Autodetektion mit AdaBoost

Magisterarbeit
an der
Technischen Universität Graz
vorgelegt von
Helmut GRABNER

Institut für Maschinelles Sehen und Darstellen (ICG)

Technische Universität Graz

in Kooperation mit

Betreuer und Begutachter:
Univ.Prof. Dr. Horst BISCHOF

Graz, 18. Oktober 2004

AdaBoost for Cardetection

Master's Thesis
at
Graz University of Technology
submitted by
Helmut GRABNER

Institute for Computer Graphics and Vision (ICG)

Graz University of Technology

in cooperation with

Advisor:
Univ.Prof. Dr. Horst BISCHOF

Graz, October 18, 2004

This thesis is written in German.

Kurzfassung

In dieser Arbeit wird die Detektion, also das Finden von Objekten, in statischen Bildern behandelt. Das eingesetzte Verfahren ermöglicht eine sehr schnelle Verarbeitung, obwohl hohe Detektionsraten erreicht werden können. Das Kernstück des Objekterkennungssystems stellt ein effizienter Klassifikator dar, für dessen Erstellung Methoden aus dem maschinellen Lernen eingesetzt werden. Es wird dabei anhand von Beispielen gelernt, speziell wird auf den Algorithmus AdaBoost eingegangen. Das System findet ein Objekt, in einem gegebenen Bild, indem dieser Klassifikator schrittweise an sehr vielen möglichen Positionen und Skalierungen evaluiert wird. Dabei werden im Allgemeinen mehrere Detektionen auftreten, die in einem Nachverarbeitungsschritt miteinander kombiniert werden. Es handelt sich dabei um ein Variante eines auf MeanShift basierenden clustering-Verfahrens.

Der Vorgang wird anhand der Detektion von Autos in Grauwertbilder demonstriert. Das System erreicht dabei eine Leistung, die vergleichbar mit den besten anderer Systemen ist.

Abstract

This thesis describes a method for object detection in static images. It allows a very fast processing while achieving high detection rate. The key contribution is to create a classifier that evaluates as efficient as possible. This classifier is the essential part of the object detection system. Therefore example-based methods from machine learning were used, specially the algorithm AdaBoost. The system finds an object by scanning this classifier over an exhaustive rang of possible locations und scales in an image. In general several detections will occur, which will be combined in a post-processing step by a variant of MeanShift clustering.

The process is shown with experiments of car detection in grey-scale images. The overall system yields detection performance comparable to the best previous systems.

Erklärung

Ich versichere hiermit, diese Arbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient zu haben.

Dank

Mein Dank gilt Herrn Dr. Horst Bischof für die Betreuung und rasche Begutachtung meiner Magisterarbeit. Weiters möchte ich den Mitarbeitern der Firma *Advanced Computer Vision*, besonders Herrn Dr. Csaba Beleznai, danken, die mich stets hilfsbereit unterstützt und meine Arbeit finanziell gefördert haben.

Besonderen Dank gebührt meinen Eltern, die mir das Studium nicht zuletzt durch ihre finanzielle Unterstützung ermöglicht haben. Last but not least möchte ich meiner Freundin Susanne danken, dass sie mich bei meiner Arbeit unterstützt hat und so verständnisvoll und geduldig war.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Das Erkennen sowie das Finden von unterschiedlichen Objekten stellt einen sehr großen Teil unseres alltäglichen Lebens dar. Wir erkennen andauernd die uns umgebenden Objekte, wie Menschen, Gesichter, Häuser, Hunde, Straßen, Autos, Kaffeetassen, usw. Der Mensch löst dieses Problem sehr genau und anscheinend mit geringem Aufwand. So kann unser Gehirn einerseits uns bekannte unterschiedliche Gesichter trennen und den dementsprechenden Personen zuordnen. Auf der anderen Seite erkennen wir auch solche Gesichter als Gesichter, die wir noch nie im Leben gesehen haben. In dieser Arbeit soll jedoch nicht versucht werden zu verstehen, wie wir Menschen das vollbringen. Das Ziel ist es ein sowohl schnelles als auch robustes, automatisches Objektdetektionssystem zu erstellen, das es ermöglicht verschiedene Objekte in einem statischen Bild zu detektieren, also zu finden. Die Entwicklung von Methoden für diese Aufgabenstellung ist ein zentraler Teil der aktuellen Forschung.

1.1 Zielsetzung

Ein Bild beinhaltet sehr viel Information - hunderte oder tausende von Pixelwerten, wobei jedes vielleicht eine wichtige, entscheidende Information beinhaltet. Dieser Sachverhalt wird treffend durch die wohlbekannte Phrase „Ein Bild sagt mehr als tausend Worte“ ausgedrückt. Genau durch diese Fülle von Informationen, ergeben sich Schwierigkeiten in der Bildverarbeitung. Die weitere Hürde, um ein verlässliches System zu erstellen, besteht darin, dass eine Instanz eines Objektes in verschiedenen Variationen auftreten kann. Autos können sich zum Beispiel in Größe, Aussehen, Farbe und weiteren Details, wie Lichter, Reifen, Spoiler, usw. unterscheiden. Dennoch handelt es sich immer um ein Auto, das von allen anderen Objekten und von dem Hintergrund unterschieden werden soll. Es muss also ein Modell erstellt werden, welches die spezifischen Eigenschaften des Objektes gut beschreibt und damit eine Unterscheidung ermöglicht.

Ein robustes System soll es weiters ermöglichen das reale 3D Objekt, das mittels einer Kamera auf ein 2D Bild abgebildet wird, unter verschiedenen Randbedingungen zu

erkennen. So soll das System mit unterschiedlichen Beleuchtungen und partiellen Überdeckungen durch andere Objekte zurechtkommen. Eine weitere Herausforderung besteht darin, das Objekt unter verschiedenen Ansichten zu erkennen. Da das System allgemein eingesetzt werden soll, muss es versuchen weitgehend ohne Vorwissen über den Aufbau der betrachteten Umgebung zurechtzukommen. Das heißt, der Hintergrund, von dem das Objekt zu trennen ist, kann „beliebig“ gestaltet sein. Für den Fall der Autodetektion, zum Beispiel, stellt das eine wesentlich schwierigere Aufgabe dar, als wenn eine Kamera fix auf einer Autobahnbrücke angebracht ist und unter diesen kontrollierten Bedingungen eine Detektion liefern soll.

Für ein Objektdetektionssystem gibt es vielfältige Motivationen und praktische Anwendungsbereiche. Vor allem für komplexere Anwendung, zum Beispiel in Überwachungssystemen, ist es erforderlich zuerst einmal die einzelnen Objekte zu lokalisieren. Natürlich spielt besonders hier die Geschwindigkeit, bis die Ergebnisse vorliegen, eine wesentliche Rolle, da die Anwendungen meist in Echtzeit (*real-time*) betrieben werden.

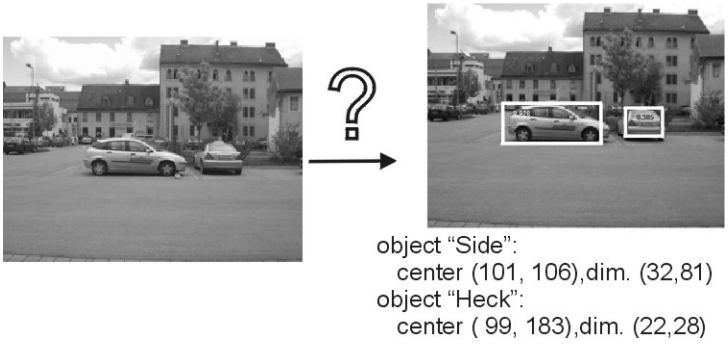

Abbildung 1.1: Ziel der Objektdetektion. Alle Objekte die eventuell in einem gegebenen Bild vorhanden sind sollen in Position und Größe gefunden werden.

Die angegebenen Punkte sollten alle weitgehend von dem, in dieser Arbeit, zu entwickelten Verfahren behandelt werden. Dies wird hier exemplarisch für die Detektion von Autos demonstriert. In einem gegebenen Bild soll also erkannt werden an welcher Position und in welcher Größe sich ein Auto befindet. Des Weiteren sollen auch verschiedene Ansichten von Autos erkannt werden. Und es soll festgestellt werden wie viele Autos in einem gegebenen Bild enthalten sind. Die grundlegende Aufgabenstellung ist in Abbildung 1.1 dargestellt.

1.2 Ansätze

In der Literatur finden sich verschiedene Ansätze, die versuchen die oben beschriebene Zielsetzung zu bewältigen. Vor allem auf dem Gebiet der Erkennung und Detektion von Gesichtern wurden in der vergangenen Zeit einige unterschiedliche Ansätze entwickelt. So werden zum Beispiel wissensbasierende Modelle (*model-based*) erstellt, bei denen für das zu findende (interessante) Objekt ein Modell definiert wird und das System versucht eine

Übereinstimmung mit Teilen des Bildes zu finden. Eine andere Möglichkeit, ist es anhand von Beispielen (*example-based*) einen Detektor zu gestalten. Das System „lernt“ aus gegebenen positiven und negativen Beispielen, die Objekte anhand von Merkmalen automatisch zu unterscheiden. Dabei werden wesentlich Methoden aus dem maschinellen Lernen eingesetzt. Durch die Anwendung eines geeigneten Lernalgorithmus wird ein Klassifikator erstellt, der die Entscheidung, auf Grund der Merkmale, fällt. Die Auswahl und die Representation dieser Merkmale stellt dabei einen wichtigen Schritt dar. Es können einerseits die rohen Pixeldaten verwendet werden, oder andererseits auch eine Beschreibung auf höheren Ebenen stattfinden, bei der Transformationen auf die Intensitätswerte der Pixel angewendet werden. Es wurden verschiedene Methoden vorgestellt, die mittels spezieller Merkmale durch Filterung und geeignetes Kombinieren versuchen das Problem zu lösen. Weiters werden statistische Beschreibungen, textturbasierter Analysen und Beschreibung der Objekte mit einzelne Komponenten verwendet. Auch werden, bei den verschiedenen Ansätzen, unterschiedliche Lernverfahren eingesetzt um den Klassifikator zu erstellen [37], [53], [25], [35], [38], [45], [57], [56], [58], [49], [5]. Ein grober Überblick von erfolgreichen Ansätzen bei der Gesichtsdetektion findet sich zum Beispiel bei Fasel et al. [18]. Auch für den Bereich Autodetektion wurden etliche verschiedene Ansätze eingesetzt, wie etwa bei [46], [43], [2], [1], [19], [24], [44], [27], [36].

Der Ansatz, der in dieser Arbeit verwendet wird, basiert auf den Arbeiten von Viola und Jones [54], [55]. Der Vorteil dieser Methode besteht darin, dass sie es ermöglicht ein schnelles System zu erstellen, das dennoch im Stande ist gute und robuste Ergebnisse zu liefern. Kernstück stellt ein Klassifikator dar, der durch Auswahl von wenigen wichtigen Merkmalen erstellt wird. Für die Auswahl und die Kombination dieser Merkmale wurde ein Ansatz aus dem maschinellen Lernen gewählt, der sich in den vergangenen Jahren steigender Beliebtheit erfreute: *Boosting*. Dabei wird versucht einen (starken) Klassifikator, mit niedrigem Fehler, durch Kombination mehrerer verschiedener (schwacher) Klassifikatoren, die einen relativ großen Fehler haben dürfen, zu „boosten“. Mit *AdaBoost* als spezieller Algorithmus wurden bisher sehr gute Ergebnisse in den unterschiedlichsten Bereichen erzielt, zum Beispiel in der Schrifterkennung (OCR). Die große Stärke von *Boosting* ist, dass es auf unterschiedliche Arten von meist einfachen Lernalgorithmen angewendet werden kann. In diesem konkreten Fall werden für die Bestimmung der einzelnen schwachen Klassifikatoren Merkmaleswerte verwendet, die den Koeffizienten einer Wavelet Transformation entsprechen. Ein schwacher Klassifikator benützt dabei nur die Information eines Merkmals. Das System wählt sich somit selbst die benötigten Merkmale, aus einer großen Menge von möglichen, aus und bildet mit ihnen einen starken Klassifikator. Die gesamte Implementierung ist so ausgelegt, dass sie eine möglichst effiziente Berechnung ermöglicht. Auf der einen Seite kann die benötigte Berechnung der Merkmale sehr effizient erfolgen. Auf der anderen Seite wird versucht, durch Erstellen einer Kaskade, den durchschnittlichen Rechenaufwand einer Evaluierung zu senken.

Die ursprüngliche Anwendung konzentrierte sich auf die Detektion von Gesichter. Die Idee wurde jedoch auch in einigen anderen Anwendungen integriert [17], [3], [50], [34], [28], [33], bei denen meist sehr gute Ergebnisse erzielt werden konnten. Die aufgezeigten Vorteile stellen den Grund dar, warum dieser Ansatz, als Teil des Objekterkennungssystem,

in dieser Arbeit eingesetzt wurde.

1.3 Übersicht

In den nachfolgenden Kapiteln wird von Grund auf die Erstellung eines Objektdetektionssystems beschrieben. Der Hauptteil beschäftigt sich damit einen möglichst schnellen und guten Klassifikator, laut dem oben beschriebenen Ansatz, zu erstellen. Die einzelnen Detektionen werden danach in einem geeigneten Nachverarbeitungsschritt verarbeitet und liefern hinterher die endgültigen Ergebnisse des gesamten Detektionssystems. Auf Testdatensätze für die seitliche Autodetektion liefert des System sehr gute Ergebnisse. Die Leistung ist durchaus mit den besten bestehenden Systemen vergleichbar, die in der Literatur dokumentierten sind. Der restliche Teil der Arbeit ist dabei wie folgt gegliedert.

- In Kapitel 2 wird zunächst die zugrundeliegende Theorie näher beschrieben und ein allgemeines Modell für die Erstellung eines Objekterkennungssystems aufgestellt. Der wesentliche Bestandteil dieses Modells bildet ein Klassifikator, der die notwendigen Entscheidungen liefert. Um einen solchen Klassifikator zu erstellen wird eine Methode aus dem maschinellen Lernen eingesetzt. Es wird dabei hauptsächlich auf den Algorithmus *AdaBoost* eingegangen und die Erkenntnisse anhand von zwei Beispielen veranschaulicht.

- Darauf aufbauend wird in Kapitel 3 die konkrete Anwendung des Klassifikators beschrieben. Dieser Klassifikator bildet das Kernstück des Objektdetektionssystems. Für jedes Objekt oder jede Ansicht eines Objektes muss dabei ein eigener Klassifikator erstellt werden.

  Um ein gegebenes Bild zu evaluieren, wird dieses schrittweise durchsucht. Dabei werden im Allgemeinen mehrere Detektion auftreten. In einem Nachverarbeitungsschritt werden alle auftreten Detektion mittels einer Variante des *adaptiven MeanShift* Algorithmus weiterverarbeitete und miteinander kombiniert um zu dem endgültigen Ergebnis zu gelangen

- Anhand einer konkreten Anwendung wird, in Kapitel 4, ein Objekterkennungssystem erstellt. Es wird zum Einen die Erstellung (das Trainieren) des in Kapitel 3 definierten Klassifikators beschrieben. Zum Anderen wird dieser Klassifikator eingesetzt um Autos in einem gegebenen Bild in Position, Größe und unter verschiedenen Ansichten zu lokalisieren. Die erreichte Performance sowie Vergleiche zu anderen Methoden in der Literatur werden durchgeführt und dokumentiert.

- Im letzten Kapitel wird nochmals eine kurze Zusammenfassung gegeben. Schlussfolgerungen und weitere Verbesserungsvorschläge werden zusammengefasst.

Kapitel 2

Theoretische Grundlagen

In diesem Kapitel soll das notwendige theoretische Hintergrundwissen vermittelt werden auf dem dann die Anwendung (Kapitel 3) aufbaut. Die eingeführten Begriffe erlauben es dann auch die resultierenden Ergebnisse anhand einer konkreten Anwendung (Kapitel 4) zu interpretieren und zu vergleichen.

In den ersten einführenden Abschnitten soll ein Überblick über die Grundlagen des maschinellen Lernens und speziell der maschinellen Mustererkennung gegeben werden. Eine formale Beschreibung der verwendeten Begriffe ist im Anschluss gegeben. Es wird gezeigt, dass es sich um überwachtes Lernen handelt und es dazu benutzt werden kann aus einer Reihe von Beispielen ohne Vorwissen ein System zu erstellen, das es ermöglicht Objekte zu erkennen bzw. zu detektieren. Das zu durchlaufende Verfahren wird anhand eines Modells vorgestellt. Bezogen auf dieses Modell, wird in den folgenden Kapiteln das Detektionssystem erstellt, wobei auf die einzelnen Schritte eingegangen wird.

Der Hauptteil dieses Kapitels beschäftigt sich dann mit einem konkreten Lernansatz, dem *Boosting*. Boosting stellt den zentralen Bestandteil des in dieser Arbeit entwickelten Detektors dar. Der vorgestellte Algorithmus *AdaBoost* wird analysiert und im Anschluss sind zwei Beispiele zur Veranschaulichung gegeben.

2.1 Maschinelles Lernen

Ziel ist es ein System zu erstellen, das basierend auf bestimmten Eingabewerten (*Input*) einen bestimmten Ausgabewert (*Output*) erzeugt. Dieser (meist nichtlineare) Zusammenhang ist jedoch nicht ausprogrammiert sondern wird „gelernt“. Dazu sei ein Vergleich zweier Möglichkeiten, die aus der künstlichen Intelligenz (KI) Forschung entstanden sind, gegeben.

Expertensysteme Wissensbasierende Systeme bilden ein Modell der Wirklichkeit nach. Das Wissen ist dabei in einer Wissensbasis, meist in Form von Fakten und Regeln, gegeben. Diese Regeln einer Wissensbasis werden von den Erfahrungen und dem Wissen

von Experten extrahiert. Durch logisches Schließen lassen sich nun zum Beispiel Fragen beantworten. Die Schwierigkeit besteht dabei dieses meist komplexe Regelwerk zu erstellen und das Wissen der jeweiligen Experten korrekt abzubilden.

Maschinelles Lernen Hierbei wird kein Experte benötigt um ein Modell zu erstellen, sondern der Zusammenhang zwischen Input und Output wird automatisch anhand von gegebenen Beispielen ermittelt. Es wird also anhand von Beispielen (Trainingsdaten), oder allgemeiner anhand von Beobachtungen „gelernt“. Der Lernalgorithmus kann direkt, indirekt oder auch keine Information über die Richtigkeit seiner Entscheidungen erhalten um seine Ergebnisse zu verbessern. Dementsprechend teilt sich maschinelles Lernen grundlegend in überwachtes (*supervised*), *reinforcement* und unüberwachtes (*unsupervised*) Lernen. Diese Aufteilung ist in Abbildung 2.1 dargestellt.

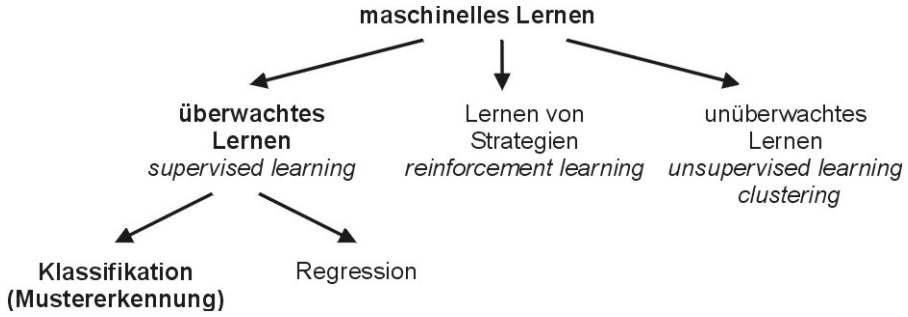

Abbildung 2.1: Allgemeiner Überblick: maschinelles Lernen

Beim überwachten Lernen stellt ein Lehrer (*Teacher*) den korrekten, gewünschten Ausgabewert direkt zur Verfügung. Dabei können die Ausgabewerte diskrete Klassen (Kategorien) darstellen und man spricht dann von Klassifikation oder Mustererkennung, auf das in der Folge näher eingegangen wird. Handelt es sich um kontinuierliche Werte spricht man von einem Regressionsproblem und es wird versucht eine Funktion zu approximieren. Beim Reinforcement Learning erhält der Lernalgorithmus eine Reaktion auf seine Aktion als Belohnung oder Bestrafung, aber es wird ihm nicht gesagt, was die richtige Aktion ist, das muss er selbst herausfinden. Beim unüberwachten Lernen oder *clustering* gibt es keinen expliziten Lehrer. Das System formt natürliche Gruppen der gegebenen Beispiele.

2.2 Maschinelle Mustererkennung

Maschinelle Mustererkennung soll einem gegebenen Muster[1] (Input) eine eindeutige Klasse (Output) zuordnen. Die Informationen sind in Form von Beispielen gegeben, aus denen auf

[1]Da diese Arbeit dem Gebiet der Bildverarbeitung zugeordnet ist handelt es sich hierbei um Bilder die typischerweise mit einer Kamera aufgenommen wurden. Es können jedoch verschiedene Daten wie Sprache oder andere Messwerte, auch kombiniert, als Input herangezogen werden.

den zugrundeliegenden datengenerierenden Prozess, mittels Methoden des maschinellen Lernens, geschlossen werden soll.

2.2.1 Allgemeines Modell

Das allgemeine Modell für ein Objekterkennungs-/Objektdektionssystems, ist in Abbildung 2.2 dargestellt und folgt dabei im Wesentlichen dem Buch von Duda und Hart et al. [16].

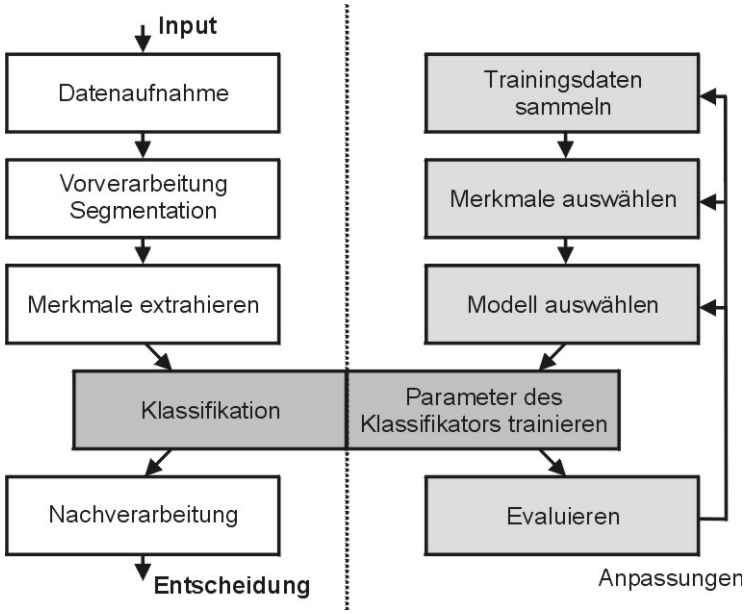

Abbildung 2.2: Schematische Darstellung eines Objekterkennungssystems; (links) Testphase; (rechts) Trainingsphase

Kernstück dabei ist ein Klassifikator, der die notwendigen Entscheidungen liefert. Auf der rechten Seite ist der Pfad für Anpassung der Klassifikationsparameter (*Trainingsphase*) dargestellt. Anhand eines so trainierten Klassifikators können danach, wie der rechte Pfad (*Testphase*) beschreibt, Entscheidungen über neue gegebene Daten gefällt werden. Die Performance wird vor allem dadurch bestimmt, wie sich das System auf neuen Daten, also in der Anwendung, verhält. Diese Fähigkeit wird durch die Generalisierungsfähigkeit ausgedrückt und kennzeichnet wie gut der Prozess aus den Trainingsbeispielen erfasst werden konnte.

Im Folgenden werden zuerst die einzelnen Schritte zum Trainieren eines Klassifikators beschrieben. Wie auch in der Abbildung angedeutet handelt es sich dabei um einen zyklisch durchlaufenen Prozess. Es sei an dieser Stelle auch darauf hingewiesen, dass die Trennung zwischen Trainings- und Testphase nicht immer so strikt einzuhalten ist. Es wurde hier *offline Lernen* beschrieben, das heißt für den Trainingsvorgang ist bereits zu Beginn eine Menge von Beispielen vorhanden. Im Gegensatz dazu können im *online Lernen (adaptive Systeme)* die Parameter des bestehenden Klassifikators im Betrieb immer wieder neu angepasst werden.

Trainingsdaten sammeln Anders als bei Expertensystemen wird nur aus diesen gegebenen Beispielen Wissen über den dahinterliegenden Prozess gewonnen. Man erkennt nun auch das Problem: Ist kein oder nur sehr wenig Vorwissen (Expertenwissen) vorhanden und kann somit kein Modell des Systems beschrieben werden, so müssen möglichst viele Beispiele gegeben sein um aus denen das Wissen über den dahinterliegenden Prozess zu gewinnen. Es ist zwar möglich eine grobe Durchführbarkeitsstudie mit relativ wenigen typischen Daten zu machen. Um für das endgültige System jedoch gute Performance Werte sicherstellen zu können müssen, in der Regel, sehr viele Trainingsdaten aufgenommen werden. Des Weiteren ist es sehr wichtig, dass die Beispiele, die dem Lernalgorithmus übergeben werden, typische Beispiele aus der gesamten Bandbreite des Prozesses sind. Das Abschneiden in der Testphase, also die Generalisierungsfähigkeit, wird ansonsten sehr schlecht sein. Das gründliche Beschaffen von Trainingsdaten kann einen sehr großen Teil des gesamten Entwicklungsprozesses betragen.

Merkmale auswählen Die Grenze zwischen dem Bestimmen von Merkmalen (*Features*, Attribute) und der eigentlichen Klassifikation kann relativ willkürlich gezogen werden. Werden ideale Merkmale berechnet, so ist die Aufgabe des Klassifikators trivial. Auf der anderen Seite benötigt ein ausgezeichneter Klassifikator kaum die Hilfe von gut extrahierten Merkmalen. Diese Unterscheidung wird wegen praktischen und nicht wegen theoretischen Gründen vorgenommen.

Ziel der Merkmale ist es charakteristische Eigenschaften von gleichen Objekten (Klassen) gut zu beschreiben. Die Berechnung der Merkmale bei Objekten einer Klasse sollte also ähnliche Werte liefern, für andere Objekte jedoch unterschiedlich sein. Ziel ist es solche Merkmale zu finden, die unabhängig zu irrelevanter Information der Inputs sind[2] und dessen Werte eine „ einfache“ Trennung der unterschiedlichen Klassen ermöglichen (*discriminative features*).

Die Auswahl der Merkmale stellt einen kritischen Schritt dar, da der nachfolgende Klassifikator nur mit Hilfe dieser Information Entscheidungen treffen kann. Auch ist es sehr problemspezifisch und daher spielt das entsprechende Vorwissen und Erfahrungen von Spezialisten eine große Rolle[3].

Modell auswählen In diesem Schritt wird festgelegt, welches Modell zur Beschreibung der gegebenen Daten verwendet werden soll. Für den konkreten Aufbau gibt es die verschiedensten Möglichkeiten und Theorien, so kann es sich zum Beispiel um ein künstliches neuronales Netz oder einen Entscheidungsbaum handeln. Für all diese Repräsentationen sind Lernalgorithmen entworfen worden.

[2] In der Bildverarbeitung zum Beispiel rotations-, translations- und skalierungsinvariante Merkmale.

[3] Wie oben erwähnt wird im Maschinellen Lernen, im Gegensatz zu Expertensystemen, im Wesentlichen anhand von gegebenen Beispielen ein Modell aufgestellt. Ist jedoch (beschränktes) Vorwissen vorhanden kann das in den Entwicklungsprozess (Merkmale und Modell) soweit möglich integriert werden.

Parameter des Klassifikators trainieren Die Aufgabe eines Klassifikators ist es, basierend auf den extrahierten Merkmalen eines gegebenen Inputs, eine möglichst korrekte Zuweisung zu einer Kategorie zu finden. Dabei werden die Parameter des gegebenen Modells mittels eines Lernalgorithmus angepasst. Da eine perfekte Klassifikation im Allgemeinen unmöglich ist, können allgemeiner, für einen Input, den Klassen Wahrscheinlichkeitswerte zugewiesen werden.

Dabei stellen sich einige Fragen: Wie kann man sicher sein, dass das Modell mächtig genug ist um eine gute (einfache) und stabile Lösung zu liefern? Wie viel Zeit und Speicher wird für die Trainingsphase benötigt und wie lange dauert die Evaluierung eines neuen Beispiels?

Evaluierung Nachdem ein Klassifikator erstellt wurde, wird dessen Performance bestimmt und gegebenenfalls Änderungen an der Struktur (Merkmale, Modell) vorgenommen.

Ist bei diesem *offline Lernen* die Trainingsphase abgeschlossen, kann der erstellte Klassifikator eingesetzt werden. Die jeweiligen Schritte der Testphase (linker Pfad der Abbildung 2.2) sind wie folgt

Datenaufnahme Dieser Punkt stellt in der Praxis einen sehr wichtigen Bestandteil des gesamten Systems dar, da es wesentlich auf die physikalischen Eigenschaften (Auflösung, Verzerrungen, Signal zu Rausch Verhältnis,...) des Aufnahmegerätes ankommt. In dieser Arbeit wird jedoch hierauf nicht näher eingegangen.

Vorverarbeitung Die Roh-Daten können in geeigneter Weise aufbearbeitet werden. So kann zum Beispiel eine Normalisierung der Daten vorgenommen werden um Beleuchtungseffekte auszugleichen. Bei einem Objekterkennungssystem müssen die möglichen Objekte erst einmal von dem Hintergrund segmentiert werden um sie danach mittels des Klassifikators einer bestimmten Kategorie zuzuordnen.

Merkmale extrahieren Die Werte, der in der Trainingsphase definierten Merkmale werden ermittelt und gesammelt dem Klassifikator übergeben.

Klassifikation Anhand der extrahierten Merkmale wird eine Entscheidung getroffen um welches Objekt es sich handelt.

Nachverarbeitung Da die Ergebnisse der Klassifikation nicht perfekt sein werden, können diese unter Zuhilfenahme weiteren Wissens über den Aufbau verbessert werden.

2.2.2 Erkennung vs. Detektion

Das oben beschriebene Verfahren stellt ein allgemeines Modell für ein Objekterkennungssystem (*object recognition system*) dar. Das heißt, es soll festgestellt werden ob es sich um

ein bekanntes Objekt in einer gegebenen Datenbank handelt und wenn ja, um welches. Hierbei handelt es sich um eine multiklassen Entscheidung (spezielles Objekt vs. alle anderen Objekte). Im Gegensatz dazu liegt die Aufgabe eines Objektdetektionssystems (*object detection system*) darin, in einem gegebenen Bild alle möglichen Auftreten von Objekten einer Klasse zu lokalisieren, wenn solche vorhanden sind. Sowohl in Erkennungssystemen wie auch in Überwachungssystemen (*surveillance system*) stellt die Objektdetektion meist einen wichtigen Schritt der Vorverarbeitung (Segmentierung) dar um die Aufmerksamkeit auf interessante Ausschnitte zu lenken. Es stellt also ein zwei Klassen Problem dar (Objekt vs. kein Objekt), welches manchmal auch als *single class-classification* bezeichnet wird.

