consuming. At this stage transfer learning comes into game. Transfer learning methods try to use additional data from a so-called source domain D_S. This domain is not equal to the target domain D_T but it is somehow related and similar. A source domain data set may contain samples which are outdated, weakly related to D_T, from a different distribution or even unlabeled. Thus, transfer learning methods provide a way to reuse data which is alread present. Transfer learning methods try to extract information from source domain D_S which is also relevant for target domain D_T. This additional information can improve the accuracy of a trained machine learning method. There are different types of transfer learning methods. The way how information is transferred depends on the type of the used transfer learning method.

Figure 2.6 shows the different types of transfer as defined by [39] . Here, the *target domain* denotes the domain where the learner should perform well at, whereas the *source domain* denotes the domain of the auxiliary data. In order to transfer knowledge from one to another domain, the number of features in these domains has to be equal. An example is the transfer from one vision task to another, where both are based on images of the same size. If target domain A is the domain of handwritten characters and source domain B the domain of handwritten digits, the transfer of information could yield good results even if there are only a few sample instances of every class in A because both domains may share some attributes like strokes or curves. The more different the source domain is

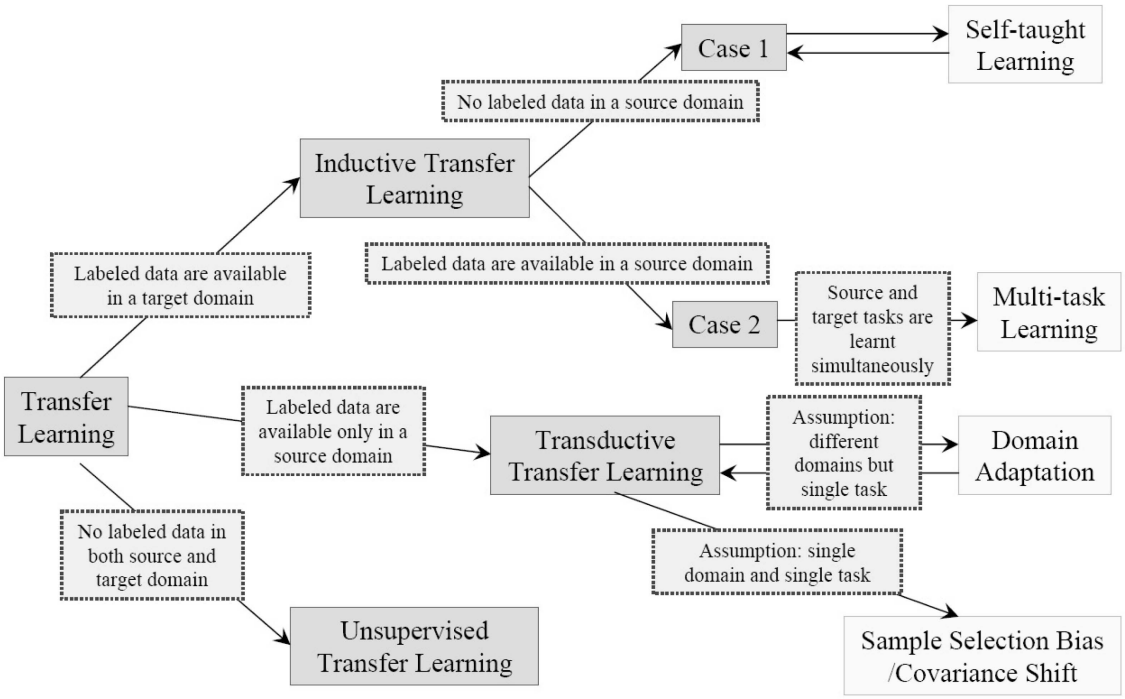


Figure 2.6: Types of Transfer (from [39])

from the target domain, the less impact transfer learning will have. In fact it is very likely to have some kind of negative transfer. This means that the use of a specific auxiliary data set lowers the performance on the target task instead of increasing it. [39]

2.4.1.1 Inductive Transfer Learning

Inductive Transfer Learning (ITL) [39] describes a case, where labeled data is available for a target domain D_T and some additional data for a so-called source domain D_S. The goal of this strategy is to increase the performance on the target domain by taking knowledge from the source domain data into account. In a real world task, only a few examples from the target domain may be available, and collecting new labeled examples is a very time consuming process. In object detection , for instance, one has to collect a large number of images that contain the detected object. In addition, valid labels have to be provided. This can be done by drawing object-bounding boxes, shapes or providing other information about contained objects.

The idea behind Inductive Transfer Learning is to use data that is already available, labeled and of a similar domain to improve learning without the cost of collecting new

examples. For ITL, the general assumption is that some parts of the source domain data may be similar to the target domain D_T and can be used as additional data. Samples from D_S, which are not similar to D_T can then be removed or assigned a lower weight to lower their impact. Methods of ITL usually have their specific way to find out which parts of D_S are similar to parts of D_T or a way to find out which parts of D_S can improve the classification on D_T. It is also very important for the use of ITL methods that the chance of improving the learned model is higher the stronger source domain D_S is related to target domain D_T. In this case more parts of D_S can be used effectively for the training process.

Examples for Inductive Transfer Learning are Transfer Boosting [10], Instance Weighting [22], Migratory-Logit (M-Logit) [30], Actively Transfer Knowledge (AcTraK) [46] and Transfer Learning for Support Vector Machines [51].

2.4.1.2 Multi-task Learning

Multi-task Learning (MTL) [9] is a special case of Inductive Transfer Learning. In MTL a learning task T^a is learned together with n other learning tasks $T^b = \left\{T_1^b, \ldots, T_n^b\right\}$. Examples D_a from T^a and examples D_b from T^b are combined to form a multi-task data set D_{ab}. The learner for task T^a is then trained on D_{ab} instead of D_a. This method can improve the accuracy on task T^a because the number of training examples increased. Tasks T^b should be related to the original task T^a and the representation of all tasks needs to be the same. This means that examples in all tasks need to share the same feature space. It is possible to use MTL on multiple image classification tasks where all images have the same size. It is not possible to use MTL to mix image classification with other reprentations like audio-signals.

Example of MTL: Task T^a is an image classification task. The goal is to identify all images which show motorcycles. The learner is trained on images of motorcycles and on imagages which show other objects. The resulting classifier produces false positive classifications for images which show bicycles because they are similar to motorcycles. In MTL one can simply add the task T_1^b of identifying bicycles to the multi-task scheme. Now the learner has the information that bicycles are not motorcycles. This may reduce the false positive rate and thus, improve the accuracy on the original task.

2.4.1.3 Self-Taught Learning

Self-Taught Learning (STL) is another special case of Inductive Transfer Learning. In this case there are no labels available for the auxiliary data drawn from source domain D_S. Since the examples from D_S are not labeled they cannot be used as additional training examples but may share some common attributes with the training examples from D_T. Thus, the auxiliary data is used to find a more expressive feature representation. The training data from D_T can be transformed to this new feature representation. This strategy can improve the performance of a machine learning algorithm on D_T as the quality of descriptive features is improved. Examples for STL methods are Transfer Learning with PCA [48] or Sparse Coding [27] [44] [37].

2.4.1.4 Transductive Transfer Learning

In Transductive Transfer Learning (TTL) labeled data X_s is only available for the source domain D_S. Data X_t drawn from target domain D_T is available but unlabeled. The examples in X_t are used to shift the distribution of examples in X_s towards the distribution of X_t. This approach is very usefull for real world tasks, especially when it is easy to collect additional samples for target domain D_T, although the costs of labeling are very high. An example for this scenario is the detection of pedestrians through a camera which is placed in a public site. Since the camera takes pictures on its own, additional examples of the target domain are available without any additional costs. This results in a high number of unlabeled examples, which increases the additional costs of labeling because of the high quantity.

Examples for TTL are Domain Adaption [12], Sample Selection Bias for Transfer Learning (SSB) [52] [18] and Covariate Shift [47] [43]. Jing Jiang presented a Survey on Domain Adaptation of Statistical Classifiers [21]. Here the term Domain Adaptation is used to denote Transductive Transfer Learning whereas there is also a method of Transductive Transfer Learning which is called Domain Adaption.

2.4.2 Comparison of TL to other Machine Learning concepts

As already mentioned one reason to use Transfer Learning (TL) is that data may be used for training which is already available. This data may have a weak relation to the domain of interest. This data X_s is called source data and is drawn from a source domain D_S. X_s can then be used by a TL algorithm in addition to a target training set X_t which is drawn from target domain D_T where D_T is the actual domain of interest. No additional costs

occur for the use of data set X_s because it is already labeled and ready to use. Besides Transfer Learning there are other concepts which have a focus on reducing labeling costs.

Semi-supervised Learning (SSL) is an approach where training data is split into a labeled part X_l and an unlabeled part X_u. Usually the number of examples in X_l is much lower than the number of examples in X_u to guarantee a low overall labeling effort. Different from TL the data sets X_u and X_l are both drawn from the domain of interest D_T but labels are missing for X_u. Many TL methods also perform SSL. For instance Transductive Transfer Learning algorithms are defined by using labeled data only for X_s and not for X_t. Self-taught Learning, on the other hand, is defined by using labeled data only for X_t and not for X_s.

In Unsupervised Learning (USL) no labels are available for training data. Thus, there is no cost for labeling at all when USL methods are used. Unsupervised Transfer Learning (UTL) is a subfield of Transfer Learning and combines USL with Transfer Learning methods. UTL defines methods where there are no labels available for X_t and X_s.

Multiple-instance learning (MIL) [13] is another approach to reduce the cost of labeling data examples. In MIL data examples are not labeled per instance. MIL introduces so called bags where one bag consists of multiple example instances. Each bag gets assigned exactly one label. In a two class scenario where the positive class has label 1 and the negative class has label 0 the labeling scheme is defined as follows. A bag is labeled 0 if all contained instances are negative as well. If one or more positive examples are contained in the bag it is assigned 1. Different from TL all data examples used for MIL are drawn from domain D_T. The bag labeling scheme which is used for MIL also results in some false positive training examples. This means that examples with a bag-label of 1 may have a true label of 0 but are labeled incorrectly.

2.4.3 Transfer Learning Algorithms

This section will describe the transfer learning algorithms which are used for our work. The following enumeration gives a overview of the used methods.

1. Transfer Boosting
2. Transfer Learning with Principal Component Analysis (PCA)
3. Transfer Learning with Sparse Coding

Transfer Boosting (TrAdaBoost) [10] is an Inductive Transfer Learning algorithm which is known to work well in combination whith various classifiers. Due to the fact that

Transfer Boosting builds an ensemble of classifiers, it is easy to combine it with Random Forest based methods. Transfer Learning with PCA and Transfer Learning with Sparse Coding [27] [44] [37] are both self-taught learning methods. They try to use information from a source domain to find more general features. These features are then used to describe the examples from the target domain. Both methods showed in [44] that they are able to improve the accuracy of support vector machines (SVM) and Gaussian discriminant analysis (GDA) on image classification tasks.

2.4.3.1 Transfer Boosting

Transfer Boosting (TrAdaBoost) [10] is an algorithm for Inductive Transfer Learning which is based on the AdaBoost algorithm [45]. Transfer Boosting takes training data X_t from a target domain D_T and additional data X_s from a source domain D_S. Both data sets are used to train a classifier where the target domain denotes the domain of interest. Information should be transferred from D_S to D_T to increase the amount of information about target domain D_T. Since examples from X_s may be used in addition to X_t it is beneficial if D_S is somehow related to domain D_T. The stronger the relation the more information can be transferred. The minimal relationship between X_t and X_s to make use of both data sets is that they need to share the same feature space. The goal of Transfer Boosting is to increase the amount of correct classifications on examples from domain D_T. Therefore, a validation set X_v can be used which is also drawn from D_T.

Transfer Boosting Process Transfer Boosting is an iterative process where a weak classifier is added to an ensemble in every iteration t. A weak classifier can be every type of classifier. For Transfer Boosting a target training set X_t and a source training set X_s are used. X_t is drawn from target domain D_T which is the domain the final classifier should perform well at. X_s is drawn from source domain D_S.For T iterations the first step of the Transfer Boosting process is the initial weighting of all samples from X_t and X_s.

$$w^0(x_i) = \frac{1}{2 \cdot \sum_{n=1} c(x_i, x_n)} \tag{2.17}$$

Equation 2.17 shows how the sample weights are initialized. Here $c(x_i, x_n) = 1$ if x_i and x_n share the same label and $c(x_i, x_n) = 0$ otherwise. Thus, the total weight sum of all samples $WAll = 1$ which makes the initial set of all weights w^0 a probability distribution. At the beginning of each iteration step t sample weights are used to calculate a probabilistic weight p_i^t for each sample x_i.

$$p_i^t = \frac{w^t(x_i)}{\sum_{n=1} w^t(x_n)} \tag{2.18}$$

Equation 2.18 shows how p_i is calculated which is the weight of corresponding sample x_i divided by the sum of all weights w^t. Now weight vector $p^t = \{p_1^t, p_2^t \dots p_{n+m}^t\}$ is used as sample weights to train a weak classifier, where n is the number of samples in X_s and m is the number of samples in X_t. After the new weak classifier C^t is trained the classifier is evaluated on data sets X_t and X_s.

$$e^t = \sum_{i=n+1}^{n+m} \frac{w_i^t \cdot \left|l_i - c_i^t\right|}{\sum_{i=n+1}^{n+m} w_i^t} \tag{2.19}$$

Equation 2.19 shows how classification error e^t for each example x_i is calculated where l_i is the true label of sample x_i, c_i^t is the voted class label for sample x_i in this iteration and w_i^t is the weight of sample x_i. Thus, the error is simply the sum of weights of false classified example divided by the total sum of weights of all samples.

After error e^t has been calculated the sample weights are updated in each iteration t. This is the most important step for Transfer Boosting because the weights of samples which can improve the performance will be increased whereas the weights for samples which may harm the perfromance will be decreased. Therefore, two weight update factors β and β^t are calculated.

$$\beta_t = \frac{e^t}{1 - e^t} \tag{2.20}$$

$$\beta = \frac{1}{1 + \sqrt{2\, ln\, n/N}} \tag{2.21}$$

Here, N is the number of transfer boosting iterations and n is the number of examples in auxiliary data set X_s. Using these values, weights for examples of X_t and X_s are updated seperately. For examples in X_s which are wrong classified in this iteration the update looks as defined by equation 2.22.

$$w_i^{t+1} = w_i^t \cdot \beta^{|l_i - c_i^t|} \tag{2.22}$$

In equation 2.22 $\left|l_i - c_i^t\right| = 0$ if example x_i is classified correctly and $\left|l_i - c_i^t\right| = 1$ otherwise. Thus, the weights of false classified samples from X_s are decreased relative to the maximum number of iterations whereas the weights for correct classified samples from

X_s are not updated at all. For examples in X_t the weight update is done as shown in equation 2.23.

$$w_i^{t+1} = w_i^t \cdot \beta_t^{-|l_i - c_i^t|} \tag{2.23}$$

Thus, the weights of wrong classified examples in X_t are increased to raise the chance that they will be classified correctly in the next iteration step. Correctly classified examples in X_t are not updated at all. An important fact is that error e^t needs to be larger than zero and smaller than 0.5 in order to get a proper weight update. When $e^t = 0$ samples in X_s do not seem to harm the performance on X_t and thus no reweighting can be applied. When $e^t \geq 0.5$ the weight updates would not work as intended and thus the iterative process needs to be aborted. A summary for the TrAdaBoost algorithm is shown in Algorithm 6.

