UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Fußgängererkennung in unstrukturierten Umgebungen

Masterarbeit
zur Erlangung des Grades
Master of Science
im Studiengang Computervisualistik

vorgelegt von

Michael Klostermann

Betreuer: Dipl.-Inform. M. Häselich. Betreuer, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Erstgutachter: Dipl.-Inform. M. Häselich. Betreuer, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Zweitgutachter: Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau

Koblenz, im Dezember 2012

Kurzfassung

Die Erkennung von Fußgängern in digitalen Bildern ist von großem Interesse in der Entwicklung autonomer Systeme und der Interaktion von Computern mit ihrer Umgebung. Die Herausforderungen an ein solches System sind hoch, da die optische Erscheinung von Fußgängern stark variiert und die Umgebung unstrukturiert ist. In dieser Masterarbeit wird ein Standardverfahren aus der Forschung implementiert und erweitert. Dabei ist eine neue Erkenntnis, dass das Merkmal der Color Self-Similarity durch Vorberechnungen um den Faktor 4 beschleunigt werden kann. Das komplette Erkennungsystem wird in dieser Masterarbeit beschrieben und evaluiert, und der Source-Code unter einer Open Source Lizenz veröffentlicht.

Abstract

Pedestrian Detection in digital images is a task of huge importance for the development of automatic systems and in improving the interaction of computer systems with their environment. The challenges such a system has to overcome are the high variance of the pedestrians to be recognized and the unstructured environment. For this thesis, a complete system for pedestrian detection was implemented according to a state of the art technique. A novel insight about precomputing the Color Self-Similarity accelerates the computations by a factor of four. The complete detection system is described and evaluated, and was published under an open source license.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Vereinbarung der Arbeitsgruppe für Studien- und Abschlussarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.	ja ☒	nein ☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	ja ☒	nein ☐

Koblenz, den 1. Dezember 2012

Danksagung

Ich möchte mich bei einer Reihe von Leuten bedanken.

Zuerst gilt der Dank meiner Verlobten, Karo, die mich immer unterstützt hat und mit der ich die beste Zeit meines Lebens habe.

Für die tolle Zeit während des Studiums und die Unterstützung möchte ich mich bei allen meinen Koblenzer Freunden und Kommilitonen bedanken.

Natürlich bin ich auch meinen Eltern, Ulrich und Anita, dankbar für alles, was sie für mich getan haben. Ohne Eure Unterstützung hätte ich dieses Ziel nicht erreicht.

Weiter gilt der Dank auch der Familie meiner Verlobten, Barbara Markus und Jakob, die sich immer interessiert und mich auch über die ganze Zeit begleitet und unterstützt haben.

Während dieser Arbeit war mir mein Betreuer Marcel Häselich eine große Hilfe. Danke für die vielen hilfreichen Gespräche und Deinen Einsatz.

Danke, Andreas, Du warst mir eine große Hilfe beim Korrekturlesen.

Ich bedanken mich bei euch allen.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Das Interesse an der automatischen Erkennung von Fußgängern in Kamerabildern ergibt sich aus den wissenschaftlichen Herausforderungen und aus den zahlreichen Anwendungsmöglichkeiten. Im Folgenden wird zunächst darauf eingegangen, wieso Fußgängererkennung in unstrukturierten Umgebungen ein schwieriges Klassifikationsproblem der Bildverarbeitung ist.

1.1 Herausforderungen

Die Schwierigkeit liegt unter anderem darin, dass die visuelle Erscheinung von Fußgängern höchst variabel ist. Einige Beispiele für Fußgänger sind in Abbildung 1.1 abgebildet. Die Bilder stammen aus dem INRIA Personen-Datensatz [DT05]. Personen unterscheiden sich in Kleidung, Größe, Form, Farbe und Pose. Als Objektklasse weisen Fußgänger also eine hohe Varianz innerhalb der Klasse auf. Eine weitere Schwierigkeit stellt die unstrukturierte Umgebung dar. Der Hintergrund ist in der Regel unbekannt, inhomogen und kann mit anderen Objekten überladen sein. Zusätzlich können Personen durch andere Objekte teilweise verdeckt sein. Die

Abbildung 1.1: Beispiele für Fußgänger aus dem INRIA Personen-Datensatz

Aufgabe eines funktionstüchtigen Detektionssystems ist es, ein geeignetes Modell zu finden, welches die Eigenschaften aller Fußgänger generalisiert, aber gleichzeitig eine Abgrenzung zu Bereichen im Bild definiert, welche keinen Fußgänger enthalten. Welche Modellierung bisher erfolgreich ist, wird in den folgenden Kapiteln dieser Masterarbeit immer wieder ein entscheidendes Thema sein. Eine weitere Herausforderung entsteht durch die unkontrollierte Beleuchtung. Das Detektionssystem hat keine Informationen über Position und Art der Lichtquelle. Im Umfang dieser Masterarbeit wird die Fußgängererkennung nur bei Beleuchtungen, die Tageslicht entsprechen, betrachtet. Die Betrachtung bei Dunkelheit wäre auch interessant, würde aber aktive Beleuchtung und andere Sensorik erforderlich machen.

1.2 Anwendungsmöglichkeiten

Wie bereits angedeutet, ergeben sich eine Reihe von Anwendungsmöglichkeiten für ein automatisches Detektionssystem zur Fußgängererkennung.

So ist es zum Beispiel in der automobilen Sicherheit relevant, Fußgänger zu erkennen. Intelligente Kameras sollen Fußgänger frühzeitig erkennen, um den Fahrer vor einer gefährlichen Situation rechtzeitig zu warnen oder gar über den Boardcomputer eine Notbremsung oder ein Ausweichmanöver einzuleiten. Notbremsassistenten sind seit einigen Jahren in verschiedenen aktuellen Serienfahrzeugen erhältlich. Die meisten Verfahren beschränken sich auf die Detektion großer Objekte, wie etwa anderer Autos. Die Sensorik, die bei diesen Systemen zum Einsatz kommt, ist in der Regel Radar oder LIDAR (Light Detection And Ranging). Die Fußgängererkennung gestaltet sich mit dieser Sensorik schwierig, da die Auflösung bei bezahlbaren Sensoren relativ gering ist und Fußgänger nur wenig Reflexionsfläche bieten. Farbkameras dagegen bieten hohe Auflösungen und sind kostengünstig. Auch deshalb ist ein großer Teil der Forschung in der Fußgängererkennung auf Farbbildsensoren ausgerichtet. Kamerasysteme sind zudem bereits in einigen Serienfahrzeugen verbaut. Ein zuverlässiges Fußgängererkennungssystem, welches auf handelsüblicher Kamerasensorik basiert, wäre also auch wirtschaftlich von Vorteil.

Ein weiteres Anwendungsgebiet ist die Überwachungstechnik. Polizei, Behörden und Sicherheitsdienste haben ein Interesse daran, Videoströme automatisch analysieren zu können. Eine umfassende Überwachung kann nur durch hohen Personalaufwand oder durch intelligente computergestützte Verfahren gewährleistet werden. Personen in diesen Videoströmen zu erkennen ist in diesem Zusammenhang von entscheidender Bedeutung. Es könnte genutzt werden, um verdächtige Personen im Nachhinein wiederzufinden oder auffälliges Verhalten automatisch im laufenden Betrieb zu erkennen.

In der Robotik kann die Fußgängererkennung auch auf vielfältige Art eingesetzt werden. Von autonom agierenden Systemen wird verlangt, dass die Umgebung und

(a) (b)

Abbildung 1.2: Beispielbilder des INRIA Personen-Datensatzes

vor allem auch Personen nicht in Mitleidenschaft gezogen werden. Weiterhin kann es gewünscht sein, dass Personen identifiziert und mit ihnen interagiert wird. Somit kann also auch die Mensch-Maschine-Interaktion von der Personenerkennung profitieren.

In der automatischen Informationsdatenverarbeitung kann die Fußgängererkennung auch genutzt werden. Personen in Bildern aus dem Internet zu erkennen, könnte für viele Akteure interessant sein (z.B. Suchmaschinenbetreiber).

1.3 Beispielbilder

In dieser Masterarbeit werden einige Beispielbilder verwendet um das Verhalten der Algorithmen zu demonstrieren. Diese Bilder sollen hier kurz eingeführt werden um deren Quelle anzugeben. Alle verwendeten Beispiele sind in Abbildung 1.2 abgebildet und stammen aus dem INRIA Personen-Datensatzes [DT05].

1.4 Gliederung

In Kapitel 2 wird der aktuelle Stand der Technik vorgestellt. Die weiteren Kaptiel widmen sich dem in dieser Masterarbeit implementierten Fußgängerdetektionssystem. Die theoretischen Grundlagen werden in Kapitel 3 dargestellt. Kapitel 4 beschäftigt sich mit einigen Aspekten der Implementierung. Die Detektionsqualität und Laufzeit wird in Kapitel 5 evaluiert. Kapitel 6 schließt mit einer Zusammenfassung sowie dem Ausblick.

Kapitel 2

Stand der Technik

Fußgängererkennung ist ein für Forschung und Anwendung gleichermaßen relevantes Gebiet. Es gibt eine ganze Reihe relevanter Literatur zum Thema Objekterkennung und speziell auch zur Personen- und Fußgängererkennung in Kamerabildern. In diesem Kapitel wird der aktuelle Stand der Technik dargestellt. Die Hauptbestandteile eines Detektionssystems sind die Repräsentation der Merkmale und das Lernverfahren. Eine Einteilung nur anhand dieser Bestandteile wäre aber verfrüht, da in aktuellen Verfahren Kombinationen auftreten. Es werden zunächst einige Verfahren vorgestellt, die im Laufe der Zeit oft referenziert und erweitert wurden. Diese Verfahren sind in aktuellen Evaluationen, wie z.B. [DWSP12] teilweise nicht enthalten.

2.1 Historische Grundlagen

Es gibt zwei wesentliche Pionierleistungen, auf denen aktuelle Verfahren aufbauen. Hier ist zum einen das Verfahren von Papageorgiou und Poggio [PP00], sowie zum anderen jenes von Viola et al. [VJS03] zu nennen.

Das trainierbare Detektionssystem von Papageorgiou und Poggio [PP00] ist Vorbild für aktuelle Verfahren in Bezug auf die einzelnen Bestandteile eines Detektionssystems. Es wird ein Detektionsfenster definiert in dem ein Merkmalsvektor berechnet wird. Eine Support Vector Machine (SVM) [VC95] dient dazu, Merkmalsvektoren binär zu klassifizieren. In Abweichung zu aktuellen Verfahren nutzen Papageorgiou et al. als Merkmale eine *überkomplette* Repräsentation von Haar-Wavelets auf Grauwertbildern. Überkomplett bedeutet, dass Rechtecke, die zur Berechnung herangezogen werden, sich auch überlappen dürfen. Die Autoren definieren zunächst eine größere Anzahl solcher Merkmale und selektieren dann manuell einige wenige, die sie für relevant erachten. Als Trainingsverfahren nutzen sie eine SVM mit polynomiellem Kernel. Das Detektionssystem wird in ihrer Ar-

beit dazu genutzt, um Gesichter, Personen und Autos zu erkennen. Die Arbeit von Papageorgiou und Poggio stellt die Grundlage einer Reihe weiterer Verfahren dar, denen auch das SVM Lernverfahren zugrunde liegt. Die umfassende Eigenschaft ist, dass Merkmale vor der Trainingsprozedur festgelegt werden. Dies steht im Kontrast zu Boosting-Verfahren, bei denen Merkmale während der Lernphase selektiert bzw. verworfen werden.

Viola et al. zeigen, wie ein auf Boosting basierendes Verfahren aufgebaut ist [VJS03]. Ihr Personenerkennungssystem basiert auf ihrer Arbeit zur Gesichtererkennung [VJ01]. Als Lernverfahren kommt die Boosting-Variante AdaBoost [FS95] in Kombination mit Entscheidungsbäumen zum Einsatz. Eine große Anzahl Haar-Wavelet ähnlicher Merkmale, die hier Rechteckfilter genannt werden, wird durch das AdaBoost Verfahren sukzessiv auf die Wesentlichen reduziert. Dies geschieht iterativ durch Minimierung des Trainingsfehlers. Zusätzlich nutzen sie eine Kaskadenarchitektur. Die Entscheidung, ob ein Fußgänger in einem Fenster sichtbar ist, wird auf eine Kette/Kaskade von „Ja/Nein"-Entscheidungen verteilt. Sobald ein Element dieser Kette eine „Nein"-Entscheidung trifft, wird das aktuelle Fenster als negativ bewertet und es findet keine weitere Verarbeitung statt. Diese Ausschlussmethode fördert die Klassifikationsgeschwindigkeit, da Merkmale jeweils nur bis zur ersten „Nein"-Entscheidung berechnet werden müssen. Weiterhin ist das Gesamtsystem, ähnlich wie bei der Gesichtererkennung, echtzeitfähig. Die Performanz resultiert zum einen aus der Kaskade, aber auch zum anderen aus der effizienten Berechnung der Merkmale. Die Auswertung der Rechteckfilter beliebiger Größe geschieht über die Summation aller Werte einer rechteckigen Region und deren Differenzenbildung. Die schnelle Berechnung der Rechtecksummen geht auf Crow et al. zurück [Cro84]. Das Bild wird in eine Integraldarstellung überführt und mit Hilfe dieser Repräsentation können Rechtecksummen in konstanter Zeit mit nur sehr wenigen Berechnungen ausgewertet werden. Beim Verfahren von Viola et al. werden Merkmale innerhalb eines Grauwertbildes, aber auch zwischen dem letzten und vorletzten Bild eines Videostroms berechnet. Hierdurch wird Bewegungsinformation in vereinfachter Form erfasst. Dem resultierenden System fehlt es allerdings an Generalität, weil die Autoren die Personenerkennung nur im Kontext einer Überwachungsanwendung untersuchen. In einem solchen Anwendungsszenario sind einige Vereinfachungen enthalten. Die Kamera ist statisch und somit werden Bewegungsinformationen nicht durch Eigenbewegung gestört. Weiterhin handelt es sich in der Regel um einen zeitlich ähnlichen Hintergrund.

2.2 Gradientenhistogramme

Die Forschung von Dalal und Triggs stellt einen Meilenstein in der Fußgängererkennung dar. Das untersuchte Merkmal „Histograms of Oriented Gradients" (HOG)

[DT05] schafft es, wie kein anderes Merkmal zuvor, die grobe Form der Fußgänger stabil zu kodieren. Das Detektionsfenster wird zunächst in sich überlappende Blöcke unterteilt. Die einzelnen Blöcke bestehen aus dreidimensionalen Histogrammen, quantisiert in lokaler Position und Gradientenorientierung. Eine durchdachte Gewichtung der Gradientenstärken und trilineare Histogrammbildung sorgen dafür, dass Randartefakte vermieden werden und die lokale Form des Objektes stabil festgehalten wird. Schließlich werden die Histogramme normalisiert um Beleuchtungsinvarianz zu erhalten. Ein Merkmalsvektor setzt sich dann aus allen Blöcken eines Detektionsfensters zusammen. Nähere Details zu diesem Merkmal werden in dieser Masterarbeit noch in Abschnitt 3.3.1 erläutert. Dalal und Triggs nutzen eine lineare SVM als Klassifikationsverfahren. Die komplexen Berechnungen der jeweiligen Blöcke machen das Verfahren deutlich langsamer als etwa jenes von Viola et al. [VJS03]. Prisacariu und Reid zeigen, dass das HOG-Verfahren ohne Qualitätseinbußen mit GPU-Unterstützung echtzeitfähig gemacht werden kann [PR09].

Das HOG-Merkmal wird von fast allen aktuell erfolgreichen Detektoren verwendet [DWSP12]. In manchen Verfahren kommen zum Teil auch veränderte Varianten der HOGs zum Einsatz. Diese Varianten tauschen Geschwindigkeit gegen Genauigkeit ein (z.B. [ZYCA06]).

Die Nutzung von Histogrammen zur Informationsreduktion lässt sich nicht nur auf Gradienten anwenden. In einer Weiterentwicklung stellten Dalal und Triggs das Merkmal der „Histograms of Oriented Flow“ (HOF) [DTS06] vor. Hierbei wird das Prinzip der HOGs auf den optischen Fluss [HS81] adaptiert und in Kombination mit den, auf Gradienten basierenden, HOG-Merkmalen konnten deutliche Qualitätsverbesserungen in der Klassifikation erreicht werden.

Die Detektionsqualität des HOG-Merkmals kann durch Kombinationen mit komplementären Merkmalen und anderen Lernverfahren verbessert werden. Dies bestätigt auch die Arbeit von Wojek et al. [WWS09]. Sie fokussieren sich auf Bewegtbilder einer mobilen Plattform und testen die Qualität der HOG, HOF und Haar-Wavelet Merkmale in Kombination mit verschiedenen Lernverfahren. Dabei wird neben der von Dalal und Triggs bekannten Kombination aus linearer SVM und HOG auch die durch Maji [MBM08] effizient berechenbare Histogram Intersection Kernel SVM (HIKSVM) verwendet. Des Weiteren wird AdaBoost und dessen Erweiterung MLPBoost zum Vergleich herangezogen. Es zeigt sich, dass die Bewegungsinformationen vor allem bei seitlich sichtbaren Fußgängern hilft und dass HIKSVMs generell bessere Ergebnisse als lineare SVMs liefern.

Auf dieser Basis entwickelten Walk et al. ein System zur Fußgängererkennung [WMSS10]. HIKSVM wird in Kombination mit HOG, HOF und einem neuen Merkmal namens Color Self-Similarity (CSS) benutzt. CSS kodiert die Farbähnlichkeit aller Zellen eines Detektionsfensters zueinander und soll somit, komplementär zu HOG, Farbrelationen von Objekt und Hintergrund beschreiben. Der resultierende

Detektor „MultiFtr+Motion“ erreicht Ergebnisse, die in der aktuellen Evaluation von Dollar et al. [DWSP12] dem Stand der Technik entsprechen.

2.3 Boosting

Während bei SVMs die Merkmale fest vorgegeben werden, kann Boosting die Merkmalsrepräsentation im Lernprozess selbst anpassen und optimieren.

Es wird ein starker Klassifikator aus einer Kette schwacher Klassifikatoren gebildet, die jeweils besser sein müssen als ein Zufalls-Klassifikator. Die Kette wird iterativ verändert um den Trainingsfehler zu minimieren. Die Merkmale sind mit den schwachen Klassifikatoren verknüpft und werden durch Selektion aus einer großen Anzahl vordefinierter Merkmale ausgewählt (z.B. Rechteck-Summen oder Haar-Wavelets aller möglichen Rechtecke eines Detektionsfensters).

Wie schon zuvor angedeutet, ist hier die Arbeit von Viola et al. als wegweisend zu betrachten [VJS03]. Gerade die hohe Effizienz der Detektion zur Laufzeit macht auf Boosting basierende Verfahren besonders interessant. Ausgehend von den guten Ergebnissen, die HOG liefert, forschten Zhu et al. [ZYCA06] mit Erfolg an einer Adaption des HOG-Merkmals auf Basis des Boosting-Ansatzes. Damit dies gelingt, müssen Gradientenhistogramme beliebiger Größe ausreichend schnell ausgewertet werden können. Die Autoren verwenden die Arbeit von Porikli [Por05], der zeigt, dass die schnelle Berechnung von Rechtecksummen auf Integralbildern auch auf Histogramme übertragbar ist. Für jede Histogramm-Klasse (engl. *bin*) des Histogramms wird ein Integralbild erstellt. Das so erhaltene Integralhistogramm kann zur effizienten Berechnung beliebiger Rechteckhistogramme genutzt werden. Die Autoren berechnen anhand dieser Darstellung Haar-Wavelets beliebiger Größe auf Integralhistogrammen der Gradienten. Diese Repräsentation approximiert das HOG-Merkmal. Es werden einige Details, wie Renormalisierung und gauß'sche Gewichtung von Gradienten weggelassen, worunter die Genauigkeit der Repräsentation leidet. Der resultierende Detektor kann die Qualität der originalen HOG-Variante nicht erreichen, aber es wird eine 70-fache Beschleunigung im Vergleich zu älteren Verfahren erreicht. Wie schon beschrieben resultiert der Qualitätsverlust aus der Weglassung einiger wichtiger Details des HOG-Merkmals.

Dollar et al. generalisieren diese Idee in ihrer entscheidenden Arbeit zu den „Integral Channel Features“ [DTPB09]. Ausgehend von der Integraldarstellung mehrerer registrierter Bildkanäle definieren Sie einfache normalisierte Rechtecksummen beliebiger Größe als mögliche Merkmale. Wieder kommt Boosting zum Einsatz um die für die Klassifikation relevanten Merkmale zu bestimmen. Es werden eine ganze Reihe von Kanälen untersucht. Darunter befinden sich das Grauwertbild, Farbkanäle, die Gradientenstärke, Kantenbilder, Gradientenhistogramme, einfache schwellwertbasierte Segmentierung, Difference-of-Gaussian- und Gaborfilter-

Antworten. Für die robuste Fußgängererkennung reichen 8 Kanäle bestehend aus Gradientenstärke, Grauwertbild und Gradientenorientierungshistogramm (6 Kanäle) aus. Obwohl HOG auch hier nur approximiert wird, übertrifft der resultierende Detektor „ChnFtrs“ das Standardverfahren von Dalal und Triggs und gehört zum aktuellen Stand der Technik nach der Evaluation von Dollar et al. [DWSP12]. Ein wichtiger Aspekt, der zu den guten Ergebnissen geführt hat, ist die Kombination verschiedener Kanäle in einem gemeinsamen Framework mit insgesamt nur sehr wenigen zusätzlichen Parametern.

Aufbauend auf den „Integral Channel Features“ publizierten Dollar et al. ein auf Geschwindigkeit optimiertes Verfahren mit dem Titel „The Fastest Pedestrian Detector in the West“ (FPDW) [DBP10]. Da die Gradienten und Gradientenhistogramme nicht skaleninvariant sind, muss eine Pyramide von Merkmalen auf verschiedenen Skalen berechnet werden. Die Berechnung dieser Pyramide wird in der genannten Arbeit als Flaschenhals der Performanz identifiziert. Nichtsdestotrotz berufen sich Dollar et al. auf die Theorie der fraktalen Struktur der visuellen Welt, in der man davon ausgeht, dass die Statistik eines Bildes sich nicht stark verändert, wenn man den Bildausschnitt nur etwas näher oder weiter weg wählt. Die Annahme wird empirisch anhand von Fotografien von Personen als haltbar erwiesen. Die Approximation der Merkmale auf benachbarten Skalen ist somit möglich und wird genutzt. Die Detektionsqualität leidet unter dieser Approximation, aber es können Fußgänger in Kamerabildern der Größe 640×480 mit mehreren Bildern pro Sekunde ohne GPU-Unterstützung erkannt werden.

2.4 Auf Teildetektionen basierende Verfahren

Die vorangegangen Betrachtung bezog sich im Wesentlichen auf rein *monolithische* Detektoren. Monolithisch bedeutet, dass das Objekt als Ganzes in einem Fenster erkannt wird oder nicht. Auf Teildetektionen basierende Verfahren (engl. *part based*) erkennen einzelne Bestandteile des Objektes und treffen eine Entscheidung aufgrund dieses Wissens.