Bei der Autodetektion sollen zum Beispiel alle Typen von Autos erkannt werden, auch solche Typen, die der Lernalgorithmus zuvor noch nicht gesehen hat. Das bedeutet, dass der Klassifikator generalisieren soll und die wesentlichen Eigenschaften, die alle Autos besitzen, herauszuarbeiten hat. Bei der Objekt Identifikation (Spezialfall eines Erkennungssystems) sollen zum Beispiel die verschiedenen Typen von Autos erkannt werden. Es muss daher mehr Augenmerk auf die feinen Details gelegt werden. Es wird dabei natürlich nur erwartet, dass das System Autotypen erkennen kann, die auch zum Trainieren verwendet wurden.

Weites sei angemerkt, dass auch mehrere Detektionssysteme, die jeweils binäre Entscheidungen treffen, zu einem Erkennungssystem ausgebaut werden können. Dies führt also zu einer multiklassen Entscheidung, basierend auf den einzelnen binären Klassifikatoren [25]. In dieser Arbeit wird anhand von zwei Beispielen die Anwendung veranschaulicht. Dabei wird je ein Detektor für die seitliche und frontale/heck Ansicht erstellt. Kombiniert man diese beiden Klassifikatoren hat man einerseits ein Detektionssystem für beide Ansichten, andererseits aber auch eine feinere Unterscheidung die eine „Erkennung“ von Seite und Heck/Front erlaubt.

2.2.3 Begriffe

In folgender kurzer Zusammenfassung wird eine formale Beschreibung über die verwendeten Begriffe und Symbole gegeben. Sie bezieht sich vorwiegend auf überwachtes Lernen und speziell auf die Klassifikation. Diese Ausführungen folgen im Wesentlichen [31] und meinen persönlichen Mitschriften aus diesbezüglichen Lehrveranstaltungen an der TU-Graz sowie ergänzend aus [16], [32], [52].

Beispiel Ein Beispiel (*Sample*) wird im Allgemeinen durch eine endliche Menge $\mathcal{M} = \{M_1, ..., M_k\}$ von Merkmalen (*Features*, Attribute), denen ein Ausgangswert $y \in Y$ zugeordnet wird, charakterisiert. Formal handelt es sich also um einen Vektor $\langle m_1, ..., m_n, y\rangle \in M_1 \times ... \times M_k \times Y$. Die einzelnen Merkmale $m_i \in M_i$ können zu einem Merkmalsvektor $\mathbf{x} = \langle m_i, ..., m_k\rangle$ im Merkmalsraum $X = M_1 \times ... \times M_k$ zusammengefasst werden. Somit lässt sich ein Beispiel als $\langle \mathbf{x}, y\rangle \in X \times Y$ schreiben.

Datensatz Ein Datensatz im maschinellen Lernen ist eine Liste von Beispielen $S = \langle\langle \mathbf{x_1}, y_1\rangle, ..., \langle \mathbf{x_n}, y_n\rangle\rangle$. Das heißt den jeweiligen Merkmalsvektoren $\mathbf{x_i}$ wird jeweils ein Ausgangswert y_i zugeordnet.

Lernumgebung Eine Lernumgebung wir modelliert durch ein Wahrscheinlichkeitsmaß P auf der Menge von Beispielen $X \times Y$.

Lernen Beim überwachten Lernen betrachtet man ein Beispiel $\langle \mathbf{x_i}, y_i \rangle$ dessen Eingabe $\mathbf{x_i}$ und dessen korrekte Ausgabe y_i vorgegeben sind. Das heißt das Lernverfahren lässt sich formal auf das Bestimmen (Lernen) einer funktionalen Beziehung $f : X \rightarrow Y$ reduzieren. Hierfür ist partielles Wissen in Form eines Datensatzes (Menge von Beispielen) S mit $\langle \mathbf{x_i}, y_i \rangle = \langle \mathbf{x_i}, f(\mathbf{x_i}) \rangle$ gegeben. Mathematisch betrachtet handelt es sich also um ein Funktions-Approximationsproblem.

Beim offline Lernen besteht dieser Vorgang aus einer Trainings- und einer anschließenden Testphase. In der Trainingsphase ist die Aufgabe des Lerners, aufgrund eines Datensatzes von Trainingsbeispielen eine Hypothese zu generieren. Lernen findet nur dann statt, wenn die in den Daten vorhandene Information komprimiert wird, zum Beispiel indem man die dahinterliegenden Prinzipien erfasst.

Klassifikation Man spricht von einem Klassifikationsproblem, wenn der Wertebereich von f (bisher mit Y bezeichnet) eine endliche (im Allgemeinen recht kleine) Menge $C = \{c_1, ..., c_l\}$ ist. Die Abbildung f ist dann eine Klassifikationsfunktion $class : X \rightarrow C$ wobei die $c \in C$ als die Klassen bezeichnet werden. Im anderen Fall spricht man von einem Regressionsproblem. Sind bei der Klassifikation nur zwei Klassen vorhanden, bezeichnet man das auch als Detektion oder *single class-classification*.

Hypothese Beim Übergang von speziellen Beispielen $S \subseteq X \times Y$ zu einer vollständigen Abbildung $X \rightarrow Y$ wird das partielle Wissen über f (hypothetisch) auf den gesamten Definitionsbereich X erweitert. Eine Hypothese $H : X \rightarrow Y$ ist ein Modell für den zugrundeliegenden datengenerierenden Prozess.

Hypothesenklasse Eine Menge aller Hypothesen, die bei einem konkreten Lernverfahren betrachtet werden, wird als Hypothesenklasse $\mathcal{H}$ bezeichnet. Verschiedene Hypothesenklassen haben unterschiedliche Komplexität. Ein Komplexitätsmaß, das unabhängig von der Art, wie die Hypothese formuliert ist, kann mittels der *VC-Dimension* [11], [52] definiert werden. Zum Beispiel bilden alle Hypothesen die mit einem Entscheidungsbaum der Tiefe zwei generiert werden können eine Hypothesenklasse.

Lernalgorithmus Ein Lernalgorithmus A ist eine Funktion, die jedem Datensatz S von Beispielen eine Hypothese $H = A(S)$ zuordnet. Beziehungsweise: $A : (X \times Y)^\star \rightarrow Y^X$ wobei $(X \times Y)^\star = \bigcup_{l=1}^{\infty} (A \times B)^l$ und Y^X die Menge aller Funktionen von $X \rightarrow Y$ bezeichnet. Jeder praktische Lernalgorithmus kann nur Hypothesen aus einer bestimmten Hypothesenklasse erzeugen. Maschinelles Lernen kann man als Suche nach einer optimalen Hypothese in der Hypothesenklasse $H^\star \in \mathcal{H}$, die am besten zu den Trainingsbeispielen passt, verstanden werden. Neben der Wahl von $\mathcal{H}$ wird die Effizienz und die Qualität des Lernverfahrens durch das verwendete Suchverfahren bestimmt.

Da der mögliche Hypothesenraum im Allgemeinen sehr groß ist, spielen heuristische Methoden eine wichtige Rolle.

Fehler Es ist wichtig die Performance einer Hypothese bzw. eines Lernalgorithmus so genau wie möglich abzuschätzen und damit eine Hypothese zu evaluieren, was einen zentralen Bestandteil vieler Lernmethoden darstellt. Die folgenden Definitionen beziehen sich dabei auf den Fall der Klassifikation[4].

Wahre Fehler Der wahre Fehler $error_P$ einer Hypothese H bezüglich der Verteilung der Lernumgebung P, ist die Wahrscheinlichkeit, dass H eine Instanz $\mathbf{x}$ die zufällig gemäß P gezogen wird, falsch klassifiziert.

$$error_P(H) := P[\{(\mathbf{x}, y) \in X \times Y \mid H(\mathbf{x}) \neq y\}]. \tag{2.1}$$

Empirischer Fehler Der sample Fehler $error_S$ einer Hypothese H bezüglich eines Datensatzes S von n Instanzen, ist der Bruchteil der falsch klassifizierten Instanzen.

$$error_S(H) := \frac{1}{|S|} \sum_{\langle \mathbf{x}, y \rangle \in S} \delta(y, H(\mathbf{x})) \qquad \delta(y, H(\mathbf{x})) = \begin{cases} 0 & \text{falls } y = H(\mathbf{x}) \\ 1 & \text{falls } y \neq H(\mathbf{x}) \end{cases} \tag{2.2}$$

Der wahre Fehler $error_P$ einer Hypothese H kann in der Praxis nicht berechnet werden, da man die Verteilung P nicht kennt[5]. Ziel ist es also mittels eines Datensatzes S den empirischen Fehler $error_S$ zu berechnen und damit eine möglichst gute Abschätzung des wahren Fehlers $error_P$ zu erreichen.

Mit statistischen Methoden[6] können Aussagen über den geschätzten Fehler getroffen bzw. auch verschiedene Hypothesen miteinander verglichen werden. Bei einem Datensatz S der $n > 30$ Instanzen enthält, die zufällig aus der Lernumgebung P gezogen wurden, liegt der der wahre Fehler $error_P$ approximativ normalverteilt, mit einer Wahrscheinlichkeit von $N\%$, innerhalb des Intervalls

$$error_S(H) \pm z_N \sqrt{\frac{error_S(H) \cdot (1 - error_S(H))}{n}} \tag{2.3}$$

Mittels einer anderen Abschätzung (*Chernoff-Bound*) erhält man einen exponentiell schnellen Abfall der Wahrscheinlichkeit, dass sich der wahre und empirische Fehler

[4]Im Fall eines Regressionsproblems wird der wahre $MSE_P(H) := P[(y - H(\mathbf{x}))^2]$ und der geschätzte $MSE_S(H) := \frac{1}{n} \sum_{i=1}^{n} (y_i - H(\mathbf{x_i}))^2$ mittlere quadratische Fehler (mean squared error) der Hypothese H definiert. Diese Fehlerdefinition wird auch für Methoden aus dem unüberwachten Lernen verwendet.

[5]Würde man P kennen, könnte man eine perfekte Hypothese erstellen.

[6]Bei dem Vergleich von y und $H(\mathbf{x})$ in Gleichung 2.1 handelt es sich um ein Bernoulli-Experiment, also ist der Fehler binomialverteilt.

nur mehr um ein $\epsilon > 0$ unterscheiden, bei wachsender Anzahl n der Beispiele in dem Sample-Datensatz.

$$\text{Prob}(|error_P(H) - error_S(H)| \geq \epsilon) \leq 2 \cdot \exp(-2 \cdot n \cdot \epsilon^2) \tag{2.4}$$

Damit ergibt sich das typische Problem mit dem man beim maschinellen Lernen leben muss: Es gibt zu wenig Trainingsbeispiele, um eine gute Performance garantieren zu können.

Bewertung von Lernalgorithmen Wie das *no free lunch theorem* [16] zeigt, kann kein Lernalgorithmus einem anderen vorgezogen werden wenn es darum geht den wahren Fehler (Generalisierungsfehler) zu minimieren. Es ist daher wichtig die Fehler der Hypothesen[7] bei einem konkreten Lernproblem zu untersuchen.

Der Fehler einer Hypothese H hängt sowohl von dem Datensatz S als auch von dem verwendeten Lernalgorithmus A und damit natürlich von der Hypothesenklasse $\mathcal{H}$ ab. Der erwartete (mittlere) Wert des wahren Fehlers von A über alle möglichen Trainingsdatensätze S die zufällig aus der Lernumgebung P gezogen werden ist informal

$$\text{Prob}_{S \sim P}[error_P(A(S))] = \text{bias}^2 + \text{variance} \tag{2.5}$$

Mittels dieser zwei Kenngrößen wird die Übereinstimmung des Lernalgorithmus mit den Daten dargestellt. Der *bias* entspricht dem durchschnittlichen Fehler und die *variance* wie stark der Fehler von dem verwendeten Datensatz abhängt. Beide Parameter sind jedoch voneinander abhängig und dies drückt sich im *bias-variance-dilemma* aus. Lernalgorithmen mit mehreren freien Parametern (komplexere Hypothesenklassen) haben im Allgemeinen einen kleinen Bias, aber dafür eine hohe Varianz. Salop gesagt: Besitzt ein Lernalgorithmus einen hohen Bias und eine kleine Varianz bedeutet das, dass er sehr falsche Ergebnisse liefert, sich dessen aber sicher ist.

Die Definition von Bias und Varianz für den Fall der Klassifikation ist leider nicht so einfach wie im Fall der Regression, also bei der Approximation einer Funktion [39], [48], [8], [16].

Resampling Um die Schätzungen des Fehlers (Bias und Varianz) zu verbessern, können statistische Methoden eingesetzt werden, die im Wesentlichen aus einem gegebenen Datensatz wiederholt unterschiedliche Datensätze generieren (*„resampling“*) und auf diesen den Lernalgorithmus anwenden. Einige Techniken sind hierbei *jackknife (leave-one-out)*, *m-fold-cross-validation* und *bootstrapping* [16].

[7] Es sei an dieser Stelle angemerkt, dass selbst wenn die Hypothesenklasse $\mathcal{H}$ beliebig groß ist, nicht unbedingt ein wahrere Fehler von Null ($\inf_{H \in \mathcal{H}} error_P(H) = 0$) erreicht werden muss. Dies liegt daran, dass die Lernumgebung, definiert durch die Wahrscheinlichkeitsverteilung P, widersprüchliche Beispiele zulassen kann. Es existiert also mindestens ein $\mathbf{x} \in X$, sodass es zwei $y_1, y_2 \in Y$ mit $y_1 \neq y_2$ gibt mit $P(\langle \mathbf{x}, y_1 \rangle) > 0$ und $P(\langle \mathbf{x}, y_2 \rangle) > 0$.

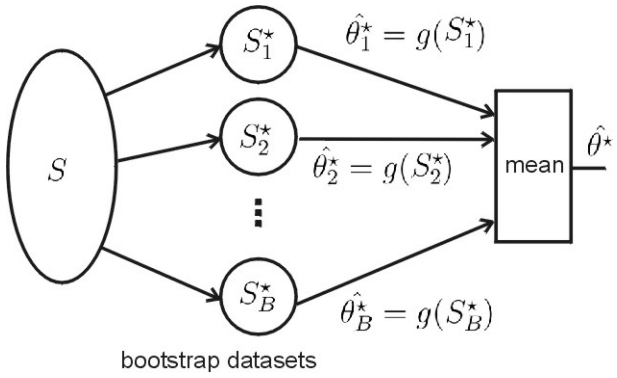

Abbildung 2.3: Bootstrapping

Die Idee des Bootstrapping[8] ist einen neuen Datensatz $S^\star$, dessen Beispiele zufällig (mit zurücklegen) aus dem gegebenen Datensatz S gezogen werden, zu erzeugen. Dieser Vorgang wird B-fach wiederholt und es ergeben sich somit Datensätze $S_1^\star, ...S_B^\star$ die als unabhängig betrachtet werden. Die Schätzung $\hat{\theta^\star}$ der Statistik θ ist nun lediglich der Mittelwert der einzelnen berechneten (bootstrap) Schätzwerte $\hat{\theta}_b^\star = g(S_b^\star)$.

$$\hat{\theta^\star} = \frac{1}{B}\sum_{b=1}^{B}\hat{\theta}_b^\star \tag{2.6}$$

Des Weiteren kann dieses Prinzip angewendet werden um die Effizients eines Lernalgorithmus zu steigern, wie in Folge noch unter dem Punkt *Meta Learning* und speziell in dem Abschnitt 2.3 *Boosting* ausgeführt wird.

Meta Learning Für zwei Hypothesen $H_1, H_2 \in \mathcal{H}$ die durch einen Algorithmus A über zwei Datensätze S_1, S_2 generiert werden gilt allgemein $H_1 = A(S_1) \neq H_2 = A(S_2)$ für $S_1 \neq S_2$. Das heißt $error_P(H_1) \neq error_P(H_2)$. Als Frage stellt sich nun, welche Hypothese bzw. welche Trainingsmenge soll verwendet werden. Die Idee ist, dass man durch eine geeignete Kombination der einzelnen Hypothesen H_i zu einer Master-Hypothese H kommt, die bessere Performance Werte liefert. Die H_i können durch Anwendung des selben Algorithmus $A : X \times Y \rightarrow \mathcal{H}$ auf verschiedenen S_i (*bagging*, *boosting*) entstehen oder es können auch verschiedene Lernalgorithmen $A_j : X \times Y \rightarrow \mathcal{H}_j$ verwendet werden (*stacking*).

Overfitting Die grundlegende Annahme des maschinellen Lernens ist, dass es einen systematischen Zusammenhang zwischen Input und Output gibt, die jedoch mit Störungen überlagert sind. Dieser grundlegende Zusammenhang soll mittels eines Lernalgorithmus erfasst werden. Die Güte einer Hypothese $H \in \mathcal{H}$ wird im Wesentlichen durch den ermittelten sample Fehler (Gleichung 2.2) bestimmt. Es kann vorkommen, dass

[8]Die Namensgebung geht auf die Lügengeschichten des Barons Münchhausen von Rudolf Erich Raspe zurück.

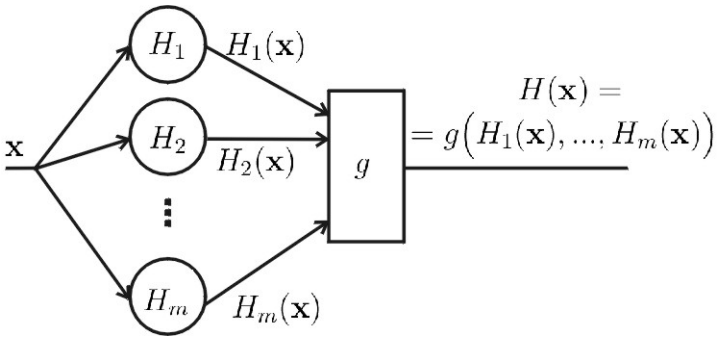

Abbildung 2.4: Meta Learning

eine in diesem Sinn optimale Hypothese $H^\star$ einen größeren wahren Fehler (Gleichung 2.1) besitzt, also eine zweite Hypothese $H \neq H^\star$. Die Minimierung des sample Fehlers führt in diesem Fall also zu einer suboptimalen Hypothese. Das Ziel ist es natürlich den wahren Fehler $error_P(H)$ und nicht den sample Fehler $error_S(H)$ zu minimieren. Auf neuen Daten, von denen anzunehmen ist, dass sie nach dem selben Prinzip wie die Trainingsdaten generiert werden, ist der Fehler aber sehr hoch. Dieser Effekt wird *overfitting* genannt.

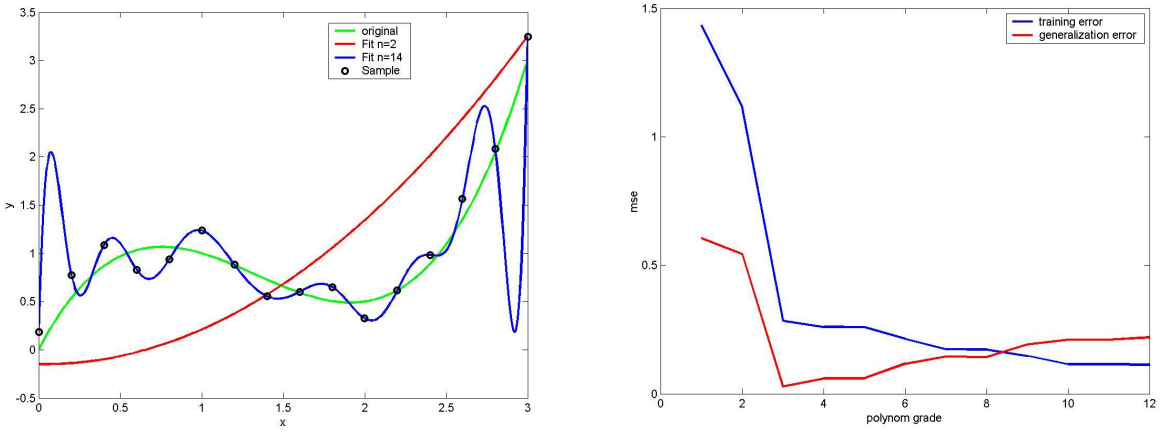

Abbildung 2.5: *Overfitting*-Beispiel: (links) gegebenes Regressionsproblem; (rechts) Verlauf des Trainings- und Generalisierungsfehlers

Ein Beispiel soll dies anhand des in Abbildung 2.5 gezeigten Regressionsproblems verdeutlichen. Die originale Funktion (grün) stellt den wahren datengenerierenden Prozess dar und ist ein Polygon dritten Grades. Davon wurden 16 Beispiele (schwarze Kreise) generiert, die mit additiven Rauschen überlagert sind. Diese Beispiele wurden herangezogen um die Funktion zu lernen. In einem Fall wurde dies durch Hypothesen aus der Hypothesenklasse, die alle Polygone zweiten Grades beinhalten versucht. Man erkennt schön (rot), dass diese Hypothesenklasse nicht mächtig genug ist um die gegebenen Daten zu erklären. Im zweiten Fall wurde die Hypothesenklasse erweitert

und umfasst nun alle Polygone bis zum Grad 14. Die gegebenen Beispiele werden nun nahezu perfekt durch die Kurve (blau) approximiert (entspricht einem sehr kleinen Trainingsfehler), jedoch ist der wahre Fehler (Generalisierungsfehler) auf allen Daten sehr groß. Auf der rechten Seite der Abbildung ist der Verlauf des Trainings- und Generalisierungsfehlers über die Mächtigkeit der Hypothesenklasse (Grad der Polynome) aufgetragen. Zu Beginn sinken sowohl Test- und Generalisierungsfehler, doch gibt es einen Punkt bei dem der Generalisierungsfehler wieder zu steigen beginnt, obwohl der Trainingsfehler immer weiter sinkt. Ab diesem Punkt tritt overfitting ein und der Lernalgorithmus lernt die gegebenen Beispiele zu „auswendig". Das heißt es wird nicht mehr der dahinterliegende Prozess sondern das überlagerte Rauschen gelernt, da die Hypothesenklassen zu mächtig sind.

Gleiche Erkenntnisse können auch im Fall von Klassifikation gezeigt werden. Die Entscheidungsgrenzen werden entsprechend „komplizierter" und unnatürlich.

Dies führt zu folgenden Fragen: Wie groß (wie mächtig) soll die Hypothesenklasse $\mathcal{H}$ für einen konkreten Datensatz S sein? Für diese gibt es aber leider kein Patentrezept sondern nur Vorschläge.

Occam's Razor „Im Zweifel bevorzuge die einfache Hypothese zur Erklärung der Daten." Gilt also $error_S(H) = error_S(H')$ für zweier Hypothesen H und H', dann gilt (wahrscheinlich) auch $error_P(H) < error_P(H')$ falls H einfacher ist als H'[9]. Ein Erklärungsversuch kann über das *minimal description length principle* gemacht werden. Es liefert dabei einen Ansatz für das Gleichgewicht der Mächtigkeit des Modells einerseits und dem erreichten Fehler andererseits [16].

Albert Einstein Regel In den Naturwissenschaften versucht man schon seit Jahrhunderten, beim Bestreben qualitative Naturgesetze zu finden, die Ergebnisse von neuen Beobachtungen möglichst gut vorherzusagen. „Ein Naturgesetz[10] soll einfach sein, aber nicht zu einfach."

Trainings-, Validierungs-, Testdatensatz Für die praktische Implementierung eines Lernverfahrens werden die gegebenen Beispiele meistens in drei Datensätze unterteilt. Einen Trainingsdatensatz für die Bestimmung der besten Hypothese in der gewählten Hypothesenklasse, ein Validierungsdatensatz für die Bestimmung der besten Hypothesenklasse (Modellstruktur) und einen, von den beiden ersten komplett unabhängigen Testdatensatz zur tatsächlichen Bewertung des zu erwarteten Generalisierungsfehlers.

ROC, RPC Um die Genauigkeit eines Detektionssystems zu bestimmen sind besonders zwei Kenngrößen ausschlaggebend. Einerseits die Anzahl der korrekt klassifizierten Beispiele (welche maximal werden soll) und andererseits die Anzahl der falschen Detektionen (welche minimal werden soll). Zwei Methoden diesen Kompromiss zu veranschaulichen sind hier aufgeführt.

[9] Typischerweise hat man eine Hierarchie von Hypothesenklassen $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \mathcal{H}_3$...

[10] entspricht in diesem Zusammenhang einer Hypothese

ROC (Receive Operator Characteristic) Die ROC-Kurve stellt ein Diagramm dar, in dem die (positive) Detektionsrate gegenüber der false-positive Rate aufgetragen wird. Die Berechnung ist in den Gleichungen 2.7 und 2.8 informell gegeben.

$$\text{detection rate} = \frac{\#\text{correct positives}}{\#\text{all positives in the dataset}} \tag{2.7}$$

$$\text{falsepositives rate} = \frac{\#\text{false positives}}{\#\text{all negatives in the dataset}} \tag{2.8}$$

Zwei Probleme treten bei diesem Ansatz auf. Erstens messen diese Werte die Genauigkeit des Systems in Bezug auf einen Klassifikator und nicht als einen Detektor. Die Anzahl der negativen Beispiele ist typischerweise sehr groß im Vergleich zu den positiven. Eine große absolute Anzahl von falschen Detektionen drückt sich nicht notwendigerweise in einer hohen false-positive Rate aus, wenn sehr viele negative Beispiele vorhanden sind.

RPC (Recall Precision Curve) Wenn ein Objektdetektionssystem in der Praxis eingesetzt wird, ist man daran interessiert zu wissen, wie viele Objekte detektiert werden und wie oft diese Detektionen falsch sind. Dieser Zusammenhang wird präziser mittels RPC dargestellt, zusammengefasst in den Gleichungen 2.9 bis 2.11. Der *recall* entspricht dabei der Detektionsrate und stellt prozentual dar wie viele Objekte richtig detektiert werden. Wie zuverlässig diese Detektionen sind wird durch den Wert der *precision* ausgedrückt.

$$\text{recall} = \frac{\#\text{correct positives}}{\#\text{all positives in the dataset}} \tag{2.9}$$

$$\text{precision} = \frac{\#\text{correct positives}}{\#\text{all detections}} \tag{2.10}$$

$$1 - \text{precision} = \frac{\#\text{false positives}}{\#\text{all detections}} \tag{2.11}$$

Ein Diagramm das den *recall* gegenüber *1-precision* darstellt, veranschaulicht die oben beschriebenen, gewünschten Eigenschaften besser als die ROC-Kurve.

Um einen Kompromiss zwischen den beiden Messgrößen darzustellen wurde in Agarwal et al. [1] die Kenngröß *F-measure* eingefügt. Sowohl *recall* als auch *precision* werden dabei als gleichwichtig betrachtet.

$$\text{F} - \text{measure} = \frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}} \tag{2.12}$$

2.3 Boosting

Boosting hat seinen Ursprung in der PAC (**P**robably **A**pproximately **C**orrect) Lerntheorie [51]. In Kearns und Valiant [26] wurde Boosting erstmals erwähnt und basiert auf der Idee verschiedene schwache PAC-Lerner zu einem starken zu „boosten“. Man versucht die Ergebnisse von einem schwachen PAC-Lerner, der vielleicht nur etwas bessere Ergebnisse erzielt als bloßes Raten, zu verbessern, indem man ihn wiederholt auf veränderte Versionen der Daten anwendet. Es handelt sich dabei also um ein Verfahren aus dem Meta-Learning, mit dem Ziel die Genauigkeit eines Lernalgorithmus unter Zuhilfenahme von *resampling* Methoden (siehe hierzu den vorigen Abschnitt) zu verbessern.

Welche Basishypothesenklasse (schwache PAC-Lerner) verwendet wird, spielt zunächst eine untergeordnete Rolle. Die verschiedenen Meta-Lernmethoden unterscheiden sich im Wesentlichen in der Art wie die Gewichte w_i der Wahrscheinlichkeitsverteilung W gewählt werden, mit dessen Hilfe der *resampling*-Prozess gesteuert wird. Anhand der so genierten Datensätze bestimmt der Lernalgorithmus eine Hypothese $hWeak$ aus der Basishypothesenklasse. Beim *Bagging* (von **b**ootstrap **agg**regat**ing**) [7] sind die einzelnen Gewichte lediglich gleichverteilt $w_i = \frac{1}{n}$. Bei Boosting und Arcing Methoden wird eine kompliziertere aber effizientere Gewichtsverteilung in jedem Zeitschritt erstellt. Um zu der finalen Hypothese zu gelangen, werden diese einzelnen schwachen Hypothesen $hWeak_t$ *konvex* mittels der Faktoren α_t kombiniert, wie in Gleichung 2.13 dargestellt. Die Varianz der Ausgabe wird durch die Kombination vieler Hypothesen verringert und man erhält einen besseren Generalisierungsfehler [8]. Weiters wird eine gewisse Unabhängigkeit von den Eigenschaften der Basislernalgorithmen geschaffen.

$$\mathcal{C} := \left\{ hStrong : \mathbf{x} \rightarrow \sum_{t=1}^{T} \alpha_t \cdot hWeak_t(\mathbf{x}) \middle| \alpha_t \geq 0,\ \sum_{t=1}^{T} \alpha_t = 1,\ hWeak_t \in \mathcal{H} \right\} \tag{2.13}$$

Es gibt eine Vielzahl verschiedener Boostingansätze. Der populärste ist dabei *AdaBoost* (von **ada**ptive **boost**ing) vorgestellt von Freund und Schapire in [23], der viele Probleme der vorangegangenen Ansätze löste. Relativ kurz nach dem Erscheinen des AdaBoost Algorithmus stellte Breiman [8], [9] eine Klasse von Algorithmen, die er unter *arcing algorithms* (von **a**daptive **r**eweighting and **c**ombin**ing** algorithms) zusammenfasste, vor. Interessanterweise ist AdaBoost eine spezielle Form eines Arcing-Algorithmus [10].

In dieser Arbeit wird nur die Klassifikation mit zwei Klassen behandelt. Die Analyse wird dadurch wesentlich vereinfacht und für die spätere Implementierung des Objektdetektionssystems in Kapitel 3 wird nur eine Klassifikation in zwei Klassen benötigt. Eine Erweiterung des Boostingansatzes auf mehrere Klassen und sogar auf Regressionsprobleme ist [23] zu entnehmen. Die wesentlichen Eigenschaften bleiben jedoch erhalten.

2.3.1 AdaBoost

AdaBoost[11] erlaubt das kontinuierliche hinzufügen von schwachen Klassifikatoren (schwachen Hypothesen, Basishypothesen) bis ein gewünschter kleiner Trainingsfehler erreicht wird. Bei einem binären Problem (zwei Klassen Fall) sind dazu zunächst schwache Klassifikatoren (*weak classifier*) mit einer Genauigkeit größer als 50% („Daumenregeln“) auf den Trainingsdaten notwendig. Die T schwachen Klassifikatoren werden dabei auf einer gezielt veränderten Trainingsmenge trainiert und anschließend in geeigneter Weise zu einem starken Klassifikator (*strong classifier*) kombiniert.

$$hStrong(x) = \text{sign}\Big(\sum_{t=1}^{T} \alpha_t \cdot hWeak_t(x)\Big) \tag{2.14}$$

Ziel des AdaBoost Algorithmus ist es also T schwache Klassifikatoren $hWeak_t(\mathbf{x})$ und die dazugehörigen Faktoren α_t zu finden um dann mittels einer Linearkombination daraus einen starken Klassifikator zu gewinnen.