2.4.3.2 Transfer Learning with PCA

Principal Component Analysis (PCA) [48] is a dimensionality reduction technique which is used for tasks as image compression. While it is able to reduce the dimensions of a given data set it does also find patterns inside the data. These patterns can be used to transform the data and to improve seperation of different classes. If a training set X_t from target domain D_T is used to train a classifier, PCA is used to calculate a transformation matrix F_{pca}. This transformation matrix encodes patterns within X_t and is used to transform X_t into a new form X_f. For Transfer Learning another data set X_s which is drawn from source domain D_S is used in addition to calculate F_{pca}. Thus, information from X_s is transferred to X_t because patterns encoded in F_{pca} are the most significant patterns found in $X_{ts} = X_t \cup X_s$.

First the mean vector $M = \{m_1, m_2 \dots m_K\}$ is calculated where K is the number of features used in $X_{ts} = X_t \cup X_s$ and m_i is the mean value for feature i.

$$m_i = \frac{\sum_{j=1}^{N} X_{ts}(i,j)}{N} \tag{2.24}$$

In Equation 2.24, i is the index of the feature and j is the index of an example inside X_{ts}. N is the total number of examples in combined data set X_{ts}. Mean vector M can then be used to normalize X_t by dividing each feature value by its corresponding mean value.

Algorithm 6 TrAdaBoost Algorithm from [10]

- **Require:** A labeled auxiliary set X_s from domain D_S with N samples
- **Require:** A labeled training set X_t from domain D_T with M samples
- **Require:** A unlabeled validation set X_v from domain D_T
- **Require:** A positive number of boosting iterations I
- **Require:** A weak classifier C
- **Require:** Vector of true labels for X_s and X_t, $l = \{l_0 \dots l_{N+M}\}$ where the first N elements belong to X_s and the latter M elements to X_t
- Assign each sample x in X_t and X_s a weight, forming weight vector $w^0 = \{w_1 \dots w_{N+M}\}$ where the first N elements belong to X_s and the latter M elements to X_t
- Initalize weight vector w
- **For** ($i = 0$ to I)
  1. Calculate distribution vector p^i by dividing each weight through the sum of all weights contained in w^i
  2. Train a weak classifier C_i on the combined set X_{ts} with use of distribution p^i and get vector $c^i = \{c_0 \dots c_{n+m}\}$, which contains the classification results for X_{ts}
  3. Calculate error e^i for classifications on X_t using the classification results from c^i

$$e^i = \sum_{j=n+1}^{n+m} \frac{w_j^i \cdot \left| l_j - c_j^i \right|}{\sum_{j=n+1}^{n+m} w_j^i}$$

  4. Calculate $\beta_i = e^i / \left(1 - e^i\right)$ and $\beta = 1 / \left(1 + \sqrt{2\, ln\, N / I}\right)$
  5. Set weights $w_j^{i+1} = w_j^i \cdot \beta^{\left|l_j - c_j^i\right|}$ if $j \leq N-1$ and $w_j^{i+1} = w_j^i \cdot \beta_i^{-\left|l_j - c_j^i\right|}$ otherwise
- Get final hypothesis

$$h_f(x) = 1 \; if \prod_{i=[I/2]}^{I} \beta_i^{-c_i(x)} \geq \prod_{i=[I/2]}^{I} \beta_i^{-1/2}, 0 \; otherwise$$

$$X_n = \begin{pmatrix} X_t(1,1)/m_1 & X_t(1,2)/m_2 & \cdots & X_t(1,N)/m_N \\ X_t(2,1)/m_1 & X_t(2,2)/m_2 & \cdots & X_t(2,N)/m_N \\ \vdots & \vdots & \ddots & \vdots \\ X_t(M,1)/m_1 & X_t(M,2)/m_2 & \cdots & X_t(M,N)/m_N \end{pmatrix} \tag{2.25}$$

Where $X_t(x, y)$ denotes the value of feature y for example x and M is the total number of examples in X_t. The next step is, to calculate covariance matrix C.

$$C = \begin{pmatrix} c(1,1) & c(1,2) & \cdots & c(1,N) \\ c(2,1) & c(2,2) & \cdots & c(2,N) \\ \vdots & \vdots & \ddots & \vdots \\ c(M,1) & c(M,2) & \cdots & c(M,N) \end{pmatrix} \tag{2.26}$$

Where $c(x, y)$ is the covariance for features x and y. Note that $c(x, y)$ is equal to $c(y, x)$ an thus does not need to be calculated twice. Equation 2.27 shows how covariance $c(x, y)$ is calculated.

$$c(x,y) = \frac{\sum_{i=1}^{M}(X_t(i,x) - m_x)(X_t(i,y) - m_y)}{M-1} \tag{2.27}$$

In Equation 2.27, $X_t(a, b)$ is the value of feature b for example a. Covariance matrix C will be of format $N \times N$ where N is the number of features used in X_t and X_s. The next step is the calculation of eigenvectors and eigenvalues of matrix C. As a result we get a matrix of eigenvectors E with eigenvectors as rows and a vector of eigenvalues e. The eigenvector with the highest eigenvalue is the so-called principle component which means the eigenvector which describes the data best. Matrix E needs now to be sorted by the eigenvalues starting with the principal component, and also each eigenvector has to be converted to unit-vector length if not done already. Now the first $R \leq N$ eigenvectors can be used as a transformation matrix. This transformation matrix will be denoted as F_{pca}. Using F_{pca}, one can transform a training data set X_t and every evaluation data set before it is passed to the learner. As one can se, matrix E can have a maximum of N eigenvectors which can not be exceeded. Thus, F_{pca} can also have N rows at maximum. Once the transformation matrix is calculated it can be stored for later use. The proccess of data transformation for training data X_t is described by equation 2.28.

$$X_f = F_{pca} \times X_m \tag{2.28}$$

Equation 2.28 shows the simple transformation step where F_{pca} is the transformation matrix with the selected eigenvectors as rows and X_m is the normalized form of data X_t as shown in equation 2.25 with data examples as columns. This step is done once during training for training data set X_t. After that the learner is trained on the transformed data X_f instead of being trained on X_t. For the trained learner every evaluation data set X_{test} also needs to be transformed befor being passed to the learner. Note that in this case, X_m represents the normalized form of X_{test} instead of X_t.

Dimensionality Reduction PCA is commonly used as a technique for the reduction of dimensionality. While it is able to find patterns in a given data set it can also be used to reduce the dimensions of a data set. Since the eigenvectors in X_f are sorted according to their descriptive power, the dimensions can be effectively reduced, minimizing the loss of performance. Thus, it is possible to use PCA as a technique for transfer learning and to decrease the dimensionality of the given data set simultaneously.

2.4.3.3 Transfer Learning with Sparse Coding

Sparse Coding [27] [44] [37] has the goal to represent every example in a data set by a *sparse vector* of *base vectors*. The base vectors encode strong patterns within the given data set. This means one base vector represents one single pattern. A sparse vector is a linear combination of base vectors where most elements are zero. This is possible because the combination of only a few base vectors is needed to represent one single example. Sparse vectors are also called activation vectors.

Figure 2.7 (from [44]) shows base vectors which are learned for a digit classification task. The base vectors on the right side of figure 2.7 show typical strokes for handwritten digits.

For Transfer Learning some target training data X_t from target domain D_T and source training data X_s from source domain D_S are used. The calculation of basis vectors is based on $X_{ts} = X_t \cup X_s$ instead of X_t. The intention is to find more general basis vectors by providing more information. The actual classifier is trained on A_t where a_t^i is the activation vector for x_t^i with respect to the basis vectors. Since X_s was involved in the calculation of basis vectors some information is transferred from X_s to X_t.

For Sparse Coding we first take target training set X_t and auxiliary training set X_s to create a combined set $X_{ts} = X_t \cup X_s$ with a total of M examples and N features. This set can then be used to find sparse feature representations which consist of a set

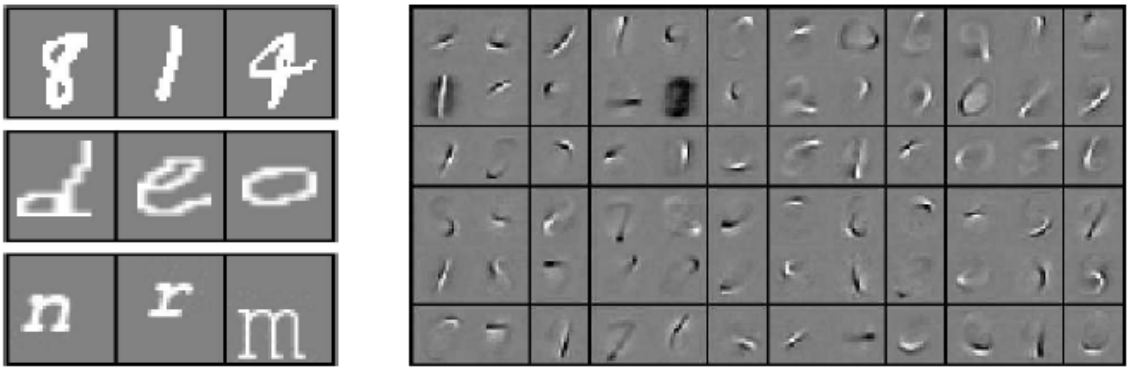


Figure 2.7: Example of Sparse Features for Digit Classification

of base vectors $b = \{b_1, b_2, \ldots, b_k\}$ and a set of activation vectors $a = \{a_1, a_2, \ldots, a_k\}$. Every example in X_{ts} can then be represented by a combination of basis vectors. The corresponding activation vector represents the weights for every basis vector.

As proposed in [27] the basis vectors and activation vectors are found by iteratively optimizing only one of them where the number of iterations I may be chosen by the user. First the basis vectors are initialized to random values. In each iteration step activations are calculated using the Feature-sign search algorithm. Then, the basis vectors are updated to increase their expressive power. After all I iterations are done, the basis vectors can be stored in matrix form as basis matrix B. When training a learner, the training data set X_t is transformed into its activation vector form A_t with respect to the basis matrix B. A test data set X_{test} also needs to be transformed into the activation vector form. An overview of the whole process is shown in algorithm 7.

Feature-sign search algorithm To find good activation vectors the Feature-sign search algorithm can be used as proposed in [27]. When given a fixed set of basis vectors $b = \{b_1, b_2, \ldots, b_k\}$ with k basis vectors which are stored in basis matrix B a combination of these basis vectors needs to be found for every example x_i in X_{ts}. For each x_i process optimization step as shown in equation 2.29.

$$minimize_{a_i}(\|x_i - Ba_i\|^2 + \gamma \|a_i\|_1) \tag{2.29}$$

In Equation 2.29, γ is a constant which is chosen by the user and a_i is the desired activation vector for example x_i.

The complete Feature-sign search algorithm as defined in [27] is shown in algorithm

Algorithm 7 Sparse Coding algorithm

- **Require:** A target training set X_t
- **Require:** A auxiliary training set X_s
- **Require:** A target test set X_{test}
- **Require:** A number of iterations I and a number of desired basis vectors k

1. Initialize k basis vectors with respect to combined training set $X_{ts} = X_t \cup X_s$ and store them in basis matrix B
   - For I iterations:
     (a) Calculate activation vectors A for X_{ts}
     (b) Update basis vectors B with respect to A
2. Train a learner on A_t which is the activation vector matrix for X_t with respect to B
3. Evaluate learner on A_{test} which is the activation vector matrix for X_{test} with respect to B

8. When given an initial activation vector a_i which is of zero-vector form the algorithm iteratively updates a_i until an optimal solution is found.

Finding basis vectors Given a fixed set of activation vectors $a = \{a_1, a_2, \ldots, a_k\}$ which are stored in activation matrix A one can use the lagrange dual algorithm, as proposed in [27] to find the corresponding k basis vectors. The optimization problem which needs to be solved in this case is shown in equation 2.30 .

$$minimize \, \|X_{ts} - BA\|_F^2 \tag{2.30}$$

In Equation 2.30, X_{ts} is the combined training set without labels, B is the matrix which holds the basis vectors and A is the activation matrix. The first step to find the optimal bases is to optimize the lagrange dual $D(\boldsymbol{\lambda})$ where each λ_i in $\boldsymbol{\lambda}$ is a dual variable.

$$D(\boldsymbol{\lambda}) = trace(X_{ts}^T X_{ta} - X_{ts} A^T (A^T A + diag(\boldsymbol{\lambda})^{-1})(X_{ts} A^T)^T - c \cdot diag(\boldsymbol{\lambda})) \tag{2.31}$$

In Equation 2.31, $diag(\boldsymbol{\lambda})$ is the diagonal matrix with elements of $\boldsymbol{\lambda}$ and c is the constraint with $\sum_{i=1}^{k} B^2(i,j) \leq c$. This optimization can be solved by using conjugate

Algorithm 8 Feature-sign search algorithm [27]

- **Require:** The matrix of basis vectors B
- **Require:** A example x_i, taken from X_{ts}
- **Require:** A activation vector a_i which needs to be optimized
- **Require:** Sign vector θ which holds the signs of basis vectors where $\theta_j =\in \{-1, 0, 1\}$
- **Require:** The active set S which holds the indices of choosen basis vectors