Felzenszwalb et al. schlagen ein solches System vor [FGMR10]. Sie nutzen HOG als Merkmal und definieren ein deformierbares zweidimensionales Sternmodell um die Teil-Ganzes Relation zu modellieren. Sie definieren einen Ursprungsfilter, der in grober Auflösung Personen als Ganzes grob detektieren soll. Es werden mehrere hochauflösende Teilfilter definiert, die zu den jeweiligen Teilen gehören (Kopf, Arme, Becken und Beine). Als Filter bezeichnen sie das Skalarprodukt des HOG-Merkmalsvektors mit der Normalen der SVM-Hyperebene. Dieses Skalarprodukt stellt jeweils einen Entscheidungswert dar. Die relative Position eines Teiles zum Ursprungsfilter wird über eine Kostenfunktion bewertet. Es wird jene Detektion angenommen, welche den höchsten Entscheidungswert im Ursprungsfilter hat und

die Objekthypothese durch Vorhandensein und Position der Teile bestätigt. Auf Teildetektionen basierende Verfahren sind schwierig zu trainieren, weil hierfür auch jedes Teil in der Trainingsdatenbank annotiert sein muss. Die meisten publizierten Fußgänger-Datensätze beinhalten nur Annotationen, welche die gesamte Person umschließen (Boundingbox). Um auch Teile zu erlernen, müssten alle Körperteile korrekt annotiert sein. Welche Teile annotiert werden müssen hängt wiederum von der Teildefinition ab. Felzenszwalb et al. schlagen hierfür eine Erweiterung der linearen SVM um latente Variablen vor (latent SVM). Diese latenten Variablen sind versteckte Variablen, welche die relative Position eines Teiles beschreiben. Diese Variablen werden auf einen Startwert gesetzt und während der Trainingsprozedur iterativ optimiert. Das Gesamtsystem tritt bei Evaluationen unter dem Namen „LatSVM“ an und erreicht bei Dollar et al. [DWSP12] Ergebnisse auf hohem Niveau, wird derzeit noch durch monolithische Verfahren knapp übertroffen. Es ist zu erwarten, dass auf Teildetektionen basierende Verfahren vor allem bei hoch aufgelösten Bildern bessere Ergebnisse erzielen. Weiterhin bietet diese Variante die Möglichkeit Annahmen über den Kontext einer Szene mit in die Detektion einfließen zu lassen. Wenn z.B. ein Auto bestimmte Teile eines Fußgängers verdeckt, kann diese Information in einem solchen Framework berücksichtigt werden.

Kapitel 3

Detektionssystem

In diesem Kapitel steht das implementierte Fußgängererkennungssystem im Fokus. Es basiert in großen Teilen auf der Arbeit von Dalal und Triggs [DT05, Dal06].

Die Hauptbestandteile des Detektionssystems sind der Detektor, der Klassifikator und die Merkmale. Abbildung 3.1 zeigt diese Bestandteile im Datenfluss der Detektionsaufgabe. Der Aufbau ist typisch für viele Objektdetektionsaufgaben. Als Eingabe wird ein Bild f übergeben. Die Aufgabe des Detektors ist es in diesem Bild Objekte zu erkennen und als Detektionen auszugeben. Detektionen werden durch umschließende Rechtecke (Boundingboxen) beschrieben. Der Detektor greift zur Detektion auf einen Klassifikator zurück, welcher in einer Traingsphase erlernt wird. Die Klassifikation geschieht nicht direkt über die Bilddaten, sondern es werden zunächst Merkmale berechnet. Merkmale bezeichnen in der Mustererkennung die Repräsentation eines Musters in Form eines mehrdimensionalen Merkmalsvektors. Dieser Merkmalsvektor beschreibt das Muster und das Ziel ist es eine möglichst markante Beschreibung zu finden anhand derer eine gute und einfache Klassifikation möglich ist. Die Gliederung dieses Kapitels orientiert sich an Abbildung 3.1. Zunächst wird die Funktionsweise des Detektors beschrieben,

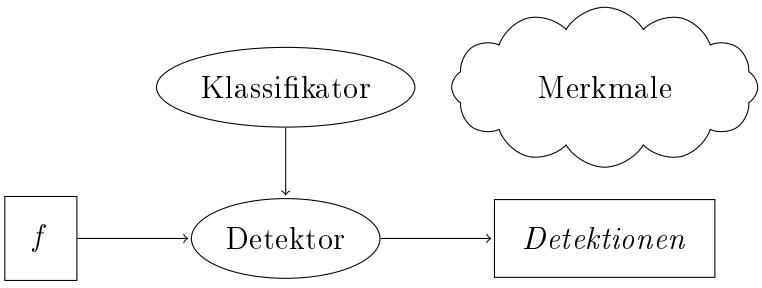
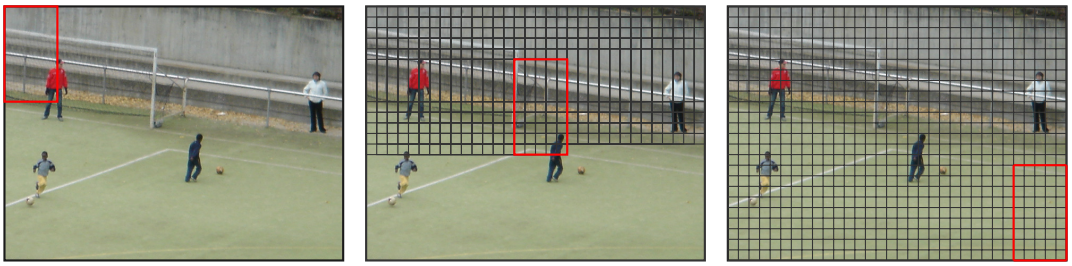


Abbildung 3.1: Komponenten und Datenfluss des Detektionssystems



(a) (b) (c)

Abbildung 3.2: Sliding-Window-Paradigma

dann wird der Klassifikator und dessen Training erläutert und schließlich wird auf die verwendeten Merkmale eingegangen.

3.1 Detektor

Der Detektor ist ein wichtiger Bestandteil der Detektionsaufgabe. Dieses Modul hat die Aufgabe ein Bild systematisch auf Fußgänger zu untersuchen. Diese Suche wird in dem hier vorgestellten System über ein Sliding-Window-Paradigma gelöst. Ein Detektionsfenster fester Größe (z.B. 64×128 Pixel) wird in festen Abständen über das gesamte Bild verschoben. Abbildungen 3.2 (a)-(c) visualisieren diesen Vorgang. Das Detektionsfenster verschiebt sich in x- und y-Richtung über das gesamte Bild. Der Versatz zwischen den Fenstern ist ein zu bestimmender Parameter und wird im Folgenden als s_x und s_y bezeichnet. In Abhängigkeit zur Größe des Bildes $(n_{img} \times m_{img})$ sowie des Detektionsfensters $(n_{win} \times m_{win})$ und des Versatzes (s_x, s_y) berechnet sich, wie viele Fenster pro Bild (w_{img}) untersucht werden:

$$w_{img} = \max(0, \left\lfloor \frac{n_{img}}{s_x} \right\rfloor - \left\lfloor \frac{n_{win}}{s_x} \right\rfloor + 1) \max(0, \left\lfloor \frac{m_{img}}{s_y} \right\rfloor - \left\lfloor \frac{m_{win}}{s_y} \right\rfloor + 1) \qquad (3.1)$$

Bei einem Bild der Größe 640×480 und einem Versatz von 8 Pixeln in beide Richtungen werden 3285 Fenster überprüft. Für jedes dieser Fenster wird nun eine Klassifikation durchgeführt. Es wird hier abstrahiert, dass die Klassifikation eine „Ja/Nein“ Entscheidung trifft. Die Entscheidung lautet „Ja“, falls in diesem Fenster laut Klassifikator ein Fußgänger enthalten ist und „Nein“ andernfalls. Zu jeder „Ja“-Entscheidung wird zusätzlich ein Entscheidungswert vom Klassifikator zurückgegeben, der angibt wie sicher der Klassifikator ist. Diese Vorgehensweise liefert für ein Eingabebild eine bestimmte Anzahl von positiv klassifizierten Fenstern.

Aufgrund der festen Fenstergröße könnte der bisher beschriebene Detektor nur Fußgänger einer bestimmten Größe erkennen. Der Detektor wäre nicht skalierungs-

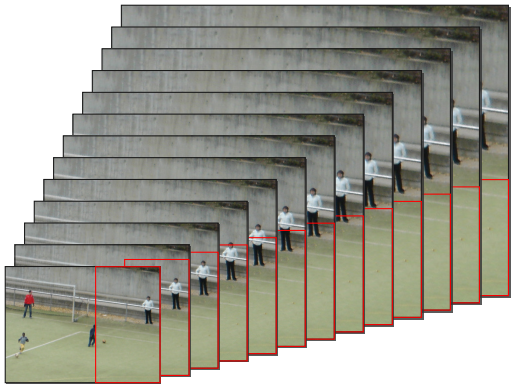


Abbildung 3.3: Skalenraum

invariant. In der Bildverarbeitung ist es üblich, dass ein nicht skalierungsinvariantes Verfahren über die Bildung eines Skalenraumes annähernd skaleninvariant gemacht wird. Auch hier bietet es sich an das Eingangsbild in mehrere Skalen zu transformieren und auf jeder Skale das Sliding-Window-Paradigma durchzuführen. Das Eingangsbild wird zunächst um einen optionalen initialen Skalierungsfaktor $s_{start} \geq 1$ vergrößert und dann sukzessive um einen Faktor $s_{shrink} < 1$ verkleinert, bis auf der untersten Stufe nur noch ein Detektionsfenster in x- oder y-Richtung hineinpasst. Auf jeder Skale wird dann, wie oben beschrieben, das Sliding-Window-Verfahren angewendet. Das Detektionsfenster sowie der Versatz bleiben dabei immer gleich groß. Der Parameter s_{shrink} steuert die Auflösung des Skalenraumes und die Anzahl der Skalen (k_{scales}) berechnet sich wie folgt:

$$k_{scales} = \left\lfloor \frac{\log(\min(\frac{n_{img}}{n_{win}}, \frac{m_{img}}{m_{win}})) + \log(s_{start}))}{-\log(s_{shrink})} + 1 \right\rfloor \tag{3.2}$$

Abbildung 3.3 zeigt den Skalenraum für ein 640×480 Pixel großes Eingabebild und Skalierungsfaktoren $s_{shrink} = 0.9$ und $s_{start} = 1$. Der Skalenraum beinhaltet 13 Skalen. Die Anzahl der Skalen ist sowohl relevant für die Qualität als auch für die Performanz der Detektion. Die Bildung des Skalenraumes selbst sowie die Berechnung der Merkmale auf jeder Skale erhöhen die benötigete Rechenzeit. Daher versuchen manche Autoren, die Berechnung eines Skalenraumes zu umgehen (z.B. [DBP10]). Wird der Skalenraum jedoch zu grob aufgelöst, so können Objekte unerkannt bleiben, weil sie nicht in der jeweiligen Größe erfasst werden. Zu viele Skalen ziehen wiederum mehr falsch Positive nach sich, weil mehr Fenster überprüft werden.

Positive Klassifikationen (Detektionen) werden anhand ihrer Position im dreidimensionalen Skalenraum als Vektoren gespeichert. Die i-te Detektion wird als

$$det_i = \left(\frac{x_i}{s_i}, \frac{y_i}{s_i}, log(s_i) \right)^{\mathrm{T}} \tag{3.3}$$

gespeichert, wobei x_i und y_i den Mittelpunkt des Detektionsfensters auf der aktuellen Skale bezeichnet. Der Skalierungsfaktor der aktuellen Skale ist mit s_i angegeben. Damit die dreidimensionalen Detektionen gleichmäßig im Raum verteilt auftreten, werden die x- und y-Koordinaten mit dem inversen Skalierungsfaktor multipliziert und in der dritten Komponente wird der Logarithmus gebildet.

Führt man die Klassifikation für alle Fenster aller Skalen des Skalenraumes durch erhält man eine Menge an Detektionen. Diese dreidimensionalen Objekte können in das Ursprungsbild rückprojeziert werden. Abbildung 3.5 (b) zeigt ein Beispiel. Die blauen Rechtecke im Bild korrespondieren zu dreidimensionalen Detektionen. Es fällt auf, dass Detektionen gehäuft dort auftreten, wo sich Personen befinden. Dieses Phänomen tritt nicht nur bei diesem Beispiel auf, sondern liegt in der Struktur der Sliding-Window-Verarbeitung begründet. Durch die Überlappung der Detektionsfenster entscheidet sich der Klassifikator auch in der Nähe einer Person mehrfach für eine Detektion. Diese mehrfachen Detektionen müssen reduziert werden um einzelne Erkennungen im Bild ausweisen zu können. Diesen Vorgang bezeichnet man als Ausdünnung oder Non-Maxima-Suppression (NMS). Es gibt verschiedene Ansätze, wie NMS hier durchgeführt werden kann. Dollar et al. [DTPB09] gehen so vor, dass sie Fenster paarweise vergleichen. Überlappen sich zwei Fenster um einen bestimmten Prozentsatz wird jenes Fenster mit dem schwächeren Entscheidungswert verworfen. Dieser Vorgang wird solange fortgesetzt bis keine Fenster mehr überlappen.

Dalal und Triggs schlagen den Mean-Shift-Algorithmus [CM02] vor. Mean-Shift wurde im Rahmen dieser Masterarbeit verwendet und wird im Folgenden beschrieben. Als Eingabe erhält der Mean-Shift-Algorithmus eine Menge von n-dimensionalen Punkten und zu jedem i-ten Punkt muss ein n-dimensionaler Unsicherheitsvektor σ_i angegeben werden, der die Unsicherheiten in jeder Dimension durch n Varianzen angibt. Optional kann zu jedem Punkt ein Gewicht angegeben werden, ansonsten wird ein Gewicht von 1 angenommen. Beim Mean-Shift-Algorithmus geht man davon aus, dass die Datenpunkte Zufallsexperimente einer oder mehrerer Gaußverteilungen sind. Das Ziel des Mean-Shift-Algorithmus ist es, die Mittelwerte dieser Verteilungen zu schätzen. Dies geschieht über einen iterativen Prozess. Jeder Punkt der Ursprungsmenge dient jeweils als Startwert. Dieser Startwert wird nun in jedem Schritt in Richtung des Mittelwertes seiner Nachbarn verschoben. Zur Nachbarschaft gehören alle anderen Punkte der Ursprungsmenge, wobei die Nachbarn über die Mahalanobis-Distanz sowie über das Gewicht abgeschwächt oder verstärkt werden. Die Mahalanobis-Distanz berücksichtigt die zuvor

übergebenen Unsicherheiten σ_i der Punkte. Der iterative Prozess bricht ab, wenn die Veränderung nach der jeweiligen Iteration einen Schwellwert unterschreitet. In diesem Zustand gilt der Punkt als konvergiert. Da dieser Prozess für jeden Punkt der Ursprungsmenge durchgeführt wird erhält man eine gleich große Menge konvergierter Punkte. Theoretisch würden alle Punkte einer Distribution im gleichen Mittelwert konvergieren. Aufgrund numerischer Ungenauigkeiten ist dies aber in der Implementierung nicht der Fall. Daher werden zuletzt alle konvergierten Punkte vereinigt, die eine bestimmte euklidische Distanz unterschreiten. Abbildung 3.4 zeigt die Arbeitsweise des Mean-Shift-Algorithmus anhand einiger Beispiele. Die Daten dieser Beispiele sind durch Zufallsgeneratoren generiert. Die Zufallsgeneratoren basieren auf den bivariaten Gaußverteilungen $\mathcal{N}_1(\boldsymbol{\mu_1}, \boldsymbol{\Sigma_1^2})$ und $\mathcal{N}_2(\boldsymbol{\mu_2}, \boldsymbol{\Sigma_2^2})$. Die Parameter der Gaußverteilungen sind

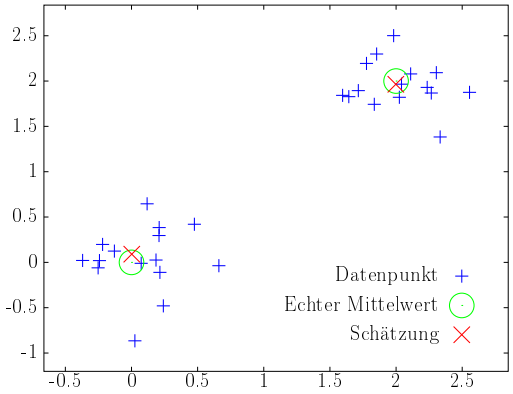
$$\boldsymbol{\mu_1} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \boldsymbol{\mu_2} = \begin{pmatrix} 2 \\ 2 \end{pmatrix} \text{ und } \quad \boldsymbol{\Sigma_1^2} = \boldsymbol{\Sigma_1^2} = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.1 \end{pmatrix}. \tag{3.4}$$

Zu jedem Punkt wird der gleiche Unsicherheitsvektor von $\sigma_i = (0.1, 0.1)^{\mathrm{T}}$ übergeben. Datenpunkte werden als blaue Plus-Zeichen dargestellt. Die durch Mean-Shift geschätzten Mittelwerte $\tilde{\boldsymbol{\mu}}_\mathbf{1}$ und $\tilde{\boldsymbol{\mu}}_\mathbf{2}$ werden als rote Kreuze markiert. Die tatsächlichen Mittelwerte $\boldsymbol{\mu_1}$ und $\boldsymbol{\mu_2}$ werden als grüne Kreise abgebildet. In Abbildung 3.4 (a) erkennt man, dass die geschätzten Mittelwerte

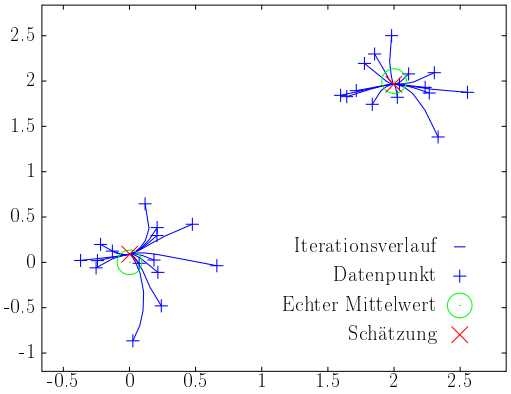
$$\tilde{\boldsymbol{\mu}}_\mathbf{1} = \begin{pmatrix} 0.0019711 \\ 0.0906366 \end{pmatrix} \text{ und } \tilde{\boldsymbol{\mu}}_\mathbf{2} = \begin{pmatrix} 1.9983116 \\ 1.9641620 \end{pmatrix} \tag{3.5}$$

den tatsächlichen Mittelwerten $\boldsymbol{\mu_1}$ und $\boldsymbol{\mu_2}$ annähernd entsprechen. Weder die tatsächlichen Mittelwerte, noch die zugrunde liegenden Varianzen waren dem Algorithmus bekannt. In Abbildung 3.4 (b) ist zusätzlich der Iterationsverlauf eines jeden Punktes in Form von blauen Linien dargestellt. Abbildung 3.4 (c) und (d) zeigen ein Beispiel, bei dem die ursprünglichen Mittelwerte nicht mehr geschätzt werden können, weil diese zu eng beieinander liegen. Was bei Anwesenheit von Ausreißern geschieht, zeigt sich in Abbildung 3.4 (e): Die Schätzungen der echten Verteilungen werden nicht gestört, aber es wird ein neuer Schätzwert für den Ausreißer hinzugefügt. Schließlich zeigt Teil (f) der Abbildung 3.4 wie sich unterschiedlich gewichtete Punkte auf die Schätzung auswirken. Die Gewichtung jedes Punktes ist durch einen zusätzlichen blauen Kreis visualisiert. Je größer das Gewicht, desto größer der Kreis. Vor allem beim Mittelwert $(2, 2)^{\mathrm{T}}$ ist zu erkennen, wie die Gewichtung im Gegensatz zu Abbildung (a) und (b) Einfluss nimmt.

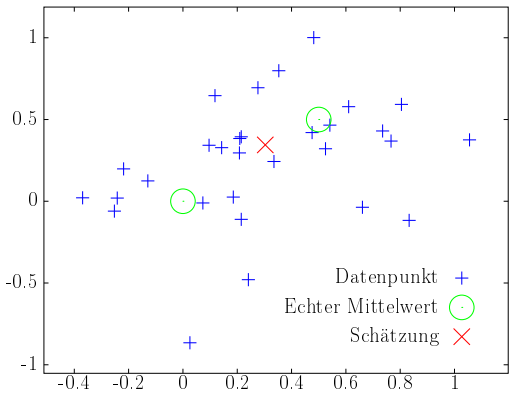
Mean-Shift wurde im Rahmen dieser Masterarbeit sowohl in *Octave/Matlab* als auch in *C++* implementiert. Die Beispiele aus Abbildung 3.4 wurden aus *Octave/-Matlab*-Code erstellt und können im Ordner `Programmcode/pd/meanshift/matlab` der beigefügten DVD ausgeführt und verändert werden.



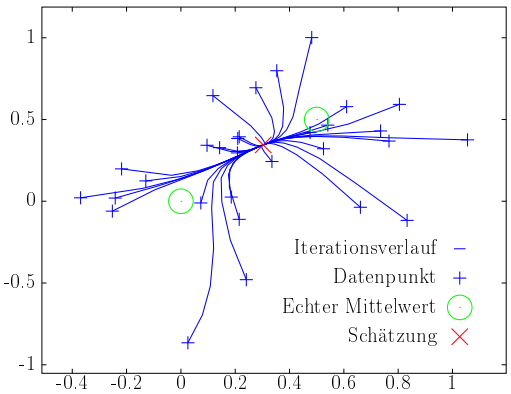
(a) $\boldsymbol{\mu_1} = (0,0)^{\mathrm{T}}$, $\boldsymbol{\mu_2} = (2,2)^{\mathrm{T}}$



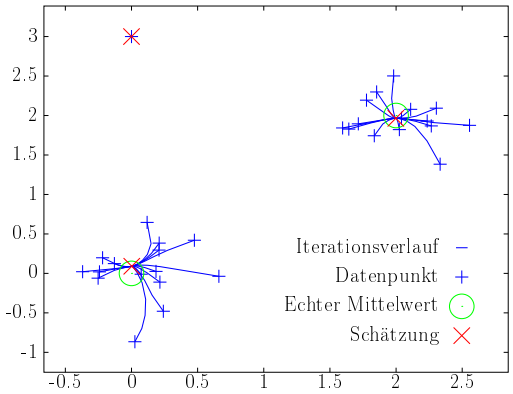
(b) $\boldsymbol{\mu_1} = (0,0)^{\mathrm{T}}$, $\boldsymbol{\mu_2} = (2,2)^{\mathrm{T}}$



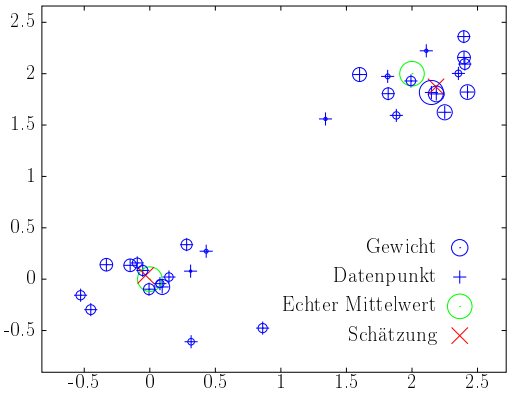
(c) $\boldsymbol{\mu_1} = (0,0)^{\mathrm{T}}$, $\boldsymbol{\mu_2} = (0.5,0.5)^{\mathrm{T}}$



(d) $\boldsymbol{\mu_1} = (0,0)^{\mathrm{T}}$, $\boldsymbol{\mu_2} = (0.5,0.5)^{\mathrm{T}}$



(e) $\boldsymbol{\mu_1} = (0,0)^{\mathrm{T}}$, $\boldsymbol{\mu_2} = (2,2)^{\mathrm{T}}$ mit Ausreißer



(f) $\boldsymbol{\mu_1} = (0,0)^{\mathrm{T}}$, $\boldsymbol{\mu_2} = (2,2)^{\mathrm{T}}$ mit Gewichten

Abbildung 3.4: Mean-Shift-Algorithmus zur Schätzung von Mittelwerten multivariater Gaußverteilungen anhand einiger Beispiele

Für eine detaillierte mathematische Betrachtung und Herleitung des Mean-Shift-Algorithmus wird hier auf die Arbeit von Comaniciu und Meer [CM02] verwiesen.

