Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg

Gesichtsdetektor

Detektion von Gesichtern in Bildern

Fortgeschrittenen Praktikum im SS 2003 von

Markus Mühling und Adam Sosnowski

Betreuer: Ralph Ewerth

Inhaltsverzeichnis

1. Aufgabenstellung

Das Fortgeschrittenen Praktikum „Gesichtsdetektor“ entstand im Rahmen des Projekts „Methoden und Werkzeuge zur rechnergestützten medienwissenschaftlichen Analyse“. Hierbei handelt es sich um ein gemeinsames Projekt von Medienwissenschaftlern und Informatikern der Universitäten Marburg und Siegen. Ein Teilprojekt beschäftigt sich mit der Detektion von medienwissenschaftlich relevanten Objekten in digitalem Videomaterial. Hierzu zählt auch die Gesichtsdetektion. Die Aufgabe der Gesichtsdetektion, angewendet auf ein gegebenes Bild, besteht darin zu bestimmen, ob in dem Bild Gesichter vorhanden sind, und wenn ja die Bestimmung der Position und der räumlichen Ausbreitung des Gesichts. Es gibt viele Aufgabenstellungen, die in enger Beziehung zur Gesichtsdetektion stehen. Die Gesichtslokalisierung beispielsweise dient der Positionsbestimmung eines Gesichts in einem Bild. Dabei handelt es sich um ein vereinfachtes Detektionsproblem mit der Annahme, dass das Eingangsbild genau ein Gesicht enthält. Ein weiteres Beispiel ist die Gesichtsidentifikation oder Wiedererkennung. Hier wird ein gegebenes Bild mit einer Datenbank abgeglichen und eine Übereinstimmung gesucht. Die Gesichtswiedererkennung wird hauptsächlich im Bereich von Zugangskontrollen verwendet, wie z.B. Europas größte Gesichtserkennungsanlage im Zoo Hannover[1]. Im Vergleich zur Wiedererkennung kann der Computer bei der Gesichtsdetektion nicht erwarten, dass sich genau ein Gesicht in einem bestimmten Bereich des Bildes befindet. Das Bild kann beliebig viele oder keine Gesichter enthalten. Die Gesichter in einem beliebigen Bild können an verschiedenen Stellen platziert sein und zudem in der Größe variieren. Außerdem sind die Gesichter dem Computer nicht bekannt, so daß ein einfaches Patternmatching-Verfahren zum Auffinden der Gesichter nicht ausreichen würde. Das Problem der Gesichtsdetektion wird noch dadurch erschwert, daß Gesichter in einem Bild nicht nur in Ort und Größe variieren können, sondern auch in Orientierung, Beleuchtung, Mimik, Pose und Verdeckung. Gesichter unterscheiden sich nicht nur in den Gesichtszügen und Augen, Nase, Mund. Weitere Faktoren wie Frisur, Brille, Bart, etc. sind als Verfremdung ein und desselben Gesichts zu berücksichtigen.

Ziel dieses Fortgeschrittenen Praktikums war es, einen Algorithmus zur Gesichtsdetektion in Bildern zu implementieren. Der Detektionsalgorithmus soll später zum Face Tracking in MPEG-Videos verwendet werden. Bei der geplanten Tracking-Variante sollen die Gesichter in einer Bildsequenz mit Hilfe der Bewegungsvektoren verfolgt werden. Dieses Anwendungsziel stellt somit in Bezug auf die Performance sehr hohe Anforderungen an den zugrunde liegenden Detektionsalgorithmus. Desweiteren sollte im Rahmen des Fortgeschrittenen Praktikums eine Testapplikation erstellt werden, in die der Detektionsalgorithmus eingebunden wird. Aufgaben der Testapplikation sind vorallem das Laden von Bildern, das Anzeigen von Detektionsergebnissen und die automatische Analyse von Testsets.

Wie kann man nun einem Computer beibringen, was ein Gesicht ausmacht oder wie ein Gesicht aussieht? Auf diese Frage wird in Kapitel 3 eingegangen. Im folgenden Kapitel wird zunächst die Funktionalität der Testapplikation (Gesichtsdetektor) beschrieben.

2. Fachkonzept

Der Gesichtsdetektor ist eine Applikation, die den Ansatz des Naive Bayes Klassifizierers (siehe Kapitel 3) zur Gesichtserkennung verwendet. Bei diesem Ansatz handelt es sich um einen trainingsbasierten Algorithmus. Der Gesichtsdetektor erlaubt es, Bilder zu laden und die Gesichtserkennung darauf anzuwenden. Abbildung 1 zeigt einen Screenshot der Applikation mit einem geladenen Bild. Die grünen Rechtecke im Bild stellen das Ergebnis der letzten Detektion dar. Das Hauptmenü der Applikation gliedert sich in die Einträge Datei, Analyse, Training, Entwicklung und Hilfe. Alle Menüpunkte werden im folgenden beschrieben.

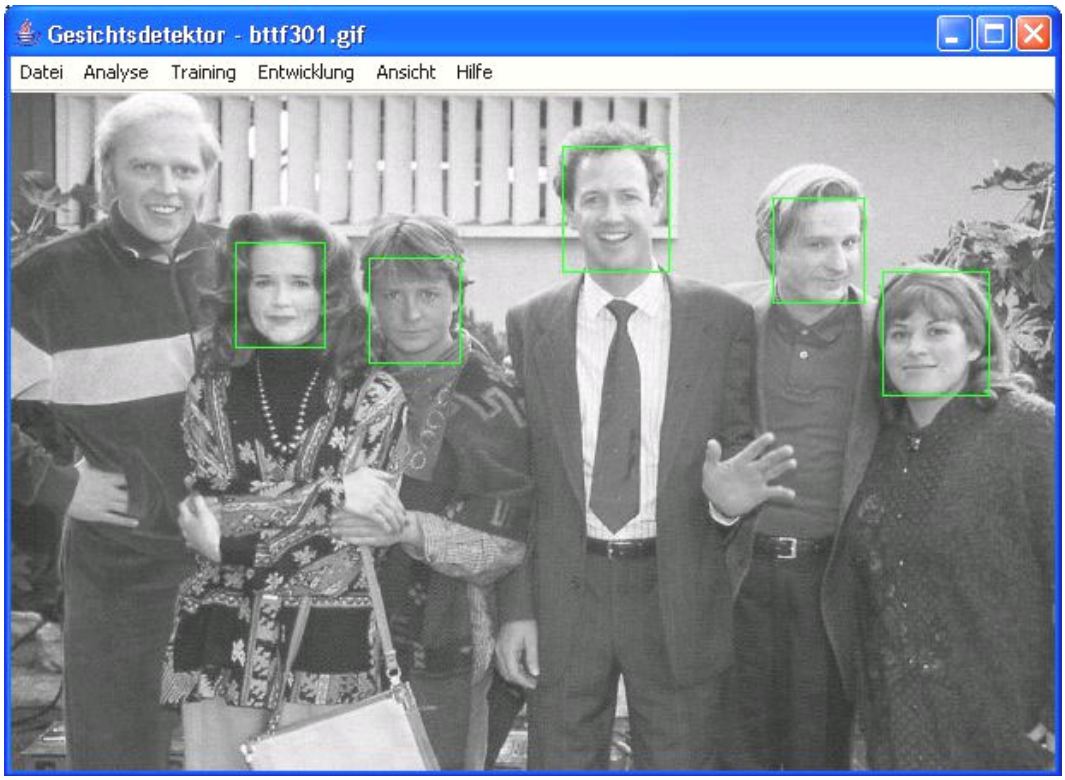


Abbildung 1

2.1. Menü Datei

Abbildung 2

2.1.1. Menüpunkt Datei | Bild laden...

Der Befehl „Bild laden...“ öffnet ein Dialogfenster zum Auswählen einer Bilddatei. Nach erfolgreichem Öffnen der Bilddatei wird der Name der Datei in der Titelzeile des Programmfensters angezeigt. Der Inhalt der Bilddatei wird im Hauptfenster dargestellt.

2.1.2. Menüpunkt Datei | Beenden

Der Befehl „Beenden“ schließt die Anwendung. Falls Trainingsdaten existieren, die noch nicht gespeichert wurden, wird eine Warnung ausgegeben und gegebenenfalls ein Datei-Speichern-Dialog geöffnet.

2.2. Menü Analyse

Abbildung 3

2.2.1. Menüpunkt Analyse | Gesichtserkennung starten

Der Menüpunkt „Gesichtserkennung starten“ ist kontextabhängig. Um ihn zu aktivieren, müssen Trainingsdaten für Frontal, Profil und Nicht-Gesicht existieren. Diese können entweder durch das Laden vorhandener Trainingsdaten oder durch Training erworben werden (siehe Abschnitt 2.3). Zusätzliche Voraussetzung für die Anwendung der Gesichtserkennung ist eine geladene Bilddatei, dessen Bild im Hauptfenster angezeigt wird. Bevor die Gesichtserkennung das erste Mal gestartet wird, sollten der Schwellenwert (siehe Abschnitt 2.2.4), die Bildgröße bei Detektion (siehe Abschnitt 2.2.5) und die Sliding Window Schrittweite (siehe Abschnitt 2.2.6) eingestellt werden. Der Befehl „Gesichtserkennung starten“ wendet den Gesichtserkennungsalgorithmus auf das aktuell geladene Bild an. Die erkannten Gesichter werden im Bild markiert (siehe Abbildung 1).

2.2.2. Menüpunkt Analyse | Batch Modus...

Der Befehl „Batch Modus ... „ führt eine Stapelverarbeitung durch. Der Gesichtserkennungsalgorithmus wird auf alle Bilder eines ausgewählten Verzeichnisses angewendet. Für den Batchmodus muss sich der Gesichtsdetektor in einem trainierten Zustand befinden. Dieser kann entweder durch Training oder das Laden einer bereits vorhandenen Trainingsdatei erreicht werden (siehe Abschnitt 2.3). Bevor der Menüpunkt ausgeführt wird, sollten der Schwellenwert (siehe Abschnitt 2.2.4), die Bildgröße bei Detektion (siehe Abschnitt 2.2.5) und die Sliding Window Schrittweite (siehe Abschnitt 2.2.6) eingestellt werden. Der Verlauf der Stapelverarbeitung wird über einen Fortschrittsbalken dargestellt. Zusätzlich wird das jeweils zuletzt untersuchte Bild samt Detektionen im Hauptfenster angezeigt (siehe Abbildung 4).