In Algorithmus 2.1 ist dieser Vorgang dargestellt. Jedem Trainingsbeispiel $\langle \mathbf{x_i}, y_i \rangle$ wird ein Gewicht w_i zugewiesen. All diese Gewichte stellen eine Wahrscheinlichkeitsverteilung W dar. Mit Hilfe dieser Wahrscheinlichkeitsverteilung wird ein schwacher Klassifikator $hWeak(\mathbf{x})$ erstellt. Es gibt grundlegend zwei Möglichkeiten, wie die Gewichte w_i beim Lernen der schwachen Klassifikatoren berücksichtigt werden können. Einerseits kann aus dem gesamten Trainingsdatensatz eine Menge gemäß der Verteilung W ausgewählt werden, die danach zum Trainieren verwendet wird. Andererseits können die Gewichte w_i direkt in dem Lernalgorithmus benützt werden um die Fehlerfunktion zu verändern[12].

Nach dem Erstellen eines schwachen Klassifikators wird der Trainingsfehler e_t in Betracht auf die aktuelle Wahrscheinlichkeitsverteilung W_t bestimmt. In der Praxis findet man oft keinen Klassifikator der e_t minimiert, sondern nur eine Näherung. Da jedoch nur $e_t < \frac{1}{2}$ gefordert wird, hat das keine negativen Auswirkungen. Außerdem kann es vorkommen, dass es keinen schwachen Klassifikator gibt für die das erreicht wird. In diesem Fall muss der Algorithmus gestoppt werden.

Der gesuchte Faktor α_t für den aktuell erstellten schwachen Klassifikator $hWeak_t$ wird aus dem berechneten Fehler e_t ermittelt (siehe hierzu in der Folge auch Abschnitt 2.3.3). Der Faktor $\alpha_t > 0$ ist umso größer, je kleiner $e_t < \frac{1}{2}$ ist.

Als letzter Schritt in einer Boosting Iteration t erfolgt die Anpassung der Wahrscheinlichkeitsverteilung W, die ja in der nächsten Iteration benutzt wird um einen neuen schwachen Klassifikator zu erstellen. Das Gewicht $w_{t,i}$ wird erhöht, wenn die letzte schwache Hypothese $hWeak_t(\mathbf{x})$ das Beispiel i nicht richtig klassifiziert hat. Auf der anderen Seite wird die Wahrscheinlichkeit der Beispiele verringert, die richtig klassifiziert werden konnten. Wie stark das Gewicht verändert wird, hängt von dem aktuellen Fehler e_t ab. Es ergibt

[11] Die nachfolgenden Betrachtungen folgen größten Teils den Arbeiten von Freund und Schapire [21].

[12] Fast alle Fehlerfunktionen haben die Form $\sum_i g((x_i), y_i)$. Man kann die Gewichte w_i direkt benutzen um eine gewichtete Fehlerfunktion zu erhalten $\sum_i w_i \cdot g((x_i), y_i)$

sich eine Multiplikation mit dem Faktor $(\frac{1-e_t}{e_t})^{\pm\frac{1}{2}}$, welche auch mittels α_t ausgedrückt werden kann[13]. Zusammenfassend misst also das Gewicht $w_{t,i}$ die Wichtigkeit des Beispiels in Bezug auf die aktuelle Hypothese $hWeak_t$. AdaBoost fokussiert sich auf die schwierig zu erlernenden Beispiele - je höher ein Gewicht ist, desto wichtiger ist das Beispiel beim Training der nächsten (schwachen) Hypothese[14].

Algorithmus 2.1 AdaBoost

Require: Trainingsdatensatz $S = \langle\langle \mathbf{x_1}, y_1\rangle, ..., \langle \mathbf{x_n}, y_n\rangle\rangle, x_i \in X, y_i \in Y = \{-1, +1\}$

- initialisiere $w_{i,1} = \frac{1}{n}, i = 1, ..., n$

for $t = 1, 2, .., T$ **do**

- normalisieren der Gewichte, sodass W_t eine Wahrscheinlichkeitsverteilung ist

$$w_{t,i} = \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$

- trainiere schwachen Klassifikator in Betracht der Verteilung W_t und erstelle schwache Hypothese

$$hWeak_t : X \to \{-1, +1\}$$

- bestimme Trainingsfehler in Betracht der Verteilung W_t

$$e_t = \text{Prob}_{i\sim W_t}[hWeak_t(\mathbf{x_i}) \neq y_i] = \sum_{i:hWeak_t(\mathbf{x_i})\neq y_i} w_{t,i}$$

- Abbruch wenn $e_t = 0$ oder $e_t > \frac{1}{2}$
- Faktor berechnen

$$\alpha_t = \frac{1}{2} \cdot \ln\left(\frac{1-e_t}{e_t}\right)$$

- Aktualisierung der Gewichte vornehmen

$$w_{t+1,i} = w_{t,i} \cdot \begin{cases} \exp(-\alpha_t) & \text{if } hWeak(\mathbf{x_i}) = y_i \\ \exp(\alpha_t) & \text{if } hWeak(\mathbf{x_i}) \neq y_i \end{cases} = w_{t,i} \cdot \exp(-\alpha_t \cdot y_i \cdot hWeak_t(\mathbf{x_i}))$$

end for

- trainierter starker Klassifikator

$$hStrong(x) = \text{sign}\Big(\sum_{t=1}^{T} \alpha_t \cdot hWeak_t(x)\Big)$$

Durch dieses Vorgehen erhöht AdaBoost die Komplexität der Hypothesenklasse. In [23] wurde gezeigt, dass die VC-Dimension der finalen Hypothese nach T Boosting Iterationen auf

$$2 \cdot (d+1) \cdot (T+1) \cdot \log_2(\exp(T+1)) \tag{2.15}$$

ansteigt, wenn $d \geq 2$ die VC-Dimension der Basishypothesenklasse der schwachen Lernalgorithmen ist. Es können somit „schwierigere“ Aufgaben gelöst werden.

[13] In der Literatur gibt es oft unterschiedliche Vorgehensweisen für die Initialisierung der Gewichte sowie deren Anpassung. Die Unterschiede sind jedoch sehr gering und die jeweiligen Vorteile sind eher praxisorientiert und für eine konkrete Implementierung relevant.

[14] Wie zu sehen, hängt die Wahrscheinlichkeitsverteilung W von der jeweils vorherigen Hypothese ab und das bedeutet, dass die Basishypothesen sequentiell trainiert werden müssen. Damit kann nicht garantiert werden, dass dabei Redundanzen auftreten. Der Algorithmus *FloatBoost* versucht zum Beispiel „ unvorteilhafte“ schwache Hypothesen wieder zu entfernen. Hierbei ergeben sich jedoch Konflikte mit dem allgemeinen Boostingansatz und es kann zu Instabilitäten kommen [57].

In den nachfolgenden Abschnitten wird der hier vorgestellte Algorithmus auf seine Performance untersucht. Wie gezeigt wird sind sowohl der Trainingsfehler als auch der Generalisierungsfehler nach oben hin beschränkt. Diese beiden Tatsachen zusammen ergeben, dass es sich um eine effiziente Umwandlung von einem schwachen PAC-Lernalgorithmus zu einem starken handelt.

2.3.2 Analyse des Trainingsfehlers

Es kann gezeigt werden, dass der Trainingsfehler von $hStrong$ exponentiell mit der Anzahl der schwachen Klassifikatoren sinkt. Der Trainingsfehler von $hWeak_t$ kann als $e_t = \frac{1}{2} - \gamma_t$ geschrieben werden. Eine Hypothese die zufällig bei jedem Beispiel „ratet“ hat einen Fehler von $\frac{1}{2}$ (50%) bei einem binären Problem. Somit misst γ_t also wieviel besser als zufällig die Entscheidung von $hWeak_t$ ist. Freund und Schapire haben in [23] bewiesen, dass der Trainingsfehler der finalen Hypothese $hStrong$ wie in Gleichung 2.16 nach oben beschränkt ist.

$$\text{Prob}_{(\mathbf{x},y)\sim S}[hStrong(\mathbf{x}) \neq y] \leq \prod_{t=1}^{T} \sqrt{4 \cdot e_t \cdot (1 - e_t)} \leq \exp\Big(-2 \sum_{t=1}^{T} \gamma_t^2 \Big), \gamma_t = \frac{1}{2} - e_t \tag{2.16}$$

Wenn also jede schwache Hypothese nur ein bisschen besser ist als zufällig $\gamma_t \geq \gamma$ für ein $\gamma > 0$, dann sinkt der Trainingsfehler exponentiell schnell.

Eine ähnliche Grenze konnte für vorangegange Boosting-Algorithmen gezeigt werden, jedoch mit der wesentlichen Einschränkung, dass eine untere Grenze für γ zu Begin bereits bekannt sein musste. Es ist jedoch praktisch schwer eine solche Grenze anzugeben. AdaBoost hingegen adaptiert die Fehlerraten der verschiedenen schwachen Klassifikatoren.

2.3.3 Analyse des Generalisierungsfehlers

Es wurde von Freund und Schapire [23] gezeigt, dass die Grenze des Generalisierungsfehlers mit hoher Wahrscheinlichkeit[15] dem in Gleichung 2.17 entspricht. Dabei ist n die Anzahl der Beispiele (Datenpunkte) eines Datensatzes S, d die VC-dimension des Hypothesenraums der schwachen Hypothesen und T die Anzahl der Boosting Iterationen.

$$\text{Prob}_{(\mathbf{x},y)\sim D}[hStrong(\mathbf{x}) \neq y] \leq \text{Prob}_{(\mathbf{x},y)\sim S}[hStrong(\mathbf{x}) \neq y] + O\Big(\sqrt{\frac{T \cdot d}{n}}\Big) \tag{2.17}$$

Diese Grenze lässt darauf schließen, dass das Boosting overfitting zeigen wird, wenn es zu lange betrieben wird, das heißt wenn T zu groß wird. Jedoch haben Experimente

[15] Der Grund für den Ausdruck „hoher Wahrscheinlichkeit“ ist in der PAC-Lerntheorie begründet, da hier (wie der Name schon sagt) nur Aussagen mit Wahrscheinlichkeiten gemacht werden. Für die genauen Grenzen sei auf die angegebenen Artikel verwiesen.

(zum Beispiel von Breiman [9]) ergeben, dass Boosting oft kein overfitting zeigt, auch wenn es tausende von Runden gelaufen ist. Es wurden sogar entgegengesetzte Ergebnisse beobachtet. AdaBoost minimiert manchmal den Fehler auf Testdaten weiter, auch wenn der Fehler auf den Trainingsdaten schon lange Null erreicht hat (perfekte Klassifikation auf den Trainingsdaten). Diese Beobachtungen scheinen der Grenze in Gleichung 2.17 zu widersprechen. Die Antwort auf diese empirischen Erkenntnisse lieferte Schapire et al. [42] mit der Margin-Theorie. Der Abstand (Margin) eines Beispiels $\langle \mathbf{x}, y \rangle$ wird dabei wie in Gleichung 2.18 definiert.

$$\text{margin}(\mathbf{x}, y) = \frac{y \cdot \sum_{t=1}^{T} \alpha_t \cdot hWeak_t(\mathbf{x})}{\sum_{t=1}^{T} \alpha_t} \tag{2.18}$$

Es ist leicht zu erkennen, dass diese Zahl im Intervall von $[-1, +1]$ liegt und genau dann positiv ist, wenn das Beispiel korrekt klassifiziert wird. Der Wert des Margins lässt auf die Sicherheit schließen mit dem das Beispiel klassifiziert wurde. Das heißt je größer er ist desto sicherer ist sich der Algorithmus.

In der oben angegebenen Arbeit konnte gezeigt werden, dass große Margin Werte bei den Trainingsdaten in eine obere Grenze des Generalisierungsfehlers umgesetzt werden können. Für ein beliebiges $\theta > 0$ kann der Generalisierungsfehler mit hoher Wahrscheinlichkeit nach oben laut Gleichung 2.19 abgeschätzt werden. Diese Grenze ist nun unabhängig von der Anzahl der Boosting Iterationen T. Boosting versucht also den Abstand zwischen den positiven und negativen Beispielen der Trainingsdaten so groß wie möglich zu machen.

$$\text{Prob}_{(\mathbf{x},y)\sim D}[hStrong(\mathbf{x}) \neq y] \leq \text{Prob}_{(\mathbf{x},y)\sim S}[\text{margin}(\mathbf{x}, y) \leq \theta] + O\Big(\sqrt{\frac{d}{n \cdot \theta^2}}\Big) \tag{2.19}$$

Mit Hilfe der Definition des Margins können nun auch weitere Aussagen über den Algorithmus getroffen werden. Es kann, in gewisser Verallgemeinerung zu Gleichung 2.16, gezeigt werden, dass Boosting den Margin auf den Trainingsdaten vergrößert und somit für alle $\theta \geq 0$ gilt

$$\text{Prob}_{(\mathbf{x},y)\sim S}[\text{margin}(\mathbf{x}, y) \geq \theta] \leq \prod_{t=1}^{T} \sqrt{4 \cdot e_t^{1-\theta} \cdot (1 - e_t)^{1+\theta}} \tag{2.20}$$

Diese Maximierung des Margins kann so erklärt werden, dass AdaBoost versucht eine Fehlerfunktion $G(\boldsymbol{\alpha})$ zu minimieren, welche offensichtlich den Margin maximiert.

$$G(\boldsymbol{\alpha}) = \sum_i \exp\Big(- \frac{\sum_t \alpha_t}{2} \cdot \text{margin}(\mathbf{x_i}, y_i)\Big) \tag{2.21}$$

In jeder Boosting Iteration muss für eine schwache Hypothese $hWeak_t$ ein dazugehöriges Gewicht α_t bestimmt werden, sodass G minimiert wird. Man kann dieses optimale Gewicht

mittels line-search finden oder im Fall von AdaBoost auch direkt berechnen. Für die direkte Berechnung ergibt sich eben die bekannt Formel $\alpha_t = \ln(\frac{e_t}{1-e_t})$ [10].

Die Berechnung der Gewichte in einer Boosting Iteration t kann auch direkt angegeben werden und entspricht dabei der Update-Regel des AdaBoost Algorithmus [40] und ist in Gleichung 2.22 dargestellt.

$$w_{t+1,i} = \frac{\exp(-\frac{\sum_t \alpha_t}{2} \cdot \text{margin}(\mathbf{x_i}, y_i))}{\sum_j \exp(-\frac{\sum_t \alpha_t}{2} \cdot \text{margin}(\mathbf{x_j}, y_j))} = \tag{2.22}$$

$$= \frac{\frac{\partial G(\boldsymbol{\alpha})}{\partial \text{margin}(\mathbf{x_i}, y_i)}}{\sum_j \frac{\partial G(\boldsymbol{\alpha})}{\partial \text{margin}(\mathbf{x_i}, y_i)}} = \tag{2.23}$$

$$= \frac{\exp(-\frac{1}{2} \cdot \text{margin}(\mathbf{x_i}, y_i))^{\sum_t \alpha_t}}{\sum_j \exp(-\frac{1}{2} \cdot \text{margin}(\mathbf{x_i}, y_i))^{\sum_t \alpha_t}} \tag{2.24}$$

Weiters ist festzuhalten, dass mit dieser Regel sich ein Update bezüglich des Gradienten der Funktion G bezogen auf den Margin ergibt (siehe Gleichung 2.23). Mit dieser Wahrscheinlichkeitserteilung der Gewichte wird also eine neue Hypothese generiert, die approximativ der besten Hypothese entspricht, wenn G direkt minimiert werden würde. AdaBoost ist also eine approximativ *gradient descent* Methode welche G asymptotisch minimiert [40] und somit den Margin maximiert.

Wie festgestellt arbeitet AdaBoost mit einer Fehlerfunktion $G(\boldsymbol{\alpha})$. Dabei spielen die Faktoren α_t eine wichtige Rolle. Wird die Bestimmung der Gewichte in der t-ten Iteration untersucht, so ergibt sich diese Abhängigkeit sehr schön. Wird die Summe der Faktoren schrittweise erhöht, was ja mittels AdaBoost geschieht, so kann man von einem Abkühlungsprozess (*simulated annealing*) sprechen. Dieses heuristische Verfahren wird oft bei nichtlinearen Optimierungsproblemen verwendet um das „Steckenbleiben“ in lokalen Minima zu vermeiden. In Analogie zu physikalischen Prozessen definiert man die Temperatur $\vartheta = \frac{1}{\sum_t \alpha_t}$ die die Sensibilität der sogenannten *softmax* Funktion regelt[16]. Wenn ϑ groß ist befindet sich das System in einem Zustand mit hoher Energie und alle Beispiele haben ein hohes Gewicht. Wird die Temperatur verringert (automatisch indem immer neue Hypothesen mit einem Gewicht $\alpha_t > 0$ hinzugefügt werden), wird den Beispielen mit geringen Margin immer ein größeres Gewicht zugeteilt.

Das Problem des Maximieren des minimalen Margins für konvexe Kombinationen ist ein Problem, das mittels eines linearen Programmes für die L_1 Norm gelöst werden kann. AdaBoost findet approximativ mittels eines Annealing Prozesses die Lösung eines solchen linearen Programmes[17]. In diesem Sinne stellt AdaBoost auch eine spezielle Art für die

[16] Die *softmax* Funktion $z_i(x_i, \vartheta) = \frac{\exp(x_i)^{1/\vartheta}}{\sum_{j=1}^{n} \exp(x_j)^{1/\vartheta}}$ stellt eine kontinuierliche Version der *winner-take-all* Funktion dar, bei der der maximale Wert auf 1 und alle anderen auf 0 gesetzt werden. Die jeweilige Nichtlinearität wird mit dem Parameter ϑ eingestellt. Für $\vartheta \to 0$ konvergiert die Funktion zu der *winner-take-all* Funktion.

[17] Die Lösung ist dabei um eine ϵ von dem Optimum verschieden, wobei die Laufzeit polynomial in $\frac{1}{\epsilon}$ ist.

Lösung eines Matrixspieles (Spieltheorie) durch wiederholtes Spielen dar [22]. Somit kann auch ein Zusammenhang zu online-Spielen erkannt werden.

2.3.4 Verbindungen zu SVM

Die Interpretation in Abschnitt 2.3.3 mit Hilfe des Margins hat einen starken Zusammenhang mit *support-vector machines* (SVM) [11]. SVM erzielen gegenwärtig bei Klassifikationsproblemen in der Regel sehr gute Ergebnisse, erfordern jedoch eine beträchtliche Rechenzeit.

Bei SVM wird versucht die negativen und positiven Beispiele eines Datensatzes durch eine Hyperebene[18] zu trennen. Da sich die Beispiele im Merkmalsraum jedoch mit dieser Hypothesenklasse in der Regel nicht linear trennen lassen, wird eine nichtlineare Projektion $\phi : \mathbb{R}^d \to \mathbb{R}^{d'}$ verwendet, die die Beispiele in einem höherdimensionalen Raum $d' \gg d$ abbildet. Dies erlaubt also das ursprüngliche Klassifikationsproblem im $\mathbb{R}^d$ auf ein Klassifikationsproblem in $\mathbb{R}^{d'}$ zurückzuführen. Wenn d' sehr groß ist (auch ∞ möglich) muss man mit overfitting rechnen. Aus diesem Grund reduziert man die Vielfalt der in Frage kommenden Hypothesen durch eine zusätzliche Forderung: Die trennende Hyperebene soll einen möglichst großen Abstand (Margin) von den beiden Mengen erzielen. Im linear trennbaren Fall gibt es stets unendlich viele trennende Hyperebenen, jedoch gibt es nur genau eine für die der minimale Abstand von positiven und negativen Beispielen einen maximalen Wert erreicht. Man nennt diese Hyperebene, diesen Klassifikator, den *maximal margin classifier* für das definierte Klassifikationsproblem.

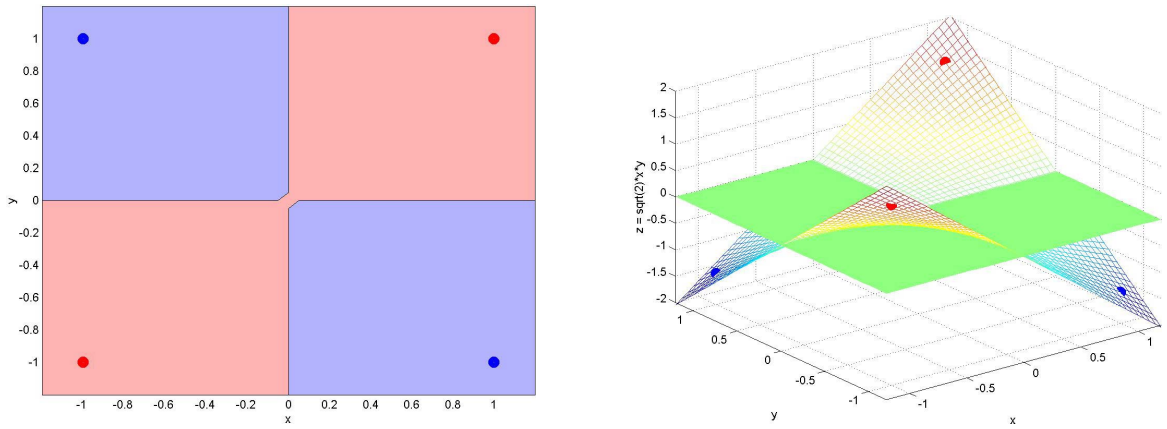

Abbildung 2.6: *XOR*-Beispiel: (links) gegebene Datenpunkte mit resultierenden Entscheidungsgrenzen; (rechts) trennende Hyperebene im höherdimensionalen Raum d'

Diese grundlegende Idee ist an dem Beispiel in Abbildung 2.6 verdeutlicht. Der gegebene Datensatz (XOR-Beispiel) lässt sich, wie leicht zu zeigen ist, nicht linear trennen. Projiziert

[18] Eine trennende Hyperebene im d-dimensionalen Raum ist eine Hypothese der Form $H = \{\mathbf{x} \in \mathbb{R}^d | \mathbf{w} \cdot \mathbf{x} \geq 0\}$ und teilt somit den Raum in zwei Hälften.

man jedoch die einzelnen 2D Beispiele in einem drei dimensionalen Raum, kann leicht eine trennende Hyperebene gefunden werden. Die sich daraus ergebenden Entscheidungsgrenzen sind wieder auf der linken Seite eingezeichnet.

Für den linear separierbaren Fall wird bei SVM also ein Gewichtsvektor $\boldsymbol{\alpha}$ bestimmt, der den minimalen Margin maximiert. Wie in den vorigen Abschnitten gezeigt wurde versucht auch Boosting das selbe Ziel, wenn auch nicht explizit, zu erreichen. Um diesen Zusammenhang etwas klarer zu machen, sei angenommen man hat die einzelnen schwachen Klassifikatoren $hWeak_t$ bereits gefunden und ist daran interessiert die zugehörigen Koeffizienten α_t zu finden um den Generalisierungsfehler (Gleichung 2.19) zu minimieren. Dies läuft also auch auf eine Maximierung des minimalen Margins hinaus. Mit der Definition des Margins aus Gleichung 2.18 und $\mathbf{hWeak}(\mathbf{x}) := \langle hWeak_1(\mathbf{x}), ..., hWeak_T(\mathbf{x})\rangle$ sowie $\boldsymbol{\alpha} := \langle alpha_1, ..., alpha_T\rangle$ erhält man also:

$$\theta = \max_{\boldsymbol{\alpha}} \min_{i} \frac{y_i \cdot \boldsymbol{\alpha} \cdot \mathbf{hWeak}(\mathbf{x_i})}{||\boldsymbol{\alpha}|| \cdot ||\mathbf{hWeak}(\mathbf{x_i})||} \tag{2.25}$$

Boosting verwendet die L_1 Norm[19] für den Gewichtsvektor $||\boldsymbol{\alpha}||_1$ und die maximale Norm L_∞ für den Vektor der Instanzen $||\mathbf{h}(\mathbf{x_i})||_\infty$ der lauf Definition 1 ist, da die Ausgabewerte der binären Klassifikatoren nur $+1, -1$ sind. Bei SVM wird hingegen immer die euklidische Norm (L_2) benutzt und das Ziel ist es explizit dieses Optimierungsproblem zu lösen. SVM und Boosting sind im Wesentlichen also das Gleiche, ausgenommen die Art, wie der Margin gemessen wird und wie die Gewichte angepasst werden [41]. Der Unterschied zwischen den Normen L_1 und L_2 mag bei niedrigdimensionalen Räumen nicht so stark ins Gewicht fallen. Jedoch gerade bei Boosting und SVM sind die Dimensionen sehr groß, sodass der Unterschied der Normen auch bei dem berechneten Margin zu sehr großen Differenzen führt[20]. Die sich ergebenden Klassifikatoren werden sich daher im Allgemeinen stark unterschiedlich.

Weiters ist der benötigte Rechenaufwand unterschiedlich. Die Berechnung der Maximierung des Margins entspricht einem mathematischen Programm. Das heißt man maximiert einen Ausdruck mit gegebenen Einschränkungen (Ungleichungen). Wie in Abschnitt 2.3.3 beschrieben kann AdaBoost als ein Algorithmus verstanden werden, der versucht ein *lineares* Programm zu approximieren. Das Optimierungsproblem bei einer SVM mit einem Trainingsdatensatz $S = \langle\langle \mathbf{x}_1, y_1\rangle, ..., \langle\langle \mathbf{x}_n, y_n\rangle\rangle$ entspricht dabei einem *quadratischen* Programm [16] der From

[19] Die *Minkowski metric* oder auch L_k Norm in einem d-dimensionalen Raum ist definiert als $L_k(\mathbf{a}, \mathbf{b}) := \left(\sum_{i=1}^{d} |a_i - b_i|^k\right)^{1/k}$. Der euklidische Abstand entspricht dabei der L_2 Norm. Die L_1 Norm stellt die *city block* distance dar, wie in der Bildverarbeitung bekannt, also die kürzeste Verbindung der Punkte $\mathbf{a}$ und $\mathbf{b}$ wobei jedes Segment parallel zu einer Koordinatenachse liegt. Die L_∞ Norm stellt die maximale Distanz aller Projektionen von $\mathbf{a}$ und $\mathbf{b}$ auf jede der d Koordinatenachsen dar

[20] Zum Beispiel, die beiden Vektoren $(1, 0)$ und $(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$. In der euklidischen Norm ergibt sich offensichtlich $||(1, 0)||_2 = ||(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})||_2 = 1$. Wohingegen in Bezug auf die L_1 Norm $||(1, 0)||_1 = 1 < ||(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})||_1 = \sqrt{2}$.

$$max \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) \tag{2.26}$$

$$s.t. \quad 0 \leq \alpha_t \leq C, \; i = 1, ..n \tag{2.27}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{2.28}$$

Der sich daraus ergebende Klassifikator ist[21]

$$hSVM(\mathbf{x}) = \text{sign}(\sum_{i=1}^{n} \alpha_i y_i \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) + b) = \text{sign}(\sum_{i:\alpha_i > 0} \alpha_i y_i \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) + b) \tag{2.29}$$

Die optimale Lösung dieses Optimierungsproblems bestimmt durch die α_i ergibt somit die maximal Margin trennende Hyperebene. Nur diejenigen Beispiele die am nächsten zur Hyperebene liegen (minimaler Margin) haben ein $\alpha_i > 0$ und tragen somit zur Entscheidung bei. Sie werden *support vectors* genannt. Auch beim Boosting gibt es ein Equivalent dazu. Die „ wichtigsten“ Beispiele (alle Beispiele mit dem minimalen Margin) werden hierbei laut [39], [40] als *S-Samples* oder *support patterns* bezeichnet.

Mittels eines mathematischen Tricks (Kernel-Trick) kann vermieden werden, dass für das Optimierungsproblem in dem hochdimensionalen Raum $\mathbb{R}^{d'}$ gerechnet werden muss. Wie in Gleichung 2.26 und 2.29 ersichtlich sind lediglich die Skalarprodukte der Abbildung ϕ zu bestimmen $K(\mathbf{x}_i, \mathbf{x}_j) := \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$. Für viele wichtige Projektionen kann man diese als *Kernel* bezeichnete Funktion $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ sehr leicht ausrechnen. Dies erlaubt einem Algorithmus also die Berechnung im niedrigdimensionalen Raum d, die jedoch mathematisch equivalent zur Berechnung des Skalarproduktes in dem „virtuellen“ hochdimensionalen Raum d' ist. Somit ergibt sich auch der Grund, warum die euklidische Norm verwendet wird, da keine andere Norm durch eine Berechnung des Skalarproduktes ausgedrückt werden kann.

Boosting führt die Berechnung jedoch explizit im Merkmalsraum durch und stellt eine *greedy search* (gierige Suche) auf einem gewichteten Trainingsdatensatz dar, mit dem Ziel neue (gute) Koordination zu finden. Die Koordinaten entsprechen dabei den Hypothesen, da das Kombinieren der einzelnen schwachen Hypothesen beim Boosting automatisch eine Abbildung der Form $\phi : \mathbf{x} \rightarrow \langle hWeak_1(\mathbf{x}), ..., hWeak_n(\mathbf{x}) \rangle$ darstellt.

Mit Boosting wird damit asymptotisch ein *Hard* Margin erreicht, so wie bei einer SVM für den Fall der Separierbarkeit. Verrauschte Beispiele (falsche Ausgabewerte y_i der Beispiele im Trainingsdatensatz) haben den gleichen Einfluss wie alle anderen. Was auch bedeutet, dass alle Trainingsdaten asymptotisch korrekt klassifiziert werden. In Betracht

[21] Der Offset b kann leicht aus den bestimmten Parametern und den Trainingsdatensatz bestimmt werden.

auf nicht korrekte Beispiele ist das nicht das richtige Konzept, da die „besten“ Entscheidungsgrenzen (vergleiche overfitting) gewöhnlich keinen Trainingsfehler von Null erreichen werden. *Soft* Margin kann bei SVM erzeugt werden indem die Bedingungen für das richtige Klassifizieren der Beispiele durch Einführen von *slack* Variablen (Schlupfvariablen) in das Optimierungsproblem etwas „aufgeweicht“ werden. Als Ergebnis ergibt sich eine Begrenzung der Gewichte α_t nach oben, die mittels des Parameters C (siehe Bedingung des Optimierungsproblems in Gleichung 2.27) eingestellt wird. Der Benutzer kann somit festlegen wieviel Wert er zum Einen auf die Minimierung der Anzahl der falsch klassifizierten Beispiele legt (großer Wert von C) und zum Anderen auf die Maximierung des minimalen Abstandes (Margin) der trennenden Hyperebene von einem richtig klassifizierten Beispiel. Es gibt Ansätze [40] um auch beim Boosting einen *Soft* Margin durch geeignete Anpassung der Fehlerfunktion zu erreichen.

In der praktischen Anwendung ist es wesentlich, im Fall von SVM, eine passende Projektion ϕ bzw. gleichbedeutend einen geeigneten Kernel zu finden[22]. Beim Boosting hingegen muss ein passender Lernalgorithmus für das erstellen der schwachen Hypothesen ausgewählt werden.

2.3.5 Beispiele

Um die oben beschriebenen theoretischen Betrachtungen zu veranschaulichen sind anbei zwei Beispiele gegeben. Jedes der Beispiele versucht wesentliche Aspekte des Boosting zu betonen. Die Attributswerte sind zweidimensional $\langle x_1, x_2 \rangle \in \mathbb{R}^2$ und als Ziel sollen möglichst gute Entscheidungsgrenzen zwischen den jeweils positiven und negativen Klassen $y \in \{-1, 1\}$ in diesem zweidimensionalen Raum gefunden werden.