Die Übertragung des Mean-Shift-Algorithmus auf das Problem der Ausdünnung von Detektionsfenstern stellt sich wie folgt dar. Die zuvor beschriebenen dreidimensionalen Detektionen det_i werden als gaußverteilt angenommen und dienen als Datenpunkte für den Algorithmus. Die Gewichte werden auf die jeweiligen positiven Entscheidungswerte des Klassifikators gesetzt, wobei ein minimales Gewicht von 1 angesetzt wird und die Entscheidungswerte addiert werden. Der Unsicherheitsvektor σ_i ist ein wichtiger Bestandteil für die Stabilität und Qualität der NMS. Zunächst definiert man einen Basis-Unsicherheit ρ. In dieser Masterarbeit wurden gute Ergebnisse mit den Werten

$$\rho = (\rho_x, \rho_y, \rho_z) = (4, 8, \log(1.6))^{\mathrm{T}} \tag{3.6}$$

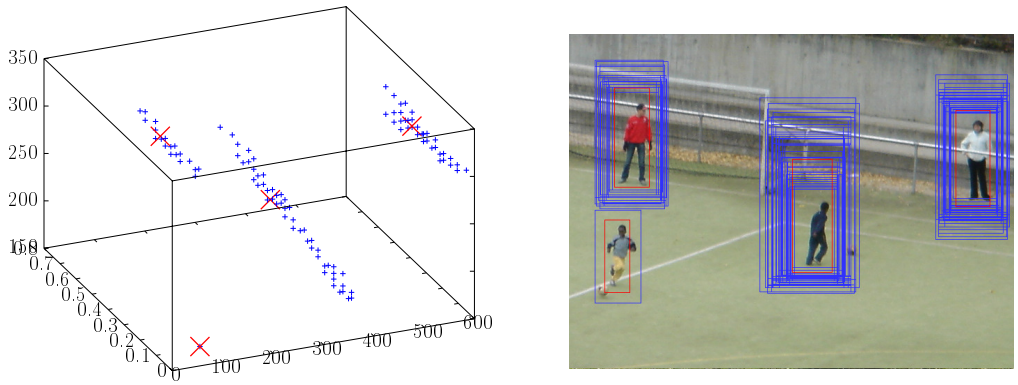
erzielt. Diese Werte werden empirisch bestimmt. In dieser Masterarbeit wurden Detektionsergebnisse in Abhängigkeit zu verschiedenen Werten für ρ untersucht und jeweils festgestellt ob die NMS bestimmte Detektionen zusammengefasst hat, die nicht zusammengefasst werden sollten. Die Unsicherheiten für jede i-te Detektion berechnen sich in Abhängigkeit zum Skalierungsfaktor s_i:

$$\sigma_i = \frac{1}{s_i}(\rho_x, \rho_y, \rho_z)^{\mathrm{T}} \tag{3.7}$$

Der Skalierungsfaktor s_i beschreibt die Skalierung der Skale im Verhältnis zum Ausgangsbild. Je kleiner dieser Wert ist, desto kleiner ist das Bild und desto größer ist die Unsicherheit bezüglich der Lokalisation von Detektionen. Dies wird durch σ_i festgelegt.

Damit sind alle erforderlichen Eingaben für den Mean-Shift-Algorithmus definiert und das Verfahren kann ausgeführt werden. Abbildung 3.5 (a) zeigt anhand eines Beispiels Detektionen als blaue Plus-Zeichen im dreidimensionalen Raum und deren per Mean-Shift geschätzten Mittelwerte in Form von roten Kreuzen. Gewichte sind aus Gründen der Übersichtlichkeit nicht visualisiert. Die Rückprojektion der geschätzten Mittelwerte und der Detektionen in das Ursprungsbild werden in Abbildung 3.5 (b) gezeigt. Die roten Rechtecke sind die final bestimmten Regionen, in denen eine Person detektiert wurde. Diese roten Rechtecke sind kleiner als das Detektionsfenster selbst, weil im Detektionsfenster um die Person herum jeweils ein Rand gelassen wird. Dieser Rand muss im letzten Schritt vom Detektionsfenster abgezogen werden.

Damit ist die Funktionsweise des Detektor-Moduls abschließend beschrieben. Es folgt ein Abschnitt, der sich mit der Trainingsphase bzw. dem Klassifikator beschäftigt.



(a) **(b)**

Abbildung 3.5: Non-Maxima-Suppression unter Verwendung des Mean-Shift-Algorithmus

3.2 Klassifikator

Das hier vorgestellte Detektionssystem basiert auf dem Sliding-Window-Paradigma. Wie im Abschnitt zuvor beschrieben, werden einzelne Fenster fester Größe in mehreren Skalen klassifiziert. Das Ziel des Trainings ist es einen möglichst guten binären Klassifikator zu trainieren, der bei Eingabe eines Detektionsfensters die Entscheidung trifft ob ein Fußgänger im aktuellen Fenster enthalten ist oder nicht. Zur Detektion werden nicht die Pixel-Werte selbst herangezogen, sondern es wird pro Fenster ein Merkmalsvektor einer bestimmten Dimension berechnet. Welche Dimension dieser Merkmalsvektor hat, und wie er berechnet wird, ist nicht Thema dieses Abschnitts, sondern wird in Abschnitt 3.3 näher erläutert. Für die Betrachtung des Trainings reicht es zunächst anzunehmen, dass der Merkmalsvektor von einem entsprechenden Modul erzeugt wird.

3.2.1 Support Vector Machines

In dieser Masterarbeit werden, wie bei Dalal und Triggs [DT05], zur Klassifikation lineare Soft-Margin Support Vector Machines (SVMs) [VC95, TK09] eingesetzt. Im Folgenden wird die binäre Klassifikation mehrdimensionaler Merkmalsvektoren in zwei verschiedene Klassen ω_1 und ω_2 betrachtet. Gegeben seien Trainingsdaten in Form einer Menge von Merkmalsvektoren x_i mit $i = 1, ..., l$. Zu jedem Trainingsvektor sei über $y_i \in \{-1, 1\}$ angegeben, zu welcher Klasse er zugehörig ist.

$$y_i = \begin{cases} -1, & \text{falls } x_i \text{ zur Klasse } \omega_1 \text{ gehört} \\ 1, & \text{falls } x_i \text{ zur Klasse } \omega_2 \text{ gehört} \end{cases} \tag{3.8}$$

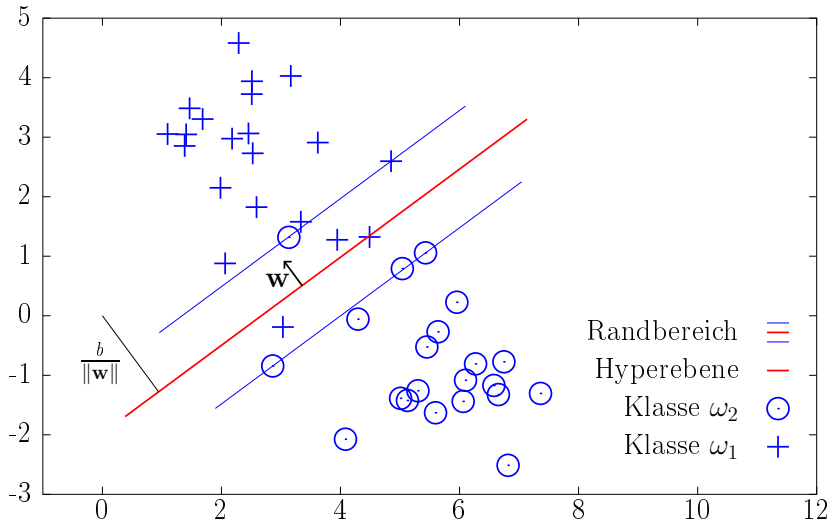


Abbildung 3.6: Ein Beispiel für die binäre lineare Klassifikation mit dem SVM-Verfahren bei nicht linear separierbaren Trainingsdaten. Die gezeigte Hyperebene entspricht der Optimierung der Gleichungen 3.9 und 3.10, wobei $C = 1$ gewählt wurde.

Eine Soft-Margin SVM löst das Problem der binären linearen Klassifikation so, dass eine optimal separierende Hyperebene berechnet wird, die zwischen positiven und negativen Merkmalsvektoren diskriminiert. Wenn Merkmalsvektoren die Dimension n haben, dann hat diese Hyperebene die Dimension $n-1$. Trainingsdaten sind selten linear separierbar. Im nicht-separierbaren Fall befindet sich ein Randbereich, auch Band genannt, in der unmittelbaren Umgebung der Hyperebene. In diesem Randbereich werden einige Vektoren falsch klassifiziert. Dieser Randbereich ist unter anderem in Abbildung 3.6 ausgezeichnet. Außerhalb dieses Randbereichs werden alle Vektoren korrekt klassifiziert. Eine so definierte SVM benötigt die Minimierung der Kostenfunktion 3.9 unter Berücksichtigung der Randbedingungen 3.10.

$$\min_{w,b,\boldsymbol{\xi}} \quad \frac{1}{2} w^{\mathrm{T}} w + C \sum_{i=0}^{l} \xi_i \tag{3.9}$$

$$\text{gemäß} \quad y_i (w^{\mathrm{T}} x_i + b) \geq 1 - \xi_i, \tag{3.10}$$

$$\xi_i \geq 0$$

Diese Formulierung fasst alle zuvor beschriebenen Fälle der Separierbarkeit in einer Formel zusammen. Der Vektor w beschreibt den Normalenvektor der Hyperebene und b beschreibt den Abstand der Hyperebene zum Ursprung des Koordinatensystems. C ist ein Parameter, der die additiven Terme relativ gewichtet. Der Vektor $\boldsymbol{\xi} = (\xi_0, ..., \xi_i, ..., \xi_l)$ beinhaltet die sogenannten *Slack-Variablen* ξ_i und beschreibt über die Randbedingungen 3.10 welchen Einfluss richtige und falsche Klassifikationen in bestimmten Abständen zur Hyperebene haben. Wie die Autoren Theodoridis und Koutroumbas in [TK09] auch beschreiben, kann der Wertebereich der Slack-Variablen in drei Bereiche unterteilt werden, die jeweils eine geometrische Bedeutung haben:

- $\xi_i = 0$ beschreibt die Randbedingung für alle Vektoren die außerhalb des Randbereichs liegen und korrekt klassifiziert werden. Dieser Fall ist optimal, da solche Slack-Variablen die Kostenfunktion 3.9 nicht belasten. Falls die Trainingsdaten linear separierbar wären, würde nur dieser Fall auftreten und es würde eine Hyperebene berechnet, die einen maximalen Rand zwischen den Trainingsdaten ließe.

- $0 > \xi_i \leq 1$ beschreibt den Fall, dass Vektoren im Randbereich liegen und korrekt klassifiziert werden.

- $\xi_i > 1$ behandelt den Fall, wobei Vektoren im Randbereich falsch klassifiziert werden. Hier muss die Slack-Variable groß sein um die falsche Klassifikation $y_i(w^{\mathrm{T}} x_i + b)$ auszugleichen, was wiederum zu hohen Kosten in Gleichung 3.9 führt, welche minimiert werden soll.

Gleichungen 3.9 und 3.10 definieren ein Optimierungsproblem, welches über Konvexe Programmierung gelöst werden kann. Für Details zur Lösung wird auf die einschlägige Literatur verwiesen [VC95, TK09]. Als Lösung erhält man die Beschreibung der Hyperebene, gegeben in Form des Normalenvektors w und der Variablen b. Ein zu klassifizierender Merkmalsvektor x kann durch folgende Gleichung klassifiziert werden:

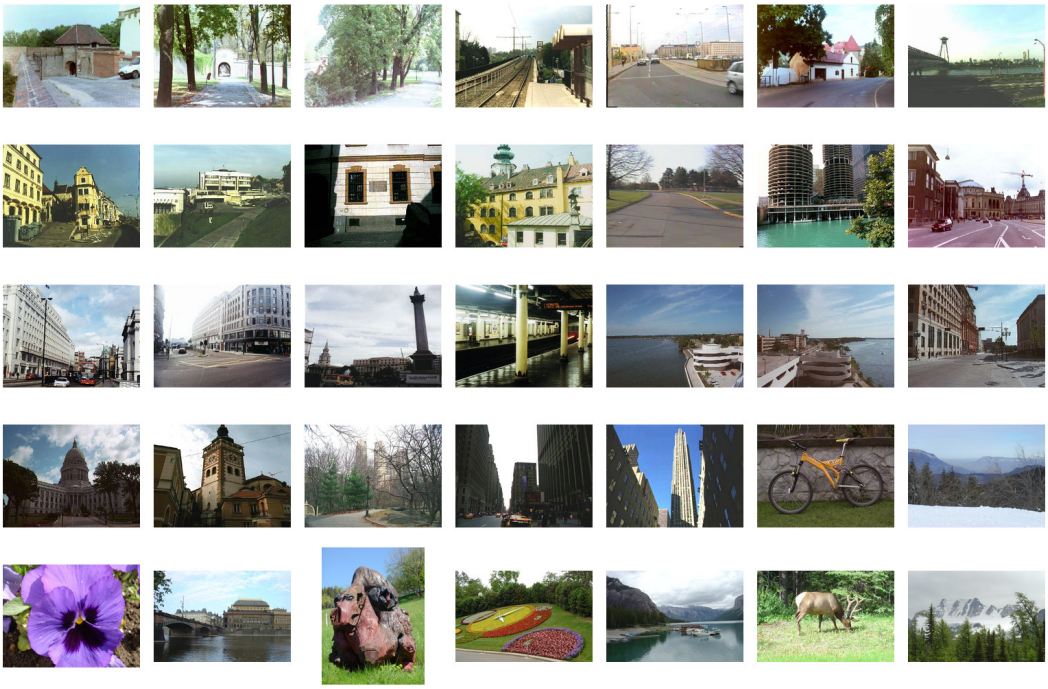
$$\operatorname{sign}\left(w^{\mathrm{T}} x + b\right) \tag{3.11}$$

Entscheidungswerte, die angeben, wie sicher die Klassifikation ist, erhält man durch Auslassung der Signum-Funktion:

$$w^{\mathrm{T}} x + b \tag{3.12}$$

Im Kontext dieser Masterarbeit wird das Programm LIBLINEAR von Fan et al. [FCH+08] verwendet, welches in Bezug auf lineare SVMs bei hohem Datenaufkommen optimiert ist.

(a) Positive Bilder



(b) Negative Bilder

Abbildung 3.7: Beispiele aus dem INRIA Trainingsdatensatz

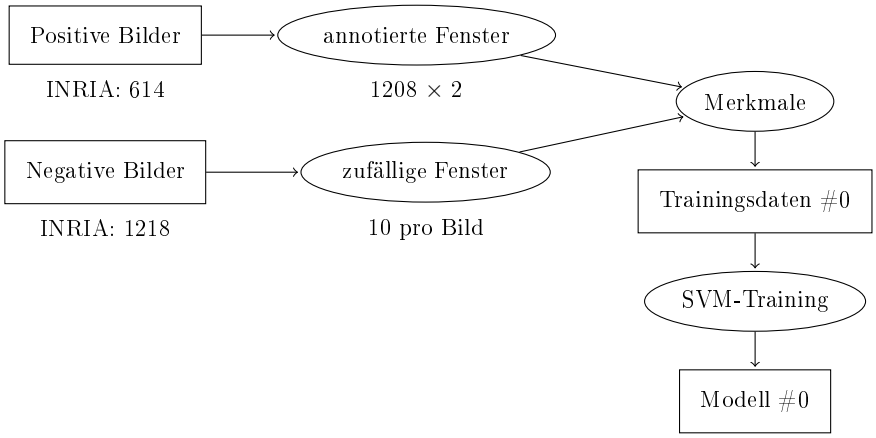


Abbildung 3.8: Datenfluss der initialen Trainingsprozedur

3.2.2 Trainingsdaten

SVMs gehören zur Klasse der *überwachten* Lernverfahren. Überwachtes Lernen bedeutet, dass der Anwender dem Lernverfahren positive wie negative Beispiele des zu klassifizierenden Objektes zuführt. Diese Beispiele nennt man auch Trainingsdaten und in dieser Masterarbeit wurde mit dem INRIA Personen-Datensatz [DT05] gearbeitet. Dieser Datensatz beinhaltet 615 positive Trainingsbilder mit insgesamt 1208 Annotationen. Weiterhin enthält der Datensatz 1218 negative Trainingsbilder, die keine Personen enthalten. Die Bilder dieses Datensatzes sind eine Kollektion verschiedener Fotos. Die Fotos wurden in verschiedenen Szenarien und mit unterschiedlichen Kameras bei variierenden Lichtverhältnissen aufgenommen. Abbildung 3.7 zeigt Beispiele.

3.2.3 Trainingsprozedur

Die Trainingsdaten müssen für das Lernverfahren in geeigneter Weise aufbereitet werden. Abbildung 3.8 zeigt den Datenfluss der verwendeten initialen Trainingsprozedur. Die rohen Bilddaten sind aufgeteilt in positive wie negative Beispielbilder. Positive Bilder sind mit Annotationen versehen, an welcher Stelle und in welchem rechteckigen Bereich sich eine Person befindet. Diese Annotationen müssen extrahiert werden und auf die richtige Größe skaliert werden. Ein übliches Detektionsfenster hat die Größe 64×128 Pixel, die annotierten Personen weisen verschiedene Größen im jeweiligen Bild auf. Weiterhin ist zu beachten, dass um die eigentliche Person ein gewisser Rand für den Hintergrund gelassen werden muss. Die so erhaltenen Fenster werden analog zu Dalal und Triggs normalisierte Fenster genannt. Aus den normalisierten Fenstern werden jeweils Merkmalsvektoren berechnet, die in dieser Form dem Trainingsverfahren als positive Beispiele zugeführt werden.

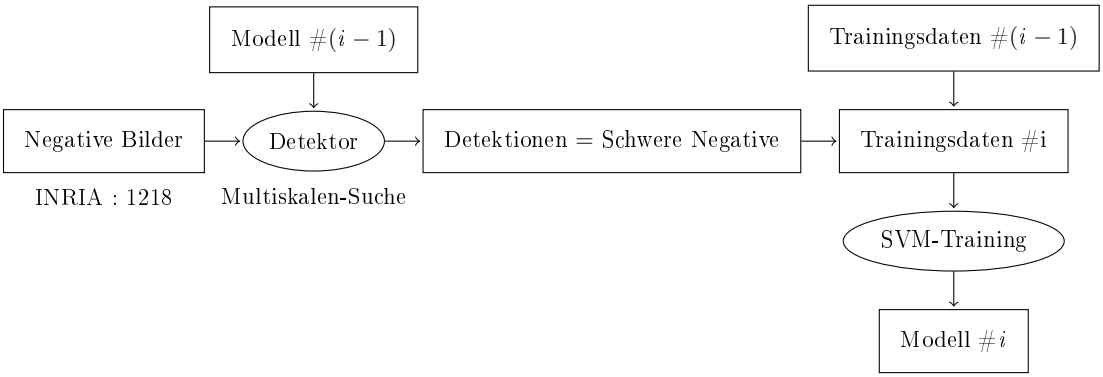
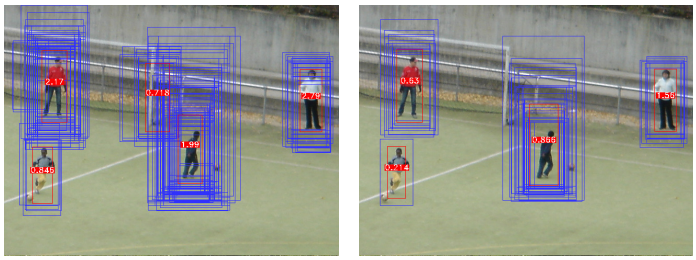


Abbildung 3.9: Datenfluss der i-ten Retraining-Phase

Das Vorgehen bei negativen Beispielen ist anders. Hier beinhalten die negativen Trainingsbilder keine Annotationen. Stattdessen werden pro Bild 10 zufällig platzierte normierte Fenster ausgewählt. Aus diesen normierten Fenstern werden dann wieder Merkmalsvektoren berechnet, die zusammen mit den positiven Beispielen die initialen Trainingsdaten darstellen. Anhand der so generierten Trainingsdaten kann nun der SVM-Klassifikator trainiert werden. Die Ausgabe dieser Prozedur ist eine Modelldatei, die nun dem zuvor beschriebenen Detektor zur Verfügung gestellt werden kann. Die Modelldatei beinhaltet die Beschreibung der separierenden Hyperebene in Form eines Normalenvektors und einige Metadaten, wie z.B. die Größe des Merkmalsvektors.

Das so erhaltene Modell weist das Problem auf, dass zu viele falsch-positive Entscheidungen getroffen werden. Abbildung 3.10 (a) zeigt ein Beispiel. Intuitiv lässt sich dieses Problem so erklären, dass der SVM in unmittelbarer Umgebung der separierenden Hyperebene zu wenige negative Beispiele als Trainingsdaten geliefert wurden. In der initialen Trainingsprozedur wurden pro negativem Trainingsbild nur 10 zufällige Fenster ausgewählt. Diese grobe Auswahl bringt es mit sich, dass nicht genügend *schwere* Negative im Trainingsdatensatz enthalten sind. Schwere Negative aus den negativen Trainingsbildern zu extrahieren und sie dem Trainingsverfahren zuzuführen ist Ziel der erweiterten *Retraining*- oder *Bootstrapping*-Phase. Abbildung 3.9 zeigt den Datenfluss dieser Phase. Das zuletzt erlernte Modell wird dazu genutzt um mit dem Detektor auf den negativen Trainingsbildern im gesamten Skalenraum Fußgängerdetektionen zu suchen. Diese Detektionen sind alle falsch-positiv, denn in den negativen Trainingsbildern sind keine Fußgänger enthalten. Die so erhaltenen Detektionen sind die zuvor beschriebenen schweren Negativen. Sie werden zusammen mit den initialen Trainingsdaten wiederum dem SVM-Training zugeführt. Der so erhaltene Detektor weist eine wesentlich verbesserte Falsch-Positive-Rate auf (siehe Abbildung 3.10 (b)). Die richtig Positiven Detektionen werden allerdings in ihrer Entscheidungssicherheit abgeschwächt.



(a) Initiales Training **(b)** 1-faches Retraining

Abbildung 3.10: Auswirkung der Retraining-Phase auf die Detektionen anhand eines Beispielbildes

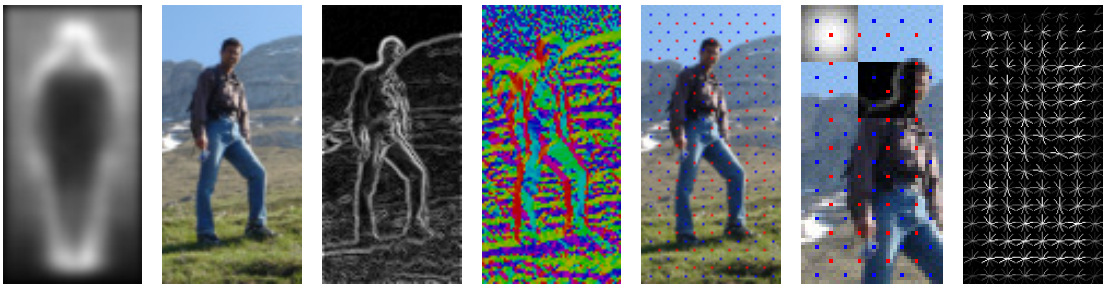
Dies liegt anschaulich an der Verschiebung der separierenden Hyperebene in Anwesenheit der schweren Negativen. Die Entfernung zur Hyperebene nimmt ab, also wird der berechnete Entscheidungswert kleiner. Weiterhin geschieht es, dass richtig-positive, die nah an der Hyperebene waren, nun nicht mehr detektiert werden. Die Retraining-Phase kann mehrfach hintereinander durchgeführt werden. Die Trainingsdaten wachsen dann immer weiter an. Es ist zu beobachten, dass die Anzahl der schweren Negativen die jeweils gefunden werden mit zunehmender Iteration abnimmt. In Kapitel 5 wird evaluiert, wie sich die mehrfache Iteration der Retraining-Phase auf die Detektionsqualität auswirkt.