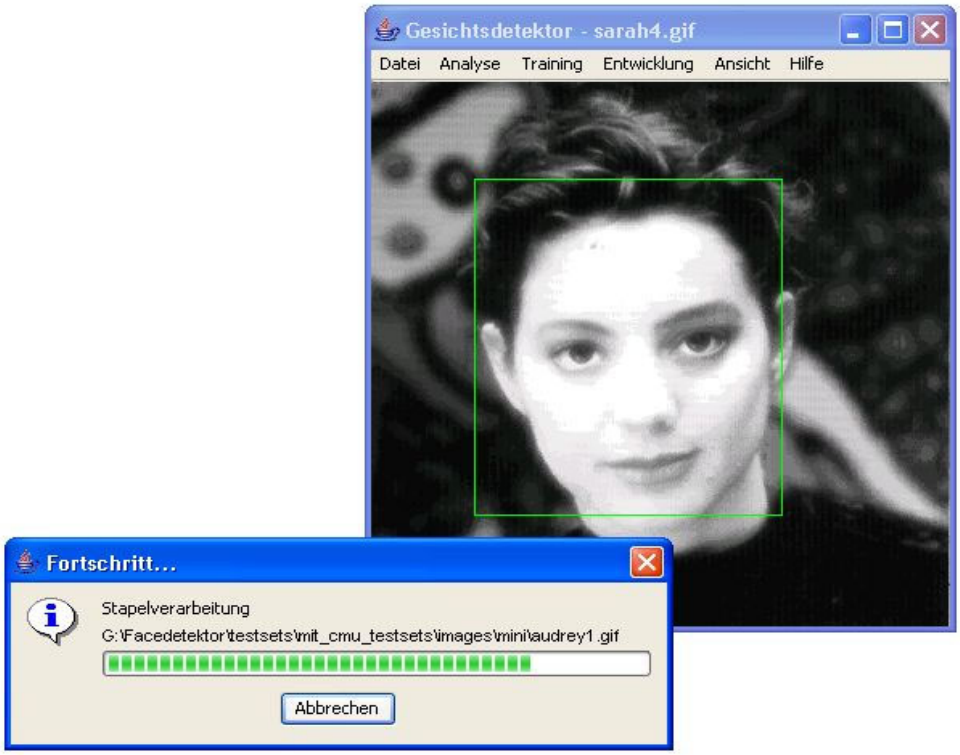


Abbildung 4

Das Gesamtergebnis kann am Ende der Analyse in einer Datei abgespeichert werden. Die Datei ist wie folgt aufgebaut. Für jedes untersuchte Bild werden die folgenden Informationen abgespeichert: Der Dateiname, die Anzahl der detektierten Gesichter und für jedes gefundene Gesicht den linken oberen und rechten unteren Begrenzungspunkt.
Das folgende Beispiel ist ein Auszug aus einer solchen Datei:

```
boat.gif   2
336   231   352   249
494   231   511   248
cnn1630.gif      1
152   75    164   88
```

2.2.3. Menüpunkt Analyse | Testset analysieren...

Um den Befehl „Testset analysieren...“ auszuführen, muss sich der Gesichtsdetektor in einem trainierten Zustand befinden. Dies kann entweder durch Training oder das Laden einer bereits vorhandenen Trainingsdatei erreicht werden (siehe Abschnitt 2.3). Bevor die Analyse des Testsets gestartet wird, sollten der Schwellenwert (siehe Abschnitt 2.2.4), die Bildgröße bei Detektion (siehe Abschnitt 2.2.5) und die Sliding Window Schrittweite (siehe Abschnitt 2.2.6) eingestellt werden. Weiterhin sollte der Menüpunkt „Mehrfachdetektionen anzeigen“ (siehe Abschnitt 2.2.7) ausgeschaltet werden. Der Befehl „Testset untersuchen ...“ analysiert ein Testsets und wertet die Ergebnisse aus. Die Gesichtserkennung wird auf alle Bilder eines ausgewählten Verzeichnisses (Testset) angewendet. Zusätzlich muß eine Datei mit Groundtruth-Daten angegeben werden, die folgende In-

formationen über die Bilder des Testsets enthält: Dateiname, Anzahl der Gesichter und für jedes Gesicht den linken oberen und rechten unteren Begrenzungspunkt. Zum Beispiel:

```
boat.gif   2
336   231   352   249
494   231   511   248
cnn1630.gif      1
152   75    164   88
```

Mit Hilfe dieser Daten wird das Detektionsergebnis ausgewertet. Details darüber, wann ein Gesicht richtig erkannt wurde, werden in Abschnitt 5 näher erläutert. Die Auswertung wird in einem Dialogfenster angezeigt (siehe Abbildung 5).

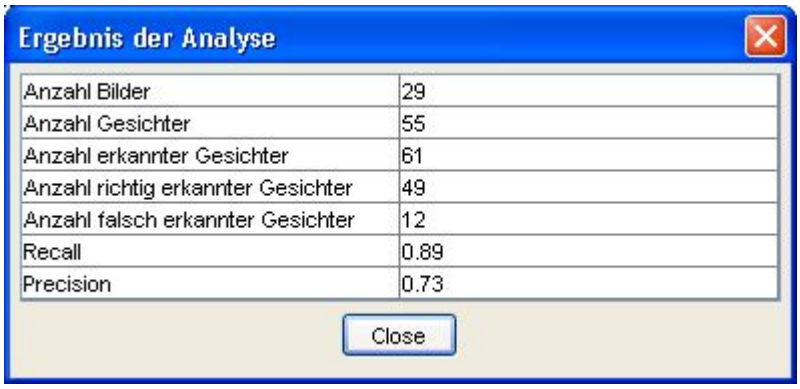


Abbildung 5

Recall und Precision sind wie folgt definiert:

$$\text{Recall} = \frac{\text{Anzahl der richtig erkannten Gesichter}}{\text{Anzahl der Gesichter}}$$

$$\text{Precision} = \frac{\text{Anzahl der richtig erkannten Gesichter}}{\text{Anzahl der Gesichter} + \text{Anzahl der falsch erkannten Gesichter}}$$

2.2.4. Menüpunkt Analyse | Schwellenwert…

Der Schwellenwert (siehe Abschnitt 3.3.5) des Naiven Bayes Klassifizierers ist abhängig von den jeweiligen Trainingsdaten. Der Detektionsalgorithmus berechnet für jeden möglichen Bildausschnitt einen Wert. Je höher dieser Wert ist, desto größer ist die Wahrscheinlichkeit, daß es sich um ein Gesicht handelt. Je niedriger der Schwellenwert also gesetzt wird, desto mehr Bildbereiche werden als Gesicht klassifiziert. Die Bestimmung eines optimalen Schwellenwertes für die aktuellen Trainingsdaten kann in unserer Applikation (Gesichtsdetektor) nur durch Testen („try and error“) erfolgen. Der Schwellenwert wird solange angepaßt, bis die Gesichtserkennung (siehe Abschnitt 2.2.1) akzeptable Ergebnisse liefert. Die einzige Hilfe beim Einstellen des Schwellenwerts liefert der Menüpunkt „Grenzen letzter Detektion...“ (siehe Abschnitt 2.2.5). Der Befehl „Schwellenwert...“ ruft das folgende Dialogfenster auf (siehe Abbildung 6). In diesem Fenster wird der aktuelle Schwellenwert des Naiven Bayes Klassifizierers angezeigt. Durch Setzen eines Wertes kann der Schwellenwert neu eingestellt werden.

Abbildung 6

2.2.5. Menüpunkt Analyse | Grenzen letzter Detektion...

Um das Einstellen des Schwellenwertes (siehe Abschnitt 2.2.4) zu erleichtern, kann man über den Menüpunkt „Grenzen letzter Detektion...“ den minimalen und maximalen Wert der letzten Analyse abrufen (siehe Abbildung 7).

Abbildung 7

2.2.6. Menüpunkt Analyse | Bildgröße bei Detektion...

Der Befehl „Bildgröße bei Detektion...“ ruft das folgende Dialogfenster auf (siehe Abbildung 8). In diesem Fenster wird die aktuelle Bildgröße angezeigt, auf die der Naive Bayes Klassifizierer das Bild vor dem Detektionsvorgang skaliert (siehe Abschnitt 3.3.4). Das Bild wird unter Beibehaltung der Proportionen so skaliert, daß die kürzere Seite des Bildes dem aktuellen Wert entspricht. Durch Setzen eines Wertes kann die Bildgröße, die bei der Detektion verwendet wird, neu eingestellt werden. Die Bildgröße ist auf den Wert 431 voreingestellt und beeinflusst in starkem Maße die Laufzeit des Detektionsalgorithmus. Beim Einstellen der Bildgröße ist darauf zu achten, daß der Detektionsalgorithmus Gesichter nur bis zu einer Größe von 48 mal 56 Pixeln erkennen kann. Kleinere Bildausschnitte können nicht klassifiziert werden.

Abbildung 8

2.2.7. Menüpunkt Analyse | Sliding Window Schrittweite...

Der Befehl „Sliding Window Schrittweite...“ ruft das folgende Dialogfenster (siehe Abbildung 9) auf. In diesem Fenster wird die aktuelle Schrittweite des Sliding Window angezeigt, mit der der Naive Bayes Klassifizierer das Bild auf verschiedenen Skalierungsstufen nach Gesichtern abtastet (siehe Abschnitt 3.3.1). Durch Setzen eines Wertes kann die Schrittweite, die bei der Detektion verwendet wird, neu eingestellt werden. Der Wert für die Sliding Window Schrittweite ist auf 6 Pixel voreingestellt. Die Sliding Window Schrittweite sollte so gering wie möglich gewählt werden. Bei der Einstellung sollten allerdings die Hardwarevoraussetzungen berücksichigt werden, da eine Halbierung der Schrittweite eine Vervierfachung der Rechenzeit bewirkt.

Abbildung 9

2.2.8. Menüpunkt Analyse | Mehrfachdetektion anzeigen

Mit dem Menüpunkt „Mehrfachdetektion anzeigen“ können Mehrfachdetektionen ein- und ausgeschaltet werden. Ein Häkchen vor dem Menüpunkt bedeutet, dass Mehrfachdetektion eingeschaltet ist. Ausgeschaltete Mehrfachdetektion verhindert das Auftreten von sich überlappenden Detektionsergebnissen. Abbildung 10 zeigt Mehrfachdetektionen, die bei eingeschaltetem Menüpunkt auftreten können.

Abbildung 10

2.3. Menü Training

Der Naive Bayes Klassifizierer ist ein trainingsbasierter Ansatz. Das Menü Training (siehe Abbildung 11) enthält alle Menüpunkte zum Laden, Erstellen und Löschen von Trainingsdaten. Bevor der Detektionsvorgang gestartet werden kann, muß sich die Anwendung in einem trainierten Zustand befinden. Dies kann entweder durch Training (siehe Abschnitt 2.3.1) oder das Laden einer bereits vorhandenen Trainingsdatei (siehe Abschnitt 2.3.2) erreicht werden.

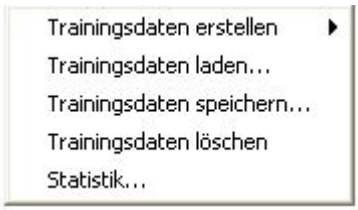


Abbildung 11

2.3.1. Trainingsdaten erstellen

Damit eine Gesichtserkennung durchgeführt werden kann, muß in allen 3 Kategorien (Frontal, Profil und Nicht-Gesicht) trainiert werden. Zum Erstellen von Trainingsdaten müssen große Mengen von Trainingsbildern geladen und verarbeitet werden (siehe Abschnitt 4). Dieser Vorgang ist sehr zeitaufwendig. Die Anzahl und die Qualität der Trainingsbilder haben entscheidenden Einfluß auf die Leistungsfähigkeit (Erkennungsrate) des Gesichtsdetektors.