XOR

Anhand dieses Beispiels soll vor allem der generelle Boosting Vorgang gezeigt werden und speziell die Eigenschaft der Generalisierung. Abbildung 2.7 zeigt die Verteilung der Beispiele. Es befinden sich je zwei Häufungen positiver bzw. negativer Beispiele diagonal liegend.

Als Trainingsdaten (Kreise) wurden an den entsprechenden Stellen jeweils 5 Datenpunkte und als Testdaten (Kreuze) wurden 50 Datenpunkte (normalverteilt mit $\sigma = 0.15$) generiert. Den Verteilungen der Trainings- und Testdaten steht im Wesentlichen derselbe datengenerierende Prozess dahinter, wobei die Testdaten etwas verrauscht sind. Das heißt aus dem partiellen Wissen, das mit den wenigen Trainingsbeispielen gegeben ist, soll auf das allgemeine Prinzip dahinter geschlossen werden, was ja einer guten Generalisierung entspricht.

Mit Hilfe des Trainingsdatensatzes wurde ein Klassifikator mittels AdaBoost laut Algorithmus 2.1 durchgeführt. Die Anzahl der Boosting Iterationen wurde auf 30 gesetzt.

[22] Ein Überblick ist zum Beispiel auf `http://www.kernel-machines.org` (14.09.2004) zu finden.

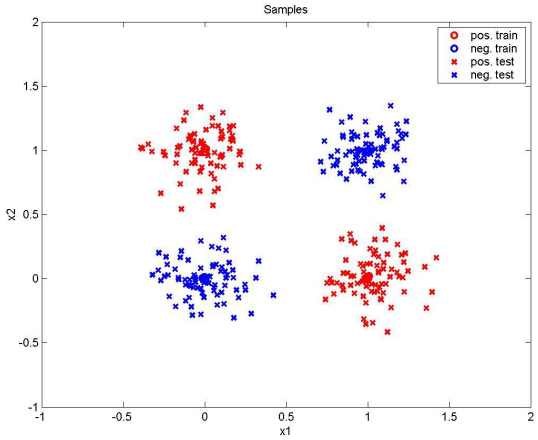

Abbildung 2.7: *XOR*-Beispiel: Trainings- bzw. Testdatensatz

Die einzelnen schwachen Klassifikationen können den Datensatz nur linear (durch eine Linie) trennen[23]. Es ist leicht zu zeigen, dass sich diese Beispiele nicht linear trennen lassen. Somit wird eine Hypothesenklasse mit größerer Komplexität benötigt um die Beispiele zu separieren. Diese Erhöhung soll durch Boosting bewirkt werden. Für das Trainieren dieser schwachen Klassifikatoren wurde, laut der Wahrscheinlichkeitserteilung W, aus den gesamten Trainingsdaten eine Untermenge bestimmt. Die Entscheidungsgrenzen der ersten vier Boosting Iterationen sind in Abbildung 2.8 zu sehen. Offensichtlich hat jeder schwache Klassifikator einen geringeren Fehler als 50% (genau 25%). Weiters erkennt man, dass jeweils eine Häufung abgetrennt wird.

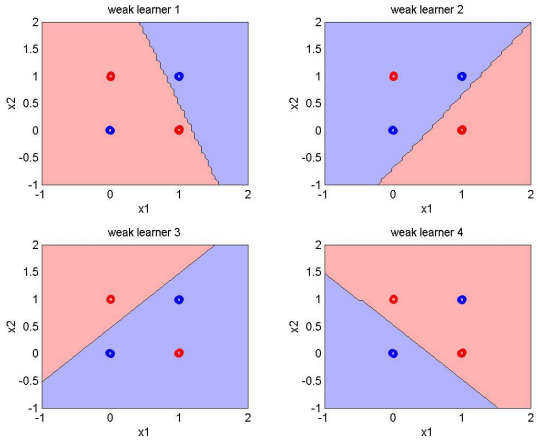

Abbildung 2.8: *XOR*-Beispiel: Entscheidungsgrenzen der schwachen Klassifikatoren für die ersten 4 Boosting Iterationen

Die jeweilige Kombination dieser schwachen Klassifikatoren zu einem starken Klassifikator ist in Abbildung 2.9 zu sehen. Auf der linken Seite ist der Rückgabewert (entspricht

[23]Die Komplexität der Hypothesenklasse entspricht dabei der VC-Dimension = 2

betragsmäßig den Margin) aufgetragen und auf der linken Seite sind die entsprechenden Entscheidungsgrenzen, also das Vorzeichen. Es ist schön zu erkennen wie die Entscheidungsgrenzen der einzelnen schwachen Klassifikatoren gewichtet und entsprechend überlagert werden. Dieser Vorgang ist bereits nach den ersten vier Iterationen klar ersichtlich. Je mehr Boosting Iterationen nun durchgeführt werden desto „feiner“ werden diese Grenzen. Der Finale Klassifikator nach 30 Iterationen ist in Abbildung 2.10 dargestellt. Obwohl also keiner der einzelnen schwachen Klassifikatoren einen Trainingsfehler unter 25% hat, erhält man bereits nach vier Boosting Iterationen einen Trainingsfehler von Null.

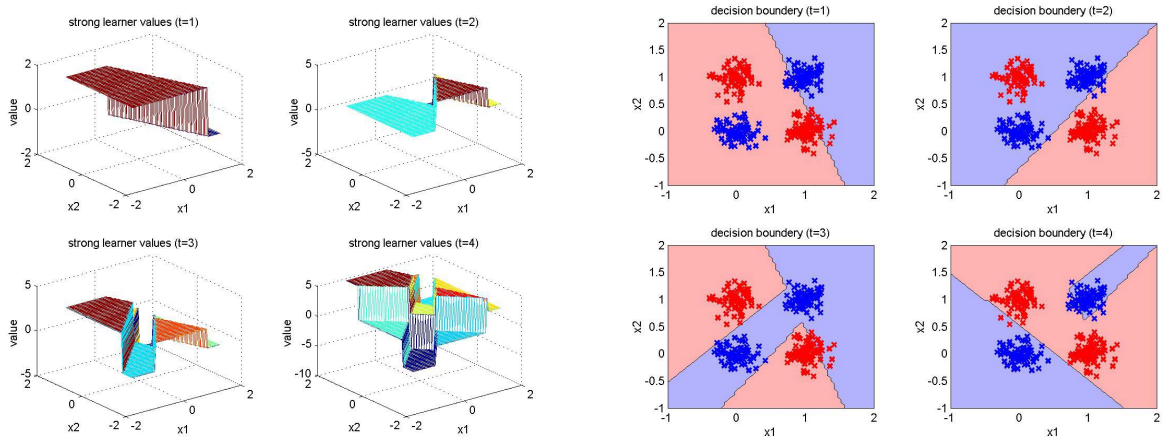

Abbildung 2.9: *XOR*-Beispiel: Ausgabewerte (links) und Entscheidungsgrenzen (rechts) der kombinierten starken Klassifikatoren in den ersten 4 Boosting Iterationen

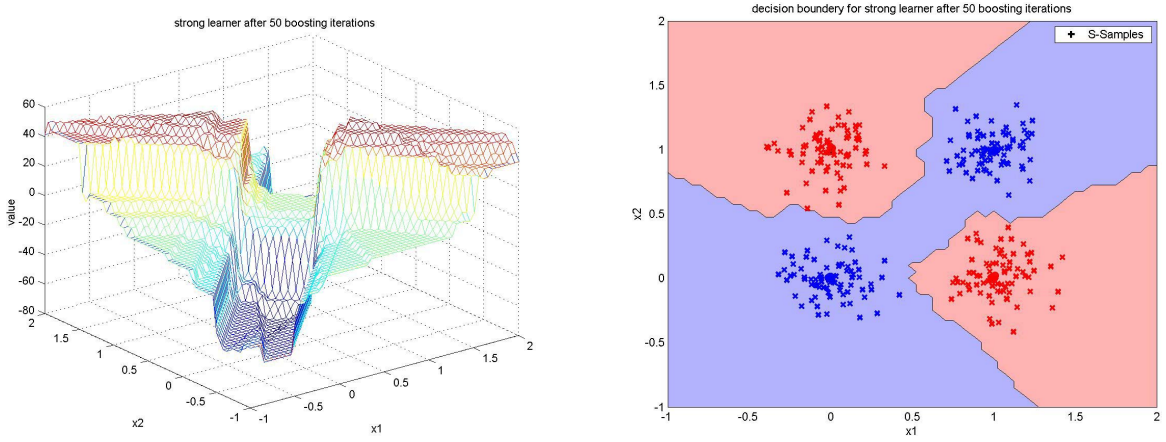

Abbildung 2.10: *XOR*-Beispiel: Ausgabewert und Entscheidungsgrenzen des finalen Klassifikators nach 50 Boosting Iterationen

Abbildung 2.11 zeigt den Verlauf des Trainings- bzw. Testfehlers über die Boosting Iterationen, also die Anzahl der verwendeten schwachen Klassifikatoren, die zum Kombinieren herangezogen wurden. Wie auch in den theoretischen Betrachtungen oben erläutert

lässt sich erkennen, dass der Generalisierungsfehler (Testfehler) weiter sinkt, obwohl der Trainingsfehler bereits nach den ersten vier Iterationen Null geworden ist. Diese Verbreiterung des Margins ist in Abbildung 2.12 für den Trainings und Testdatensatz bei drei unterschiedlichen Zeitpunkten dargestellt. Die Diagramme stellen die akkumulierten Wahrscheinlichkeiten über den erreichten Margin dar. Wie zu erkennen ist treten zunächst auch negative Werte auf. Dies bedeutet, dass einige Trainingsbeispiele noch falsch klassifiziert werden. Nach weiteren Boosting Iterationen nähert sich die Grenze aber immer weiter dem Optimum ($\frac{1}{2}$).

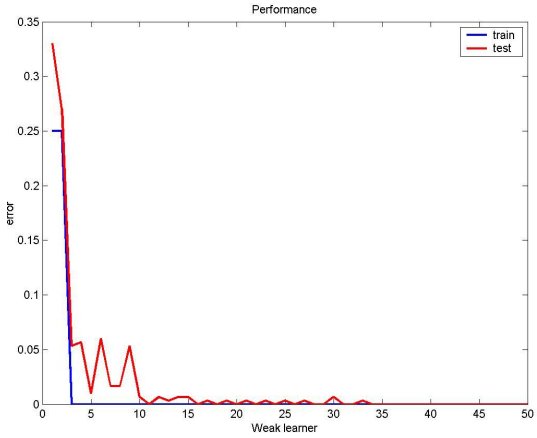

Abbildung 2.11: *XOR*-Beispiel: Verlauf des Trainings- und des Testfehlers über die Boosting Iterationen (Anzahl der schwachen Klassifikatoren)

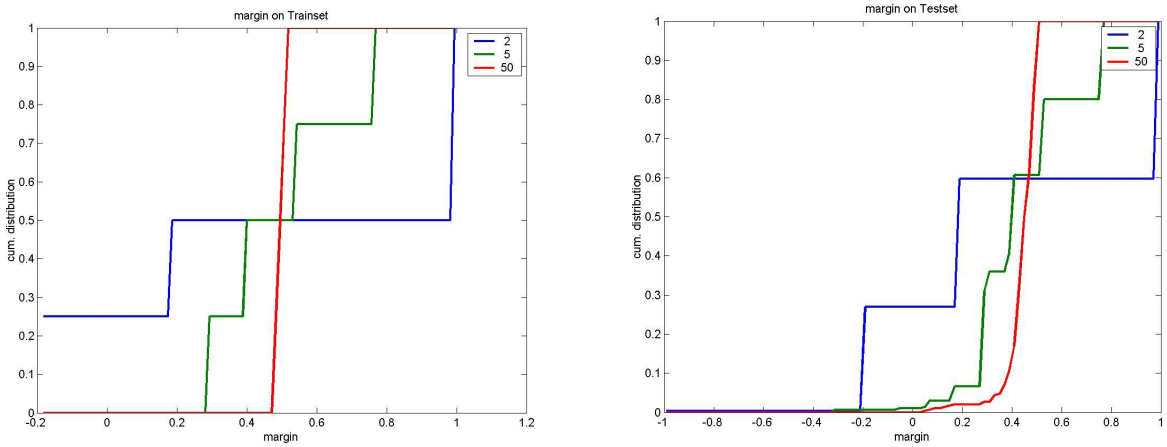

Abbildung 2.12: *XOR*-Beispiel: Verlauf des Margins der Trainingsbeispiele (links) und der Testbeispiele (rechts) bei drei unterschiedlichen Zeitpunkten des Trainings

Outlier

Dieses Beispiel soll demonstrieren, wie sich Boosting bei ungenauen Daten verhält und sich dabei auf die schwer zu erlernenden Beispiele einstellt. Die positiven bzw. negativen Beispiele dieses Datensatzes sind sehr leicht trennbar, jedoch ist in den Trainingsdaten ein Ausreißer (Outlier) vorhanden. Die Trainings- sowie Testdaten sind in Abbildung 2.13 zu sehen.

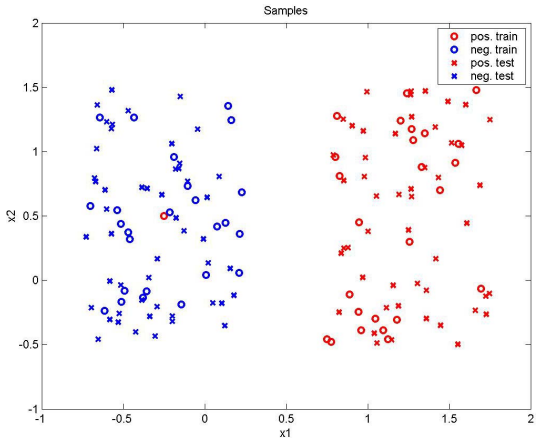

Abbildung 2.13: *Outlier*-Beispiel: Trainings- bzw. Testdatensatz

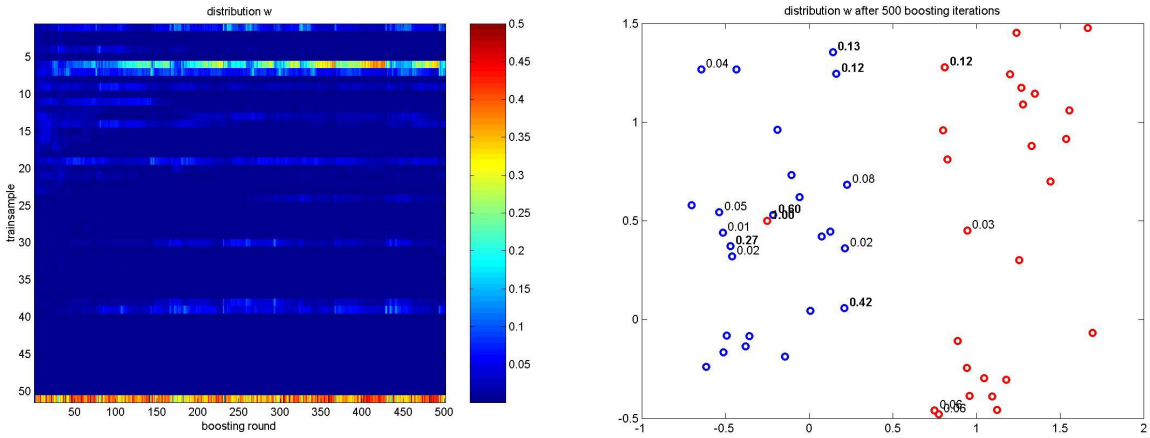

Abbildung 2.14: *Outlier*-Beispiel: (rechts) Verlauf der Wahrscheinlichkeitsverteilung W der Trainingsbeispiele über die Boosting Iterationen; (links) normierte Gewichte w_i nach 500 Boosting Iterationen

Bei diesem Beispiel erkennt man sehr schön, dass sich Boosting auf die schwierig zu klassifizierenden Beispiele konzentriert. In Abbildung 2.14 ist auf der linken Seite der Verlauf der Wahrscheinlichkeitsverteilung W über die Boosting Iterationen dargestellt. Das 40. Beispiel (letztes) stellt den positiven Ausreißer in der negativen Verteilung dar. Da der

Wahrscheinlichkeitswert über den gesamten Bereich sehr groß ist, wird er mit sehr hoher Wahrscheinlichkeit immer wieder zum Trainieren eines weiteren schwachen Klassifikators verwendet werden. Auf der rechten Seite der Abbildung sind die normierten Werte der Wahrscheinlichkeitsverteilung am Ende (nach 500 Iterationen) zu sehen. Der größte Wert ist der Ausreißer selbst und die Beispiele in seiner nahen Umgebung. Diese Tatsache kann auch dazu benutzt werden um Beispiele, die sehr hohe Wahrscheinlichkeitswerte besitzen, als Outlier zu erkennen. Da beim Boosting auf jeden Fall ein Trainingsfehler von Null erreicht wird, ist jedes Beispiel für die Beschreibung gleichwertig. Es wird also ein *Hard* Margin erreicht, das heißt es wird auch auf jeden Fall der Ausreißer richtig klassifiziert. Die Beispiele mit einem hohen Gewicht entsprechen den Beispielen des minimalen Margin und stellen die *S-samples* dar, die als Analogie zu den *support vectors* einer SVM angesehen werden können. Sie stellen die schwierigen Beispiele dar, die für die Bestimmung der Entscheidungsgrenzen von maßgeblicher Bedeutung sind. Man erkennt auch in diesem Fall der Margin Verteilung (Abbildung 2.15), dass mit dem Steigen der Boosting Iterationen versucht wird, den minimalen Margin zu maximieren.

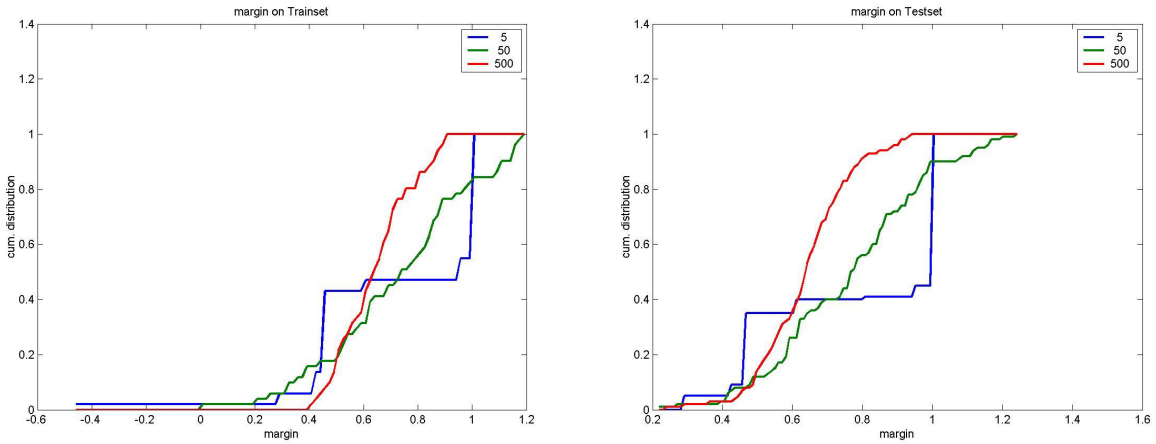

Abbildung 2.15: *Outlier*-Beispiel: Verlauf des Margins der Trainingsbeispiele (links) und der Testbeispiele (rechts) bei drei unterschiedlichen Zeitpunkten des Trainings

Die Entscheidungsgrenzen des finalen starken Klassifikators ist aus Abbildung 2.16 ersichtlich. In der Abbildung 2.17 ist ein Vergleich mit einer SVM dargestellt (erstellt mittels [12]. Man erkennt, dass die support vectors in etwa den Beispielen mit hohen Gewichten entsprechen (S-Samples).

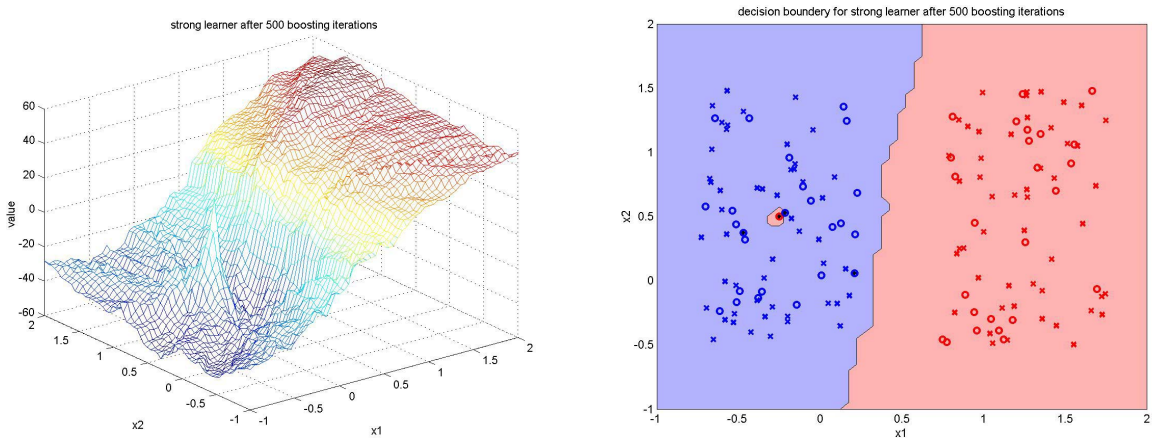

Abbildung 2.16: *Outlier*-Beispiel: Ausgabewert und Entscheidungsgrenzen des finalen Klassifikators nach 500 Boosting Iterationen

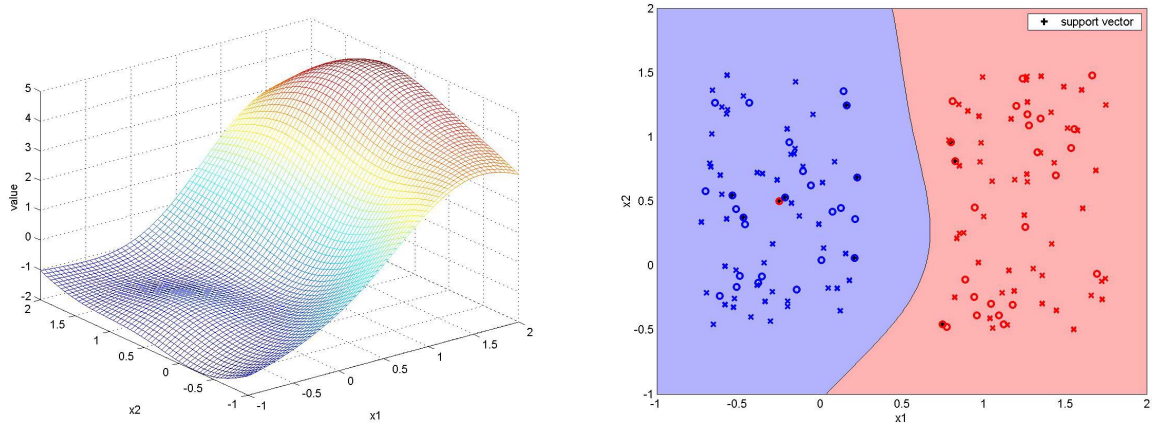

Abbildung 2.17: *Outlier*-Beispiel: Vergleich zu einer SVM (RBF-Kernel; C=500)

Kapitel 3

Objekt Detektionssystem

Anhand des allgemeinen Modells für ein Objektdetektionssystem, wie in Abschnitt 2.2.1 beschrieben, soll ein konkretes System entwickelt werden. In diesem Abschnitt wird auf die Erstellung eines Klassifikators eingegangen. Die wesentlichen Punkte sind also Merkmale auswählen, Modell auswählen, Klassifikation und Evaluieren. Da es sich bei dem vorgestellten Verfahren um eine Merkmalsselektion handelt verschwimmen die Grenzen der einzelnen Punkte. Ziel dieses Kapitels ist es also einen Klassifikator, das Kernstück, für die Erstellung eines beliebigen Objektdetektionssystems zu schaffen. Die weiteren noch offenen Punkte, werden im Anschluss, in Kapitel 4, an konkreten Objekten (Ansichten von Autos) erläutert.

Das hier gezeigte Verfahren basiert auf der Methode von Viola und Jones [55], [54] die für die Detektion von Gesichter vorgestellt wurde. Nach einem kurzen Überblick wird auf die einzelnen Komponenten des Klassifikators eingegangen und wie diese benutzt werden können um einen robusten und schnellen Detektor zu gestalten. Dieser Detektor wird dann schrittweise auf das gesamte Bild angewendet. Dabei werden im Allgemeinen mehrere Detektionen auftreten, die anschließend in einem Nachverarbeitungsschritt geeignet miteinander kombiniert werden.

Die Implementierung der Trainings- als auch der Testphase wurde in *Matlab* durchgeführt. Um eine schnelle Detektion zu erreichen, könnte der Code für die Evaluierung einer bereits trainierten Kaskade zum Beispiel in *C++* oder einer anderen Programmiersprache erfolgen, die eine schnelle Auswertung erlaubt.

3.1 Grundlage

Ein Objekt kann an einer beliebigen Stelle und in beliebiger Größe in einem gegebenen Bild auftreten. Auch können mehrere Objekte, oder keines, in dem Bild vorhanden sein. Es gibt grundlegend zwei Methoden um dieses Problem zu bewältigen. *Invariante Methoden* versuchen Merkmale oder Filter zu verwenden, die invariant bezüglich der Geometrie sind. Die Schwierigkeit bei diesen Methoden ist diese Merkmale zu finden, die einerseits invariant sind aber dennoch zwischen den unterschiedlichen Klassen unterscheiden können.

Der zweite Ansatz besteht darin das gesamte Bild schrittweise zu durchsuchen. Diese Suche muss also für jede Skalierung und jede mögliche Position eines Suchfensters durchgeführt werden und feststellen ob dort ein gesuchtes Objekt enthalten ist oder nicht. Da diese Suche sehr viel Zeit in Anspruch nehmen kann wird die Methode als *Exhaustive Search*, also als ermüdende Suche bezeichnet. Dieser Suchvorgang ist in Abbildung 3.1 dargestellt. Sollen mehrere unterschiedliche Objekte in einem Bild gefunden werden, muss der gesamte Vorgang für jedes Objekt wiederholt werden.

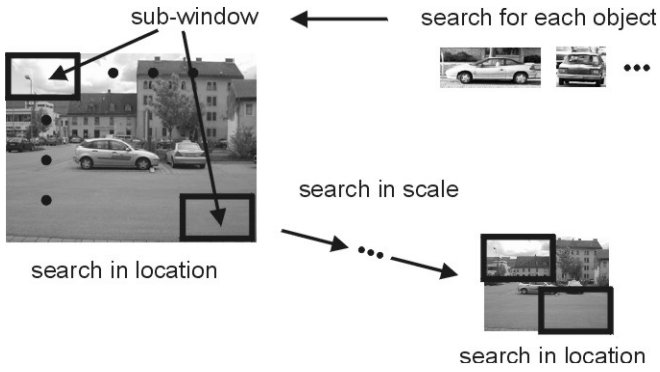

Abbildung 3.1: *Exhaustive Search* für Objektdetektion. Für jedes Objekt muss sowohl in Position als bei unterschiedlichen Skalierungen ein Suchfenster evaluiert werden.

In dieser Arbeit wird *exhaustive search* verwendet um die verschiedenen Objekte zu lokalisieren. Es sei noch erwähnt, dass kein weiteres Vorwissen über den Aufbau der Szene oder die Beleuchtung verwendet wird um den Suchraum einzuschränken. Das hier vorgestellte Verfahren ist also sehr allgemein einsetzbar, kann jedoch in speziellen Fällen durch Einbringen von Vorwissen weiter beschleunigt werden. Ist zum Beispiel die Grundebene des Aufbaus bekannt, kann die Größe und Position des Suchfensters drastisch eingeschränkt werden.

3.2 Merkmale

Der Hauptgrund warum (kompliziertere) Merkmale (*Features*) als Input für einen Lernalgorithmus verwendet werden als einfach die rohen Pixeldaten ist, dass diese Merkmale eine leichtere Trennung zwischen den positiven und negativen Beispielen ermöglichen. Merkmals basierende Methoden können ad-hoc Wissen einbringen auch wenn die Anzahl der Trainingsbeispiele relativ gering ist. Die Variabilität der positiven Trainingsbeispiele soll also verringert und im Gegensatz die der negativen Trainingsbeispiele erhöht werden um typische Konstellationen leichter zu erkennen.

Die verwendeten Merkmale lehnen sich an eine Wavelet Transformation mit *Haar basis functions* an. Eine kurze Einführung in das Thema Wavelets ist zum Beispiel in [47] gegeben. Es werden die in Abbildung 3.2 gezeigten sechs verschiedenen Prototypen verwendet. Der Wert f_i eines Merkmales i wird berechnet, indem die Differenz der Pixelwerte in dem

weißen und dem schwarzen Bereich gebildet wird. Ein kleiner Koeffizient stellt dabei eine gleichmäßige Fläche dar, wogegen eine hoher Koeffizient eine starke Änderung in der Intensität der Bereiche entspricht.

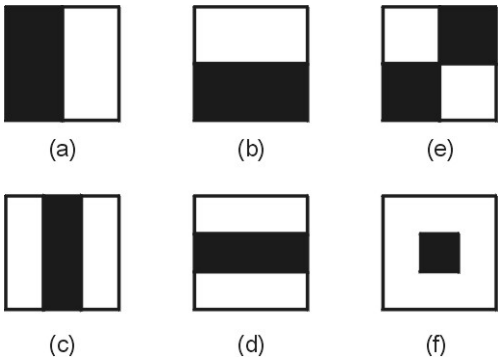

Abbildung 3.2: Verwendete rechteckige Prototypen von *Haar-like* (a)-(e) und *center-surround* (f) Merkmale. Jeder Prototyp kann Merkmale erzeugen, die sich sowohl in Position als auch Skalierung (Höhe und Breite) unterscheiden. Der Wert eines Merkmals wird berechnet, indem man die Differenz der weißen und schwarzen Bereiche bildet.

Die Anzahl der so erstellten Merkmale ist sehr groß, da jedes Merkmal sich in Position und Skalierung beliebig verändern kann. Die genaue Anzahl verändert sich von Prototype zu Prototype. Sei H und W die Höhe und Breite eines Rasterbildes, h und b jeweils die Höhe und Breite des kleinstmöglichen Merkmals dieses Prototypes und weiters $h_{maxScale} = \left\lfloor \frac{H}{h} \right\rfloor$ und $w_{maxScale} = \left\lfloor \frac{W}{w} \right\rfloor$ die maximalen Skalierungsfaktoren, die sich daraus ergeben. Die Anzahl der verschiedenen Möglichkeiten für das Auftreten dieses Prototypes ergibt sich dann laut Gleichung 3.1 [30].

$$h_{maxScale} \cdot w_{maxScale} \cdot \left(W + 1 - w\frac{w_{maxScale} + 1}{2}\right) \cdot \left(H + 1 - h\frac{h_{maxScale} + 1}{2}\right) \qquad (3.1)$$

In einem 10×10 großen Pixelbild sind 5250 mögliche Merkmale enthalten, je 1375 für Typ (a) und (b), je 825 für Typ (c) und (d) sowie 625 für Typ (e) und 225 für Typ (f). Diese Zahl steigt für größere Bilder enorm an. Im Gegensatz zur *Haar basis* der Wavelet Transformation ist diese Darstellung *over-complete*[1]. Ziel ist es nun aus dieser großen Auswahl von möglichen Merkmalen diejenigen auszuwählen, die die Objekte am Besten beschreiben.

