Vom Institut für Neuro- und Bioinformatik
an der Universität zu Lübeck

Direktor:
Prof. Dr. rer. nat. Thomas Martinetz

Gesichtsdetektion mit Time-of-Flight-Kameras

Diplomarbeit
im Rahmen des Informatik-Hauptstudiums

vorgelegt von
Kolja Riemer

ausgegeben von
PD Dr.-Ing. Erhardt Barth

betreut von
Dipl.-Inf. Martin Böhme

Datum: 17. Oktober 2008

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die dazu beigetragen haben, dass diese Arbeit entstanden ist.
Ganz besonders möchte ich mich bei meinem Betreuer Martin Böhme für seine unermüdliche Geduld bei all meinen Fragen bedanken. Ohne seine Hilfe wäre diese Arbeit nicht möglich gewesen. Ann-Kristin Grimm und Michael Dorr danke ich für die langen Debugging-Sitzungen, bei der sie mich tatkräftig unterstützt haben. Ebenso danke ich Herrn PD Dr.-Ing. Erhardt Barth für die Begutachtung dieser Arbeit.
Ein besonderer Dank geht an meine Eltern, ohne deren moralische sowie finanzielle Unterstützung ich dieses Studium sicherlich nicht hätte beenden können. Auch möchte ich allen meinen Freunden aus Lübeck für die gute Zeit danken.

Erklärung zum eigenständigen Arbeiten

Ich versichere an Eides statt durch meine Unterschrift, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

. . . . . .

Kolja Riemer Lübeck, den 17. Oktober 2008

Zusammenfassung

In dieser Arbeit wird Gesichtsdetektion auf einer Time-Of-Flight-Kamera in Echtzeit umgesetzt. Gesichtsdetektion ist die Aufgabe, ein Gesicht in einer Aufnahme zu finden. Der existierende Ansatz zur Objekterkennung von Paul Viola und Michael Jones, wird in dieser Arbeit auf eine Time-Of-Flight-Kamera umgesetzt. Eine Time-Of-Flight-Kamera ist ein kombinierter Helligkeits- und Tiefensensor, bei der modulierte Infrarotstrahlen ausgesendet werden und deren Reflektion auf ein Objekt von einem Sensor der Kamera gemessen wird. Dieses reflektierte Signal ermöglicht sowohl ein Intensitätsbild über die Amplitude des Signals, als auch ein Tiefenbild, welches über die Laufzeit des ausgesendeten Signals zum reflektierenden Objekt und zurück, bestimmt wird.
Die Implementierung des Klassifikators wurde mittels eines bekannten 2D Trainingsdatensatz vom MIT Massachusetts validiert. Anschließend wurde die Implementation auf Daten der Time-Of-Flight-Kamera angewendet, die am Institut für Neuro- und Bioinformatik aufgenommen wurden. So wurde versucht einen Vergleich zwischen einem Klassifikator der auf Grauwertbildern und einem Klassifikator der auf Time-Of-Flight Bildern arbeitet zu geben.
Dabei erwies sich der verwendete Trainingsdatensatz als problematisch, da es nicht gelang einen Detektor auf Time-Of-Flight Bildern, mit der von uns erhofften Klassifikationsleistung, fertig zu trainieren. Unter Verwendung der Tiefeninformationen gelang es aber, einen Klassifikator mit typischerweise nur ein bis max. zwei Fehldetektionen bei nahezu 100% Detektionsrate, auf Time-Of-Flight Bildern der Größe 176×144 Pixel, zu trainieren. Dies ist erstaunlich, da jegliche Versuche scheiterten, einen Klassifikator ausschließlich auf den Intensitätsbildern des Time-Of-Flight Datensatzes zu trainieren. Dabei lieferte sogar der auf dem MIT Datensatz trainierte Klassifikator gute

Ergebnisse auf den Intensitätsbildern der Time-Of-Flight-Kamera, obwohl dieser Detektor auf Grauwertbildern trainiert wurde und nicht auf den mittels Infrarotlicht erzeugten Bildern der Time-Of-Flight-Kamera. Diese Ergebnisse zeigen, dass die Tiefeninformationen der Time-Of-Flight-Kamera hilfreiche Informationen für die Klassifikation beisteuern.

Das Problem könnte behoben werden, indem Verbesserungen an dem Trainingsdatensatz ausgeführt werden. Da das Training eines Klassifikators auf einem aktuellen PC (ca. 3 GHz Taktfrequenz) sehr zeitaufwendig ist, ist es nur mit großem Zeitaufwand möglich die Auswirkung einer Änderung an den Trainingsdaten nachzuvollziehen. Um dieses Problem künftig besser handhaben zu können wird beschrieben wie sich der Trainingsprozeß für eine Multiprozessormaschiene optimieren lässt.

Abstract

This thesis deals with real time face detection with the help of a Time-Of-Flight camera. The purpose of face detection algorithms is to locate faces within pictures or video frames. We used a method of object detection by Paul Viola and Michael Jones and adapted it for the Time-Of-Flight camera. Such a camera is a combined intensity and range sensor. By sending out modulated IR rays, the Time-Of-Flight camera is able to measure their reflections upon objects with its sensor. This enables us to measure intensity using the amplitude of the signal as well as range, which is determined by measuring the time the light takes to travel to the object and back to the camera.

We validated the classifier with the well-known MIT face dataset. Additionally we used a face dataset which we generated with the Time-Of-Flight camera. When the detector was trained only on the intensity channel of the Time-Of-Flight dataset, training did not complete because the false-positive rate remained far below the goal rate. When the detector was trained on both the intensity and range channels, training eventually had to be aborted as well, but the goal for the false-positive rate had almost been reached and the resulting classifier was usable. This classifier achieves a detection rate of nearly 100% while having about one to two false-positive detections per frame.
At the end of this thesis, we discuss how the face dataset could be improved to reduce the false-positive rate further.

Inhaltsverzeichnis

1 Einleitung

Gesichtsdetektion ist ein Teilbereich der Bildverarbeitung, bei dem es darum geht, in einem gegebenen Bild alle vorhandenen Gesichter zu finden. Auf gewöhnlichen Grauwertbildern existieren einige Verfahren, die diese Aufgabe lösen. In dieser Arbeit soll die Frage geklärt werden, ob man ein bekanntes Verfahren auf einen kombinierten Helligkeits- und Tiefensensor übertragen kann und ob dadurch die Detektionsrate verbessert wird. Als Sensor wird eine Time-Of-Flight-Kamera eingesetzt. Bei einer solchen Kamera werden modulierte Infrarotstrahlen ausgesendet, deren Reflektion auf ein Objekt von einem Sensor der Kamera gemessen wird. Dieses reflektierte Signal ermöglicht sowohl ein Intensitätsbild über die Amplitude des Signals, als auch ein Tiefenbild, welches über die Laufzeit des empfangenen Signals gegenüber dem ausgesendeten Signal bestimmt wird. (vgl. Kapitel 2 ab Seite 5). Die Time-Of-Flight-Kamera, die in dieser Arbeit zum Einsatz kommt, ist die SR 3000 von der Firma MESA [Mes08]. Ein weiterer Vorteil bei dieser Kamera ist, dass sie mit nur einem Sensor sowohl Helligkeits- als auch Tiefeninformationen aufzeichnet.
Die Aufgabe der Gesichtsdetektion ist abzugrenzen von Verfahren, die der Gesichtserkennung dienen. Bei solchen Verfahren ist die Gesichtsdetektion ein Vorverarbeitungsschritt. Nach diesem Schritt sollen die gefundenen Gesichter dann Personen einer Datenbank zugeordnet werden.
Häufig wird die Gesichtsdetektion als Vorverarbeitungsschritt für andere Anwendungen verwendet. Das heißt es werden zunächst Gesichter lokalisiert und anschließend weiter verarbeitet. Eine Reihe solcher Anwendungen sei hier aufgeführt:

- Biometrische Datenbanken: Abgleich mit biometrischen Datenbanken, wie sie z.B. durch den künftigen biometrischen Pass zur Verfügung

stehen.

- Intelligente Überwachungstechnologie: Für den Objektschutz könnte die Gesichtsdetektion helfen, verdächtige Personen zu registrieren, um bei Bedarf den betreffenden Bildausschnitt zu vergrößern.

- Mensch-Maschine-Interaktion: In Museen könnten Terminals mittels Gesichtsdetektion die Anwesenheit eines Besuchers erkennen und daraufhin eine spezifische Aktion ausführen (z.B. den Start eines Films).

- Video-Chat: Nachdem das Gesicht im Kamerabild lokalisiert worden ist, können unterhaltende Elemente wie z.B. ein zum Gesicht passender Hut über das Videobild gelegt werden (Augmented Reality).

- Verwaltung von Bildern: Mit der digitalen Photographie ist die Menge an Bildern, die eine privater Anwender aufnimmt, deutlich gestiegen. Mittels Gesichtsdetektion lassen sich beispielsweise intelligente Bilddatenbanken realisieren, die z.B. Portraitfotos von Landschaftsfotos unterscheiden können.

- Präsentation von Fotos: Bildbetrachter bieten häufig die Möglichkeit, Bildausschnitte bewegt anzuzeigen. Mittels Gesichtsdetektion ließen sich diese Animationen soweit kontrollieren, dass Gesichter möglichst im sichtbaren Bereich dargestellt werden.

- Fokussierung in Digitalkameras: Mittels Gesichtsdetektion wird der Fokuspunkt automatisch auf erkannte Personen ausgerichtet [Fuj08, Nik07].

- Automatische Alterskontrolle: An Zigarettenautomaten können Gesichter detektiert werden, um anschließend das Alter der aufgenommen Person grob zu bestimmen. Hierfür wird in dem detektierten Gesicht nach Merkmalen wie z.B. Runzeln, Krähenfüße, Hautstruktur oder anderen Merkmalen älterer Menschen gesucht [hei08].

- Alkoholmissbrauch: Es gibt Systeme für Autos, die einen Motorstart nur zulassen, wenn zuvor der Atemalkoholgehalt überprüft wurde [Lüb07].

Um sicherzustellen, dass wirklich der registrierte Fahrer überprüft wird, könnte hier das Gesicht der Person gefunden und dann überprüft werden.

- Blickbewegungsregistrierung: Blickbewegungsregistrierung (Okulographie, Eye Tracking) ist das Aufzeichnen der hauptsächlich aus Fixationen (Punkte, die man genau betrachtet) und Sakkaden (schnellen Augenbewegungen) bestehenden Blickbewegungen einer Person. Als Vorverarbeitungsschritt wird das Gesicht der Person gesucht, deren Augenbewegungen gemessen werden sollen.

Wir sehen also, dass es für das Gebiet der Gesichtsdetektion vielfältige Anwendungsmöglichkeiten gibt. Zur Lösung des Problems gibt es mittlerweile eine Reihe von Algorithmen. Zu den bekannteren zählen folgende:

- Schneiderman und Kanade benutzen mehrere unterschiedliche Detektoren, die jeweils spezialisiert auf bestimmte Ausrichtungen der zu detektierenden Objekte sind. Die Ergebnisse der verschiedenen Detektoren werden dann in einem zweiten Schritt mittels Methoden aus der Statistik zu einer Gesamtaussage kombiniert [SLM+00].

- Rowley, Baluja und Kanade verwenden künstliche neuronale Netzwerke, die auf Gesichter trainiert sind. Für das Training wurden dynamisch mittels eines Bootstrap-Algorithmus Negativbeispiele generiert [RBK96].

- Viola und Jones verwenden eine Kombination von schwachen Klassifikatoren, die nur wenig besser als der Zufall sind. Diese schwachen Klassifikatoren werden mittels AdaBoost zu einem starken Klassifikator kombiniert [VJ04].

Der Algorithmus von Viola und Jones hat breite Resonanz gefunden und ist von vielen Gruppen aufgegriffen und weiterentwickelt worden. Der Grund hierfür dürfte die Tatsache sein, dass dieser Algorithmus eine gute Kombination aus Effizienz und Klassifikationsleistung bietet, so dass er für Echtzeitanwendungen zu gebrauchen ist. Auch diese Arbeit nutzt den Algorithmus von Viola und Jones als Grundlage. Während der Algorithmus in seiner

ursprünglichen Form Grauwertbilder herkömmlicher Kameras verwendet, werden in dieser Arbeit zusätzlich die Tiefeninformationen einer Time-Of-Flight-Kamera genutzt.

Die Motivation für diese Erweiterung liegt in der Hoffnung, einerseits robuster und andererseits auch schneller in der Detektion zu werden, da Gesichter eine dreidimensionale Struktur haben und diese Struktur mit einer Time-Of-Flight-Kamera nutzbar wird. Als konkretes Beispiel ist hier eine Anwendung aus Japan zu nennen [Spi08]. Dort werden Zigarettenautomaten eingesetzt, die eine Kamera eingebaut haben. Der Kunde wird von dieser Kamera gefilmt. Mittels Gesichtsdetektion wird nun das Gesicht analysiert und bezüglich des Alters klassifiziert. Leider hat sich in der Praxis gezeigt, dass sich mittels eines Fotos einer älteren Person die Sicherheitsmaßnahmen des Automaten umgehen lassen. Würden nun Tiefeninformation mit einbezogen, könnte man solche Betrugsversuche unterbinden.

2 Time-of-Flight-Kamera

Eine Time-Of-Flight-Kamera liefert zusätzlich zu einem gewöhnlichen Helligkeitsbild wie bei herkömmlichen Kameras ein Tiefenbild. Dieses Tiefenbild liefert zu jedem Pixel des Helligkeitsbildes den Abstand zum aufgenommen Objekt. Dadurch enthält man Informationen über die dreidimensionale Struktur der Szene. Sowohl das Tiefenbild als auch das Helligkeitsbild werden gleichzeitig über denselben Sensor aufgenommen. Dies hat den Vorteil, dass kein nachträgliches Registrieren des Tiefen- auf das Helligkeitsbild notwendig ist, wie bei Kameras die zur Erfassung von Tiefeninformationen einen zweiten Sensor einsetzen (z.B. die Zcam vom 3DV Systems[3DV08]).

2.1 Das Time-Of-Flight-Prinzip zur Distanzbestimmung

Die in dieser Arbeit verwendete Kamera nutzt das Time-Of-Flight-Prinzip als Verfahren zur Bestimmung des Abstands eines Objektes zur Kamera. Eine solche Kamera misst die Reflektion eines Signals, das von Infrarot-LEDs ausgesendet wird. Diese Infrarot-LEDs sind um das Objektiv der Kamera herum angeordnet. Das Signal wird von den Objekten, auf die das modulierte Signal trifft, reflektiert und vom Sensor der Time-Of-Flight-Kamera aufgenommen. Eine Time-Of-Flight-Kamera besteht also im wesentlichen aus zwei Komponenten: Zum einen aus dem durch die Infrarot-LEDs Infrarotlicht ausstrahlenden Teil der Kamera sowie einem speziellen Sensor, der dieses Infrarotlicht aufnimmt. Der verwendete Bildsensor ermöglicht für jeden Pixel (bei der verwendeten Kamera 176×144 Pixel) die gleichzeitige Messung der Phasenverschiebung φ, des Offsets B sowie der Amplitude A des empfangenen Signals (vgl. Abbildung 2.1 auf Seite 6). Die Phasen-

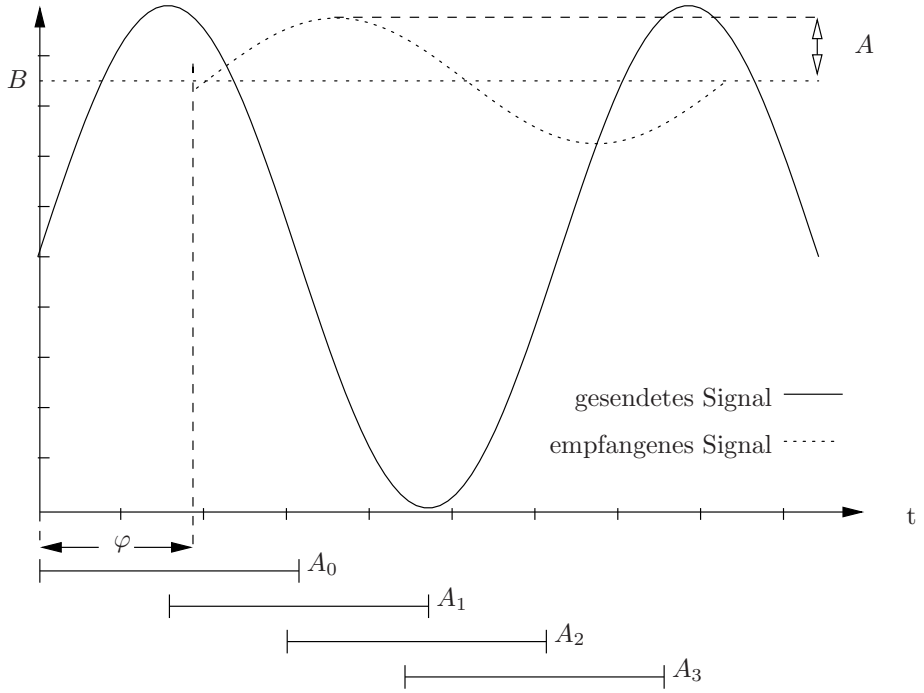


Abbildung 2.1: Das Prinzip einer Time-Of-Flight-Kamera: Es wird ein moduliertes Infrarot-Signal gesendet. Dieses Signal wird an einer Oberfläche reflektiert und vom Sensor der Kamera aufgenommen. Dabei wird über vier versetzte Intervalle von jeweils einer halben Periodenlänge integriert. Aus diesen Werten $A_0, \ldots, A_3$ werden φ als Maß für die Entfernung und die Amplitude A, also die Intensität des reflektierten Signals bestimmt.

verschiebung φ stellt die Zeit dar, die die Reflektion des ausgesendeten Signals bezüglich des Ursprungsignals verschoben ist.Mittels φ und der Tatsache, dass die Lichtgeschwindigkeit mit $c \approx 3 \cdot 10^8 \frac{\mathrm{m}}{\mathrm{s}}$ bekannt ist, kann so direkt die Entfernung des Objektes bestimmt werden, das für die Reflektion des Signals verantwortlich ist. Über die Phasenverschiebung φ wird also das Tiefenbild der Kamera bestimmt. Der Offset B lässt sich durch die Hintergrundbeleuchtung, also durch alle anderen Lichtquellen neben den IR-LED's der Kamera, erklären. Die Amplitude A des aufgenommen Signals hängt ab von der Reflektivität des Objektes. Denn die Amplitude A wird stärker, je stärker das aufgenommene Objekt reflektiert. Über die Amplitude A wird also das Helligkeitsbild der Time-Of-Flight-Kamera konstruiert.
Wie werden die Werte φ, B sowie A aus dem empfangenen reflektierten Signal rekonstruiert? Die Rekonstruktion geschieht mittels Integration von vier Zeitintervallen $(A_0, \ldots, A_3)$ des Eingangssignals über jeweils eine halbe Periodenlänge bzgl. des Ausgangssignals.