Abbildung 12

2.3.1.1. Menüpunkt Training | Trainingsdaten erstellen | Frontal...

Der Befehl „Frontal...“ öffnet ein Dialogfenster zum Auswählen eines Verzeichnisses mit Trainingsbildern für das Frontaltraining. Jedes Bild muss einer Größe von 48x56 Pixel entsprechen. Der Verlauf wird über einen Fortschrittsbalken angezeigt. Abbildung 13 zeigt einige Beispielbilder.

Abbildung 13

2.3.1.2. Menüpunkt Training | Trainingsdaten erstellen | Profil...

Der Befehl „Profil...“ öffnet ein Dialogfenster zum Auswählen eines Verzeichnisses mit Trainingsbildern für das Profiltraining. Jedes Bild muss einer Größe von 64x64 Pixel entsprechen. Der Verlauf wird über einen Fortschrittsbalken angezeigt. Abbildung 14 zeigt einige Beispielbilder.

Abbildung 14

2.3.1.3. Menüpunkt Training | Trainingsdaten erstellen | Nicht-Gesicht...

Der Befehl „Nicht-Gesicht...“ öffnet ein Dialogfenster zum Auswählen eines Verzeichnisses mit Trainingsbildern, die kein Gesicht enthalten. Trainingsbilder der Klasse Nicht-Gesicht müssen einer Größe von 64x64 Pixel entsprechen. Größere Bilder werden entsprechend zerlegt. Die Schrittweite bei der Zerlegung beträgt 32 Pixel. Der Verlauf wird über einen Fortschrittsbalken angezeigt.

2.3.1.4. Menüpunkt Training | Trainingsdaten erstellen | Markierter Bereich...

Um die Detektionsergebnisse zu verbessern, können mit dem Befehl „Markierter Bereich...“ falsch oder nicht detektierte Gesichter eines Bildes dem Detektionsalgorithmus zum Training übergeben werden (Bootstrapping). Ein Bildausschnitt läßt sich mit gedrückter linker Maustaste auswählen. Der ausgewählte Bereich wird durch ein blaues Rechteck hervorgehoben (siehe Abbildung 15).

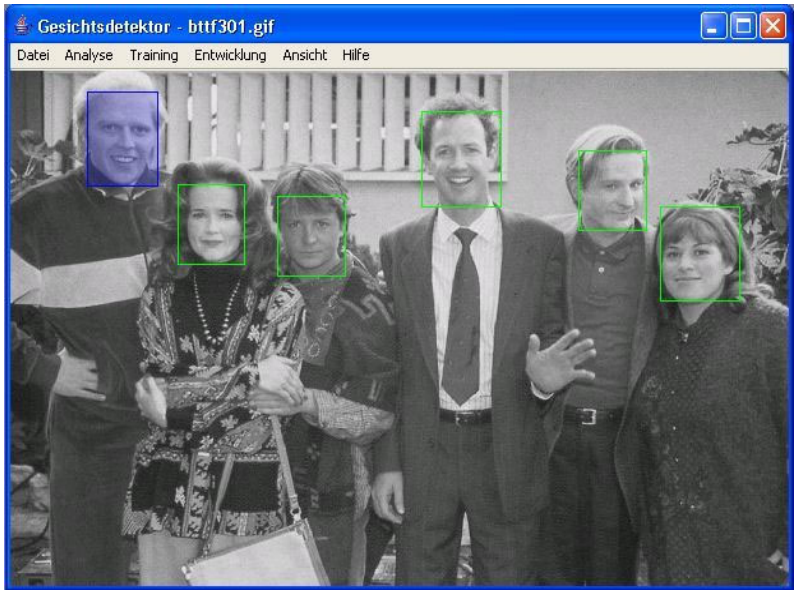


Abbildung 15

Der Befehl „Markierter Bereich...“ übergibt den aktuell selektierten Bildausschnitt dem Detektionsalgorithmus zum Training. Die Parameter, mit denen der Bildausschnitt übergeben wird, werden im folgenden Dialog (siehe Abbildung 16) festgelegt. Vor dem Training wird der Bildausschnitt automatisch auf die richtige Größe skaliert.

Abbildung 16

2.3.2. Menüpunkt Training | Trainingsdaten laden...

Der Befehl „Trainingsdaten laden...“ zeigt einen Datei-Öffnen-Dialog an. Eine ausgewählte, bereits existierende Trainingsdatei wird geladen. Das Laden der Trainingsdaten kann mehrere Minuten in Anspruch nehmen.

2.3.3. Menüpunkt Training | Trainingsdaten speichern...

Der Befehl „Trainingsdaten speichern...“ öffnet einen Datei-Speichern-Dialog, um den aktuellen Trainingszustand abzuspeichern. Das Speichern der Trainingsdaten kann mehrere Minuten in Anspruch nehmen.

2.3.4. Menüpunkt Training | Trainingsdaten löschen

Der aktuelle Trainingszustand wird gelöscht. Nach dem Befehl „Trainingsdaten löschen“ befindet sich der Detektor im untrainierten Zustand.

2.3.5. Menüpunkt Training | Statistik...

Der Menüpunkt Statistik ruft das folgende Dialogfenster auf (siehe Abbildung 17). Darin wird der aktuelle Trainingszustand angezeigt.

Abbildung 17

2.4. Menü Entwicklung

Die Menüpunkte des Menüs „Entwicklung“ visualisieren Zwischenergebnisse des Trainings- und Detektionsvorgangs. Sie dienten bei der Implementierung des Detektors zur visuellen Überprüfung von Zwischenergebnissen sowie zum Debuggen.

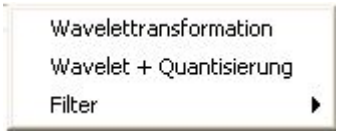


Abbildung 18

2.4.1. Menüpunkt Entwicklung | Wavelettransformation

Die im Detektionsalgorithmus verwendete Wavelettransformation (siehe Abschnitt 3.3.2) wird auf das atuelle Bild angewendet. Abbildung 19 zeigt das Ergebnis einer solchen Wavelettransformation.

Abbildung 19

2.4.2. Menüpunkt Entwicklung | Wavelet + Quantisierung

Der Menüpunkt „Wavelet + Quantisierung“ wendet die im Detektionsalgorithmus verwendete Wavelettransformation (siehe Abschnitt 3.3.2) gefolgt von einer Quantisierung (siehe Abschnitt 3.3.3) auf das aktuelle Bild an. Abbildung 20 zeigt das Ergebnis der Quantisierung. Diese Funktion diente während der Entwicklung zum Vergleich der Qualität von verschiedenen Quantisierungsverfahren.

Abbildung 20

2.4.3. Menü Entwicklung | Filter

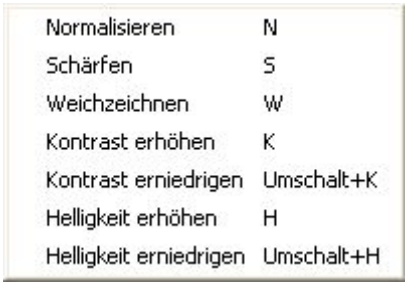


Abbildung 21

2.4.3.1. Menüpunkt Entwicklung | Filter | Normalisieren

Das aktuelle Bild wird normalisiert. Alternativ kann hierfür der Shortcut „n“ verwendet werden.

2.4.3.2. Menüpunkt Entwicklung | Filter | Schärfen

Der Filter „Schärfen“ wendet folgende Convolutionsmatrix auf das aktuelle Bild an.

0	-1	0
-1	5	-1
0	-1	0

Alternativ kann hierfür der Shortcut „s“ verwendet werden.

2.4.3.3. Menüpunkt Entwicklung | Filter | Weichzeichnen

Der Filter „Weichzeichnen“ wendet folgende Convolutionsmatrix auf das aktuelle Bild an.

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

Alternativ kann hierfür der Shortcut „w“ verwendet werden.

2.4.3.4. Menüpunkt Entwicklung | Filter | Kontrast erhöhen

Über den Menüpunkt „Kontrast erhöhen“ wird der Kontrast des aktuellen Bildes um 10% erhöht. Alternativ kann hierfür der Shortcut „k“ verwendet werden.

2.4.3.5. Menüpunkt Entwicklung | Filter | Kontrast erniedrigen

Über den Menüpunkt „Kontrast erniedrigen“ wird der Kontrast des aktuellen Bildes um 10% verringert. Alternativ kann hierfür der Shortcut „Umschalt+k“ verwendet werden.

2.4.3.6. Menüpunkt Entwicklung | Filter | Helligkeit erhöhen

Über den Menüpunkt „Helligkeit erhöhen“ wird die Helligkeit des aktuellen Bildes um 10% erhöht. Alternativ kann hierfür der Shortcut „h“ verwendet werden.

2.4.3.7. Menüpunkt Entwicklung | Filter | Helligkeit erniedrigen

Über den Menüpunkt „Helligkeit erniedrigen“ wird der Kontrast des aktuellen Bildes um 10% verringert. Alternativ kann hierfür der Shortcut „Umschalt+h“ verwendet werden.

2.5. Menü Ansicht

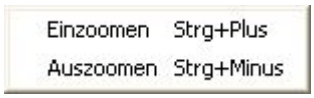


Abbildung 22

2.5.1. Menüpunkt Ansicht | Einzoomen

Mit dem Befehl „Einzoomen“ wird das aktuelle Bild 10% größer dargestellt. Das Fenster der Applikation wird entsprechend angepaßt.

2.5.2. Menüpunkt Ansicht | Auszoomen

Mit dem Befehl „Auszoomen“ wird das aktuelle Bild 10% kleiner dargestellt. Das Fenster der Applikation wird entsprechend angepaßt.

2.6. Menü Hilfe

2.6.1. Menüpunkt Hilfe | Info

Der Menüpunkt „Info“ ruft das folgende Dialogfenster auf (siehe Abbildung 23).

Abbildung 23

3. Technische Umsetzung

Das Erkennen von Gesichtern in Bildern stellt selbst heute noch eines der großen Probleme der Informatik dar. In [2] wurde versucht, einen Überblick über die bisher bekannten Verfahren zu schaffen, die sich mit dieser Problematik befassen. Dieses Paper stellte für uns gleichzeitig den Einstiegspunkt in dieses Thema dar.

Im Großen und Ganzen verfolgen alle Verfahren einen der beiden folgenden Ansätze:

- Beim Template Matching wird versucht, dem Computer zu „erklären", wie ein Gesicht aussieht, d.h. aus welchen Teilen es besteht und wie diese Teile geometrisch zueinander angeordnet sind. Verfahren, die auf dem Template Matching basieren, scheitern jedoch an den vielfältigen Variationen und Verfremdungen, die bei Gesichtern in Bildern möglich sind (vgl. Kapitel 1). Daher sind diese Verfahren die schwächsten unter den in [2] vorgestellten.
- Appearance-Based Methods verfolgen eine andere Strategie. Der Mensch trainiert den Computer, indem er ihm Beispiele für Gesichter und Beispiele für keine Gesichter zeigt. Um dies zu realisieren, bedienen sich die entsprechenden Verfahren Techniken wie z.B. der statistischen Analysis oder dem maschinellen Lernen.