In dem System von Viola und Jones [55] wurden nur die ersten fünf Typen (a)-(e) verwendet. Vor allem in Hinblick auf die Detektion von Rädern wurde noch ein weiteres Merkmal *center-surround*, Typ (f) hinzugefügt. Es können natürlich noch weitere Merkmale

[1]Eine *complete basis* hat keine linearen Abhängigkeiten zwischen den Basis-Elementen und hat demzufolge gleich viele Elemente wie das Bild, hier also 100.

dieser Art hinzugefügt werden, wie es zum Beispiel in [33] gemacht wurde um explizit globale Symmetrien zu erfassen. Des weiteren gibt es Ansätze in denen auch noch rotierte Versionen der ursprünglichen Merkmale verwendet werden (Lienhart et. al. [30], [29] und [3]).

3.2.1 Integral Image

Rechteckige Merkmale können sehr effizient berechnet werden, wenn das Bild in der Datenstruktur eines *Integral Image* vorliegt. Der Wert eines Integral Image an der Position x, y stellt die Summe aller Pixel darüber und links davon dar. Die formale Definition ist in Gleichung 3.2 gegeben.

$$intImage(x, y) = \sum_{x'=1}^{x} \sum_{y'=1}^{y} image(x', y') \tag{3.2}$$

Die Berechnung kann effizient in einem Durchlauf berechnet werden, wird der angegebene Algorithmus 3.1 verwendet.

Algorithmus 3.1 erstellen eines Integral Image $intImage$

Require: Bild $image$ und dessen Größe $sizeX, sizeY$
  for $x = 1, 2, ..sizeX$ **do**
    • $cumRow(x, -1) = 0$
    for $y = 1, 2, ..sizeY$ **do**
      • $intImage(-1, y) = 0$
      • $cumRow(x, y) = cumRow(x, y - 1) + image(x, y)$
      • $intImage(x, y) = intImage(x - 1, y) + cumRow(x, y)$
    end for
  end for

Hat man einmal das Integral Image berechnet, können sämtliche Merkmale in konstanter Zeit berechnet werden und zwar unabhängig von der Position und der Skalierung des Bildes. Jeder einzelne rechteckige Teil eines Feature stellt ja die Summe der darin enthaltenen Pixelwerte dar. Wie Abbildung 3.3 zeigt, kann von einem Integral Image $intImage$ die Summe des grau unterlegten Bereiches, definiert durch die Eckpunkte $P_1, ...P_4$, einfach durch $value = intImage(P_4) + intImage(P_1) - intImage(P_2) - intImage(P_3)$ berechnet werden. Mit Hilfe einer Kombination und möglichen Vereinfachungen können damit also sämtliche in Abbildung 3.2 gezeigten Merkmale mit nur sehr wenigen Referenzen berechnet werden. Die Skalierung des Bildes wird auf eine Skalierung der Merkmale zurückgeführt. Es muss also nur einmal ein Integral Image berechnet werden, die Berechnung der Merkmalswerte verläuft immer in konstanter Zeit. Es ist jedoch bei der Skalierung der Merkmale zu beachten, dass automatisch eine Interpolation des ursprünglichen Bildes vorgenommen wird. Die errechneten Werte müssen mit einem Korrekturfaktor multipliziert werden um bei der nachfolgenden Schwellwertoperation der schwachen Klassifikatoren die gewünschten (gleichen) Ergebnisse zu erreichen als wenn das Bild skaliert worden wäre.

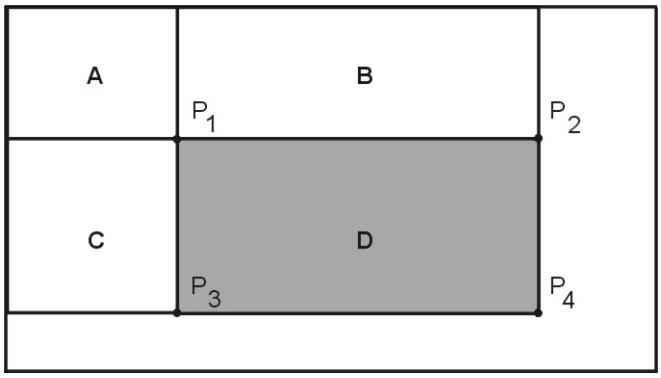

Abbildung 3.3: Effiziente Berechnung der Summe eines rechteckigen Fensters mit Hilfe eines Integral Image. Der Wert des Integral Images bei Punktes P_1 stellt dei Summe der Pixelwerte in Rechteck A dar. P_2 entspricht $A+B$, P_3 ist $A+C$ und P_4 ist $A+B+C+D$. Die Summe des unterlegten Rechtecks D kann somit mit nur vier Referenzen als $P_4+P_1-P_2-P_3$ erfolgen.

3.3 Klassifikator

Ist eine Menge von möglichen Merkmalen und ein Datensatz mit positiven und negativen Beispielen gegeben, kann ein beliebiges Lernverfahren verwendet werden um die Beispiele zu separieren. Wie oben beschrieben sind in einem Suchfenster jedoch sehr viele mögliche Merkmale vorhanden. Auch wenn die einzelnen Werte effizient berechnet werden können, so ist doch die Berechnung der kompletten Menge sehr aufwendig. Die grundlegende Annahme von Viola und Jones [55] ist, dass eine Kombination von einer kleinen Anzahl dieser Merkmale einen effizienten Klassifikator bilden kann. Um dies zu erreichen wird eine Variante des *AdaBoost*-Algorithmus verwendet.

Nach einer allgemeinen Vorstellung des verwendeten Detektionssystems wird in den einzelnen Abschnitten auf die einzelnen verwendeten Klassifikatoren und deren Zusammenhang eingegangen um ein möglichst schnelles und robustes Detektionssystem zu erstellen.

3.3.1 Übersicht

Für die Detektion wird das in Abschnitt 3.1 vorgestellte Suchverfahren *exhaustive search* verwendet. Wie oben erwähnt ist diese Suche sehr zeitaufwendig. Man versucht daher den benötigten Klassifikator, der für jedes mögliche Suchfenster die Entscheidung liefert, möglichst schnell zu gestalten. Dies geschieht hier mittels des Kaskadenansatzes. Es wird sich herausstellen, dass der durchschnittliche Rechenaufwand pro Suchfenster gering ist obwohl sehr gute Ergebnisse erzielt werden können.

Die schematische Darstellung des Kaskadenansatzes ist in Abbildung 3.4 gezeigt. Die grundlegende Idee ist, dass in einem gegebenen Input-Bild sehr viele Suchfenster das zu detektierende Objekt nicht enthalten. Ein Kaskaden Klassifikator besteht aus einer Hintereinanderschaltung verschiedener Klassifikatoren. Jeder dieser Klassifikatoren macht eine

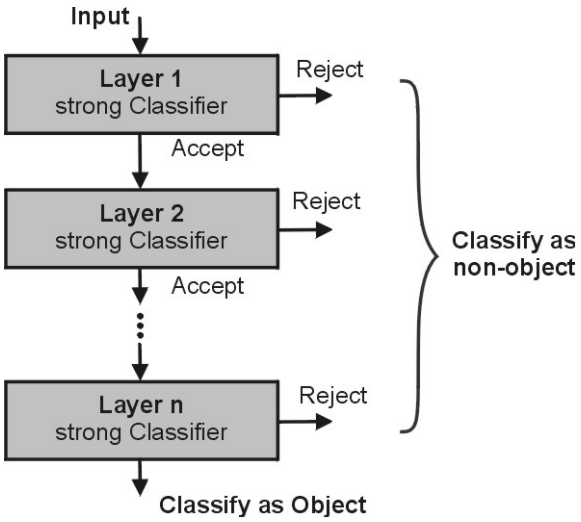

Abbildung 3.4: Schematische Beschreibung eines Kaskaden Klassifikators. Viele Suchfenster werden das Objekt nicht beinhalten. Diese sollen bereits in frühen Schichten (mit hoher Sicherheit) verworfen werden wobei Rechenzeit gespart wird.

Entscheidung ob es sich um ein potentielles Zielobjekt handelt oder nicht. Nur wenn das Suchfenster alle Kaskadenschichten (*Layer*) passiert hat wird es als Objekt detektiert. Ziel ist es also in möglichst frühen Schichten der Kaskade, Suchfenster, die das Objekt nicht enthalten (Hintergrund,...), zu verwerfen und damit den benötigten Rechenaufwand einzuschränken. Wie noch gezeigt wird, wird das Lernproblem in den Kaskadenschichten immer schwieriger. Das heißt, dass für die Entscheidung immer kompliziertere Klassifikatoren benötigt werden, die rechenintensiver sind als jene in den ersten Kaskadenschichten.

In den nächsten Abschnitten wird gezeigt wie ein solcher Kaskaden Klassifikator erstellt werden kann. Aufbauend auf den Merkmalen werden zunächst schwache Klassifikatoren erstellt, die danach mittels des Boosting Ansatzes zu starken Klassifikatoren kombiniert werden und danach in einer Kaskadenschicht verwendet werden.

3.3.2 Schwacher Klassifikator

Die verwendeten Merkmale (siehe Abschnitt 3.2) stellen die Basis der schwachen Lerner dar. Die zugrundeliegende Idee hinter einem schwachen Klassifikator $hWeak$ ist, einen Schwellwert $\theta Weak$ zu finden, der die Werte des Merkmals j zwischen den positiven und negativen Beispiel gut separiert. Dies ist in Gleichung 3.3 dargestellt, wobei p_j (Parity) festlegt ob der Wert des Merkmals f_j größer oder kleiner als der Schwellwert θ_j sein soll. Ein schwacher Klassifikator (*weak classifier*) besteht also aus diesen zwei Parametern. Die einzige Forderung die an diesen schwachen Klassifikator gestellt wird ist, dass er einen Fehler kleiner als 50% (binäres Problem) besitzt[2].

[2]Diese Forderung ist mit dem angegebenen schwachen Lernen leicht zu erfüllen. Nur in dem Fall, dass beide Verteilungen (positive und negative Trainingsbeispiele) deckungsgleich sind ergibt sich ein Fehler

$$hWeak_j(x) = \begin{cases} 1 & \text{falls } p_j \cdot f_j(x) < p_j \cdot \theta Weak_j \\ -1 & \text{sonst} \end{cases} = p_j \cdot \text{sign}(f_j(x) - \theta Weak_j) \quad (3.3)$$

Eine schnelle Variante einen Threshold und ein Parity-Flag zu bestimmen beschreibt der Algorithmus 3.2. Hierbei werden die Mittelwerte der Verteilung der positiven und negativen Trainingsbeispiele ermittelt, die bei der Berechnung des Merkmals j auftreten. Der Schwellwert $\theta Strong_j$ stellt den arithmetischen Mittelwert der beiden dar und das Parity-Flag p_j ergibt sich direkt aus dem Vorzeichen der Differenzen.

Natürlich könnten hierbei auch andere (bessere) Ansätze verwendet werden. Ein Vorteil des angegebenen Verfahrens ist zweifellos, dass die Entscheidung sehr schnell berechnet werden kann, da es sich im Prinzip nur um eine Schwellwertoperation handelt. In der Praxis werden sich aber in der Regel recht hohe Fehlerraten ergeben.

Algorithmus 3.2 trainieren eines schwachen Klassifikators $hWeak$

Require: positive und negative Trainingsdaten $P = \{\langle \mathbf{x_i}, y_i \rangle | y_i = 1\}$ bzw. $N = \{(\mathbf{x_i}, y_i) | y_i = -1\}$

Require: Feature j

- bestimme Mittelwerte der positiven bzw. negativen Verteilungen

$$meanP = \frac{1}{|P|} \cdot \sum_{x \in P} f_j(x)$$
$$meanN = \frac{1}{|N|} \cdot \sum_{x \in N} f_j(x)$$

- bestimme $\theta Weak_j$ und p_j

$$\theta Weak_j = \frac{1}{2}(meanP + meanN)$$
$$p_j = \text{sign}(meanP - meanN)$$

- trainierter schwacher Klassifikator

$$hWeak(x) = p_j \cdot \text{sign}(f_j(x) - \theta Weak_j)$$

3.3.3 Starker Klassifikator

Aus den einzelnen schwachen Klassifikatoren wird nun ein starker Klassifikator (*strong classifier*) generiert. Die schwachen Klassifikatoren erfüllen alle die notwendige Bedingung, dass sie einen Fehler kleiner 50% besitzen. Somit kann der Boosting Algorithmus 3.3 angewandt werden. Es handelt sich hierbei um eine leicht veränderte Variante von AdaBoost (vergleiche Algorithmus 2.1 aus Abschnitt 2.3.1). Es werden zunächst einmal für eine gegebene Menge von Merkmalen die dazugehörigen schwachen Klassifikatoren trainiert. Diese werden dann mittels des Boosting Ansatzes kombiniert. Da es sich bei den einzelnen schwachen Klassifikatoren eigentlich um verschiedene Merkmale handelt, entspricht dieser Vorgang einer Merkmales-Selektion (*feature selection*).

von 50%

Damit ergibt sich für den starken Klassifikator eine Linearkombination von verschiedenen schwachen Klassifikatoren, die verschiedenen Merkmale entsprechen. Dies ist mathematisch in Gleichung 3.4 veranschaulicht.

$$hStrong(x) = \text{sign}\Big(\sum_{t=1}^{T} \alpha_t \cdot hWeak_t(x) - \theta Strong\Big) \tag{3.4}$$

In dieser Definition tritt der unbekannte Term $\theta Strong$ auf. Dieser wird dazu verwendet um die Detektionsrate auf Kosten der false-positive Rate zu steigern. Die Eigenschaft, dass ein starker Klassifikator eine möglichst große Detektionsrate haben muss, wird noch näher beschrieben und hat seinen Ursprung in dem Kaskadenansatz.

Algorithmus 3.3 trainieren eines starken Klassifikators $hStrong$

Require: ein Trainingsdatensatz $S = \langle\langle \mathbf{x_1}, y_1\rangle, ..., \langle \mathbf{x_n}, y_n\rangle\rangle, x_i \in X, y_i \in Y = \{-1, +1\}$

Require: mögliche Merkmale $j \in J$

Require: Anzahl der Boosting Iterationen T (entspricht $numFeatures$)

- für alle Merkmale j trainiere einen schwachen Klassifikator $hWeak_j$
- initialisiere $w_{i,1} = \frac{1}{n}, i = 1, ..., n$

for $t = 1, 2, .., T$ **do**

- normalisieren der Gewichte, sodass W_t eine Wahrscheinlichkeitsverteilung ist
$$w_{t,i} = \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
- bestimme Trainingsfehler in Betracht der Verteilung W_t und all den zuvor trainierten schwachen Klassifikatoren $hWeak_j$
$$e_j = \sum_{i:hWeak_j(\mathbf{x_i}) \neq y_i} w_{t,i}$$
- wähle jenen Klassifikator $hWeak_j$ mit dem geringsten Fehler e_j
$$e_t = e_j \qquad hWeak_t = hWeak_j \qquad \text{wobei } \forall_i e_j \leq e_i$$
- Abbruch wenn $e_t = 0$ oder $e_t > \frac{1}{2}$
- Faktor berechnen
$$\alpha_t = \frac{1}{2} \cdot \ln\left(\frac{1-e_t}{e_t}\right)$$
- Aktualisierung der Gewichte vornehmen
$$w_{t,i} = w_{t,i} \cdot \exp(-\alpha_t \cdot y_i \cdot hWeak_t(\mathbf{x_i}))$$

end for

- trainierter starker Klassifikator
$$hStrong(x) = \text{sign}\Big(\sum_{t=1}^{T} \alpha_t \cdot hWeak_t(x)\Big)$$

Das Trainieren des starken Klassifikators laut Algorithmus 3.3 liefert also ein gutes, *allgemeines* Ergebnis (Detektionsrate und false-positive Rate „optimiert“). Für die Entscheidung wird die sign(·) Funktion verwendet, also ein Schwellwert bei Null. Wird der Schwellwert $\theta Strong^{\star}$ verringert (negativer) so ergibt sich zwangsläufig eine Erhöhung der Detektionsrate, wobei jedoch auch zwangsläufig die false-positive Rate erhöht wird. Abbildung 3.5 veranschaulicht dieses.

Der Schwellwert $\theta Strong$ wird in dieser Stufe des Trainings jedoch auf Null gesetzt und erst bei dem Trainieren der Kaskade entsprechend angepasst.

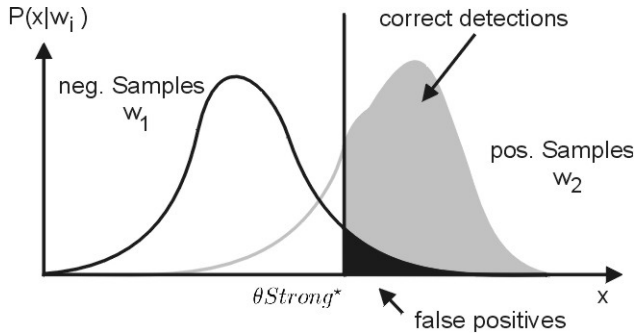

Abbildung 3.5: Schwellwertanpassung um die Detektionsrate zu steigern. Alle Werte die größer als der aktuelle Schwellwert $\theta Strong^{\star}$ sind werden als positive erkannt. Wird nun der Schwellwert verringert, so steigt die Detektionsrate (mehr correct detections) auf Kosten der false-positive Rate (mehr false-positives) an.

3.3.4 Kaskaden Klassifikator

Wie in den vorangegangenen Abschnitten beschrieben, werden, basierend aus Merkmalen j, zunächst die zugehörigen schwachen Klassifikatoren $hWeak_j$ trainiert. Diese werden dann zu starken Klassifikatoren $hStrong$ „geboostet“. Nun werden jeweils unterschiedliche starke Klassifikatoren zu einem Kaskaden Klassifikator (*cascade classifier*) zusammengeschlossen. Dabei besitzen die einzelnen starken Klassifikatoren unterschiedliche Eigenschaften. Es wird in jeder Kaskadenschicht i gefordert, dass ein starker Klassifikator eine minimale Detektionsrate d_i und eine maximale false-positive Rate fp_i hat.

Für eine gesamte Kaskade multiplizieren sich somit die einzelnen Detektionsraten d_i bzw. die false-positive Raten fp_i somit zu der generellen Detektionsrate D bzw. false-positive Rate FP zusammen. Dieser Zusammenhang der Hintereinanderschaltung ist in Gleichung 3.5 bei einer Kaskade mit $numLayer$ Anzahl von Kaskadenschichten formal beschrieben.

$$FP = \prod_{layer=1}^{numLayer} fp_{layer} \qquad D = \prod_{layer=1}^{numLayer} d_{layer} \tag{3.5}$$

Ziel ist es natürlich eine möglichst hohe Detektionsrate bei einer möglichst kleinen false-positive Rate zu erreichen. Laut der Kaskadenstruktur ist es daher erforderlich eine sehr große Detektionsrate (nahe bei 1, zum Beispiel $d_i = 0.995$) in jeder Schicht zu fordern. Die false-positive Raten können im Vergleich dazu relativ groß gewählt werden (zum Beispiel $fp_i = 0.4$). Die positiv erkannten Beispiele und somit auch alle false-positives werden ja an die nächsten Kaskadenschicht weitergeleitet und dort weiter untersucht (vergleiche Abbildung 3.4). In dieser Implementierung sind die Raten in der Kaskadenschicht jeweils gleich. Bei einer Kaskade mit 10 Schichten ergibt sich mit den angegebenen Werten somit im schlechtesten Fall eine Detektionsrate von $D = d^{10} = 0.995^{10} = 0.9511$ und eine false-positive Rate von $FP = fp^{10} = 0.4^{10} = 1.05 \cdot 10^{-4}$. Man erkennt, dass beide Kenngrößen

exponentiell sinken, jedoch wirkt sich das aufgrund der unterschiedlichen Basis verschieden stark aus.

Daraus ergibt sich direkt, dass in der Regel ein Kaskaden Klassifikator nicht eine so gute Leistung (*Performance*) wie ein einziger Klassifikator mit einer großen Anzahl von Merkmalen erreichen kann. Der Vorteil liegt in der benötigten Rechenzeit. Wird ein einziger Klassifikator mit n Merkmalen verwendet, müssen in jedem Suchfenster $N = n$ Merkmale evaluiert werden. Die durchschnittliche Anzahl der Evaluierungen pro Suchfenster bei einem Kaskaden Klassifikator ist dabei laut Gleichung 3.6 zu erwarten aber in den meisten Fällen wesentliche geringer. Dabei ist n_i die Anzahl der Merkmale und p_i die positive Rate (alle positiven Ernennungen) in der i-ten Kaskadenschicht.

$$N = n_1 + \sum_{layer=1}^{numLayer} \left(n_{layer} \prod_{i<layer} p_i \right) \tag{3.6}$$

Der Kaskadenansatz versucht einen guten Kompromiss zwischen der zu erreichenden Performance und den dazu notwendigen Rechenaufwand zu erzielen. Klassifikatoren mit mehreren Merkmalen werden in der Regel eine höhere Detektionsrate und eine geringere false-positive Rate (gute Performance) erzeugen, jedoch gleichzeitig länger für diese Berechnung benötigen. Prinzipiell könnte also ein Optimierungsproblem, in das die Anzahl der Kaskadenschichten $numLayer$, die Anzahl der verwendeten Merkmale in den Kaskadenschichten n_i und die jeweiligen Detektionsraten d_i bzw. false-positive Raten fp_i eingeht, aufgestellt werden. Unglücklicherweise stellt das Finden eines Optimums eine extrem schwierige Aufgabe dar. In der Praxis wird daher ein sehr simples Verfahren verwendet. Der gesamte Trainingsvorgang ist schematisch in Abbildung 3.6 dargestellt bzw. als Pseudocode in Algorithmus 3.4.

Zu Beginn des Trainings ist eine Menge von positiven Beispielen vorhanden. Aus einer Datenbank von Bilder, in denen keine positiven Objekte enthalten sind, werden zufällig negative Beispiele generiert. Der Benutzer gibt die gewünschten Performance-Parameter an, das heißt die maximale false-positive Rate pro Kaskadenschicht fp, die minimale Erkennungsrate pro Kaskadenschicht d und als Abbruchkriterium die zu erreichende false-positive Rate FP_{target}.

Als erster Schritt wird zufällig, aus der enorm großen Menge von allen möglichen Merkmalen, eine kleinere Untermenge (nur etwa 1%, vom Benutzer anzugeben) ausgewählt. Dies liegt nicht zuletzt daran, die benötigte Rechenzeit und den benötigten Speicher in Grenzen zu halten. Es wäre interessant zu untersuchen ob eine größere Auswahl oder eine heuristische Anpassung, wie etwa in [33] beschrieben ist, eine wesentliche Verbesserung der Performance bringen würde, da auch mit diesem Ansatz sehr gute Ergebnisse erzielt werden konnten.

Aufbauend auf diesen generierten Merkmalen und dem Trainingsdatensatz wird in jeder Kaskadenschicht ein starker Klassifikator, wie in Abschnitt 3.3.3 beschrieben, trainiert. Da immer mehr schwache Klassifikatoren, die ja den Merkmalen und der Anzahl der Boosting Iterationen entsprechen, verwendet werden, wird auch der starke Klassifikator immer

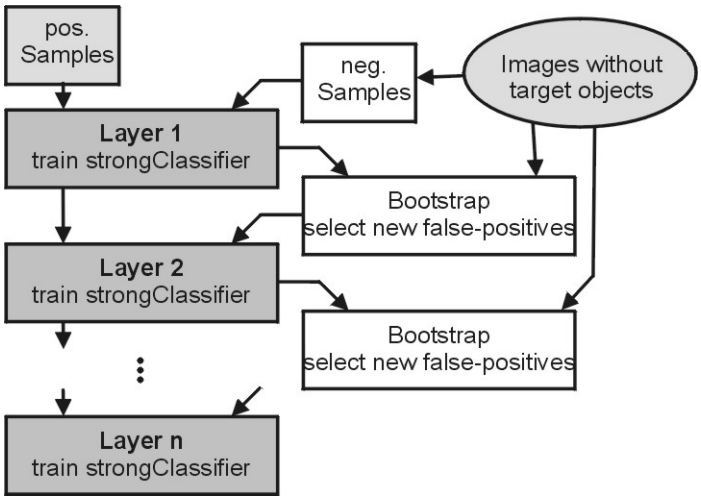

Abbildung 3.6: Schematische Beschreibung zum Trainieren eines Kaskaden Klassifikators. Die einzelnen starken Klassifikatoren werden sequentiell trainiert, wobei die zu erreichenden Performance Parameter und der negative Datensatz in jeder Kaskadenschicht angepasst werden.

mächtiger. Das Hinzufügen neuer Merkmale wird solange fortgeführt, bis sich auf einem unabhängigen Validierungsdatensatz die gewünschten Performancewerte eingestellt haben. Dies entspricht also der Bestimmung der verwendeten Hypothesenklasse. Wie aus Gleichung 2.15 zu erkennen ist steigt die Mächtigkeit (VC-Dimension) der Hypothesenklasse mit der Anzahl der verwendeten schwachen Klassifikatoren an.

Nachdem eine Kaskadenschicht erfolgreich trainiert wurde, wird mittels einer Bootstrap Technik aus der Datenbank von negativen Bildern eine neue Menge von negativen Beispielen generiert, die bis zu diesem Zeitpunkt noch als richtige Objekte (entspricht den false-positives) erkannt werden. Das Training der nächsten Kaskadenschicht wird nun mit den angepassten Parametern durchgeführt, bis ein globales Abbruchkriterium erreicht wird.

Ein so trainierter Kaskaden Klassifikator kann dann einen gegebenen Input $\mathbf{x}$ laut Algorithmus 3.5 auswerten. Wird dies für jedes mögliche Suchfenster in einem gegebenen Bild durchgeführt ergibt sich eine Detektionskaskade (*detection cascade*), die es ermöglicht das gewünschte Objekt in Position und Skalierung zu lokalisieren. Auf die genaue Erstellung wird im nächsten Kapitel eingegangen.

3.4 Nachverarbeitung

Der Klassifikator liefert auch bei leichten Veränderungen in Position und Skalierung noch immer ein positives Ergebnis. Somit kann es zu unerwünschten Mehrfachdetektionen (*multiple detections*) von ein und demselben Objekt kommen. Um eben diese Detektionen zu einer zusammenzufassen, bzw. auch um Ergebnisse des Klassifikators weiter zu verbessern wird ein Nachverarbeitungsschritt durchgeführt.

Algorithmus 3.4 trainieren eines Kaskaden Klassifikators $hCascade$

Require: positive P sowie negative N Trainingsbeispiele
Require: d, minimal akzeptierte Erkennungsrate pro Layer
Require: fp, maximal akzeptierte false-positive Rate pro Layer
Require: FP_{target}, gesamte zu erreichende false-positive Rate

- Initialisiere $FP_0 = 1 \qquad D_0 = 1 \qquad curLayer = 0$

while $FP_{curLayer} > FP_{target}$ oder weiteres Abbruchkriterium **do**

- $curLayer = curLayer + 1 \qquad FP_{curLayer} = FP_{curLayer-1}$
- generiere zufällige Menge J von Merkmalen
- $numFeatures = 0$

while $FP_{curLayer} > fp \cdot FP_{curLayer-1}$ **do**

- $numFeature = numFeature + 1$
- verwende $S = P \cup N$ und J um einen starken Klassifikator $hStrong_{curLayer}$ mit $numFeatures$ schwachen Klassifikatoren zu generiern

while $D_{curLayer} < d \cdot D_{curLayer-1}$ **do**

- evaluiere aktuellen Kaskaden Klassifikator $hStrong_{1,..,curLayer}$ auf einem Validierungsset um $FP_{curLayer}$ und $D_{curLayer}$ zu bestimmen.
- führe eine Schwellwertanpassung bei $hStrong_{curLayer}$ durch um $D_{curLayer}$ zu steigern (steigert auch $FP_{curLayer}$!)

end while
end while

- Bootstrap: evaluiere aktuellen Kaskaden Klassifikator $hStrong_{1,..,curLayer}$ auf einen Datensatz von Bilder die keine positiven Objekte beinhalten und bilde mit allen auftretenden false-positives einen neuen Datensatz N mit negativen Beispielen

end while

Algorithmus 3.5 evaluieren eines Kaskaden Klassifikators $hCascade$

Require: Kaskaden Klassifikator als Liste von starken Klassifikatoren $hStrong_{1,..,numLayer}$

for $curLayer = 1, .., numLayer$ **do**

if $hStrong_{curLayer}(\mathbf{x}) = -1$ **then**

- return -1

end if
end for

- return 1

In der Literatur werden dafür verschieden Ansätze verwendet. In [55] wird ein sehr einfaches Verfahren benutzt: Überlappen sich zwei oder mehrere Detektionen so werden diese zu einer zusammengefasst. Die Eckpunkte der finalen Detektion werden dabei aus den Mittelwerten der Eckpunkte von den einzelnen Detektionen berechnet. Ist ein Klassifikator auch in der Lage einen Wert für die „Sicherheit“ (Aktivierungswert) zusätzlich zu der jeweiligen Detektion zu liefern, kann dieser natürlich mit in die Berechnung einfließen. So kann eine *classifier activation map* erstellt werden. Negative Ergebnisse der Suchfensterevaluierung (keine Detektion) ergeben keinen Beitrag, positive Detektionen werden entsprechend des Aktivierungswertes gewichtet. Die so erstellte Matrix kann dann mit einem geeigneten Algorithmus analysiert werden um die einzelnen Detektionen zu liefern [2]. Der Algorithmus kann einerseits nur die besten Detektionen in einer gewissen Nachbarschaft zurückliefer, oder auch etwas komplizierter gestaltet sein.

Diese Idee wird in dem hier vorgestellten Nachverarbeitungsschritt verfolgt. Der, in den vorigem Abschnitt, vorgestellten Klassifikator liefert für jede Detektion auch einen skalaren Wert, den Margin, der für die Sicherheit verwendet werden kann. Mittels diesen wird eine Wahrscheinlichkeitsverteilung modelliert die mit dem nachfolgend beschriebenen Verfahren untersucht wird. Es handelt sich dabei um ein Verfahren aus dem unüberwachten Lernen. Der verwendete Algorithmus basiert auf dem Ansatz von *MeanShift*, einem relativ neuen (wieder neu entdeckten) *clustering* Ansatz. Gearbeitet wird auf einer Wahrscheinlichkeitsverteilung I, mit dem Ziel *modes* (Cluster, Verteilungszentren) zu finden, die den einzelnen Häufungen der Verteilung entsprechen. Die MeanShift Methode evaluiert den lokalen Gradienten der Verteilung und konvergiert, entlang der maximalen Gradienten, zum nächstliegenden Mode (*mode seeking*). Dabei wird eine Fenstergröße festgelegt, die sich während des gesamten Vorgangs nicht verändert, des Weiteren wird eine Ausgangsposition des Suchfensters $center_x, center_y$ gewählt. Innerhalb des Fensters wird mittels eines Kernelprofils das gewichtete Mittel über die Datenpunkte gebildet. Bei einem uniformen Kernelprofil entspricht das also der „mittleren Position“, an die das Suchfenster verschoben wird. Dieser Vorgang wird solange ausgeführt, bis eine Konvergenz zu einem Mode stattgefunden hat. Für die diskrete zweidimensionale Darstellung einer Wahrscheinlichkeitsverteilung, kann die mittlere Position über die ersten Momente in dem Suchfenster berechnet werden. Für weitere theoretische Betrachtungen sowie Anwendungen sei auf Comaniciu und Meer [14], [15] verwiesen.