Das ausgesendete sinusförmige Signal hat folgende Struktur:

$$e(t) = e \cdot \left[1 + \sin\left(\frac{F}{2\pi} \cdot t\right)\right], \tag{2.1}$$

wobei e den Mittelwert der ausgesandten Lichtenergie darstellt und F für die Modulationsfrequenz steht, die bei der eingesetzten Kamera 20 MHz beträgt. Das empfangene Signal trifft dann in der Form

$$s(t) = BG(t) + e \cdot k \cdot \left[1 + \sin\left(\frac{F}{2\pi} \cdot t - \varphi\right)\right] \tag{2.2}$$

auf den Sensor. Hierbei ist $BG(t)$ die Intensität der Hintergrundbeleuchtung, k der Dämpfungsfaktor sowie φ die Phasenverschiebung. Da abhängig von der Reflektivität des gefilmten Objektes nur ein Teil der ausgesandten Lichtenergie e auf den Sensor der Kamera trifft, wird diese Dämpfung durch den Dämpfungsfaktor k beschrieben. Die Hintergrundbeleuchtung $BG(t)$ kann als Konstante BG angenommen werden, da sie mit einer sehr viel kleineren Frequenz moduliert ist (z.B. 50 Hz durch konventionelle Neonröhren). Das eingehende Signal $s(t)$ kann nun vollständig rekonstruiert werden, indem wir das Eingangssignal über vier Intervalle integrieren $(A_0, \ldots, A_3)$,

wobei jedes Integrationsinterval A_i in einem Abstand einer halben Periode der Modulationsfrequenz des gesendeten Ausgangssignals $e(t)$ aufgezeichnet wird, dabei ist ein einzelnes Intervall A_i definiert durch

$$A_i = \int_{k\frac{\pi}{2\omega}}^{(k+1)\frac{\pi}{2\omega}} s(t)\,dt.$$

Mittels folgender Formeln lassen sich φ, B sowie A aus den Integrationsintervallen $(A_0, \ldots, A_3)$ berechnen (siehe hierzu: [OKL+04] sowie [OLK+04]):

$$\varphi = \operatorname{atan}\left(\frac{A_3 - A_1}{A_0 - A_2}\right), \tag{2.3}$$

$$B = \frac{A_0 + A_1 + A_2 + A_3}{4}, \tag{2.4}$$

$$A = \frac{\sqrt{[A_3 - A_1]^2 + [A_0 - A_2]^2}}{2}. \tag{2.5}$$

Die so ermittelten Werte ermöglichen die vollständige Rekonstruktion des Eingangssignals $s(t)$ und die Berechnung des Helligkeits- und Tiefenbildes. Das Helligkeitsbild wird aus der Amplitude A bestimmt, während das Tiefenbild über die Gleichung

$$L = \frac{L_0}{2\pi} \cdot \varphi \tag{2.6}$$

zu bestimmen ist. L_0 steht für den Eindeutigkeitsbereich der Time-Of-Flight-Kamera. Alle Kameras, die nach dem Time-Of-Flight-Prinzip arbeiten, haben das Problem der Doppeldeutigkeit der Tiefeninformation. Das Ausgangssignal $e(t)$ ist 2π-periodisch, daher kann auch nur eine Phasenverschiebung φ im Intervall $[0, \ldots, 2\pi]$ gemessen werden. Sei L_0 wie bei der von uns eingesetzten Kamera 7,5 m so hat ein Objekt mit einer Entfernung von 0,5 m die gleiche Phasenverschiebung wie ein Objekt, das 8 m entfernt ist. Der Bereich der Eindeutigkeit einer Time-Of-Flight-Kamera hängt von der Modulationsfrequenz des Ausgangssignals ab und lässt sich folgendermaßen berechnen:

$$L_0 = \frac{c\pi}{\omega}. \tag{2.7}$$

Die in dieser Arbeit verwendete Kamera hat eine Modulationsfrequenz von f = 20 MHz, entsprechend ist dann die Kreisfrequenz:

$$\omega = 2\pi f = 2\pi \cdot 2 \cdot 10^7 \mathrm{s}^{-1}. \tag{2.8}$$

Somit bestimmt sich der Eindeutigkeitsbereich der verwendeten Kamera zu:

$$L_0 = \frac{3 \cdot 10^8 \mathrm{ms}^{-1}}{4 \cdot 10^7 \mathrm{s}^{-1}} = \frac{3}{4} \cdot 10\mathrm{m} = 7.5\,\mathrm{m}. \tag{2.9}$$

3 Methoden zur Gesichtserkennung

Der in dieser Arbeit implementierte Gesichtsdetektor basiert auf dem von Paul Viola und Michael Jones [VJ04] entwickelten Verfahren zur Erkennung von Objekten. Dieses Verfahren baut auf drei Kernideen auf:

1. Die Klassifikation basiert auf Features, die Informationen aus einem rechteckigen Bereich des Bildes extrahieren. Durch eine geschickte Vorverarbeitung der Eingabebilder, die so genannten Integralbilder, können diese Features effizient berechnet werden.

2. Features bilden zusammen mit einem Schwellenwert und einem Vorzeichen einen schwachen Klassifikator. Die Auswahl und Kombination von guten schwachen Klassifikatoren zu starken Klassifikatoren mittels des Klassifikators AdaBoost stellt den zweiten Schwerpunkt des Verfahrens von Paul Viola und Michael Jones da.

3. Beim Viola-Jones-Verfahren liegt ein großes Augenmerk auf Echtzeitfähigkeit des Detektors. Daher beschäftigt sich der dritte Kernaspekt des Verfahrens mit der Organisation der schwachen Klassifikatoren in einer Kaskadenstruktur die durch ihren Aufbau dazu beiträgt, dass nicht alle trainierten schwachen Klassifikatoren für die Klassifikation jedes Eingabebildes benötigt werden.

Features bilden die Basis des Klassifikators. Ein solches Feature besteht aus einer einfachen Struktur. Beispielsweise werden die Summen von Pixeln innerhalb von zwei horizontalen Balken voneinander subtrahiert. Der so berechnete Wert wird dann als Repräsentant für die gesamte Umgebung, auf der das Feature berechnet wird, verwendet. Mit Hilfe dieser Features bildet

man also einen Teilbereich eines Bildes auf einen möglichst aussagekräftigen Wert ab. Da Features im Verfahren von Viola und Jones auf unterster Ebene verwendet werden, gehört die Berechnung der Features zu den zeitkritischsten Bereichen des Verfahrens. Durch das Konzept der Integralbilder lassen sich die Features unabhängig von ihrer Größe aber in konstanter Zeit berechnen.
Unter der Annahme, man findet ein spezielles Feature, das möglichst viel mit der Struktur eines Gesichts zu tun hat, hätten wir nun eine einfache Möglichkeit zur Klassifikation. Benötigt wird nur noch ein passender Schwellenwert um eine einfache Klassifikation, basierend auf einem simplen Wertevergleich, zu realisieren. Man nimmt hierfür das besagte Feature und berechnet es auf den zu klassifizierenden Bildern. Diese Werte werden jetzt mittels des zum Feature gehörenden Schwellenwerts verglichen und entsprechend in die Klassen Gesicht und Nichtgesicht aufgeteilt.

Solch ein Feature mit Schwellenwert nennen wir im folgenden auch einen schwachen Klassifikator. Ein solcher einzelner schwacher Klassifikator erreicht aber typischerweise nur eine Fehlerrate von knapp unter 50%, klassifiziert also nur wenig besser als ein Klassifikator, der zufällig ist. Um jetzt einen starken Klassifikator zu finden, der deutlich besser als Raten ist, wird bei Viola und Jones das maschinelle Lernverfahren AdaBoost[1] verwendet. AdaBoost war der erste Polynomialzeit-Algorithmus, der es erlaubte, dass man mehrere schwache Klassifikatoren zu einem starken Klassifikator kombinieren kann. AdaBoost ist also ein Verfahren, um schwache Klassifikatoren zu finden, die das gesamte Klassifikationsergebnis eines kombinierten Klassifikators verbessern.

Um jetzt ein Bild auf die Präsenz von Gesichtern zu untersuchen, werden Teilfenster des Bildes überprüft. Die Anzahl der zu untersuchenden Teilfenster ist hierbei deutlich höher als die Anzahl der Pixel des Bildes. So fallen für die Untersuchung eines 1 Megapixel großen Bildes – bei einem Klassifikator der auf 19×19 Pixel großen Bildern trainiert wurde – schon ca.

[1] AdaBoost: Kurzform für Adaptives Boosting (Verstärkung). Ein Maschinen-Lernalgorithmus entwickelt von Yoav Freund und Robert Schapire.

2,6 Millionen zu untersuchende Teilfenster an. Auf allen diesen 2,6 Millionen Teilfenstern müssten die mittels des AdaBoost-Verfahrens bestimmten starken Klassifikatoren ausgewertet werden. So wird schnell klar, dass auch mittels der effizienten Berechnung der Features über die Integralbilder einfach zu viele Features ausgewertet werden müssten. Eine Echtzeitfähigkeit lässt sich so noch nicht erreichen.
Hier kommen wir dann zur dritten Kernidee von Viola und Jones. Glücklicherweise sind nicht alle Teilfenster gleich schwierig zu klassifizieren. Ein Teilfenster, das beispielsweise aus einem nahezu einfarbigen Bereich des Bildes stammt (z.B. Himmelsregion), kann auch von einem einzelnen schwachen Klassifikator gut als Nichtgesicht erkannt werden. Aufgrund dieser Erkenntnis haben Viola und Jones die schwachen Klassifikatoren in Form einer Kaskade organisiert.
Diese Kaskade ist so aufgebaut, dass die ersten Stufen der Kaskade aus nur sehr wenigen schwachen Klassifikatoren bestehen, da diese auf allen Teilfenstern berechnet werden müssen. So filtert man die leicht zu klassifizierenden Teilfenster mit relativ wenig Rechenaufwand aus der Liste der Teilfenster, die ein mögliches Gesicht enthalten, heraus. Die folgenden Kaskadenstufen werden dann komplexer, da man weitere schwache Klassifikatoren hinzufügt, die das kombinierte Klassifikationsergebnis verbessern. Die längere Berechnungsdauer der mit mehr schwachen Klassifikatoren ausgestatteten Kaskadenstufen wird durch die deutlich geringere Anzahl an zu bearbeitenden Teilfenstern kompensiert. Denn ein Teilfenster, das von einer Kaskadenstufe einmal als Nichtgesicht klassifiziert worden ist, wird endgültig verworfen und nicht mehr betrachtet. Somit ist es natürlich wichtig, dass jede Kaskadenstufe ein hohe Detektionsrate hat. Toleriert man aber eine hohe Rate an fälschlich als Gesicht erkannten Teilfenstern, so ist diese hohe Detektionsrate gut zu erreichen.

3.1 Features

Die Methode von Viola und Jones zur Erkennung von Objekten trainiert einen Klassifikator auf Basis von Features. Alle verwendeten Features sind

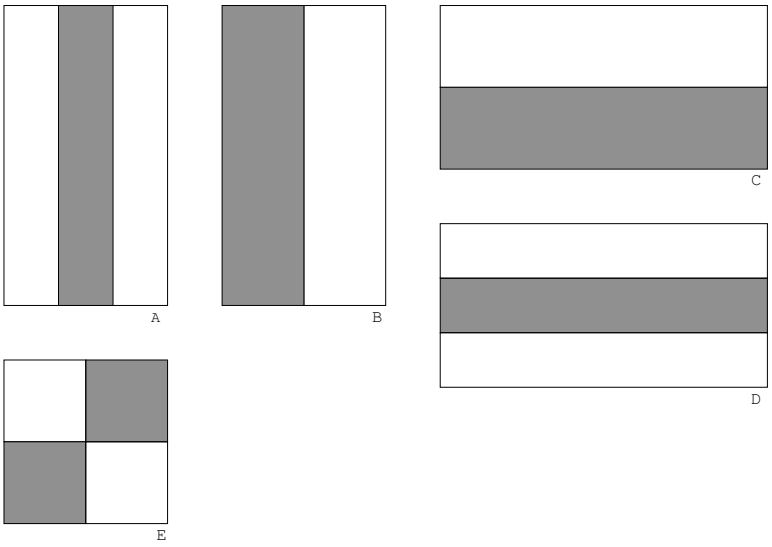


Abbildung 3.1: Alle fünf verwendeten Feature-Typen. Alle Typen sind aus Rechtecken zusammengesetzt.

in der Abbildung 3.1 auf Seite 14 zu sehen. Diese Features werden aus rechteckigen Bereichen zusammengesetzt. Bei allen Features gilt das die weißen Balken von den grauen Balken abgezogen werden. Die beiden Features, die aus zwei Rechtecken bestehen B und C, werden über die Differenz der Summe aller Pixel des grauen Rechtecks sowie der Summe aller Pixel des weißen Rechtecks berechnet. Abbildung 3.2 auf Seite 14 veranschaulicht diese Berechnung auf einem Bild. Die beiden aus drei Rechtecken bestehenden Features (A und D) berechnen die Summe der äußeren Rechtecke und subtrahieren diese von der Summe des mittleren Rechtecks. Das verbleibende Feature (E) wird ebenfalls über die Differenz der grauen zu den weißen Rechtecken berechnet.
Motivation, einen Klassifikator auf Basis solcher Features aufzubauen, ist in mehrfacher Hinsicht gegeben. Ein Feature kodiert Informationen über die gesamte Umgebung, auf der es im Bild berechnet wurde, während ein Pixel diese Umgebungsinformation vermissen lässt. Ein weiteres Argument für den Aufbau eines featurebasierten Systems liegt im Geschwindigkeitsvorteil gegenüber einem pixelbasierten System. Der Grund hierfür ist, dass man bei einem featurebasierten System weniger Werte miteinander verrechnen muss als bei einem Klassifikator, der direkt auf den Pixelwerten arbeitet.

123	78	36	235	131	188	245
45	58	99	187	222	251	131
221	235	34	4	65	97	43
46	78	35	73	77	163	35
97	37	244	56	94	167	872
23	73	137	83	62	86	171
21	77	83	127	193	221	237
84	93	158	200	252	93	54

Abbildung 3.2: Beispielhafte Berechnung eines aus drei vertikalen Balken bestehenden Features (Breite = 3 und Höhe = 4) auf einem Bild. Der Feature-Wert ergibt sich aus der Differenz der Summen aller Pixel eines Rechtecks. Hier -450 + 216 - 298 = -532.

Trotzdem müssen wir zum Trainieren eines Klassifikators auf Trainingsbildern der Größe 19×19 berücksichtigen, dass bei fünf Featuretypen bereits 51705 gültige Feature existieren, die vom Trainingsalgorithmus verarbeitet werden müssen.

3.2 Integralbilder

Die beim Viola-Jones-Verfahren verwendeten schwachen Klassifikator basieren maßgeblich auf Features. Daher müssen sehr viele Features ausgewertet werden. Zudem zielt das Verfahren auf eine Klassifikation in Echtzeit, darum ist die Berechnung dieser Features zeitkritisch für das gesamte Verfahren. Die Berechnung der Rechtecksummen über klassische Iteration mittels for-Schleifen würde das Verfahren für die angestrebte Echtzeittauglichkeit unbrauchbar machen.
Allerdings lassen sich Rechtecksummen sehr effizient über eine einmalig zu berechnende, alternative Repräsentation der zugrundeliegenden Bilder berechnen. Diese Repräsentation wird von Viola und Jones als Integralbild bezeichnet. Ein Integralbild notiert an der Stelle (x,y) die Summe aller Pi-

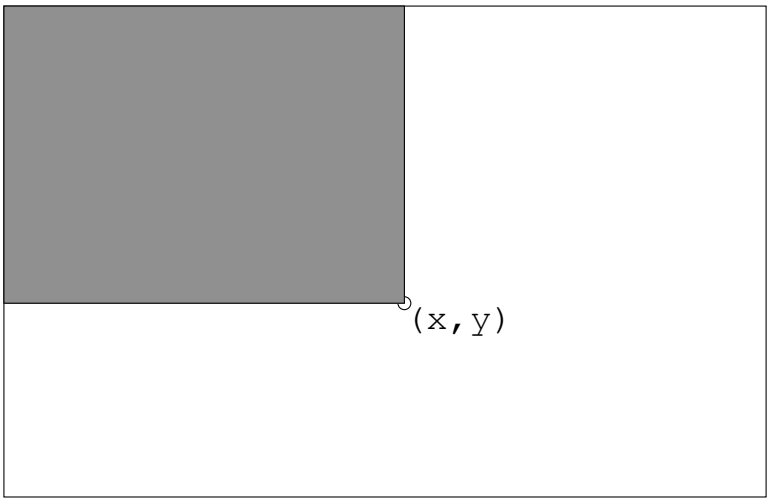


Abbildung 3.3: Der Wert des Integralbildes an der Stelle (x,y) entspricht der Summe aller Pixel links und oberhalb dieser Stelle.

xel links und oberhalb dieser Stelle. Somit ist

$$i(x, y) = \sum_{x'<x, y'<y}^{n} b(x', y'),$$

wobei i(x,y) das Integralbild ist und b(x,y) für das Originalbild steht. Die Berechnung des Integralbildes geschieht in der hier vorgestellten Implementation in zwei Schritten (siehe Algorithmus 1 auf Seite 17 sowie Abbildung 3.4 auf Seite 17).

Zunächst werden spaltenweise die Summen berechnet, die bei der Addition aller Pixel oberhalb eines Eintrags entstehen. Diese Summen werden in einem temporären Bild der Größe (w,h+1) gespeichert, sofern das Ursprungsbild die Größe (w,h) hat. Anschließend wird durch zeilenweiser Summierung dieser Zwischensummen das endgültige Integralbild berechnet. Die endgültige Dimension des Integralbildes entspricht schließlich (w+1,h+1).

Über das Integralbild lassen sich nun beliebige Rechtecksummen mit nur vier Speicherzugriffen berechnen (siehe Abbildung 3.5 auf Seite 18). Somit können die Features, die auf zwei Rechtecksummen basieren, mit acht Speicherzugriffen berechnet werden. Da alle Features aber auf benachbarten Rechtecken basieren, lässt sich die Anzahl der Speicherzugriffe auf sechs

Algorithmus 1 Die Berechnung der Integralbilder in Pseudocode

Eingabe:
Bild b
$W \leftarrow$ Breite des Bildes b
$H \leftarrow$ Höhe des Bildes b

{Berechne in einem temporären Bild t zunächst spaltenweise die Summen, die sich bei der Addition aller Pixel oberhalb eines Eintrags ergeben.}

```
1: for w = 0, ..., W do
2:    summe = 0
3:    for h = 0, ..., H + 1 do
4:       t(w, h) ← summe
5:       if (h < H) then
6:          summe ← (summe + b(w, h))
7:       end if
8:    end for
9: end for
```

{Berechne mittels des temporären Bildes t nun zeilenweise die Summen, die sich bei der Addition aller Pixel links eines Eintrags ergeben.}

```
10: for h = 0, ..., H + 1 do
11:    summe = 0
12:    for w = 0, ..., W + 1 do
13:       i(w, h) ← summe
14:       if (h < H) then
15:          summe ← (summe + t(w, h))
16:       end if
17:    end for
18: end for
```

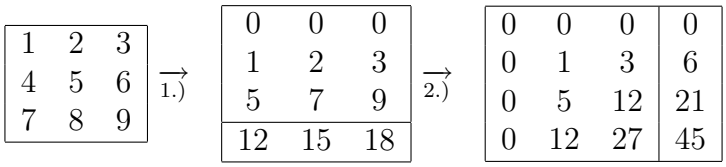


Abbildung 3.4: Beispiel einer Berechnung eines Integralbildes: 1.) Zunächst werden die Spaltensummen berechnet und 2.) dann mittels der Zeilensummen das endgültige Integralbild.

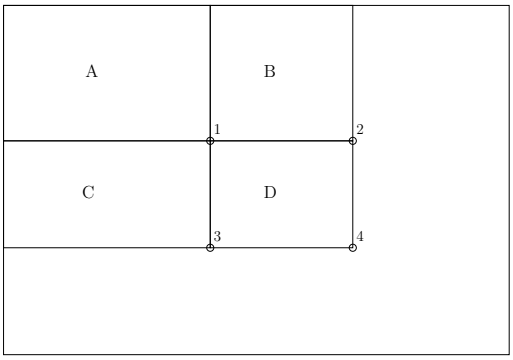


Abbildung 3.5: Über das Integralbild lässt sich die Summe der Pixel des Rechtecks D mit vier Speicherzugriffen berechnen. Der Wert des Integralbildes an der Position 1 entspricht der Summe der Pixel des Rechtecks A. An Position 2 haben wir die Summe A + B, sowie an Position 3 die Summe A + C. Position 4 entspricht der Summe A + B + C + D. Somit lässt sich die Summe aller Pixel in D mittels 4 + 1 - (2 + 3) ermitteln.

reduzieren (siehe Abbildung 3.6 auf Seite 19). Für die auf drei Rechtecksummen basierenden Feature-Typen kommt man auf acht Speicherzugriffe, sowie auf neun für Features, die über vier Rechtecksummen bestimmt werden.