Obwohl keines der Verfahren eine 100%ige Erkennungsrate liefert, sind die Appearance-Based Methoden die vielversprechendsten. Sie liefern Erkennungsraten von bis zu 96%. Die meisten dieser Verfahren würden jedoch für uns eine tiefere Einarbeitung in spezielle Gebiete der Informatik oder der Mathematik erfordern, wie z.B. Verfahren die auf neuronalen Netzen oder auf Eigenvektoren aufbauen. Im Hinblick auf den zeitlichen Rahmen des Fortgeschrittenen Praktikums haben wir uns deshalb für ein Verfahren entschieden, welches auf einer vergleichsweise einfachen Überlegung beruht: Dem Naiven Bayes Klassifizierer, wie er auch in diversen Spam-Filtern zum Einsatz kommt (siehe [3]).

3.1. Überblick

Aufgrund des hohen zu erwartenden Rechenaufwands beim Detektionsvorgang stand von Anfang an C++ als Programmiersprache der Wahl fest. Implementiert wurde der Detektor unter MS Windows mit MS Visual C++ 6.0. Wie in Kapitel 1 erwähnt, entstand der Detektor im Kontext eines Projekts zur Filmanalyse. Daher war eine weitere Vorgabe für den Detektor, daß die Plattformunabhängigkeit gewährleistet werden sollte. Dies wurde dadurch erreicht, indem strikt auf den Gebrauch von plattformspezifischen Bibliotheken wie z.B. der MFC (Microsoft Foundation Classes) verzichtet wurde. Stattdessen wurde auf die STL (Standard Template Library) zurückgegriffen, welche auf allen Systemen verfügbar ist.

Das Projekt wurde in zwei Teile aufgeteilt. Der Detektor wurde als Bibliothek implementiert, d.h. unter MS Windows als DLL (Dynamic Link Library). Diese DLL wird von einer Applikation aufgerufen, welche als Testumgebung für den Detektor fungiert. Die Testumgebung ermöglicht einen einfachen Zugriff auf alle Funktionen der DLL (siehe Kapitel 2).

Bei der Programmierung von grafischen Anwendungen ist es nicht möglich, auf plattformspezifische Bibliotheken zu verzichten. Daher haben wir uns bei der Implementierung der Testumgebung für Java entschieden. Dies hat neben der Plattformunabhängigkeit noch einen anderen Vorteil. Java erlaubt eine sehr schnelle und einfache Entwicklung von grafischen Anwendungen. Damit konnte die Anwendung ohne großen Aufwand stetig um neue Funktionen des Detektors erweitert werden.

Die Anbindung der DLL an die Java-Anwendung erfolgt per JNI (Java Native Interface). Folgende Abbildung zeigt die Aufteilung des Projekts in der Übersicht.

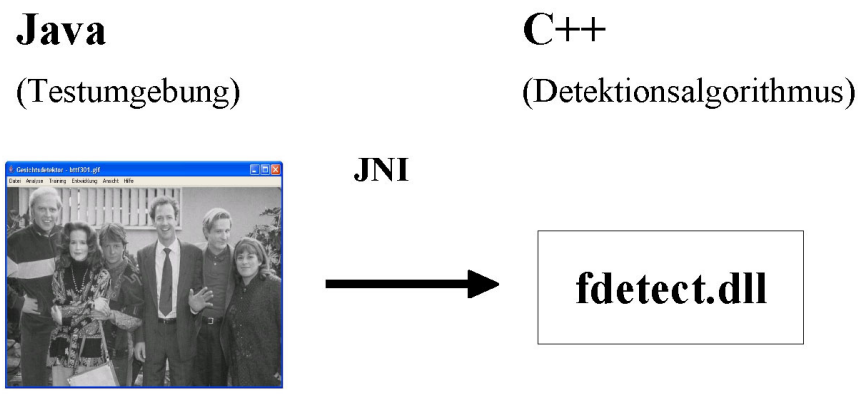


Abbildung 24

3.2. Testumgebung

Die Testumgebung wurde unter Verwendung des Java 2 SDK in der Version 1.4.2 entwickelt. Als Entwicklungsumgebung diente Eclipse in der Version 2.1. Die grafische Oberfläche der Applikation wurde in Swing realisiert. Ein großer Vorteil von Java sind die umfangreichen Möglichkeiten zum Umgang mit Bildern, die das Image I/O Framework bietet. Es werden die Bildformate bmp, jpeg, jpeg 2000, png, pnm, raw und tiff unterstützt. Die Anbindung der DLL an die Java-Applikation erfolgt mit Hilfe von Shared Stubs. Ein Shared Stub ist eine JNI (Java Native Interface) Methode, mit der man unterschiedliche C-Funktionen einer Bibliothek aufrufen kann. Das Package, welches wir verwenden, wurde von Sheng Liang im Rahmen der Präsentation "Java™ Native Interface Technology Programming" (siehe [4]) erstellt.

3.3. Detektor

Der Detektionsalgorithmus beruht auf der Doktorarbeit von Henry Schneiderman [5]. Wir weichen aber mit unserem Detektor an einigen Stellen vom Original ab. Zwar benutzen wir die gleiche Waveletttransformation, jedoch quantisieren wir die Waveletkoeffizienten auf nur drei Werte. Die Methode der Quantisierung wird in der Doktorarbeit nicht näher genannt. Daher haben wir mit mehreren Quantisierungsmethoden getestet. Ebenso wie Schneiderman erfassen auch wir mit 17 Attributen die Merkmale eines Bildes. Allerdings mußten wir einige Attribute mangels genauerer Beschreibung selbst definieren. Für eine detaillierte Beschreibung des Verfahrens verweisen wir auf [5] und [6].

Kern des Detektors ist der Klassifizierer, welcher auf der Bayes Formel basiert:

(1)
$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

Dabei bedeutet P(A|B) die Wahrscheinlichkeit, daß Ereignis A eintritt, wenn bereits B eingetreten ist. Diesen Zusammenhang kann man nutzen, um einen trainingsbasierten Klassifizierer herzuleiten:

(2)
$$\prod_{k}^{17} \prod_{x,y} \frac{P_k(attribute_k(x,y) \mid object)}{P_k(attribute_k(x,y) \mid non-object)} > \lambda$$

object ist in unserem Fall ein Gesicht, non-objects werden Bilder genannt, die kein Gesicht darstellen. Die Wahrscheinlichkeiten P_k werden über das Training in Form von Histogrammen aufgebaut

und sind daher bekannt. Wenn das Produkt dieser Wahrscheinlichkeiten größer als der Schwellenwert λ ist, dann liegt ein Gesicht vor. Die ausführliche Herleitung der Formel erfolgt in Kapitel 4 in [5].

Für den Einsatz im Computer eignet sich die Formel in dieser Form nicht, da das Produkt u.U. sehr klein wird und der Prozessor an die Grenzen seiner Fließkommaarithmetik stößt. Aus diesem Grund arbeitet man mit dem Logarithmus:

$$\sum_{k}^{17} \sum_{x,y} \log \frac{P_k(attribute_k(x,y)|object)}{P_k(attribute_k(x,y)|non-object)} > \lambda \tag{3}$$

3.3.1. Der Detektions-Algorithmus

Im Folgenden wird der grundlegende Ablauf des Detektions-Algorithmuses skizziert. Der Einfachheit halber wird hier auf einige Details verzichtet. Hier soll vorerst nur ein grober Überblick über den Algorithmus und das Zusammenspiel der verschiedenen Klassen dabei aufgezeigt werden. Für weitere Details verweisen wir auf die Beschreibungen der einzelnen Klassen in den folgenden Abschnitten sowie auf den Quellcode.

Es wird nun anhand der Detektion von frontalen Gesichtern gezeigt, wie der Detektor arbeitet. Normalerweise erfolgt während eines Detektionsvorgangs die Suche nach allen drei Arten von Gesichtern. Dies würde jedoch das Verständnis an dieser Stelle nur unnötig verkomplizieren. Zur Veranschaulichung des Zusammenspiels der verschiedenen Klassen wird der Detektionsvorgang zusätzlich in Form eines Sequenzdiagrammes dargestellt (siehe Abbildung 25). Sich wiederholende Vorgänge sind im Diagramm mit einem * gekennzeichnet.

1. Die Testumgebung übergibt dem Detektor durch Aufruf der Schnittstellenfunktion getFaces() das Bild, in dem nach Gesichtern gesucht werden soll. Bevor das Bild an den Detektor weitergegeben wird, erfolgt eine Normalisierung des Bildes mit Hilfe der Klasse Filter. Dieser Schritt wurde im Diagramm aus Platzgründen weggelassen.
2. Das normalisierte Bild wird nun mit der Funktion detectFaces() der Klasse Detector übergeben. Als erstes skaliert der Detektor das Bild mit der Funktion interpolate() der Klasse Scale auf die eingestellte Anfangsgröße. Dann fährt der Detektor mit einem 48x56 Pixel großen Fensters („Sliding Window“) mit der eingestellten Schrittweite zeilenweise von links nach rechts über das Bild. Für jede Position des Sliding Window wird sein Inhalt daraufhin untersucht, ob sich das Sliding Window über einem Gesicht befindet.
3. Dazu wird die Funktion detectFrontFace() der Klasse Detector aufgerufen. Dort wird das Bild einer Wavelettransformation mit anschließender Quantisierung der Waveletkoeffizienten unterzogen. Aus den quantisierten Waveletkoeffizienten werden nun die Werte („Patterns“) für alle 17 Attribute berechnet. Für die Details der Patternberechnung siehe Abschnitt 3.3.8.
4. Bei jedem errechneten Pattern wird dessen Wahrscheinlichkeit für sein Vorkommen in einem Gesicht zur Summe aus Ungleichung (3) hinzuaddiert. Dazu wird die Funktion addProbabilityForFront() der Klasse BayesClassifier aufgerufen. In dieser Funktion wird der Bruch der Wahrscheinlichkeiten für Gesicht und für non-object gebildet und zur Summe addiert. Die Wahrscheinlichkeiten sind in Form von Histogrammen in der Klasse FrontFaceTrainingData gespeichert. Zum Auslesen der Wahrscheinlichkeiten wird die Funktion getProbability() der Klasse FrontFaceTrainingData aufgerufen.
5. Wenn für alle 17 Attribute alle Patterns berechnet und die zugehörigen Wahrscheinlichkeiten aufsummiert wurden, wird mit der Funktion isFrontFace() der Klassifizierer gefragt, ob es sich

bei dem aktuellen Inhalt des Sliding Window um ein Gesicht handelt. Wenn die Summe der Wahrscheinlichkeiten größer ist als der eingestellte Schwellenwert, dann klassifiziert die Klasse BayesClassifier den Inhalt des Sliding Window als Gesicht. In diesem Fall werden aus der aktuellen Position des Sliding Window im skalierten Bild die Koordinaten des Gesichts im Ursprungsbild errechnet. Diese Koordinaten werden in Form eines Rechtecks in einer Liste gespeichert.