1. Initialize $a_i = \mathbf{0}$, $\theta = \mathbf{0}$ and $S = \{\}$
2. Choose the basis vector which maximizes $\left|\frac{\partial \|x_i - Ba_i\|^2}{\partial a_i(j)}\right|$, where j is the index of a basis vector
   - If $\frac{\partial \|x_i - Ba_i\|^2}{\partial a_i(j)} > \gamma$ then set $\theta_j = -1$ and add basis vector index j to active set S.
   - If $\frac{\partial \|x_i - Ba_i\|^2}{\partial a_i(j)} < -\gamma$ then set $\theta_j = 1$ and add basis vector index j to active set S.
   - Otherwise go to step 2 and choose another basis vector.
3. The Feature-sign step:
   - Create matrix $\hat{B}$ which contains the basis vectors defined in active set S
   - Create vector $\hat{a}$ which contains the elements of a_i for the active set S
   - Create sign vector $\hat{\theta}$ which contains the elements of θ for the active set S
   - Calculate $\hat{a}_{new} = (\hat{B}^T\hat{B})^{-1}(\hat{B}^T x_i - \gamma\hat{\theta}/2)$
   - Perform a discrete line search from $\hat{a}$ to $\hat{a}_{new}$. At every point where the sign of a coefficient changes and at the final point $\hat{a}_{new}$ calculate the objective value $\frac{\partial \|x_i - Ba_i\|^2}{\partial a_i(j)}$. Choose the point with the lowest objective value and update a_i
   - Remove al zero values from a_i
4. Optimality checks:
   - If $\frac{\partial \|x_i - Ba_i\|^2}{\partial a_i(j)} + \gamma \cdot sign(a_i(j)) = 0$ and $a_i(j) \neq 0$ proceed to next step, otherwise go to step 3 and do not add any activations
   - If $\left|\frac{\partial \|x_i - Ba_i\|^2}{\partial a_i(j)}\right| \leq \gamma$ and $a_i(j) = 0$ return a_i as optimal solution, otherwise go to step 2

gradient or Newton's method. The calculation of the gradient and the Hessian are shown in equations 2.32 and 2.33 .

$$\frac{\partial D(\boldsymbol{\lambda})}{\partial \lambda_i} = \left\| X_{ts} A^T (AA^T + diag(\boldsymbol{\lambda})^{-1} u_i) \right\|^2 - c \tag{2.32}$$

$$\frac{\partial^2 D(\boldsymbol{\lambda})}{\partial \lambda_i \partial \lambda_j} = -2((AA^T + diag(\boldsymbol{\lambda}))^{-1}(X_{ts}A^T)^T X_{ts} A^T (AA^T + diag(\boldsymbol{\lambda}))^{-1})_{i,j}((AA^T + diag(\boldsymbol{\lambda}))^{-1})_{i,j} \tag{2.33}$$

In Equation 2.33, u_i is the unit vector with index i. Once the optimal solution for $D(\boldsymbol{\lambda})$ is found, the matrix of basis vectors B can be calculated in one step by using the equation shown in 2.34 .

$$B^T = (AA^T + diag(\boldsymbol{\lambda}))^{-1}(X_{ts}A^T)^T \tag{2.34}$$

Where $diag(\boldsymbol{\lambda})$ denotes the diagonal matrix with respect to the optimal solution found for $D(\boldsymbol{\lambda})$.

2.5 Chapter Summary

This chapter described the basic concepts which were used for our work. Section 2.1 describes two kinds of image features which are methods to extract information from digital images. Section 3.3 gives a short introduction to the field of object detection. Section 2.3 describes basic machine learning concepts such as Random Forests, Hough Forests and Boosting. Section 2.4 gives an introduction to Transfer Learning (TL), describing the main categories of TL. The Transfer Learning methods which we are using are discussed in detail.

Chapter 3

Transfer Learning for Random Forest based Methods

Contents

This chapter will describe our different methods to combine Transfer Learning with Random Forest based methods. The chapter is divided into three sections. The first section describes Transfer Learing for Random Forest which can be applied on all machine learning tasks. The second section will describe Tranfer Learning for Random Forests in an image classification scenario. The third section will describe Transfer Learning for Hough Forests which are Random Forest based object detectors.

3.1 General Machine Learning

The Transfer Learning methods discussed in this section can be used on every machine learning task when Random Forest based classifiers are used. In addition to a training data set X_t from target domain D_T another data set X_s from a source domain D_S needs to be availabe so that information can be transferred from D_S to D_T.

3.1.1 Transfer Learning with Outlier Detection

As mentioned in section 2.3.2.3, Random Forests have a strong, built in mechanism to find outliers in a given set of data. Outliers denote examples which may lie outside the common distribution of a certain class. A single example may be an outlier because it is a very special and rare instance of the corresponding class but it can also be an outlier because it is labeled incorrectly and does not belong to this class. For transfer learning we will use this outlier measure to find examples in auxiliary data X_s which are different from the examples in training data X_t. All outliers are removed from X_s forming a new set X_o which does not contain the outliers. After this step the classifier can be trained using X_t and X_o.

Outlier Suppression After the outlier measures are obtained for all samples in $X_{ts} = X_t \cup X_s$ those examples which are declared outliers are removed from X_s to form X_o. If the number of examples in X_s is greater than the number of examples in X_t, the process of outlier suppression may be divided into smaller batches. The use of small batches ensures that data in X_s does not corrupt the outlier measure. While a single example from X_s may be regarded an outlier in a certain leaf, it may not be an outlier when there are many similar examples in X_s which end up in the same node. Thus, we want X_t to be larger in number of examples than X_s so that the measurment of outlierness is mostly based on X_t. Therefore, X_s can be divided into k smaller sets $\left\{X_s^1, X_s^2 \dots X_s^k\right\}$. The evaluation of outliers is then processed for every pair (X_t, X_s^j). The examples from X_s^j which are not declared outliers are then added to X_o. Finally, the classifier is trained on $X_{to} = X_t \cup X_o$. The summarized algorithm for this approach is shown in Algorithm 9.

Memory saving solution for outlier calculation: The calculation of outlyingness incorporates the calculation of a $N \times N$ proximity matrix where N is the number of examples in a given training set X_t. When X_t contains a high number of examples this may cause memory issues because the proximity matrix could not fit into memory.

Algorithm 10 shows a way to solve this problem. Instead of calculating a proximity matrix we store only the final leaf node id or pointer for each sample and tree. This results in a $T \times N$ matrix Y. This matrix Y can be used to look up proximities when needed. To calculate the outlier measure we first lop over all examples to calculate their average proximity a_i and raw outlier measure r_i. After that is done class medians of raw outlier measures can be calculated and the final outlier measure for each example x_i can be set

Algorithm 9 Transfer Learning with Outlier Suppression

- **Require:** A target training set X_t
- **Require:** A auxiliary training set X_s

1. Train classifier on X_t to build a model for prediction
2. Divide X_s into smaller batches $\left\{X_s^1, X_s^2 \dots X_s^k\right\}$ so that every batch X_s^i has m samples, where $m < M$ and M is the total number of examples in X_t
3. For every pair $X_{ts}^j = X_t \cup X_s^j$
   - Evaluate classifier on pair X_{ts}^j
   - Calculate outlier measures
   - Add all examples from X_s^j which are not declared an outlier to X_o
4. Retrain classifier on X_o

within another loop over all examples. This algorithm needs less memory. The drawback is that it takes longer because proximities cannot be stored and need to be calculated by looping over all examples each time they are needed.

Drawbacks of this method: The major drawback of this approach is that there is no mechanism to measure whether or not an outlier may harm the performance. In comparison, transfer boosting reweights examples according to the error rate on training set X_t. Thus, the quality of the final solution is strongly dependent on the quality of data set X_t. If some important parts of the target distribution are missing in X_t, they will also not be taken from X_s. Even if X_s contains these missing parts they will very likely be regarded as outlying examples and will be discarded. Thus, the use of this method may imply a tradeoff between possible negative transfer and possible waste of usefull information.

3.1.2 Transfer Boosting

Transfer Boosting as described in section 2.4.3.1 can be used in combination with Random Forests by simply using single Decision Trees as weak classifiers. Then, the final classifier found by the Transfer Boosting process is a Random Forest with I trees where I is defined by the number of Transfer Boosting iterations. Algorithm 11 illustrates the adapted version of the TrAdaBoost algorithm for Random Forest classifiers.

Algorithm 10 Memory saving solution for outlier calculation

- **Require:** A labeled data set X_t with N samples and C classes
- **Require:** A number of trees T inside a forest
- **Require:** A initial set of raw outlier measures R with N elements $r_i = 0$
- **Require:** A initial set of outlier measures O with N elements $o_i = 0$

1. Show X_t to a trained or untrained Random Forest F
2. For each sample x_i in X_t store the final leaf node id for each tree. Result is a $T \times N$ matrix Y.
3. For $(i = 0 : to : N)$
   - Set average proximity $a_i = 0$
   - For $(j = 0 : to : N)$
     - Calculate proximities $p(i, j)$ by the use of Y.
     - Set $a_i = a_i + p(i, j)^2$ if $i \neq j$ and if both share the same class label
   - Set raw outlier measure $r_i = N/a_i$
4. Calculate class medians m_c for each class c
5. For $(i = 0 : to : N)$
   - Set final outlier measure $o_i = \frac{r_i - m_c}{d_i}$ where d_i is the absolute deviation from r_i to m_c

Drawbacks of this method: Since the Transfer Boosting process is stopped when the classification error e^i within an iteration is $e^i = 0$ or $e^i \geq 0.5$ this method may be problematical on specific machine learning tasks. When a single Decision Tree classifier C_i can predict the examples in X_t with $e^i = 0$ the process would be stopped. To avoid this effect the stopping criteria for a single Decision Tree can be adjusted so that a smaller tree is generated with lower predictive power. When $e^i \geq 0.5$ the task is too complex to get reasonable classification results from a single Decision Tree. In this case a Random Forest with R trees can be used as weak classifier C_i instead of a single Decision Tree. The resulting classifier would then be an Ensemble of Random Forest where R needs to be chosen that $0 < e^i < 0.5$ if possible.

Algorithm 11 Adapted TrAdaBoost Algorithm for Random Forests

- **Require:** A labeled auxiliary set X_s from domain D_S with N samples
- **Require:** A labeled training set X_t from domain D_T with M samples
- **Require:** A unlabeled validation set X_v from domain D_T
- **Require:** A positive number of boosting iterations I
- **Require:** Vector of true labels for X_s and X_t, $l = \{l_0 \dots l_{N+M}\}$ where the first N elements belong to X_s and the latter M elements to X_t
- Assign each sample x in X_t and X_s a weight, forming weight vector $w^0 = \{w_1 \dots w_{N+M}\}$ where the first N elements belong to X_s and the latter M elements to X_t
- Initalize weight vector w
- **For** ($i = 0$ to I)
  1. Calculate distribution vector p^i by dividing each weight through the sum of all weights contained in w^i
  2. Train a Decision Tree classifier C_i on the combined set X_{ts} with use of distribution p^i and get vector $c^i = \{c_0 \dots c_{n+m}\}$, which contains the classification results for X_{ts}
  3. Calculate error e^i for classifications on X_t using the classification results from c^i

$$e^i = \sum_{j=n+1}^{n+m} \frac{w_j^i \cdot \left| l_j - c_j^i \right|}{\sum_{j=n+1}^{n+m} w_j^i}$$

  4. Calculate $\beta_i = e^i / \left(1 - e^i\right)$ and $\beta = 1 / \left(1 + \sqrt{2\, ln\, N/I}\right)$
  5. Set weights $w_j^{i+1} = w_j^i \cdot \beta^{\left|l_j - c_j^i\right|}$ if $j \leq N-1$ and $w_j^{i+1} = w_j^i \cdot \beta_i^{-\left|l_j - c_j^i\right|}$ otherwise
  6. Add the trained Decision Tree to the Random Forest
- Get final hypothesis

$$h_f(x) = 1\, if \prod_{i=[I/2]}^{I} \beta_i^{-c_i(x)} \geq \prod_{i=[I/2]}^{I} \beta_i^{-1/2}, 0\, otherwise$$

3.2 Image Classification

For image classification Transfer Learning with Outlier Detection and Transfer Boosting can be used as decribed in section 3.1. In addition, information about example images needs to be provided by some sort of image feature such as pixel pairs or Haar Features. This section will describe how Transfer Learning with PCA and Sparse Coding can be used to calculate image feature representations. These image feature representations include information about target domain D_T as well as information about source domain D_S. Thus, information is transferred from D_S to D_T.

3.2.1 Transfer Learning with PCA

Principal Component Analysis (PCA) can be used to form feature representations for image data. When used for Transfer Learning a data set X_t from target domain D_T and a auxiliary data set X_s from source domain D_S are used to calculate these feature representations. Knowledge is transferred from D_S to D_T and encoded in the new feature representation. Algorithm 12 illustrates the method. The data in X_t and X_s is available in form of digital images. The images from both data sets are used to calculate the PCA transformation matrix F_{pca}. This matrix can then be used to transform every given example x_i into a new form x_i^f. For this transformation x_i is provided in form of a raw pixel value vector with n elements. Since the dimension of data can be reduced using PCA the resulting feature vector x_i^f has $m \leq n$ elements. Transforming all examples from X_t we get X_t^f. Since the classifier is trained only on X_t^f, labels need to be available for X_t but not for X_s. Every example that is shown to the classifier during training and testing needs to be transformed first. In our case a Random Forest is used as classifying method. Inside this forest every node split is based on transformed values x_i^f instead of x_i.

3.2.2 Sparse Coding

Algorithm 13 illustrates how we combined Sparse Coding with Random Forests. Image data sets X_t and X_s are used to calculate basis vectors B. For this calculation the raw pixel value vector x_i of every example image is used. The number of basis vectors in B is not dependent on the size of x_i. Thus, it is possible to decrease the number of basis vectors for high dimensional images and to increase the size of basis vectors if image dimensions are very small. Once B is calculated every example image x_i can be represented by an activation vector a_i. The size of a_i is exactly the number of base vectors as a_i represents a

Algorithm 12 Transfer Learning with PCA

- **Require:** A labeled training set X_t from domain D_T with M sample images
- **Require:** A unlabeled auxiliary set X_s from domain D_S with N sample images
- **Require:** A unlabeled validation set X_v from domain D_T

1. Combine X_t and X_s to form X_{ts}
2. Calculate PCA transformation matrix F_{pca} for X_{ts} as described in section 3.2.1
3. Use F_{pca} to transform X_t into X_t^f
4. Train a classifier on X_t^f
5. Use F_{pca} to transform X_v into X_v^f
6. Evaluate classifier on X_v^f

specific linear combination of base vectors to describe x_i. Instead of training and evaluating of the raw pixel values the activation vectors are used to train and test the classifier. Data set X_t is transformed into a set of activation vectors A_t which is used for training. To evaluate the trained classifier A_v is used which is the set of activation vectors for X_v. In our case a Random Forest is used as classifier. Every node split within the forest is based on a specific element inside an activation vector a_i instead of a raw pixel value from x_i.

3.2.3 Image Feature Boosting

As already mentioned in section 3.2.1 and section 3.2.2 Transfer Learning with PCA and Sparse Coding use a fixed set of features for one task. For Sparse Coding the number of features N is defined by the number the basis vectors contained in B. For Transfer Learning with PCA the number of features N is defined by the size of transformation matrix F_{pca}. Since we want to compare these methods to typical vision feature methods like pixel pairs and Haar Features we need a method to find a reasonable set of these vision features.