3.3 Kontrast- und Beleuchtungsausgleich

Da der Klassifikator unabhängig von Beleuchtungs- sowie Kontrastverhältnissen sein soll, muss eine entsprechender Kontrast- und Beleuchtungsausgleich auf den Quellbildern geschehen. So soll verhindert werden, dass schlecht beleuchtete Gesichter anders als gut beleuchtete Gesichter behandelt werden und in Folge dessen eventuell nicht erkannt werden. Ein Beleuchtungsausgleich lässt sich durch Subtraktion des Mittelwertes von jedem Pixel des Bildes erreichen. Der Kontrastausgleich wird realisiert, indem alle Pixel des Bildes durch die Standardabweichung σ geteilt werden.

Den Beleuchtungsausgleich können wir aber jetzt nicht als direkten Vorverarbeitungsschritt auf den Quellbildern ausführen. Denn wir wollen keinen

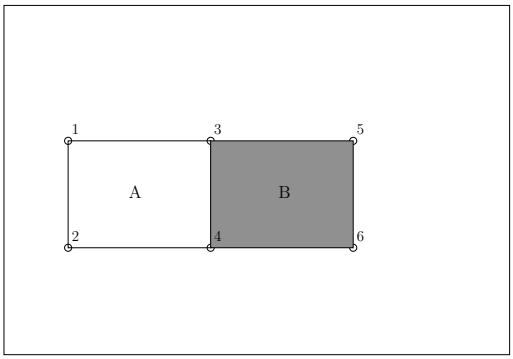


Abbildung 3.6: Über das Integralbild lässt sich der Wert eines Features vom Typ B über sechs Arrayzugriffe sowie fünf einfache Rechenoperationen (Plus/Minus) berechnen. In der Abbildung sieht man, dass sich die angrenzenden Rechtecke A und B in den Punkten 3 und 4 auf die gleichen Informationen des Integralbildes beziehen und somit nur sechs anstelle von acht Speicherzugriffen nötig sind.

Beleuchtungsausgleich über das gesamte Bild machen, sondern nur auf den Teilbereich, des Bildes der gerade untersucht wird. Sonst würden wir beispielsweise bei einem Bild, das teilweise überbelichtet aber auch teilweise unterbelichtet ist, wahrscheinlich kein Gesicht mehr erkennen können. Außerdem benötigen wir auf allen symmetrischen Features (also Features, bei denen genauso viele Pixel des Bildes addiert wie subtrahiert werden) gar keinen Beleuchtungsausgleich, da hier der Mittelwert n-mal addiert wird und n-mal subtrahiert wird und somit in der Summe Null ergibt. Daher liegt es nahe, den Beleuchtungsausgleich direkt in die Berechnung der Features einfließen zu lassen, die diesen auch benötigen. Bei den von uns verwendeten Features bleiben also die aus drei Balken bestehenden Typen A und D (siehe Abbildung 3.1 auf Seite 14), bei denen ein Beleuchtungsausgleich notwendig ist. Der Mittelwert eines Bildes b wird über die Formel

$$\overline{x} = \frac{1}{N} \sum_{\substack{w=1,\\h=1}}^{\substack{w<W,\\h<H}} b(w,h)$$

berechnet, also die Summe über alle Pixel des Bildes geteilt durch die Anzahl

aller Pixel. Um dies in konstanter Zeit zu ermöglichen, greifen wir wieder auf unsere Integralbilder zurück, die ja die benötigten Summen schon vorrätig haben und sich über

$$\overline{x} = \frac{i(w,h)}{(w-1)\cdot(h-1)}$$

berechnen lassen. i steht hier für das verwendete Integralbild, w für die Breite und h für die Höhe des Integralbildes. Den so berechneten Mittelwert des Quellbildes nutzen wir zur Korrektur der Features A und D. Diese beiden aus drei Balken bestehenden Features subtrahieren die äußeren Balken von dem in der Mitte liegenden Balken. Dadurch wird doppelt so oft subtrahiert wie addiert; um dies nun zu kompensieren wird der Mittelwert für alle Pixel eines äußeren Balkens addiert und man erreicht so einen Beleuchtungsausgleich für die Features A und D.
Im Gegensatz zum Beleuchtungsausgleich muss der Kontrastausgleich bei allen Featuretypen durchgeführt werden. Hierfür werden alle Pixel des Bildes durch die Standardabweichung σ geteilt. Da wir aber den Kontrastausgleich auch wie den Beleuchtungsausgleich wieder während der Featureberechnung erledigen wollen, teilen wir einfach den berechneten Feature-Wert durch die Standardabweichung σ. Wie berechnen wir jetzt die Standardabweichung σ? Die Formel

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

berechnet die Standardabweichung, wobei $\overline{x}$ wieder für den Mittelwert steht. Um diese Berechnung mit einem Integralbild durchzuführen, würden wir jetzt aber wieder ein neues Integralbild benötigen, das bereits auf den Subfenstern mittelwertbefreit ist. Aus Effizienzgründen möchten wir aber auch den Kontrastausgleich in unsere Featureberechnung integrieren. Dies lässt sich erreichen, da sich die Berechnung der Standardabweichung σ zu

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}x_i^2 - \overline{x}^2}$$

vereinfachen lässt. Mittels dieser Formel können wir einmalig das Integralbild und ein Integralbild der quadrierten Einträge berechnen. Diese beiden Bilder müssen dann nicht für jedes Subfenster, auf dem ein Beleuchtungsausgleich, gemacht werden soll, sondern nur einmalig. Die Gleichheit der Formeln lässt sich wie folgt nachrechnen:

$$\begin{aligned}\sum_{i=1}^{N}(x_i-\overline{x})^2 &= \sum_{i=1}^{N}(x_i^2-2x_i\overline{x}+\overline{x}^2)\\ &= \sum_{i=1}^{N}x_i^2-2\overline{x}\sum_{i=1}^{N}x_i+N\overline{x}^2\\ &= \sum_{i=1}^{N}x_i^2-2\overline{x}N\overline{x}+N\overline{x}^2\\ &= \sum_{i=1}^{N}x_i^2-2N\overline{x}^2+N\overline{x}^2\\ &= \sum_{i=1}^{N}x_i^2-\overline{x}^2.\end{aligned}$$

Dank dieser Vereinfachung benötigen wir nur noch die Summe $\sum_{i=1}^{N}x_i^2$ den verwendeten Mittelwert bestimmen wir wie bisher über die Integralbilder. Die Summe lässt sich aber auch einfach und in konstanter Zeit berechnen, wenn wir neben unserem Integralbild außerdem ein quadratisches Integralbild anlegen, also erst alle Pixel eines Bildes quadrieren und anschließend über den bekannten Weg ein Integralbild zu diesem quadrierten Bild berechnen.

3.4 AdaBoost

Mit den in Kapitel 3.1 vorgestellten Features möchten wir nun einen Klassifikator für Gesichter konstruieren. Hierfür verwenden wir den AdaBoost-Klassifikator, der auf dem Prinzip des Boostings basiert.
Boosting bietet sich an, wenn wir für ein Problem eine Menge von schwachen Klassifikatoren haben. Das sind Klassifikatoren, die nur wenig besser klassifizieren als Raten, also Fehlerraten von fast 50% besitzen. Nun möchten wir diese schwachen Klassifikatoren zu einem starken Klassifikator kombinieren, der wesentlich besser klassifiziert als die einzelnen schwachen Klassifikatoren. AdaBoost war der erste Algorithmus, der Boosting praktisch einsetzbar machte, da er beweisbar eine polynomielle Laufzeit hat. Er wurde 1989 von Robert E. Schapire [Sch90] entwickelt.

Der von AdaBoost trainierte Klassifikator hat folgende Gestalt:

$$h(x) = \begin{cases} 1 & \underbrace{\sum_{t=1}^{T} \alpha_t h_t(x)}_{\text{Entscheidungswert}} \geq \underbrace{\frac{1}{2} \sum_{t=1}^{T} \alpha_t}_{\text{Schwellenwert}} \\ 0 & \text{sonst.} \end{cases}$$

Dabei ist x die Eingabe des Klassifikators (in unserem Fall ein Bild), $h(x)$ die Ausgabe des starken Klassifikators, h_t ist ein schwacher Klassifikator und α_t das Gewicht zu dem schwachen Klassifikator. Die Struktur des starken Klassifikators gleicht also einem Perzeptron. Die schwachen Klassifikatoren für unser Problem der Gesichtsdetektion haben folgende Struktur:

$$h_w(x) = \begin{cases} 1 & \sigma f(x) \geq \sigma \vartheta \\ 0 & \text{sonst} \end{cases}$$

Dabei ist $h_w(x)$ die Ausgabe des schwachen Klassifikators, $f(x)$ der Wert eines Features, ϑ ist der Schwellenwert, und $\sigma \in \{-1, 1\}$ bestimmt, auf welcher Seite des Schwellenwertes die Gesichter liegen. Für jedes Feature lässt sich ein solcher schwacher Klassifikator generieren.

Wie lässt sich nun ein starker Klassifikator trainieren? Das heißt, wie wählen wir schwache Klassifikatoren mit den zugehörigen Gewichten aus, so dass die Klassifikationsleistung optimal wird? Zunächst wählen wir den besten schwachen Klassifikator, das heißt den schwachen Klassifikator, der die kleinste Fehlerrate hat, aus. Der nächste schwache Klassifikator soll nun möglichst die Fehler des ersten kompensieren, das heißt Bilder, die der erste schwache Klassifikator falsch klassifizierte, korrekt klassifizieren. Zu diesem Zweck haben wir für jedes Trainingsbeispiel ein Gewicht w_i. Bilder, die vom ersten schwachen Klassifikator falsch klassifiziert wurden, bekommen ein größeres Gewicht; umgekehrt bekommen korrekt klassifizierte Bilder ein kleineres Gewicht. Den zweiten schwachen Klassifikator wählen wir nun so aus, dass der gewichtete Fehler minimal ist. Der gewichtete Fehler ist die Summe der Gewichte der falsch klassifizierten Trainingsbeispiele. Auf diese

Algorithmus 2 Der Lernalgorithmus AdaBoost in Pseudocode

Eingabe:

$(x_1, y_1), \ldots, (x_n, y_n)$, x_i: Trainingsbilder y_i: Bildklasse

Initialisierung:

$$w_{1,i} = \begin{cases} \frac{1}{2l} & \text{falls } y_i = 1 \text{ (Gesicht)} \\ \frac{1}{2m} & \text{falls } y_i = 0 \text{ (Nichtgesicht)}, \end{cases}$$

wobei m und l für die Anzahl der Nichtgesichter bzw. Gesichter steht.

1: **for** $t = 1, \ldots, T$ **do**

2: Normalisierung der Gewichte,

$$\hat{w}_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$

damit der Gewichtsvektor w_t einer Wahrscheinlichkeitsverteilung entspricht. Anschließend $w_{t,i} \leftarrow \hat{w}_{t,i}$.

3: **for** jedes mögliche Feature j **do**

4: Trainiere einen schwachen Klassifikator h_j zu dem Feature j.

5: Berechne den gewichteten Fehler ϵ_j von h_j mittels $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$.

6: **end for**

7: Wähle den Klassifikator h_t mit dem kleinsten gewichteten Fehler ϵ_t.

8: Aktualisiere den Gewichtsvektor $\vec{w}$:

$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i},$$

wobei

$$e_i = \begin{cases} 1 & x_i \text{ wurde falsch klassifiziert} \\ 0 & \text{sonst} \end{cases}$$

und

$$\beta_t = \frac{\epsilon_t}{1 - \epsilon_t}.$$

9: **end for**

10: Der starke Klassifikator nach T Runden ist dann:

$$h(x) = \begin{cases} 1 & \underbrace{\sum_{t=1}^{T} \alpha_t h_t(x)}_{Entscheidungswert} \geq \underbrace{\frac{1}{2} \sum_{t=1}^{T} \alpha_t}_{\text{Schwellenwert}} \\ 0 & \text{sonst} \end{cases}$$

mit $\alpha_t = \log \frac{1}{\beta_t}$.

Weise ist es wahrscheinlich, dass die Trainingsbeispiele, die ein hohes Gewicht haben, also vorher falsch klassifiziert wurden, nun richtig klassifiziert werden. Auch nach Auswahl des zweiten schwachen Klassifikators werden die Gewichte der Trainingsbeispiele vergrößert, die vom zweiten schwachen Klassifikator falsch klassifiziert wurden. Entsprechend werden auch wieder die Gewichte der korrekt klassifizierten Bilder verkleinert.
Die weiteren schwachen Klassifikatoren werden jetzt auf dieselbe Art ausgewählt, solange bis ausreichend viele schwache Klassifikatoren gefunden wurden, um ein vorher spezifiziertes Klassifikationsziel zu erreichen.
Nachdem nun mittels AdaBoost ausreichend viele schwache Klassifikatoren gefunden wurden, um ein gewünschtes Klassifikationsergebnis zu erreichen, stellt sich die Frage, wie man die gefundenen schwachen Klassifikatoren zu einem starken Klassifikator kombiniert. Wie bereits erwähnt ähnelt der starke Klassifikator in seiner Struktur einem Perzeptron, und wie bei einem Perzeptron werden die schwachen Klassifikatoren als gewichtete Eingaben benutzt. Aus diesen gewichteten Eingaben der schwachen Klassifikatoren werden dann die Ausgaben des starken Klassifikators bestimmt. Die hierbei den schwachen Klassifikatoren zugeordneten Gewichte werden direkt aus ihrer Klassifikationsleistung über die Formel

$$\alpha_t = \log \frac{(1 - \epsilon_t)}{\epsilon_t}$$

bestimmt. Hierbei ist α_t das Gewicht des t-ten schwachen Klassifikators und ϵ_t der gewichtete Fehler des t-ten schwachen Klassifikators. Je kleiner also der gewichtete Fehler ϵ_t des schwachen Klassifikators beim Training war, desto stärker trägt dieser schwache Klassifikator zur Gesamtausgabe des starken Klassifikators bei, während die schwachen Klassifikatoren mit einem größeren gewichteten Fehler ϵ_t entsprechend weniger Einfluss auf das Gesamtergebnis des starken Klassifikators haben.
Unter Algorithmus 2 auf Seite 23 ist der AdaBoost-Klassifikator in Pseudocode angegeben.

3.5 Training der schwachen Klassifikatoren

Für den AdaBoost-Klassifikator haben wir schwache Klassifikatoren zu einem starken Klassifikator kombiniert. Ein solcher schwacher Klassifikator hat folgende Struktur:

$$h_{\mathrm{w}}(x) = \begin{cases} 1 & \sigma f(x) \geq \sigma\vartheta \\ 0 & \text{sonst} \end{cases}.$$

Hierbei ist $f(x)$ der Wert des Features f auf dem zu klassifizierenden Bild x, σ ist das Vorzeichen und ϑ der Schwellenwert des schwachen Klassifikators. Für den AdaBoost-Klassifikator ist es nun wichtig aus der unendlichen Menge schwacher Klassifikatoren den besten schwachen Klassifikator auszuwählen, also das Feature (bestehend aus dem Feature-Typ, Startkoordinate (x,y), Breite und Höhe), das mit einem Schwellenwert am besten alle gegebenen Bilder in die Klassen Gesicht und Nichtgesicht aufteilt.
Um diesen besten schwachen Klassifikator zu finden, unternimmt man folgende Schritte. Man berechnet für jedes gültige Feature den gewichteten Fehler, den das Feature bei einem optimalen Schwellenwert ϑ auf den Trainingsbildern macht. Für unseren schwachen Klassifikator wird schließlich das Feature ausgewählt, das den kleinsten gewichteten Fehler auf den Trainingsdaten hat. Um jetzt zu verstehen, wie dieser gewichtete Fehler für ein ausgewähltes Feature berechnet wird, schauen wir uns die Berechnung anhand eines Features genau an. Hierfür wählen wir uns also ein Feature aus, das bedeutet man wählt einen Feature-Typen mit Startkoordinaten auf dem Bild sowie Breite und Höhe aus. Dieses Feature wird nun auf allen Bildern der Trainingsmenge berechnet. Hierdurch erhält jedes Bild der Trainingsmenge einen zum ausgewählten Feature gehörenden Wert. Diese Werte ordnen wir jetzt sortiert auf einem Zahlenstrahl an (siehe Abbildung 3.7 auf Seite 26), um den Schwellenwert für dieses Feature zu bestimmen. Die Hoffnung ist jetzt, dass sich bei einem guten Feature, die Werte der Gesichter räumlich gruppieren, beispielsweise eher am Ende des Zahlenstrahls zu finden sind und Nichtgesichter dann entsprechend eher am Anfang des Zahlenstrahls. Bei solch einem Feature lassen sich die beiden Klassen Ge-

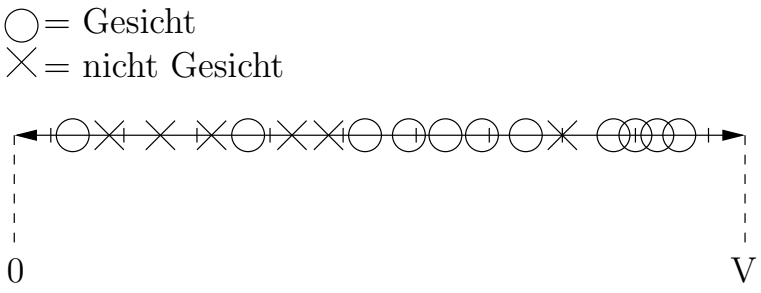


Abbildung 3.7: Beispiel für einen Zahlenstrahl, der durch die Berechnung des ausgewählten Features auf den Trainingsbildern entsteht. V steht hier für den größten Feature-Wert, da der Wertebereich für die Feature-Werte nicht beschränkt ist.

sicht und Nichtgesicht gut durch einen Schwellenwert in der Mitte trennen.

Zum Bestimmen dieses Schwellenwertes betrachten wir nun auf unserem Zahlenstrahl die Menge aller Schwellenwerte, die aus Sicht der Klassifikation unterschiedliche Ergebnisse liefern. Bezogen auf unseren reellen Zahlenstrahl sind dies alle Werte, welche die Mitte zwischen zwei Featurewerten bilden sowie zwei Werte für die Ränder des Zahlenstrahls. Nun ist die Idee, diese Menge potentieller Schwellenwerte durchzugehen und dabei für jeden Schwellenwert den gewichteten Fehler $\varepsilon(\vartheta)$ zu berechnen. Hierfür vergleichen wir jedes Bild auf den Zahlenstrahl mit dem aktuellen Schwellenwert und testen, ob es mit diesem Schwellenwert korrekt klassifiziert würde. Ist dies nicht der Fall, addieren wir zu unserem $\varepsilon(\vartheta)$ das Gewicht des gerade getesteten Bildes. Anschließend kann der Schwellenwert verwendet werden, der den kleinsten gewichteten Fehler $\varepsilon(\vartheta)$ hatte.
Nun ist das Auswerten des aktuellen gewichteten Fehlers $\varepsilon(\vartheta)$ relativ aufwendig. Besser wäre es, wenn wir diesen Schritt nicht für jeden in Frage kommenden Schwellenwert ausführen müssten. Wenn wir nur einmal den gewichteten Fehler für den kleinsten sinnvollen Schwellenwert (der Schwellenwert links oder rechts vom Rand des Zahlenstrahls) auswerten, können wir uns das ständige Neuberechnen des gewichteten Fehlers sparen, indem wir fortan einfach den gewichteten Fehler aktualisieren. Denn von einem potentiellen Schwellenwert zum nächsten ändert sich ja maximal nur die Klassifikation eines Trainingsbeispiels. So können wir mittels eines Durchlaufs durch den Zahlenstrahl zu jedem potentiellen Schwellenwert den ge-

wichteten Fehler $\varepsilon(\vartheta)$ bestimmen.

Im Detail läuft diese Suche jetzt wie folgt ab. Um den von einem Schwellenwert produzierten Fehler zu bestimmen, müssen wir zunächst festlegen, auf welcher Seite des Schwellenwertes die Gesichter sein sollen. In unserer Terminologie des schwachen Klassifikators $h_w(x)$ (vgl. Definition von $h_w(x)$ auf Seite 25) bestimmen wir also das σ. Haben wir uns nun beispielsweise entschieden, dass Gesichter links vom Schwellenwert liegen und wir auch von links gesehen durch unseren Zahlenstrahl gehen (jeder Mittelwert zwischen zwei Featurewerten ist ja ein potentieller Schwellenwert), müssen wir also zunächst die Anzahl der Gesichter zählen, da wir diese ja auch alle falsch klassifizieren würden, wenn wir diesen Schwellenwert links des Zahlenstrahls nehmen. Von diesem initialen Fehler ausgehend laufen wir nun nach rechts über den Zahlenstrahl. Laufen wir an einem Gesicht vorbei, haben wir einen Fehler weniger gemacht, während wir einen weiteren Fehler machen, wenn wir an einem Nichtgesicht vorbei laufen.