6. Wenn das Sliding Window in der unteren rechten Ecke des Bildes angekommen ist, wird das Bild um den Faktor $\sqrt[4]{2}$ verkleinert und das Sliding Window wird wieder in die linke obere Ecke gesetzt. Das Sliding Window fährt wieder das Bild ab und es wiederholen sich die Schritte 3 bis 5. Dies wird solange gemacht, bis das Bild auf eine Größe skaliert wurde, so daß das Sliding Window nicht mehr in das skalierte Bild reinpaßt. Damit ist der eigentliche Detektionsvorgang abgeschlossen.
7. In der Liste der gefundenen Gesichter können sich Mehrfachdetektionen ein und desselben Gesichtes befinden. Wenn die Anzeige der Mehrfachdetektionen in der Testumgebung ausgeschaltet ist, wird jetzt eine Strategie zum Raussortieren der Mehrfachdetektionen angewendet. Hierfür wird die Funktion discardMultipleDetections() der Klasse Detector aufgerufen.
8. Abschließend wird die fertige Liste der gefundenen Gesichter mit der Funktion getFrontFaces() abgerufen und an die Testumgebung weitergereicht.

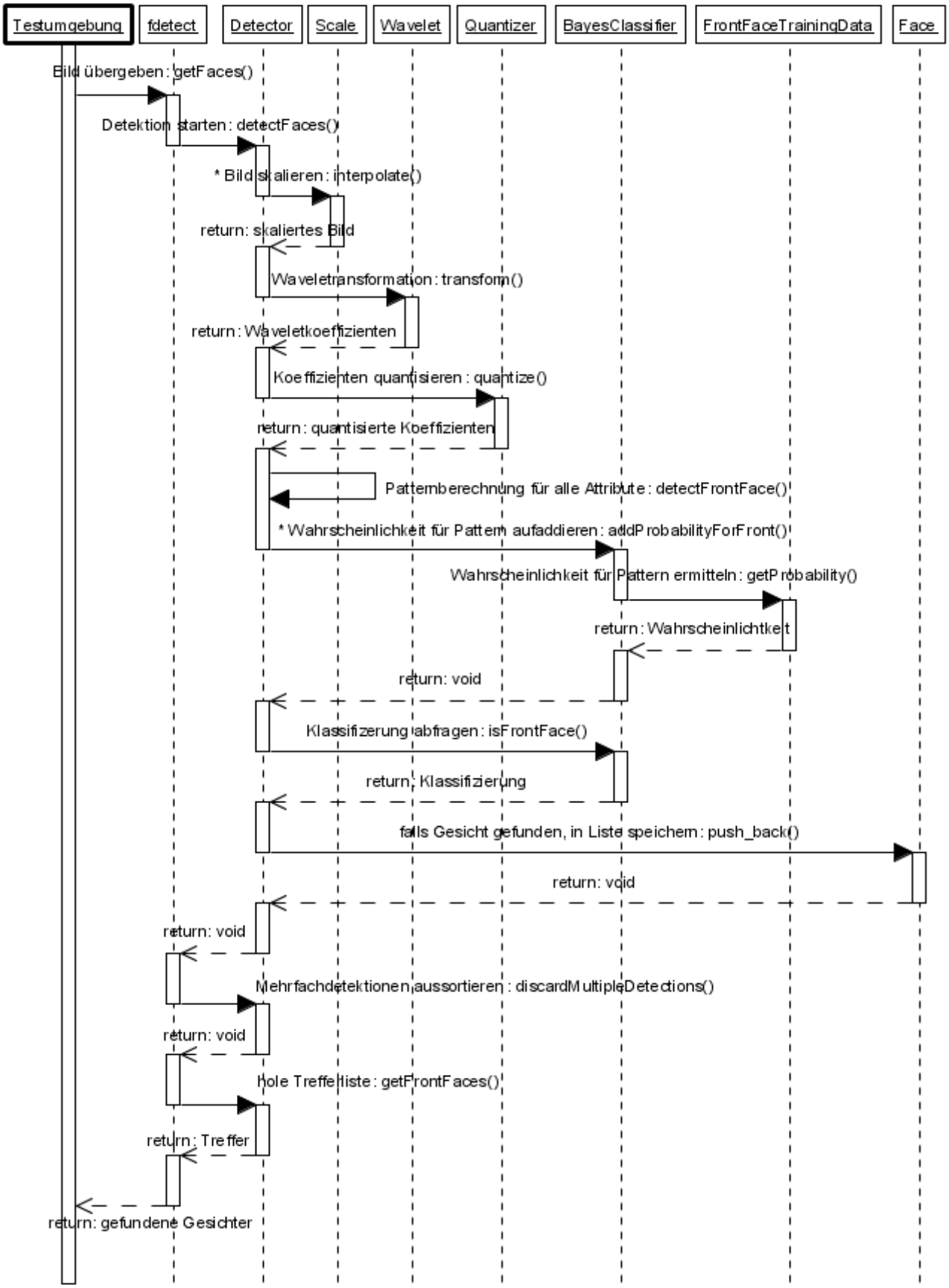


Abbildung 25

3.3.2. Die Klasse Wavelet

In dieser Klasse wird die Wavelettransformation implementiert. Bei der Wavelettransformation handelt es sich um eine Villasenor 5/3 Filterbank, auch bekannt als „Villasenor4“ Wavelettransformation (siehe Abschnitt 3.5.1 in [5] für Details). Wie der Name schon andeutet, besteht die Skalierungsfunktion aus fünf Tiefpasskoeffizienten und die Waveletfunktion aus drei Hochpasskoeffizienten. Im Endeffekt erfolgt bei der Transformation nichts anderes als eine Faltung der Pixel eines Bildes mit diesen beiden Funktionen. Dabei werden durch die Skalierungsfunktion jeweils fünf nebeneinander bzw. untereinander stehende Pixel gefaltet und durch die Waveletfunktion jeweils drei. Bei der Behandlung der Ränder stehen zwei Verfahren zur Auswahl. Beim „Circular Padding“ werden die fehlenden Pixel am rechten bzw. unteren Rand durch Pixel vom linken bzw. oberen Rand der gleichen Zeile bzw. Spalte ersetzt. Beim „Mirror-Padding“ wird dagegen das jeweils letzte Pixel des rechten bzw. unteren Rands einfach wiederholt. Für den Detektor wird Mirror-Padding verwendet.

Obwohl für den Detektionsalgorithmus nur eine Transformationstiefe von drei benötigt wird, kann man die Anzahl der Transformationsschritte frei wählen. Als weitere Verallgemeinerung können Bilder mit geraden und ungeraden Seitenlängen transformiert werden. Da aber sowohl beim Training als auch bei der Detektion nur (Teil-)Bilder der Dimension 48x56 Pixel und 64x64 Pixel vorkommen, können die Seiten bis zu drei mal ohne Rest halbiert werden, so daß auch diese Fähigkeit im Detektionsalgorithmus nicht zum Einsatz kommt.

Für weitere Details der Transformation verweisen wir an dieser Stelle auf den reichlich kommentierten Quellcode sowie auf Abschnitt 3.3.8.

3.3.3. Die Klasse Quantizer

Mit dieser Klasse werden die Waveletkoeffizienten auf ganzzahlige Werte von 0 bis N quantisiert. Um die bestmögliche Quantisierung zu erreichen, wird jedes Subband (LL, LH, HL, etc.) separat quantisiert. Dazu muß die Tiefe der Wavelettransformation berücksichtigt werden, ebenso wie das Verhalten der Klasse Wavelet bei Bildern mit ungeraden Seitenlängen.

Die Anzahl N der Quantisierungsstufen kann zwischen 1 und 256 gewählt werden. Quantisierungen auf mehr als drei Werte erfolgen per gleichförmiger Quantisierung („Uniform Quantization“). Im Detektionsalgorithmus werden die Waveletkoeffizienten stets auf drei Werte quantisiert. Es gehen also sehr viel Informationen verloren. Da sich Ausreißer unter den Waveletkoeffizienten bei so wenig Quantisierungsstufen verzerrend auf die Verteilung der Koeffizienten zu den Quantisierungsstufen auswirken, wird bei der Quantisierung auf drei Werte der „Median Cut“ Algorithmus vorgezogen. Diese Art der Quantisierung ist immun gegen Ausreißer, da jeder Quantisierungsstufe gleich viele Waveletkoeffizienten zugeordnet werden. Im Vergleich zur gleichförmigen Quantisierung liefert diese Art der Quantisierung deutlich bessere Detektionsraten. Auch mit bloßem Auge erkennt man bei einem so quantisiertem Trainingsbild wesentlich mehr Details als nur zwei dunkle Punkte für die Augen (siehe Abschnitt 2.4.2).

3.3.4. Die Klasse Scale

Diese Klasse dient zum Verändern der Größe von Bildern. Bei der Implementierung wurde viel Wert auf Geschwindigkeit gelegt. Daher wurde zur Skalierung die bilineare Interpolation ausgewählt. Sie hat im Vergleich zu anderen Verfahren wie z.B. der bikubischen Interpolation den Vorteil einer geringeren Komplexität. Außerdem wird beim Herunterskalieren das Bild nicht so übermäßig weichgezeichnet, wie es bei der bikubischen Variante der Fall ist. Auch dies paßte besser zu den Anforderungen des Detektors.

3.3.5. Die Klasse BayesClassifier

Der Bayes Klassifizierer ist letztendlich nichts anderes als Ungleichung (3). Die Implementierung dieser Summe besteht aus einer Zeile. Viel mehr als das Aufsummieren von Wahrscheinlichkeiten geschieht in dieser Klasse auch nicht. Am Ende wird lediglich verglichen, ob die Summe größer ist als ein Schwellenwert. Falls dies zutrifft, ist ein Gesicht erkannt worden.

3.3.6. Die Klasse Detector

In dieser Klasse wird der Detektionsalgorithmus implementiert. Nachfolgend wird schematisch der Detektionsvorgang beschrieben, für Details siehe Funktion detectFaces().

Als erstes wird das zu analysierende Bild auf eine bestimmte Anfangsgröße skaliert. Diese Anfangsgröße kann in der Testumgebung eingestellt werden (siehe Abschnitt 2.2.6). Dabei bezieht sich der Wert für die Anfangsgröße auf die kleinere der beiden Seiten des Bildes. Die andere Seitenlänge wird unter Beibehaltung des ursprünglichen Seitenverhältnisses errechnet.

Das so skalierte Bild wird mit Hilfe eines Fensters, des sogenannten „Sliding Window“, ausgehend von der oberen linken Ecke zeilenweise nach unten rechts durchlaufen. Die Schrittweite des Sliding Window kann über die Testumgebung eingestellt werden (siehe Abschnitt 2.2.7). Der jeweilige Inhalt des Sliding Window wird nun einer dreistufigen Wavelettransformation unterzogen und die Waveletkoeffizienten werden auf drei Werte quantisiert (siehe Funktionen detectFrontFace() bzw. detectLeftProfileFace()). Aus den quantisierten Waveletkoeffizienten werden nun für 17 Attribute eine Reihe von Werten („Patterns“) berechnet. Für jedes Pattern wird seine Wahrscheinlichkeit, daß es in dem jeweiligen Attribut vorkommt, aus der Tabelle mit den Trainingsdaten ermittelt. All diese Wahrscheinlichkeiten werden aufaddiert. Wenn die Summe der Wahrscheinlichkeiten größer ist als ein Schwellenwert (siehe Abschnitt 2.2.4), dann wurde in dem Sliding Window ein Gesicht erkannt. In diesem Falle wird aus der aktuellen Position des Sliding Window die Position und Größe des gefundenen Gesichts im Ursprungsbild berechnet. Für die spätere Markierung des Gesichts innerhalb des Bildes werden die Koordinaten der linken oberen und der rechten unteren Ecke eines Rechtecks in einer Liste gespeichert.