3.3 Merkmale

Die Merkmale sind ein weiterer zentraler Bestandteil eines Objektdetektionssystems. Ein Merkmal beschreibt auftretende Muster in Form eines mehrdimensionalen Merkmalsvektors. Das Ziel ist es eine möglichst markante Beschreibung zu finden, welche die Klassifikation zwischen der Objekt-Klasse und der Nicht-Objekt-Klasse ermöglicht. Bei der Fußgängererkennung geschieht die Klassifikation anhand normalisierter Fenster. Ein Merkmal hat die Aufgabe ein solches Fenster in einen mehrdimensionalen Merkmalsvektor zu konvertieren. Im Folgenden werden die Merkmale detailliert beschrieben, welche in dem hier vorgestellten Detektionssystem zum Einsatz kommen.

3.3.1 Histograms of Oriented Gradients

Obwohl die visuelle Gestalt von Fußgängern sehr divers sein kann (siehe Abbildung 1.1), lassen sich dennoch einige Beobachtungen machen, welche für viele Fußgänger gelten. Eine Statistik, die in diesem Zusammenhang interessant ist, ist das durchschnittliche Gradientenstärkebild aller 1208 positiv annotierter Fenster im INRIA



(a) (b) (c) (d) (e) (f) (g)

Abbildung 3.11: Visualisierungen zum HOG-Merkmal

Personen-Datensatz (Abbildung 3.11 (a)). Gradienten sind also ein guter Hinweis auf Fußgänger. Das Durchschnittsbild zeigt, dass vor allem die Kopf-Partie und die vertikalen Strukturen dominant sind. Im Arm- und Beinbereich ist die Ausprägung nicht ganz so stark, welches an den variierenden Posen der Fußgänger liegt. Das Merkmal der Histograms of Oriented Gradients von den Autoren Dalal und Triggs schafft es die Gradienteninformationen auf geschickte Art zu bündeln und trotzdem Variationen in der Pose zuzulassen. Dieses Merkmal wird in seiner Berechnung nun im Detail beschrieben.

Zunächst wird aus dem möglicherweise mehrkanaligen Farbbild die Gradientenstärke und Gradientenorientierung berechnet. Dabei wird bei Farbbildern dieser Prozess auf jedem Kanal einzeln durchgeführt, wobei im Endresultat nur die Gradientenstärke und Gradientenorientierung pro Pixel gespeichert wird, deren Kanal die größte Gradientenstärke aufweist. Abbildungen 3.11 (b-d) zeigen das Eingabebild sowie das berechnete Gradientenstärke- und Gradientenorientierungsbild. Die Gradienten berechnen sich aus der Faltung des Eingangsbildes mit den Filterkernen $[-1\ 0\ 1]$ in x-Richtung sowie $[-1\ 0\ 1]^{\mathrm{T}}$ in y-Richtung. Dies entspricht einer Approximation der Ableitung des Bildes $f(x, y)$ in beide Richtungen. Die Gradientenstärke berechnet sich aus

$$g_{mag}(x, y) = \sqrt{\left(\frac{\mathrm{d}}{\mathrm{dx}} f(x, y)\right)^2 + \left(\frac{\mathrm{d}}{\mathrm{dy}} f(x, y)\right)^2} \tag{3.13}$$

Die Gradientenorientierung ist definiert als

$$g_{ori}(x, y) = \tan^{-1}\left(\frac{\frac{\mathrm{d}}{\mathrm{dy}} f(x, y)}{\frac{\mathrm{d}}{\mathrm{dx}} f(x, y)}\right) \tag{3.14}$$

Ausgehend von diesen Berechnungen wird das Bild in sich überlappende, gleich große *HOG-Blöcke* unterteilt. Jeder Block beinhaltet eine feste Anzahl von $\xi \times \xi$

Zellen, die jeweils die Größe $\eta \times \eta$ Pixel aufweisen. Die Überlappung der Blöcke in x- sowie in y-Richtung beträgt $\frac{\xi\eta}{2}$. Abbildung 3.11 (e) zeigt ein Beispiel mit den Parametern $\xi = 2$ und $\eta = 8$. Die Mittelpunkte der HOG-Blöcke werden in dieser Abbildung in Form von roten Punkten visualisiert, die Mittelpunkte der Zellen sind durch blaue Punkte gekennzeichnet.

Für jeden HOG-Block werden nun die zuvor berechneten Gradienten betrachtet. Die Gradientenstärken werden mit einer zweidimensionalen Gaußfunktion f_g (Gleichung 3.15) gewichtet.

$$f_g(x, y) = \exp\left(-\left(\frac{(x - x_0)^2}{2\sigma^2} + \frac{(y - y_0)^2}{2\sigma^2}\right)\right) \tag{3.15}$$

Die Gaußfunktion hat ihren Mittelpunkt (x_0, y_0) im Zentrum des Blockes und die Varianz $\sigma = \frac{1}{2}\xi\eta$ entspricht einer halben Block-Breite. Die Gaußfunktion ist beispielhaft für den ersten HOG-Block $(0, 0)$ in Abbildung 3.11 (f) dargestellt. Weiterhin ist für den Block $(2, 2)$ die gewichtete Gradientenstärke abgebildet. Durch die Multiplikation mit der Gaußfunktion werden Gradientenstärken zum Rand eines Blockes abgeschwächt.

Der nächste Schritt ist die Generierung eines dreidimensionalen Histogramms für jeden HOG-Block. Das Histogramm hat die Auflösung $\xi \times \xi \times \beta$, wobei β die Auflösung der Gradientenorientierung bezeichnet. Dieses Histogramm wird nun mit den $\xi\eta \times \xi\eta$ Pixeln des Blockes befüllt. Jeder Pixel wird in Abhängigkeit zu seiner relativen Position im Block und seiner Gradientenorientierung in das Histogramm eingetragen. Der Eintrag in das Histogramm ist ein Wert, welcher hier Gewicht genannt wird. Das Gewicht ist durch die zuvor gewichtete Gradientenstärke im jeweiligen Pixel gegeben. Abbildung 3.12 zeigt beispielhaft ein solches Histogramm mit den Werten $\xi = 2$ und $\beta = 9$.

Einträge in das Histogramm werden über trilineare Interpolation vorgenommen. Ein Eintrag kann dabei entsprechend seiner Lage und Orientierung zu mehreren Histogramm-Klassen (engl. *bins*) beitragen. Im Anhang A wird beschrieben wie lineare Interpolation bei eindimensionalen Histogrammen funktioniert. Trilineare Interpolation lässt sich daraus ableiten. Bei der Interpolation von Einträgen ist zu beachten, dass die Gradientenorientierung ein Winkel ist und den Wertebereich $[0, 360[\in \mathbb{R}$ hat. Orientierungen, welche an den Grenzbereichen dieses Wertebereichs liegen, sollen entsprechend zur jeweils entgegengesetzten Histogramm-Klasse beitragen. Dalal [Dal06] bietet im Anhang auf den Seiten 117-118 eine theoretische Beschreibung der trilinearen Interpolation an.

Das resultierende Histogramm pro Block wird als Vektor betrachtet. Dieser Vektor wird schließlich einer Normalisierung unterzogen. Verwendet wird hier die $\mathcal{L}_2$-*Hys*-Normalisierung, welche zunächst eine $\mathcal{L}_2$-Normierung durchführt. Danach werden alle Komponenten des Vektors, die den Schwellwert von 0.2 überschreiten auf 0.2 gesetzt. Im Anschluss wird die $\mathcal{L}_2$-Normierung erneut durchgeführt.

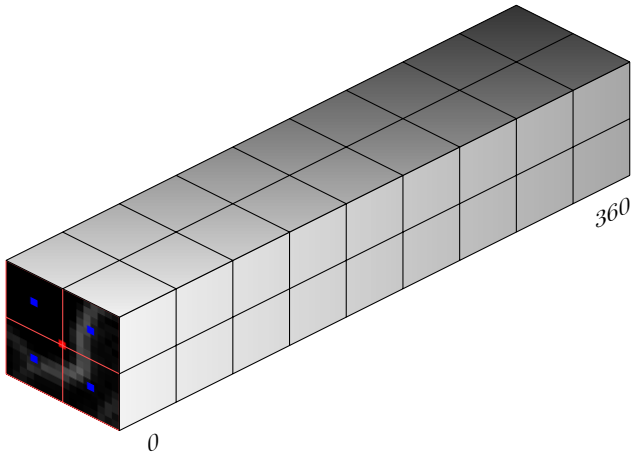


Abbildung 3.12: Histogramm eines HOG-Blocks

Berechne Gradientenstärke und Gradientenorientierung	
Unterteile das Eingabebild in n HOG-Blöcke der Größe $\xi\eta \times \xi\eta$ mit Überlappung $\frac{\xi\eta}{2}$	
Für alle HOG-Blöcke, $i = 0, 1, ..., n$	
	Erstelle das 3D-Histogramm H_i mit der Auflösung $\xi \times \xi \times \beta$
	Gewichte die Gradientenstärke mit einer 2D Gaußfunktion
	Befülle das Histogramm H_i mit allen Pixeln des HOG-Blocks gewichtet nach Gradientenstärke (Trilineare Interpolation)
	Normalisiere H_i ($\mathcal{L}_2$-*Hys*-Normalisierung)

Abbildung 3.13: Struktogramm zur Berechnung der Gradientenhistogramme

Diese Normalisierung bewirkt, dass ähnliche Gradientenstärken-Verhältnisse bei unterschiedlichen Gradientenstärke-Werten trotzdem zu ähnlichen Histogrammen führen. Das Struktogramm 3.13 fasst die einzelnen Schritte zur Histogrammberechnung zusammen.

Der Merkmalsvektor entsteht durch Konkatenation aller normalisierten Histogramme solcher HOG-Blöcke, die in einem Detektionsfenster enthalten sind. Für ein Detektionsfenster der Größe 64×128 und Parametern $\eta = 8$, $\xi = 2$ und $\beta = 9$ entsteht ein Merkmalsvektor der Dimension 3780. Eine Visualisierung dieses Merkmalsvektors ist in Abbildung 3.11 (g) abgebildet. Zusammenfassend lässt sich festhalten, dass das HOG-Merkmal die grobe Form eines Objektes kodiert. Dabei ist die Repräsentation durch die lokal grobe Ortsauflösung der Histogramme tolerant

gegenüber kleinen Veränderungen der Form. In Bezug auf das Sliding-Window-Paradigma lässt sich feststellen, dass die HOG-Blöcke für das gesamte Eingangsbild vorberechnet werden können. Einzelne Fenster können durch die Konkatenation aller jeweils enthaltener HOG-Blöcke ausgewertet werden.

3.3.2 Color Self-Similarity

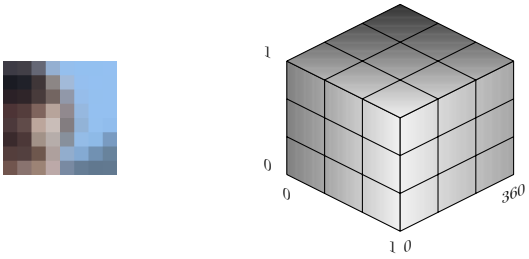
Ein weiteres Merkmal, welches zur Fußgängerdetektion in dieser Masterarbeit implementiert wurde ist das Merkmal der Color Self-Similarity (CSS) von Walk et al. [WMSS10]. Mit diesem Merkmal sollen Eigenschaften der Fußgänger bezüglich der auftretenden Farben kodiert werden. Farbwerte direkt zu verwenden sollte theoretisch nicht sinnvoll sein, denn jeder Fußgänger kann andere Kleidung tragen und auch die Hautfarbe kann variieren. Weiterhin werden die Bilder von Fußgängern mit unterschiedlichen Kameras aufgenommen und verschiedene Kameras produzieren unterschiedliche Farbwerte. Trotzdem haben Dollar et al. [DTPB09] gezeigt, dass bei den Integral Channel Features im Hue Kanal ein Peak im Kopfbereich zu verzeichnen ist. Für die Bereiche der Kleidung konnte dies aber nicht realisiert werden. CSS umgeht diese Limitierung dadurch, dass nicht Farbwerte direkt betrachtet werden, sondern Farbähnlichkeiten. Durch eine solche Statistik zweiter Ordnung gehen somit nicht die Werte direkt, sondern deren Differenzen in die Bewertung mit ein.

Die Farbähnlichkeit wird nach Walk et al. pro Fenster wie folgt berechnet. Zunächst wird das RGB-Bild in den HSV-Farbraum konvertiert (siehe Abbildung 3.14 (a-b)). Das Fenster wird in quadratische Zellen der Größe $\zeta \times \zeta$ unterteilt. Abbildung 3.14 (c) zeigt die Unterteilung. Zu jeder Zelle wird ein dreidimensionales Farbhistogramm der Dimension $3 \times 3 \times 3$ erstellt. Das Histogramm befüllt sich ähnlich wie bei HOG durch die Pixel, die in der Zelle enthalten sind. Es wird auch die trilineare Interpolation verwendet. Es muss wieder darauf geachtet werden, dass der Hue Kanal ein Winkel ist und dementsprechend eine geeignete Randbehandlung durchgeführt wird. Werte, die am Rand liegen sollten jeweils auch zu der gegenüber liegenden Zelle im Histogramm beitragen. Abbildung 3.14 (f) zeigt das Histogramm.

Nun soll zu jeder Zelle die Farbähnlichkeit zu jeder anderen Zelle des Fensters berechnet werden. Das Maß, welches zur Berechnung der Ähnlichkeit herangezogen wird, ist die Histogramm-Schnittmenge (engl. *histogram intersection*). Die Histogramm-Schnittmenge d_{HI} berechnet sich für zwei Histogramme H und V durch

$$d_{HI}(H, V) = \sum_k \min(H(k), V(k)) \ , \tag{3.16}$$

(a) RGB **(b)** HSV **(c)** Zellen



(d) 8×8 Zelle und 3×3×3 HSV-Histogramm

Abbildung 3.14: Visualisierungen zum CSS-Merkmal

wobei k über alle Klassen des Histogramms iteriert. Diese Ähnlichkeit kann normalisiert werden, denn der maximale Wert $d_{max} = \zeta^2$ ist durch die Anzahl der Pixel innerhalb einer Zelle beschrieben. Somit erhält man das Ähnlichkeitsmaß

$$d_{CSS}(H, V) = \frac{d_{HI}(H, V)}{\zeta^2} \tag{3.17}$$

mit dem Wertebereich $[0, 1]$. Um alle Ähnlichkeiten aller Zellen zu jeder anderen Zelle abzubilden sind $n(n-1)/2$ Werte erforderlich, wobei n die Anzahl der Zellen eines Detektionsfensters ist. Diese $n(n-1)/2$ Werte bilden sogleich auch den zu berechnenden Merkmalsvektor des CSS-Merkmals. Für den Parameter $\zeta = 8$ und einem Detektionsfenster der Größe 64×128 werden für $n = 128$ Zellen 8128 Ähnlichkeiten berechnet. Abbildung 3.15 visualisiert für diesen Fall alle Ähnlichkeiten des Beispielbildes. Man erkennt, dass die Ähnlichkeiten der jeweiligen Körperteile sich in den Werten abbildet. Weiterhin wird auch die Ähnlichkeit des Hintergrunds beschrieben, welche jeweils durch die Gestalt des Fußgängers unterbrochen wird. Somit kodiert CSS die Farbähnlichkeit sowohl des Objekts als auch des Hintergrunds.

In der Originalpublikation von Walk et al. werden die Ähnlichkeiten für jedes Fenster neu berechnet. Abschnitt 4.2 beschreibt, wie durch Vorberechnungen die Verarbeitung deutlich beschleunigt wird.

Abbildung 3.15: Farbähnlichkeiten aller Zellen des Beispielbildes

Kapitel 4

Implementierung

In dieser Masterarbeit wurde eine eigenständige Implementierung des zuvor beschriebenen Fußgängerdetektionssystems erstellt. Bis auf das Lernverfahren (SVM) wurden alle nötigen Programme und Algorithmen implementiert. Es wurde nur an wenigen Stellen auf rudimentäre Methoden, wie z.B. das Laden und Speichern von Daten, auf Bibliotheken wie OpenCV und Boost zurückgegriffen. Eine eigenständige Lösung birgt die Herausforderung, dass alle Aspekte des Detektionssystems korrekt implementiert werden müssen, bevor gute Ergebnisse produziert werden können. Allerdings bietet es auch die Chance alle Aspekte zu überblicken und Raum für Verbesserungen identifizieren zu können. Das hier implementierte System wird unter einer Open Source Lizenz veröffentlicht. Andere Autoren können das System erweitern oder es für praktische Anwendungen nutzen.

Die Beschreibung einiger Teilaspekte der Implementierung des Fußgängerdetektionssystems ist Aufgabe dieses Kapitels. Zunächst wird ein grober Überblick über die realisierten Programme und Bibliotheken gegeben. Weiterhin soll die in dieser Masterarbeit entwickelte, effiziente Auswertung des CSS-Merkmals beschrieben werden. Zuletzt wird auf die objektorientierte Merkmalsgenerierung eingegangen.

4.1 Überblick über realisierte Programme und Bibliotheken

Die Implementierung eines Fußgängerdetektionssystems lässt sich in verschiedene Softwaremodule unterteilen. Auch in dieser Masterarbeit wurde von dieser gängigen Praxis Gebrauch gemacht. Der Vorteil liegt darin, dass Module abgeschlossene Einheiten bilden, die mehrfach verwendet und einzeln getestet werden können. Alle relevanten Programme und Bibliotheken sind in Tabelle 4.1 aufgelistet. Zu jedem Modul ist jeweils der Name, die Programm-Bezeichnung, die Programmiersprache und die Anzahl Zeilen an Quelltext (engl. *lines of code*) angegeben. Bei der Anga-

Name	Programm	Sprache	LOC
Pedestrian Detector	pd	C++	744
Learn-Data-Generator	ldg	C++	1397
FeatGen Bibliothek		C++	2281
Mean-Shift Bibliothek		C++, Matlab	354, 296
Automatisierte Lernprozedur	learnpd.sh	Shell-Skript	73
Automatisierte Evaluation	evalpd.sh	Shell-Skript	64
SVM-Random-Data-Selector	svmrandselect	C++	289
Skalenraum Bibliothek		C++	129
HOG-Demo Matlab	rhog_demo.m	Matlab	634

Tabelle 4.1: Erstellte Programme und Bibliotheken

be der Zeilen sind keine Kommentare oder Leer-Zeilen enthalten. Nachfolgendend wird jedes Modul kurz vorgestellt.

Pedestrian Detector (pd) Dieses Programm realisiert die Multiskalen-Detektion der Fußgänger. Es wird auf die Klassen und Funktionen der FeatGen, Mean-Shift und Skalenraum Bibliotheken zurückgegriffen. Der Pedestrian Detector selbst implementiert die Suche nach Detektionen und die Verarbeitung von Eingangs- und Ausgangsdaten. Das Programm wird per Command-Line gesteuert und ein typischer Aufruf, sieht wie folgt aus:

```
./Release/pd -f rhog6 -m /path/to/model/rhog6.model -v imagefile.png
```

Dieser Aufruf würde auf dem Bild `imagefile.png` mit dem Modell `rhog6.model` unter Verwendung der Merkmals-ID `rhog6` Fußgänger erkennen und sie wegen der Option `-v` in einem Anzeigefenster darstellen. Alternativ kann über die Option `-d /path/to/images/` auch ein Ordner mit Bilddateien angegeben werden. In diesem Fall würde dann der Detektor auf allen Bildern des Ordners arbeiten und für jedes Bild eine Ausgabe produzieren.

Learn-Data-Generator (ldg) Dieses Programm ist dafür zuständig, aus den Bilddaten des Trainingsdatensatzes Trainingsdaten für das SVM-Verfahren zu produzieren. Dies beinhaltet die Extraktion der positiven Beispiele aus den Vollbilddaten, die zufällige Selektion negativer Beispiele und die Retraining-Prozedur. Alle Bilddaten werden dabei unter Verwendung der FeatGen Bibliothek in Merkmalsvektoren umgewandelt. Die Ausgabe dieses Programms ist eine große Textdatei in der im libSVM-Format Merkmalsvektoren enthalten sind. Diese Trainingsdatei kann dann an das SVM-Programm zum Training weitergegeben werden. Der

Aufruf des Learn-Data-Generator Programms hat zahlreiche Parameter und sollte deshalb automatisiert erfolgen. Das Shell-Skript learnpd.sh realisiert dies.

FeatGen Bibliothek Diese Bibliothek implementiert eine generische Lösung zur Merkmalsextraktion auf Basis eines Sliding-Window-Ansatzes. Es wird die Merkmals-Generierung abstrahiert und mit Spezialisierungen für die jeweils verwendeten Merkmale gearbeitet. Dabei wir auf modulare objektorientierte Programmierung Wert gelegt. Abschnitt 4.3 beschreibt die Struktur der Bibliothek näher.

Mean-Shift Bibliothek Dieses Modul beinhaltet alle Funktionen und Testfälle zum Mean-Shift Algorithmus. Er wurde sowohl in Matlab als auch in C++implementiert. Für das Gesamtsystem wird nur die C++-Variante verwendet. Die Matlab-Version dient der Anschauung.

Automatisierte Lernprozedur In diesem Shell-Skript wird die komplette Lernprozedur mit initialem Training und mehrfachem Retraining definiert. Dies hat den Vorteil, dass viele Parameterkombination auf einfache und automatisierte Weise ausprobiert werden können. Dabei wird der Learn-Data-Generator mehrfach aufgerufen und das SVM-Programm LIBLINEAR [FCH+08] genutzt. Auch das svmrandselect Programm wird aufgerufen um bei Engpässen des Arbeitsspeichers zufällíg Instanzen aus den Trainingsdaten auszuwählen. Beispielhaft für einen Detektor befindet sich die Ausgabe des Shell-Skriptes in Anhang B.

Automatisierte Evaluation Auch für die Evaluation existiert ein Shell-Skript. Es wird der Pedestrian Detector für den jeweils gewünschten Datensatz mit einem gewünschten Modell aufgerufen:

```
bash evalmodel.sh rhog6 /path/to/model/rhog6.model InriaTest
```

Auf Wunsch wird auch das zugehörige Matlab-Skript zur Generierung der in Kapitel 5 beschriebenen Evaluationskurven aufgerufen. Die Evaluationskurven selbst werden durch die Matlab-Toolbox von Dollar [Dol12] realisiert.

SVM-Random-Data-Selector (svmrandselect) Bei diesem Programm handelt es sich um eine generische Lösung, die allgemein für Trainingsdaten im libSVM-Format gültig ist. Wenn Trainingsdaten größer werden als der verfügbare Arbeitsspeicher, dann kann die Trainingsprozedur entweder fehlschlagen oder sehr lange dauern, weil das Betriebssystem dann mit dem Swap-Bereich der Festplatte arbeitet. Dieses Programm umgeht dieses Problem, indem es berechnet, wie viel Arbeitsspeicher von dem übergebenen Datensatz benötigt wird. Falls der Bedarf den tatsächlich physikalisch vorhandenen Arbeitsspeicher übersteigt, werden per

Zufall so viele Instanzen aus dem Trainingsdatensatz ausgewählt, wie in den Arbeitsspeicher passen. Aufgerufen wird das Programm wie folgt:

```
./Release/svmrandselect svmfile svmoutfile
```

In diesem Fall würden aus dem svmfile Datensatz wie oben beschrieben je nach Speicherbedarf Daten ausgewählt, und in der Datei svmoutfile gespeichert werden. Die Option `-l class-id` belässt die Instanzen der Klasse mit dem Bezeichner `class-id` unverändert. Dies wird zum Beispiel für das Training des hier implementierten Systems genutzt, weil der Datensatz deutlich mehr negative als positive Beispiele enthält. Hier wird nur aus den negativen Beispielen per Zufall ausgewählt, damit die Positiven nicht unterrepräsentiert werden. Das Programm kann aber auch allgemein dazu genutzt werden per Zufall Instanzen aus Trainingsdaten auszuwählen. So gibt es einen Parameter `-p n`, wobei `n` $\in \mathbb{N}$. Wenn dieser Parameter gesetzt ist, wird nicht der Arbeitsspeicher betrachtet sondern eine feste Anzahl `n` per Zufall aus den Trainingsdaten ausgewählt. Es ist darauf zu achten, dass nicht zu wenig Daten ausgewählt werden. Wenn zum Beispiel 50% der Daten verworfen werden würden, dann wäre das resultierende erlernte Modell unter Umständen schlechter als jenes mit 100%. Deshalb wird hier empfohlen immmer ausreichend Arbeitsspeicher vorzuhalten (8 GB).