Haben wir nun über das Bild mit dem größten Feature-Wert das Ende des Zahlenstrahls erreicht, wählen wir die Position für den Schwellenwert, an der wir die wenigsten Fehler gemacht haben. Beachten müssen wir natürlich, dass wir beide Bedeutungen des Schwellenwertes ($\sigma = -1$ bedeutet Gesichter liegen links des Schwellenwertes, $\sigma = 1$ bedeutet Gesichter liegen rechts des Schwellenwertes) durchtesten müssen, da nur so sichergestellt ist, dass wir einen optimalen schwachen Klassifikator bezüglich eines gegebenen Features finden. Die Variante, dass die Gesichter rechts vom Schwellenwert liegen, läuft aber völlig analog zum ersten Fall ab, bis darauf dass initial die Nichtgesichter gezählt werden und ein Passieren eines Gesichts den Fehler erhöht und nicht das Passieren eines Nichtgesichts.

Auf diese Art und Weise können wir jetzt zu einem gegebenen Feature ein σ sowie einen Schwellenwert bestimmen und erhalten in Kombination den optimalen schwachen Klassifikator bezüglich des gegebenen Features. Um jetzt den besten schwachen Klassifikator bezüglich aller möglichen Features zu finden, iterieren wir über alle möglichen Features. Für jedes mögliche Feature bestimmen wir dann wieder auf bekannte Art die passende Interpre-

Algorithmus 3 Der Lernalgorithmus für die schwachen Klassifikatoren in Pseudocode

Eingabe:

$(x_1, y_1), \ldots, (x_n, y_n)$, x_i: Trainingsbilder, y_i: Bildklasse
Gewichtsvektor $\vec{w}$ mit Gewichten zu jedem Bild des Trainingsvektors.

for jedes mögliche Feature j **do**
  Berechne Liste, die zu jeden gegebenen Bild x_i der Trainingsmenge den zugehörigen Featurewert $f_j(x_i)$ bereithält.
  Sortiere die Liste aufsteigend als $\hat{x}_i$, ursprünglicher Index: $o(i)$
  for $\sigma \in \{-1, 1\}$ **do**
    if $\sigma = -1$ **then**

$$\varepsilon = \sum_{\substack{y_i = \\ \text{Gesicht}}} w_i$$

      {Addiere alle Gewichte von Gesichtern auf der Liste}
    else if $\sigma = 1$ **then**

$$\varepsilon = \sum_{\substack{y_i = \\ \text{Nichtgesicht}}} w_i$$

      {Addiere alle Gewichte von Nichtgesichter auf der Liste}
    end if
    Initialisiere $\vartheta = f(\hat{0}) - 1$, $\epsilon_{min} = \epsilon$
    for jede Position i der Liste **do**
      {Berechne gewichteten Fehler ϵ_i zum i-ten Schwellenwert}
      if Bild $\hat{x}_i$ ist ein Gesicht **then**
        aktualisiere Fehler: $\varepsilon = \varepsilon + (w_{o(i)} \cdot \sigma)$
      else if Bild $\hat{x}_i$ ist ein Nichtgesicht **then**
        aktualisiere Fehler: $\varepsilon = \varepsilon$ - $(w_{o(i)} \cdot \sigma)$
      end if
      if $\epsilon_i < \epsilon_{min}$ **then**
        $\epsilon = \epsilon_{min}$ {Merke kleinsten Fehler}
        $\vartheta = \frac{f(\hat{x}_i) + f(\hat{x}_{i+1})}{2}$ {Merke Schwellenwert}
      end if
    end for
  end for
end for

tation des Schwellenwertes σ sowie den Schwellenwert ϑ selber. Aus dieser Suche über alle Features erhalten wir dann den besten schwachen Klassifikator (also Feature f, das Vorzeichen σ sowie den Schwellenwert ϑ).
Für den AdaBoost-Klassifikator reicht ein optimaler schwacher Klassifikator auf Basis eines absoluten Fehlers aber nicht aus. AdaBoost benötigt einen Algorithmus, der einen optimalen schwachen Klassifikator auf Grund eines gewichteten Fehlers bestimmt (vergleiche hierfür Kapitel 3.4). Daher modifizieren wir unseren Algorithmus zur Bestimmung des optimalen schwachen Klassifikators entsprechend. Dies ist relativ einfach machbar. Dafür müssen wir nur die Berechnung des absoluten Fehlers abändern. Hier wird nun das Gewicht des Trainingsbeispiels verrechnet und nicht der absolute Fehler um eins inkrementiert bzw. dekrementiert, wenn beim Durchtesten der Schwellenwerte ein Bild passiert wird. Der Schwellenwert wird dann entsprechend an die Position auf den Zahlenstrahl gesetzt, die den kleinsten gewichteten Fehler hat.
Zu beachten ist bei der Rechnung mit den Gewichten der Trainingsbeispiele noch, dass man beim Sortieren der Feature-Werte die Zuordnung (also den Index) zu den Gewichten merken muss, damit auch sichergestellt ist, dass das richtige Gewicht in den gewichteten Fehler einbezogen wird. Mit dieser Anpassung an den gewichteten Fehler sind alle Mittel vorhanden, um einen schwachen Klassifikator zu trainieren. Algorithmus 3 auf Seite 28 fasst die Schritte zum Training eines schwachen Klassifikators zusammen.

3.6 Die Kaskade

Mittels des AdaBoost-Klassifikators haben wir die Möglichkeit, unsere schwachen Klassifikatoren zu einem starken Klassifikator zu kombinieren. Dieser starke Klassifikator wird besser, je mehr schwache Klassifikatoren zu einem starken Klassifikator kombiniert werden. Leider wird der starke Klassifikator aber mit jedem zusätzlichen schwachen Klassifikator auch immer langsamer. Daher ist es nicht praktikabel, einen einzigen starken Klassifikator zu trainieren, der anschließend die gesamte Klassifikation übernimmt. Dies zeigt sich zum Beispiel schnell, wenn man versucht, mittels AdaBoost einen

Klassifikator zu trainieren, der echtzeitfähig ist. Möchte man einen starken Klassifikator der eine hohe Detektionsrate bei niedriger Falsch-Positiv-Rate hat erhalten so wird AdaBoost diesen starken Klassifikator nur mit relativ vielen schwachen Klassifikatoren realisieren können. Diese vielen schwachen Klassifikatoren sind aber problematisch, bedenkt man die riesige Auswahl an Subfenstern, die der Klassifikator durchtesten muss. Der Klassifikator soll in der Lage sein, beliebig große Gesichter zu detektieren, daher kann der Klassifikator skaliert werden (siehe Abbschnitt 3.8 auf Seite 39), um auf Teilbereichen beliebiger Größe zu arbeiten. Bei einem 1 Megapixel großen Bild und 20 durchgeführten Skalierungen haben wir beispielsweise schon 2,6 Millionen zu untersuchende Subfenster. Und selbst mit der effizienten Berechnung der Features mittels der Integralbilder ist so eine Echtzeitfähigkeit nicht zu erreichen. Entscheiden ist hier der Punkt, dass für viele Subfenster in einem typischen Bild schon wenige schwache Klassifikatoren ausreichen um diese korrekt zu klassifizieren. Beispielsweise wäre hier an einen weißen Hintergrund bei einem Gruppenfoto zu denken. Hier würde die komplette Berechnung eines einzigen starken Klassifikators mit vielen schwachen Klassifikatoren wahrscheinlich das richtige Ergebnis Nichtgesicht (eine weiße Wand ist kein Gesicht) errechnen, doch für diese vergleichsweise einfache Klassifikation werden nur wenige starke Klassifikatoren benötigt.
Aus diesem Grund wird bei Viola und Jones ein kaskadenartiger Aufbau (siehe Abbildung 3.8 auf Seite 31) verwendet. Man arbeitet also nicht mit einem starken Klassifikator, sondern mit mehreren starken Klassifikatoren, die nacheinander geschaltet werden. Hierbei werden alle Subfenster zunächst der ersten Stufe präsentiert. Bilder, die von der ersten Stufe als mögliches Gesicht klassifiziert werden, werden an die nächste Stufe weitergereicht. Die restlichen Bilder, die nicht als mögliches Gesicht in Fragen kommen, werden verworfen. Die folgenden Stufen arbeiten genauso.
Die Idee ist nun, dass die erste Stufe nur relativ wenige schwache Klassifikatoren hat, dabei aber trotzdem eine hohe Detektionsrate hat. Dies zu erreichen scheint zunächst schwierig; gleichzeitig zur hohen Detektionsrate wird aber auch eine hohe Falsch-Positiv-Rate zugelassen. Das bedeutet: Die erste Stufe darf ruhig relativ viele Subfenster fälschlicherweise als Gesicht klassifizieren, da diese Subfenster alle noch einmal den nachfolgenden Stu-

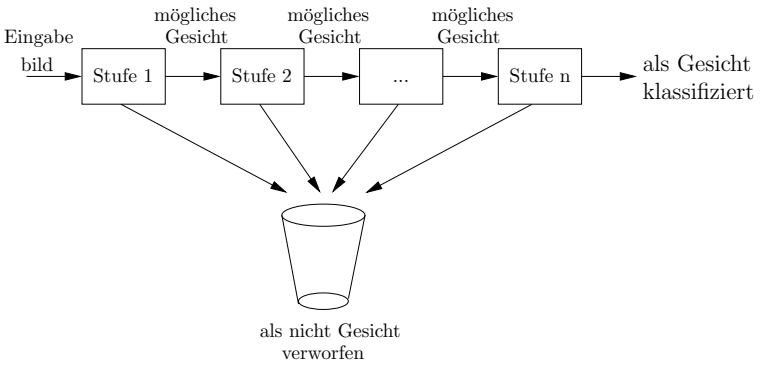


Abbildung 3.8: Kaskadenartiger Aufbau des Klassifikators. Anstelle von einem starken Klassifikator, werden aus Effizienzgründen n starke Klassifikatoren trainiert.

fen genauer betrachtet werden. Mit „genauer“ ist gemeint, dass die Anzahl der verwendeten schwachen Klassifikatoren mit jeder Kaskadenstufe steigt. Dies ist sinnvoll, wenn man bedenkt, dass die Klassifikation als mögliches Gesicht für die späteren Stufen der Kaskade deutlich schwieriger ist als für die ersten Stufen, da diese ja die Subfenster, die sehr leicht als Nichtgesicht zu identifizieren sind, schon herausgefiltert haben.
Wie trainiert man nun eine Kaskade, die dies leistet? Man gibt feste Detektionsraten und Falsch-Positiv-Raten vor, die die einzelnen Stufen jeweils erreichen müssen. Hierbei ist wichtig, dass die Detektionsrate nahezu eins ist, also fast alle Gesichter erkannt werden. Denn die Kaskade ist ja so aufgebaut, dass sie ein Subfenster nicht mehr betrachtet, sobald eine Stufe es als Nichtgesicht klassifiziert hat. Auf der anderen Seite ist es weniger schlimm, wenn eine Kaskadenstufe fälschlicherweise ein Bild als Gesicht labelt, da ja noch die weiteren Stufen das Bild klassifizieren. Durch diese Struktur der Kaskade können wir also eine hohe Falsch-Positiv-Rate von fast 0,5 erlauben, die es dann ermöglicht, dass man die erste Kaskadenstufe mit wenigen schwachen Klassifikatoren trainiert bekommen kann.

3.6.1 Training von starken Klassifikatoren für die Kaskade

Um starke Klassifikatoren entsprechend den von uns vorgegebenen Klassifikationsraten zu trainieren, müssen wir AdaBoost entsprechend anpas-

sen, da wir jetzt nicht mehr eine feste Anzahl von zu trainierenden schwachen Klassifikatoren vorgeben, sondern unsere gewünschte Detektionsrate und Falsch-Positiv-Rate. Da wir diese Raten nicht auf der Trainingsmenge berechnen können, teilen wir unsere Trainingsdaten gleichmäßig in eine Trainingsmenge und eine Validierungsmenge auf. Diese beiden Mengen bekommt der AdaBoost-Klassifikator mit dem Ziel, einen starken Klassifikator zu konstruieren, der eine Detektionsrate von nahezu eins und eine Falsch-Positiv-Rate von etwa 0,5 hat.
Hierfür müsste AdaBoost also wissen, mit wievielen schwachen Klassifikatoren die vorgegebenen Raten erreicht werden. Da dies nicht bekannt ist, werden zwei Änderungen eingeführt: Zum einen wird der feste Schwellenwert $\ldots \geq \frac{1}{2}\sum_{t=1}^{T}\alpha_t$ angepasst, zum anderen wird nach jedem Durchlauf anhand der Validierungsmenge überprüft, wie hoch die Falsch-Positiv-Rate des bisherigen starken Klassifikators ist. Der Schwellenwert des AdaBoost-Klassifikators wird anhand der vorgegebenen Detektionsrate eingestellt, indem man mittels der Detektionsrate zunächst berechnet, wieviele Gesichter der starke Klassifikator erkennen muss. Nun baut man wieder einen Zahlenstrahl auf, dessen Einträge aus Entscheidungswerten, also aus den Summen der α-Werte der schwachen Klassifikatoren bestehen, die das zugehörige Bild als Gesicht klassifizieren. Gibt es beispielsweise zwei α-Werte (da bisher zwei schwache Klassifikatoren trainiert wurden), werden diese Werte genutzt, um den Zahlenstrahl aufzubauen. Für jedes Bild der Validierungsmenge werden nun die α-Werte der schwachen Klassifikatoren aufsummiert, die das Bild der Validierungsmenge als Gesicht klassifizieren. Anschließend werden die berechneten Werte sortiert.

Auf diesem Zahlenstrahl (siehe Abbildung 3.9 auf Seite 33) hat ein Bild also einen umso höheren Wert, je mehr schwache Klassifikatoren es als Gesicht klassifizierten. Nun ist die Idee, einfach von links kommend den Zahlenstrahl solange hinab zugehen, bis entsprechend der Detektionsrate ausreichend viele Gesichter passiert wurden. Die gefundene Stelle wird dann als Schwellenwert für den starken Klassifikator genommen. Der verwendete Algorithmus zur Bestimmung des Schwellenwertes des starken Klassifikators ist im Algorithmus 4 auf Seite 34 angegeben.

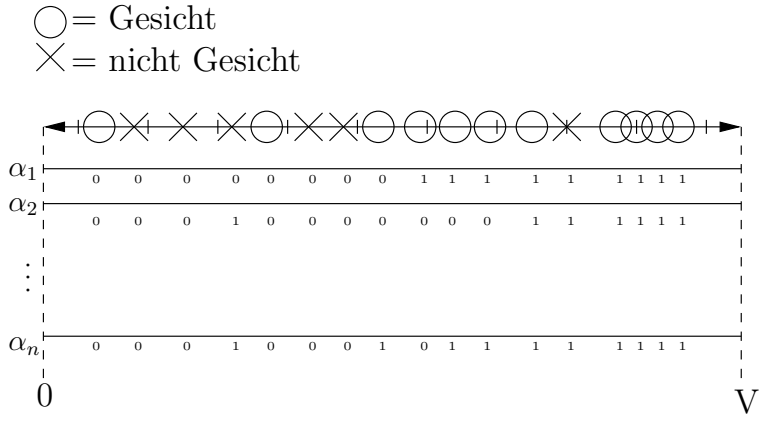


Abbildung 3.9: Der Zahlenstrahl ist sortiert anhand der aufsummierten α-Werte der schwachen Klassifikatoren. Je mehr schwache Klassifikatoren ein Bild als Gesicht klassifizieren, je größer ist der Wert des Bildes.

Hat man beispielsweise 1000 Gesichter in der gegebenen Validierungsmenge, so müssen bei einer Detektionsrate von 98,5% 985 Gesichter erkannt werden, dafür sind dann 15 Gesichter zu überspringen. Der dann gefundene Schwellenwert stellt sicher, dass 98,5 % der Gesichter noch auf dem Zahlenstrahl oberhalb des Schwellenwertes liegen. Dabei kann die Falsch-Positiv-Rate natürlich beliebig hoch sein, vor allem wenn von AdaBoost erst wenige schwache Klassifikatoren trainiert wurden. Daher werden solange schwache Klassifikatoren hinzugefügt, bis der starke Klassifikator die geforderte Falsch-Positiv-Rate erfüllt. Der vollständig angepasste AdaBoost Algorithmus ist unter Algorithmus 5 auf Seite 36 zu finden.

3.6.2 Bestimmung der Kaskadenklassifikationsleistung

Mittels des angepassten AdaBoost-Algorithmus stehen nun alle Mittel zur Verfügung, um eine Kaskade mit ausreichender Genauigkeit zu trainieren. Möchte man beispielsweise eine Kaskade mit einer gesamten Falsch-Positiv-Rate von $F = 1 \cdot 10^{-6}$ (also maximal eine Fehldetektion auf einem ein Megapixel großen Bild), so lässt sich schon vorweg sagen, wie viele Kaskadenstufen für diese angepeilte Falsch-Positiv-Rate zu erwarten sind: Die Falsch-Positiv-Raten, sowie die Detektionsraten der einzelnen Stufen mul-

Algorithmus 4
Bestimmung des Schwellenwertes des starken Klassifikators

Eingabe:
$(x_1, y_1), \ldots, (x_n, y_n)$, x_i: Testbilder, y_i: Bildklasse
Detektionsrate D die der starke Klassifikator erreichen muss.

1: Berechne Anzahl an Gesichtern die bei der gegebenen Detektionsrate zu überspringen sind. $f2cc = D \cdot \#\text{Gesichter}$
2: **for** $i = 1, \ldots, n$ **do**
3: Berechne Entscheidungswert:

$$\varphi_i = \sum_{t=1}^{T} \alpha_t h_t(x_i)$$

4: Speichere ursprünglichen Index von i in $o(i)$
5: **end for**
6: Sortiere Menge der Entscheidungswerte $\varphi_1, \ldots, \varphi_n$ als $\hat{\varphi}_1, \ldots, \hat{\varphi}_n$
7: Initialisiere Zählvariable für Gesichter $f = 0$
8: Initialisiere Indexvariable $i = -1$
{Durchlaufe nun die sortierte Menge bis $f2cc$ Gesichter gefunden wurden und speichere diesen Index}
9: **while** f < #Gesichter - f2cc **do**
10: $i = i + 1$
11: **if** Bild $x_{o(i)}$ ein Gesicht **then**
12: $f = f + 1$
13: **end if**
14: **end while**
{ Laufe ausgehend vom gefundenen Index zurück und nehme als Schwellenwert den ersten möglichen Wert}
15: **while** $i \geq 0 \wedge \phi_i = \phi_{i+1}$ **do**
16: $i = i - 1$
17: **end while**
18: **if** $i \geq 0$ **then**
19: $\vartheta = \frac{\phi_i + \phi_{i+1}}{2}$)
20: **else**
21: $\vartheta = \phi_0 - 1$ {Wir haben keinen Schwellenwert gefunden, also nehmen wir den kleinsten Wert. }
22: **end if**

tiplizieren sich. Sei f_i die Falsch-Positiv Rate der i-ten Stufe und d_i die Detektionsrate der i-ten Stufe, so ergibt sich die gesamte Falsch-Positiv-Rate zu

$$F = \prod_{i=1}^{n} f_i$$

sowie die endgültige Detektionsrate der Kaskade zu

$$D = \prod_{i=1}^{n} d_i.$$

Unsere als Ziel angepeilte Falsch-Positiv-Rate der Kaskade von $F = 1 \cdot 10^{-6}$ haben wir also bei einer Falsch-Positiv-Rate von 50 % pro Stufe nach

$$\frac{\log(1 \cdot 10^{-6})}{\log 0.5} \approx 19.9316 \approx 20$$

Stufen erreicht. Allgemein berechnet man die Anzahl n an benötigten Kaskadenstufen bei einer gegebenen Falsch-Positiv-Rate pro Stufe f und einer für die gesamte Kaskade angepeilten Falsch-Positiv-Rate F also mittels der Formel

$$\frac{\log F}{\log f} = n.$$

Die Anzahl n der benötigten Stufen lässt sich analog auch über die Detektionsrate d einer Stufe, sowie der Gesamt Detektionsrate D der Kaskade bestimmen:

$$\frac{\log D}{\log d} = n.$$

Bei 20 Stufen beträgt die Detektionsrate der gesamten Kaskade bei 98,5 % pro Stufe dann

$$(0.985)^{20} \approx 0.7399 \approx 74\%.$$

Algorithmus 5
Der für die Kaskade angepasste Lernalgorithmus AdaBoost in Pseudocode

Eingabe:

$(x_1, y_1), \ldots, (x_n, y_n)$ x_i: Trainingsbilder, y_i: Bildklasse
F_{Ziel}: Falsch-Positiv-Rate
D_{Ziel}: Detektionsrate des zu trainierenden starken Klassifikators.