Viola and Jones [50] [24] already showed a robust way to select the most powerful features when a high quantity of random features is available. Therefore, an adapted version of the AdaBoost [45] algorithm is used. We take a set of random features and use the AdaBoost algorithm on a given data set X_t as described in section 2.3.4. As weak classifiers pixel pair features or Haar Features are used as described in section 2.1. For the

Algorithm 13 Transfer Learning with Sparse Coding

- **Require:** A labeled training set X_t from domain D_T with M sample images
- **Require:** A unlabeled auxiliary set X_s from domain D_S with N sample images
- **Require:** A unlabeled validation set X_v from domain D_T

1. Combine X_t and X_s to form X_{ts}
2. Calculate basis vectors B for X_{ts} as described in section 2.4.3.3
3. Calculate activation vectors A_t for data set X_t with respect to B as described in section 2.4.3.3
4. Train a classifier on A_t istead of X_t
5. Calculate activation vectors A_v for data set X_v with respect to B as described in section 2.4.3.3
6. Evaluate classifier on A_v

boosting process we set the number of boosting iterations I to $I = N$ to get an ensemble of N features. These N features are then used to convert image data before shown to our Random Forest.

Image Channels: When pixel pair features or Haar Features are used we also calculate 32 image channels as proposed in [15]. Each feature is restricted to use one channel to calculate the feature value. Thus, the 32 channels need to be calculated for training images as well as for test images. Table 3.1 shows how the channels are calculated.

Channel(s)	Description
1-3	Channels of the lab color space for the input image
4	Absolute values of first-order derivative $\frac{\partial}{\partial x}$
5	Absolute values of first-order derivative $\frac{\partial}{\partial y}$
6	Absolute values of second-order derivative $\frac{\partial^2}{\partial x^2}$
7	Absolute values of second-order derivative $\frac{\partial^2}{\partial y^2}$
8-16	Nine HOG-like channels [11]
17-32	Channels 1-16 after min and max filtration with filtersize 5×5

Table 3.1: 32 Image Channels from [15]

3.3 Object detection

For our object detection method we use a Hough Forest detector. Algorithm 14 illustrates how the Transfer Learning (TL) methods discussed in sections 3.1 and 3.2 can be combined with Hough Forests. Image data set I_t from target domain D_T and an image data set I_s from source domain D_S are used. For training random patches are extracted from I_t and I_s to form patch sets X_t and X_s. The combined patch set $X_{ts} = X_t \cup X_s$ is used to train a Transfer Learner T which internally builds the final trained Hough Forest H. As usual for Hough Forests every positive image patch x_i inside X_{ts} has a corresponding center offset. This center offset denotes the distance and direction from the patch center to the center of the object it is part of. The center offsets will be stored inside the Hough Forests leaf nodes as two dimensional offset vectors.

TL method	Combination with Hough Forests
TL with Outlier Detection	is used to find outlying examples in X_s with respect to X_t. $X_{to} = X_t \cup X_o$ is used to train a Hough Forest where X_o is X_s without outlying examples. This was explained in section 3.1.1.
Transfer Boosting	is used to iteratively add single trees to the Hough Forest. The trees are trained on X_{ts} where all examples in X_{ts} are weighted so that the classification on X_t is as good as possible. This was explained in section 3.1.2.
TL with PCA	is used to transform X_t to another representation X_t^f. For this transformation $X_{ts} = X_t \cup X_s$ is taken into account as explained in section 3.2.1. The final Hough Forest is trained on X_t^f instead of X_t. During testing all example patches x_v are also to be transformed into x_v^f before being evaluated.
Sparse Coding	is used to transform X_t into a set of activation vectors A_t. For this transformation $X_{ts} = X_t \cup X_s$ is taken into account as explained in section 3.2.2. The final Hough Forest is trained on A_t instead of X_t. During testing all example patches x_v are also to be transformed into activation vectors a_v before being evaluated.

Table 3.2: Combination of TL methods with Hough Forests

Table 3.2 shows how the different Transfer Learning methods are combined with Hough Forests. TL with Outlier Detection and Transfer Boosting try to use additional examples from X_s during training. TL with PCA and Sparsecoding are used to create image feature

representations which encode knowledge about $X_{ts} = X_t \cup X_s$. These learned features are used to transform the data before it is shown to the Hough Forest. Thus, there is no need to use additional image features like pixel pairs or Haar Features within tree nodes. In addition also the image patches in the testing phase need to be transformed to this representation so that the input format stays consistent.

Algorithm 14 Transfer Learning with Hough Forests

- **Require:** A labeled image data set I_t from domain D_T
- **Require:** A labeled image data set I_s from domain D_S
- **Require:** A unlabeled validation set I_v from domain D_T with V sample images
- **Require:** A Transfer Learner T which internally builds the Hough Forest H

1. Extract random image patches from I_t to form patch set X_t with M sample image patches
2. Extract random image patches from I_s to form patch set X_s with N sample image patches
3. Combine X_t and X_s to form X_{ts}
4. Use X_{ts} on T to obtain the trained Hough Forest H
5. For(k = 0 to V)
   - Get evaluation data X_v^k by extracting a patch for every pixel inside image I_v^k
   - Use X_v^k to obtain a hough image hypothesis h_v^k for image I_v^k
   - Postprocess h_v^k to get a set of detections d_v^k for image I_v^k
   - Compare d_v^k to the original labels of I_v^k to get a performance measure

3.4 Chapter Summary

In this chapter it is explained how different Transfer Learning (TL) methods can be combined with Random Forest based methods. Within this chapter X_t denotes a target training set from target domain D_T which is the domain of interest. X_s denotes an additional training set from source domain D_S which should provide additional information for training.

Section 3.1.1 describes how Transfer Learning with Outlier Detection can be used to remove examples from X_s which are outlying to the distribution of X_t. $X_{to} = X_t \cup X_o$ is

then the final training set where X_o is X_s without outlying examples.

Section 3.1.2 describes how Transfer Boosting can be used to add Decision Trees to a Random Forest where each of these trees is trained on a specific weighted version of $X_{ts} = X_t \cup X_s$. During this process the Transfer Boosting algorithm tries to transfer as much information from X_s to X_t.

Section 3.2.1 shows how TL with PCA can be used to create image feature representation which take into account $X_{ts} = X_t \cup X_s$ instead of X_t. These feature representations can then be used to train classifiers like Random Forests.

Section 3.2.2 describes how Spase Coding can be used to represent images as linear combination of base vectors. The calculation of base vectors takes into account $X_{ts} = X_t \cup X_s$. These linear combinations are called activation vectors and can then be used to train Random Forests or other classifiers.

Section 3.3 shows how the TL methods discussed in sections 3.1 and 3.2 can be combined with Hough Forest object detectors. For each method of TL a specific way to combine it with Hough Forests is discussed.

Chapter 4

Experiments

Contents

In this section we will present the results of experiments done and compare our methods to results of state-of-the-art methods. All experiments have been done using an Acer Aspire 7750G Notebook with an Intel Core 2 Duo T9400 at 2.53GHz, 4GB RAM and a 32Bit Windows Vista Home as operating system. For image and matrix operations openCV [38] was used. For Sparse Coding we used a matlab implementation which was provided by Lee et al. [27]. For our object detection tasks the comparison of found detection rectangles to the original bounding box labels and for plotting **Piotr's Image & Video Toolbox for Matlab** [40] was used.

4.1 20 Newsgroups Dataset part 1

Experiment Description: The first task is a classification task on the $20newsgroups_2$ * data set [39]. It is one version of the popular 20newsgroups † data set. This version was

*http://www.cs.columbia.edu/~wfan/software.htm
† http://people.csail.mit.edu/jrennie/20Newsgroups/

processed for the use of the *AcTraK* [46] algorithm. A downloadable version of the data as well as an implementation of the *AcTraK* algorithm can be found at [14] [‡].

The data set is designed for an Inductive Transfer Learning setting as defined in section 2.4.1. It consist of 4 different categories which are Comp, Rec, Sci and Talk. Each of these categories defines a single domain. They are similar but not identical. For each of these 4 domains a training set and a evaluation set is available. Each set contains about 2000 examples of 2 different classes. Class 0 denotes negative examples and class 1 positive ones. As for every inductive transfer setting, we have a target training set T, a target test set V and an source data set S, whereas T and V share the same domain. S is drawn from a different but related domain and should improve the performance on V. When testing one domain against another, T and V are drawn from the same domain, whereas S is drawn from a different one. As an example, Comp vs. Rec means that Comp is the source domain and Rec is the target domain. Thus, T and V are the training and evaluation sets from category Rec and S is the training set from category Comp.

This experiment was chosen to compare our solution of Tranfer Learning with Outlier Detection (**RF-O**) and Random Forests with Transfer Boosting (**RFTB**) to TrAdaBoost without Random Forests and *AcTraK* [46]. Our Random Forest methods with Transfer Learning are also compared to non-Transfer Learning methods. As baseline methods we used a simple Random Forest (RF) and the results of a Support Vector Machine (SVM) from [39]. Our Sparse Coding solution could not be used for this experiment since it requires image data to calculate Sparse Features. Transfer Learning with PCA was also excluded in this experiment since it works similar to our Sparse Coding solution and should be compared to that.

Experiment Setup: Each of our Random Forest based methods was configured to grow all trees to full extent to get the best possible performance per tree. This was done by setting the maximum number of trees to 100 which could never be reached. Our Random Forest with Transfer Boosting (**RFTB**) was configured to create a forest of 10 trees. For this specific data set we could not increase this number because after 10 trees the internal classification error within the Transfer Boosting process was 0 and caused the algorithm to stop. Thus, all other Random Forests have also been configured to create 10 trees per forest. For RFTB the source data set S was used in addition to T for training. The Random Forest with Outlier Detection (**RF-O**) used data set T for training and additional samples from S which were not declared outliers. As first baseline method we

[‡] http://www.cs.columbia.edu/~wfan/software.htm

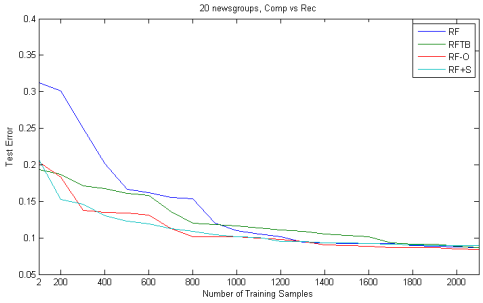
used a simple Random Forest (**RF**) which was trained solely on T. As second baseline we trained a Random Forest (**RF+S**) on the combined set of T and S. This should give information about the relationship between the domains of T and S. If they are strongly related the performance of RF+S should be better than the performance of RF. If the relation is weak, negative transfer should occur in this case and the performance of RF+S should be lower than the performance of RF.

Experiment Results: Figure 4.1 compares RFTB and RF-O against our 2 baseline methods RF and RF+S. The error graphs show the classification error on the test set V in relation to the number of used training samples from T. The number of samples used from S was always set to the maximum. Image (a) in Figure 4.1 shows that domains of Comp and Rec are strongly related since there is no negative transfer for RF+S. In this case all of our methods had about an equal performance when all samples from T were used. All other images show a negative transfer for RF+S which means that samples in S harmed the learned model. In these 4 cases out of 5 RFTB showed the best overall performance and RF+S the worst. All images in Figure 4.1 show that methods which use additional data from S are better than RF when only a few samples from T are used for training.

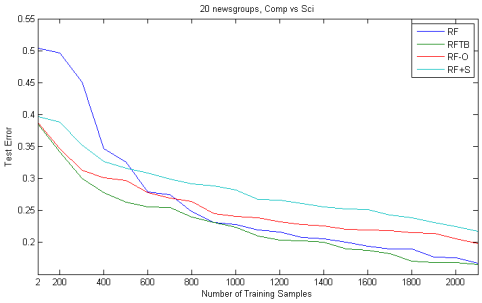
$20newsgroups_2$ [39].						
Source v.s. Target	Accuracy %					
	SVM	RF	TrAdaBoost	RF-O	RFTB	AcTraK
Comp v.s. Rec	49.1	68.8	77.2	79.5	80.6	**82.1**
Comp v.s. Sci	52.7	49.9	57.3	61.1	61.4	**78.0**
Rec v.s. Sci	59.1	50.1	67.4	62.7	65.1	**70.6**
Rec v.s. Talk	60.2	58.1	72.3	69.5	70.8	**75.4**
Sci v.s. Talk	57.6	66.8	71.3	73.7	74.5	**75.1**

Table 4.1: $20newsgroups_2$. Accuracy of different methods using only 2 labeled training samples.

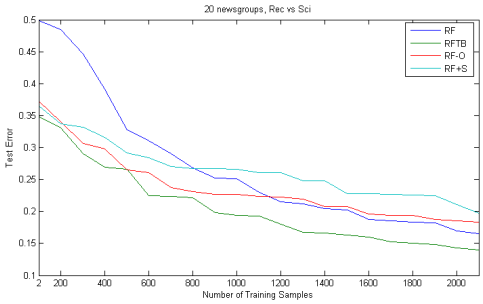
Table 4.1 compares the accuracy of different methods in percent of correct classifications when only using 2 labeled samples in set T as stated in [46]. In 3 of 5 cases RFTB performs better than the TrAdaBoost with other classifers than Random Forests. In 2 of 5 cases even RF perfoms better than the SVM baseline. *AcTraK* has the best performance in all 5 cases, but it uses a domain expert to label an unknown number of additional samples from T to increase performance.



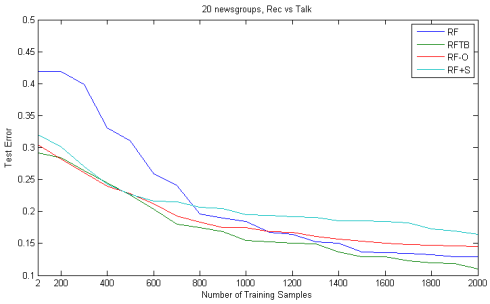
(a) Comp v.s. Rec



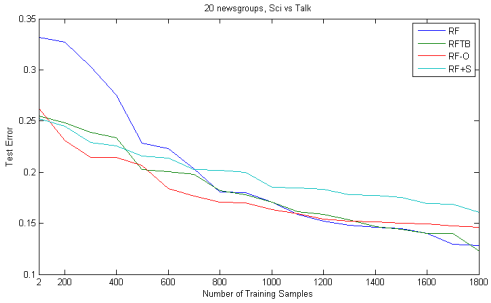
(b) Comp v.s. Sci



(c) Rec v.s. Sci



(d) Rec v.s. Talk



(e) Sci v.s. Talk

Figure 4.1: 20 Newsgroups Inductive Transfer: Error Graphs.