Skalenraum Bibliothek In dieser Bibliothek befinden sich Funktionen, die einen Skalenraum beschreiben und bilden können. Es ist ein Visualisierungsprogramm pyramidviz enthalten, welches die Arbeitsweise der Funktionen zeigt. Es wird bei der Skalierung auf bilineare Interpolation der OpenCV-Bibliothek zurückgegriffen.

HOG-Demo Matlab Dieses Projekt beinhaltet eine Implementierung des HOG-Merkmals in Matlab. Dies wurde nur zu Demonstrations-Zwecken erstellt. Die Berechnung des verwendeten HOG-Merkmals für das Gesamtsystem ist in der FeatGen Bibliothek enthalten.

4.2 Effiziente Berechnung der Color Self-Similarity

Das CSS-Merkmal wurde in Abschnitt 3.3.2 eingeführt. Die Berechnung dieses Merkmals ist aufwendig und es stellt sich die Frage, ob hier eine effizientere Methode der Auswertung existiert. Wird die Berechnung des CSS-Merkmals in Bezug auf ein einziges Detektionsfenster betrachtet, so lässt sich feststellen, dass die Berechnung durch die Ähnlichkeitsberechnung der einzelnen Zellen dominiert wird. Es müssen $n(n-1)/2$ Ähnlichkeiten in Form von Histogramm-Schnittmengen berechnet werden um alle Ähnlichkeiten eines Fenster zu bestimmen, wobei n die

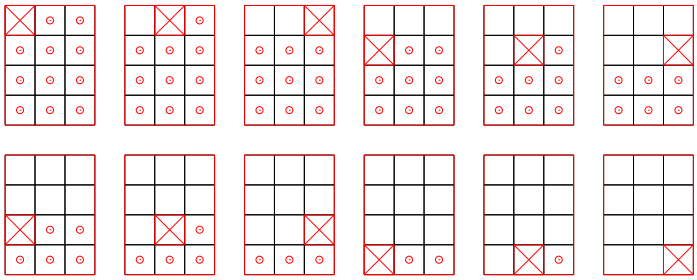


Abbildung 4.1: Ähnlichkeitsberechnungen für ein Detektionsfenster

Anzahl der Zellen eines Detektionsfensters angibt. Die Zellen eines Fensters haben die Größe $\zeta \times \zeta$ Pixel und n resultiert in Abhängigkeit zur Größe des Detektionsfensters $(n_{win} \times m_{win})$:

$$n = \left\lfloor \frac{n_{img}}{\zeta} \right\rfloor \left\lfloor \frac{m_{img}}{\zeta} \right\rfloor \tag{4.1}$$

Die Berechnung der Ähnlichkeiten für ein Fenster ist in Abbildung 4.1 an einem Beispiel visualisiert. Für jede Zelle des Detektionsfensters wird zu allen übrigen Zellen die Histogramm-Schnittmenge berechnet. Startet man mit der Zelle ganz links oben und wandert von links nach rechts und von oben nach unten, müssen immer nur die Nachfolger betrachtet werden. In der Abbildung visualisiert ein Kreis die Berechnung der Ähnlichkeit dieser Zelle zu der mit dem Kreuz gekennzeichneten Zelle des jeweiligen Fensters. Dies sind für das Beispiel mit 12 Zellen insgesamt 66 Berechnungen. Die Komplexität dieser Berechnung für ein einzelnes Fenster in Abhängigkeit zur Anzahl der Zellen n entspricht

$$\mathrm{O}\left(\frac{n(n-1)}{2}\right) . \tag{4.2}$$

Aus Gleichung 3.1 ist bekannt, wie viele Fenster w_{img} auf einem Bild bei bestimmten Parametern des Sliding-Window-Verfahrens überprüft werden. Diese neue Betrachtung führt zu der Erkenntnis, dass die Berechnungen für ein gesamtes Bild mit allen untersuchten Fenstern durch

$$\mathrm{O}\left(\frac{n(n-1)}{2} w_{img}\right) \tag{4.3}$$

bestimmt sind.

Auf den ersten Blick kann man die Anzahl der Ähnlichkeitsberechnungen nur reduzieren, indem man ζ groß wählt und dadurch n klein wird. Große Zellen verfehlen jedoch ab einer gewissen Größe die Aufgabe entscheidende Bereiche des Objektes oder des Hintergrunds zu integrieren.

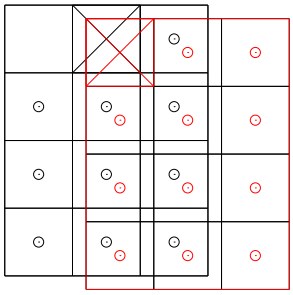
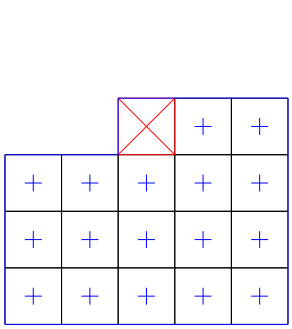


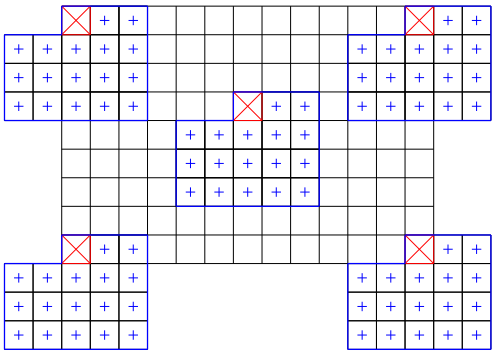
Abbildung 4.2: Redundanz durch Sliding-Window-Verfahren

Bei Betrachtung des Sliding-Window-Paradigmas ergibt sich eine weitere Verbesserungsmöglichkeit. Geht man davon aus, dass immer alle Fenster eines Bildes evaluiert werden müssen, so stellt man fest, dass Ähnlichkeitsberechnungen redundant ausgeführt werden. Um ein Beispiel zu nennen sei auf Abbildung 4.2 verwiesen. Dort ist angedeutet, wie Ähnlichkeiten zu der mit Kreuz gekennzeichneten Zelle nach dem Versatz um eine Zelle in einem anderen Fenster erneut berechnet werden. Die sieben Ähnlichkeiten, die sich in der Mitte überlappen, haben die gleichen Werte, wurden aber doppelt berechnet. Diese Redundanzen können durch geschickte Vorberechnungen für das gesamte Bild aufgelöst und somit ein entscheidender Geschwindigkeitsvorteil verzeichnet werden.

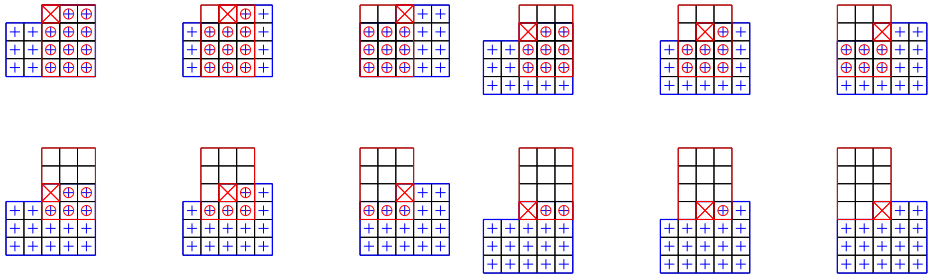
Zunächst betrachtet man ein ganzes Bild und unterteilt es in die $\zeta \times \zeta$ großen Zellen. Das Ziel ist es, für jede Zelle des Bildes alle relevanten Ähnlichkeiten vorab zu berechnen. Während der Sliding-Window-Verarbeitung soll dann jeweils nur aus diesen Vorberechnungen ausgelesen werden. Es stellt sich die Frage wie viele und welche Ähnlichkeiten vorberechnet werden müssen, damit dennoch alle Fenster eines Bildes gebildet werden können. Durch die begrenzte Größe eines Detektionsfensters ist der Einfluss einer Zelle nicht über das gesamte Bild verteilt, sondern es gibt einen minimalen Bereich, der den Einfluss begrenzt. Dieser Bereich wird hier Support oder auch Strukturelement genannt. Für das Beispiel aus Abbildung 4.1 mit einem Detektionsfenster, welches die Größe von 3×4 Zellen aufweist, ist der Support durch das Strukturelement in Abbildung 4.3 (a) gegeben. In Abhängigkeit zur Anzahl der Zellen des Detektionsfensters in x- und y-Richtung $c_x \times c_y$ ist der Support $sp_x \times sp_y$ mit $sp_x = 2c_x - 1$ und $sp_y = c_y$ beschrieben. Resultierend aus der zuvor beschriebenen Berechnung von links oben nach rechts unten interessieren in der ersten Zeile nur die Zellen nach der c_x-ten Zelle. Es kann nun für jede Zelle des Bildes ein Zwischenspeicher angelegt werden. In diesen Zwischenspeicher werden die Ähnlichkeiten zu allen Zellen, die im registrierten Strukturelement liegen gespeichert. Abbildung 4.3 (b) deutet diesen Prozess an. In der Abbildung sind die Strukturelemente nur beispielhaft für 5 Zellen angegeben. Die tatsächli-



(a) Strukturelement



(b) Vorberechnungen



(c) CSS-Merkmal aus Strukturelementen

Abbildung 4.3: Vorberechnung der Farbähnlichkeiten

che Berechnung muss für alle Zellen durchgeführt werden. Die Abbildung zeigt auch, dass Vorsicht an Randbereichen geboten ist. Hier wird der Speicher zwar angelegt, aber nicht beschrieben. Dies ist unproblematisch, weil in der späteren Verarbeitung keine der Zellen außerhalb des Bildes ausgelesen wird.

Es soll nun noch gezeigt werden, dass die Vorberechnungen tatsächlich ausreichen um alle möglichen Fenster des Bildes auszuwerten. Betrachtet wird zunächst ein einzelnes Fenster. Abbildung 4.3 (c) zeigt, dass die gleichen Ähnlichkeiten, die zuvor in Abbildung 4.1 durch Berechnung entstanden sind nun durch das Auslesen der entsprechenden Strukturelemente realisiert werden kann. Dieser Zusammenhang gilt für alle Fenster eines Bildes, da für alle Zellen Strukturelemente vorberechnet werden.

Die Berechnung der Histogramm-Schnittmengen hängt durch diese Veränderung nicht mehr von der Anzahl der evaluierten Fenster w_{img} eines Bildes ab, sondern viel mehr von der Anzahl der Zellen des Bildes c_{img} und der Größe des Strukturelementes:

$$\mathrm{O}\left((sp_x sp_y - c_x)\, c_{img}\right) = \mathrm{O}\left(2\, n\, c_{img}\right) \tag{4.4}$$

Die Anzahl der Histogramm-Schnittmengen-Berechnungen ist damit minimal. Die Größe des Strukturelements ist mit $2n$ überabgeschätzt, damit ein Vergleich mit Gleichung 4.3 möglich ist. Es muss noch berücksichtigt werden, dass nun zwar nicht für alle Fenster alle Histogramm-Schnittmengen berechnet werden, aber dennoch für alle Fenster das Auslesen des Zwischenspeichers anfällt, was wiederum quadratisch in n ist. Somit gelangt man schließlich zu einer Gesamtkomplexität von

$$\mathrm{O}\left(2\, n\, c_{img}\right) + \mathrm{O}\left(\frac{n(n-1)}{2} w_{img}\right) . \tag{4.5}$$

Die Komplexitätsklasse wird also nicht verändert, allerdings benötigt das Auslesen von Werten deutlich weniger Rechenoperationen als die Histogramm-Schnittmengen-Bildung. Abschnitt 5.7 der Evaluation zeigt, dass die Vorberechnung im Gesamtsystem einen Vorteil um Faktor 4 bringt. Diese Beschleunigung geschieht ohne Qualitätsverlust. Die Speicherkomplexität liegt bei

$$\mathrm{O}\left((sp_x sp_y - c_x)\, c_{img}\right) , \tag{4.6}$$

weil für jede Zelle jeweils alle Ähnlichkeiten zu den im Strukturelement enthaltenen Zellen gespeichert werden. Bei einem typischen Detektionsfenster von 64×128 Pixeln, einer Zellengröße $\zeta = 8$ und einem Eingabebild von 640×480 wird 8,859 MB Speicher benutzt, wenn jede Ähnlichkeit durch 4 Byte repräsentiert wird.

4.3 Objektorientierte Merkmalsberechnung

Die Berechnung der Merkmale wurde in eine eigene Bibliothek ausgelagert. Dies wurde notwendig, weil sowohl der Detektor, als auch das Training jeweils mit den gleichen Merkmalen arbeiten müssen. Weiterhin war es wichtig die Merkmale einzeln testen und optimieren zu können. Es wurde ein objektorientierter Entwurf realisiert, welcher in Form eines UML-Diagramms in Abbildung 4.4 skizziert ist. Die grundlegende Idee ist, dass man die Berechnung eines Merkmals im Kontext des Sliding-Window-Paradigmas abstrahiert. Dies ist durch die Klasse FeatGen realisiert. Diese abstrakte Klasse stellt ein Interface bereit. Folgender Beispiel-Code (C++) soll zeigen, wie mit diesem Interface gearbeitet wird:

```
1:> RHOGGen rhog;
2:> FeatGen* fg = &rhog;
3:> fg->setImage(img);
4:> fg->calcFullImageFeature();
...
5:> double* feature = fg->getWindowFeatureOnFullImageAt(i,j);
6:> int featureLength = fg->getWindowFeatLen();
...
7:> processFeature(feature,featureLength);
...
```

Die Sliding-Window-Verarbeitung erfordert es, dass zunächst ein Vollbild an den Algorithmus übergeben werden muss. Dies wird durch die öffentliche Methode *setImage(img:Mat*)* ausgedrückt. Ist das Bild übergeben, so wird die Methode *calcFullImageFeature()* aufgerufen. Dies bietet dem Algorithmus die Möglichkeit Berechnungen anzustellen, die für das gesamte Bild vorberechnet werden können. Die Entscheidung ob und was hier vorberechnet wird liegt bei den Spezialisierungen der FeatGen Klasse. Zum Beispiel werden bei der CSSGen Klasse nur die Farbhistogramme für alle Zellen des Bildes vorberechnet, während die Ähnlichkeitsberechnungen beim Aufruf der Methode *getWindowFeatureOnFullImageAt(...)* durchgeführt werden. Diese Methode ist im Übrigen ein weiterer zentraler Bestandteil der FeatGen Klasse. Es wird abstrahiert, dass ein Merkmalsvektor für eine zuvor definierte Fenstergröße an der Stelle (i, j) im Bild berechnet und zurückgegeben wird. Weiterhin muss das Merkmal auch die Länge eines Merkmalsvektors bestimmen können. Hierfür ist die Methode *getWindowFeatLen()* zuständig. Die Eigenschaften des Detektionsfensters werden zuvor bereits im Konstruktor festgelegt und in privaten Variablen gespeichert.

Die Komposition mehrerer Merkmale zu einem neuen Merkmal wird durch die Klasse FeatCombinator realisiert. Es kommt das Composite-Pattern [GHJV95] zum Einsatz. Nutzer der FeatGen Bibliothek können Kompositionen von FeatGen Objekten wie ein einzelnes Objekt behandeln. Dieses Design-Pattern ermöglicht es verschiedene Merkmale auf einfache Weise zu kombinieren und zu nutzen. Merkmale können einzeln instanziiert werden und dann über die Methode *pushFeatGen(fg:FeatGen)* kombiniert werden.

Durch das Design der FeatGen Bibliothek ist ein hohes Maß an Wiederverwendbarkeit gewährleistet. Beispielsweise definiert die CSSGenFast Klasse nur die für die Berechnung nötigen Methoden neu. Andere Methoden, wie etwa *genVisualization()* zur Visualisierung ändern sich nicht und die Implementierung der Oberklasse wird genutzt.

Nutzern der FeatGen Bibliothek wird über die Klasse FeatGenSelector ein möglichst einfacher Zugang zu einigen bereits instanziierten Merkmalen geben. In die-

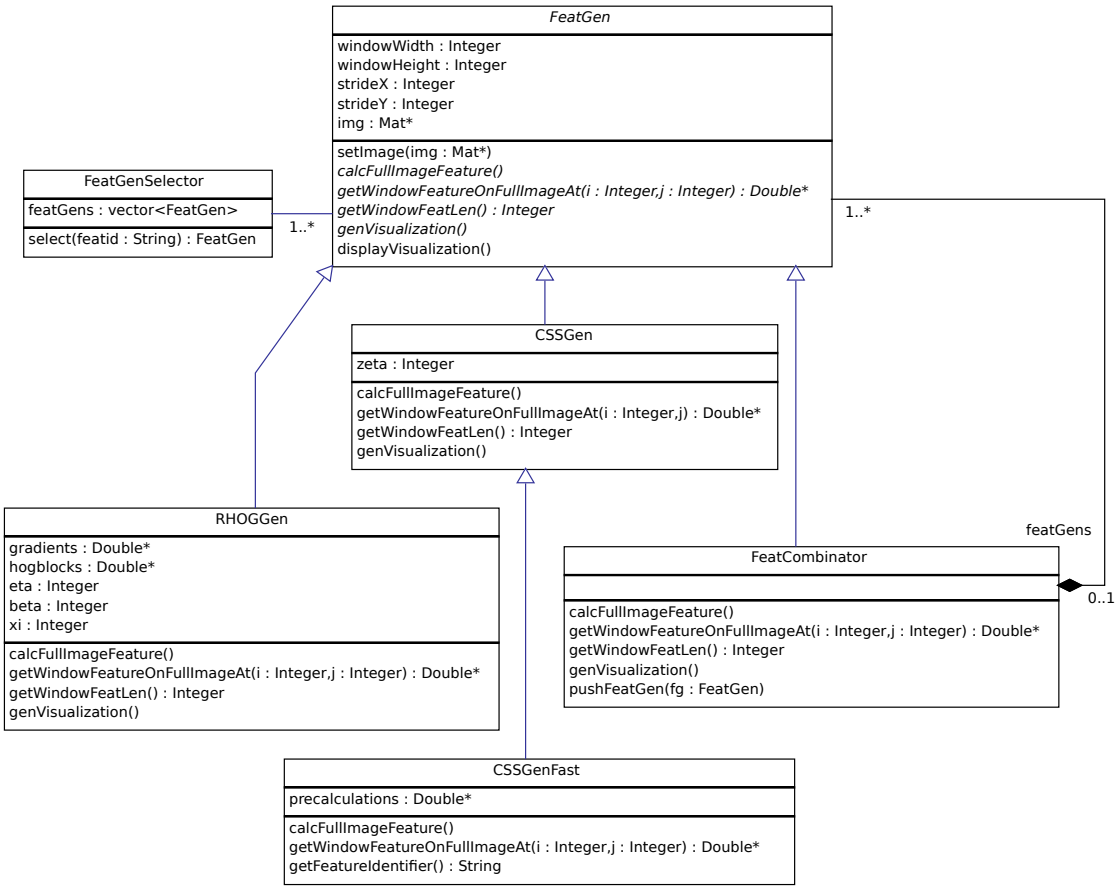


Abbildung 4.4: UML-Diagramm der FeatGen Bibliothek

ser Klasse werden mehrere Merkmale mit unterschiedlichen Parametern erstellt. Nutzer können über einen Identifikations-String, wie z.B. “rhog6”, ein Objekt der jeweiligen Klasse über die Methode *select(featid:String):FeatGen* anfordern. In dieser Klasse werden alle für die Fußgängererkennung verwendeten Merkmale definiert. Der Detektor und das Training können zur Laufzeit ohne erneutes kompilieren verschiedene Merkmale mit dem gleichen FeatGen Interface nutzen.

Kapitel 5

Evaluation

5.1 Evaluationsprotokoll

Die Evaluation des implementierten Fußgängerdetektionssystems geschieht nach dem gleichen Prinzip wie auch Dollar et al. [DWSP12] es zur Evaluation von Detektoren einsetzen. Dies hat den Vorteil, dass das Evaluationsprotokoll bereits in Form von Matlab-Code vorliegt und deshalb ein direkter und exakter Vergleich mit bereits evaluierten Detektoren angestellt werden.

Die Evaluation geschieht anhand mehrerer Testbilder aus verschiedenen Testdatenbanken. Zu jedem Testbild sind von Menschen erstellte Annotationen gegeben, welche die Position und Fläche der enthaltenen Fußgänger angeben. Diese Daten werden Ground Truth genannt. Ein zu evaluierendes Detektionssystem erhält die selben Testbildern ohne die menschlichen Annotationen und muss zu jedem Testbild selbst Annotationen in Form einer Textdatei generieren. Die Annotationen sehen dann beispielsweise wie folgt aus:

```
541 253 67 188 0.337299
219 355 38 107 0.0485812
820 165 58 162 0.826339
234 123 61 171 0.315234
```

Jede Zeile beinhaltet eine Detektion und das Format jeder Zeile ist durch

```
X Y Breite Höhe Entscheidungswert
```

gegeben. Die Evaluationsroutine vergleicht nun die vom Detektionssystem ausgegebenen Detektionen mit den menschlichen Annotationen und kann so die Detektionsqualität bestimmen. Um dies tun zu können, muss definiert werden wann Boundingboxen genügend überlappen, so dass eine Übereinstimmung festgestellt wird. Hierfür wird das PASCAL-Kriterium [EVGW+10] herangezogen. Es besagt,

dass zwei Boundingboxen genügend übereinstimmen, wenn ihre Fläche zu über 50% überlappt. Für die Boundingbox der Detektion B_{det} und der Ground-Truth B_{gt} ergibt sich die folgende Eigenschaft:

$$\frac{\text{area}\,(B_{det} \cap B_{gt})}{\text{area}\,(B_{det} \cup B_{gt})} > 0.5 \tag{5.1}$$

Alle detektierten Boundingboxen, die keine Übereinstimmung in den Ground-Truth Werten finden, werden als falsch-positiv eingestuft. Entsprechend werden alle Ground-Truth Boundingboxen, welche keine Übereinstimmung in den Detektionen des Detektors finden, als falsch-negativ gewertet. Es werden immer nur einzelne Übereinstimmungen gewertet. Dies kommt zum tragen, wenn mehrere detektierte Boundingboxen auf eine Ground-Truth-Annotation zutreffen. In diesem Fall wird die detektierte Boundingbox mit dem höchsten Entscheidungswert verwendet und alle anderen werden als falsch-positiv bewertet. Hier muss der Detektor also gute Non-Maxima-Suppression betreiben, damit dieser Fall nicht auftritt.

Die Qualität eines Detektors wird mit einer Evaluationskurve beschrieben, wobei die x-Achse falsch Positive pro Bild (engl. *false positives per image* (FPPI)) und die y-Achse die Miss-Rate angibt. Die Miss-Rate ist durch

$$\text{Miss-Rate} = \frac{FN}{TP + FN} \tag{5.2}$$

gegeben, wobei TP die richtig Positiven und FN die falsch Negativen bezeichnet. Die falsch Positiven pro Bild (FPPI) ergeben sich aus der Formel

$$\text{FPPI} = \frac{FP}{\text{Anzahl Testbilder}}\ , \tag{5.3}$$

wobei FP die falsch Positiven angibt. Die Evaluationskurven entstehen, wenn man verschiedene Schwellwerte für die Entscheidungswerte der Detektionen anlegt. Für jeden Schwellwert erhält man eine gewissen Anzahl FPPI und eine Miss-Rate. Trägt man diese Werte gegeneinander auf und verbindet sie, erhält man die beschriebenen Evaluationskurven. Dieses Maß ist interessant für praktische Anwendungen, weil oft die Anforderung nach einer bestimmten maximal tolerierbaren FPPI-Rate besteht.