Basierend auf diesen Ansatz wurde eine Erweiterung vorgenommen, die der Idee des *CAMShift* (von **c**ontinuously **a**daptive **m**ean shift) Ansatzes von Bradski [6] entspricht. Die Fenstergröße des MeanShift Algorithmus wird dabei adaptive angepasst mit dem Ziel einzelne Modes zu finden, die den Detektionen entsprechen. Bei der Anpassung der Fenstergröße können jedoch Probleme auftreten, die von Collins [13] beleuchtet wurden.

Die grundlegende Struktur des hier implementieren Verfahrens ist in Algorithmus 3.6 ersichtlich. Die Berechnung kann dabei sehr effizient (ebenfalls mit *Integral Image*) erfolgen und ist in [4] dargestellt. In diesem Artikel ist auch eine Erweiterung, *Model-based validation*, beschrieben, die jedoch in der aktuellen Version nicht eingesetzt wird. Es wäre interessant diese Auswirkung zu untersuchen, besonders für den Fall, dass sich zwei Detektionen teilweise überlappen.

Wie oben gezeigt arbeitet der MeanShift Algorithmus auf einer Wahrscheinlichkeitsverteilung I. Diese Verteilung wird aus allen Detektionen erstellt, die der Klassifikator liefert. An den Bereich der Detektion wurde eine 2D-Gausglockenkurve[3] mit der Ausdehnung des gefundenen Suchfensters gelegt. Der Maximalwert entspricht dabei dem Margin, also der Sicherheit dieser Detektion, der automatisch mit dem Klassifikationsergebnis mitgeliefert wird. Alle Detektionen werden additive überlagert und ergeben somit die benötigte Verteilung I.

Algorithmus 3.6 scale adaptive MeanShift

Require: Menge M von möglichen Detektionsergebnissen

- erstelle Verteilung I aufgrund von M
- finden lokaler Maxima in I
- normalisieren, sodass I einer Wahrscheinlichkeitsverteilung entspricht

for alle lokalen Maxima **do**

- initialisiere Suchfenster $center_x$, $center_y$, $width$ und $hight$

while nicht konvergiert **do**

- bestimme statistische Eigenschaften des Suchfensters

$$M_{00} = \sum_x \sum_y I(x,y) \qquad M_{01} = \sum_x \sum_y y \cdot I(x,y)$$
$$M_{10} = \sum_x \sum_y x \cdot I(x,y) \qquad M_{02} = \sum_x \sum_y y^2 \cdot I(x,y)$$
$$M_{20} = \sum_x \sum_y x^2 \cdot I(x,y) \qquad M_{11} = \sum_x \sum_y y \cdot x \cdot I(x,y)$$

- bestimme neues Zentrum (MeanShift)

$$center_x = \frac{M_{10}}{M_{00}} \qquad center_y = \frac{M_{01}}{M_{00}}$$

- bestimme neue Fenstergröße

$$a = \frac{M_{20}}{M_{00}} - center_x^2 \qquad c = \frac{M_{02}}{M_{00}} - center_y^2$$
$$b = 2 \cdot \left(\frac{M_{11}}{M_{00}} - center_x \cdot center_y \right)$$
$$width = 4 \cdot \sqrt{\frac{(a+c) - \sqrt{b^2 + (a-c)^2}}{2}}$$
$$height = width \cdot aspectRatio$$

end while

- speichere ermittelten Mode

end for

- ähnliche Modes verschmelzen

Man sollte die MeanShift Iterationen aus sinnvollen Punkten starten, das heißt in der Nähe von wahrscheinlichen Modes. Diese können durch lokale Intensitätsmaxima der Verteilung I angegeben werden. Die gefundenen lokalen Maxima ergeben also die jeweiligen Startpunkte $center_x, center_y$ für die den MeanShift mit adaptiver Fenstergröße. Die initiale Fenstergröße wird relativ gering (15% des zu findenden Objektes (*cluster*)) gewählt. Danach wird iterative die Position, mittels MeanShift, und die Fenstergröße angepasst. Der Vorgang wird solange wiederholt, bis eine Konvergenz zu einem Mode stattfindet, was typi-

[3]Die verwendete Varianz beeinflusst dabei wesentlich die gefundenen Modes und deren Anzahl, wenn sich die ursprünglichen Detektionen überlappen. Hier wurde $\sigma = 0.3$ verwendet.

scherweise in wenigen Iterationen der Fall ist. Für die Bestimmung der neuen Fenstergröße, wird eine Abschätzung der Haupt- und Nebenachse einer Ellipse verwendet, die die gleichen zweiten statistischen Momente besitzt wie die Region. Also eine lokale Abschätzung der Kovarianz der Verteilung unter der Annahme, dass sie in der engen Umgebung der aktuellen Position nahezu normalverteilt ist. Die Anpassung der Fenstergröße wird hierbei durch das erwartete Verhältnis zwischen der Höhe und Breite des Suchfensters ($aspectRatio$) eingeschränkt. Somit entfällt die explizite Berechnung der Höhe[4]. Dieses Verfahren wird für alle Startpunkte (lokale Maxima) durchgeführt und im Anschluss werden die so ermittelten Modes gegebenenfalls miteinander kombiniert (verschmolzen).

[4]Die allgemeine Berechnung ist $height = 4 \cdot \sqrt{\frac{(a+c)+\sqrt{b^2+(a-c)^2}}{2}}$, sowie der Winkel $\phi = \frac{1}{2} \cdot \arctan(\frac{b}{a-c})$

Kapitel 4

Autodetektion

In diesem Kapitel wird eine konkrete Anwendung des in Kapitel 3 entwickelten Klassifikators beschrieben um damit ein Detektionssystem zu erstellen. Es werden dabei die seitliche und frontal/heck Ansichten von Autos behandelt. Anhand dieser Beispiele soll die Vorgehensweise erläutert und die gewünschten Eigenschaften veranschaulicht werden. Es können natürlich weitere Klassifikatoren für andere Ansichten beziehungsweise für andere Objekte trainiert werden. Die einzige Forderung ist, dass ein genügend mächtiger Datensatz von positiven Beispielen gegeben ist.

Der Aufbau dieses Kapitels hält sich im Wesentlichen an die einzelnen Schritte des Modells für ein Objektdetektionssystems aus Abschnitt 2.2.1. Es wird zunächst die Trainingsphase beleuchtet und danach anhand eines konkreten Beispielbildes die Testphase erklärt. Im Anschluss sind weitere Ergebnisse und Vergleiche zu anderen Methoden die verwendet werden aufgeführt. Ein kurzer Abschnitt am Ende fasst einige Schlussbemerkungen aus praktischer Sicht zusammen.

4.1 Trainingsphase

4.1.1 Trainingsdaten sammeln

Als erster Punkt steht die Datenaufnahme und deren Aufbereitung (Vorverarbeitung) für den anschließenden Lernprozess. Die positiven Beispiele (Bilder) wurden aus verschiedenen im Internet zugänglichen Datensätzen zusammengesucht. Für die seitliche Ansicht wurden der *UIUC Car Database* [1] Datensatz verwendet. Für die frontale/heck Ansicht wurde eine Kombination von den Datensätzen *Caltech Cars (Rear) dataset* [2], *Caltech Cars (Rear) 2 dataset* [2] und *CBCL Car Database #1*[3] erstellt. Die einzelnen Bilder wurden auf eine

[1] `http://l2r.cs.uiuc.edu/ cogcomp/Data/Car/` (21.06.2004)

[2] `http://www.robots.ox.ac.uk/ vgg/` (21.06.2004)

[3] `http://www.ai.mit.edu/projects/cbcl/software-datasets/CarData.html` (21.06.2004)

einheitliche Größe skaliert und eine Normalisierung des Histogramms vorgenommen, um unterschiedliche Beleuchtungseffekte weitgehend auszugleichen.

Die Anzahl der negativen Beispiele wurde dreifach so groß gewählt wie die Anzahl der positiven Trainingsbeispiele. Die negativen Beispiele der Datensätze wurde durch zufällige Auswahl von Teilbildern (Position und Auflösung) aus einer Bilddatenbank von über 2000 beliebigen Bildern, in denen kein zu detektierendes Objekt vorkommt, generiert. Zum Erstellen dieser Datenbank wurde ebenfalls auf Datensätze aus dem Internet zurückgegriffen: *UIUC Car Database* [1], *Caltech Cars (Rear) background dataset* [2] und *Caltech Cars background dataset* [2].

4.1.2 Parameter des Klassifikators trainieren

Für jeden der gegebenen Datensätze wurde ein Kaskaden Klassifikator separat trainiert. Anhand des Detektors für die Erkennung seitlicher Autos wird die Vorgehensweise und der Lernvorgang genauer erläutert. Bei dem Heck/Front Detektor wurde hierauf verzichtet und nur mehr die Ergebnisse dokumentiert. Die verwendeten Parameter sind in Tabelle 4.1 zusammengefasst.

	Seite	Heck/Front
positive Trainingsbeispiele	550	950
Aufteilung in Trainings-, Val.Datensatz	6:4	6:4
Größe der Bilder	50x20	25x20
mögliche Merkmale	522 064	129 420
verwendete Merkmale	5000 ($\approx 1\%$)	2500 ($\approx 2\%$)
Detektionsrate pro Layer d_i	0.99	0.99
false-positive Rate pro Layer fp_i	0.4	0.4
false positive Rate FP_{target}	10^{-4}	10^{-4}

Tabelle 4.1: Parameter für das Trainieren des jeweiligen Kaskaden Klassifikators

Seite

Zu Beginn des Trainings wurden die gegebenen Daten aufgeteilt in einen Trainings-, Validierungs- und Testdatensatz. Die durchschnittlichen Bilder sind in der Abbildung 4.1 zu sehen, wobei auf die unterschiedliche Skalierung zu achten ist. Eine typische Auswahl von positiven Beispielen ist auf der rechten Seite der Abbildung ersichtlich.

Das eigentliche Trainieren der schwachen Klassifikatoren wurde mittels des Trainingsdatensatzes durchgeführt. Der davon unabhängige Validierungsdatensatz wurde verwendet um die Hypothesenklasse festzulegen, das heißt für die Entscheidung ob noch weitere Merkmale (schwache Klassifikatoren) mittels *Boosting* zu dem starken Klassifikator hinzugefügt

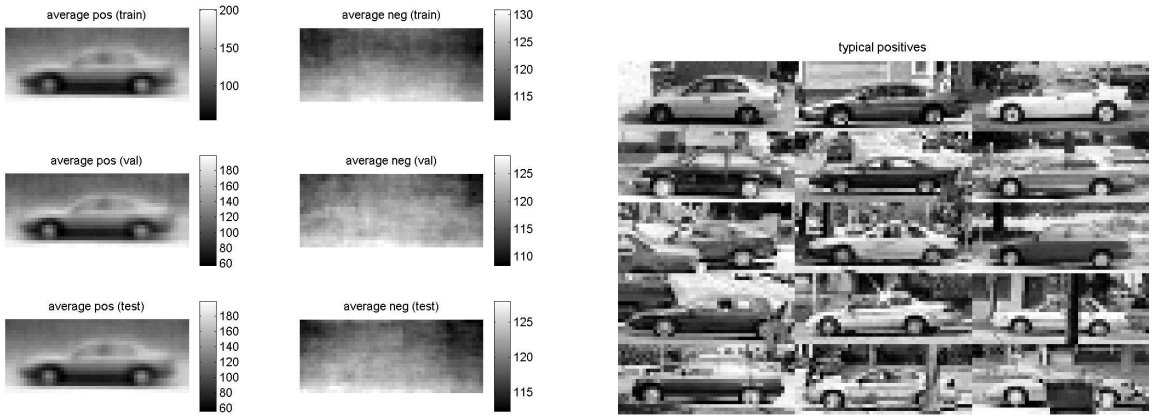

Abbildung 4.1: *Seite*-Datensatz: (links) Trainings-, Validierungs- und Testdatensätze und (rechts) typische positive Beispiele

werden sollten. Der Testdatensatz wurde für die Berechnungen der Performancewerte verwendet. Da nur beschränkte Daten (positive Beispiele) vorhanden waren wurde die Aufteilung in Trainings- und Validierungsdatensatz im Verhältnis 60:40 aufgeteilt. Für den Testdatensatz wurden alle positiven Beispiele verwendet[4] und eine große Anzahl von negativen Beispielen (um die Detektion zu „simulieren“).

Mit Hilfe dieser Datensätze wurde das Training des Kaskaden Klassifikators gestartet. Im Folgenden ist ein Auszug aus den Debug-Meldungen des Matlab Programmes während des Trainings gegeben (vergleiche hierzu auch die schematische Darstellung aus Abbildung 3.6 sowie den Pseudocode aus Algorithmus 3.4).

```
25-Jul-2004 11:21:39  START training Side
25-Jul-2004 11:21:39  generate data sets
25-Jul-2004 11:25:36  generate feature set (5000 Features)
25-Jul-2004 11:25:45  CASCADE LAYER 1
25-Jul-2004 11:25:45  training weak learner
25-Jul-2004 11:29:05    10%
25-Jul-2004 11:32:17    20%
25-Jul-2004 11:35:34    30%
25-Jul-2004 11:38:51    40%
25-Jul-2004 11:42:07    50%
25-Jul-2004 11:45:22    60%
25-Jul-2004 11:48:35    70%
25-Jul-2004 11:51:52    80%
25-Jul-2004 11:55:08    90%
25-Jul-2004 11:58:21    100%
25-Jul-2004 11:58:24  weak classifier error: min=0.06861; max=0.49978; average=0.37153
25-Jul-2004 11:58:24  boosting with 1 Features
25-Jul-2004 11:58:25    Feature: 564; type=2 (error= 0.07251)
25-Jul-2004 11:58:42    ValSet: corr=0.14286; f=1.00000; d=1.00000; threshold=-2.17828
25-Jul-2004 11:58:42  boosting with 2 Features
25-Jul-2004 11:58:42    Feature: 2127; type=1 (error= 0.12679)
```

[4]Um ein seriöses Ergebnis zu erhalten müssten diese Daten unabhängig von den ersten beiden Datensätzen sein. Es wurden jedoch in Hinblick auf die Anwendung möglichst viele Daten für das Trainieren verwendet. Die Resultate auf Testbilder wurden (siehe Abschnitt 4.3) für den eigentlichen „Test“ herangezogen.

```
25-Jul-2004 11:58:58    ValSet: corr=0.14286; f=1.00000; d=1.00000; threshold=-3.95384
25-Jul-2004 11:58:58  boosting with 3 Features
25-Jul-2004 11:58:59    Feature: 768; type=2 (error= 0.13417)
25-Jul-2004 11:59:23    ValSet: corr=0.14286; f=1.00000; d=1.00000; threshold=-5.42575
25-Jul-2004 11:59:23  boosting with 4 Features
25-Jul-2004 11:59:24    Feature: 3278; type=3 (error= 0.22188)
25-Jul-2004 11:59:46    ValSet: corr=0.64188; f=0.41705; d=0.99543; threshold=-4.27894
25-Jul-2004 11:59:46  boosting with 5 Features
25-Jul-2004 11:59:46    Feature: 961; type=4 (error= 0.10161)
25-Jul-2004 12:00:23    ValSet: corr=0.64188; f=0.41705; d=0.99543; threshold=-5.37064
25-Jul-2004 12:00:23  boosting with 6 Features
25-Jul-2004 12:00:24    Feature: 355; type=3 (error= 0.15154)
25-Jul-2004 12:00:41    ValSet: corr=0.77952; f=0.25571; d=0.99087; threshold=-5.33586
25-Jul-2004 12:00:41  CASCADE LAYER 1 finished

25-Jul-2004 12:00:41  Goal not hit...
25-Jul-2004 12:00:44  select pos samples 329 / 329 (start)
25-Jul-2004 12:04:30  select new false positives 968 / 1974  (f=0.24519)
25-Jul-2004 12:07:53  select new false positives 1954 / 1974  (f=0.24975)
25-Jul-2004 12:11:16  select new false positives 1974 / 1974  (f=0.23987)
25-Jul-2004 12:11:16  generate feature set (5000 Features)
25-Jul-2004 12:11:25  CASCADE LAYER 2
25-Jul-2004 12:11:25  training weak learner
25-Jul-2004 12:14:44    10%
25-Jul-2004 12:17:57    20%

...

25-Jul-2004 21:24:33  boosting with 39 Features
25-Jul-2004 21:24:34    Feature: 2349; type=3 (error= 0.44527)
25-Jul-2004 21:24:34  boosting with 40 Features
25-Jul-2004 21:24:34    Feature: 2841; type=5 (error= 0.48659)
25-Jul-2004 21:25:32    ValSet: corr=0.99152; f=0.00152; d=0.94977; threshold=-5.05023
25-Jul-2004 21:25:32  CASCADE LAYER 6 finished

25-Jul-2004 21:25:32  Goal not hit...
25-Jul-2004 21:26:54  select pos samples 324 / 329 (start)
25-Jul-2004 21:30:42  select new false positives 18 / 1944  (f=0.00463)
25-Jul-2004 21:34:06  select new false positives 34 / 1944  (f=0.00412)

...

26-Jul-2004 04:14:21  select new false positives 1927 / 1944  (f=0.00412)
26-Jul-2004 04:17:45  select new false positives 1940 / 1944  (f=0.00334)
26-Jul-2004 04:21:07  select new false positives 1944 / 1944  (f=0.00180)
26-Jul-2004 04:21:07  generate feature set (5000 Features)
26-Jul-2004 04:21:16  CASCADE LAYER 7
26-Jul-2004 04:21:16  training weak learner
26-Jul-2004 04:24:26    10%
26-Jul-2004 04:27:38    20%
26-Jul-2004 04:30:51    30%
26-Jul-2004 04:34:03    40%
26-Jul-2004 04:37:16    50%
26-Jul-2004 04:40:28    60%
26-Jul-2004 04:43:40    70%
26-Jul-2004 04:46:54    80%
26-Jul-2004 04:50:05    90%
26-Jul-2004 04:53:18    100%
26-Jul-2004 04:53:21  weak classifier error: min=0.23545; max=0.50000; average=0.45765
26-Jul-2004 04:53:21  boosting with 1 Features
26-Jul-2004 04:53:21    Feature: 24; type=5 (error= 0.26720)
26-Jul-2004 04:53:21  boosting with 2 Features
26-Jul-2004 04:53:22    Feature: 2379; type=2 (error= 0.27028)
26-Jul-2004 04:53:22  boosting with 3 Features
```

...

```
26-Jul-2004 05:44:25  boosting with 348 Features
26-Jul-2004 05:44:26    Feature: 563; type=4 (error= 0.24074)
26-Jul-2004 05:44:26  boosting with 349 Features
26-Jul-2004 05:44:27    Feature: 383; type=6 (error= 0.47134)
26-Jul-2004 05:44:27  boosting with 350 Features
26-Jul-2004 05:44:28    Feature: 3447; type=2 (error= 0.40961)
26-Jul-2004 05:55:32    ValSet: corr=0.99152; f=0.00000; d=0.94064; threshold=-13.76149
26-Jul-2004 05:55:32  CASCADE LAYER 7 finished
```

Trainiert wurde auf einem 1 GHz Intel Pentium III Prozessor mit 256 MByte Arbeitsspeicher. Wie aus den angegebenen Zeitstempeln ersichtlich, dauerte das Trainings etwa 20 Stunden. Sehr viel Zeit geht dabei in die Bootstrapping Prozedur, um die neuen false-positives für das Lernen der nächsten Kaskadenschicht zu generieren. Dies tritt vor allem auf, wenn schon ein guter Klassifikator besteht, der somit eine kleine false-positive Rate hat. Als Ergebnis ergab sich eine Kaskade der Tiefe sieben mit 6, 8, 30, 40, 40, 40, 350 Merkmalen (schwachen Klassifikatoren) der starken Klassifikatoren in der jeweiligen Kaskadenschicht.

Die Performance ist der ROC Kurve in Abbildung 4.2 zu entnehmen. Um die ROC Kurve erstellen zu können, wurde der Schwellwert $\theta Strong$ der letzten Kaskadenschicht in gewissen Grenzen verändert. Bei einem Schwellwert von $+\infty$ wird sowohl die Detektionsrate als auch die false-positive Rate Null. Geht Schwellwert gegen $-\infty$ steigen beide Raten an. Sie können jedoch auf Grund der Kaskadenstruktur nicht höher werden als in den vorangegangenen Schichten. Eine Senkung des Schwellwertes auf $-\infty$ entspricht also der Entfernung der letzten Kaskadenschicht. Wenn nur mehr die erste Schicht vorhanden ist und der Schwellwert bei dieser $-\infty$ ist, ergibt sich eine Detektionsrate und false-positive Rate von jeweils 100%. Somit sind beide Extreme dargestellt. Das Interessante ist jedoch der nichtlineare Anstieg der Kurve, also eine hohe Detektionsrate obwohl die false-positive Rate noch gering ist.

Um eine komplette Kurve zu erzeugen, müssten also sukzessive Kaskadenschichten entfernt werden. Darauf wurde hierbei verzichtet und nur das interessante Gebiet um den Anstieg des finalen Kaskaden Klassifikators dargestellt (rechte Seite der Abbildungen). Um dennoch ein Gefühl für die Steigerung der Performance mit den einzelnen Kaskadenschichten zu geben, sind auf der linken Seite der Abbildung die Kurven abhängig von der verwendeten Kaskadenschicht zusammengefasst.

Wie sich die Leistungsfähigkeit über die Anzahl der verwendeten Kaskadenschichten verändert ist in Abbildung 4.3 ersichtlich. Etwa 75% aller Inputbilder werden bereits in der ersten Kaskadenschicht verworfen und somit der vorhandene Suchraum drastisch eingeschränkt. Es werden jedoch (fast) keine richtigen Beispiele entfernt, wodurch die Detektionsrate sinken könnte.

Abbildung 4.4 zeigt die ersten vier Merkmale des ersten Layers - also die wichtigsten. Wie man erkennt werden jene Merkmale verwendet die sich auf die Konturen des Objektes und auf die spezifischen Eigenschaften der Objekte (Schatten unter dem Auto) beziehen. Auf der rechten Seite der Abbildung sind die jeweils zugehörigen Verteilungen

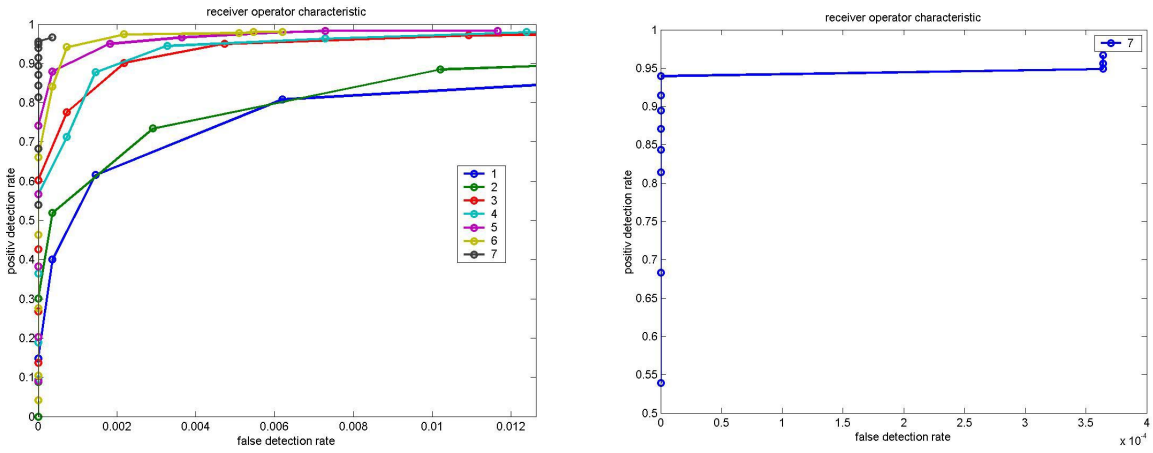

Abbildung 4.2: Seite-Datensatz - ROC: Verlauf der Performance abhängig von den gewählten Kaskadenschichten (links) bzw. für den gesamten Kaskaden Klassifikator (rechts). Wie gewünscht, tritt eine stetige Verbesserung, eine höhere Detektionsrate bei kleinerer false-positive Rate, ein.

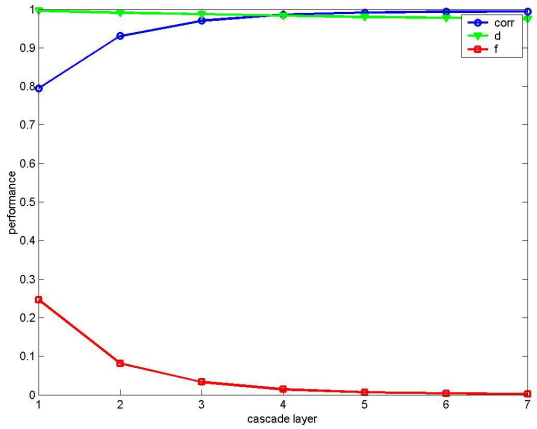

Abbildung 4.3: Seite-Datensatz: Performance abhängig von den gewählten Kaskadenschichten. In den ersten Schichten werden nahezu alle Objekte erkannt (Detektionsrate d), jedoch auch noch relativ viele zu unrecht (false-positive Rate f). Desto mehr Schichten hinzugefügt werden desto geringer wird die false-positive Rate obwohl die Detektionsrate hoch bleibt. Die spiegelt sich auch in der Kurve für alle korrekt klassifizierten Beispiele (corr) wieder.

der Merkmalswerte aufgetragen, die ja zum Trainieren der schwachen Klassifikatoren benutzt wurden. Weiters ist dabei der verwendete Schwellwert eingezeichnet und die damit ermittelte Performance. Zu Beginn des Prozesses sind die Fehlerraten noch relativ gering, in einem Bereich zwischen 0.1 und 0.3. Merkmale, die erst später hinzugefügt werden da die Aufgabe schwieriger wird, erreichen Fehlerraten von 0.4 bis 0.5.

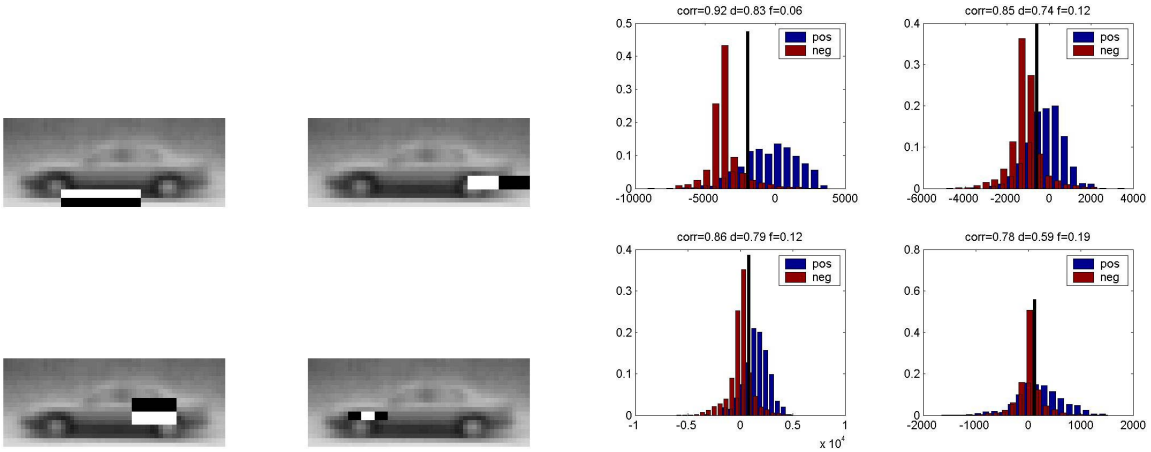

Abbildung 4.4: Ersten vier (most dominant) verwendeten Merkmale des starken Klassifikators in Kaskadenschicht 1 (links) und die dazugehörigen Verteilungen der Merkmalswerte von den positiven und negativen Trainingsbeispielen (rechts). Die schwarze Balken kennzeichnen die ermittelten Schwellwerte und die damit erreichten Performancewerte sind in den Überschriften der jeweiligen Abbildungen angeführt.

Die Verteilung der starken Klassifikatoren sind für die Schicht 1 und 7 (letzte) in Abbildung 4.5 dargestellt. Sie entsprechen den Margin Werten. Die Lernaufgabe wird für jeden Layer immer schwieriger, da immer bessere Detektionsraten und false-positive Raten erreicht werden müssen. Ein weiterer Grund ist, dass die „einfachen “ negativen Beispiele bereits von den vorherigen Kaskadenschichten entfernt werden und sich der Lernalgorithmus daher auf die „schweren“ fokussieren muss. Die ist auch durch die Wahl von sehr kleinen Merkmale in den höheren Kaskadenschichten belegt.

Heck/Front

Anhand anderer positiver Beispiele wurde ein weiterer Klassifikator für die Detektion von heck/front Ansichten von Autos trainiert. Typische Beispiele, sowie die gemittelten Bilder der Datensätze sind in Abbildung 4.6 dargestellt.

Es ergaben sich hierbei ähnliche Ergebnisse, die zusammengefasst als ROC Kurve in Abbildung 4.7 dargestellt ist. Analog zu den Abbildungen für die seitliche Ansicht, sind in den Abbildungen 4.8 bis 4.10 detaillierte Ergebnisse angeführt. Als Ergebnis ergab sich eine Kaskade der Tiefe sechs mit 3, 5, 25, 25, 200, 400 Merkmalen. In den ersten Schichten werden etwa gleich viele Merkmale wie bei der seitlichen Detektion verwendet, die die

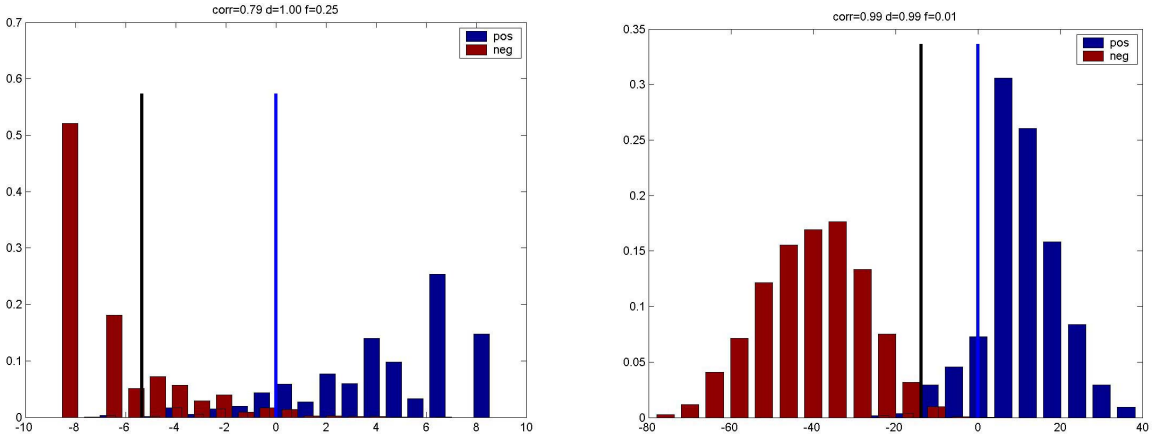

Abbildung 4.5: Kombinierter starker Klassifikator in Schicht 1 (links) und Schicht 7 (rechts). Der blaue Balken kennzeichnet jeweils den Schwellwert bei Null und der schwarze den angepassten Schwellwert, um die Detektionsrate zu steigern. In den Überschriften der Abbildungen sind die entsprechenden Performancewerte angeführt. Die Werte auf der Abszisse entsprechen betragsmäßig den Margin Werten, also der Sicherheit.