Experiment Conclusion: The experiment showed that the use of an additional data set S is especially usefull when only a few training samples in T are available and Transfer Learning should be used in ths case. When domains of T and S are weakly related negative transfer can occur if T and S are used for training without a Transfer Learning method. RF-O reduces negative transfer because the most outlying examples are removed from S but can not find all examples which harm the learned model. RFTB can avoid negative transfer very well. Even when many examples are used from T, RFTB can still increase the performance by extracting parts of S which do not harm but improve the learned model. Thus, RFTB can also be used when many trainig samples are available for T and the relationship between domains of T and S are weak or unknown. The additional computational effort is moderate and additional cost for labeling does not occur.

4.2 20 Newsgroups Dataset part 2

Experiment Description: Another text classification task based on the 20newsgroups data set has been discussed by [16] [39]. Here, the data is preprocessed in a different way and will be denoted as $20newsgroups_3$ [39]. This data set has been used for the *LWE* [16] algorithm. The data and an implementation of the *LWE* weighting scheme can also be found online [14].

The data set is designed for an Inductive Transfer Learning setting as defined in section 2.4.1. The task structure is the same as for the $20newsgroups_2$ data set. There are 4 domains which are Comp, Rec, Sci and Talk. For each domain a training set and a validation set is available which both include about 2000 examples. T and V are drawn from the same domain and S is drawn from another one. Domains are tested against each other. For instance Comp vs. Rec means that T and V are drawn from domain Rec and S is drawn from domain Comp.

This experiment has been done to compare our solutions for Tranfer Learning with Outlier Detection (**RF-O**) and Random Forests with Transfer Boosting (**RFTB**) to other Transfer Learning methods such as **LWE** [16], **pLWE** [16], **TSVM** [23]. Our Transfer Learning methods are also compared to some non-Transfer Learning baselines such as simple Random Forests (**RF**) and Support Vector Machines (**SVM**). As for the experiment discussed in section 4.1 our solutions for Sparse Coding and Transfer Learning with PCA are excluded from this experiment because the experiment does not involve image data.

Experiment Setup: As for the experiment shown in section 4.1 only 10 trees could be trained for RFTB since an internal classification error of 0 caused the Transfer Boosting process to stop. Thus, all other Random Forest based methods have also been restricted to use 10 only trees. All trees have been allowed to grow to their full extent by setting the maximum tree depth to 100. The maximum depth which was actually reached was about 20. RFTB used target training set T and source training set S, RF-O used T and in addtion those examples from S which have not been declared outliers. As baseline we trained a simple Random Forest (RF) only on T.

$20newsgroups_3$ [39].							
Source v.s. Target	Accuracy %						
	SVM	RF	TSVM	RF-O	RFTB	pLWE	LWE
Comp v.s. Rec	81.5	91.8	89.6	92.2	92.6	91.9	**98.1**
Comp v.s. Sci	71.1	83.2	76.9	80.1	83.4	78.7	**97.4**
Rec v.s. Sci	78.1	83.5	89.9	81.7	86.0	88.4	**98.2**
Rec v.s. Talk	68.2	87.1	89.9	85.5	88.9	72.1	**99.2**
Sci v.s. Talk	75.7	87.1	85.5	85.4	87.6	83.3	**96.9**

Table 4.2: $20newsgroups_3$. Accuracy of different methods.

Experiment Results: Table 4.2 shows the accuracy of different transfer learning algorithms on this data set when all 2000 samples from T and all 2000 samples from S are used. The LWE algorithm has the best performance in all 5 cases. In 4 of 5 cases RFTB performs better than pLWE, and in 3 of 5 cases it performs better than TSVM. RF-O shows negative transfer which results in its performance being slightly lower than the performance of RF in 4 out of 5 cases.

Experiment Conclusion: This experiment shows that RFTB is at a par with TSVM and pLWE. Only the LWE algorithm could outperform the RFTB method. RF-O showed a loss of performance due to negative transfer. This occured because RF-O could not remove all harmfull examples from S. Thus, the outlier measure is not a good way to determine whether an example helps or harms the learned model. RFTB did not show negative transfer since its performance is always better than the performance of RF.

4.3 Character Image Classification

Experiment Description: For the Self-taught Learning [44] setting the experiment of handwritten character classification from [44] is used to compare Transfer Learning with Random Forests to other approaches. The task consists of a target training set T which is drawn from the domain of handwritten characters, a target test set V which is also drawn from this domain, and a source training set S which is drawn from the domain of handwritten digits. T contains 1000 images of handwritten characters from a to z but without x. This results in 25 classes where the class labels are about equally distributed. V contains about 37000 images of handwritten characters from a to z, without x. S consists of 2500 images of handwritten digits from 0 to 9. The 10 classes within S are equally distributed. Every image in T, S and V is available in a 28×28 grayscale format. This scale is not changed during the whole experiment.

This experiment has been performed to compare Random Forests with Sparse Coding (**RF+SC**) and Random Forest with PCA (**RF+PCA**) to other Self-taught Learning algorithms which are not using Random Forests. Since RF+SC and RF+PCA provide learned feature representations for digital images, they are also compared to Random Forests with different kinds of traditional vision features such as Haar Features and pixel pairs. Random Forests with Transfer Boosting and Transfer Learning with Outlier Detection can not be evaluated on this data set because the label space of T and S is not related to each other. Since Self-taught Learning does not take the labels of S into account, it does not matter that labels for S do not correspond to the labeling scheme for T. In this case, S is only used to find better features to classify V and all classifiers are trained solely on T.

Experiment Setup: All of our Random Forest based methods have been allowed to grow their trees to full extent. Each forest was configured to train 100 trees. This was done because more trees could not further increase performance. For Random Forests with Sparse Coding (**RF+SC**) 100 Basis Vectors have been calculated from all images contained in T and S to transfer additional information from S to T. Therefore, 1000 iterations of Sparse Coding [44] have been done in matlab. These Basis Vectors are shown in Figure 4.2.

Random Forest with PCA (**RF+PCA**) calculated 100 PCA-Eigenvectors as feature representations from T and S as defined in section 3.2.1. Random Forest with Haar Features (**RF+Haar**) used a fixed set of 100 random Haar Features. Random Forest with pixel pair features (**RF+PixPair**) used a fixed set of 100 random pixel pair features. Thus, all of these methods used a fixed set of 100 features. As a baseline we used a Ran-

dom Forest with raw pixel value features (**RF+Raw**). RF+Haar and RF+PixPair calculated their feature values from 32 image channels as described in section 3.2.3. RF+SC, RF+PCA and RF+Raw used only 1 gray scale channel. All methods were trained iteratively on 25 to 1000 examples from T. S was only used for RF+SC and RF+PCA to transfer information during the feature learning phase.

Experiment Results: Figure 4.3 shows the error rates of our methods in relation to the number of used training samples from T. As expected RF+Raw yields the lowest performance values. RF+SC performed slightly better than RF+Raw and RF+PCA. RF+PixPair and RF+Haar outperformed the other 3 methods although the features used were created at random. RF+Haar performed slightly better than RF+PixPair.

Table 4.3 compares the results of our methods to result of equivalent algorithms using either support vector machines (SVM) or Gaussian discriminant analysis (GDA). SVM/GDA denotes the result of the algorithm that performed better. In all cases, RF+Haar had the best performance followed by RF+PixPair. RF+PCA and RF+SC performed better than the best of the SVM/GDA baselines in most cases but even RF+Raw could achieve almost equal performance.

Experiment Conclusion: Random Forest classifiers yielded a better performance in combination with PCA and Sparse Coding than SVM or GDA classifiers. Thus, they

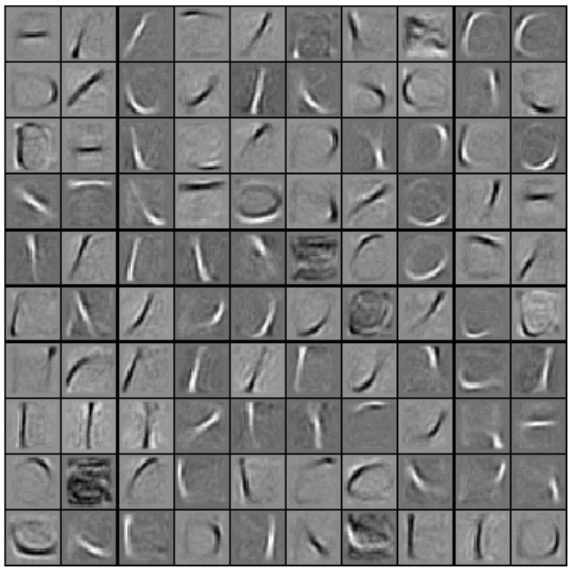


Figure 4.2: Handwritten Character Classification: Basis Vectors

Handwritten Character Classification			
Classifier	Accuracy %		
	Sample Count		
	100	500	1000
SVM/GDA	39.8	54.8	61.9
PCA+SVM/GDA	25.3	54.8	64.5
SC+SVM/GDA	39.7	58.5	65.3
RF+Raw	38	60.6	69.3
RF+PixPair	60.4	81.3	85.1
RF+Haar	**63.8**	**82.7**	**85.5**
RF+PCA	32.8	59.8	66.3
RF+SC	31.8	62	70.8

Table 4.3: Accuracy of different methods on the classification of Handwritten Characters. Accuracy given in percent of correct classifications

should be favoured over SVM and GDA. RF+Haar and RF+PixPair outperform RF+PCA and RF+SC even when random Haar Features and pixel pairs are used. Thus, traditional vision features are a better choice than our PCA and Sparse Coding methods.

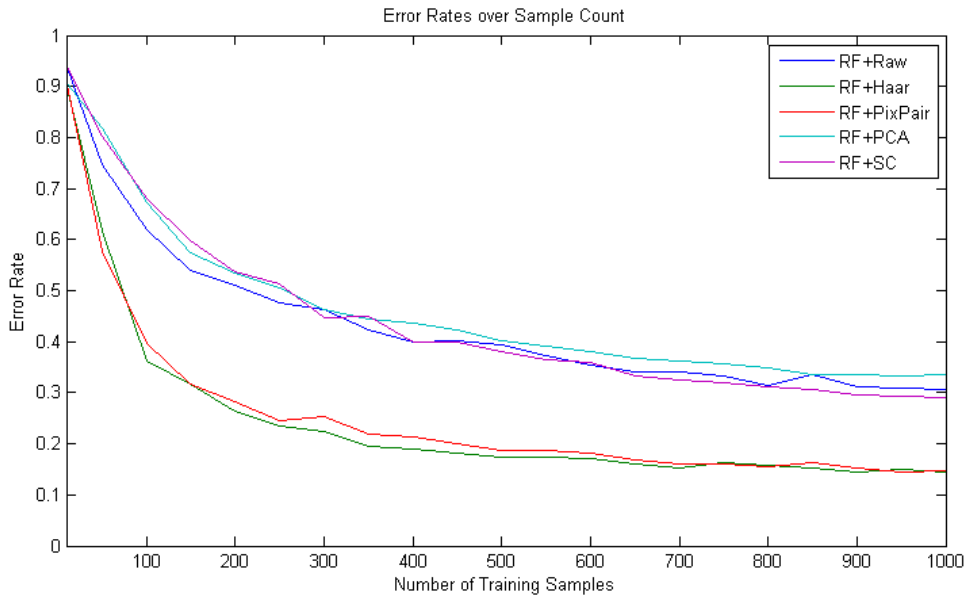


Figure 4.3: Handwritten Character Classification

4.4 Pedestrian Detection

Experiment Description: In this section we will present the detection performance of our Hough Forest with transfer learning on a pedestrian detection task. Since we wanted to compare Hough Forests with different transfer learning methods to the original Hough Forest [15] we tried to reproduce the experiment settings of [15] as good as possible. The TUD pedestrian data set which was presented in [3] is used to train the Hough Forest and parts of this data set are used for testing. The following list shows which parts of the TUD data set have been used.

- train-400: This subset contains 400 images. Each image shows one walking person. The set contains different persons and different background locations. This set is used as training set.
- tud-campus-sequence: This subset contains 71 images. Each image shows multiple persons walking in front of the TUD campus. This set is used as test set.
- tud-crossing-sequence: This subset contains 201 images. Each image shows multiple persons crossing a street. This set is used as test set.
- tud-pedestrians: This subset contains of 250 images. Each image shows one ore more pedestrians. There are no overlapping objects. Persons and background locations vary. This set is used as test set.

The second data set we used is the INRIA person data set which has been presented in [11]. The set consists of 1126 positive examples and 453 negative background images. Each positive example shows one person from a front view while the samples from the TUD data set show pedestrians from a side view. As in [15] we used the negative samples from the INRIA set as negative background samples for all classifiers during training. For transfer learning the positive examples from the INRIA set formed our auxiliary data set.

This experiment was chosen to compare our Hough Forest with Transfer Learning (HFTL) to the original Hough Forest as presented in [15]. We also compare HFTL to our Hough Forest implementation without Transfer Learning to show how Transfer Learning can improve the learned model.

Experiment Setup: Before training we needed to preprocess the positive training examples to fit a certain object scale. As in [15] we scaled all images in the train-400 data

TUD: Average number of Tree Nodes in 1000			
Classifier	Nodes	Classifier	Nodes
Combined	73.2436	Outliers	60.4332
Gall	69.4239	TrBoost	100.0755
PixPair	37.151	PCA	38.0556
Haar	30.8102	SparseCoding	58.0488

Table 4.4: TUD: Average number of Tree Nodes

set so that each contained object had a height of 100 pixels. For the positive examples from the INRIA set we chose to use the ones which where of 64×128 pixel size. We removed 12 pixels from the left- and right side and removed 14 pixels from the top- and bottom border to reduce the amount of false positives in the training set. The resulting images were then scaled to 40×100 pixel images. The negative examples from the INRIA set were used as they are.

For the training process patches of size 16×16 were used. 100 patches were extracted from every positive example object and 100 patches from each negative example image. Since some of our transfer learning methods took very long to process the data we decided to skip bootstrapping as described in [15]. We tried to compensate the missing bootstrap patches by extracting not only foreground patches from every object in every positive training image but also extracting 100 background patches from every positive training image. This setup results in 40000 positive training patches and 85300 negative training patches in target training set T and 112600 positive auxiliary patches in source training set S. For each forest 15 trees were trained as done in [15]. The maximum depth of each tree was set to 100, simply to ensure that each tree is grown to its full extent. Table 4.4 shows the final number of tree nodes within each classifier. The values are averaged over all trees in the forest.