Das Finetuning des Detektionssytems ist keine triviale Aufgabe. Zum einen besteht die Herausforderung, dass es eine Reihe von Parametern gibt, die die Detektionsqualität beeinflussen. Zum anderen ist die Trainings- und Testprozedur sehr zeitaufwändig. In den folgenden Abschnitten wird auf die entscheidenden Parameter eingegangen um gute Ergebnisse zu erzielen. Weiterhin wird die Erweiterung der Standardvariante um das CSS-Merkmal evaluiert. Schließlich wird die Laufzeit des implementierten Detektionssystems mit unterschiedlichen Parametern evaluiert.

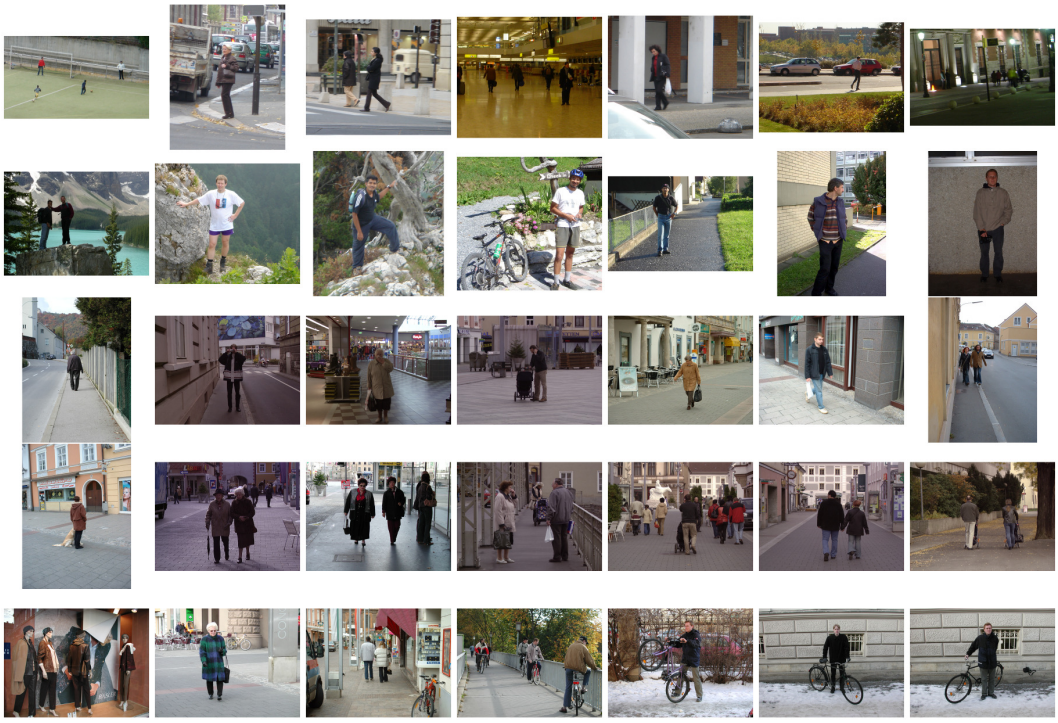
5.2 Testdaten

Zur Evaluation wurden die Testbilder des INRIA Personen-Datensatzes (*InriaTest*) [DT05] und des TudBrussels Datensatzes (*TudBrussels*) [WWS09] verwendet. InriaTest beinhaltet 288 Fotos unterschiedlicher Größe, Ausleuchtung und Szenerie. Abbildung C.8 (a) zeigt Beispiele. Diese Testbilder sind von den zuvor für das Training verwendeten Bildern getrennt. In einigen wenigen Bildern des Testdatensatzes sind Personen oder Szenen zu sehen, die auch im Trainingsdatensatz enthalten sind. Dies resultiert daraus, dass der Datensatz aus einer Kollektion von Urlaubsfotos entstanden ist und dann in manchen Fällen aus einer Foto-Reihe jeweils Bilder für Training- und Test-Datensatz verwendet wurden. Eine ungewöhnliche Begebenheit des Datensatzes ist auch, dass die Bilder 276–281 sowie 200–268 die selben zwei Personen in unterschiedlichen Posen zeigen. Weiterhin befinden sich diese Personen auf, vor oder hinter einem Fahrrad. Vermutlich wurden die Bilder hinzugefügt um auch stark unterschiedliche Posen von Personen zu testen.

Die Bilder des TudBrussels Datensatzes sind anders entstanden, und er ist auf die konkrete Anwendung eines Fußgängerdetektionssystems im Kontext der Fahrerassistenz ausgerichtet. Es wurde ein Auto mit einem Kamerasystem ausgestattet und alle Bilder des Datensatzes sind eine Bildfolge einer Fahrt durch die belgische Hauptstadt Brüssel. Der Datensatz beinhaltet 508 Bilder der Auflösung 640×480 Pixel mit 1498 Annotationen. Abbildung C.8 (b) zeigt Beispiele. Die Fußgänger sind in sehr kleiner Größe schon ab 50 Pixel Höhe annotiert und zeigen variierende Posen. Es gibt Verdeckungen durch Autos und andere Objekte. Die Evaluation des Detektionssystems auf einem vom Trainingsdatensatz völlig unterschiedlichen Datensatz gibt einen Eindruck, wie gut das System generalisiert oder ob es zu Overfitting neigt, also sich zu sehr auf das Trainingsmaterial spezialisiert.

5.3 Merkmale

Zunächst wird untersucht welchen Einfluss die Merkmale auf das Ergebnis des Gesamtsystems haben. In dieser Masterarbeit wurde HOG von Dalal und Triggs sowie CSS von Walk et al. implementiert. Abbildung 5.2 zeigt die Ergebnisse sowohl auf dem InriaTest als auch auf dem TudBrussels Datensatz unter Verwendung der zuvor beschriebenen Evaluationskurven. Die Bezeichner rhog6 und rhog6css8comb6 werden im Abschnitt 5.4 über Parameterkombination beschrieben. Die Achsen dieser Graphen werden traditionell logarithmisch dargestellt um relevante Bereiche der Kurve hervorzuheben. Die Legende der Evaluationsgrafiken zeigt die Miss-Rate in Prozent am Referenzpunkt bei 10^{-1} FPPI an. Von der Anzahl der Bilder im Datensatz hängt ab, wie vielen falschen Positiven dieser Referenzpunkt entspricht (siehe Gleichung 5.3). Bei InriaTest entsprechen 10^{-1} FPPI ca. 29 falsch Positiven



(a) InriaTest (35 von 288 Bildern)

(b) TudBrussels (35 von 508 Bildern)

Abbildung 5.1: Bilder aus beiden Testdatensätzen

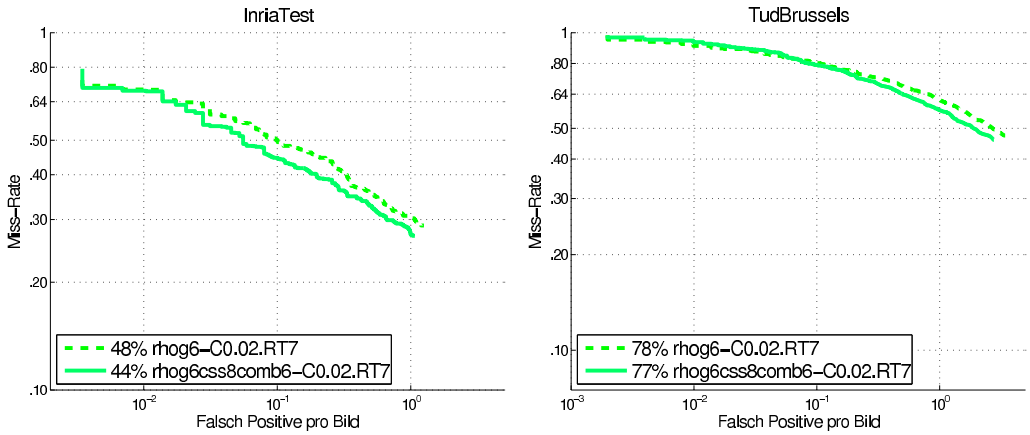


Abbildung 5.2: Vergleich HOG gegen HOG und CSS

und bei TudBrussels ca. 51. Der Detektor rhog6 erkennt an diesem Referenzpunkt 294 von 566 Personen bei InriaTest korrekt und 329 von 1498 Personen bei TudBrussels. Abgesehen vom Referenzpunkt ist auch der Verlauf der Kurve interessant und wichtig. Je flacher die Kurve ist, desto besser ist der Detektor. Diese Kurven zeigen, dass beide Datensätze herausfordernd sind und noch viel Raum für Verbesserung gegeben ist.

Die Kombination des HOG-Merkmals mit dem CSS-Merkmal ist mit dem Detektor rhog6css8comb6 angegeben. Man erkennt, dass die Kurve bei beiden Datensätzen bessere Ergebnisse anzeigt als jene von rhog6. Dieses Ergebnis bestätigt, dass das CSS-Merkmal tatsächlich hilfreiche Information für die Klassifikation von Fußgängern bereitstellt.

5.4 Parameterkombinationen

In Kapitel 3 wurde das Fußgängerdetektionssystem im Detail beschrieben. An vielen Stellen wurden Parameter eingeführt. Tabelle 5.1 listet alle für die Evaluation relevanten Parameter auf. Diese Parameter haben einen großen Einfluss auf die Detektionsqualität des Gesamtsystems. Um gute Ergebnisse zu erzielen müssen systematisch die Werte der Parameter optimiert werden. Dabei kann in der Regel nicht davon ausgegangen werden, dass die Parameter unabhängig voneinander sind. Das macht die Optimierung um so schwieriger. Jede Parameterkombination führt dabei zu einer Instanz des Detektionssystems, welche hier auch einfach als Detektor bezeichnet wird. Den Detektoren werden Namen gegeben, die nach folgender Regel gebildet werden:

Parameter	Beschreibung	Bereich
ξ	$\xi \times \xi$ HOG-Block	Merkmale
η	$\eta \times \eta$ Pixel pro Zelle (HOG)	Merkmale
ζ	$\zeta \times \zeta$ Pixel pro Zelle (CSS)	Merkmale
β	Anzahl Histogramm-Klassen für die Gradientenorientierung (HOG)	Merkmale
s_{shrink}	Skalierungsfaktor des Skalenraumes	Detektor
s_x, s_y	Versatz des Sliding-Windows	Detektor
ρ	Vektor der Basis-Unsicherheiten (Varianzen) für die NMS	Detektor
RT	Anzahl der Retraining-Phasen	Klassifikator
C	C-Parameter der SVM	Klassifikator

Tabelle 5.1: Parameter des Fußgängerdetektionssystems

Detektor–ID	ξ	η	β	ζ	s_{shrink}	s_x, s_y	ρ
rhog6	2	6	9		0.93301	6,6	$(4, 8, \log(1.6))^{\mathrm{T}}$
rhog8	2	8	9		0.93301	8,8	$(4, 8, \log(1.6))^{\mathrm{T}}$
rhog6css8comb6	2	6	9	8	0.93301	6,6	$(4, 8, \log(1.6))^{\mathrm{T}}$

Tabelle 5.2: Parameterkombinationen (Detektoren)

```
[Detektor-ID]-C[SVM C]-RT[Anzahl Retraining-Phasen]
```

Beispiele:

```
rhog6-C0.02-RT7
rhog8css6comb6-C0.001-RT1
```

In den Folgenden Abschnitten werden einige entscheidende Teilaspekte untersucht. Tabelle 5.2 führt alle untersuchten Detektor-IDs ein.

5.4.1 Zellengröße HOG

Die Zellengröße des HOG-Merkmals ist ein Parameter der in Abhängigkeit zu dem zu detektierenden Objekt steht. Dalal und Triggs haben dies bereits gezeigt [DT05]. Der Grund ist, dass eine Zelle jeweils einen Bereich im Gradientenbild abbilden sollte, so dass die Form des Objektes ausreichend gut aufgenommen wird. Ist die Zellengröße zu groß, dann gehen Details verloren. Ist sie zu klein ist keine generelle Form für die gesamte Objektklasse erkennbar. Für Fußgänger stellt sich heraus, dass Zellengrößen von 6 oder 8 gute Ergebnisse erzielen, weil zum Beispiel die Beine eines Fußgängers in einem Detektionsfenster von 64×128 ungefähr 6-8

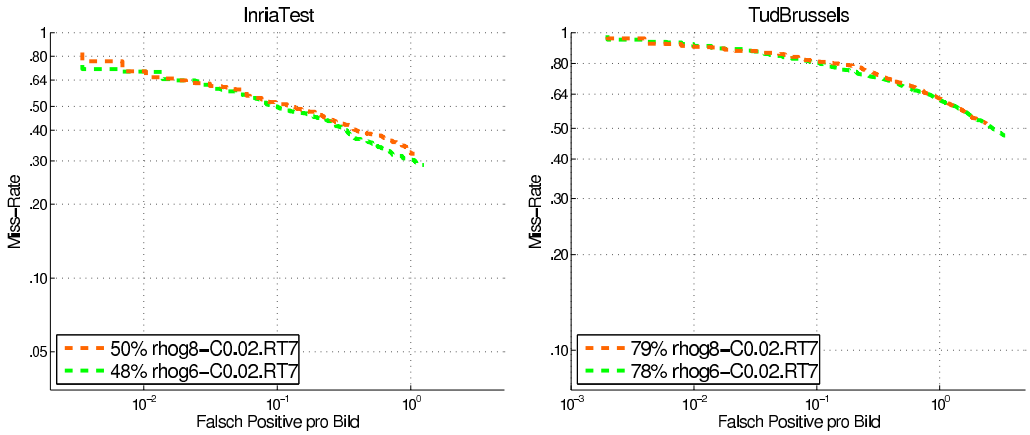
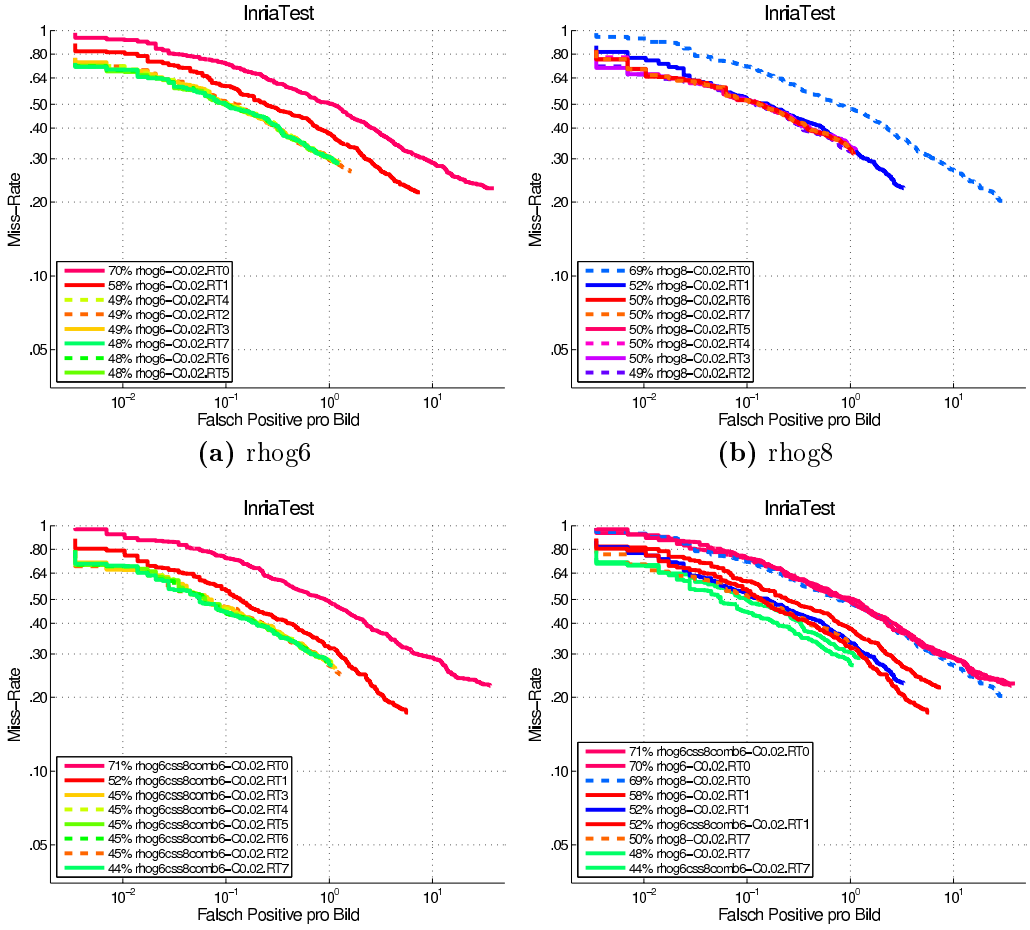


Abbildung 5.3: Zellengröße

Pixel breit sind. Diese Größe ist also vom Objekt abhängig. Anders als in [DT05] wird hier die Detektionsqualität des Gesamtsystems auf Vollbildern ausgewertet und es stellt sich die Frage ob die Werte 6 bis 8 auch hier Unterschiede bringen. Abbildung 5.3 zeigt, dass dem so ist und dass die Zellengröße von 6 bei beiden Datensätzen die besseren Ergebnisse zeigt.

5.4.2 Anzahl der Retraining-Phasen

Eine in der Literatur kontrovers diskutierte Thematik ist die Frage, wie viele Retraining-Phasen eingesetzt werden sollten um beste Ergebnisse zu erzielen. Dalal und Triggs gaben an, dass sie ein bis zwei Retraining-Phasen für sinnvoll halten und danach keine weiteren Verbesserungen sehen konnten. Walk et al. [WMSS10] dagegen sagen, dass viele Retraining-Phasen nötig sind und die Detektionsqualität nur gesteigert werden kann. In dieser Masterarbeit wurde diese Frage auch behandelt. Für den Autor beantwortet sich die Frage so, dass so viele Retraining-Phasen nötig sind, bis nur noch wenige (ca. < 30) schwere Negative gefunden werden. Man kann keine pauschale Anzahl empfehlen. Es hängt davon ab, wie viele schwere Negative jeweils noch gefunden werden. Um diese Annahme zu belegen, kann man Abbildung 5.4 betrachten. Man stellt fest, dass die mehrfache Iteration ab einer gewissen Anzahl keine zusätzlichen Vorteile bringt. In allen hier untersuchten Fällen war diese Konvergenz immer dann zu beobachten, wenn nur noch wenige schwere Negative gefunden wurden. Bis zu diesem Punkt hilft die mehrfache Iteration allerdings immer. Insofern kann ich beiden Autorengruppen Recht geben, denn gerade bei HOG stellte ich auch fest, dass nach zwei Iterationen meist eine Konvergenz eintrat, bei anderen Merkmalskombinationen waren mehrfache Iterationen nötig.



(a) rhog6 **(b)** rhog8

(c) rhog6css8comb6 **(d)** Verschiedene

Abbildung 5.4: Auswirkungen des mehrfachen Retrainings

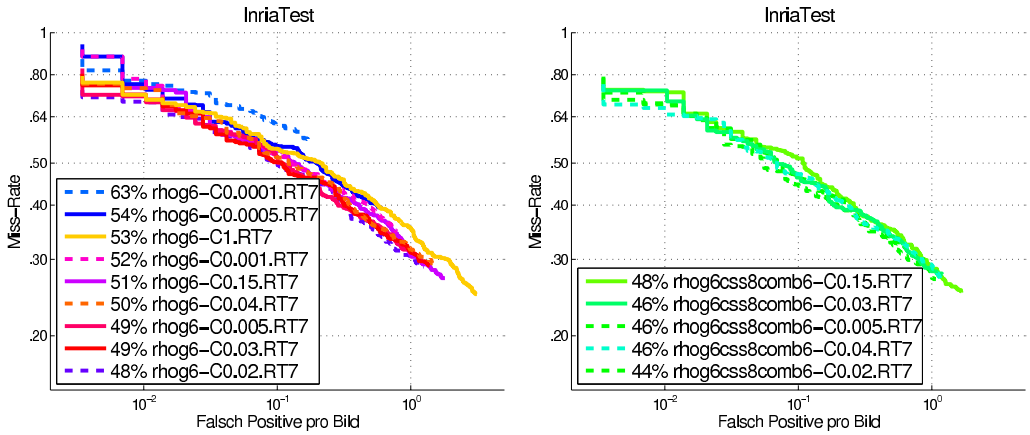


Abbildung 5.5: SVM C-Parameter

5.4.3 SVM C-Parameter

Lineare Support Vector Machines haben nur einen Parameter. Dieser Parameter ist die C-Variable, welche in Gleichung 3.9 zwischen den additiven Termen des Minimierungsproblems gewichtet. Dieser Wert muss experimentell optimiert werden. Normalerweise geht man so vor, dass man zunächst C auf die Werte $1.0{\times}10^{-k}$ mit einigen $k \in \mathbb{Z}$ überprüft. Falls sich in bestimmten Bereichen von k gute Ergebnisse zeigen, kann man die Auflösung erhöhen und dort weiter suchen. Bei dem hier implementierten System kann das Lernverfahren für einen Detektor mit mehrfachen Retraining und das Testen viel Zeit (bis zu einigen Stunden) in Anspruch nehmen. Eine weitere Schwierigkeit ist, dass der Parameter neu optimiert werden muss sobald sich etwas am System verändert. Der C-Parameter sollte deshalb möglichst erst ganz am Ende der Arbeit an einem Detektionssystem optimiert werden. Abbildung 5.5 zeigt die Detektionsqualität bei verschiedenen C-Werten. Man erkennt, dass Werte im Bereich $1.0{\times}10^{-2}$ gute Ergebnisse zeigen, wobei 0.02 die beste Evaluationskurve liefert. Es fällt weiterhin auf, dass der C-Parameter großen Einfluss auf die Detektionsqualität hat.

5.5 Vergleich mit dem Stand der Technik

In diesem Abschnitt werden die besten Detektoren dieser Masterarbeit in Relation zum aktuellen Stand der Technik dargestellt. Der aktuelle Stand der Technik richtet sich dabei nach der Evaluation von Dollar et al. [DWSP12]. Dollar et al. haben einige Detektoren von den jeweiligen Autoren optimiert erhalten und evaluiert. Einige der Detektoren sind in Kapitel 2 beschrieben.