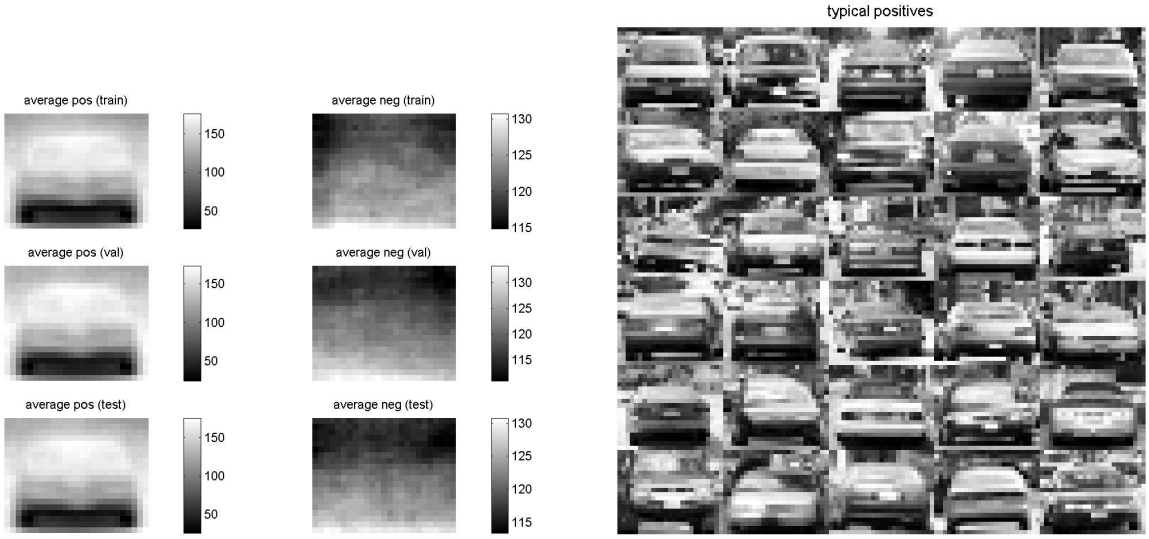

Abbildung 4.6: *Heck*-Datensatz: (links) Trainings-, Validierungs- und Testdatensätze und (rechts) typische positive Beispiele

grundliegende Struktur erfassen. Um die gewünschte Performance zu erreichen müssen in den höheren Schichten jedoch sehr viele detaillierte Merkmale verwendet werden. Somit stellt sich dieses Problem als „schwieriger“ heraus. Dies ist auch leicht einzusehen, da das Objekt kleiner ist und daher nicht so viele Merkmale auftreten, die es ermöglichen es von allem Anderen zu unterscheiden. Hierbei konzentrieren sich die Merkmale meist auf horizontale und seitliche Begrenzungen. Bei der seitlichen Ansicht bieten sich zusätzlich auch die Regionen um die Räder an.

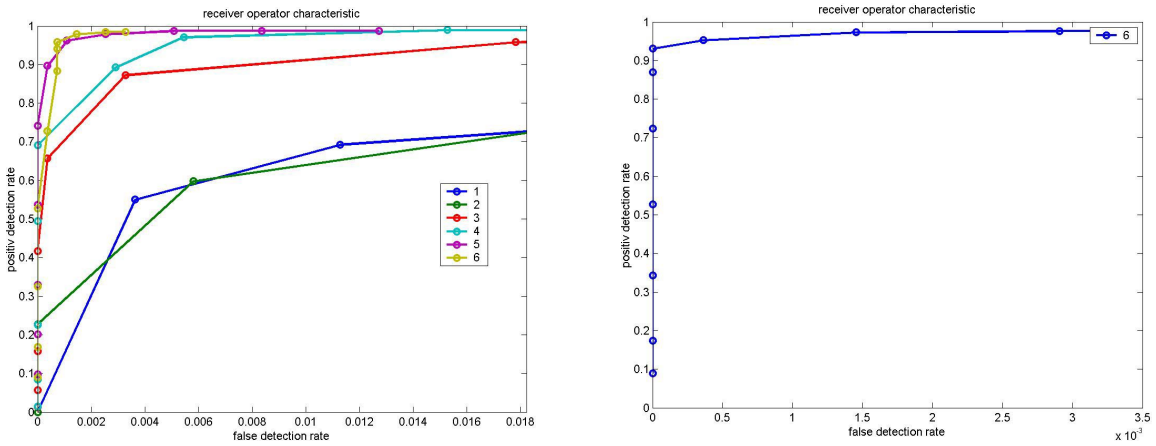

Abbildung 4.7: Heck-Datensatz - ROC: Verlauf der Performance abhängig von den gewählten Kaskadenschichten (links) bzw. für den gesamten Kaskaden Klassifikator (rechts). (vergleiche auch Abbildung 4.2)

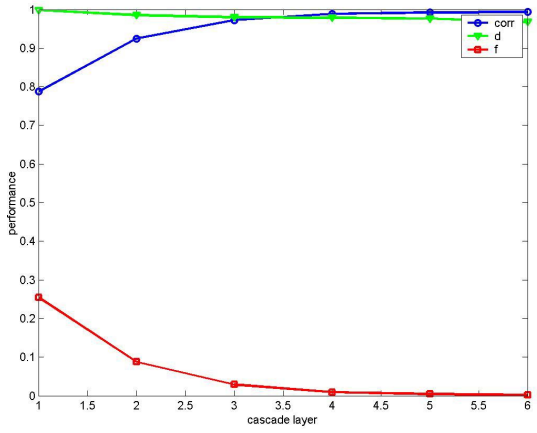

Abbildung 4.8: Heck-Datensatz: Performance abhängig von den gewählten Kaskadenschichten. (vergleiche auch Abbildung 4.3)

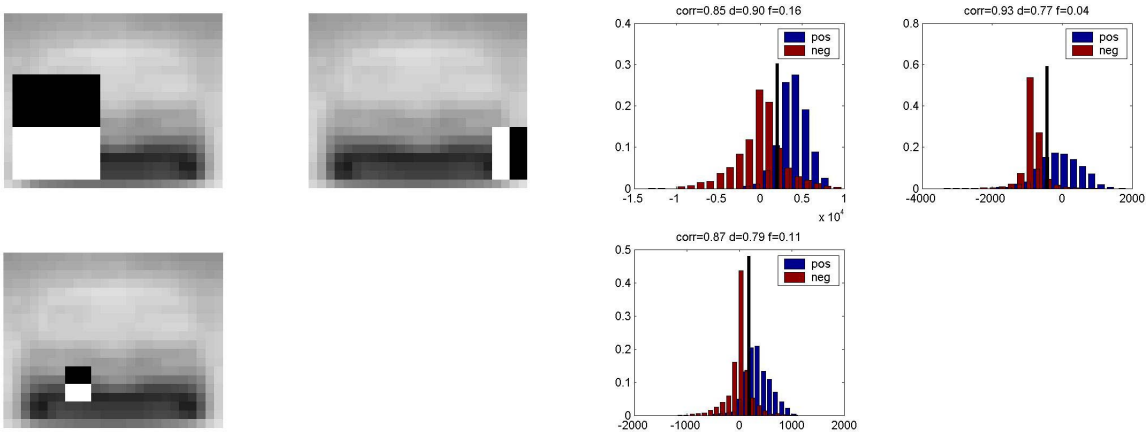

Abbildung 4.9: Heck-Datensatz: Ersten vier (most dominant) verwendeten Merkmale des starken Klassifikators in Kaskadenschicht 1 (links) und die dazugehörigen Verteilungen der Merkmalswerte von den positiven und negativen Trainingsbeispielen (rechts). (vergleiche auch Abbildung 4.4)

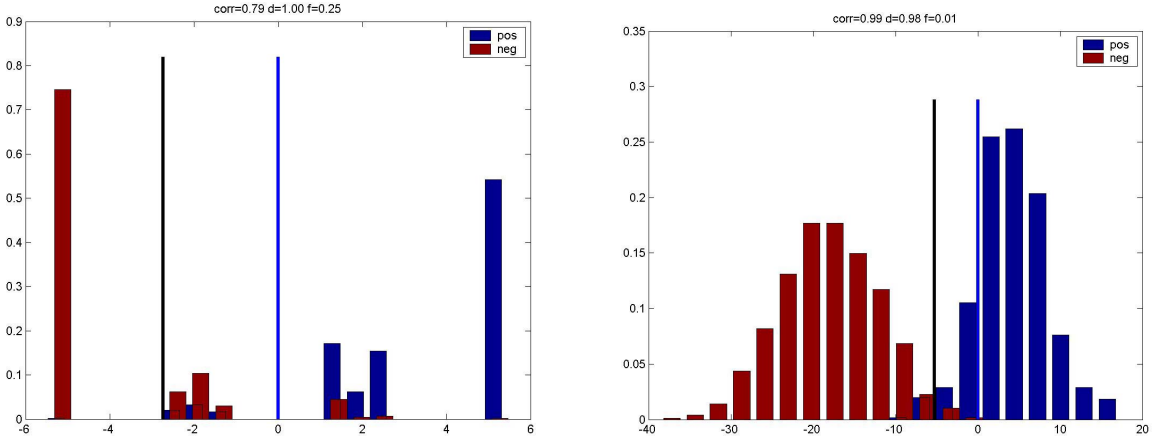

Abbildung 4.10: Heck-Datensatz: Kombinierter starker Klassifikator in Schicht 1 (links) und Schicht 6 (rechts). (vergleiche auch Abbildung 4.5)

4.2 Testphase

In diesem Abschnitt wird beschrieben, wie anhand der trainierten Klassifikatoren das Detektionssystem ein Bild verarbeitet. Die einzelnen Schritte werden am Beispiel eines Testbildes erläutert. Auf die Punkte Datenaufnahme und Merkmale extrahieren wird nicht näher eingegangen. Bei den einzelnen Testbildern wurde keine Vorverarbeitung durchgeführt (siehe dazu auch Abschnitt 4.4).

In Abbildung 4.11 ist das gewählte Testbild dargestellt. Weitere Ergebnisse sind in den nachfolgenden Abschnitten angeführt.

Abbildung 4.11: Testbild für die Demonstration der einzelnen Schritte in der Testphase.

4.2.1 Klassifikation

Es wird nun laut dem Suchansatz in Position und Skalierung bezüglich jedes Objektes durchsucht und der erstellte Klassifikator evaluiert. Jedoch wird nicht an jeder möglichen Position bzw. bei jeder Skalierung gesucht. Die grundlegende Annahme ist es, dass der Klassifikator auch dann ein positives Ergebnis für ein Suchfenster liefert, wenn es in Skalierung und Position etwas von dem wirklichem Objekt abweicht.

Vom Benutzer kann eine Startskalierung *scaleStart*, eine Endskalierung *scaleEnd* und ein Faktor zur Anpassung *scaleFactor* angegeben werden. Somit ergeben sich $numScales = \lfloor \frac{\log(scaleEnd/scaleStart)}{\log(scaleFactor)} \rfloor$ Skalierungen bei denen das Bild durchsucht wird. Die jeweils aktuelle Skalierung ergibt aus $scaleStart \cdot scaleFactor^i$ wobei $i = 0, .., numScales$ ist. Des weiteren wird nicht an jeder Pixelposition gesucht, sondern das Suchfenster in x und y Richtung jeweils um die Konstante Δ_{xy} verschoben.

In der hier vorgestellten *Standard-Methode* sind die Werte $scaleStart = 1$, $scaleEnd = 3$, $scaleFactor = 1.4$. Die Verschiebung Δ_{xy} beträgt 10% der gesamten Fenstergröße bei der aktuellen Skalierung. Bei der seitlichen Detektion, mit einer Basisfenstergröße von 50x20, und Skalierung 1 ergibt sich somit eine Verschiebung um je 5 Pixel in x und 2 Pixel in y-Richtung. Wie sich die unterschiedliche Wahl dieser Parameter auf das Detektionsergebnis

auswirken wird im Anschluss in Abschnitt 4.3.2 beleuchtet. Es ist vor allem Wichtig hier Vorwissen einzubringen.

Für das gegebene Testbild sind diese Detektionsergebnisse auf der linken Seite der Abbildung 4.12 zu sehen. Rechts sind die jeweils vier „besten“ Detektionen dargestellt. Die jeweiligen Werte im Titel entsprechen der Sicherheit (Margin) der Detektion.

Abbildung 4.12: (links) Alle Detektionen in Skalierung, Position und Art des Objektes; (rechts) besten Ergebnisse (höchste Sicherheit)

4.2.2 Nachverarbeitung

Wie zu erkennen liefert der Klassifikator an verschieden stellen eine Detektion. Um all diese Detektionen zusammenzufassen wird laut dem, in Abschnitt 3.4, beschriebenen Verfahren vorgegangen.

Abbildung 4.13 zeigt die Wahrscheinlichkeitsverteilung I für die Detektionen des seitlichen Detektors. Das, laut dem MeanShift Algorithmus mit adaptiver Fenstergröße, gelieferte Ergebnis ist in Abbildung 4.14 dargestellt. Die roten Punkte entsprechen den gefundenen lokalen Maxima und somit den Startwerten der Suchfenster, die gelbe Ellipse stellt den ermittelten Mode dar.

Dieses Verfahren wurde auch auf die Detektionen des front/heck Detektors angewendet. In diesem Fall wurden zwei Modes ermittelt. Das vorläufige Ergebnis ist auf der linken Seite der Abbildung 4.15 dargestellt. Wie zu erkennen ist, gibt es dabei einen Konflikt, da ein Bereich bei beiden Detektionen vorkommt. Dies liegt daran, dass bei dieser Anwendung zwei voneinander unabhängige Detektionssysteme entwickelt wurden.

Mittels einer einfachen Heuristik, also einem weiteren Nachverabreitungsschritt, können diese zwei Verfahren kombiniert werden. Die Annahme ist, dass nicht zwei Ansichten an ein und derselben Stelle auftreten können. Überlappungen der Detektionen um mehr als 33% wurden eliminiert und der Vorzug wird dabei jener Detektion gegeben, die einen höheren Wert (Wahrscheinlichkeit) besitzt. Die endgültigen Ergebnisse sind auf der rechten Seite

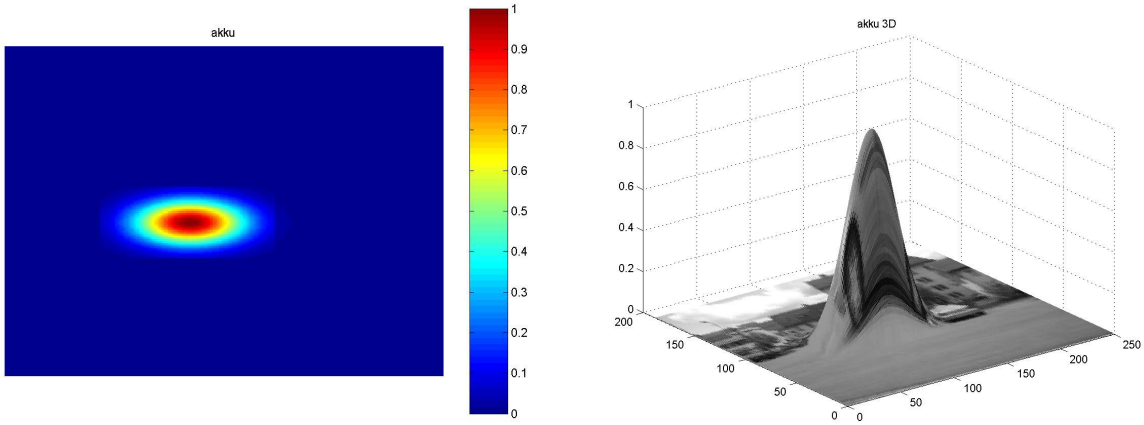

Abbildung 4.13: Wahrscheinlichkeitsverteilungen I für den scale adaptive MeanShift (seitlicher Detektor)

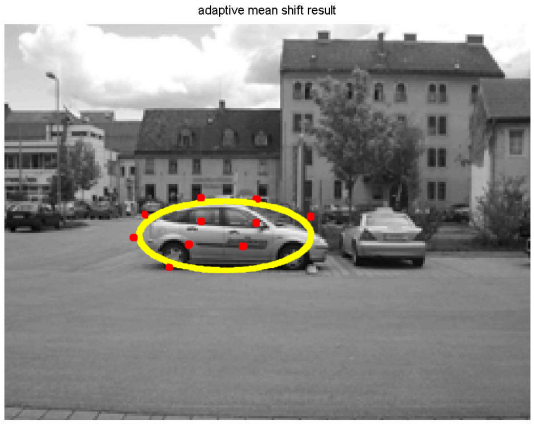

Abbildung 4.14: Gefundener Mode der Wahrscheinlichkeitsverteilung I durch den scale adaptive MeanShift Algorithmus (seitlicher Detektor)

der Abbildung 4.15 dargestellt. Beide in dem Bild vorkommenden Objekte werden dabei sehr gut erfasst.

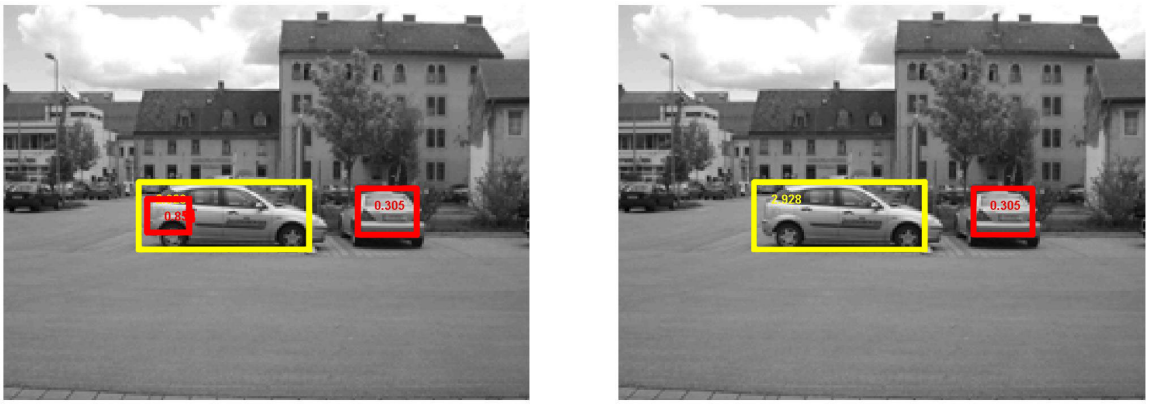

Abbildung 4.15: Ergebnisse des Testbildes; (links) nach scale adaptive MeanShift; (rechts) endgültiges Ergebnis nach weiterer Nachverarbeitung. Der Zahlenwert in den farbigen Rechtecken entspricht dabei der Sicherheit dieser Detektion.

4.2.3 Anmerkungen

Im Folgenden sind die Debug-Meldungen des Matlab Programmes während der Evaluierung dieses Bildes gegeben. Wie zu erkennen, wird für jedes Objekt (*Side*, *Heck*) bei bestimmten Skalierungen gesucht. Die Anzahl der Detektionen vor und nach dem Nachverarbeitungsschritt, sowie einige statistische Werte sind daraus zu entnehmen.

```
START
file: Camera0001
detect: Side
  on scale 1.00  # detection = 0
  on scale 1.40  # detection = 0
  on scale 1.96  # detection = 2
  on scale 2.74  # detection = 0
detect: Heck
  on scale 1.00  # detection = 2
  on scale 1.40  # detection = 1
  on scale 1.96  # detection = 1
  on scale 2.74  # detection = 0

STATISTICS
object: Side
  subwindows         6303
    average features 7.45
  detections         2
    final            1
      center (101, 106), dimension (32, 81)
  time               18.50
object: Heck
  subwindows         17324
    average features 4.83
  detections         4
```

```
   final            1
     center (99, 183), dimension (22, 28)
time                30.88
```

Die benötigte Rechenzeit scheint sehr hoch zu sein. Dies liegt jedoch an der ineffizienten Implementierung in Matlab. Wie aus der Literatur ersichtlich, ermöglicht dieser Ansatz eine schnelle und effektive Implementierung und ist dabei durchaus echtzeitfähig (real-time) [55], [34].

Um dennoch eine Abschätzung der Rechenzeit zu geben, wurden die durchschnittlichen Merkmale, die pro Suchfenster evaluiert werden müssen, herangezogen. In Abbildung 4.16 sind die Verteilungen dargestellt in welcher Schicht die Entscheidung fällt, ob es sich um ein Objekt handelt oder nicht. Es werden jeweils an die 90% der Suchfenster bereits in der ersten Kaskadenschicht verworfen. Diese Entscheidung wird nur anhand von sechs (Seite) und drei (Heck/Front) verwendeten Merkmalen getroffen. Die durchschnittlich benötigten Merkmale pro Suchfenster sind bei diesem Testbild nur 7.4 (Seite) und 4.9 (Heck/Front). Die Anzahl der Suchfenster für ein Bild ist abhängig von der Bildgröße und wie genau das Bild durchsucht wird. Durch Multiplikation der möglichen Suchfenster mit den durchschnittlichen Merkmale[5] ergibt sich der Rechenaufwand. Auf dieses wird in den nächsten Abschnitt genauer eingegangen.

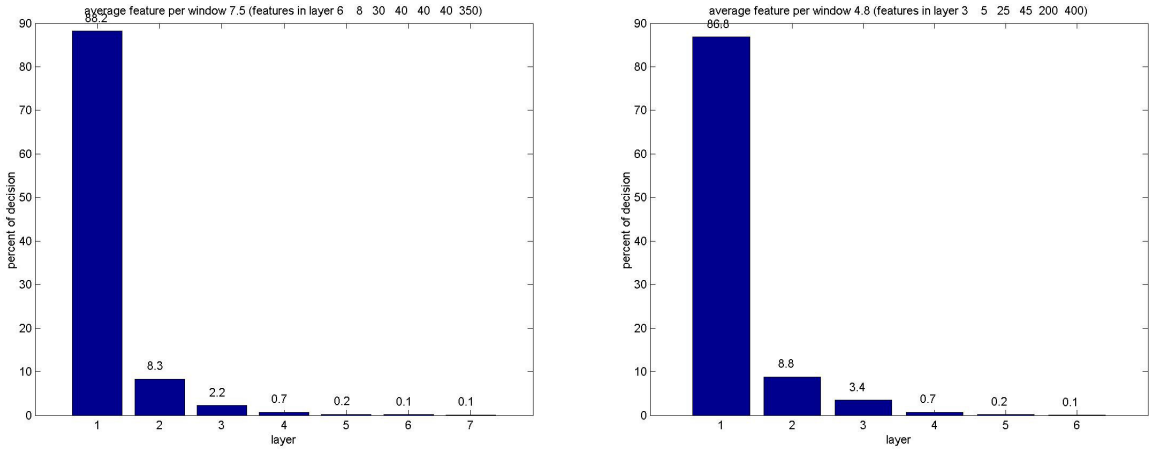

Abbildung 4.16: Verteilung in welcher Kaskadenschicht die Entscheidung des aktuellen Suchfensters getroffen wird (links) Seite (rechts) Heck.

4.3 Ergebnisse

Zum Einen wurden selbst aufgenommene Bilder verwendet um praktisch, ohne statistische Auswertung, die Leistungsfähigkeit und die noch bestehenden Probleme beider Detektoren und deren Kombination zu veranschaulichen. Zum Anderen wurde auf einen allgemein

[5] Wie in Abschnitt 3.2 dargestellt, kann die Berechnung der einzelnen Merkmale in konstanter Zeit, sehr schnell, erfolgen.

zugänglichen Testdatensatz die Performance ermittelt um einen Vergleich zu anderen Ergebnissen, in der Literatur, herzustellen. Dabei werden jedoch nur die seitlichen Detektionen berücksichtigt.

4.3.1 Praktischer Test

In der Abbildung 4.17 sind einige Ergebnisse dargestellt. Die Bilder stellen typische Straßenszenen dar und sind mittels einer Digitalkamera aufgenommen worden. Wie in dem oben gezeigten Beispiel, wurde keine Vorverarbeitung durchgeführt und die Parameter für die Suche entsprechen der Standard Methode.

Sowohl die seitlichen als auch die front/heck Ansichten werden dabei gut detektiert. Die letzten beiden Bilder zeigen Beispiele bei denen der Detektor noch Probleme hat. Vor allem sollte die Kombination der unterschiedlichen Detektion noch verbessert werden.

4.3.2 UIUC Testdatensatz I

Um die von diesem Detektionssystem erreichte Performance, mit anderen vergleichen zu können wurde der UIUC-Testdatensatz[6] für seitliche Autodetektion verwendet. Es handelt sich dabei um 170 Testbilder, die 200 Autos in seitlicher Ansicht beinhalten. Alle Autos haben etwa die gleiche Größe. Die enthaltenen Autos sind teilweise durch andere Objekte verdeckt oder besitzen geringen Kontrast zu dem Hintergrund. Da es sich dabei um reale Straßenszenen handelt ist auch meist ein sehr texturierter Hintergrund vorhanden.

Die Ergebnisse, Performancewerte von *recall*, *precision* und *F-measure*, sind in Tabelle 4.2 zusammengefasst. Die Definitionen finden sich in Abschnitt 2.2.3. Es handelt sich dabei jeweils um einen Punkt in der RPC-Kurve. Um eine komplette Charakteristik des Systems zu erstellen, müssten die Parameter des Klassifikators und/oder die des Nachverarbeitungsschrittes angepasst werden. Für die 10. Serie sind in den Abbildungen 4.18 und 4.19 jeweils ein paar positive bzw. negative Testergebnisse aufgezeigt.

Wie in Abschnitt 4.2.1 erläutert, wird nicht an jeder möglichen Position bzw. Skalierung gesucht, sondern nur an stark diskretisierten Punkten. Serie 1 stellt das oben vorstellte Standard Verfahren dar. Die Ergebnisse können jedoch wesentlich verbessert werden. Es stellt sich heraus, dass in diesem Testdatensatz nur Autos mit der Skalierung von etwa 2 (doppelt so groß wie der trainierte Klassifikator) vorkommen. Serie 2 sucht nur bei dieser Skalierung und verschiebt dabei das Suchfenster um $\Delta_{xy} = 10\%$ der Fenstergröße (entspricht 10 Pixel in x und 5 Pixel in y-Richtung). Die erreichten Werte sind nun jedoch noch schlechter. Wird nun die Auflösung der Position erhöht, eine Verschiebung um nur mehr $\Delta_{xy} = 5\%$, so ergibt sich eine wesentliche Steigerung (Serie 3). Wird das Verfahren nun auch noch bei mehreren Skalierungen angewendet ergibt sich eine weitere Steigerung, wie Serie 4 und 5 belegen. Das Detektionssystem liefert also um so bessere Ergebnisse,

[6] `http://l2r.cs.uiuc.edu/ cogcomp/Data/Car/` (25.09.2004)

Abbildung 4.17: praktische Testergebnisse

Serie	Beschreibung	detect.	false-pos.	recall	precision	F-measure
1	Standard	174	27	87.0 %	86.6 %	86.8%
2	scale = 2, $\Delta_{xy} = 10\%$	164	36	82.0 %	82.0 %	82.0%
3	scale = 2, $\Delta_{xy} = 5\%$	180	10	90.0 %	94.7 %	92.3%
4	scale = 1.8 - 2.2, $\Delta_{xy} = 10\%$	180	11	90.0 %	94.2 %	92.1%
5	scale = 1.8 - 2.2, $\Delta_{xy} = 5\%$	189	7	94.5 %	96.4 %	95.4%
6	wobble-rotation, $\Delta_{xy} = 10\%$	174	13	87.0 %	93.0 %	89.9%
7	wobble-scale, $\Delta_{xy} = 10\%$	182	8	91.0 %	95.2 %	93.1%
8	wobble-scale, $\Delta_{xy} = 5\%$	190	3	95.0 %	98.4 %	96.7%
9	wobble, $\Delta_{xy} = 10\%$	189	6	94.5 %	96.9 %	95.2%
10	wobble, $\Delta_{xy} = 5\%$	190	5	95.0 %	97.4 %	96.2%

Tabelle 4.2: UIUC Testdatensatz I: Ergebnisse mit unterschiedlichen Einstellungen

je „feiner“ das Suchfenster in Position und Skalierung über das gegebene Bild geschoben wird.

Bei den Serien 6 bis 10 wurde diese Feststellung ausgenutzt und führt zu einem Ansatz der als *wobble* bezeichnet wird. Dabei handelt es sich um eine zufällige affine Transformation[7], eine zufällige Verzerrung, des gegeben Bildes. Bei den Verzerrungen wurde die Position nicht und die anderen Parameter in geringen Maße verändert (etwa ±5 Grad Drehung und eine unabhängige Skalierung der Breite und Höhe von etwa 5%). Es wurden je 4 Instanzen der Bilder erzeugt und das Detektionssystem darauf angewendet. Vor allem die unterschiedliche und unabhängige Skalierung in x und y-Richtung scheinen die Ergebnisse zu verbessern. Die verschiedenen Detektionen werden durch die Kombination im Nachverarbeitungsschritt zusammengefasst. Es ist zu erkennen, dass sich mehrere Detektionen, vor allem an einer Stelle an der sich ein Objekt (Auto) befindet, erfolgen. Einzelne falsche Detektionen (false-positives) mit geringer Sicherheit (Margin-Wert) liefern bei der Erstellung der Wahrscheinlichkeitsverteilung I in der Nachverarbeitung nur einen geringen Betrag und es wird wahrscheinlich an dieser Stelle kein Mode erkannt (senken der false-positive Rate). Tritt jedoch bei einer Instanz des verzerrten Bildes eine Detektion mit hoher Sicherheit auf, wird diese sehr wohl berücksichtigt (steigern der Detektionsrate). Das gleiche gilt für viele Detektionen an einer Stelle (Unterschied in Position und Skalierung) mit relativ geringer Sicherheit, da sich diese ja additive überlagern. Desto mehr Instanzen (Verzerrungen) erzeugt werden, desto stabiler werden die Ergebnisse, da die Nachverarbeitung mit mehr statistischer Information arbeiten kann. Der Nachteil, den man sich jedoch dabei einhandelt, ist, dass die Berechnungszeit ansteigt. Es müssen nicht nur mehr Suchfenster vom Kaskaden Klassifikator evaluiert werden, sondern auch die Anzahl der durchschnittlich zu evaluierenden Merkmale steigt an. Dies liegt daran, dass mehr Detektionen auftreten, bei denen die Entscheidung erst in der letzten Schicht des Kaskaden Klassifikators getroffen

[7]Eine Transformation mit sechs Freiheitsgraden, die eine Ebene auf eine andere Ebene abbilden kann (Verschiebung und Skalierung in x und y-Richtung, Scherung und Drehung)

wird. So werden in Serie 2 etwa 13 Merkmale pro Suchfenster evaluiert, bei Serie 3 hingegen schon etwa 17.