Initialisierung:

Gewichte:

$$w_{1,i} = \begin{cases} \frac{1}{2l} & \text{falls } y_i = 1 \text{ (Gesicht)} \\ \frac{1}{2m} & \text{falls } y_i = 0 \text{ (Nichtgesicht)}, \end{cases}$$

wobei m und l für die Anzahl der Nichtgesichter bzw. Gesichter steht.

1: **while** $F_{\text{aktuell}} > F_{\text{Ziel}}$ **do**
2: t = t + 1
3: Normalisierung der Gewichte: $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$.
4: **for** jedes mögliche Feature j **do**
5: Trainiere einen schwachen Klassifikator h_j zu dem Feature j (vgl. Algorithmus 3 auf Seite 28).
6: Berechne den gewichteten Fehler ϵ_j für h_j: $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$.
7: **end for**
8: Wähle den Klassifikator h_t mit dem kleinsten gewichteten Fehler ϵ_t.
9: Berechne Güte des gefundenen schwachen Klassifikators: $\alpha_t = \log(\frac{1-\epsilon_t}{\epsilon_t})$
10: Aktualisiere den Gewichtsvektor $\vec{w}$:

$$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i} \text{ mit } \beta_t = \frac{\epsilon_t}{1-\epsilon_t} \quad , \quad e_i = \begin{cases} 1 & \text{falsch klassifiziert} \\ 0 & \text{sonst} \end{cases}$$

11: Berechne anhand der Detektionsrate D_{Ziel} den Schwellenwert ϑ des starken Klassifikators (vgl. Algorithmus 4 auf Seite 34).
12: Berechne die Falsch-Positiv-Rate $F_{\text{aktuell}} = \frac{\#Detektionen}{\#Gesichter}$ des bisher trainierten Klassifikators.
13: **end while**

Der starke Klassifikator bestehend aus t schwachen Klassifikatoren ist dann:

$$h(x) = \begin{cases} 1 & \underbrace{\sum_{t=1}^{T} \alpha_t h_t(x)}_{\text{Entscheidungswert}} \geq \underbrace{\vartheta}_{\text{Schwellenwert}} \\ 0 & \text{sonst} \end{cases}$$

mit $\alpha_t = \log \frac{1}{\beta_t}$.

Algorithmus 6
Training der Kaskade in Pseudocode

Eingabe:
$(x_1, y_1), \ldots, (x_n, y_n)$ x_i: Trainingsbilder y_i: Bildklasse
F_{Ziel}: Falsch-Positiv-Rate der zu trainierenden Kaskade.
F_{Stufe}: Falsch-Positiv-Rate einer einzelnen Kaskadenstufe.
D_{Stufe}: Detektionsrate einer einzelnen Kaskadenstufe.

1: **while** $F_{\text{aktuell}} > F_{\text{Ziel}}$ **do**
2: Berechne starken Klassifikator über (vgl. Algorithmus 5 auf Seite 36): $SC_t = \text{AdaBoost}((x_1, y_1), \ldots, (x_n, y_n), F_{\text{Stufe}}, D_{\text{Stufe}})$
3: Berechne $F_{\text{aktuell}} = \prod_{i=1}^{t} f_i$, wobei f_i die Falsch-Positiv-Rate der i-ten Kaskadenstufe ist.
4: Generiere neue Negativebeispiele für das Training der nächsten Kaskadenstufe.
5: **end while**

3.7 Dynamische Generierung von Negativbeispielen

Die Kaskade wird sukzessive mit jeder trainierten Kaskadenstufe genauer im Sinne der erreichten Falsch-Positiv-Rate. Haben wir beispielsweise eine Falsch-Positiv-Rate von 40% als Lernziel für eine einzelne Kaskadenstufe vorgegeben, so hat eine Kaskade, die erst eine Stufe trainiert hat, auch nur eine Falsch-Positiv-Rate von 40%. 40% Falsch-Positiv-Rate bedeutet aber auch, dass 60% der Negativbeispiele, die der Kaskade zum Trainieren gegeben werden, korrekt ausgefiltert werden. Es macht jetzt also keinen Sinn, diese ausgefilterten Negativbeispiele für das Training der zweiten Stufe vorzulegen, da so ja nur redundante Informationen gelernt werden würde, die die erste Stufe der Kaskade schon gelernt hat. Also müssen wir diese 60% an Negativbeispielen durch neue Negativbeispiele, die der Klassifikator noch nicht gesehen hat, austauschen. Für folgende Beispielrechnung gehen wir der Einfachheit halber davon aus, dass die Kaskade 1000 Gesichter sowie 1000 Nichtgesichter bekommt. Dann müssen wir also 60% der Nichtgesichter, also 600 Nichtgesichter, austauschen. Die vielleicht naheliegende Idee, hierfür Rauschbilder oder zufällig ausgewählte Bilder zu nehmen, löst das Problem nicht hinreichend. Denn diese zufällig ausgewählten Bilder können ja auch

wieder zu der Art von Bildern gehören, die die erste Stufe der Kaskade schon gut als Nichtgesicht erkennen kann. Stattdessen werden große Bilder, die größer als ein Megapixel in der Auflösung sind, verwendet. Alle diese Bilder wurden manuell gesichtet um sicherzustellen das sie kein Gesicht abbilden. Jedes dieser Bilder wird dann vom bisher trainierten Klassifikator nach Gesichtern durchsucht. Jedes Subfenster das vom Klassifikator fälschlicherweise als Gesicht klassifiziert wurde, wird in die neue Trainingsmenge aufgenommen. Nachdem für ein Bild alle Subfenster so auf Fehldetektionen untersucht wurden, wird das Bild runterskaliert und erneut durchsucht. Diese Skalierung hat zwei Gründe, zum einen erhält man so pro Quellbild mehr für das Training verwendbare Fehldetektionen, zum anderen soll unser Klassifikator später auch Gesichter beliebiger Größe erkennen und wir erwarten durch das Skalieren eine bessere Trainingsmenge für den Klassifikator. Nachdem zweiten Durchlauf des Klassifikators über das runterskalierte Bild wird weiter runterskaliert, solange wie die Auflösung des Quellbildes noch größer als die Auflösung der Trainingsbilder ist. Erst dann wird das nächste große Quellbild für die Suche nach Fehldetektionen verwendet. So ist sichergestellt, dass die neue Trainingsmenge auch einen ausreichenden neuen Informationsgehalt hat, um eine neue Stufe zu trainieren.
Nun ist es nach der ersten Kaskadenstufe noch relativ einfach, die 600 Negativbeispiele aus unserer Beispielrechnung zu generieren, da der Klassifikator etwa 40% aller ihm vorgestellten Nichtgesichter fälschlicherweise als Gesicht klassifizieren wird. Schwieriger wird es aber bei den folgenden Stufen, denn an den 600 zu generierenden Bildern ändert sich nichts, da wir ja laut der Kaskadenstruktur jeweils die gleiche Falsch-Positiv-Rate vorgeben. Nur wird die gesamte Falsch-Positiv-Rate der Kaskade multiplikativ besser mit jeder gelernten Stufe (vergleiche Abschnitt 3.6.2 auf Seite 33). Also werden beim Generieren von Negativbeispielen für den dritten AdaBoost Trainingslauf nur noch etwa 16% (0.4^2) aller präsentierten Negativbeispiele fälschlicherweise als Gesicht klassifiziert.
Somit wird es nach jeder trainierten Stufe, mit der die Kaskade besser geworden ist, deutlich aufwendiger, die Menge an Negativbeispielen sinnvoll aufzufüllen, so dass für den AdaBoost-Klassifikator noch etwas zu lernen ist. Um eine Größenvorstellung zu bekommen, wieviele Bilder man für die

Erzeugung von Negativbeispielen benötigt, sei folgende Beispielrechnung gegeben. Bei einer anvisierten Falsch-Positiv-Rate der gesamten Kaskade von $1 \cdot 10^{-6}$ werden bei einer für die Kaskadenstufen vorgegebenen Falsch-Positiv-Rate von 40% 14 Stufen benötigt. Die Anzahl benötigter Quellbilder mit einer Auflösung von einem Megapixel lässt sich nun wie folgt überschlagen:

$$\#\text{Bilder} = \left\lceil \frac{\sum_{i=1}^{n} \frac{\#\text{Benötigte\_neg\_Beispiele}}{f^i}}{\#\text{Generierbare\_Teilfenster\_aus \_Quellbild}} \right\rceil .$$

Für unser konkretes Beispiel ergibt sich also :

$$\left\lceil \frac{\sum_{i=1}^{14} \frac{600}{0.4^i}}{(1000-19)\cdot(1000-19)} \right\rceil \approx 155.$$

Anhand dieser Überlegung sieht man noch einen weiteren Grund, weshalb man die gesamt Falsch-Positiv-Rate der Kaskade nicht beliebig klein machen kann. Würde man entsprechend viele negativ Beispiele noch mit einem gewissen Aufwand generiert bekommen, so reicht je nach Einsatzgebiet der Kamera auch eine bestimmte Falsch-Positiv-Rate aus. Eine Kaskade die zur Echtzeitklassifikation auf einer Kamera mit weniger als einem Megapixel trainiert wird, kann mit einer Fehldetektion auf einem Megapixel (was eine Falsch-Positiv-Rate von $1 \cdot 10^{-6}$ bedeutet) arbeiten. Haben wir zusätzlich noch eine Vorabinformation wie viele Gesichter auf dem späteren Bild zu erwarten sind, lässt sich die Anzahl der Fehldetektionen noch weiter reduzieren. Denn wir haben ja zusätzlich zu einer Detektion auch noch eine Information über die Güte der Detektion (die gewichtete Summe der Alpha Werte der Klassifikatoren). Mittels dieser haben wir die Möglichkeit, beispielsweise nur die besten drei Detektionen zu verwenden.

3.8 Skalierung des Klassifikators

Da der Klassifikator natürlich nicht nur Gesichter in der Größe der zum Training verwendeten Bilder erkennen können soll, muss der Klassifikator auf verschieden große Subfenster angewendet werden. Die einfachste Möglichkeit wäre wohl, das Eingabebild auf die Auflösung der beim Training ver-

wendeten Bilder zu skalieren. Dies hätte den Vorteil, dass keine weiteren Veränderungen an dem Klassifikator vorgenommen werden müssten. Hierfür würden aber viele verschiedene Skalierungen des zu untersuchenden Bildes benötigt. Zu jedem dieser Skalierungen müsste dann aber auch wieder das Integralbild des skalierten Bildes berechnet werden. Die Berechnung dieser Bildpyramide würde unser Verfahren stark verlangsamen, da eine Kernidee des Ansatzes von Viola und Jones die einmalige Vorverarbeitung der zu untersuchenden Bilder mittels der Integralbilder. Zusätzlich würde man ein weiteres Performanceproblem bekommen. Verfahren zur Skalierung die Interpolation nutzen sind für die angepeilte Echtzeitfähigkeit des Klassifikators zu aufwendig. Daher bleibt nur die Verwendung eines einfachen Verfahren, wie z.B. das Löschen jeder zweiten Zeile/Spalte. Ein solches einfaches Verfahren geht aber mit einem relativ hohen Informationsverlust der Bilder einher.

Besser ist es die von den schwachen Klassifikatoren verwendeten Features zu skalieren. Die trainierten Features lassen sich natürlich auch skaliert verwenden. Hierfür wird zunächst festgelegt, welche Größe das neue skalierte Subfenster haben soll auf dem der Klassifikator arbeiten soll. An den verwendeten Features müssen nun zwei Änderungen vorgenommen werden. Erstens müssen die Features passend zur Größe des ausgewählten Subfensters skaliert werden. Zweitens müssen wir den Ursprung des Features an die Größe des skalierten Subfensters anpassen (siehe Abbildung 3.10 auf Seite 41). Bei der Skalierung der Breite sowie Höhe des Features ist natürlich zu bedenken, dass die jeweiligen Featuretypen jeweils nur bestimmte Breiten und Höhen zulässig sind. Beim Featuretyp A (drei vertikale Balken, vgl. Abbildung 3.1 auf Seite 13) muss beispielsweise die Breite durch drei teilbar sein. Hier muss also beim Skalieren so gerundet werden, dass die Voraussetzungen zur Berechnung der Features nach dem Skalieren erfüllt sind. Die Translation des Features ist weniger problematisch, wobei natürlich zu beachten ist, dass das translatierte und skalierte Feature in die Grenzen des Subfensters passt.

Zu bedenken ist jetzt lediglich noch, wie man mit dem zum Feature gehörenden Schwellenwert ϑ umgeht. Mit dem Skalieren der Feature ändert sich auch der Wertebereich des Features. Somit müssten wir den Schwellenwert

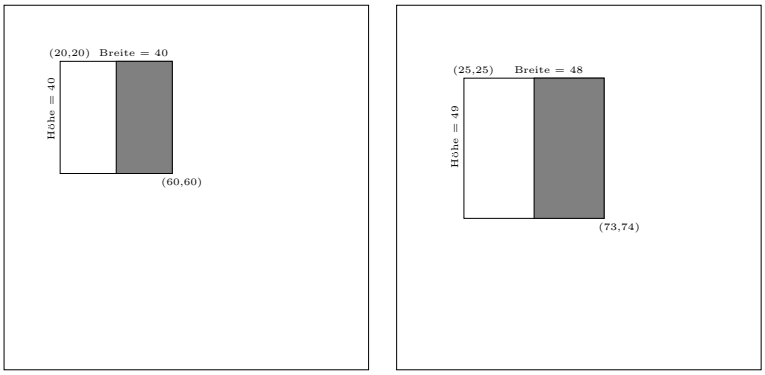


Abbildung 3.10: Beispiel für die Skalierung eines Features. Ein zweibalkiges Feature wird hier mit dem Skalierungsfaktor 1,225 skaliert. Da die Breite dieses Features durch zwei teilbar bleiben muss, wird die Breite des Features von 49 auf 48 abgerundet.

entsprechend der Skalierung des Features mittels des Skalierungsfaktors $\frac{\hat{w}\hat{h}}{wh}$ anpassen:

$$h_w(x) = \begin{cases} 1 & \sigma\hat{f}(x) \geq \sigma\vartheta\frac{\hat{w}\hat{h}}{wh} \\ 0 & \text{sonst} \end{cases}.$$

$\hat{f}(x)$ steht hier für das skalierte Feature und entsprechend $\hat{w}$ und $\hat{h}$ für die Breite und Höhe des skalierten Subfensters. Für die Implementation ist diese Variante aber nicht optimal. Daher verwenden wir folgende äquivalente Darstellung bei der wir die Feature-Werte immer über die Fläche normalisieren. Dann ergibt sich folgender Zusammenhang:

$$h_w(x) = \begin{cases} 1 & \sigma\frac{\hat{f}(x)}{\hat{w}\hat{h}} \geq \frac{\sigma\vartheta}{wh} \\ 0 & \text{sonst} \end{cases}.$$

Den Schwellenwert ϑ müssen wir nicht explizit durch die Fläche $w \cdot h$ teilen, da wir durch die Normalisierung der Feature-Werte auch während des Trainings entsprechend normalisierte Schwellenwerte lernen.

3.9 Abtastung eines Eingabebildes

Um Gesichter in einem Bild zu finden, muss die trainierte Kaskade alle Subfenster des Bildes untersuchen. Ist dieser Suchlauf beendet wird die Kaskade skaliert, es werden jetzt also größere Subfenster untersucht. Dieses wird jetzt solange wiederholt wie ein Subfenster noch kleiner als das zu untersuchende Quellbild ist.
Die Kaskade wurde auf Bildern mit Gesichtern fester Größe trainiert. Um ein Eingabebild auf Gesichter beliebiger Größe zu untersuchen, wird die Kaskade skaliert, das heißt die verwendeten Features der schwachen Klassifikatoren werden um einen Faktor s skaliert (vgl. Abbildung 3.10 auf Seite 41). Als Skalierungsfaktor s wird initial 1,25 gewählt und dieser in Schritten von 0,25 gesteigert, solange wie die Größe der skalierten Kaskade noch kleiner ist als die Größe des Eingabebildes.
Zusätzlich möchte man natürlich Gesichter an beliebigen Positionen im Eingabebild erkennen. Hierfür wird die skalierte Kaskade an Subfenstern des Eingabebildes ausgewertet. Mittels einer Schrittweite Δ werden jetzt alle gültigen Subfenster für ein Eingabebild durchiteriert. Die Schrittweite Δ hat starken Einfluss auf die endgültige Laufzeit des Klassifikators, da sie ja festlegt, an wie vielen Subfenstern des Eingabebildes die skalierte Kaskade klassifiziert. Daher sollte die Schrittweite Δ nicht nur einen Pixel betragen. Außerdem macht es Sinn, die Schrittweite Δ in Verhältnis zum Skalierungsfaktor s zu setzen, da man so in Relation zur Größe des gesuchten Gesichts scannt. In dieser Arbeit wurde dem Beispiel von Viola und Jones gefolgt und ein Δ von 3 gewählt.
Das Eingabebild wird also für jede Skalierung mit einer Schrittweite von $s \cdot \Delta$ in x- und y-Richtung durchiteriert.

3.10 Filterung überlappender Detektionen

Bei der Abtastung eines Eingabebildes entstehen üblicherweise Mehrfachdetektionen ein und desselben Gesichtes (siehe Abbildung 3.11 auf Seite 43).

Der Grund hierfür ist, dass sich die Werte der vom Detektor verwendeten

Abbildung 3.11: Ausgabe des Detektors: Im linken Bild ohne die Filterung von überlappenden Detektionen sowie im rechten Bild die Ausgabe des Detektors mit der Nachbearbeitung, die überlappende Detektionen zu einer zusammenfasst.

Features nur wenig ändern, wenn sie beispielsweise auf einem Subfenster berechnet werden, das nur um einen Pixel verschoben ist. Wir möchten nun aber nur eine Detektion pro Gesicht haben. Daher filtern wir alle Detektionen, die das gleiche Gesicht markieren, nach dem Lauf des Detektors noch einmal.
Zu allen Detektionen haben wir die Güte der Detektion in Form des Entscheidungswertes, also ist der α-Wert einer Detektion d:

$$\alpha_d = \sum_{s=1}^{S} \sum_{t=1}^{T_s} \alpha_{s,t} h_{s,t}(x),$$

wobei S die Anzahl der starken Klassifikatoren ist und T_s die Anzahl der schwachen Klassifikatoren des s-ten starken Klassifikators. Anhand sortieren wir alle unsere gefundenen Detektionen eines Bildes anhand ihrer Relevanz. Denn ein größerer α_d-Wert einer Detektion bedeutet, dass mehr schwache Klassifikatoren diese Detektion als Gesicht klassifiziert haben als eine Detektion, die über einen kleineren α_d-Wert verfügt. Nun gehen wir alle Detektionen durch, angefangen mit der Detektion die den höchsten α_d-Wert hat. Auf einem Hilfsbild, das die gleiche Größe hat wie das Eingabebild für den Detektor, markieren wir nun alle Pixel, die diese relevanteste Detektion abdeckt. Bei den darauf folgenden Detektionen überprüfen wir zunächst über unser Hilfsbild, ob es ein Pixel unser zu verarbeitenden Detektion gibt, das schon auf dem Bild markiert ist. Ist dem so, so ist unsere aktuelle Detek-

tion irrelevant, da sie eine überlappende Detektion eines schon markierten Gesichtes ist. Auf diese Art überprüfen wir alle weiteren Detektionen, bis wir nur noch nicht überlappende Detektionen über behalten. Der zugrundeliegende Algorithmus 7 ist auf Seite 44 zu finden.