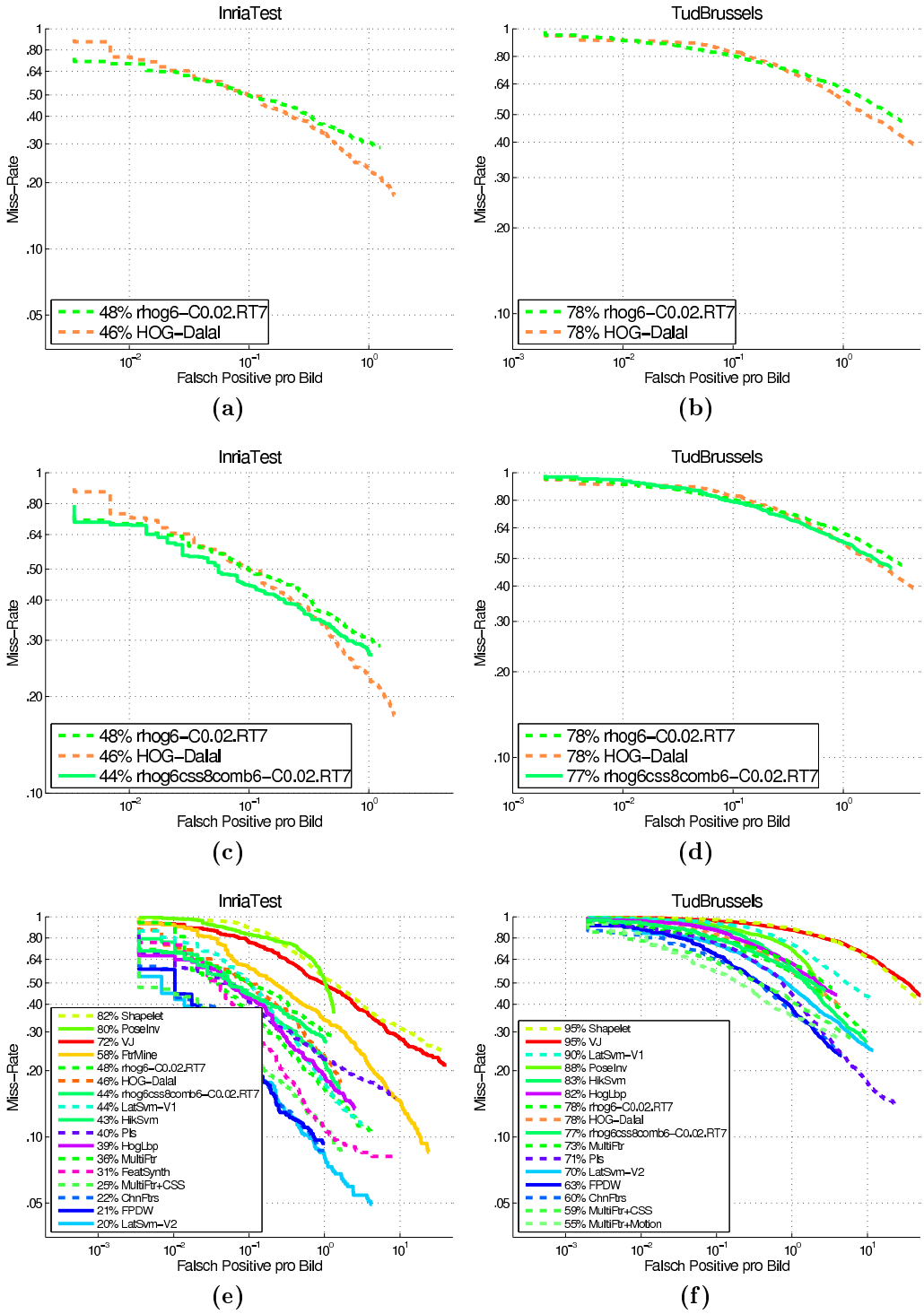


Abbildung 5.6: Vergleich mit dem Stand der Technik

Abbildungen 5.6 (a) und (b) zeigen den direkten Vergleich mit der Referenz von Dalal und Triggs auf beiden Datensätzen. Die Referenz von Dalal und Triggs wird mit HOG-Dalal angegeben. Die Kurven sind sehr ähnlich und der Unterschied beim Referenzpunkt beträgt nur 2 %. Gegenüber der Referenz sind kleine Unterschiede zu erkennen, die auf die völlig getrennte Implementierung und Optimierung zurückzuführen sind. Der Unterschied bei 10^0 FPPI, also 288 falsch Positive beim InriaTest-Datensatz, ist etwas deutlicher ausgeprägt. Bei einer solch hohen Anzahl von falsch Postiven (ca. 288 auf 288 Bilder) können kleine Details, wie etwa der benutzte Versatz des Sliding-Window, die Skalenraumauflösung oder Details aus dem Training eine Rolle spielen. In der Evaluation von Dollar et al. wurden leider nicht alle Parameter der Referenz angegeben und es ist anzunehmen, dass Dalal und Triggs den Detektor noch nach ihrer originalen Publikation in den Parametern für die Evaluation optimiert haben. Abbildungen 5.6 (c) und (d) zeigen, dass die Kombination mit dem CSS-Merkmal besser abschneidet als die Referenz HOG-Dalal. In den Abschnitten (e) und (f) der Abbildung 5.6 ist der Vergleich mit allen von Dollar et al. evaluierten Detektoren abgebildet. Für weitere Details zu den Bezeichnern und Detektoren sei auf die Publikation [DWSP12] verwiesen.

5.6 Visuelle Detektionsergebnisse

Evalutionskurven sind die statistisch korrekte Art die Qualität eines Detektors zu beurteilen. Dennoch ist es interessant die Detektionen anhand der Testbilder selbst zu begutachten. Da die Ausgabe des Detektors aber eine Textdatei mit Annotationen ist, muss ein spezielles Programm genutzt werden um Detektionen anzuzeigen. Hierfür kann der Code von Dollar [Dol12] auch verwendet werden. Die entsprechende Funktion ist *dbBrowser*. Es sei hier auf die entsprechende Dokumentation [Dol12] verwiesen. Als zusätzliche Hilfe werden in dieser Masterarbeit in Anhang C die Dektionsergebnisse des Detektors rhog6css8comb6 abgebildet. Weiterhin befinden sich für einige weitere Detektoren die jeweiligen Visualisierungen auf der beigefügten DVD.

5.7 Laufzeit

Die Laufzeit des Fußgängerdetektionssystems wird in diesem Abschnitt evaluiert. Es wird im Folgenden mit dem TudBrussels Datensatz gearbeitet. Er enthält 508 Bilder der Auflösung 640×480 Pixel. Diese Bilder bieten eine gute Grundlage um die Performanz des Gesamtsystems zu evaluieren. Weil die Laufzeit des Detektors hauptsächlich von der Größe des Bildes abhängt, werden die Bilder noch auf weitere Größen skaliert (320×240 und 1280×960). Zur Evaluation treten die Detektoren

	320×240			640×480			1280×960		
Detektor–ID	μ	σ	fps	μ	σ	fps	μ	σ	fps
HOG-Dalal	n.a.	n.a.	n.a.	4.1841s	n.a.	0.239	18.519s	n.a.	0.054
rhog8	0.176s	0.007s	5.691	0.939s	0.023s	1.065	4.166s	0.094s	0.240
rhog6	0.233s	0.008s	4.286	1.389s	0.024s	0.720	6.607s	0.096s	0.151
rhog6css8comb6	0.488s	0.009s	2.051	3.612s	0.028s	0.277	19.093s	0.109s	0.052
rhog6css-slow8comb6	1.500s	0.008s	0.667	13.886s	0.027s	0.072	78.986s	0.106s	0.013

Tabelle 5.3: Laufzeit der Detektoren

Detektor–ID	SVM C	RT	Zeit
rhog6	0.02	7	1h 43m
rhog8	0.02	7	1h 29m
rhog6css8comb6	0.02	7	6h 11m

Tabelle 5.4: Laufzeit der Trainingsprozedur

aus Tabelle 5.2 an und die langsame CSS-Variante rhog6css-slow8comb6 ohne die in Abschnitt 4.2 beschriebene Beschleunigung. Es wurde für alle Bilder jeweils der Detektor aufgerufen und die Zeit mit dem UNIX-Programm „time“ gemessen. Das Programm gibt die vollständig genutzte Zeit zurück. Dies beinhaltet also auch das Laden der Daten, die Detektion, die Non-Maxima-Suppression und schließlich die Ausgabe der Detektionen. Dieses Maß lässt sich gut messen und man kann so die verschiedenen Detektoren vergleichen. Weiterhin wurde dann aus allen gemessenen Werten jeweils der Mittelwert μ und die Standardabweichung σ berechnet. Tabelle 5.3 und Abbildung 5.7 zeigen die Ergebnisse, wobei die Werte für HOG-Dollar aus der Evaluation [DWSP12] entnommen wurden. Das Testsystem für alle Detektoren bis auf HOG-Dollar ist ein Ubuntu Linux PC mit einem Intel Quad Core mit 2.66 Ghz und 8 GB RAM. Der Detektor rhog8 liefert die besten Laufzeitergebnisse. Zum Vergleich geben Dollar et al. in der Evaluation [DWSP12] die Laufzeit der Referenz HOG-Dalal für 640×480 mit 0.239 fps und für 1280×960 mit 0.054 fps an. Dabei weisen die Autoren darauf hin, dass ein “aktueller PC” genutzt wurde. Die Detektoren rhog6 und rhog8 sind beide schneller und sogar die Kombination mit CSS ist gleichgestellt obwohl ein zusätzliches Merkmal berechnet wird.

Weiterhin fällt auf, dass die effizientere Berechnung des CSS-Merkmals mit rhog6css8comb6 deutlich besser abschneidet als die langsame Variante. Es wird ein Performanz-Zuwachs um den Faktor 4.137 bei Bildern der Größe 1280×960 erzielt.

Tabelle 5.4 zeigt die Laufzeit der Trainingsprozedur jeweils für die verschiedenen Detektoren nach sieben Retraining-Runden. Es fällt auf, dass die Kombination aus HOG und CSS deutlich mehr Zeit benötigt. Die Gründe dafür sind, dass die Merkmalsvektoren größer sind, im Retraining mehr Daten generiert werden und dass die Berechnung der Merkmale länger dauert.

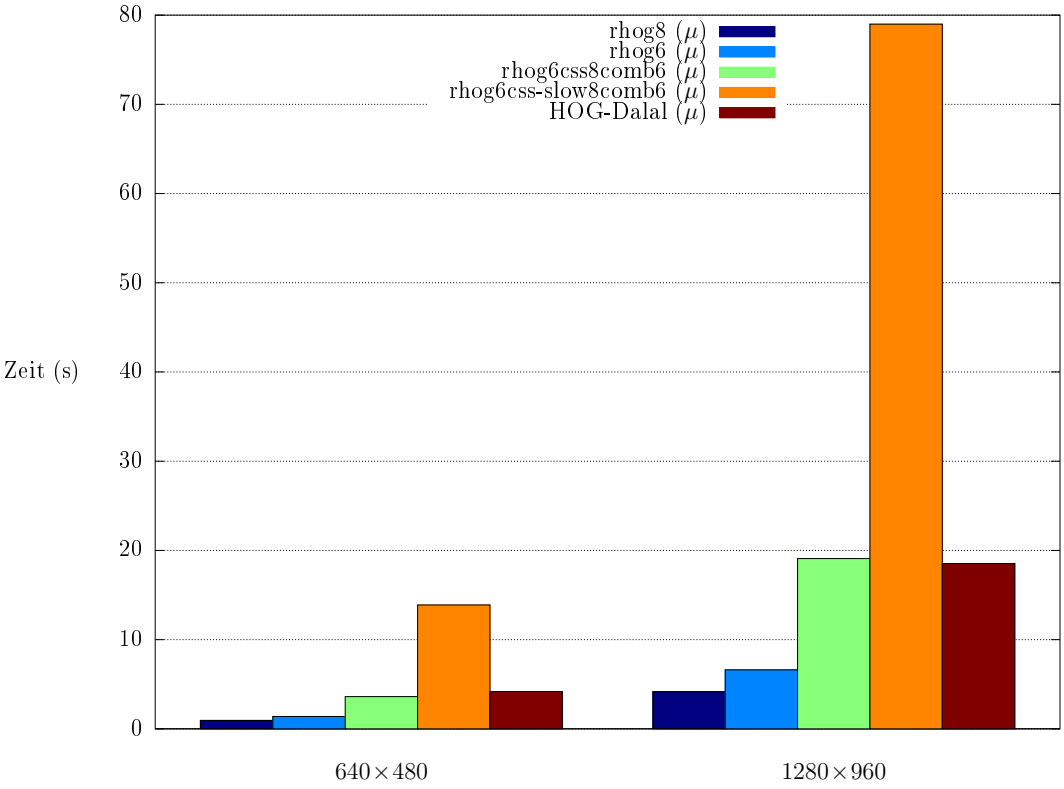


Abbildung 5.7: Laufzeiten im Vergleich

Kapitel 6

Zusammenfassung

Fußgängerdetekion in unstrukturierten Umgebungen ist ein spannendes und herausforderndes Feld der Bildverarbeitung und Mustererkennung. Die zahlreichen Anwendungsmöglichkeiten und die mögliche Übertragbarkeit auf andere Objekt-Detektionsaufgaben sind Grund genug, weitere Forschungsarbeit in diesem Bereich zu leisten. In dieser Masterarbeit wurde die Problemstellung sowohl theoretisch als auch praktisch behandelt. Das Ziel war es, ein etabliertes Verfahren aus der Literatur zu implementieren, zu erweitern und Schwachstellen zu identifizieren.

Es wurde ein vollständiges Detektionssystem implementiert. Die theoretischen Grundlagen des Systems wurden aus der Arbeit von Dalal und Triggs [DT05] übernommen. Die Implementierung und Optimierung wurde eigenständig durchgeführt und dokumentiert. Im Gegensatz zu [DT05] wurde die Kombination aus linearer SVM mit HOG-Merkmal als Gesamtsystem auf Vollbildern verschiedener Test-Datensätze evaluiert. Dalal und Triggs beschränkten sich auf den InriaTest Datensatz und führten die Evaluation anhand normalisierter Fenster durch. In dieser Masterarbeit wurde auf das Evaluationsprotokoll der Autoren Dollar et al. [DWSP12] zurückgegriffen.

Das implementierte System bietet eine modulare Basis für weitere Entwicklungen. So konnte das Merkmal der Color Self-Similarity von den Autoren Walk et al. [WMSS10] integriert werden. Zunächst konnte hier völlig unabhängig gezeigt werden, dass diese Erweiterung verbesserte Detektionsergebnisse bringt. In der Originalpublikation wurde das CSS-Merkmal mit Histogram-Intersection-Kernel SVMs verwendet und in dieser Arbeit mit linearer SVM. Es stellt sich heraus, dass CSS tatsächlich ein weiteres hilfreiches Merkmal ist. Zusätzlich wurde dieses Merkmal detailliert beschrieben und analysiert. Es wurde erkannt, dass das CSS-Merkmal effizienter berechnet werden kann, als es in der Originalpublikation beschrieben wurde. Diese Erweiterung wurde selbständig erkannt und implementiert. Das effizienter berechnete Merkmal produziert dabei exakt die gleichen Merkmalsvektoren

wie die langsamere Variante. Der zusätzliche Speicherverbrauch ist moderat und es konnte ein Performanz-Vorteil mit einem Faktor von ca. 4 gezeigt werden.

Eine weitere Erkenntnis ist, dass ein solches Detektionssystem ein hohes Maß an Finetuning erfordert. Im Verlauf dieser Masterarbeit wurden lange Zeit nicht die erwartete Detektionsqualität erreicht. Dies konnte nur durch systematische Überprüfung und Revision aller beteiligter Module und Details erreicht werden. Gleichzeitig stellt das Ausmaß des Problems in Form von großen Datenmengen und langen Rechenzeiten zusätzlich eine Herausforderung dar, Fehler oder Probleme schnell zu identifizieren.

Das implementierte Detektionssystem wird unter einer Open-Source Lizenz im Internet veröffentlicht [Klo12]. Es kann für weitere Entwicklungen oder praktische Anwendungen verwendet werden.

6.1 Ausblick

Aus Sicht des Autors geht der Trend in der bildbasierten Fußgängererkennung in zwei Richtungen. Zum einen wird versucht bessere Detektionsergebnisse durch die Kombination neuer Merkmale und Lernverfahren zu erreichen. Zum anderen wird versucht bestehende Systeme schneller zu machen um praktische Anwendungen zu ermöglichen. Die nächsten Erweiterungen des hier beschriebenen Detektionssystems könnten vielfältig sein. Auf der Seite der Kombination von Merkmalen könnten Bewegungsinformationen (optischer Fluß) integriert werden. Einige Autoren zeigen deutliche Verbesserungen durch die Nutzung dieser zusätzlichen Informationsquelle (z.B. [WMSS10]). Die Schwierigkeiten des Detektionssystems sowohl bei sehr klein abgebildeten und verdeckten Fußgängern müsste weiter analysiert und verbessert werden. Weiterhin könnte das System auf eine praktische Anwendung hin optimiert werden. Geht man von einem solchen Anwendungsfall aus, können zusätzliche Informationen ausgenutzt werden. So könnten zum Beispiel Lasermessungen eines mobilen Systems dazu verwendet werden den Bodenbereich zu schätzen und somit im Bild den Suchbereich nach Fußgängern deutlich einzuschränken. Weiterhin könnte die Verarbeitung jeder Skale des Skalenraumes parallelisiert ausgeführt werden. Dies könnte das System je nach Anzahl der Prozessoren deutlich beschleunigen. Insgesamt ist davon auszugehen, dass solche Optimierungen letztlich das hier implementierte System echtzeitfähig machen könnten.

Anhang A

Histogrammbildung durch Interpolation

Ein einfaches Beispiel der eindimensionalen Histogrammbildung soll im Folgenden beschreiben, wie sich lineare Interpolation auswirkt. Gegeben sei das Interval $[0, 2[$ der reellen Zahlen. Das Histogramm sei in zwei Klassen $k \in \{0, 1\}$ unterteilt. Der Abstand zwischen den Klassen ist $w = 1$ und die Mittelpunkte der Klassen liegen bei 0.5 und 1.5. Einträge (x_i, γ_i) mit $x_i \in [0, 2[$ und Gewicht $\gamma_i > 0$ sollen nun in das Histogramm eingetragen werden. Ohne Interpolation ist das Histogramm durch

$$H_{clamp}(k) = \sum_i \begin{cases} \gamma, & \text{falls } \lfloor x_i \rfloor = k \\ 0, & \text{sonst} \end{cases} \tag{A.1}$$

definiert. Mit linearer Interpolation ist das Histogramm H_{interp} gegeben durch

$$H_{interp}(k) = \sum_i \begin{cases} \left(1 - \frac{\left|x_i - (k + \frac{w}{2})\right|}{w}\right)\gamma, & \text{falls } \left\lfloor x_i - \frac{w}{2} \right\rfloor = k \vee \left\lceil x_i - \frac{w}{2} \right\rceil = k \\ 0, & \text{sonst} \end{cases} . \tag{A.2}$$

Für den Eintrag $(x_0 = 1, \gamma_0 = 1)$ erhält man die Histogramme $H_{interp}(0) = 0.5$ und $H_{interp}(1) = 0.5$ sowie $H_{clamp}(0) = 0$ und $H_{clamp}(1) = 1$. Verändert man den Eintrag nur leicht auf $(x_0 = 0.999, \gamma_0 = 1)$, so erhält man die Histogramme $H_{interp}(0) = 0.501$ und $H_{interp}(1) = 0.499$ sowie $H_{clamp}(0) = 1$ und $H_{clamp}(1) = 0$. Dieses Beispiel zeigt, dass das linear interpolierte Histogramm *stabil* ist und das nicht interpolierte *instabil*. Die mathematische Eigenschaft der Stabilität beschreibt ob leicht veränderte Anfangswerte nur zu leicht veränderten Resultaten führen. Hinsichtlich der Stabilität ist also ein linear interpoliertes Histogramm zu bevorzugen, wenngleich die Rechenzeit größer ist als im uninterpolierten Fall. Das Prinzip der Interpolation lässt sich auf beliebig hohe Dimensionalitäten des Histogramms übertragen.

Anhang B

Automatisiertes Lernverfahren

Nachfolgend wird die Ausgabe des Shell-Skripts zum automatisierten Training beispielhaft für den rhog6css8comb6 Detektor aufgelistet:

```
$ bash learnpd.sh rhog6css8comb6

**********************
*  learnpd starting
*  rhog6css8comb6-C0.02
*  C=0.02 RT=7
**********************

*  Computing initial training dataset...
*  Running LearnDataGenerator on INRIA Training set...
FeatCombinator - RHOG-2-6-9 + CSSGenFast-8 selected.
Random Seed: 42
Generating svm data...
Generating initial negative training instances...
0% ... 100%
Generating intial positive training instances...
0% ... 100%
Training data generated.
*  Selecting instances from full training data...
unchanged classes: 1  instances: 2416
features: 14608 pick: 12180+2416 instances: 14596/14596 RAM: 3253.45 of 6655.21 MB
Enough RAM for whole dataset... no selection needed.
*  Running SVM training
*  training model inria.model.rhog6css8comb6-C0.02.RT0
..*
optimization finished, #iter = 22
Objective value = -9.429095
nSV = 1885
*  Executing 7 round(s) of retraining...
**********************
*  retraining-round 1...
```

```
**********************
*  Generating false positives as negative training instances...
FeatCombinator - RHOG-2-6-9 + CSSGenFast-8 selected.
Random Seed: 43
genHardNegatives using model: inria.model.rhog6css8comb6-C0.02.RT0
0% ... 100%
6770 hard negatives found.
*  Selecting instances from full training data...
unchanged classes: 1  instances: 2416
features: 14608 pick: 18950+2416 instances: 21366/21366 RAM: 4762.49 of 6655.21 MB
Enough RAM for whole dataset... no selection needed.
*  Running SVM training
*  training model inria.model.rhog6css8comb6-C0.02.RT1
..*
optimization finished, #iter = 28
Objective value = -35.645638
nSV = 5221
**********************
*  retraining-round 2...
**********************
*  Generating false positives as negative training instances...
FeatCombinator - RHOG-2-6-9 + CSSGenFast-8 selected.
Random Seed: 44
genHardNegatives using model: inria.model.rhog6css8comb6-C0.02.RT1
0% ... 100%
1299 hard negatives found.
*  Selecting instances from full training data...
unchanged classes: 1  instances: 2416
features: 14608 pick: 20249+2416 instances: 22665/22665 RAM: 5052.04 of 6655.21 MB
Enough RAM for whole dataset... no selection needed.
*  Running SVM training
*  training model inria.model.rhog6css8comb6-C0.02.RT2
..*
optimization finished, #iter = 29
Objective value = -46.830815
nSV = 6299
**********************
*  retraining-round 3...
**********************
*  Generating false positives as negative training instances...
FeatCombinator - RHOG-2-6-9 + CSSGenFast-8 selected.
Random Seed: 45
genHardNegatives using model: inria.model.rhog6css8comb6-C0.02.RT2
0% ... 100%
89 hard negatives found.
*  Selecting instances from full training data...
unchanged classes: 1  instances: 2416
features: 14608 pick: 20338+2416 instances: 22754/22754 RAM: 5071.88 of 6655.21 MB
Enough RAM for whole dataset... no selection needed.
```

```
*  Running SVM training
*  training model inria.model.rhog6css8comb6-C0.02.RT3
..*.
optimization finished, #iter = 30
Objective value = -48.397859
nSV = 6381
**********************
*  retraining-round 4...
**********************
*  Generating false positives as negative training instances...
FeatCombinator - RHOG-2-6-9 + CSSGenFast-8 selected.
Random Seed: 46
genHardNegatives using model: inria.model.rhog6css8comb6-C0.02.RT3
0% ... 100%
31 hard negatives found.
*  Selecting instances from full training data...
unchanged classes: 1  instances: 2416
features: 14608 pick: 20369+2416 instances: 22785/22785 RAM: 5078.79 of 6655.21 MB
Enough RAM for whole dataset... no selection needed.
*  Running SVM training
*  training model inria.model.rhog6css8comb6-C0.02.RT4
..*
optimization finished, #iter = 29
Objective value = -48.902320
nSV = 6428
**********************
*  retraining-round 5...
**********************
*  Generating false positives as negative training instances...
FeatCombinator - RHOG-2-6-9 + CSSGenFast-8 selected.
Random Seed: 47
genHardNegatives using model: inria.model.rhog6css8comb6-C0.02.RT4
0% ... 100%
6 hard negatives found.
*  Selecting instances from full training data...
unchanged classes: 1  instances: 2416
features: 14608 pick: 20375+2416 instances: 22791/22791 RAM: 5080.12 of 6655.21 MB
Enough RAM for whole dataset... no selection needed.
*  Running SVM training
*  training model inria.model.rhog6css8comb6-C0.02.RT5
..*
optimization finished, #iter = 29
Objective value = -49.019689
nSV = 6437
**********************
*  retraining-round 6...
**********************
*  Generating false positives as negative training instances...
FeatCombinator - RHOG-2-6-9 + CSSGenFast-8 selected.
```

```
Random Seed: 48
genHardNegatives using model: inria.model.rhog6css8comb6-C0.02.RT5
0% ... 100%
2 hard negatives found.
*  Selecting instances from full training data...
unchanged classes: 1  instances: 2416
features: 14608 pick: 20377+2416 instances: 22793/22793 RAM: 5080.57 of 6655.21 MB
Enough RAM for whole dataset... no selection needed.
*  Running SVM training
*  training model inria.model.rhog6css8comb6-C0.02.RT6
..*
optimization finished, #iter = 29
Objective value = -49.066742
nSV = 6449
**********************
*  retraining-round 7...
**********************
*  Generating false positives as negative training instances...
FeatCombinator - RHOG-2-6-9 + CSSGenFast-8 selected.
Random Seed: 49
genHardNegatives using model: inria.model.rhog6css8comb6-C0.02.RT6
0% ... 100%
3 hard negatives found.
*  Selecting instances from full training data...
unchanged classes: 1  instances: 2416
features: 14608 pick: 20380+2416 instances: 22796/22796 RAM: 5081.24 of 6655.21 MB
Enough RAM for whole dataset... no selection needed.
*  Running SVM training
*  training model inria.model.rhog6css8comb6-C0.02.RT7
..*
optimization finished, #iter = 29
Objective value = -49.123663
nSV = 6454

**********************
*  learnpd finished
*  rhog6css8comb6-C0.02
*  C=0.02 RT=7
**********************
```