Diese Erkennung wird zuerst für frontale Gesichter durchgeführt. Wenn das Sliding Window in der rechten unteren Ecke des Bildes angekommen ist, wird das gleiche für Profile wiederholt. Je nachdem, ob eine Erkennung von einem frontalen Gesicht oder nach einem Profil stattfindet, hat das Sliding Window eine Größe von 48x56 Pixel oder 64x64 Pixel. Für die Erkennung von rechten Profilen wird der Inhalt des Sliding Window vor der Wavelettransformation horizontal gespiegelt. Sind beide Durchläufe des Sliding Window beendet, wird das Bild um den Faktor $\sqrt[4]{2}$ verkleinert und der Vorgang wird wieder in der linken oberen Ecke von neuem begonnen. Dies wird solange wiederholt, bis das skalierte Bild so klein ist, daß keines der beiden Sliding Windows mehr in das Bild paßt.

Neben dem Detektionsalgorithmus enthält die Klasse noch eine Strategie zum Aussortieren von Mehrfacherkennungen (siehe Funktion discardMultipleDetections()). Da sich das Sliding Window nur in Schritten von wenigen Pixeln über das Bild bewegt, werden immer ganze „Cluster“ von Detektionen über demselben Gesicht gemeldet. Diese führen dazu, daß mehrere überlappenden Rechtecke über einem Gesicht gezeichnet werden. Wenn man nur an einem einzigen Rechteck interessiert ist (z.B. für den Abgleich mit Groundtruth-Daten eines Testsets), dann kann man dies in der Testumgebung einstellen (siehe Abschnitt 2.2.8). Das Aussortieren von mehrfachen Erkennungen verläuft nach folgendem Schema:

Zuerst wird aus der Liste mit allen Erkennungen (in Form von Rechtecken) diejenige mit der höchsten Detektionswahrscheinlichkeit gesucht und in eine zweite Liste verschoben. Alle Erkennungen, die sich mit dieser höchsten Erkennung überlappen, werden aus der Usrprungsliste entfernt. Dieser Vorgang wird solange wiederholt, bis die ursprüngliche Liste der Erkennungen leer ist.

3.3.7. Die Klasse Filter

Diese Klasse bietet Filter zum Weichzeichnen, Schärfen und Normalisieren von Bildern an. Außerdem können der Kontrast und die Helligkeit eines Bildes erhöht und verringert werden. Durch Kombination der verschiedenen Filter untereinander werden beim Training des Klassifizierers zu jedem Trainingsbild automatisch 77 Variationen erzeugt (siehe Funktion setNextTrainingVariation()). Dies geschieht mit Hilfe eines einfachen Iterators über alle sinnvollen Kombinationen der Filter. Zusammen mit der horizontalen Spiegelung von Trainingsbildern für frontale Gesichter und non-objects, die in den Schnittstellen der Datei fdetect.cpp erfolgt, erhält man insgesamt 154 Variationen pro Bild.

Die einzelnen Filter können auch auf ein in der Testumgebung geladenes Bild angewendet werden (siehe Abschnitt 2.4.4). Dies diente – wie der Hauptmenüpunkt „Entwicklung“ schon sagt – bei der Entwicklung der Filter zur Endkontrolle und zur Auswahl von sinnvollen Filter-Kombinationen für den Iterator.

3.3.8. Die TrainingData Klassen

In den Klassen FrontFaceTrainingData, ProfileFaceTrainingData und NonObjectTrainingData wird die Erstellung, Speicherung und Bereitstellung der Trainingsdaten für frontale Gesichter, Profile und non-objects gehandhabt. Die Trainingsdaten werden in Form von mehrdimensionalen Tabellen im Speicher aufgebaut. Beim Training werden für jedes Trainingsbild die gleichen 17 Attribute berechnet wie beim Detektionsvorgang. Hierfür wird jedes Trainingsbild vorher einer Wavelettransformation mit anschließender Quantisierung der Koeffizienten unterzogen.

Jedes Attribut, in [5] auch „Local Operator“ genannt, setzt sich aus jeweils acht Waveletkoeffizienten zusammen. Je nach dem, aus welchen Bändern der Wavelettransformation die Koeffizienten stammen, unterscheidet man die Lokalen Operatoren in Intra-Orientation, Inter-Orientation, Inter-Frequency und Inter-Orientation/Inter-Frequency Attribute. In Abbildung 26 ist die Aufteilung einer zweistufigen Wavelettransformation in die einzelnen Bänder zu sehen. Diese Art der Wavelettransformation bezeichnet man als „non-standard decomposition“. Die Abkürzung LH bedeutet, daß die Koeffizienten dieses Bandes während der Wavelettransformation zuerst mit dem Hochpassfilter (H) und danach mit dem Tiefpassfilter (L) gefaltet wurden. Die restlichen Abkürzungen ergeben sich analog. Zu erwähnen bleibt, daß die Koeffizienten aus den HH-Bändern nicht verwendet werden.

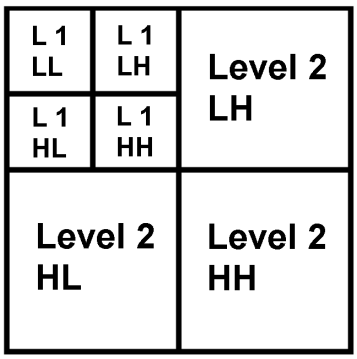


Abbildung 26

Bei den Intra-Subband Attributen werden alle acht Koeffizienten aus jeweils einem der folgenden Bänder genommen: Level 1 LL, HL und LH, Level 2 HL und LH sowie Level 3 HL und LH. Diese Attribute stellen die ersten sieben Attribute dar. Für eine bestimmte Position (x,y) innerhalb eines Bandes werden die Koeffizienten c(x,y) wie folgt gewichtet:

$$\begin{aligned} \text{pattern} = \; & c(x,y)*3^0 + c(x+1,y)*3^1 + c(x+2,y)*3^2 + \\ & c(x,y+1)*3^3 + c(x+1,y+1)*3^4 + c(x+2,y+1)*3^5 + \\ & c(x,y+2)*3^6 + c(x+2,y+2)*3^7 \end{aligned}$$

Das heißt, die räumliche Anordnung der acht Koeffizienten ergibt folgendes Muster:

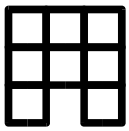


Abbildung 27 zeigt das Attribut Level 3 LH.

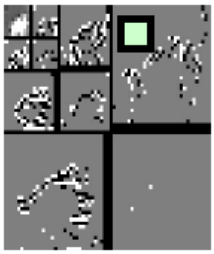


Abbildung 27

In den HL Bändern sind die horizontalen Merkmale eines Bildes ausgeprägt und in den LH Bändern die vertikalen. Daher werden Lokale Operatoren, die Koeffizienten aus beiden Bändern zusammenfassen, als Inter-Orientation Attribute bezeichnet. Von Ihnen gibt es nur drei, nämlich eine HL-LH-Kombination in jedem der drei Transformations-Level. Dies sind die Attribute acht bis zehn. Auch bei den Inter-Orientation Attributen werden alle Attribute auf die gleiche Art gebildet. Daher wird die Gewichtung der Koeffizienten exemplarisch nur für das Attribut Level 3 HL/LH gezeigt:

$$\begin{aligned} \text{pattern} = \; & L3\_HL(x,y)*3^0 + L3\_HL(x+1,y)*3^1 + \\ & L3\_HL(x,y+1)3^2 + L3\_HL(x+1,y+1)*3^3 + \\ & L3\_LH(x,y)*3^4 + L3\_LH(x+1,y)*3^5 + \\ & L3\_LH(x,y+1)3^6 + L3\_LH(x+1,y+1)*3^7 + \end{aligned}$$

Das heißt, es werden aus dem HL und dem LH Band jeweils vier Koeffizienten zu nebenstehendem Muster gruppiert:

Abbildung 28 zeigt das Attribut Level 3 HL/LH.

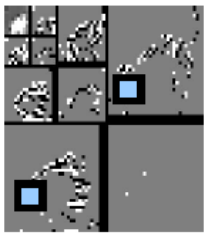


Abbildung 28

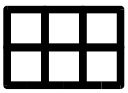
Jedes Level einer Wavelettransformation stellt eine andere Frequenz dar. Die Inter-Frequency Attribute sind daher die Kombinationen aus Level 1 LL/HL, Level 1/2 HL, Level 2/3 HL, Level 1 LL/LH, Level 1/2 LH und Level 2/3 LH. Dies sind die Attribute elf bis 16. Wegen der unterschiedlichen Größe der Bänder können nun nicht mehr alle Attribute auf die gleiche Weise zusammengesetzt werden. Die Koeffizienten der Attribute Level 1 LL/HL und Level 1 LL/LH werden noch wie die Koeffizienten der Inter-Orientation Attribute angeordnet. Die anderen Attribute müssen anders

gebildet werden. Die Anordnung der Koeffizienten der Attribute Level 1/2 HL und Level 2/3 HL ist gleich. Nachfolgend die Gewichtung der Koeffizienten für das Attribut Level 2/3 HL:

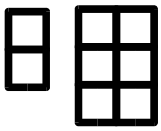
```
pattern =  L2_HL(x,y)*3^0    + L2_HL(x+1,y)*3^1      +
           L3_HL(x,y)*3^2    + L3_HL(x+1,y)*3^3      + L3_HL(x+2,y)*3^4 +
           L3_HL(x,y+1)*3^5  + L3_HL(x+1,y+1)*3^6    + L3_HL(x+2,y+1)*3^7
```

Das heißt, die Koeffizienten aus dem niedrigeren Level werden wie folgt gruppiert:

Die Koeffizienten aus dem höheren Level werden dagegen auf folgende Art zusammengesetzt:



Die Gewichtung der Koeffizienten in den Attributen Level 1/2 LH und Level 2/3 LH ist analog. Allerdings muß hierbei berücksichtigt werden, daß in den LH Bändern die vertikalen Merkmale eines Bildes ausgeprägt sind. Die Anordnung der Koeffizienten ist daher wie folgt:



In Abbildung 29 ist das Attribut Level 2/3 LH zu sehen.

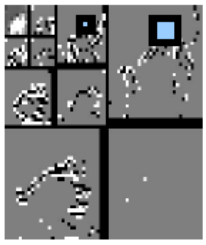


Abbildung 29

Das Inter-Orientation/Inter-Frequency Attribut ist das 17. und damit letzte Attribut. Es faßt jeweils zwei Koeffizienten aus den LH und HL Bändern der Level 2 und 3 zusammen (siehe Abbildung 30).