Each classifier used a fixed set of 100 features to ensure equal conditions for all our methods. The 100 pixel pair features and 100 Haar Features were chosen out of 10^6 total features. The 100 final features were found by a boosting process as described in section 3.2.3. Both, pixel pair features and Haar Features were calculated on top of 32 image patch channels as described in section 3.2.3. The **PixPair** Hough Forest used the pixel pair features and was trained on T without the auxiliary set S. The **Haar** Hough Forest used the Haar Features and was also trained on T without the auxiliary set S. The **Combined** Hough Forest used the pixel pair features and was trained on the combined set of training data T and auxiliary data S without any modifications on these data sets. The

TUD: Processing Time in seconds per image					
Classifier	Training	Detect	Classifier	Training	Detect
Combined	1.6793	88.2	Outliers	8.0971	85.1857
Gall	105.6338	25.3954	TrBoost	6.2749	86.5857
PixPair	1.5833	27.7	PCA	1.4578	295.5143
Haar	1.6197	104.9714	SparseCoding	11.2504	312.7
Feature Preparation Time in seconds per image					
Boost PixPairs	112.3474	Boost Haar	109.8357	Sparse Features	173.9376
Sample Image Count					
TUD training	853	INRIA training	1126	Sum training	1979

Table 4.5: TUD: Processing Time

Outlier Hough Forest also used pixel pair features and was trained on the combined set of training data T and auxiliary data S. In this case the outlier measures for all examples in the auxiliary set S have been calculated. These outliers have then been removed from S before final training. 50% of the patches from the INRIA person images were considered outliers. The **TrBoost** Hough Forest also used pixel pair features and was trained on the combined data set of T and S. The iterations of transfer boosting were configured to produce an ensemble of 15 trees. The **PCA** Hough Forest used 100 PCA features calculated on top of only 1 grayscale image patch channel. The **SparseCoding** Hough Forest used 100 Basis Vectors which have also been calculated on top of one grayscale image patch channel. PCA Features and Basis Vectors were calculated from the combined data set of T and S and used raw pixel intensities as input values. Table 4.5 shows the processing time which was needed to find features and to train the final classifiers.

The resulting Hough Forests were tested on the 3 test data sets tud-campus-sequence, tud-crossing-sequence and tud-pedestrians. 4 scales were used to form the image pyramid of hough images. The scales were [0.30.40.50.6]. For the **SparseCoding** Hough Forest every fourth pixel was used to add hough votes to the hypothesis image. This was done to reduce the processing time. The original Hough Forest [15] (**Gall**) and all of our other Hough Forests used every pixel to generate their Hypothesis. The postprocessing method which was described in section 2.3.3.2 was then used to find the final detection rectangles. To find the optimal rectangle size a simple brute force search was used. For this procedure the rectangle width was in the range of $[20 - 50]$, and the rectangle height in the range of $[80 - 100]$.

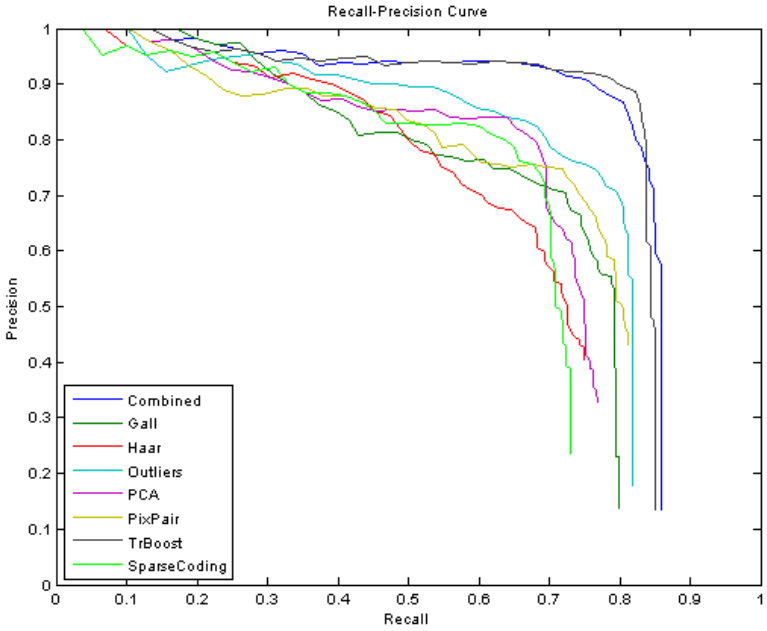


Figure 4.4: TUD-Campus: Recall-Precision Curve

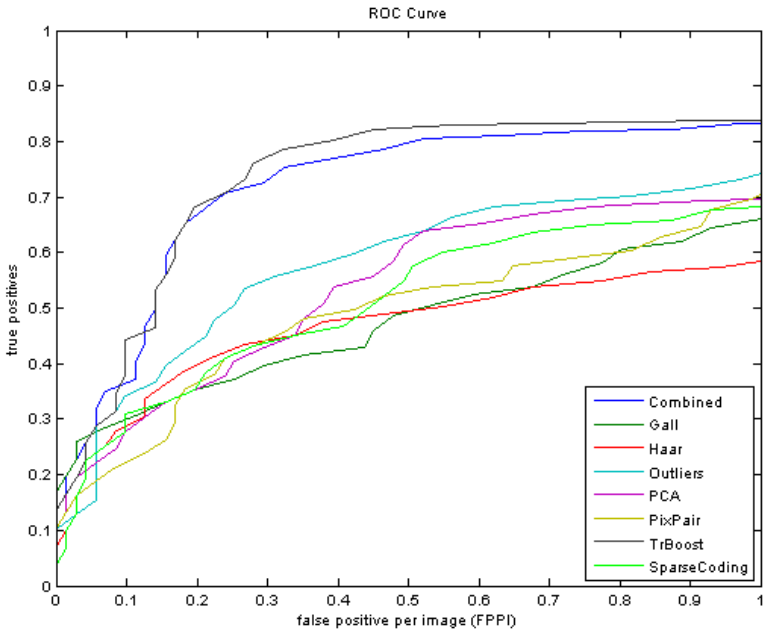


Figure 4.5: TUD-Campus: ROC Curve

TUD-Campus: Equal-Error Rates			
Classifier	EER	Classifier	EER
Combined	0.839378	Outliers	0.759076
Gall	0.714754	TrBoost	**0.853701**
PixPair	0.736134	PCA	0.738351
Haar	0.662162	SparseCoding	0.715789

Table 4.6: TUD-Campus: Equal-Error Rates

Experiment Results: Figure 4.4 shows the recall precision curve for the tud-campus-sequence set, Figure 4.5 shows the corresponding ROC curve and Table 4.6 shows the equal-error-rates (EER) for this test set. This data set contains many overlapping objects, and thus the recall is lower than for the other TUD test sets. The **TrBoost** Hough Forest had the best performence but the **Combined** Hough Forest reached almost the same level. The **Outlier** Hough Forest did not perform that well, thus the removing of specific outliers lowered the performance. The **SparseCoding** Hough Forest performed almost as good as the non-transfer learning algorithms and the **PCA** Hough Forest even had a slightly better EER than the non-transfer learning forests.

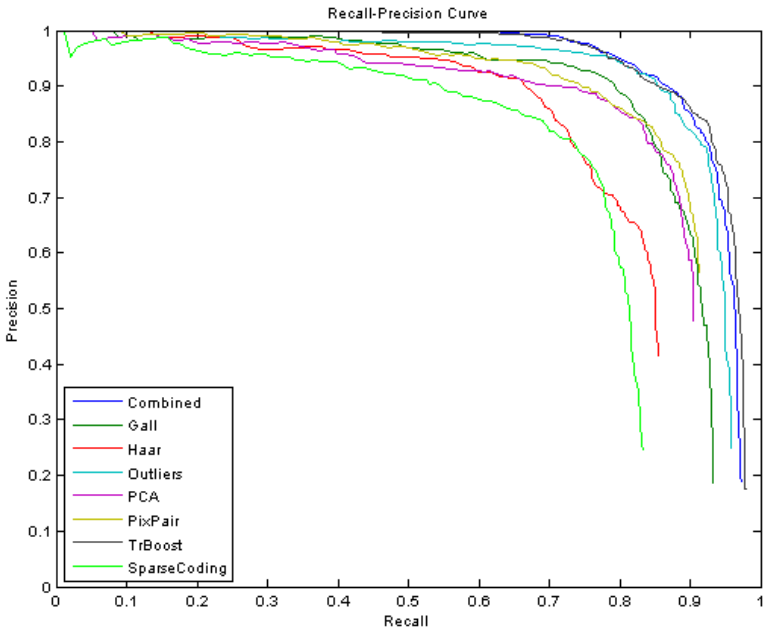


Figure 4.6: TUD-Crossing: Recall-Precision Curve

Figure 4.6 shows the recall precision curve for the tud-crossing-sequence set, Figure 4.7

TUD-Crossing: Equal-Error Rates			
Classifier	EER	Classifier	EER
Combined	**0.885899**	Outliers	0.880781
Gall	0.845158	TrBoost	0.884634
PixPair	0.837506	PCA	0.833000
Haar	0.772475	SparseCoding	0.766112

Table 4.7: TUD-Crossing: Equal-Error Rates

shows the corresponding ROC curve and Table 4.7 shows the equal-error-rates (EER) for this test set. This set does not contain overlaps and the objects in the scene do not vary much in their size. Thus, this task is a less difficult task for the detector than the other 2 pedestrian sets. On this set the **TrBoost** Hough Forest achieved the best performance which was just slightly better than the performence of the **Combined** Hough Forest. Again the removing of outliers reduced the quality of the learned model. The **PCA** Hough Forest was almost as good as the non-transfer methods. Only the **SparseCoding** Hough Forest performed significantly worse than all other methods.

Figure 4.8 shows the results for the tud-crossing-sequence (left) and the tud-campus-sequence (right). The image was taken from [4]. The dark blue curve represents the

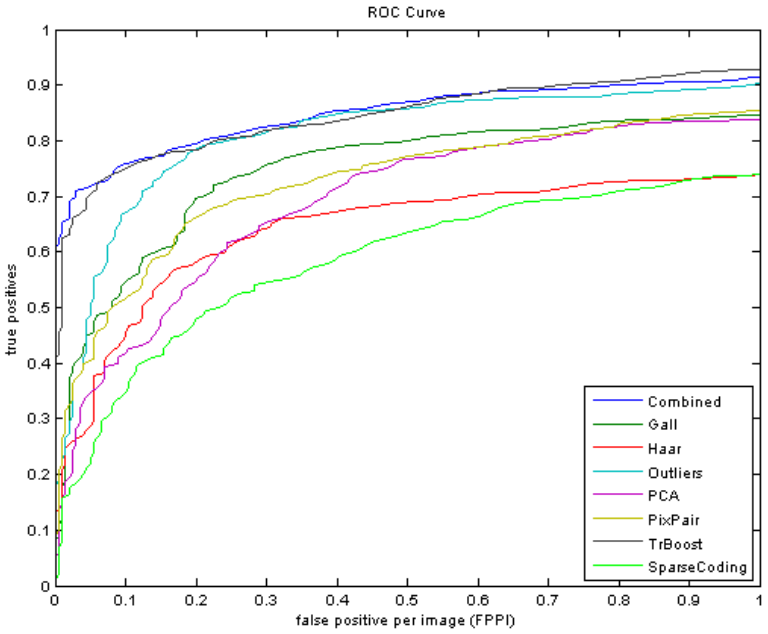


Figure 4.7: TUD-Crossing: ROC Curve

method of Gall et al. [15] while the light blue curve represents the method of Barinova et al. [4]. On the tud-crossing-sequence the method of Barinova et al. [4] yields the best performance because they developed a postprocessing which is specialized in detection of overlapping objects. On the tud-campus-sequence our inductive transfer learning methods performed better.

Figure 4.9 shows the recall precision curve for the tud-pedestrians set, Figure 4.10 shows the corresponding ROC curve and Table 4.8 shows the equal-error-rates (EER) for

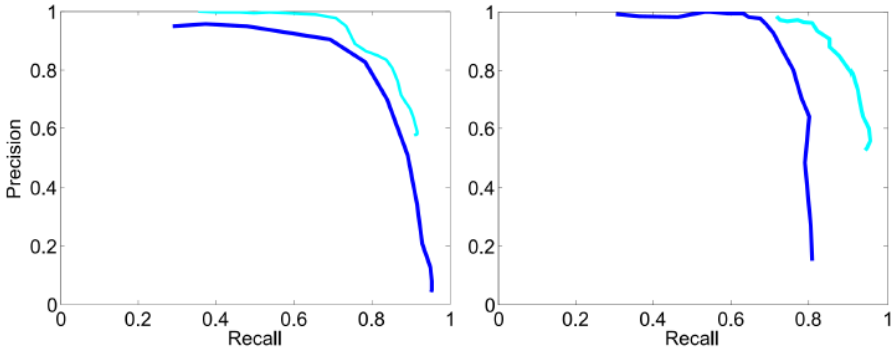


Figure 4.8: TUD-Crossing and TUD-Campus: Barinova and Gall Recal-Precision Curve from [4]

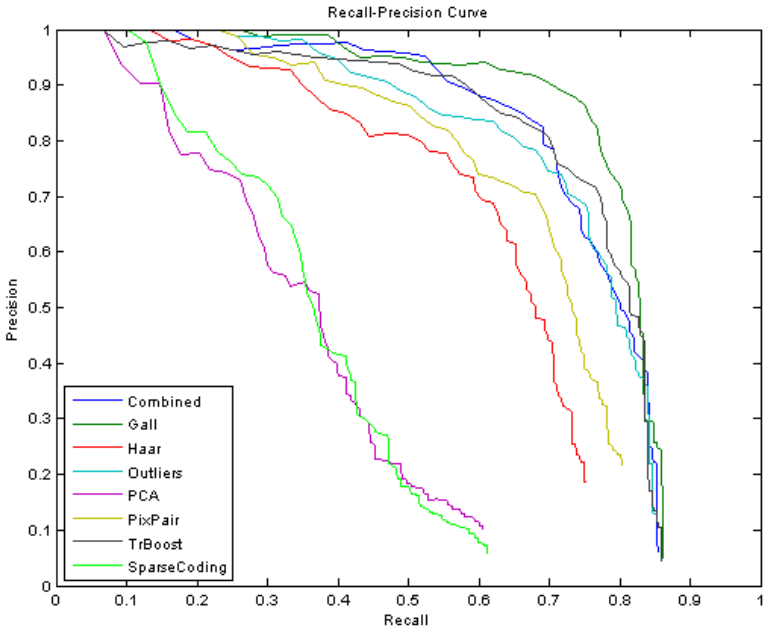


Figure 4.9: TUD-Pedestrians: Recall-Precision Curve

TUD-Pedestrians: Equal-Error Rates			
Classifier	EER	Classifier	EER
Combined	0.753065	Outliers	0.730051
Gall	**0.805508**	TrBoost	0.752613
PixPair	0.696223	PCA	0.436090
Haar	0.654867	SparseCoding	0.441718
Gall+Bootstrap [15]	**0.865**		

Table 4.8: TUD-Pedestrians: Equal-Error Rates

this test set. The tud-pedestrians data set contains no overlaps but the pedestrians have high variations in size and the scenes show different locations. Some locations include very difficult backgrounds or other objects which could yield false positive detections. On this set the original Hough Forest by Gall et al. [15] performed significantly better than all of our methods. For our methods the **Combined** Hough Forest had the best performance which was slightly better than the performance of the **TrBoost** Hough Forest. Removing the outliers in the auxiliary set resulted again in a loss of performance and the **PCA-** and **SparseCoding** Hough Forests had very low performance.