Zusammenfassend kann also gesagt werden, dass ein Kompromiss zwischen Rechenaufwand und den erzielten Ergebnissen bzw. dem abgedeckten Größenbereich zu ziehen ist. Steht viel Rechenleistung zur Verfügung oder ist die Aufgabe nicht zeitkritisch (nicht real-time) können bessere Ergebnisse erzielt werden.

Ein Vergleich mit anderen Ansätzen, die ebenfalls auf diesen Datensatz evaluiert wurden, ist in Tabelle 4.3 gegeben[8]. Die Evaluierung kann mittels der wahren Positionen und einem kleinen Programm, das auf der angegeben Internetseite vorhanden ist, automatisch durchgeführt werden. Das hier vorgestellte Verfahren liefert im Vergleich also sehr gute Ergebnisse.

Nr.	Methode	recall	precision	F-measure	Anmerkungen
1	Agarwal, Roth (Auszug) [2]	90.5% $\sim$ 79% 70.0%	64.9% $\sim$ 80 82.8%	75.6% 79.5% 75.9%	(*)
2	Agarwal et. al (Auszug) [1]	72.5%	81.5%	76.7%	
3	Fergus et al. [19]	88.5%	$\sim$ 80%	84.0%	Vergleich zu (*)
4	Garg et al. [24]	94.0%	75.5%	83.7%	
5	Schneiderman [44]	97.0%	$\sim$ 80%	87.7%	Vergleich zu (*)
6	Leibe et al. [27]	97.5%	97.5%	97.5%	
7	diese	95.0%	97.4%	96.2%	

Tabelle 4.3: UIUC-Datensatzes I: Vergleich der Ergebnisse mit anderen Ansätzen aus der Literatur

4.3.3 UIUC Testdatensatz II

Die Eigenschaften dieses Testdatensatzes entsprechen im Wesentlichen dem UCUI Testdatensatz I, mit der Ausnahme, dass die darin vorhandenen Autos unterschiedliche Größe besitzen. Es sind 139 Autos in 108 Bildern enthalten.

Das erzielte Ergebnis, sowie der Vergleich zu einer anderen Methode ist auszugsweise in Tabelle 4.4 dargestellt. Auch hierbei ergibt sich eine wesentliche Steigerung aller Performancewerte. Bei der hier vorgestellten Methode wurde bei sechs Skalierungen (1.4 bis 3.48, jeweils um den Faktor 1.2 erhöht) das Suchfenster um $\Delta_{xy} = 5\%$ über das gegebene Bild verschoben. Einige positive bzw. negative Testbilder sind den Abbildungen 4.20 und 4.21 zu entnehmen. Durch „gezielteres“ Suchen bzw. mit Hilfe des *wobble* Ansatzes könnten wahrscheinlich noch bessere Ergebnisse erzielt werden.

[8]Die unterschiedlichen Ergebnisse zwischen Methode Nummer 1 und 2 sind dadurch begründet, dass die Evaluierung mit einem neuen strengeren (genaueren) Kriterium durchgeführt wurde.

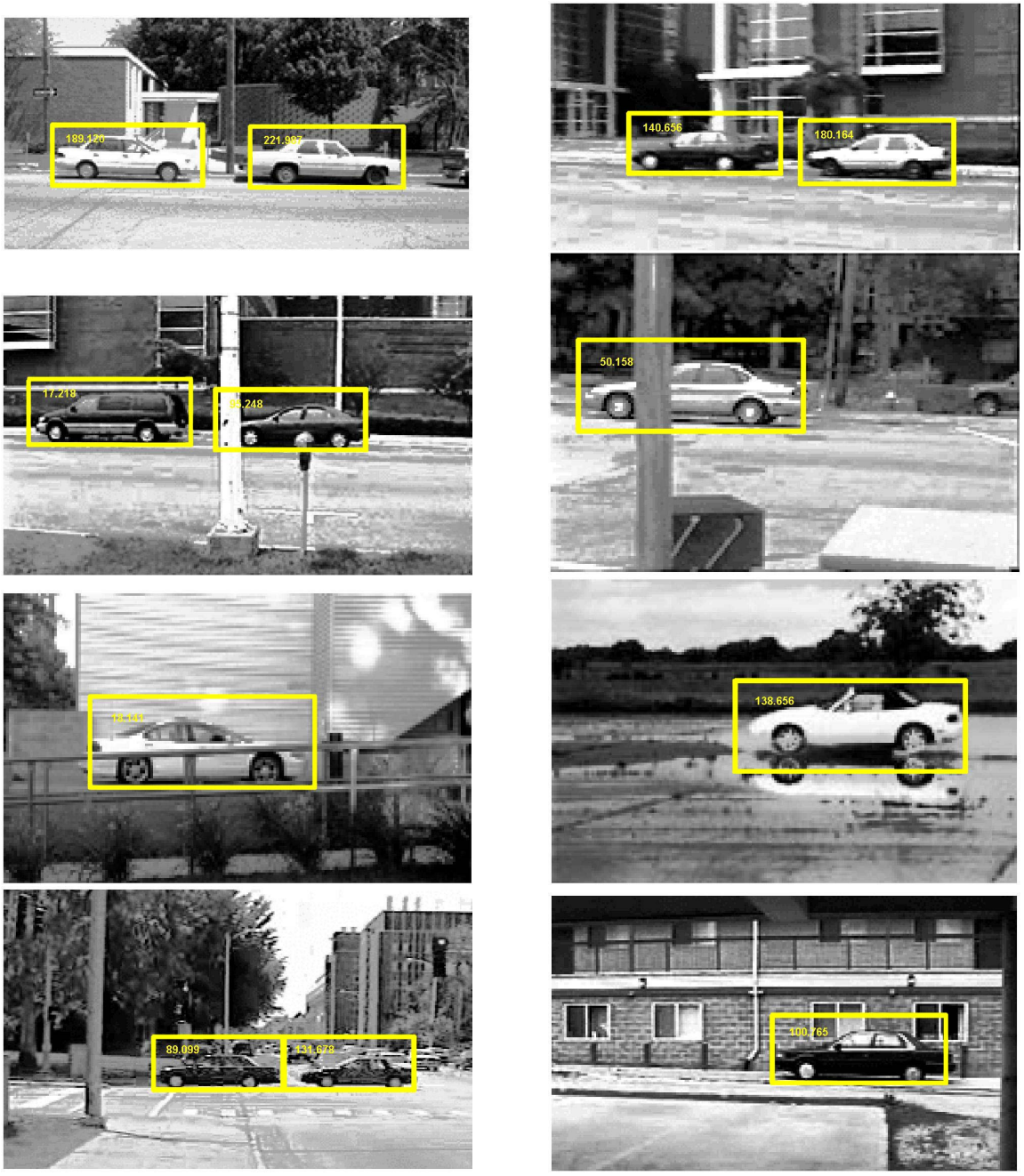

Abbildung 4.18: UIUC Testdatensatz I: positive Testergebnisse

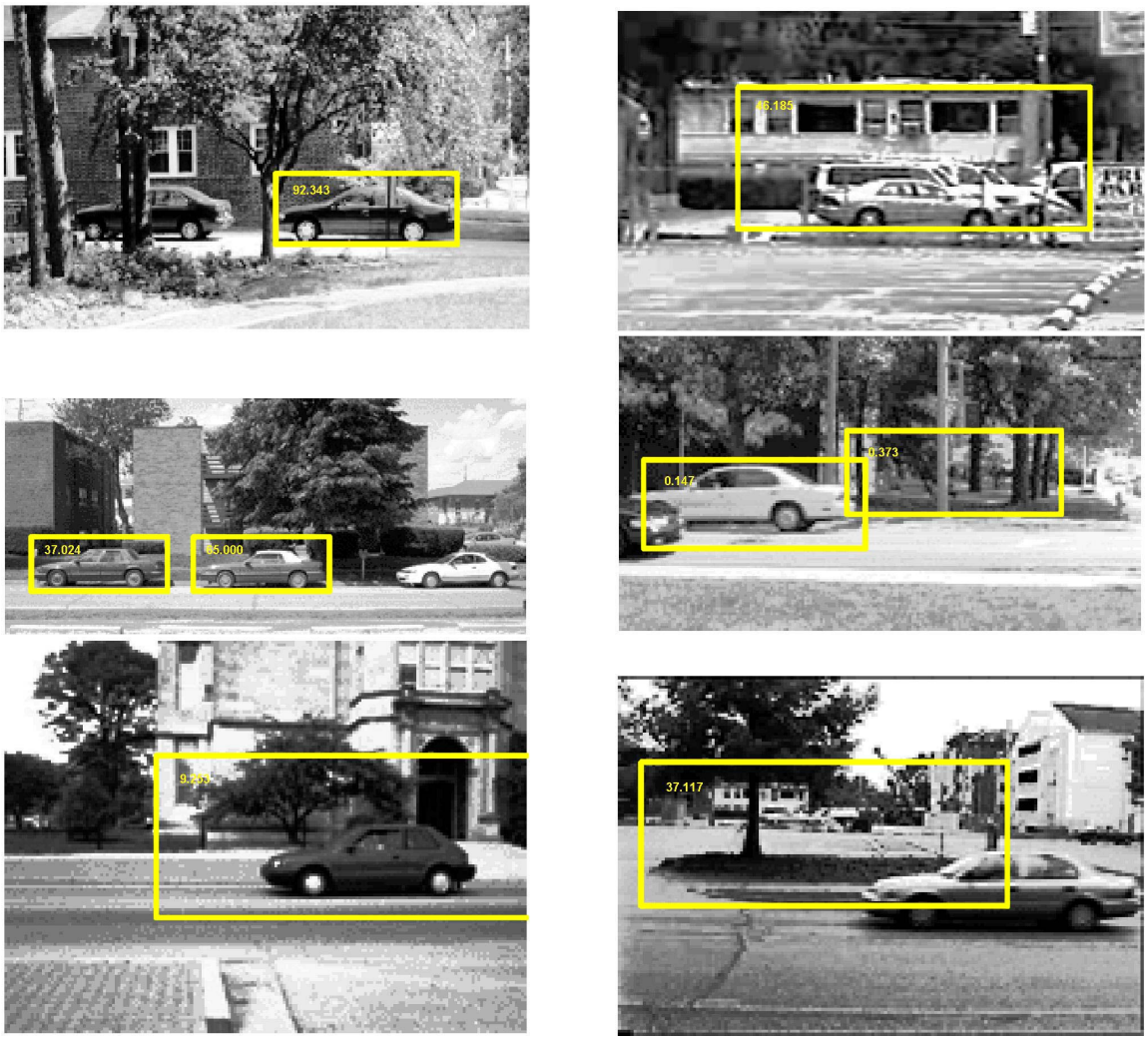

Abbildung 4.19: UIUC Testdatensatz I: negative Testergebnisse

Nr	Methode	detections	false-pos.	recall	precision	F-measure
1	Agarwal et. al (Auszug)[1]	112	1216	80.6%	8.4%	15.3%
		55	56	39.6%	49.6%	44.0%
		17	5	12.2%	77.3%	21.2%
2	diese	122	32	87.8%	79.2%	83.3%

Tabelle 4.4: UIUC-Datensatzes II: Vergleich der Ergebnisse mit anderen Ansätzen aus der Literatur

Abbildung 4.20: UIUC Testdatensatz II: positive Testergebnisse

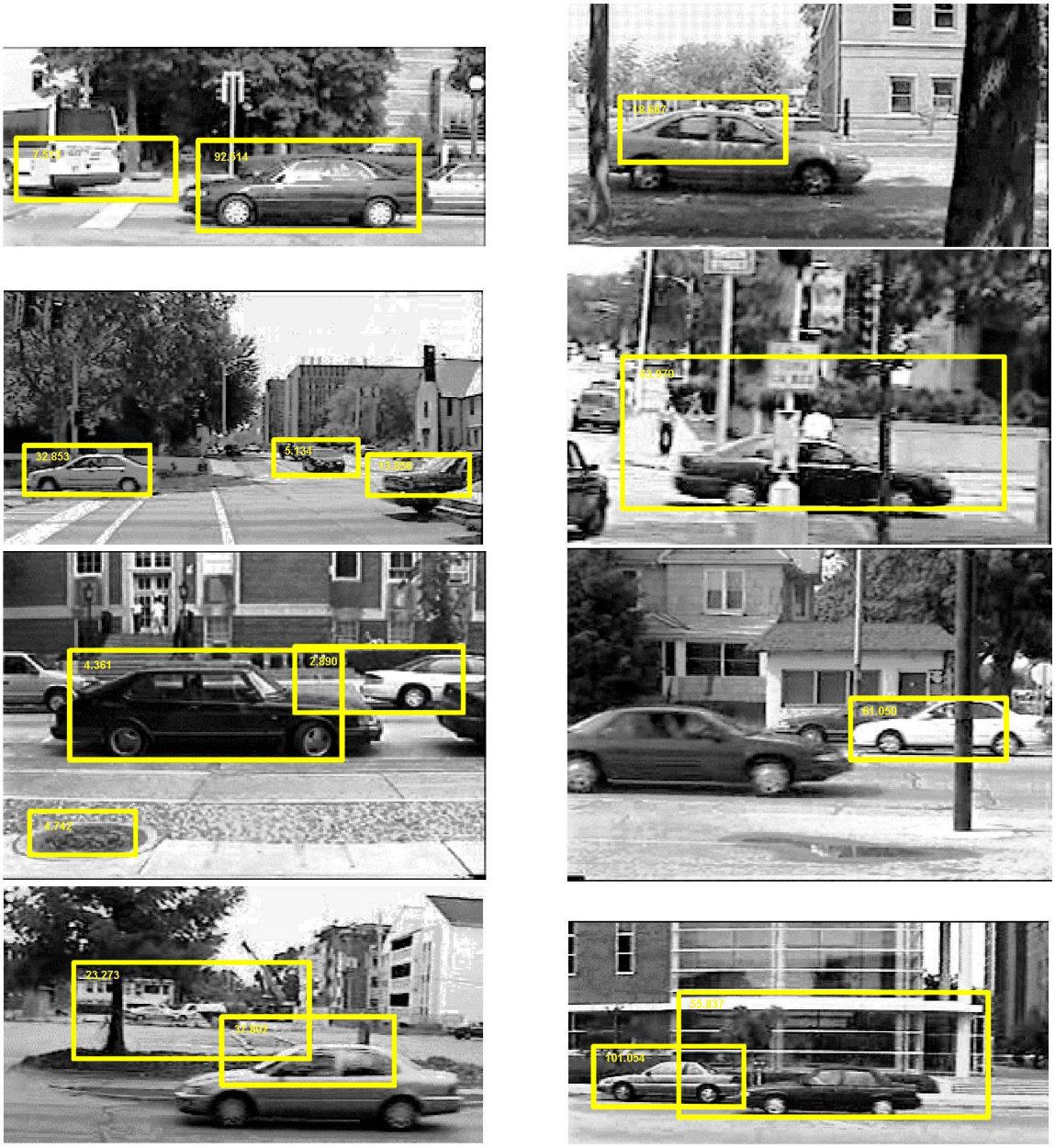

Abbildung 4.21: UIUC Testdatensatz II: negative Testergebnisse

4.4 Schlussfolgerungen

In der Folge sind einige Anmerkungen gegeben die während der Implementierung des Objektdetektionssystems und dessen Anwendung gemacht wurden. Sie sollen einerseits Probleme aufzeigen und andererseits Hilfe für weitere Entwicklungen geben.

- Es ist sinnvoll die positiven Beispiele nicht zu „knapp“ auszuschneiden sondern auch noch Hintergrund zuzulassen, da ansonsten die Konturen von den einzelnen Prototypen der Merkmale nicht gut erfasst werden können.

- Es ist nicht nur bei der Auswahl der positiven Beispiele besonders Wert zu legen, sondern auch auf die negativen Beispiele, bzw. der Datenbank aus denen diese Beispiele, durch zufällige automatische Auswahl, generiert werden.

  Dies ist zum Beispiel bei dem UIUC Testdatensatz II zu beobachten. Der Datensatz enthält allgemeinere Straßenszenen als UIUC Testdatensatz I, da hier auch Teile von anderen Autos vorhanden sind. Es wäre daher sinnvoll auch Bilder mit solchen, „abgeschnittenen“ Autoteilen in die Menge der negativen Trainingsbeispiele aufzunehmen.

- Bei realen Testbilder wurde keine Normalisierung (Vorverarbeitung) durchgeführt, da sich daraus eine Vielzahl von false-positiven einstellte[9]. Dies liegt daran, dass bei relativ gleich verteilten Flächen (Bäume, Straßen) im Wesentlichen das darin enthaltene Rauschen als Objekt detektiert wird. Obwohl das Trainieren mit einem normalisierten Datensatz durchgeführt wurde, sind die Ergebnisse nicht schlechter. Das liegt daran, dass zwar die Sicherheit (Margin) von Detektionen nicht so groß ist, diese aber immer noch als solche erkannt werden.

- Die Überlappungen (doppelten Erkennungen) des Seiten- und des Heck/Front-Detektors sind wahrscheinlich darauf zurückzuführen, dass beide Objekte wesentlich durch horizontale Kanten definiert sind. So tritt zum Beispiel in beiden Fällen ein dunkler Bereich am unteren Rand auf, der dem Schatten entspricht.

- Sind zwei oder mehr Objekte in einem Bild enthalten und wird eines davon sehr viel besser erkannt (viel größere Margin Wert) als alle anderen, so wird in den meisten Fällen nur dieses eine Objekt detektiert. Das Problem liegt an dem Nachverarbeitungsschritt und kann in gewissen Rahmen mittels des *wobble* Ansatzes verringert werden. Ist auf der anderen Seite kein Objekt enthalten, so kann durch zu „genaues

[9] Eine Varianznormalisierung könnte jedoch sehr effizient erfolgen [55]. Die Varianz eines Suchfensters kann durch $\sigma^2 = \frac{1}{N}\sum_{i,j} x^2 - \left(\frac{1}{N}\sum_{i,j} x_{i,j}\right)^2$ berechnet werden. Dabei stellt der zweite Term den quadrierten Mittelwert über die N Anzahl der Pixel mit dem Wert $x_{i,j}$ in dem Suchfenster dar. Der Mittelwert kann leicht mittels des bereits berechneten Integral Image ermittelt werden. Der Mittelwert von der Summe der quadrierten Pixelwerte kann ebenfalls mit einem Integral Image, dass aus dem quadrierten Bild erstellt wurde, berechnet werden. Es werden also zwei Integral Image benötigt. Die Normalisierung muss übrigens nicht auf Pixelebene stattfinden, sondern kann durch eine Multiplikation des Merkmalswertes erfolgen.

Suchen“, das heißt bei mehreren Skalierungen und vielen Positionen, dennoch eine Detektion erfolgen. Da in diesem Fall viel mehr Suchfenster evaluiert werden und die Wahrscheinlichkeit eines false-positives erhöht wird. Das Objekt wird zwar meist nur mit einer relativ geringen Sicherheit (Margin), aber eben doch, erkannt. Durch weiteres Einbringen von Vorwissen bzw. der Kombination von verschiedenen Verfahren in dem Nachverarbeitungsschritt könnten diese Probleme eventuell in den Griff zu bekommen sein.

Kapitel 5

Zusammenfassung

Es wurde anhand von Autos gezeigt, dass ein robustes und schnelles Objektdetektionssystem erstellt werden konnte. Zunächst wurde ein Klassifikator, laut dem Ansatz von Viola und Jones, erstellt, der das Kernstück des Systems darstellt. Als ersten Schritt werden dazu einfache schwache Klassifikatoren trainiert, die auf den einzelnen Merkmalen basieren. Die Merkmale lehnen sich dabei an eine Wavelet Transformation an. Mittels des Algorithmus *AdaBoost* wird eine kleine Menge dieser Merkmale ausgewählt um das Objekt zu beschreiben. Die Evaluierung wurde weiter beschleunigt indem mehrere dieser Klassifikatoren erstellt und in einer Kaskadenstruktur zusammengefasst wurden.

Dieser Kaskaden Klassifikator bildet den Hauptbestandteil des Objektdetektionssystems, mit denen beliebige Objekte in einem Bild detektiert werden können. Für die Detektion wurde das Bild schrittweise mittels eines Suchfensters in Position und Skalierung durchsucht. Es ergibt sich eine relative geringe Anzahl der durchschnittlich zu berechneten Merkmale je Suchfenster. Der dadurch erreicht Geschwindigkeitsvorteil kann dazu genutzt werden eine genauere Suche durchzuführen. So führt der vorgestellte wobble-Ansatz die Detektion wiederholt auf leicht verzerrte Instanzen des ursprünglichen Bildes aus. Die verschiedenen auftretenden Detektionen werden danach in einem Nachverarbeitungsschritt miteinander kombiniert. Es handelt sich dabei um Variante des auf MeanShift basierenden clustering-Verfahrens mit adaptiver Fenstergröße. Aufgrund dieses Nachverarbeitung in Zusammenarbeit mit dem wobble-Ansatz konnte eine Steigerung der Leistung des gesamten Systems erreicht werden.

Das Verfahren wurde anhand von zwei konkreten Beispielen (seitliche und frontal/heck Ansichten von Autos) getestet und evaluiert. Wie der Vergleich mit anderen Ergebnissen aus der Literatur zeigt, stellen sich sehr gute Performance Werte (hohe Detektionsrate bei gleichzeitig niedriger false-positives Rate) ein.

5.1 Ausblick

Nachfolgend sind ein paar Punkte gegeben die einen Ausblick und mögliche Anknüpfungspunkte für Verbesserungen geben.

- Es wäre wünschenswert eine schnelle Implementierung der Testphase und eventuell auch des Trainings zu haben. Dies würde es ermöglichen eine genauere Analyse des benötigten Rechenaufwandes und der zu erreichenden Performance zu erstellen. Auch könnten damit schneller weitere Tests auf anderen Datensätzen, sowie bei realen Anwendungen durchgeführt werden. Dabei sollte auch die Auswirkung des wobble-Ansatzes genauer untersucht und mit anderen Ansätzen verglichen werden.

- In der Literatur wurden auch mögliche Erweiterungen für die Erstellung des Klassifikators aufgezeigt, um zum Beispiel die Anzahl der benötigten Merkmale weiter zu verkleinern [57]. Auch könnten andere Merkmale eingesetzt werden, andere Lernmethoden für die schwachen Klassifikatoren und auch die Kombination mit anderen Ansätzen (siehe hierzu auch die jeweiligen Anmerkungen in den entsprechenden Abschnitten).

- In dem kompletten System wird, bis auf die Dimension des Objektes (Höhe und Breite), keine weiteres Vorwissen eingebracht. Es ist also zu erwarten, dass, vor allem bei dem Nachverarbeitungsschritt, eine nochmalige Steigerung der Ergebnisse durch das Einbringen von Vorwissen erreicht werden kann.

- Um Objekte von verschiedenen Ansichten, bzw. unterschiedliche Objekte zu detektieren, könnten die Klassifikatoren weiter verbessert und „intelligenter“ kombiniert werden.

- Ein weiterer Ausblick wäre, diese Detektion nicht nur auf statische Bilder, sondern auch auf Videos anzuwenden und mit anderen geeigneten Methoden zum Verfolgen von Objekten (*tracking*) zu koppeln.

Literaturverzeichnis

[1] S. Agarwal, A. Awan, and D. Roth. Learning to Detect Objects in Images via a Sparse, Part-Based Representation. In *Proceedings of the IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 26, pages 1475–1490, 2004.

[2] S. Agarwal and D. Roth. Learning a Sparse Representation for Objet Detection. In *Proceedings of the ECCV*, 2002.

[3] J. Barrets, P. Menezes, and J. Dias. Human-Robot Interaction based on Haar-like Features and Eigenfaces, 2002.

[4] C. Beleznai, B. Frühstück, H. Bischof, and W. Kropatsch. Detecting Humans in Groups Using a Fast Mean Shift Prozedure. In *Workshop of the AAPR/ÖAGM*, pages 71–78, Hagenberg, Austria, 2004.

[5] M. Betke, E. Haritaoglu, and L.S. Davis. Multiple Vehicle Detection and Tracking in Hard Real Time. Technical Report CS-TR-3667, 1996.

[6] G.R. Bradski. Computer Vision Face Tracking For Use in a Perceptual User Interface. *Intel Technology Journal*, 1998.

[7] L. Breiman. Bagging Predictors. *Machine Learning*, 26(2):123–140, 1996.

[8] L. Breiman. Bias, Variance, and Arcing Classifiers. Technical Report 460, Statistics Department, University of California, 1996.

[9] L. Breiman. Arcing Classifiers. *The Annals of Statistics*, 26(3):801–849, 1998.

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

[11] C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[12] G.C. Cawley. Matlab Support Vector Machine Toolbox. University of East Anglia, School of Information Systems, 2000.

[13] R.T. Collins. Mean-Shift Blob Tracking through Scale Space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 234–240, 2003.

[14] D. Comaniciu and P. Meer. Mean Shift Analysis and Applications. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1197–1203, Kerkyra, Greece, 1999.

[15] D. Comaniciu, V. Ramesh, and P. Meer. Real-Time Tracking of Non-Rigid Objects using Mean Shift. pages 142–151, 2000.

[16] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley-Interscience, 2 edition, 2000.

[17] N. Faggian. Implementation of the Viola-Jones Face Detection System. Master's thesis, Monash University, 2003.

[18] I. Fasel and J.R. Movellan. A Comparision of Face Detection Algorithms. In J.R. Dorronsoro, editor, *Proceedings of the ICANN*, pages 1325–1330. Springer-Verlag Berlin Heidelberg, 2002.

[19] R. Fergus, P. Perona, and A. Zisserman. Object Class Recognition by Unsupervised Scale-Invariant Learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2003.

[20] J.D. Foly, A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics, Principles and Practice*. Addison-Wesley, second edition, 1990.

[21] Y. Freund and R. Schapire. A Short Introduction to Boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, 1999.

[22] Y. Freund and R.E. Schapire. Game Theory, On-Line Prediction and Boosting. In *Proceedings of the 9th Annual Conference on Computational Learning Theory*, pages 325–332, 1996.

[23] Y. Freund and R.E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[24] A. Garg, S. Agarwal, and T. Huang. Fusion of Global and Local Information for Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2002.

[25] B. Heisele, A. Verri, and T. Poggio. Learning and Vision Machines. *Proceedings of the IEEE*, 90:1164–1177, 2002.

[26] M. Kearns and L.G. Valiant. Cryptographic Limitations on Learning Boolean Formulae and Finite Automata. *Journal of the Association for Computing Machinery*, 41(1):67–95, 1994.

[27] B. Leibe, A. Leonardis, and B. Schiele. Combined Object Categorization and Segmentation with an Implicit Shape Model. In *ECCV'04 Workshop on Statistical Learning in Computer Vision*, Prague, May 2004.

[28] A. Levin, P. Viola, and Y. Freund. Unsupervised Improvement of Visual Detectors using Co-Training, 2001.

[29] R. Lienhart, A. Kuranov, and V. Pisarevsky. Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection. Technical Report MRL, 2002.

[30] R. Lienhart and J. Maydt. An Extended Set of Haar-like Features for Object Detection. In *Proceedings of the IEEE ICIP*, pages 900–903, 2002.

[31] W. Maass and G. Kubin. Key Definitions for the Cource Computational Intelligence, 2003.

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

[33] J.R. Movellan, B. Fortenberry, and I. Fasel. A Generative Framework for Real Time Object Detection. Technical Report MPLab TR 2003.03, 2003.

[34] K. Okuma, A. Taleghani, N. De Freitas, J.J. Little, and D.G. Lowe. A Boosted Partivel Filter: Multitarget Detection and Tracking, 2003.

[35] C. Papageorgiou, A. Mohan, and T. Poggio. Example-Baes Object Detection in Images by Components. In *Proceedings of the IEEE Transaction on Pattern Analysis und Machine Intelligence*, volume 23, 2001.

[36] C. Papageorgiou and T. Poggio. A Trainable Object Detection System: Car Detection in Static Images. Technical Report AI-Memo-1673, CBCL-180, Massachusetts Institute of Technology, Artificial Intelligence Laboratory and Center for Biological and Computational Learning, 1999.

[37] C.P. Papageorgiou, M. Oren, and T. Poggio. A General Framework for Object Detection. In *Proceedings of the International Conference on Computer Vision*, 1998.

[38] T.D. Rikert, M. Jones, and P. Viola. A Cluster-Based Statistical Model for Object Detection. In *Proceedings of the 2nd IEEE International Conference on Computer Vision*, pages 1046–1053, 1999.

[39] G. Rätsch. Ensemble-Lernmethoden zur Klassifikation, Eine Analyse von Boosting-Algorithmen. Master's thesis, Universität Potsdam, 1998.

[40] G. Rätsch, T. Onoda, and K.R. Müller. Soft Margins for AdaBoost. *Machine Learning*, 42(3):287–320, 2001.

[41] G. Rätsch, B. Schökopf, S. Mika, and K.R. Müller. SVM and Boosting: One Class, 2000.

[42] R.E. Schapire, Y. Freund, P. Bartlett, and W.S. Lee. Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods. In *Proceedings of the 14th International Conference on Machine Learning*, pages 322–330. Morgan Kaufmann, 1997.

[43] H. Schneiderman. Learning Statistical Structure of Object Detection. In *Proceedings of the CAIP*, 2003.

[44] H. Schneiderman. Feature-Centric Evaluation for Efficient Cascaded Object Detection, 2004.

[45] H. Schneiderman and T. Kanade. Probabilistic Modeling of Local Appearance and Spatial Relationsships for Object Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1998.

[46] H. Schneiderman and T. Kanade. Object Detection Using the Statistics of Parts. *Proceedings of the International Journal of Computer Vision*, 56(3):151–171, 2004.

[47] E.J. Stollnitz, T.D. DeRose, and D.H. Salesin. Wavelets for Computer Graphics: A Primer, Part 1. *IEEE Computer Graphics and Applications*, 15(3):76–84, 1995.

[48] R. Tibshirani. Bias, Variance and Prediction Error for Classification Rules. Technical report, Statistics Department, University of Toronto, 1996.

[49] K. Tieu and P. Viola. Boosting Image Retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 228–235, 2000.

[50] A. Treptow, A. Masselli, and A. Zell. Real-Time Object Tracking for Soccer-Robots without Color Information, 2003.

[51] L.G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[52] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

[53] P. Viola. Complex Feature Recognition: A Bayesian Approach for Learning to Recognize Objects. Technical Report AIM-1591, 1996.

[54] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 511–518, Kauai, Hawai, 2001.

[55] P. Viola and M. Jones. Robust Real-time Object Detection. *International Journal of Computer Vision*, 2002.

[56] J. Wu, J.M. Rehg, and M.D. Mullin. Learning a Rare Event Detection Cascade by Direct Feature Selection, 2003.

[57] R. Xiao, L. Zhu, and H. Zhang. Boosting Chain Learning for Object Detection. In *Proceedings of the 9th IEEE International Conference on Computer Vision*, pages 709–715, Nice, France, 2003.

[58] D. Zhang, S.Z. Li, and D. Gatica-Perez. Real-Time Face Detection Using Boosting in Hierarchical Feature Spaces, 2002.