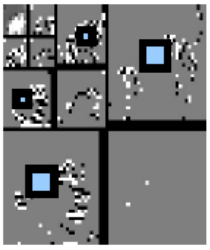


Abbildung 30

Die Gewichtung der Koeffizienten ist hier wie folgt:

```
pattern =  L2_HL(x,y)*3^0    +
           L2_LH(x,y)*3^1    +
           L3_HL(x,y)*3^2    + L3_HL(x+1,y)*3^3      +
           L3_HL(x,y+1)*3^4  +
           L3_LH(x,y)*3^5    + L3_LH(x+1,y)*3^6      +
           L3_LH(x,y+1)*3^7
```

Das heißt, wir ziehen aus dem Level 2 HL und LH Band jeweils nur einen Koeffizienten, und aus den anderen beiden jeweils drei, die auf folgende Art angeordnet sind:

Die Werte der Attribute werden berechnet, indem aus den acht quantisierten Waveletkoeffizienten eine Zahl gebildet wird. Dabei wird jeder Koeffizient gemäß seiner Stelligkeit mit einer Dreierpo-

tenz multipliziert. Insgesamt ergeben sich somit für jedes Attribut $3^8 = 6561$ mögliche Werte. Beim Training wird jedes Vorkommen eines Patterns gezählt (siehe Funktion incTableEntry()). Am Ende erhält man ein Histogramm mit den Häufigkeiten der verschiedenen Patterns. Dieses Histogramm wird dann vom Klassifizierer zu der Berechnung der Wahrscheinlichkeiten herangezogen (siehe Funktion getProbability()). Die Größe eines Histogramms für 64x64 Pixel große Trainingsbilder beträgt ca. 105 MB, für 48x56 Pixel große Trainingsbilder ca. 83 MB. Für das Training von Profilen und non-objects werden jeweils Bilder der Größe 64x64 Pixel verwendet, für frontale Gesichter Bilder der Größe 48x56 Pixel. Für alle Trainingsdaten zusammen fallen also insgesamt ca. 290 MB an.

3.3.9. Die Schnittstellen-Funktionen

Die Datei fdetect.cpp ist von zentraler Bedeutung. Sie stellt die Schnittstellen zum Zugriff auf die Bibliotheksfunktionen bereit. Die einzelnen Funktionen sind jedoch zu unterschiedlich, um hier auf alle einzugehen. Auch hier verweisen wir auf den reichlich kommentierten Quellcode.

4. Training

Da die vereinfachte Bayes-Gleichung (3) nur für genügend große Trainingsdaten gültig ist (siehe Herleitung in [3]), sind sehr viele Trainingsbilder zum Trainieren des Klassifizierers nötig. Schneiderman gibt in Kapitel 5 seiner Doktorarbeit [5] an, daß er für das Training frontaler Gesichter und Profile jeweils mindestens eine Million Trainingsbilder verwendet hat. Diese wurden aus jeweils ca. 2000 per Hand zugeschnittenen Gesichtern synthetisch erzeugt. Für non-objects hat Schneiderman eine ebenso große Zahl an Bildern erzeugt. Leider stellte uns Schneiderman seine Trainingsets auch nach Anfrage nicht zur Verfügung, so daß wir selbst genügend große Trainingsets erstellen mußten.

Durch das Ablichten der näheren Bekanntenkreise (Familie, WG-Mitbewohner, Kommilitonen, etc.) per Digitalkamera in verschiedenen Posen konnten wir alleine über 1000 frontale Gesichter gewinnen. Ca. 1500 weitere Gesichter haben wir durch Zurechtschneiden von Bildern aus dem Internet erhalten. Der Freeware Bild-Betrachter und -Konverter XnView [7] hat sich bei dieser Arbeit als äußerst hilfreich erwiesen. So sind viele Variationen ein und desselben Gesichtes entstanden, indem beim Ausschneiden der Gesichter sowohl das Seitenverhältnis als auch das Zentrum des Ausschnitts leicht variiert wurden. Durch das anschließende Skalieren auf die Trainingsbildgröße von 48x56 Pixel haben sich Gesichter von unterschiedlicher Breite und Höhe ergeben. Mit dem Ändern des Zentrums des Ausschnitts, wobei bewußt mal das linke oder das rechte Ohr, mal Hals oder Haaransatz weggelassen wurden, ergaben sich weitere Variationen.

Da zu jedem Trainingsbild von der DLL automatisch 154 Variationen erzeugt werden, konnten wir durch mehrfaches Antrainieren von verschiedenen Gruppen von Gesichtern die Millionenmarke überschreiten. Das Erreichen dieser Marke gestaltete sich für das Training von non-objects wesentlich einfacher. Die Applikation zerschneidet non-object-Trainingsbilder, die größer sind als 64x64 Pixel, automatisch in 64x64 Pixel große Teilbilder, die dann dem Trainer übergeben werden. Aus einem einzigen großen Foto entstehen somit schnell über Tausend Trainingsbilder. Da auch zu diesen Trainingsbildern automatisch 154 Variationen erzeugt werden, erhält man bereits mit wenigen hundert Fotos genügend große Trainingsbestände.

Leider garantiert eine große Anzahl Bilder allein noch nicht eine gute Trainingsdatei. Ein wahlloses Antrainieren von mehreren Millionen Bildern hat die Detektionsrate teilweise drastisch verschlechtert. So mußten wir, obwohl wir allein für non-objects ca. 18000 Bilder zur Verfügung stehen hatten, selektiver vorgehen. Diese große Sammlung an non-objects-Bildern setzte sich zum einen

zusammen aus hochauflösenden Fotosammlungen und zum anderen aus Clipartsammlungen und Bildern aus dem Internet Explorer Cache, welche schon auf Trainingsbildgröße von 64x64 Pixel vorskaliert wurden. Mit der Zeit hat sich herauskristallisiert, daß die zufälligen Inhalte des Internet Explorer Caches eigenartigerweise besonders gutes Material für das non-objects-Training lieferten. Aus dieser Erfahrung heraus haben wir auch die Clipartsammlungen auf Trainingsbildgröße runterskaliert. Desweiteren gelangten wir zur Erkenntnis, daß ein in etwa gleiches Verhältnis zwischen den Trainingsdaten für Gesichter und non-objects offenbar eine gute Erkennungsrate ergab. Da wir ansonsten keinerlei Zusammenhang zwischen der Art und Anzahl der Trainingsbilder und der Erkennungsrate feststellen konnten, können wir an dieser Stelle keine Strategie zum Erstellen von guten Trainingsdaten angeben. Es scheint wohl eine Wissenschaft für sich zu sein, besonders gute Trainingsdaten zu erstellen.

Eine relativ gute Trainingsdatei kam heraus, als versucht wurde, den Klassifizierer soweit anzutrainieren, bis ein bestimmtes „Problembild“ fehlerfrei erkannt wurde. Diese Trainingsdatei hat sich dann auch bei anderen Bildern als gut herausgestellt, sodaß wir sie auch für die Benchmarks verwendet haben. Das Verhältnis von frontalen Gesichtern zu non-objects in dieser Trainingsdatei beträgt ca. 1,1 Mio. zu 2 Mio. Bilder.

Mittlerweile sind wir der Auffassung, daß unsere Trainingsbilder für frontale Gesichter nicht besonders geschickt ausgeschnitten waren. Wir orientierten uns beim Erstellen der Gesichtssausschnitte an den Trainingsbildern, wie sie Schneiderman in seiner Doktorarbeit abgebildet hat. Wie in den Abbildungen 13 und 14 zu erkennen ist, enthalten unsere Trainingsbilder sehr viel, was nicht zum eigentlichen Gesicht gehört. Schneiderman ist sogar der Überzeugung, daß verschiedene Hintergründe antrainiert werden sollten (siehe [5], Seite 51). Dies mag auf jeden Fall für Gesichter im Profil zutreffen, weil dort der Klassifizierer aus den sich vom Hintergrund absetzenden Gesichtskonturen die Hauptmerkmale eines Profils extrahieren kann. Bei frontalen Gesichtern sind die Konturen jedoch nicht von großer Bedeutung – eher im Gegenteil. Insbesondere die Vielfalt der Frisuren macht es dem Klassifizierer nicht gerade einfach, übereinstimmende Merkmale zwischen verschiedenen Gesichtern zu bestimmen. Das Fatale daran ist zudem, daß für diese unwichtigen Informationen sehr viel von der ohnehin schon knappen Größe des 48x56 Pixel kleinen Trainingsbildes verschwendet wird. Daher sollten sich Trainingsbilder für frontale Gesichter allein auf die Hauptmerkmale eines Gesichtes beschränken, wie etwa Augen, Nase und Mund. Ganz nebenbei würden dann auch die markierenden Rechtecke des Detektors genau den Rechtecken aus den Groundtruth-Daten der bekannten Testsets [8] entsprechen.

5. Testergebnisse

Die Detektionsergebnisse sind sehr stark abhängig von dem verwendeten Trainingsmaterial. Die Anzahl der verwendeten non-objects sollte möglichst gering gehalten werden. Die Verwendung sehr vieler non-objects führte zu schlechten Trainingsdaten. Wir konnten leider keinen Zusammenhang zwischen den verwendeten Trainingsbildern und dem resultierenden Schwellenwert erkennen. Dadurch musste der Schwellenwert nach jeder Änderung der Trainingsdaten neu bestimmt werden. Der Detektionsvorgang erwies sich ebenfalls als sehr zeitaufwendig. Die benötigte Zeit für ein Bild hängt von den beiden folgenden Faktoren ab: Dem Wert für die Größe des Bildes bei der Detektion (d.h. auf welche Größe das Bild vor der Detektion skaliert wird) und dem Wert für die Sliding Window Schrittweite. Aus Zeitgründen haben wir uns auf die Erstellung von Trainingsdaten für frontale Gesichter beschränkt.

Nachfolgend möchten wir die Detektionergebnisse von unserem Algorithmus präsentieren und anhand von Standardtestsets mit den Ergebnissen von Henry Schneiderman vergleichen.

De facto Benchmarksets sind die Testbilder der CMU (siehe [8]). Das CMU Testset 1, zusammengestellt von Sung und Rowley, besteht aus 130 Bildern mit 507 frontalen Gesichtern. CMU Testset 2, zusammengestellt von Sung und Poggio, besteht aus 23 Bildern mit 157 Gesichtern. Testset 1 beinhaltet die Testbilder von Testset 2 (siehe Abbildung 31).