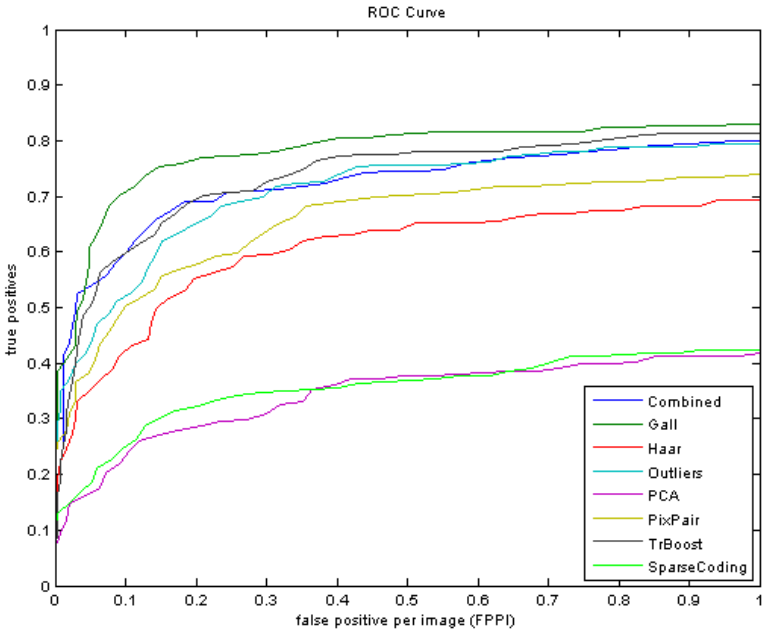


Figure 4.10: TUD-Pedestrians: ROC Curve

Experiment Conclusion: The use of additional auxiliary training data improved the performance of Hough Forest based methods. The results of the **Combined** Hough Forest showed that the domains of target data and source data have a very strong relationship to each other since no negative transfer occurred. The **TrBoost** Hough Forest could further increase the performance in 2 out of 3 cases and thus showed to be the most robust of our Transfer Learning methods. The **Outlier** Hough Forest removed examples from the auxiliary set which could have improved the learned model. This behaviour makes them a bad choice for strongly related data sets.

We tried to achieve the performance of our **TrBoost** Hough Forest with our non-Transfer Learning methods. Therfore, we increased the number of image patches which were extracted from positive training images. When we increased this number from 100 to 500 the performance of the **PixPair** Hough Forest was about equal to the performance of our **TrBoost** Hough Forest. Thus, we needed the 5-fold number of samples in T to achieve the performance of the **TrBoost** Hough Forest. This means that the 112600 patches in S are worth about as much as 160000 additional patches in T. For this comparison it is also important that increasing the number of training patches in T is only possible to some extent. Since patches are sampled randomly from training images, patches will contain more redundant information if more patches are sampled per image. Thus, there is a limit were performance can not be further increased. When this limit is reached data S can be used to add additional information. Taking this into account the best solution would be to maximize the patch sample count to this limit and then use additional data S to further improve the learned model.

For our Self-Taught Learning methods **PCA** Hough Forest and **SparseCoding** Hough Forest we found that there is no benefit when they are used to learn feature representations. While the **PCA** Hough Forest achieved about the same performance as our Hough Forests with vision features, the **SparseCoding** Hough Forest showed lower performance. Both methods should not be used when it is possible to use vision features such as pixel pairs and Haar Features.

Faces in the Wild: Average number of Tree Nodes in 1000			
Classifier	Nodes	Classifier	Nodes
Combined	54.34148	Outlier	24.39938
Gall	70.11037	TrBoost	78.956683
PixPair	22.067	PCA	44.3858
Haar	20.20478	SparseCoding	46.542511

Table 4.9: Faces in the Wild: Average number of Tree Nodes

4.5 Face Detection

Experiment Description: The second detection task was the detection of faces in various scenes on the FDDB data set [20] which was presented in [19]. This data set is using the images from the "Faces in the Wild" data set [5]. The FDDB is divided into 10 folds, each of them providing a training set T containing about 300 images. This leads to a total of about 3000 images. The images show faces of different persons at different angles and size. Every image contains one or more faces. We resized the images so that the average height of face rectangles was at 50 pixels, preserving the aspect ratio.

As auxliliary data S we used the GENKI-4K [36] data set which is part of the "MPLab GENKI database" [35]. This data set contains 4000 images of persons in frontal view. Each image contains exactly one face rectangle. All images in this set were also rescaled so that the height of each face rectangle was at 50 pixels.

This experiment was chosen to compare our Inductive Transfer Learning methods **TrBoost** Hough Forests and **Outlier** Hough Forests to non-Transfer Learning methods. It should also provide information about the quality of features learned by **PCA** Hough Forests and **SparseCoding** Hough Forests.

Experiment Setup: Our experiment setup followed the guidelines described in [19]. Thus, we trained a detector for each of the 10 folds. Every detector had to be evaluated on one fold and trained on the nine remaining folds. For every Hough Forest detector 15 trees were trained as in the experiment described in section 4.4. Each tree was allowed to grow to its full extent by setting the maximum tree depth to 100. The size for image patches was set to 16×16 pixels. For training 20 image patches were extracted from every face rectangle and another 20 patches were taken from the background of the same image. This was done for the FDDB set as well as for the GENKI-4K set. This leads to 54000 background patches and about 77000 positive patches per fold for the FDDB data and 80000 positive and 80000 negative patches for the auxiliary set.

FDDB: Processing Time in seconds per image					
Classifier	Training	Detect	Classifier	Training	Detect
Combined	0.1271	14.4948	Outliers	0.3927	14.2803
Gall	9.3954	13.2784	TrBoost	0.5462	14.1869
PixPair	0.0426	14.7958	PCA	0.0486	13.8858
Haar	0.0412	21.0173	SparseCoding	0.5560	144.6540
Feature Preparation Time in seconds per image and fold					
Boost PixPairs	11.3887	Boost Haar	7.6333	Sparse Features	18.23856
Sample Image Count per fold					
FDDB training	2554	GENKI training	4000	Sum training	6554

Table 4.10: FDDB: Processing Time

For the **PixPair** Hough Forest 100 pixel pair features were used which were found by a boosting process from 40000 random pixel pairs. For the **Haar** Hough Forest 100 haar features were used. These features were also found by a boosting process from 40000 random haar features. Both pixel pair features and haar features were calculated on top of 32 image patch channels as described in section 3.2.3 and both were trained solely on T. The **Combined** Hough Forest also used the 100 pixel pair features and used the auxiliary data S from the GENKI-4K set in addition to the remaining FDDB folds. The **Outlier** Hough Forest used the 100 pixel pairs, data set T and parts of S. For auxiliary data S only patches from the GENKI-4K images were used which were not considered outliers in this case. About 60% of the GENKI-4K patches were considered outliers in this case. The **TrBoost** Hough Forest also used the 100 pixel pair features, data set T and the auxiliary data set S. The sample patches and the trained trees themselves were weighted in the transfer boosting process. The **PCA** Hough Forest used 100 PCA features which were computed from raw pixel values of one grayscale channel. The **SparseCoding** Hough Forest used 100 Base Vectors which were also calculated from raw pixel values of one grayscale channel. Both the **PCA** Hough Forest and the **SparseCoding** Hough Forest calculated their feature representations from the combined set of T and S. Table 4.9 shows the average number of tree nodes for each detector. The values are averaged over the number of trees contained in each forest. Table 4.10 shows the processing time for each detector.

For testing each fold was evaluated on the Hough Forests which were trained on the 9 remaining folds. The **SparseCoding** Hough Forest extracted an evaluation patch for every fourth pixel inside each image. The **Gall** Hough Forest and all of our Hough Forests extracted one patch for every pixel inside the test image. For the hough image pyramid

FDDB: Equal-Error Rates			
Classifier	EER	Classifier	EER
Combined	0.740430	Outlier	0.737380
Gall	0.735457	TrBoost	**0.744109**
PixPair	0.737324	PCA	0.602386
Haar	0.743814	SparseCoding	0.541836

Table 4.11: FDDB: Equal-Error Rates

we used the following six scale values $[0.1, 0.2, 0.4, 0.6, 0.8, 1]$. The resulting hough images were then postprocessed all at once per method. The postprocessing method which was described in section 2.3.3.2 was used to find the detection rectangles. Using this postprocessing method we performad a brute force search to find the optimal rectangle size. The range for the rectangle width was set to $[30 - 60]$ while the range for rectangle height was set to $[50 - 90]$.

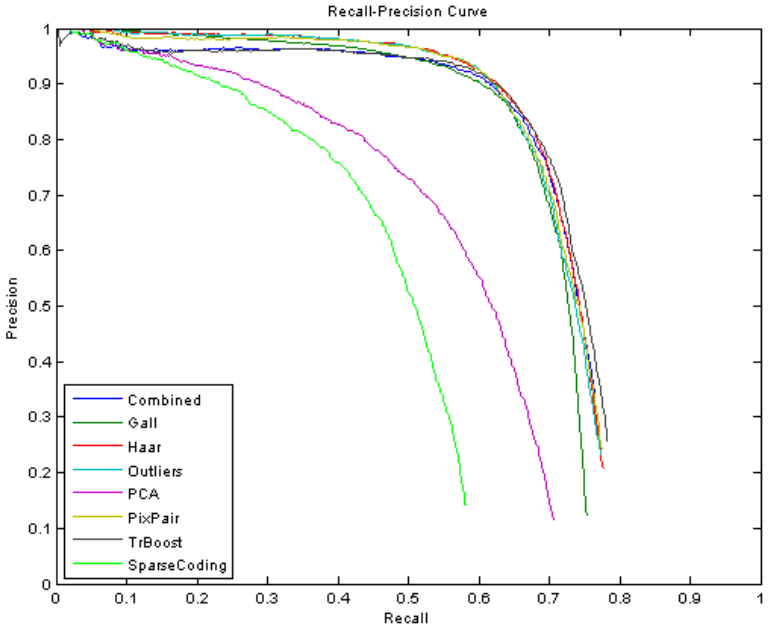


Figure 4.11: FDDB: Recall-Precision Curve

Experiment Results: Figure 4.11 shows the recall-precision curve and Figure 4.12 shows the corresponding ROC curve for this task. The equal-error rates (EER) for all detectors are shown in Table 4.11. The **PCA** Hough Forest and the **SparseCoding**

Hough Forest had a lower performance in comparison to all other methods. The methods which used the additional auxiliary set performed slightly better than the ones which were trained on the training set only. The **TrBoost** Hough Forest had the best equal-error rate. The **Outlier** Hough Forest did not perform as well as the simple **Combined** Hough Forest. This means that the outliers could have helped even more to refine the learned model.

Figure 4.13 shows the official ROC curve from [20]. When we compare the results from this image to our results the true positive rate of our methods is slightly higher. The best method shown in Figure 4.12 had a lower false-positive rate than our methods. The false-positive rate is shown in total false-positives in figure 4.12 while the false-positive rate is shown in false-positives per image for 3000 images in Figure 4.12.

Experiment Conclusion: This experiment showed that the use of additional source data S can improve the performance on this task. Again the results of the **Combined** Hough Forest showed that the domains of T and S are strongly related. Thus, only minimal negative transfer occurred for this method. The **TrBoost** Hough Forest could further improve the performance on this task and showed the best overall performance. The **Out-**

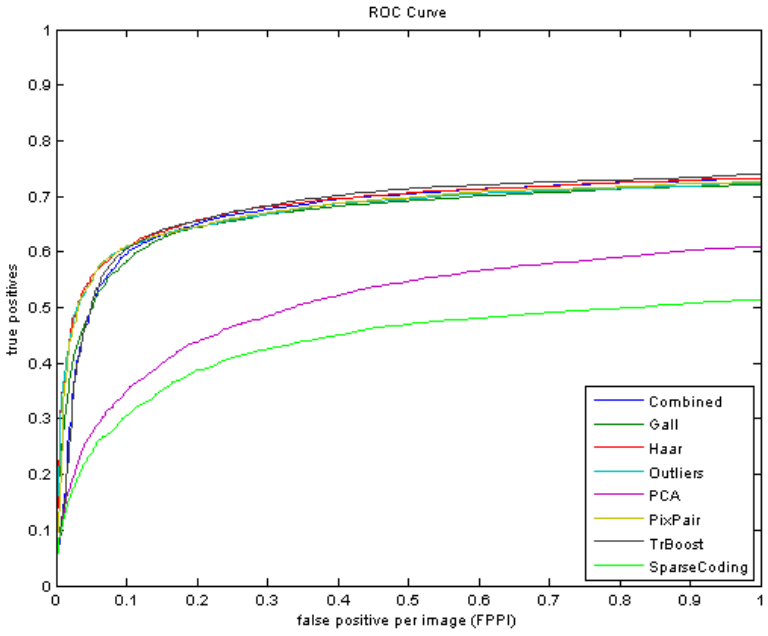


Figure 4.12: FDDB: ROC Curve

lier Hough Forest again removed usefull parts of S which resulted in a lower performance. Thus, the **TrBoost** Hough Forest is the most reliable method which transfers the most usefull parts of S and also avoids negative transfer.

The **PCA** Hough Forest and the **SparseCoding** Hough Forest showed the lowest performance on this task. Thus, vision features should be favoured over the feature representations found by these methods.

4.6 Experiment Conclusions

Our experiments were based on a Transfer Learning setting with a target training set T and a source training set S where information should be transferred from S to T. We evaluated Random Forest based methods with Transfer Boosting (RFTB), Transfer Learning with Outlier Detection (RF-O), Random Forest based methods with PCA (RF+PCA) and Random Forest based methods with Spase Coding (RF+SC).