Algorithmus 7
Filterung überlappender Detektionen

Eingabe:
Menge Detektionen der Kaskade mit Gewichtung: $((d_1, \alpha_1) \dots (d_n, \alpha_n))$
w: Breite des Eingabebildes
h: Höhe des Eingabebildes

Initialisierung:
Temporäres Bild t mit Nullen vorinitialisieren.

1: Sortiere die Detektionen $((d_1, \alpha_1) \dots (d_n, \alpha_n))$ absteigend anhand α_i-Wertes.
2: **for** $i = 1, \dots, n$ **do**
3: **if** $\{\forall$ Pixel $j \in d_i \mid t(j) = 0\}$ **then**
4: $\forall$ Pixel $j \in d_i$: $t(j) = 1$
5: Übernehme i-te Detektion in die Liste der endgültigen Detektionen.
6: **end if**
7: **end for**

3.11 Klassifikation auf Bilddaten mit mehreren Kanälen

Der Viola und Jones Klassifikator soll in dieser Arbeit implementiert auf Bilder einer Time-Of-Flight-Kamera angewendet werde. Die Bilder dieser Kamera liefern zeitgleich zwei Kanäle: Einen Helligkeitskanal und einen Tiefenkanal (siehe Kapitel 2 zur Time-Of-Flight-Kamera). Wie passen wir das Training des Klassifikators an, um auf diesen Bildern mit mehreren Kanälen zu rechnen? Wir erweitern die Suche nach dem besten schwachen Klassifikator auf beide Bildkanäle. Für die Klassifikation auf reinen Grauwertbildern haben wir für die Suche nach dem besten schwachen Klassifikator den gewichteten Fehler für alle möglichen Features berechnet (vgl. Abschnitt 3.5 ab Seite 25). Gewählt wurde dann der schwache Klassifikator, der den kleinsten gewichteten Fehler hatte. Bei Verwendung von mehrkanaligen Bildern

erweitert sich nun die Menge potentieller schwacher Klassifikatoren. Haben wir vorher für jeden Feature-Typ jede Position auf den Trainingsbildern bei jeder Größe des Subfensters ausgewertet, so erweitern wir diese Suche einfach auf jeden Kanal der Trainingsbilder. In unsere Fall haben wir also jeweils doppelt so viele potentielle schwache Klassifikatoren im Vergleich zu einem Training auf reinen Helligkeitsbildern.
Bei der späteren Nutzung des trainierten Klassifikators ändert sich wenig. Es muss lediglich bei der Auswertung der einzelnen schwachen Klassifikatoren der passende Kanal des Bildes zu dem trainierten Klassifikator verwendet werden. Die weitere Verrechnung der schwachen Klassifikatoren zu einem starken Klassifikator bleibt exakt gleich zu dem Fall der einkanaligen Bilder.

4 Implementation und Ergebnisse

4.1 Implementation

4.1.1 Softwarearchitektur

Der Klassifikator wurde in C++ implementiert. Diese Implementationssprache wurde schon zu Beginn der Arbeit gewählt, da eine dieser Arbeit vorausgehende Arbeit „Gesichtsdetektion mit dem Verfahren von Viola und Jones“ [Lec07] gezeigt hatte, dass eine echtzeitfähige Implementation des Verfahrens von Viola und Jones nicht mit der in dieser Arbeit eingesetzten Sprache MATLAB zu erreichen ist. Dabei wurden in der Arbeit besonders zeitkritische Bereiche in C implementiert (eingebunden als sogenannte MEX-Dateien).
Die Implementation baut auf einer am Institut für Neuro- und Bioinformatik entwickelten Bibliothek auf, die Funktionen zur Verarbeitung von Bildern bereitstellt und selber auf der Bibliothek OpenCV [OI] von Intel aufbaut. Kompiliert wurde mittels des GNU project C and C++ compiler (gcc) in der Version 4.1.3 auf Ubuntu Linux 7.10 mit 2.6.22-15-generic Kernel.

4.1.2 Hardwarearchitektur

Der verwendete PC ist ein Intel Core2 Duo E6750 mit 2,66 GHz pro Rechenkern und 4 GB Arbeitsspeicher. Die verwendete Time-Of-Flight-Kamera ist eine SR3000 von der Firma MESA [Mes08]. Die Leistungsdaten der Kamera sind in Tabelle 4.1 auf Seite 48 zusammengefasst.

Abbildung 4.1: Verwendete 3D-Time-Of-Flight Kamera: MESA SR3000. Technische Details siehe Tabelle 4.1 auf Seite 48.

Pixel array Auflösung	176 × 144 Pixel
effektives Messfeld	47.5 x 39.6 Grad
Eindeutigkeitsbereich	7.5 m
Modulationsfrequenz	20 MHz
Anschluss	USB 2.0
optische Linse	f/1.4
optische Beleuchtungsstärke	1 W bei 850 nm
Maße Kameragehäuse	50 × 67 × 42.3 mm (Aluminium)
Stromversorgung	12 V, 1A
Betriebstemperatur	-10°C to +50°C
Ausgabe Daten	i (Intensität) , r (Entfernung)

Tabelle 4.1: Leistungsdaten der SR3000.

4.2 Verwendete Trainingsdaten

Die Entwicklung des Klassifikators wurde in zwei Schritten durchgeführt. Zunächst wurde das Training für eine Kaskade implementiert, die auf reinen Graustufenbildern lernt. Motivation hierfür waren im wesentlichen drei Punkte:

1. Das Training eines Klassifikators auf Graustufenbildern entspricht exakt dem zugrunde liegenden Viola-Jones-Verfahren [VJ04]. Somit war es gut möglich, Teilergebnisse während der Entwicklung mit den Er-

fahrungen aus dem Viola/Jones Paper zu vergleichen.

2. Für das Training auf Graustufenbildern steht ein Trainingsdatensatz vom „Center for Biological and Computational Learning at MIT and MIT“ [MIT00] bereit. Dieser Datensatz wird von vielen Gruppen zum Training und Test von Gesichtsdetektion verwendet.

3. Dieser Arbeit ist eine Arbeit von M. Lechner [Lec07] vorausgegangen, welche ebenfalls gute Ergebnisse auf dem CBCL-Datensatz des MIT erzielen konnte.

Als die Ergebnisse der Neuimplementation des Detektors auf Graustufenbildern gute Ergebnisse lieferte, wurde der zweite Schritt vollzogen und der Detektor für die Verwendung von Time-Of-Flight-Bildern angepasst (vgl. Abschnitt 3.11 ab Seite 44).

4.2.1 Graustufenbilder

Der verwendete Trainingsdatensatz stammt vom „Center for Biological and Computational Learning at MIT and MIT“ [MIT00]. Dieser beinhaltet 2,431 Gesichter sowie 4,548 Nichtgesichter als 19×19 Graustufen Bilder im PGM Format. Um die Trainingsmenge noch weiter zu vergrößern, wurden alle Bilder horizontal gespiegelt, so dass letztendlich 4862 Gesichter und 9096 Nichtgesichter für das Training der Kaskade bereitstanden. Zusätzlich wurde die Anzahl der Nichtgesichter weiter erhöht, indem 9096 Rauschbilder generiert wurden. Diese Rauschbilder wurden als Nichtgesichter gelabelt. Anschließend wurden alle Bilder gleichmäßig auf die Trainings- und Validierungsmenge aufgeteilt. Die genaue Aufteilung der Bilder des Trainingsdatensatz auf die vom Trainingsalgorithmus verwendeten Bildvektoren ist in Tabelle 4.2 auf Seite 51 zu sehen. Für das Training müssen jetzt noch die Detektionsrate, also die prozentuale Rate an Gesichtern, die eine Stufe der Kaskade detektieren muss, sowie die Falsch-Positiv-Rate die angibt, wie viele Fehldetektion eine Stufe tolerieren kann, festgelegt werden.

Um diese Raten festzulegen, muss der kaskadenartige Aufbau des Detektors beachtet werden (vgl. Kapitel 3.6.2 ab Seite 33). Das bedeutet: Jede einzelne Stufe der Kaskade muss eine sehr hohe Detektionsrate von nahezu

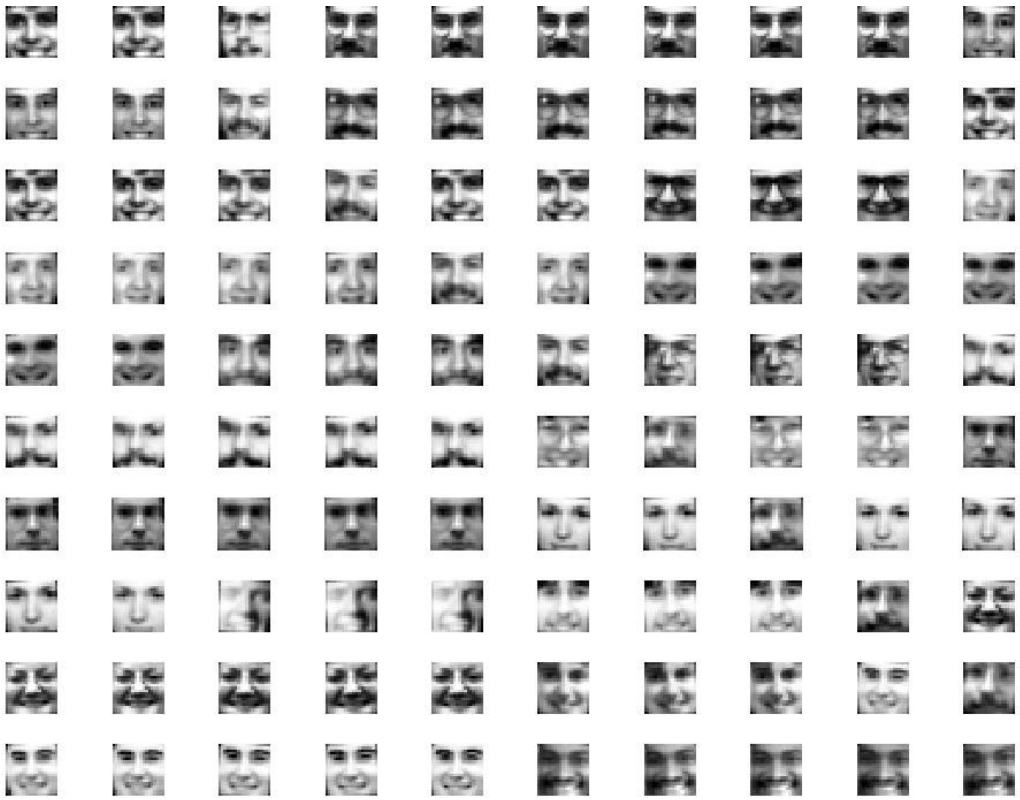


Abbildung 4.2: Beispiel Bilder aus dem MIT-Trainingsdatensatz. Hierbei ist gut zu sehen, dass die Größe der Bilddatenbank durch Rotation sowie horizontale Spiegelung der Eingabebilder erreicht wurde. .

100% aufweisen, da einmalig als Nicht-Gesicht aussortierte Subfenster nicht mehr bearbeitet werden. Daher legen wir die zu erreichende Detektionsrate einer einzelnen Kaskadenstufe auf 98% fest. Weiterhin möchten wir, dass die komplette Kaskade noch eine Detektionsrate von mindestens 75% hat. Also darf unsere Kaskade aus

$$\frac{\log 0.75}{\log 0.98} \approx 14$$

Stufen bestehen (vgl. Kapitel 3.6.2 ab Seite 33). Die Falsch-Positiv-Rate einer einzelnen Kaskadenstufe legen wir auf 45% fest. Diese Raten wurden durch Experimente am Institut für Neuro- und Bioinformatik bestimmt.

Zwar würden wir bei dieser Falsch-Positiv-Rate von 45% pro Kaskadenstufe im schlechtesten Fall nur auf eine Gesamt-Falsch-Positiv-Rate von $1.4 \cdot 10^{-5}$ kommen, aber davon ist nicht auszugehen, da gerade die ersten Stufen der Kaskade ein vergleichsweise leichtes Problem zu lösen haben und wir daher erwarten besser als dieser theoretische Wert zu werden. Zusammengefasst geben wir also für das Training auf den MIT Datensatz 98% Detektionsrate pro Stufe vor und erlauben bei dieser Detektionsrate 45% Falsch-Positive Detektionen.

	Trainingsmenge	Validierungsmenge
	Grauwert-bilder	Grauwert-bilder
Gesichter	2431	2431
Nichtgesichter	9096	9096
Gesamt	11527	11527

Tabelle 4.2: Verteilung der Trainingsdaten für das Training des Detektors auf dem MIT-Datensatz [MIT00].

4.2.2 Time-Of-Flight-Bilder

Das eigentliche Ziel dieser Arbeit ist das Training eines Gesichtsdetektors auf Bildern der Time-Of-Flight-Kamera. Hierfür wurden am Institut für Neuro- und Bioinformatik 17 Personen mit der Time-Of-Flight-Kamera aufgenommen. Es wurden jeweils kleine Videosequenzen der einzelnen Personen aufgenommen, bei der die Person sich zur Kamera hin und von Ihr weg bewegten, um für Varianz in den Bildern zu sorgen. Im Schnitt entstanden so 77 verwertbare Bilder pro gefilmter Person (vgl. Tabelle 4.3 auf Seite 51).

Person	1	2	3	4	5	6	7	8	9
Bilder	65	62	56	52	47	36	55	119	96

Person	10	11	12	13	14	15	16	17	∅
Bilder	84	88	77	118	70	95	91	101	ca. 77

Tabelle 4.3: Struktur des Time-Of-Flight-Trainingsdatensatz.

Diese ungefähr 1300 Bilder lagen in der nativen Auflösung von 176×144 Pixeln der Time-Of-Flight-Kamera vor. Um später die Detektionsleistung der trainierten Kaskade bestimmen zu können, wurde von diesen 1300 Bildern eine zufällige Auswahl von 200 Bildern als Testmenge abgetrennt. Würde man beispielsweise die ersten 200 aus der Menge nehmen, die ja zeitlich sortiert sind, würde man dem Klassifikator im Test nur die Person 1,2 und 3 zeigen. Es ist aber nicht auszuschließen das gerade die Personen besonders gute, oder auch besonders schlechte Merkmale für den Klassifikator haben.
Für das Training des Klassifikators wurde auf jedem Bild der Kopf manuell markiert, ausgeschnitten und auf eine Größe von 24 × 24 gebracht.

	Trainingsmenge	Validierungsmenge
	Time-Of-Flight-bilder	Time-Of-Flight-bilder
Gesichter	3330	3330
Nichtgesichter	4000	4000
Gesamt	7330	7330

Tabelle 4.4: Verteilung der Trainingsdaten für das Training des Detektors auf dem Time-Of-Flight Datensatz.

Um aus den 176×144 Pixel großen Bildern die 24×24 großen Bilder zu generieren, wurden die Bilder in einem ersten Schritt von Hand gelabelt. Bei dieser Labelung wurde darauf geachtet, dass die gefilmten Gesichter großzügiger als bei dem MIT-Datensatz markiert wurden. Dies wurde gemacht da Viola und Jones [VJ04] festgestellt hatten, dass sie bessere Ergebnisse mit großzügigeren Labelungen erreichen. Entsprechend diesen Labelungen wurden dann schließlich quadratische 24×24 Pixel große Trainingsbilder ausgeschnitten. Nach diesem Schritt standen dann also 1100 Gesichter für das Training zur Verfügung. Dies sind wesentlich weniger Trainingsbeispiele sind als im MIT-Datensatz. Während des Trainings der MIT-Kaskade zeigte sich, dass eine größere Trainingsmenge zu einer besseren Generalisierungsleistung des Klassifikators führte. Die große Anzahl der Trainingsbilder im MIT-Datensatz wurde erreicht, indem die Gesichter der Personen jeweils um wenige Grad rotiert wurden und horizontal gespiegelt (vgl. hierzu Ab-

bildung 4.2 auf Seite 50) wurden. Aufgrund der guten Ergebnisse, die mit den rotierten Bildern erreicht wurden (vgl. Abschnitt 4.3.1 ab Seite 54), wurden dann auch die 1100 quadratisch ausgeschnittenen Trainingsbilder noch jeweils noch um 5° nach links und rechts rotiert. So standen letztendlich ca. 3300 gelabelte Gesichter für das Training einer Kaskade mit Tiefenfeatures zur Verfügung. Um Bilder für die Generierung der Nichtgesichter zu bekommen, wurden ebenfalls Videosequenzen mit der Time-Of-Flight-Kamera aufgenommen. Bei diesen Videos wurden Personen gefilmt, deren Gesichter verdeckt waren, zusätzlich haben diese Personen irgendwelche Bewegungen ausgeführt. Die Idee für solche Aufnahmen war, möglichst schwierige Beispiele für das Training zu generieren, also Bilder die den Bildern mit Gesichtern möglichst ähnlich sind bis auf das keine Gesichter auf diesen Bildern abgebildet sind.
Die Bilder dieser Videosequenzen wurden in nicht-überlappenden Kacheln von 24×24 Pixeln zerlegt, um Nichtgesichter für das Training zu erhalten. In Tabelle 4.4 auf Seite 52 ist eine genaue Übersicht zu den verwendeten Trainingsdaten gegeben.
Das Training wurde mit den gleichen Zielraten für die Detektionsrate und die Falsch-Positiv-Rate gestartet wie zuvor das Training der MIT-Kaskade, also pro Stufe 98.5% Detektionsrate bei 45% Falsch-Positiv-Rate.

4.3 Ergebnisse

4.3.1 MIT-Intensitätsbilder

Die erreichten Klassifikationsleistungen sowie der strukturelle Aufbau der Kaskade sind in Tabelle 4.5 auf Seite 55 dokumentiert.
Der AdaBoost Algorithmus benötigt in der ersten Stufe nur zwei schwache Klassifikatoren um die geforderte Detektionsrate von 98% bei maximal 45% Falsch-Positiv-Rate zu erreichen. Hierbei liegt die Falsch-Positiv-Rate mit 20% sogar noch deutlich unter der geforderten Falsch-Positiv-Rate.

Ein Grund hierfür könnten die Rauschbilder sein, mit denen die Trainingsmenge der Nichtgesichter initial gefüllt war. Denn unter der Annahme, dass diese Rauschbilder zu ähnlichen Featurewerten führen, hätte dies dann zur Folge, dass von AdaBoost ein schwacher Klassifikator ausgewählt wird, der diese Rauschbilder korrekt als Nicht-Gesicht klassifiziert. Denn die Rauschbilder stellen in der Trainingsmenge für den ersten starken Klassifikator ja bereits 50% aller Nichtgesichter.