Abbildung 31

Bei der Analyse ergibt sich folgendes Problem: Wann ist ein Gesicht richtig erkannt? Die Detektionsergebnisse (siehe Abbildung 32) des Naiven Bayes Klassifizierers lassen sich nur schwer mit den Groundtruth-Daten der CMU Testsets vergleichen. Die Groundtruth-Daten der CMU Testsets enthalten folgende Daten: Position des linken Auges, Position des rechten Auges, Position der Nase, Position des linken Mundwinkels, Mundzentrum und die Position des rechten Mundwinkels.
Zur Überprüfung unserer Detektionsergebnisse berechnen wir nun aus den Groundtruth-Daten (Position des linken und rechten Auges und des Mundzentrums) ein Rechteck. Ein Gesicht gilt als richtig erkannt, wenn dieses Rechteck enthalten ist und die Fläche des gefundenen Rechtecks höchstens zwölfmal (Erfahrungswert) so groß ist.
Wir haben mehrere Analysen der Testsets durchgeführt (siehe folgende Tabelle). Die größten Probleme hatte unser Algorithmus bei Gesichtern mit sehr geringer Auflösung. Einige Gesichter der Testsets besitzen eine Auflösung von nur 31 x 38 Pixel. Die Größe des Sliding Window beträgt 48 x 56 Pixel. Damit der Algorithmus überhaupt alle Gesichter erkennen kann, muß der Wert für die Bildgröße bei Detektion auf mindestens 1600 eingestellt werden. Bei dieser Bildgröße und einer

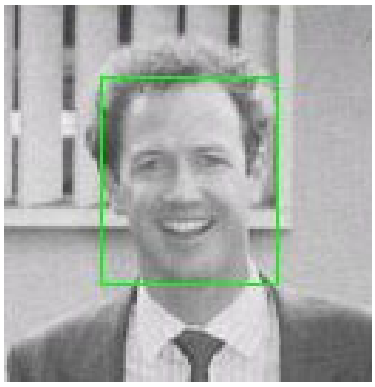


Abbildung 32

Sliding Window Schrittweite von 4 benötigt der Algorithmus ca. 18 Min. für ein Bild auf einem Pentium 4 2,6 Ghz Prozessor mit 512 MB Hauptspeicher.

	Testset 1		***Testset 2***	
	Recall	***False Detections***	***Recall***	***False Detections***
Henry Schneidermann	93,00%	88	91,20%	12
Setup 1 (400, 850, 4)	55,18%	1001	47,13%	126
Setup 2 (500, 850, 4)	49,71%	384	41,40%	55
Setup 3 (400, 1600, 4)	63,71%	3037	49,04%	297

Der Vergleich zeigt, daß die verschiedenen Setups (Schwellenwert, Bildgröße, Schrittweite) unseres Algorithmus nicht annähernd die Detektionsraten von Henry Schneiderman erreichen. Aufgrund des enormen zeitlichen Aufwands konnten wir leider keine weiteren Analysen der Testsets durchführen. Festzuhalten bleibt, daß unser Algorithmus bei Gesichtern mit sehr niedriger Auflösung Probleme hat (siehe Abbildung 33), während Gesichter mit hoher Auflösung sehr gut erkannt werden (siehe Abbildung 34). Leider enthalten die Bilder der CMU Testsets sehr viele Gesichter mit sehr geringer Auflösung.

Abbildung 33

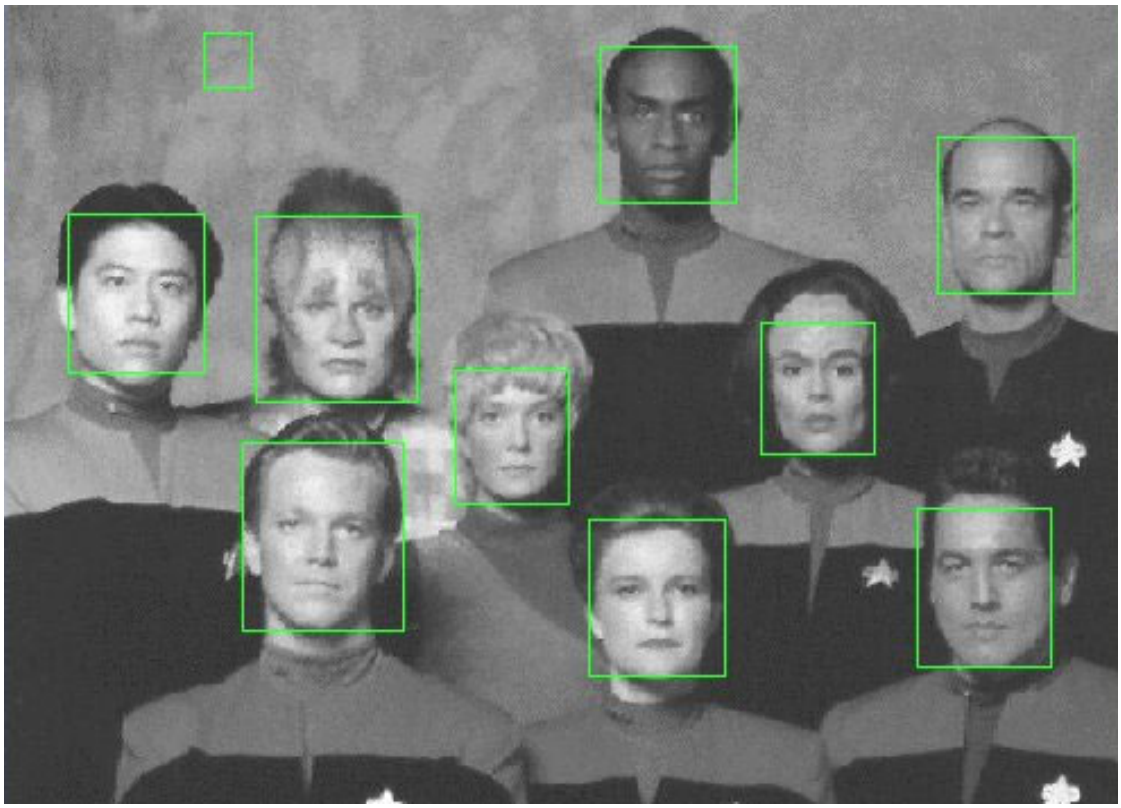


Abbildung 34

6. Fazit

In der aktuellen Version 1.0 ist die Detektion von Profilen ausgeschaltet. Dies hat zwei Gründe: Zum einen war das Antrainieren des Detektors auf frontale Gesichter schon sehr zeitaufwendig, ein zusätzliches Training für Profile wäre im zeitlichen Rahmen dieses Fortgeschrittenen Praktikums nicht zu bewerkstelligen. Zum anderen handelt es sich bei dem aktuellen Detektionsalgorithmus um eine weitestgehend unoptimierte Version. Die Detektion von frontalen Gesichtern dauert bereits sehr lange, das Erkennen von Profilen würde die Dauer des Detektionsvorgangs etwa verdreifachen (da die Überprüfung auf linkes und rechtes Profil nötig ist). Daher ist es ratsam, die Profil-Detektion erst anzugehen, wenn der Detektionsalgorithmus hinreichend optimiert ist.

Schneiderman konnte seinen Algorithmus mit den von ihm in Kapitel 6 seiner Doktorarbeit beschriebenen Optimierungen so beschleunigen, daß eine Detektion eines 240x256 Pixel großen Bildes auf einem Pentium II 450 MHz ca. 90 Sek. benötigt (siehe [5], Seite 62). Unser Algorithmus benötigt im Vergleich dazu für die Detektion eines Bildes in der Auflösung 256x256 auf einem Pentium IV 1,6 Ghz ca. 15 Sek. bei Schrittweite 4. Wenn man bedenkt, daß Schneidermans Algorithmus auf einem wesentlich schwachbrüstigeren System lief und dabei auch die Detektion von Profilen durchgeführt wurde, erscheinen die Ergebnisse in einem anderen Licht.

Die größte Optimierung von Schneiderman dürfte in der Wiederverwendung der Waveletkoeffizienten liegen (siehe [5], Abschnitte 6.2 und 6.3). Unser Detektor berechnet für jeden Inhalt des Sliding Window die Wavelettransformation samt Quantisierung neu. Schneiderman dagegen führt die Wavelettransformation nur einmal pro Skalierungsstufe des Bildes durch. Mit der Verwendung des Skalierungsfaktors $\sqrt[4]{2}$ wird nach jeweils vier bzw. acht Skalierungen eine Halbierung bzw. Viertelung des Ursprungsbildes erreicht, wie sie auch bei einer dreistufigen Wavelettransformation vorkommt. Durch Kombinieren der vorberechneten Waveletkoeffizienten der verschiedenen Skalierungsstufen erspart sich Schneiderman den meisten Rechenaufwand.

Bei der als „Coarse to Fine Search Strategy“ bezeichneten Optimierung werden zuerst die Attribute aus den hohen Bändern der Wavelettransformation berechnet und dann ggf. die Patternberechnung vorzeitig abgebrochen, was nocheinmal Zeit spart (siehe [5], Abschnitt 6.4.1 und [6], Abschnitt 4.2).

In einem Forschungsbericht von 2002 beschreibt Schneiderman, wie er durch Hinzunahme von weiteren Heuristiken („Adjacent Heuristic“, „Color Heuristic“) die Detektionsgeschwindigkeit für o.g. Bild auf der gleichen Maschine auf nur 5 Sek. reduzieren konnte (siehe [6], Seite 30). Es bleibt also noch reichlich Spielraum für Optimierungsmaßnahmen am vorliegenden Detektor. Unsere Aufgabe war es in erster Linie zu untersuchen, ob und wie der Algorithmus funktioniert. Es dürfte noch einmal den Aufwand eines Fortgeschrittenen Praktikums in Anspruch nehmen, um eine vergleichbare Performance hinsichtlich Detektionsgeschwindigkeit auf der einen und Erkennungsgenauigkeit auf der anderen Seite zu erreichen.

7. Installationsanleitung

Um alle Dateiformate öffnen zu können, müssen die Java Advanced Imaging Image I/O Tools installiert werden. Diese können von der Java Image I/O API Download Page [9] heruntergeladen werden. Unter Windows muß die Datei clib_jiio.dll in das Verzeichnis jre\bin und die Datei jai_imageio.jar in das Verzeichnis jre\lib\ext des Java-Homeverzeichnisses kopiert werden. Zur Installation auf anderen Plattformen sei auf die den Image I/O Tools beiliegende readme-Datei verwiesen.

Dem Gesichtsdetektor ist die Batch-Datei „run.bat“ beigefügt. Mit ihr kann die Java-Applikation direkt gestartet werden. Falls nötig, muß vorher noch der Java-Pfad an die lokalen Gegebenheiten angepaßt werden.

8. Literaturverzeichnis

[1] Peter-Michael Ziegler, "Adlerauge, Europas größte Gesichtserkennungsanlage im Zoo Hannover", c't, 9/2003, S.26, Heise Verlag

[2] Ming-Hsuan Yang, David J. Kriegman, Narendra Ahuja, "Detecting Faces in Images: A Survey", Januar 2002, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, Nr. 1

[3] Dr. Andreas Linke, "Spam oder nicht Spam?, E-Mail sortieren mit Bayes-Filtern", c't, 17/2003, S.152, Heise Verlag

[4] "Java Native Interface Technology Programming", http://java.sun.com/javaone/index.html

[5] Henry Schneiderman, "A Statistical Approach to 3D Object Detection Applied to Faces and Cars", 2000, Carnegie Mellon University

[6] Henry Schneiderman, Takeo Kanade, "Object Detection Using the Statistics of Parts", 2002, Carnegie Mellon University

[7] "XnView", http://www.xnview.de/

[8] "Face Detection Project", http://www-2.cs.cmu.edu/~har/faces.html

[9] Java(TM) Image I/O API Download Page, http://java.sun.com/products/java-media/jai/downloads/download-iio.html