Anhang C

Visuelle Detektionsergebnisse

Abbildung C.1: Detektionsergebnisse (rhog6css8comb6) 1/8

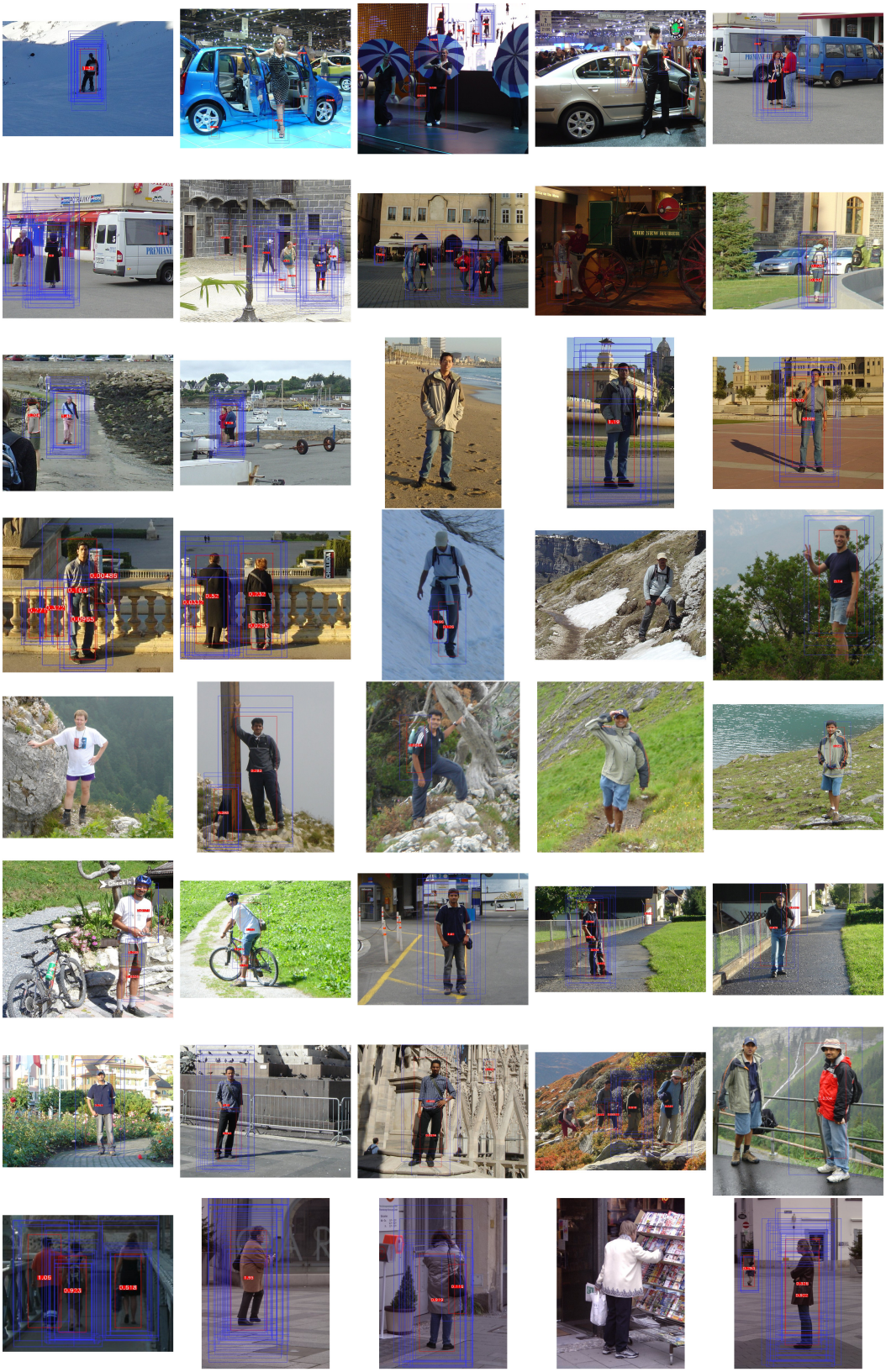


Abbildung C.2: Detektionsergebnisse (rhog6css8comb6) 2/8

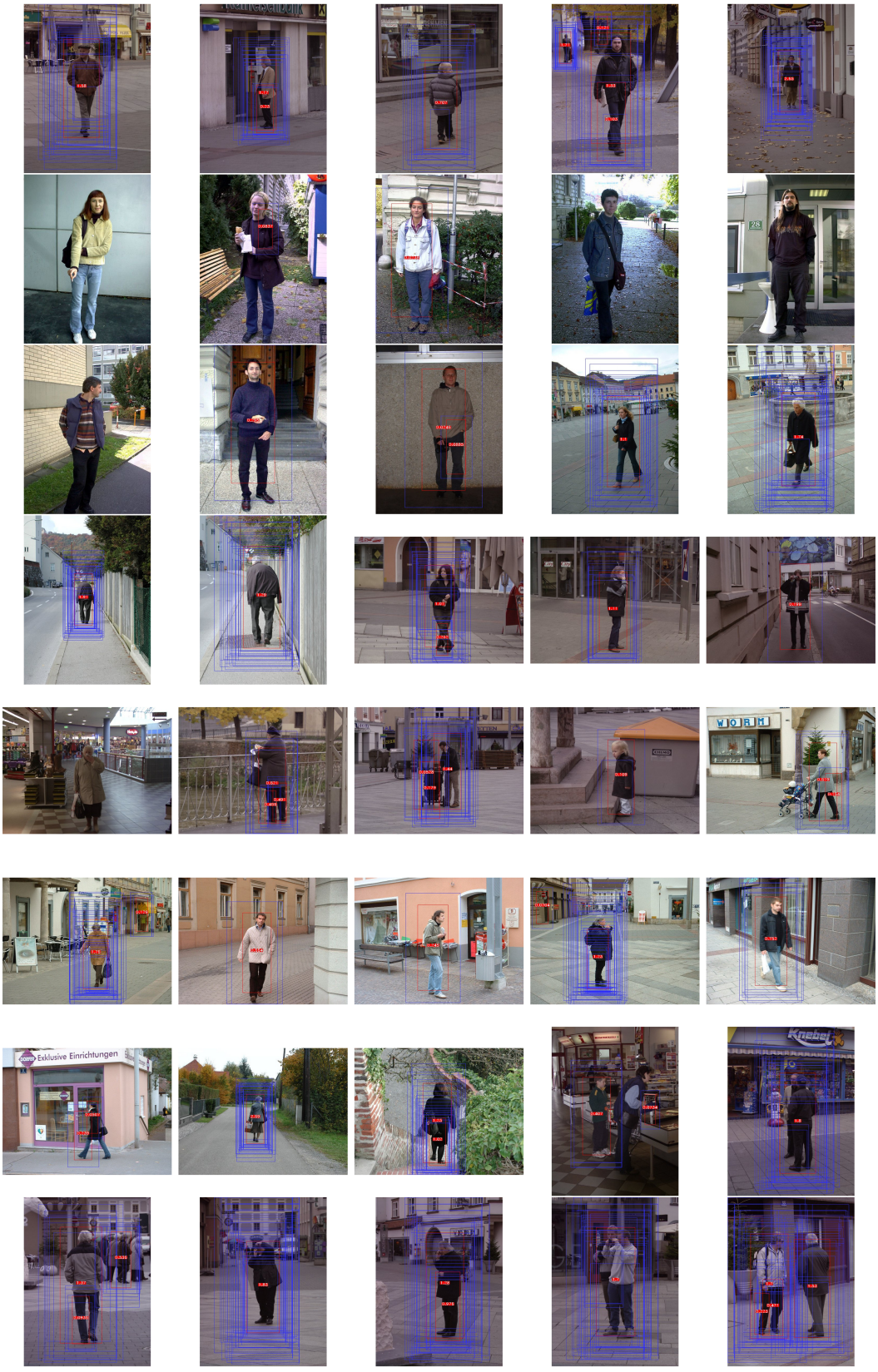


Abbildung C.3: Detektionsergebnisse (rhog6css8comb6) 3/8

Abbildung C.4: Detektionsergebnisse (rhog6css8comb6) 4/8

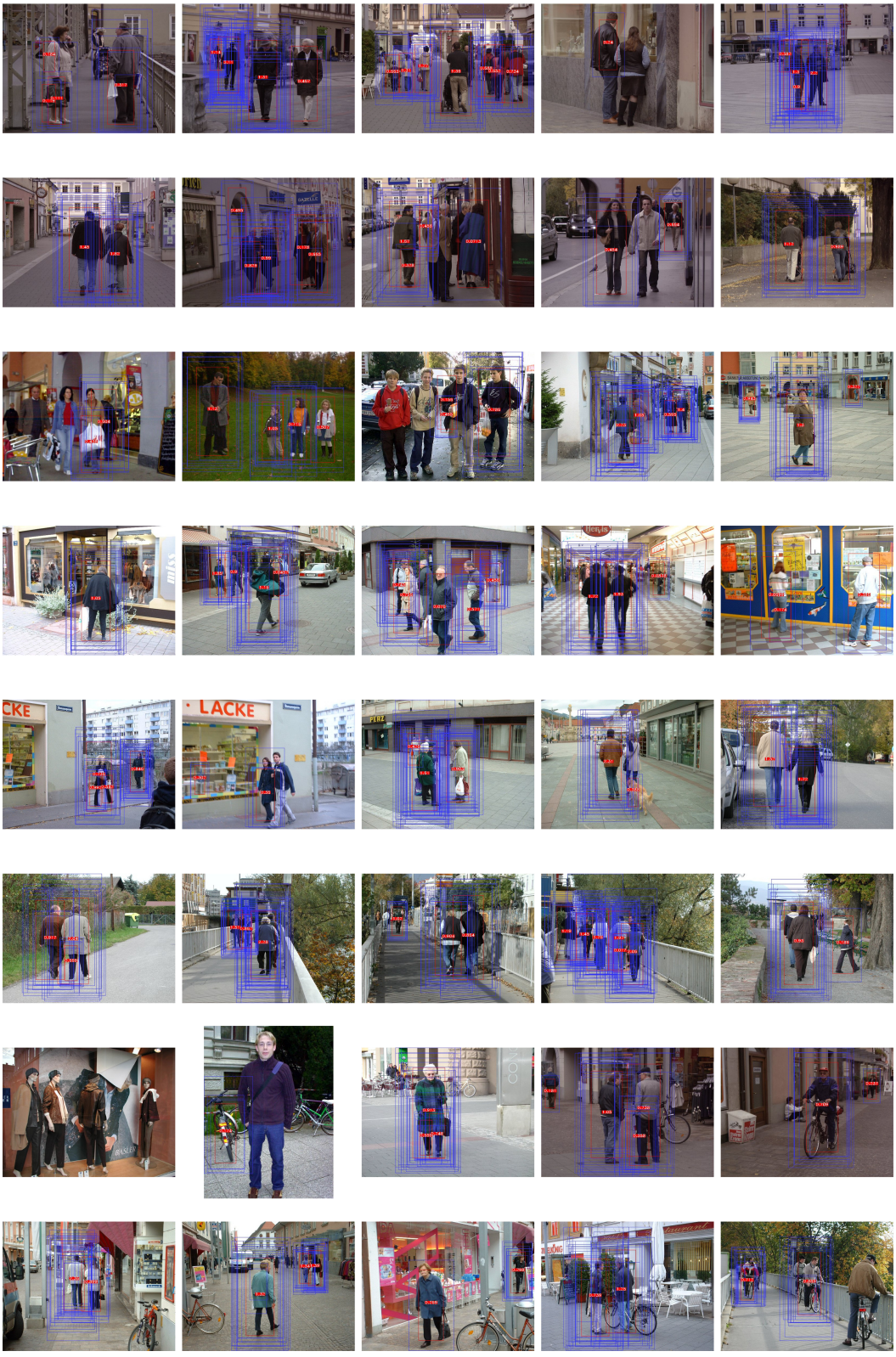


Abbildung C.5: Detektionsergebnisse (rhog6css8comb6) 5/8

Abbildung C.6: Detektionsergebnisse (rhog6css8comb6) 6/8

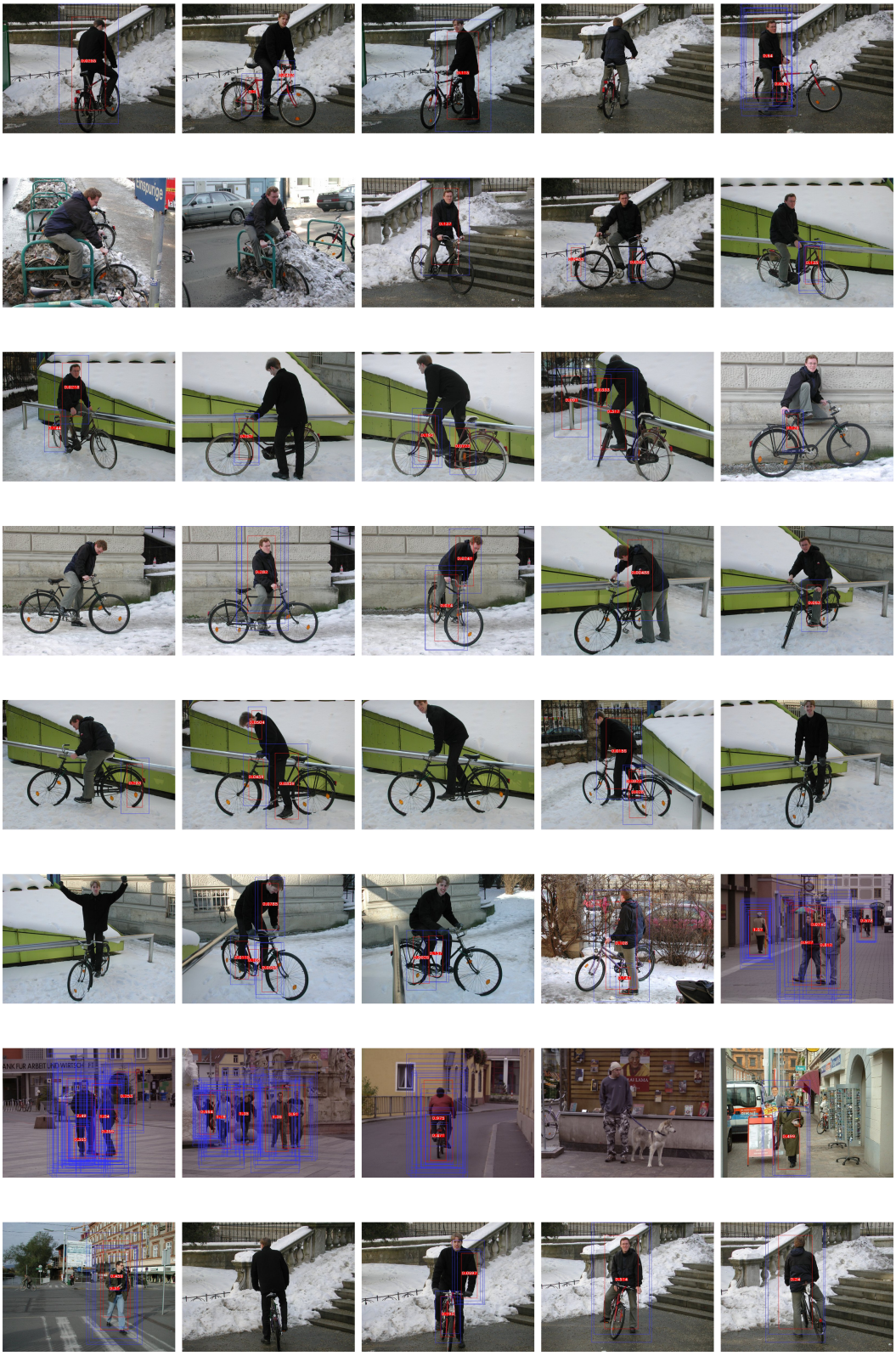


Abbildung C.7: Detektionsergebnisse (rhog6css8comb6) 7/8

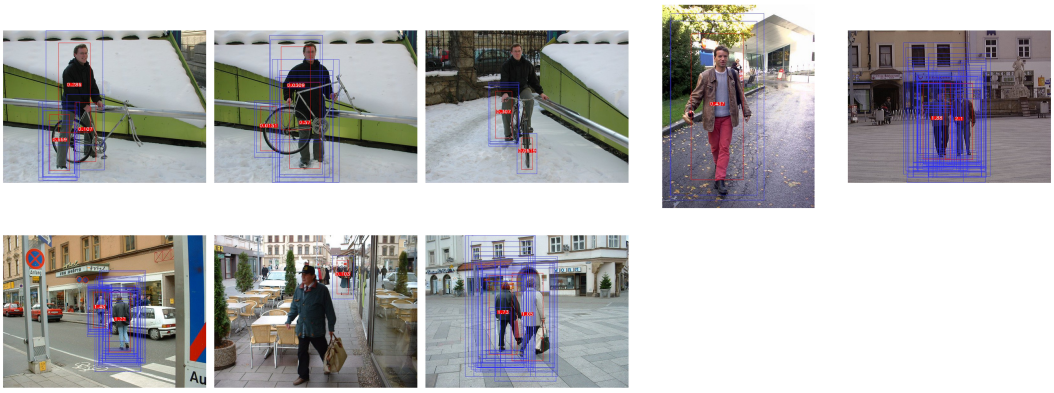


Abbildung C.8: Detektionsergebnisse (rhog6css8comb6) 8/8

Literaturverzeichnis

[CM02] Comaniciu, D. ; Meer, P.: Mean Shift: A Robust Approach Toward Feature Space Analysis. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24 (2002), Nr. 5, S. 603–619

[Cro84] Crow, F. C.: Summed-Area Tables for Texture Mapping. In: *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques.* New York, USA, 1984, S. 207–212

[Dal06] Dalal, N.: *Finding People in Images and Videos*, Institut National Polytechnique de Grenoble, Diss., 2006

[DBP10] Dollar, P. ; Belongie, S. ; Perona, P.: The Fastest Pedestrian Detector in the West. In: *British Machine Vision Conference, BMVC 2010, Aberystwyth, UK, August 31 - September 3, 2010. Proceedings.* Aberystwyth, UK, 2010, S. 68.1–68.11

[Dol12] Dollar, P.: *Piotr's Image & Video Matlab Toolbox.* `http://vision.ucsd.edu/~pdollar/toolbox/doc`, Dezember 2012

[DT05] Dalal, N. ; Triggs, B.: Histograms of Oriented Gradients for Human Detection. In: *International Conference on Computer Vison and Pattern Recognition.* San Diego, USA, 2005, S. 886–893

[DTPB09] Dollar, P. ; Tu, Z. ; Perona, P. ; Belongie, S.: Integral Channel Features. In: *Proceedings of the British Machine Vision Conference.* London, UK, 2009, S. 91.1–91.11

[DTS06] Dalal, N. ; Triggs, B. ; Schmid, C.: Human Detection Using Oriented Histograms of Flow and Appearance. In: *Computer Vision - ECCV 2006, 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006, Proceedings, Part II.* Graz, Austria, 2006, S. 428–441

[DWSP12] Dollar, P. ; Wojek, C. ; Schiele, B. ; Perona, P.: Pedestrian Detection: An Evaluation of the State of the Art. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34 (2012), Nr. 4, S. 743–761

[EVGW+10] Everingham, M. ; Van Gool, L. J. ; Williams, C. K. I. ; Winn, J. M. ; Zisserman, A.: The Pascal Visual Object Classes (VOC) Challenge. In: *International Journal of Computer Vision* 88 (2010), Nr. 2, S. 303–338

[FCH+08] Fan, R.-E. ; Chang, K.-W. ; Hsieh, C.-J. ; Wang, X.-R. ; Lin, C.-J.: LIBLINEAR: A Library for Large Linear Classification. In: *Journal of Machine Learning Research* 9 (2008), S. 1871–1874

[FGMR10] Felzenszwalb, P. F. ; Girshick, R. B. ; McAllester, D. A. ; Ramanan, D.: Object Detection with Discriminatively Trained Part Based Models. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32 (2010), Nr. 9, S. 1627–1645

[FS95] Freund, Y. ; Schapire, R.: A Decision-Theoretic Generalization of Online Learning and an Application to Boosting. In: *Computational Learning Theory, Second European Conference, EuroCOLT '95, Barcelona, Spain, March 13-15, 1995, Proceedings*. Barcelona, Spain, 1995, S. 23–37

[GHJV95] In: Gamma, E. ; Helm, R. ; Johnson, R. ; Vlissides, J.: *Design Patterns – Elements of Reusable Object-Oriented Software*. 1. Amsterdam : Addison-Wesley Longman, 1995, S. 163–174. – 37. Reprint (2009)

[HS81] Horn, B. K. P. ; Schunck, B. G.: Determining Optical Flow. In: *Artificial Intelligence* 17 (1981), S. 185–203

[Klo12] Klostermann, M.: *Pedestrian Detection in Unstructured Environments*. `http://userpages.uni-koblenz.de/~michaelk/pdiue`, Dezember 2012

[MBM08] Maji, S. ; Berg, A. C. ; Malik, J.: Classification Using Intersection Kernel Support Vector Machines is Efficient. In: *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*. Anchorage, USA, 2008, S. 2245–2260

[Por05] Porikli, F.: Integral Histogram: A Fast Way To Extract Histograms in Cartesian Spaces. In: *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*. Washington, USA, 2005, S. 829–836

[PP00] Papageorgiou, C. ; Poggio, T.: A Trainable System for Object Detection. In: *International Journal of Computer Vision* 38 (2000), Nr. 1, S. 15–33

[PR09] Prisacariu, V. ; Reid, I.: fastHOG - a real-time GPU implementation of HOG / Department of Engineering Science, Oxford University. 2009 (2310/09). – Forschungsbericht

[TK09] In: Theodoridis, S. ; Koutroumbas, K.: *Pattern Recognition*. Academic Press, 2009, S. 119–127

[VC95] Vapnick, V. ; Cortes, C.: Support-Vector Networks. In: *Machine Learning* 20 (1995), Nr. 3, S. 273–297

[VJ01] Viola, P. A. ; Jones, M. J.: Rapid Object Detection using a Boosted Cascade of Simple Features. In: *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001), with CD-ROM, 8-14 December 2001, Kauai, HI, USA*. Kauai, USA, 2001, S. 511–518

[VJS03] Viola, P. A. ; Jones, M. J. ; Snow, D.: Detecting Pedestrians Using Patterns of Motion and Appearance. In: *9th IEEE International Conference on Computer Vision (ICCV 2003), 14-17 October 2003, Nice, France*. Nice, France, 2003, S. 734–741

[WMSS10] Walk, S. ; Majer, N. ; Schindler, K. ; Schiele, B.: New Features and Insights for Pedestrian Detection. In: *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*. San Francisco, USA, 2010, S. 1030–1037

[WWS09] Wojek, C. ; Walk, S. ; Schiele, B.: Multi-Cue Onboard Pedestrian Detection. In: *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. Miami, USA, 2009, S. 794–801

[ZYCA06] Zhu, Q. ; Yeh, M.-C. ; Cheng, K.-T. ; Avidan, S.: Fast Human Detection Using a Cascade of Histograms of Oriented Gradients. In: *2006 IEEE Computer Society Conference on Computer Vision and*

Pattern Recognition (CVPR 2006), 17-22 June 2006,. New York, USA, 2006, S. 1491–1498