Den kleinsten gewichteten Fehler würde ein schwacher Klassifikator liefern, der einen Schwellenwert hat, der diese Rauschbilder korrekt klassifiziert. Zusammengefasst haben wir dem Trainingsalgorithmus also wahrscheinlich unnatürlich einfache Beispiele für Nichtgesichter gegeben. Nun kommt in diesen Zusammenhang die Frage auf, inwieweit Rauschbilder als Beispiele für Nichtgesichter überhaupt ihre Berechtigung haben. Die ist aber gegeben, da in der Arbeit von Lechner zur Gesichtsdetektion mit Viola und Jones [Lec07] festgestellt wurde, dass der Detektor ohne Rauschbilder in der Trainingsmenge schlechter generalisiert.
Allerdings liegen die Raten des zweiten starken Klassifikators auf der Validierungsmenge ebenfalls in einem sehr guten Bereich von 21,5% Falsch-Positiv Rate bei ebenfalls nur zwei verwendeten schwachen Klassifikatoren. Da die Rauschbilder nach der Kaskadenstufe zu einem Großteil schon ausgefiltert wurden, deutet dies Ergebnis an, dass AdaBoost auf der gegebenen Trainingsmenge gut lernt. Die weitere Struktur der Kaskade deckt sich dann aber mit den Erwartungen die wir aufgrund der Struktur des Algorithmus erwarten würden. Als Faktoren hierfür wären zu nennen:

Kaskaden stufen	Schwache Klassifikatoren	Trainings-menge		Validierungs-Menge	
	Helligkeitsfeat.	Det-Rate	FP-Rate	Det-Rate	FP-Rate
1	2	98,6%	20,1%	98,3%	20,4%
2	2	99,1%	20,8%	98,3%	21,5%
3	3	99,8%	40,1%	98,6%	39,8%
4	4	99,4%	46,2%	98,8%	45,0%
5	5	98,7%	39,4%	98,3%	39,0%
6	7	99,1%	29,8%	98,0%	30,0%
7	8	99,3%	43,0%	98,0%	43,3%
8	12	99,3%	39,9%	98,0%	39,2%
9	18	99,8%	43,6%	98,0%	44,3%
10	17	99,6%	43,7%	98,0%	43,9%
11	30	100%	43,1%	98,0%	44,5%
12	27	100%	42,7%	98,0%	43,0%
13	29	99,8%	41,5%	98,0%	43,5%
14	38	100%	43,2%	98,0%	42,9%
15	35	100%	44,5%	98,0%	45,0%
Gesamt	237				

Gesamt-Det-Rate	75,59%
Gesamt-FP-Rate	$4.87 \cdot 10^{-7}$

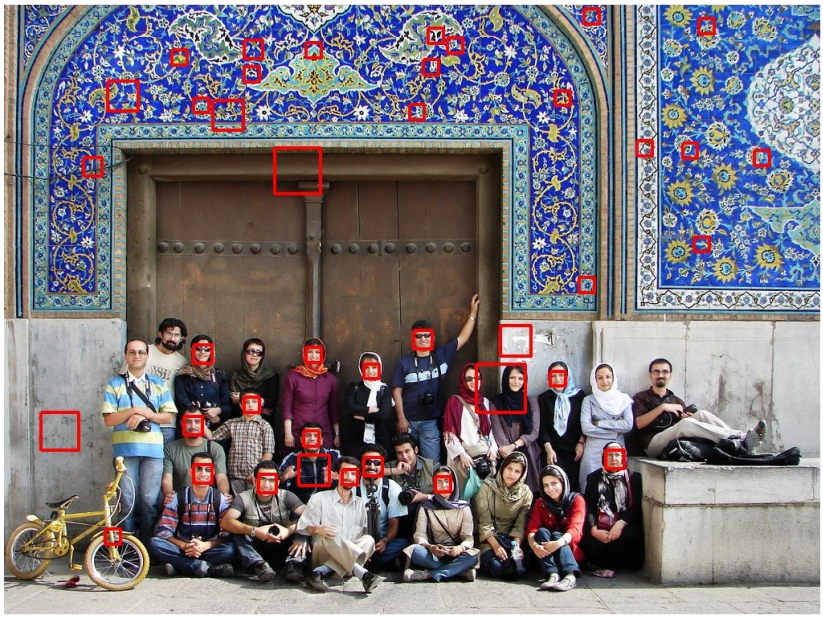
Tabelle 4.5: Struktur der auf dem MIT-Datensatz trainierten 2D-Detektor im Detail. Die berechneten Falsch-Positiv- und Detektionsraten wurden während des Trainings bestimmt. Für beispielhafte Detektionen siehe Abbildung 4.3 auf Seite 57.

- Die Zahl der benötigten schwachen Klassifikatoren steigt mit jeder Kaskadenstufe.
- Die Falsch-Positiv-Raten der einzelnen Stufen erfüllen die geforderten Raten nur knapp.

Dass wir für die gleiche vorgegebene Klassifikationsleistung (98% Detektionsrate bei max 45% Falsch-Positiv-Rate) zunehmend mehr schwache Klassifikatoren benötigen, entspricht unseren Erwartungen, da das zu lösende Problem für den Trainingsalgorithmus mit jeder Stufe schwieriger wird.

Nach jeder fertig trainierten Kaskadenstufe werden korrekt klassifizierte Nichtgesichter ausgefiltert und durch Nichtgesichter ersetzt, die die bisherige Kaskade noch falsch klassifiziert. Und dass die Falsch-Positiv-Rate der einzelnen Stufen nur knapp die geforderte Rate erfüllt, geht mit dem ersten Punkt einher: Da ein einzelner schwacher Klassifikator bei steigender Schwierigkeit der Klassifikation die gesamte Klassifikationsleistung eines starken Klassifikators nur wenig verbessern kann, erwarten wir, dass eine fertige Kaskadenstufe die geforderte Rate auf der Validierungsmenge nur knapp erfüllt.
Insgesamt benötigt die Kaskade 237 schwache Klassifikatoren. Dieser Wert ist, verglichen mit der Kaskade die mittels der Implementation in MATLAB von M. Lechner [Lec07] trainiert wurde, ziemlich gut. Die Kaskade der MATLAB Implementation benötigte 636 schwache Klassifikatoren, wurde aber auch auf nur 1000 Gesichtern trainiert, während die in dieser Arbeit trainierte Kaskade auf dem kompletten Satz der MIT-Daten (2431 Gesichter) trainiert wurde. Dieser Faktor hat scheinbar auch zu einer besseren Generalisierungsleistung des Detektors geführt.

15 Stufen



10 Stufen

Abbildung 4.3: Ausgabe des Detektors: Im oberen Bild wurde der vollständig austrainierte Klassifikator mit 15 Kaskadenstufen verwendet, während in der unteren Abbildung nur die ersten 10 Stufen der Kaskade verwendet wurden. Anhand dieses Beispiels sieht man gut, dass mit jeder Kaskadenstufe die Falsch-Positiv-Rate sinkt, aber auch die Detektionsrate.

4.3.2 Time-Of-Flight-Bilder

Um einen Vergleich anstellen zu können, was Tiefeninformationen für das Training eines Klassifikators mit dem Viola-Jones-Verfahren bringen, wurden sowohl Trainingsläufe auf den Time-Of-Flight-Bildern sowie auf den Intensitätsbildern der Time-Of-Flight-Bilder gestartet. Bei diesen Trainingsläufen kam es aber zu Problemen, so dass keiner dieser Trainingsläufe bis zur vorgegebenen Klassifikationsleistung zu Ende trainiert werden konnte. Bei einem Training, das Probleme hatte zeigten sich folgende zwei Faktoren:

1. Es werden sehr viele schwache Klassifikatoren benötigt.

2. Die Falsch-Positiv-Rate einer zu trainierenden Kaskadenstufe sinkt nicht mehr kontinuierlich, sondern steigt teilweise wieder an.

Ein weiteres Problem war die lange Trainingsdauer einer Kaskade. Im Vergleich zu der MIT-Kaskade ist das Training einer Time-Of-Flight-Kaskade bis zu einem Faktor drei langsamer. Der Grund hierfür ist zum einen, dass die Trainingsbilder mit 24×24 eine höhere Auflösung als die des MIT-Datensatzes mit 19×19 haben. Zum anderen müssen bei einem Training auf Tiefen- und Intensitätsbildern natürlich auch doppelt so viele Bilder verarbeitet werden.
Trotz dieser Probleme lassen sich einige Erkenntnisse aus den Ergebnissen der Trainingsläufe ziehen. Bei dem Trainingslauf, der nur auf den Intensitätsbildern des Time-Of-Flight-Datensatzes gestartet wurde, stellte sich heraus, dass die Intensitätsbilder alleine nicht gut genug sind, damit der AdaBoost-Klassifikator die geforderten Raten erfüllen kann. Auf den Intensitätsbildern wurde nur eine Gesamt-Falsch-Positiv-Rate von $2.6 \cdot 10^{-2}$ erreicht, während $1 \cdot 10^{-6}$ anvisiert war. Da das Training auf den Bildern des MIT-Datensatzes gut funktionierte und zusätzlich beim Training auf den Intensitätsbildern des Time-Of-Flight-Datensatzes die gleiche Software eingesetzt wurde, zeigt dies das die Trainingsdaten nicht optimal sind. Im Vergleich kann klar festgestellt werden das der Klassifikator auf dem MIT-Datensatz besser trainiert hat.

Das Training auf den Intensitätsbildern des Time-Of-Flight-Datensatzes wurde nach über einer Woche Laufzeit abgebrochen, nachdem die oben be-

sprochenen Symptome aufgetreten waren. Die bis dahin erzielten Ergebnisse dieses Klassifikators sind in Tabelle 4.6 auf Seite 60 dokumentiert. In dieser Tabelle sieht man auch die Ursache für die schlechte Gesamt-Falsch-Positiv-Rate in Form von sehr vielen schwachen Klassifikatoren ab der dritten Stufe. Im Vergleich dazu wird in Tabelle 4.7 auf Seite 61 ein Klassifikator präsentiert der Tiefenfeatures verwendet. Die Ergebnisse dieses Klassifikators sind schon deutlich besser und beantworten positiv die Frage, ob Tiefeninformationen für das Training eines Klassifikators mittels des Viola-Jones-Verfahrens von Wert sind. Die erste Stufe der Kaskade erreicht mit nur einem schwachen Klassifikator eine Detektionsrate von 99.8% bei einer Falsch-Positiv-Rate von 1.2%. Da die MIT-Kaskade (vgl. Tabelle 4.5 auf Seite 55), solche Raten nicht erreichen konnte, ist es nun nicht verwunderlich, dass der schwache Klassifikator der ersten Kaskadenstufe auf einem Tiefenfeature basiert. Es handelt sich um ein Feature, das über das gesamte Bild geht und aus drei vertikalen Balken besteht. Dass dieses Feature solch gute Raten erzeugt, liegt wahrscheinlich daran, dass die äußeren beiden Balken des Features meist höhere Tiefenwerte aufweisen als der mittlere Balken. Dies sieht man deutlich auf der Abbildung 4.4 auf Seite 62, die beispielhaft ein Trainingsgesicht des Time-Of-Flight-Datensatzes zeigt. Ein weiterer Grund, der zu der guten Klassifikationsleistung der ersten Kaskadenstufe beigetragen haben kann, sind die wieder verwendeten Rauschbilder, aus denen die Menge der Nichtgesichter initial wieder zu 50% bestand.
Auch die weiteren Stufen des Klassifikators kommen im Vergleich zu der MIT-Kaskade zunächst mit einer geringeren Anzahl an schwachen Klassifikatoren aus, um die geforderten Raten zu erfüllen. Zusätzlich ist zu beobachten, dass das Verhältnis zwischen Tiefen- und Helligkeitsfeatures in den ersten vier Stufen zunächst gleich ist, später dann aber leicht mehr schwache Klassifikatoren Helligkeitsfeatures verwenden. Ein Grund hierfür könnte sein, dass die Tiefenbilder der Time-Of-Flight-Kamera verrauschter sind als die Intensitätsbilder und somit weniger Information zur Klassifikation beisteuern.

Unter der Beobachtung, dass der Trainingsdatensatz nicht gut genug ist, um einen Klassifikator nur basierend auf den Helligkeitsdaten zu trainieren, ist das erzielte Ergebnis auf den kompletten Time-Of-Flight-Daten beacht-

Kaskaden stufen	Schwache Klassifikatoren	Trainings- menge		Validierungs- Menge	
	Helligkeitsfeat.	Det-Rate	FP-Rate	Det-Rate	FP-Rate
1	9	100%	31,2%	98,0%	33%
2	26	100%	40,2%	98,0%	40%
3	103	100%	42,9 %	98,0%	44%
4	821	100%	44,7%	98,0%	44%
Gesamt	959				

Gesamt Det-Rate	100%
Gesamt FP-Rate	$2.6 \cdot 10^{-2}$

Tabelle 4.6: Klassifikationsleistung des nur auf den Helligkeitsdaten des Time-Of-Flight Datensatz trainierten 2D-Detektor im Detail. Dieser Klassifikator konnte trotz 959 verwendeter schwacher Klassifikatoren nicht annähernd die geforderte Klassifikationsleistung erbringen.

lich. Die Tiefeninformationen ermöglichen es letztlich, dass auf einem nachweislich weniger guten Trainingsdatensatz ein Klassifikator trainiert werden kann, der fast die geforderte Klassifikationsleistung erreicht. Ein weiterer Punkt der dafür spricht, dass der Trainingsdatensatz verbessert werden sollte, ist folgender: Die Klassifikation auf den Helligkeitsbildern der Time-Of-Flight Kamera funktioniert mit der auf dem MIT-Datensatz trainierten Kaskade erstaunlich gut, obwohl die Amplitudenbilder der Time-Of-Flight Kamera im Gegensatz zu den MIT-Trainingsdaten nicht unter sichtbaren Licht aufgenommen wurden, sondern unter Infrarotlicht.

Zur Bestimmung der Detektionsleistung des trainierten Klassifikators wurden von den 1300 Time-Of-Flight-Bildern wie bereits erwähnt eine Testmenge von 200 Bildern zufällig ausgewählt. Um die Anzahl der korrekten Detektionen, sowie Falsch-Positiv-Detektionen automatisch bestimmen zu können, wurde für jedes Bild der Testmenge jede Detektion des Klassifikators mit den manuell gelabelten Gesichtern verglichen. Lagen die obere rechte Ecke sowie die untere linke Ecke der Detektion abzüglich einer Toleranz im gleichen Bereich wie das gelabelte Gesicht und überlappten zusätzlich

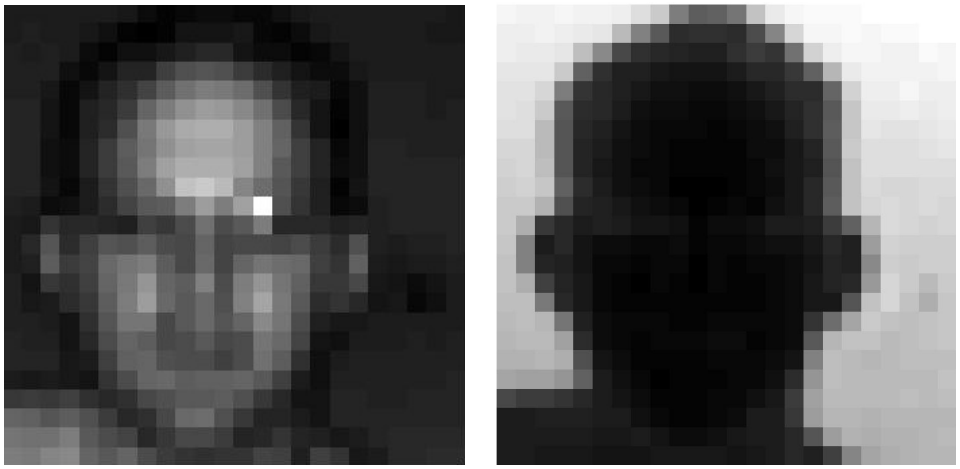
Kaskaden-stufen	Schwache Klassifikatoren		Trainings-menge		Validierungs-Menge	
	Hellig.	Tiefe	Det-Rate	FP-Rate	Det-Rate	FP-Rate
1	0	1	99.8%	1.3%	99.8%	1.2%
2	2	2	100%	33.6%	98.7%	33%
3	1	1	100%	11.9%	99.9%	12%
4	1	1	99.6%	8.1%	99.1%	8.4%
5	3	2	100%	40.6%	99.2%	40%
6	4	3	100%	44.1%	98.1%	44%
7	13	9	100%	32.8%	98%	34%
8	44	22	100%	42.9%	98%	44%
Gesamt (Kanal)	68	41				
Gesamt	109					
Prozent	62.39%	37.61%				

Gesamt Det-Rate	91,24%
Gesamt FP-Rate	$1.12 \cdot 10^{-6}$

Tabelle 4.7: Klassifikationsleistung des 3D-Detektors der auf einem Time-Of-Flight Datensatz trainiert wurde im Detail. Die berechneten Falsch-Positiv- und Detektionsraten wurden während des Trainings bestimmt. Für beispielhafte Detektionen siehe Abbildung 4.5 auf Seite 63

die beiden Bereiche um mindestens 30% so wurde die Detektion als korrekt gewertet. Andernfalls handelte es sich um eine Fehldetektion.
Die trainierte Time-Of-Flight-Kaskade aus Tabelle 4.7 auf Seite 61 erreichte in diesem Test eine Detektionsrate von 100%, hat also alle 200 Gesichter der Testmenge detektieren können. Hierbei aber auch eine relative hohe Anzahl an Fehldetektionen produziert. Auf die 200 Bilder, kommen 125 Fehldetektion, also ca 1.6 Fehldetektion pro Bild. Typische Werte an Fehldetektionen pro Bild sind ein bis zwei Fehldetektionen.

Auch wenn diese Werte noch gut aussehen, so ist an den Fehldetektionen der Time-Of-Flight-Kaskade (vgl. 4.5 auf Seite 63) zu sehen, das nicht die gleiche Klassifikationsleistung erbracht wird wie von der MIT-Kaskade (vgl.



Amplitudenbild (Helligkeit) Phasenbild (Tiefe)

Abbildung 4.4: Beispiel eines verwendeten Trainingsbild für das Training der Time-Of-Flight Kaskade. Die verwendete Auflösung beim Training beträgt 24×24 Pixel.

4.3 auf Seite 57). Die Fehldetektionen der MIT-Kaskade sehen schon optisch betrachtet teilweise wie Gesichter aus, während für die Fehldetektionen der Time-Of-Flight-Kaskade kein optischer Grund auffällt.

Die genaue Verteilung der Fehldetektionen pro Bild ist in Tabelle 4.8 auf Seite 62 dokumentiert.

Anzahl Fehl-detektionen	Häufigkeit
0	92
1	91
2	17
> 2	0
Summe	200

Tabelle 4.8: Verteilung der Falsch-Positiv Detektionen auf die 200 Bilder der Testmenge.

Zusammenfassend kann festgestellt werden, dass in den Tiefeninformationen ein großes Potential zur Detektion von Gesichtern liegt. Obwohl leider im Rahmen dieser Arbeit kein zur MIT Kaskade vergleichbar guter Klassifikator auf den Time-Of-Flight Daten trainiert werden konnte, ist die erreich-

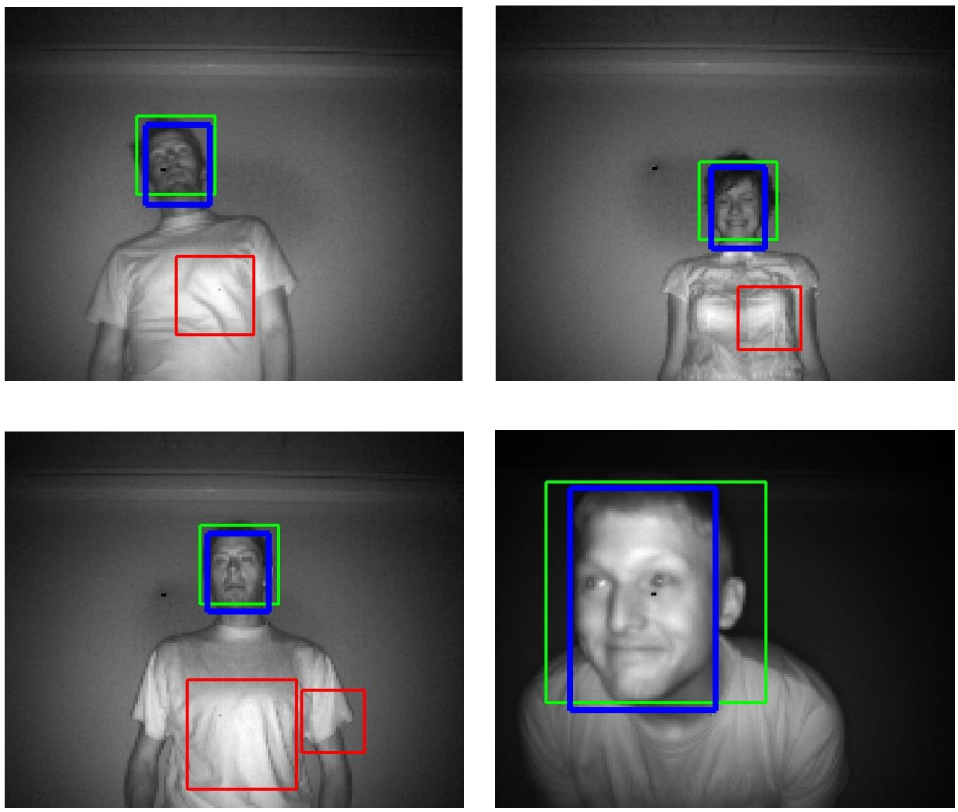


Abbildung 4.5: Beispiel Detektionen des Time-Of-Flight Klassifikators: Die Detektionsrate liegt bei allen Bildern der Validierungsmenge bei 100%, allerdings bei ca. 1-2 Falsch-Positiv-Detektionen pro Aufnahme. Legende: Rote Markierungen: Falsch-Positive Detektionen, Blaue Markierungen: Handlabelungen, Grüne Markierungen: Korrekte Detektionen

te Detektionsleistung von 91% bei einer Falsch-Positiv Rate von $1.12 \cdot 10^{-6}$ achtbar.