RFTB showed the best use of additional data S. This method could transfer the highest amount of usefull information and avoid negative transfer very well. RF-O showed to remove usefull information from S when domains of T and S were strongly related. When the relationship was weak, RF-O showed to lower negative transfer but could not

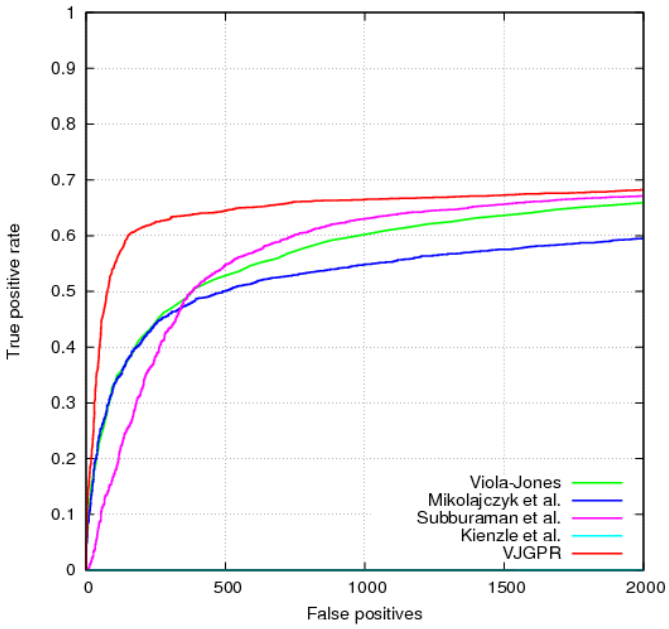


Figure 4.13: FDDB: Official ROC Curve [19]

avoid it as well as RFTB. Thus, RFTB should always be used when some additional data S is available. The additional computation time is moderate. In a worst case scenario the performance with Transfer Boosting should be at least as good as without Transfer Learning at all. Transfer Boosting is also easy to combine with classifiers different from Random Forests.

Self-Taugh Learning methods Transfer Learning with PCA and Sparse Coding have been used to learn feature representations for image data. Therfore, they use T and S so that additional information from S is transferred to learned feature representations. In combination with Random Forest based methods these learned feature representations showed to yield lower performance than traditional vision features such as pixel pair features or Haar Features. Thus, vision features should be preferred.

4.7 Chapter Summary

This chapter presented the results of our experiments. We compared the results of our different Random Forest based methods with Transfer Learning to results of Random Forest based methods without Transfer Learning. We also compared our results of Random Forest based methods with Transfer Learning to other state-of-the-art methods.

In sections 4.1 and 4.2 we showed our results for two classification tasks on different versions of the 20Newsgroups data set. This set was designed for transfer learning purpose. We compared the results of a simple Random Forest to the results of Random Forests which used additional auxiliary data during the training phase. We also presented some reference results which were achieved by other methods for classification.

In section 4.3 we showed our results on a character classification task. We compared Random Forests using pixel pair features, haar features and raw pixel values to a Random Forest using PCA features and in one case sparse features. The latter two methods also used additional data from a handwritten digit database to compute the feature representations. We also presented some reference results from [44] for this task.

In section 4.4 we performed a pedestrian detection task on the TUD-pedestrian data set [3]. We compared the results of Hough Forests with different feature representations to Hough Forests using axuiliary data from the INRIA person data set. We also compared PCA features and sparse features to other feature representations which are typical for vision tasks. Our results were also compared to the results from Gall et al. [15] and Barinova et al. [4] which used the same set of data.

In section 4.5 we showed our results for a face detection task on the FDDB [20] data

set. We compared the simple Hough Forest methods with different feature representations to Hough Forests which used additional auxiliary data from the GENKI-4K [36] data set. The results for different feature representations were also compared among each other.

Section 4.6 summarized the results of our experiments. The usefullnes of our Tranfer Learning methods were also discussed within this section.

Chapter 5

Conclusion and Outlook

Contents

5.1 Conclusion

In section 3 we presented our methods to combine Random Forests and Transfer Learning. The Transfer Learning methods we tested in our experiments where a simple combination of training data and auxiliary data with and without outliers in the auxiliary set, Transfer Boosting, PCA and Sparse Coding. PCA and Sparse Coding also introduced new feature representations which have been compared to traditional vision features such as pixel pairs and Haar Features.

Our experiments in section 4 showed that the use of an auxiliary data set can improve the performance of Random Forest based methods. The experiments on the 20newsgroup data sets in sections 4.1 and 4.2 showed how our methods are able to cope with negative transfer. Transfer Boosting avoided negative transfer better than all other tested methods. This implies that Transfer Boosting is a valid choice when additional data is used which is only weakly related to the target domain. For the vision tasks we showed in section 4.4 and in section 4.5 the Hough Forest with Transfer Boosting showed slightly better performance than the Hough Forest which was simply adding the auxiliary data to the training data. The removing of outliers in the auxiliary set lowered the performance. This happened because the auxiliary data we used was still strongly related to the original training data. In this case the outlier patches were providing useful information to refine

the learned model. It can be difficult to find out how much an auxiliary data set is related to the original training set. Thus, Transfer Boosting is the best choice which can ensure a good performance while reducing the risk that the learned model is polluted by outlying examples.

For PCA and Sparse Coding our experiments showed that traditional vision features like pixel pairs and haar features yield better results in combination with Random Forest classifiers. The experiments in section 4.3 show that PCA and Sparse Coding features are on a par with the raw pixel value representation. The computation of PCA features is at least as fast as for pixel pairs and haar features while the computation time for sparse features is very high. Thus, vision features are superior to these Self-Taugh Learning methods.

5.2 Outlook

The detection tasks in section 4.4 and in section 4.5 could be repeated with different auxiliary data sets which are less related to the training data. This should indroduce negative transfer when both sets are only combined without special Transfer Learning methods. In an optimal case a detector trained on only the training set should perform better than a detector using the auxiliary data as well during training. Transfer learning approaches can then be used to find out whether they can improve the quality of the learned model.

The framework we used for our experiments could be expanded by new Transfer Learning methods and feature representations for image based tasks. Even the Random Forest itself could be exchanged by other classifiers such as support vector machines. All experiments we discussed can be repeated with different methods. The framework also includes loading and saving schemes for some popular machine learning data formats.

The Hough Forest detector we used could be extended so that it is possible to learn and detect multiple classes instead of only foreground and background. Therefore, a seperate hough image pyramid needs to be created for each class. The background would also need to be defined as seperate class in this case. The Transfer Learning methods we used and the Random Forest itself are able to handle multiple classes. The postprocessing method we used could then be applied to the hough images for each class seperately. Thus, objects of different scales can be trained and detected. Another interesting experiment would be to combine data sets which were used for experimens in section 4.4 and section 4.5. This approach is known as multi-task learning [9] and could improve the performance on both

data sets.

Bibliography

[1] Agarwal, S., Awan, A., Roth, D., and Society, I. C. (2004). Learning to detect objects in images via a sparse, part-based representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:1475–1490.

[2] Amit, Y. and Geman, D. (1997). Shape quantization and recognition with randomized trees. *Neural Computation*, 9:1545–1588.

[3] Andriluka, M., Roth, S., and Schiele, B. (2008). People-tracking-by-detection and people-detection-by-tracking. In *CVPR*. Online Reference: http://www.mis.tu-darmstadt.de/node/382, Visited on October 30, 2011.

[4] Barinova, O., Lempitsky, V., and Kohli, P. (2010). On detection of multiple object instances using hough transforms. In *CVPR*, pages 2233–2240.

[5] Berg, T. L., Berg, A. C., Edwards, J., and Forsyth, D. (2004). Faces in the wild dataset. Online Reference: http://tamaraberg.com/faceDataset/index.html, Visited on October 30, 2011.

[6] Breiman, L. (1996a). Bagging predictors. In *Machine Learning*, volume 24, pages 123–140.

[7] Breiman, L. (1996b). Out-of-bag estimates. Technical report, Statistics Department, University of California, Berkeley, CA. 94708.

[8] Breiman, L. (2001). Random forests. *Machine Learning Journal*, 45:5–32. Online Reference: http://www.stat.berkeley.edu/~breiman/RandomForests/, Visited on October 30, 2011.

[9] Caruana, R. and Pomerleau, D. (1997). Multitask learning. In *Machine Learning*, volume 28, pages 41–75.

[10] Dai, W., Yang, Q., rong Xue, G., and Yu, Y. (2007). Boosting for transfer learning. In *Proc. 24th Int'l Conf. Machine Learning*, pages 193–200.

[11] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *International Conference on Computer Vision & Pattern Recognition (CVPR)*, volume 2, pages 886–893.

[12] Daumé, H. and Marcu, D. (2006). Domain adaptation for statistical classifiers. In *Journal of Artificial Intelligence Research*, volume 26, pages 101–126.

[13] Dietterich, T. G. and Lathrop, R. H. (1997). Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89:31–71.

[14] Fan, W. (2011). Collection of data and software for machine learning problems. Online Reference: http://www.cs.columbia.edu/~wfan/software.htm, Visited on October 30, 2011.

[15] Gall, J. and Lempitsky, V. (2009). Class-specific hough forests for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, pages 1022–1029. Online Reference: http://www.vision.ee.ethz.ch/~gallju/projects/houghforest/index.html, Visited on October 30, 2011.

[16] Gao, J., Fan, W., Jiang, J., and Han, J. (2008). Knowledge transfer via multiple model local structure mapping. In *Proc. 14th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, pages 283–291.

[17] Geman, D., Amit, Y., and Wilder, K. (1997). Joint induction of shape features and tree classifiers. *IEEE Trans. PAMI*, 19:1300–1305.

[18] Heckman, J. (1979). Sample selection bias as a specification error. In *Econometrica*, pages 153–161.

[19] Jain, V. and Learned-Miller, E. (2010a). Fddb: A benchmark for face detection in unconstrained settings. Technical Report UM-CS-2010-009, University of Massachusetts, Amherst.

[20] Jain, V. and Learned-Miller, E. (2010b). Fddb: Face detection data set and benchmark. Online Reference: http://vis-www.cs.umass.edu/fddb/, Visited on October 30, 2011.

[21] Jiang, J. (2008). A literature survey on domain adaptation of statistical classifiers. Online Reference: http://sifaka.cs.uiuc.edu/jiang4/domain_adaptation/survey/, Visited on October 30, 2011.

[22] Jiang, J. and Zhai, C. (2007). Instance weighting for domain adaptation in nlp. In *ACL 2007*, pages 264–271.

[23] Joachims, T. (1998). Making large-scale support vector machine learning practical. In Schölkopf, B., Burges, C. J. C., and Smola, A. J., editors, *Advances in Kernel Methods: Support Vector Learning*, pages 41–56. The MIT Press. Online Reference: http://svmlight.joachims.org/, Visited on October 30, 2011.

[24] Jones, M. J. and Viola, P. (2003). Face recognition using boosted local features. Technical Report TR2003-25, Mitsubushi Electric Research Laboratories.

[25] Kim, H. and Loh, W.-Y. (2001). Classification trees with unbiased multiway splits. *Journal of the American Statistical Association*, 96:589–604.

[26] Kim, H. and Loh, W.-Y. (2003). Classification trees with bivariate linear discriminant node models. *Journal of Computational and Graphical Statistics*, 12:512–530.

[27] Lee, H., Battle, A., Raina, R., and Ng, A. Y. (2007). Efficient sparse coding algorithms. In *NIPS*, pages 801–808. Online Reference: http://ai.stanford.edu/~hllee/softwares/nips06-sparsecoding.htm, Visited on October 30, 2011.

[28] Leibe, B., Leonardis, A., and Schiele, B. (2008). Robust object detection with interleaved categorization and segmentation. In *International Journal of Computer Vision*, volume 77, pages 259–289.

[29] Leibe, B. and Schiele, B. (2003). Interleaved object categorization and segmentation. In *British Machine Vision Conference*, pages 759–768.

[30] Liao, X., Xue, Y., and Carin, L. (2005). Logistic regression with an auxiliary data source. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, pages 505–512. ACM Press.

[31] Loh, W.-Y. (2009). Improving the precision of classification trees. *Annals of Applied Statistics*, 3:1710–1737.

[32] Loh, W.-Y. and Shih, Y.-S. (1997). Split selection methods for classification trees. *Statistica Sinica*, 7:815–840.

[33] Loh, W.-Y. and Vanichsetakul, N. (1988). Tree-structured classification via generalized discriminant analysis. *Journal of the American Statistical Association*, 83:715–728.

[34] Marée, R., Geurts, P., Piater, J., and Wehenkel, L. (2005). Random subwindows for robust image classification. In *CVPR*, pages 34–40. IEEE.

[35] MPLab (2011a). The MPLab GENKI Database. Online Reference: http://mplab.ucsd.edu, Visited on October 30, 2011.

[36] MPLab (2011b). The MPLab GENKI Database, GENKI-4K Subset. Online Reference: http://mplab.ucsd.edu, Visited on October 30, 2011.

[37] Olshausen, B. A. and Fieldt, D. J. (1997). Sparse coding with an overcomplete basis set: a strategy employed by v1. *Vision Research*, 37:3311–3325.

[38] OpenCV (2011). *OpenCV online Reference and Download*. Open Source Computer Vision by Intel and Willow Garage. Online Reference: http://opencv.willowgarage.com/wiki/, Visited on October 30, 2011.

[39] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359. Online Reference: http://www1.i2r.a-star.edu.sg/~jspan/SurveyTL.htm, Visited on October 30, 2011.

[40] Piotr, D. and Vincent, R. (2011). Piotr's image & video toolbox for matlab. Online Reference: http://vision.ucsd.edu/~pdollar/toolbox/doc/, Visited on October 30, 2011.

[41] Quinlan, J. R. (1992). Learning with continuous classes. In *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, pages 343–348. World Scientific.

[42] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann. ISBN: 1-55860-238-0.

[43] Quionero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. (2009). *Dataset Shift in Machine Learning*. MIT Press. ISBN: 0-262-17005-1.

[44] Raina, R., Battle, A., Lee, H., Packer, B., and Ng, A. Y. (2007). Self-taught learning: Transfer learning from unlabeled data. In *Proceedings of the Twenty-fourth International Conference on Machine Learning*, pages 759–766.

[45] Schapire, R. E., Freund, Y., Barlett, P., and Lee, W. S. (1997). Boosting the margin: A new explanation for the effectiveness of voting methods. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 322–330.

[46] Shi, X., Fan, W., and Ren, J. (2008). Actively transfer domain knowledge. In *Proc. European Conf. Machine Learning and Knowledge Discovery in Databases (ECML/PKDD 08)*, pages 342–357.

[47] Shimodaira, H. (2000). Improving predictive inference under covariate shift by weighting the log-likelihood function. In *Journal of Statistical Planning and Inference, 90 (2)*, pages 227–244.

[48] Shlens, J. (2005). A tutorial on principal component analysis. In *Systems Neurobiology Laboratory, Salk Institute for Biological Studies*. Online Reference: http://www.snl.salk.edu/˜shlens/, Visited on October 30, 2011.

[49] Timofeev, R. (2004). Classification and regression trees (cart) theory and applications. Master's thesis, CASE - Center of Applied Statistics and Economics Humboldt University, Berlin.

[50] Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition (CVPR 2001)*, volume 1, pages 511–518.

[51] Wu, P. and Dietterich, T. G. (2004). Improving svm accuracy by training on auxiliary data sources. In *ICML*, pages 871–878.

[52] Zadrozny, B. Z. (2004). Learning and evaluating classifiers under sample selection bias. In *In International Conference on Machine Learning ICML04*, pages 903–910.