4.4 Echtzeitfähigkeit des trainierten Detektors

Ein wesentliches Kriterium bei der Wahl des Algorithmus sowie der Implementationssprache war die gewünschte Echtzeitfähigkeit des Gesichtsdetektors. Das bedeutet, dass der Detektor später als Modul für die Time-Of-Flight-Kamera zur Verfügung stehen sollte. Dieses Modul muss dann in der Lage sein, ein von der Kamera aufgenommenes Time-Of-Flight-Bild

auf Gesichter zu untersuchen sowie überlappende Detektionen zu einer Detektion zusammenzufassen (vgl. Abschnitt 3.10 ab Seite 42). Abschließend müssen diese Detektionen ins gelieferte Time-Of-Flight-Bild der Kamera eingezeichnet werden und angezeigt werden. Für diese ganzen Aufgaben stehen dem Programm 42 Millisekunden zur Verfügung, wenn man von einer Bildwiederholrate von 24 Bildern pro Sekunde ausgeht.

Da der komplette Algorithmus von Viola und Jones auf eine Gesichtsdetektion in Echtzeit ausgelegt ist, ist diese Leistung möglich. Im wesentlichen trägt hierzu die einmalige Vorverarbeitung des Eingabebildes zu einem Integralbild (vgl. Abschnitt 3.2 ab Seite 15) sowie der kaskadenartige Aufbau der Detektors (vgl. Abschnitt 3.6 ab Seite 29) bei.

Um zu messen, bis zu welcher Eingabegröße auf dem eingesetzten PC (Core2 Duo E6750 siehe Daten im Abschnitt 4.1.2 ab Seite 47) eine Verarbeitung in Echtzeit erfolgen kann, wurden einige Laufzeittest durchgeführt. Dem trainierten Detektor (siehe Tabelle 4.5 auf Seite 55) wurden Intensitätsbilder mit Gesichtern in Auflösungen von 0,1 bis 0,8 Megapixeln gegeben. Aus den gemessen Zeiten ließ sich dann Erkennen, dass eine Verarbeitung in Echtzeit bei Bildern bis zu 0,2 Megapixeln zu erreichen ist. Die genauen Daten der Messung sind in Abbildung 4.6 auf Seite 66 protokolliert. Bei der Abbildung fällt auf, dass eine genaue Zuordnung von Bildwiederholrate zur Auflösung des Eingabebildes nicht möglich ist: Dies lässt sich aber dadurch erklären, dass nicht jedes Eingabebild gleich „schwierig“ ist. Damit ist gemeint, dass der Detektor bei leicht zu klassifizierenden Bildern weniger schwache Klassifikatoren (vgl. Abschnitt 3.5 ab Seite 25) auswerten muss und somit bei einen solchen Bild weniger rechnen muss als bei einem Bild mit gleicher Auflösung und schwierig zu klassifizierenden Strukturen.

Die Performance auf den Time-Of-Flight-Daten wurde mit der nativen Auflösung der verwendeten Kamera gemessen. Da die Kamera nur eine Auflösung von 176×144 (ca. 0,05 Megapixel) hat, wurde hier erwartet, dass eine Echtzeitverarbeitung problemlos möglich ist. Diese Erwartungen bestätigten sich dann auch in den Messungen.

Zusammenfassend wurde in dieser Arbeit ein Detektor präsentiert, der auf einem aktuellen PC und mit dem Verfahren von Viola und Jones Gesichtsdetektion in Echtzeit bis zu einer Auflösung von 0,2 Megapixel ermöglicht.

Dabei ist zu beachten, das in dieser Arbeit rein auf der CPU des PCs gerechnet wurde. Mittels speziell auf die Berechnung von Features angepasste Hardware in Form von FPGAs[1] könnte die Geschwindigkeit weiter gesteigert werden.

[1]FPGA: Field Programmable Gate Array (programmierbarer Integrierter Schaltkreis (IC) der Digitaltechnik)

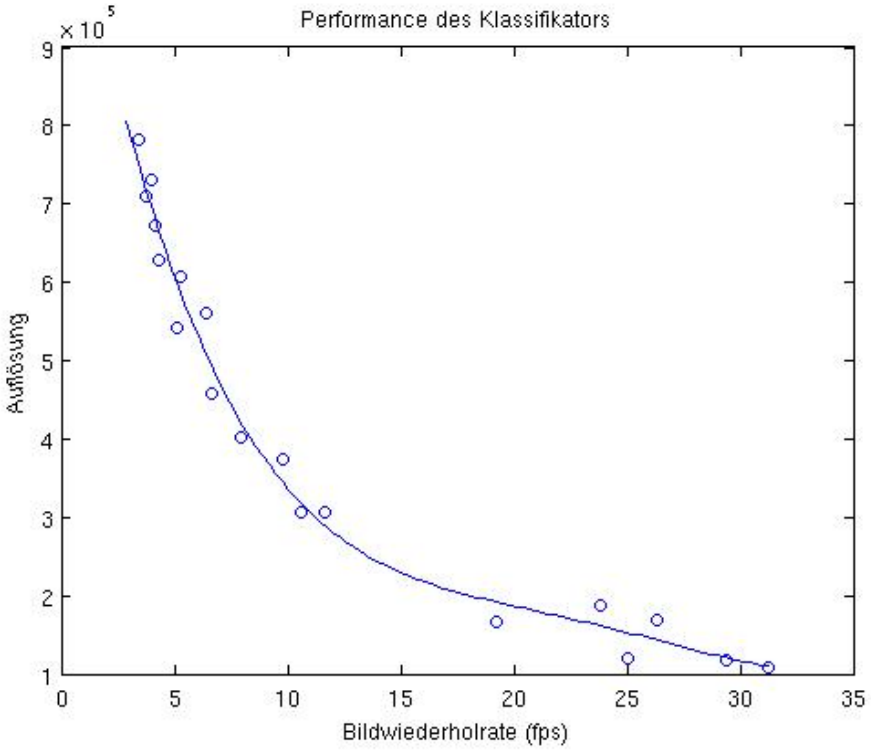


Abbildung 4.6: Bildwiederholraten des Detektors im Verhältnis zur Größe der Eingabebilder. Gemessen wurde die Verarbeitungsdauer des auf dem MIT-Datensatz trainierten Detektors (siehe Tabelle 4.5 auf Seite 55). Mit steigender Auflösung der Eingabebilder sinkt erwartungsgemäß die erreichte Framerate des Klassifikators. Eine Echtzeitklassifikation ist mit der trainierten Kaskade auf Bildern von bis zu 0,2 Megapixeln möglich.

5 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Gesichtsdetektor nach dem Verfahren von Viola und Jones in C++ implementiert. Dieser Detektor ermöglicht es, auf einem aktuellen PC Videostreams bis zu einer Auflösung von 0,2 Megapixeln in Echtzeit nach Gesichtern zu durchsuchen und diese zu markieren. Als Erweiterung des Verfahrens von Viola und Jones wurde die Anpassung des Klassifikators für Time-Of-Flight-Daten vorgenommen. Hierbei wurden die Tiefeninformationen einer Time-Of-Flight-Kamera verwendet, um zu überprüfen, ob sich das Verfahren so weiter verbessern lässt.
Bei der Beantwortung dieser Frage stießen wir auf einige Probleme mit dem Trainingsdatensatz der Time-Of-Flight-Bilder, da es nicht gelang, einen Klassifikator mit den gewünschten Klassifikationsraten zu trainieren.
Die Antwort auf die Frage, ob Tiefeninformationen von Nutzen sind, kann trotz der Probleme klar gegeben werden. Tiefeninformation, wie sie die von uns verwendete Kamera liefert, sind definitiv von Nutzen für die Gesichtsdetektion, da AdaBoost schwache Klassifikatoren auswählt, die auf diesen Tiefeninformationen arbeiten. Folgende Ergebnisse konnten mit der Implementation dieser Arbeit erreicht werden:

- Es konnte ein Klassifikator auf dem im Institut erstellten Time-Of-Flight-Trainingsdaten trainiert werden, der die geforderte Klassifikationsrate annähernd erfüllte. Die Rate der Fehldetektionen ist noch zu hoch. Es ist aber zu hoffen, dass ein verbesserter Trainingsdatensatz noch bessere Ergebnisse erzielt.

- Auf dem gleichen Time-Of-Flight-Datensatz konnte kein funktionierender Klassifikator erstellt werden, sofern nur die Amplitudendaten (Helligkeit) der Time-Of-Flight-Bilder verwendet wurden. Das zeigt,

dass die Tiefeninformationen einen erheblichen Beitrag zur Klassifikation leisten.

- Der auf AdaBoost basierende Trainingsalgorithmus sucht schwache Klassifikatoren, die auf den Tiefeninformationen arbeiten, aus. Das Verfahren ist so aufgebaut, dass es sich immer den schwachen Klassifikator heraussucht, der den kleinsten gewichteten Fehler macht. Wenn keine relevante Information für die Gesichtsdetektion in den Tiefenbildern der Time-Of-Flight-Kamera vorhanden wäre, würde der AdaBoost-Klassifikator keine schwachen Klassifikatoren verwenden, die auf den Tiefeninformationen der Time-Of-Flight-Kamera arbeiten.

Da das Training auf den Intensitätsdaten alleine keine zufriedenstellenden Detektionsraten erreicht, vermuten wir, dass die Trainingsdaten nicht optimal waren. Nachfolgend werden einige der Probleme mit dem Trainingsdatensatz aufgeführt:

- Alle 3300 für das Training verwendeten Trainingsbilder wurden aus Filmsequenzen von nur 17 Personen generiert. Ein Datensatz mit einer solch geringen Anzahl an Personen deckt wahrscheinlich den Raum aller Gesichter nicht gut genug ab.

- Viola und Jones haben zunächst den MIT-Datensatz verwendet und dann festgestellt, dass sie bessere Ergebnisse erzielen, wenn sie die Subfenster mit den Gesichtern größer ausschneiden, also größere Ränder um das Gesicht herum lassen. Diesem Beispiel folgend haben auch wir die Trainingsbeispiele aus den Time-Of-Flight-Bildern mit einem größeren Rand ausgeschnitten (vgl. hier das exemplarische Time-Of-Flight-Trainingsbild (Abbildung 4.4 auf Seite 62) mit den Trainingsbildern des MIT-Datensatzes (Abbildung 4.2 auf Seite 50)). Da unsere Labelung doch einen sehr großen Rand um das Gesicht läßt, könnte diese Labelung vielleicht doch zu großzügig gewesen sein.

- Um die Menge der Trainingsbilder weiter zu vergrößern, wurde ähnlich wie beim MIT-Datensatz jedes Gesicht nochmals um ±5°rotiert.

Bei weiteren Versuchen sollte hier mit kleineren Gradzahlen gearbeitet werden. So kann getestet werden, wie stark die Rotierung der Trainingsbeispiele wirklich Einfluss auf die Generalisierungsleistung des Klassifikators hat.

- Die Labelungen der Gesichter könnten zu inkonsistent durchgeführt worden sein.

Eine praktische Frage, die sich zu Anfang dieser Arbeit stellte, war die Frage, ob das Verfahren klar 3D-Gesichter von 2D-Gesichtern (also Fotos von Gesichtern) unterscheiden kann. Diese Fragestellung kam auf, als wir auf einige Pressemitteilungen [hei08, Spi08] über Zigarettenautomaten aus Japan gestoßen sind. Diese Automaten verwenden Gesichtsdetektion als Vorverarbeitungsschritt, um im gefundenen Gesicht nach Merkmalen für ein altes (erwachsenes) Gesicht zu suchen. Diese Automaten lassen sich mittels eines Fotos täuschen und ermöglichen es so, den Prozess zur Altersbestimmung zu täuschen. Leider konnte in dieser Arbeit kein Detektor trainiert werden, der genau diese Aufgabe erfüllt.
Es bestehen aber gute Chancen, einen Klassifikator zu trainieren, der die Aufgabe, ein 2D-Gesicht (Foto) von einem echten Gesicht zu unterscheiden erfüllt. Am meisten Potential bietet das Training mit einem Time-Of-Flight-Datensatz, der die angesprochenen Probleme berücksichtigt. Problematisch ist hierbei der große zeitliche Aufwand für die Erstellung des Trainingsdatensatzes. Deshalb war beispielsweise eine Idee, die auf den reinen Helligkeitsdaten des MIT-Datensatzes trainierte Kaskade zu verwenden, um eine automatische Labelung von Time-Of-Flight-Bildern auf Grundlage des Amplitudenbildes zu ermöglichen. So könnte der Arbeitsaufwand des Handlabelungsprozess erheblich reduziert werden, da so nur noch von einem Menschen kontrolliert werden muss, ob die Detektionen korrekt sind.
Ein weiteres Problem beim Experimentieren mit einem neuen Trainingsdatensatz ist die Zeit, die das Training eines Klassifikators benötigt. Für das Training der Time-Of-Flight-Kaskade wird mit dem verwendeten Trainingsdatensatz auf einem aktuellen PC mindestens eine Woche pro Trainingslauf benötigt.
Um dieses Problem zu lösen, bietet sich die Parallelisierung des Trainings-

prozesses perfekt an. Denn beim Training des Klassifikators wird ja über eine vollständige Suche über alle existierenden schwachen Klassifikatoren bestimmt, welches der beste schwache Klassifikator ist. Diese Suche lässt sich aber auch parallel ausführen. Dies ist so gut möglich, da alle schwachen Klassifikatoren unabhängig voneinander trainiert werden können. Letztendlich entspricht diese Aufgabe der Minimumssuche auf einem Vektor. Man würde also das Training dieser schwachen Klassifikatoren auf N Prozessoren aufteilen. Nachdem nun jeder Prozessor die gewichteten Fehler von $\frac{1}{N}$ aller zu testenden schwachen Klassifikatoren bestimmt hätte, müsste die Haupt-CPU nur noch eine Minimumssuche auf den N Ergebnissen der Prozessoren ausführen. Da die Berechnung der schwachen Klassifikatoren der weitaus rechenintensivste Bereich beim Training des Detektors ist, ließe sich allein über die Parallelisierung dieses Abschnitts ein Speedup von nahezu N erreichen. Angemerkt sei hier noch, dass diese Überlegungen für eine Maschine mit gemeinsamen Speicher gelten. Aber auch um das Training für einen Cluster zu realisieren, bei dem jeder Rechenknoten seinen eigenen Speicher hat, wäre der Aufwand nicht viel größer, da der Kommunikationsaufwand bei unserem Problem gering ist. Es muss nur der beste schwache Klassifikator an den Hauptknoten des Clusters zurückgeliefert werden. Der Hauptknoten wertet dann wieder genau wie bei einer Maschine mit gemeinsamen Speicher den besten Klassifikator aus. Der einzige Unterschied wäre, dass die Bilddaten zu Anfang auf jeden Knoten des Clusters geladen werden müssten, beispielsweise von einem verteilten Dateisystem.
Ein weiterer interessanter Punkt, den es sich lohnen würde zu untersuchen, wäre die Analyse, inwieweit Farbinformationen für die Gesichtsdetektion von Nutzen sind. Um den Nutzen von Tiefeninformationen zu testen, wurde die Implementation des Klassifikators in dieser Arbeit auf der Abstraktionsebene von Bildkanälen realisiert. Unter dieser Voraussetzung wäre es nun mit wenig Aufwand möglich, die Zahl der verwendeten Kanäle von einem (bei den MIT-Graustufenbildern) auf drei bei Farbbildern zu erhöhen, um dann zu untersuchen, ob sich so bessere Ergebnisse erzielen lassen als bei der alleinigen Verwendung von einem Kanal.

Verzeichnisse

Literaturverzeichnis

[3DV08] 3DV Systems, 2nd Carmel St. Industrial Park Building 1 P.O.Box 249 Yokneam, 20692 Israel, *Cost-effective 3D camera: ZCam*, 2008, Hersteller der RGBD Zcam: `http://www.3dvsystems.com/technology/product.html#1`.

[Fuj08] Fujifilm Corporation, *Newly developed face detection 3.0*, 08 2008, `http://www.fujifilm.com/products/digital_cameras/f/finepix_f100fd/features/page_03.html`.

[hei08] heise Newsticker, *Gesichtserkennung am Zigarettenautomat zur Altersverifikation*, http://www.heise.de/newsticker/Gesichtserkennung-am-Zigarettenautomat-zur-Altersverifikation–/meldung/107796, 2008.

[Lec07] Martin Lechner, *Gesichtsdetektion mit dem Verfahren von Viola und Jones*, 2007, Studienarbeit am Institut für Neuro- und Bioinformatik.

[Lüb07] Dräger Lübeck, *Dräger Interlock XT*, 08 2007, `http://www.draeger.com/ST/internet/pdf/Master/De/gt/interlock/9044622_interlock_d.pdf`.

[Mes08] Mesa Imaging AG, Technoparkstrasse 1, 8005 Zürich, Schweiz, *Time-of-Flight Sensors*, 2008, Hersteller der SR3000 Time-Of-Flight-Kamera: `http://www.mesa-imaging.ch/prodviews.php`.

[MIT00] MIT Center For Biological and Computation Learning, *Face database*, 2000, `http://cbcl.mit.edu/projects/cbcl/software-datasets/FaceData1Readme.html`.

[Nik07] Nikon USA, *Newest Nikon Coolpix Digital Dameras feature enhanced face detection and wireless capabilities.*, 08 2007, `http://press.nikonusa.com/2007/08/newest_nikon_coolpix_digital_c.php`.

[OI] Opensource community and Intel, *Intel open computer vision library.*

[OKL+04] T. Oggier, R. Kaufmann, M. Lehmann, P. Metzler, G. Lang, M. Schweizer, M. Richter, B. Büttgen, N. Blanc, K. Griesbach, B. Uhlmann, K.-H. Stegemann, and C. Ellmers, *3D-Imaging in Real-Time with Miniaturized Optical Range Camera*, 2004, Mesa Publikationseite: `http://www.mesa-imaging.ch/publications.php`.

[OLK+04] T. Oggier, M. Lehmann, R. Kaufmann, M. Schweizer, M. Richter, P. Metzler, G. Lang, F. Lustenberger, and N. Blanc, *An all-solid-state optical range camera for 3d real-time imaging with sub-centimeter depth resolution (swissranger)*, Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series (L. Mazuray, P. J. Rogers, and R. Wartmann, eds.), Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, vol. 5249, February 2004, pp. 534–545.

[RBK96] Henry A. Rowley, Shumeet Baluja, and Takeo Kanade, *Neural Network-Based Face Detection*, CVPR '96: Proceedings of the 1996 Conference on Computer Vision and Pattern Recognition (CVPR '96) (Washington, DC, USA), IEEE Computer Society, 1996, p. 203.

[Sch90] Robert E. Schapire, *The Strength of Weak Learnability*, Mach. Learn. **5** (1990), no. 2, 197–227.

[SLM+00] Henry Schneiderman, Tai Sing Lee, Carnegie Mellon, Dean Pomerleau, and Assistware Technology, *A Statistical Approach to 3D Object Detection Applied to Faces and Cars*, 2000.

[Spi08] Spiegel Online Netzwelt Redaktion, *Gesichtserkennung Japanische Zigarettenautomaten mit Fotos getäuscht!*, `http://www.spiegel.de/netzwelt/tech/0,1518,562940,00.html`, 2008.

[VJ04] Paul Viola and Michael J. Jones, *Robust Real-Time Face Detection*, Int. J. Comput. Vision **57** (2004), no. 2, 137–154.

Algorithmenverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis