Object recognition with Boosting

Hauptseminar *Lernverfahren der Neuroinformatik und Robotik*,
Universität Ulm, Abteilung Neuroinformatik, SS 2007, Betreuer Friedhelm Schwenker

Zusammenfassung

Diese Ausarbeitung basiert auf den beiden Paper [1,2] von P. Viola und M. Jones. Im Kerninhalt wird ein Framework beschrieben, dass visuelle Objekte, die im Bereich der Neuroinformatik als schwer zu klassifizieren gelten, in Echtzeit erkennt. Dieses Framework ist ein Zwei-Klassen Klassifikationssystem, dass im trainierten Zustand Gesichter in Bildern von anderen Objekten unterscheiden kann. Es besteht aus einer Klassifizierer-Kaskade mit mehreren Stufen und wird mit einer AdaBoost Variante trainiert.

Key words: Objekterkennung, object detection, AdaBoost, Asymmetric AdaBoost, Boosting, Klassifizierer-Kaskade, Classifier Cascade, Integral-Bild, Integral Image, Echtzeit, Real-time, Gesicht-Erkennung, face detection, rectangle features

1 Einleitung

Die Echtzeit Objekterkennung gewinnt zunehmend an Bedeutung im Bereich der Robotik und der Automobilindustrie. Echtzeit in diesem Kontext bedeutet, dass das Objekt früh genug erkannt wird, um rechtzeitig darauf reagieren zu können. Vereinfacht ausgedrückt bedeutet das, Objekte so schnell wie möglich zu erkennen.

Das Framework von Viola et al., dass zu Demonstrationszwecken und für vergleichende Resultate mit anderen Gesichterkennungs-Algorithmen, für die Gesicht-Erkennung trainiert wurde, erkennt Gesichter in Graustufen-Videos mit einer Auflösung von 384x244 Punkten und 15 Bilder / Sekunde auf herkömlicher PC-Hardware [1]. Dabei werden mit der von P. Viola und M. Jones vorgestellten Klassifiziererkaskade über 90 % aller Gesichter bei einer false positive rate [2] von 1 in 1.000.000 Testbildern erkannt.

Die Performance des Zwei-Klassen Klassifikationssystems wird durch drei Beiträge von P. Viola und M. Jones erreicht, die ausführlich in Ihrem Paper [1] beschrieben werden. Der erste Beitrag ist das Integral-Bild, dass eine Zwischendarstellung des Graustufen-Bildes repräsentiert, das verwendet werden kann, um Rectangle features schnell zu berechnen. Der zweite Beitrag ist eine Methode zum Bau eines effektiven Klassifizierers, indem bestimmte Features selektiert werden. Dazu wird eine Variante von AdaBoost verwendet um die Features auszuwählen und um den Klassifizierer zu trainieren. Der dritte Beitrag

[1] Intel Pentium III Prozessor, 700 MHz

[2] false positive rate $= \frac{\text{Anzahl der fehlerhaft-positiv erkannten Objekte}}{\text{Anzahl der Bilder mit negativen Objekten}}$

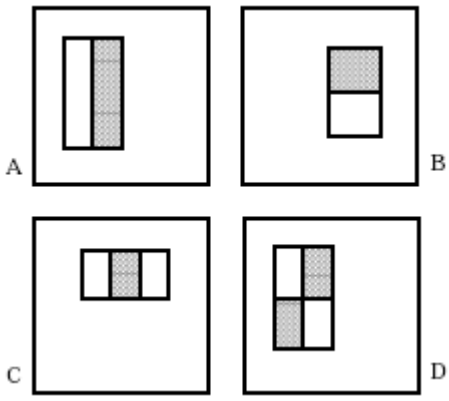

Abbildung 1. Verschiedene Arten von Merkmalen - Zwei-Rectangle feature in (A) und (B), Drei-Rectangle feature in (C) und Vier-Rectangle feature in (D)

ist eine Methode die komplexere Klassifizierer zu einem binärem Entscheidungsbaum, der Klassifizierer-Kaskade, kombiniert. Die Klassifizierer-Kaskade ist stufenformig aufgebaut und besteht im Beispiel [1] aus 32 Stufen und insgesamt 4297 features bzw. 38 Stufen [2]. Die Anzahl der auszuwertenden Features aus dem Bild nimmt mit der Stufenzahl zu, wobei bei einer großen Anzahl von Testbildern nur durchschnittlich 2 Stufen durchlaufen werden.

Im weiteren Verlauf der Ausarbeitung wird im folgenden Kapitel 2 auf Rectangle features und das Integral-Bild eingegangen, Kapitel 3 beschäftigt sich generell mit dem Thema Boosting und mit der Anwendung des modifizierten AdaBoost Algorithmus um einen starken Klassifizierer für die Gesichterkennung zu trainieren. In Kapitel 4 wird die Klassifizierer-Kaskade besprochen, Kapitel 5 bildet mit einer Zusammenfassung der experimentalen Resultate den Abschluss.

2 Merkmalsextraktion

2.1 Rectangle features

Anstatt die Pixel in einem Bild direkt für die Klassifizierung zu nutzen, werden Dezimalzahlen einfacher Merkmale verwendet, die aus dem Bild extrahiert werden. Für die Objekterkennung werden drei verschiedene Arten von Merkmalen genutzt: Zwei-Rectangle features, Drei-Rectangle features und Vier-Rectangle features (siehe Abbildung 1). Ein x-Rectangle feature besteht aus x rechteckigen Regionen, die jeweils die selbe Größe haben und nebeneinander oder untereinander im Erkennungsfenster liegen. Um die Dezimalzahl eines dieser Merkmale auszurechnen, wird das Erkennungsfenster über das Graustufen Bild gelegt und die Summe der Intensitätswerte der Pixel in den weißen Regionen von der Summe der Intensitätswerte der Pixel in den schwarzen Regionen subtrahiert. Der Wert eines Merkmals dient im späteren Verlauf als Eingabe eines einfachen Klassifizierers (siehe Kapitel 3.2). Mit den gegebenen Daten, dass das Basis-Erkennungsfenster 24x24 Punkte groß ist und die Ausrichtung (senkrecht/waagrecht) wie die relative Position im Erkennungsfenster eines x-Rectangle features variabel ist, ergibt sich eine Anzahl von 45.396 möglichen x-Rectangle features. Die Anzahl von features ist deutlich größer wie

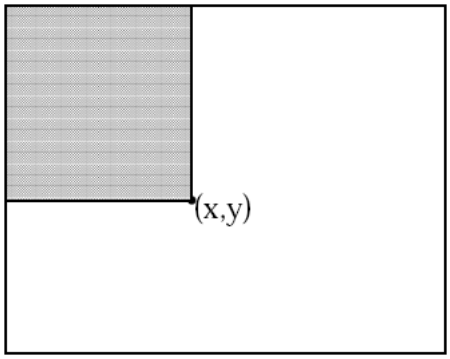

Abbildung 2. Die für die Berechnung des Integral-Bildes an Punkt (x,y) notwenigen Punkte sind grau gekennzeichnet.

die Eingabedimension (Anzahl von Pixel, $24*24 = 576$) und somit mehrfach overcomplete.

Vorteil der rectangle features gegenüber einzelnen Pixeln ist die schnelle Berechnungszeit bei Verwendung eines Integral Images (siehe Kapitel 2.2). Vor allem kann das Erkennungsfenster beliebig skaliert werden, ohne zusätzliche Rechenzeit zu benötigen. Im Vergleich dazu kommt in den meisten Objekterkennungssystemen eine Pyramide bestehend aus mehreren Bildern zum Einsatz, wobei das erste Bild eine bestimmte Anfangsgröße hat und weitere Bilder jeweils um einen bestimmten Faktor (1,25) größer sind. Diese Objekterkennungssysteme verwenden ein nicht-skalierbares Erkennungsfenster, dass über jedes dieser Bilder scannt. Viola et al. haben in Ihrem Paper [1] berechnet, dass ein Objekterkennungssystem, dass solch eine Pyramide mit 11 Stufen und einer maximalen Auflösung von 384x288 Punkten verwendet, ca. 900.000.000 mathematische Operationen in der Sekunde ausführen müsste um die Pyramide zu berechnen und 15 Bilder pro Sekunde abzuarbeiten.

2.2 Das Integral-Bild

Der Wert des Integral Bildes an Punkt (x,y) ist die Summe aller Pixel links und über des Punktes (x,y).

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{1}$$

Wobei ii(x,y) ein Punkt (x,y), auch Feld genannt, des Integral Bildes und i(x,y) ein Punkt (x,y) des originalen Graustufen Bildes darstellt. Diese Formel (1) ist jedoch suboptimal, da für die Berechnung des Integral Bildes einzelne Punkte mehrfach aus dem Graustufen Bild entnommen werden. Durch die folgenden Rekursionsgleichungen kann das Integral Bild in einem einzigen Durchlauf über das Graustufen Bild berechnet werden:

$$s(x, y) = s(x, y - 1) + i(x, y) \tag{2}$$

$$ii(x, y) = ii(x - 1, y) + s(x, y) \tag{3}$$

s(x,y) ist die kumulative Zeilensumme und es gilt: s(x,-1) = 0 und ii(-1,y) = 0. Durch Benutzung dieser Zwischendarstellung kann ein Ein-Rectangle feature mit maximal vier Feld Angaben des Integral Bildes berechnet werden (siehe Abbildung 3): A = 1, B = 2 - 1, C = 3 - 1, D = 4 + 1 - (2 + 3). Zwei-Rectangle features benötigen aufgrund der Eigenschaft, dass die beiden Regionen untereinander oder nebeneinander liegen, sechs Feld Angaben des Integral Bildes, Drei-Rectangle features acht und Vier-Rectangle features neun.

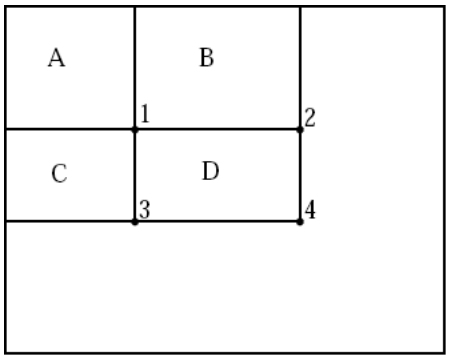

Abbildung 3. Beispiel Rectangles: Das Rectangle feature D kann durch Angabe von vier Integral-Bild Punkten berechnet werden.

3 Boosting

3.1 Einführung

Boosting [3] ist ein generelles Verfahren, um die Genauigkeit eines Lernalgorithmus zu verbessern. Dies geschieht durch gezielte Selektion und Kombination mehrerer schwacher Klassifikatoren zu einem einzigen starken Klassifikator. Ein einzelner schwacher Klassifikator konzentriert sich auf wenige Eingaben (ein einziges Merkmal im spezialisierten Falle Objekterkennung, wie in Kapitel 3.2 beschrieben) und wird schwach genannt, da er nur wenig besser wie der Zufall ($> 50\%$) korrekt klassifizieren muss. Diese Vorgehensweise steht kontrovers zu dem Einsatz eines Mehrschicht-Perzeptron, dass direkt als starker Klassifizierer von zwei (oder auch mehreren) Klassen trainiert wird. Um den Unterschied der beiden Klassifizierungssysteme weiter hervorzuheben, greife ich auf ein Beispiel zurück, dass Robert E. Schapire in seinem Paper [3] in der Einführung genannt hat. Gesucht ist ein E-Mail Spam Filter, der unerwünschte Werbe E-Mails (Junk) von anderen trennen soll. Dazu wird eine bestimmte Anzahl von Spam und Nicht-Spam E-Mails gesammelt und als Eingabe (Text und Grafik binär kodiert) in das Mehrschicht-Perzeptron gegeben. Dieses versucht nun im Trainingsmodus Spam von Nicht-Spam E-Mails zu separieren, d.h. die beiden gegebenen Mengen durch eine möglichst generelle Klassifizierer-Regel zu trennen. Komplexe Problemstellungen wie das Klassifizieren von Spam E-Mails erfordern jedoch komplexe Mehrschicht-Perzeptron Netze, die mit einer durch einen Experten bestimmten Ausgewogenheit zwischen maximalem Trainingsfehler und Komplexität des Netzes trainiert werden müssen, um das Problem des Überlernens [4] zu minimieren. Das Mehrschicht-Netz versucht eine einzige starke Regel aufzustellen, die die E-Mails separieren soll. Boosting jedoch basiert auf der Beobachtung, dass es sehr viel einfacher ist einige Daumenregeln anstatt eine einzige komplexe Regel aufzustellen. Werden diese einfachen Daumenregeln gewichtet und zu einer starken Regel kombiniert so besteht die Möglichkeit, dass diese Kombination bessere Ergebnisse liefert wie die (starke) allgemeine Regel, die von einem Mehrschicht-Perzeptron erstellt wurde.

- **Gegeben**: n Trainingsbilder $(x_1, y_1), ..., (x_n, y_n)$, **wobei** $y_i = 1$ **falls** zu erkennendes Objekt in Bild vorkommt, **sonst** 0. j verschiedene Merkmale.
- Initialisieren der Gewichte: $w_{1,i} = \frac{1}{2m}$ **falls** $y_i = 0$, **sonst** $\frac{1}{2l}$. Dabei ist l = Anzahl der Trainingsbilder, in denen das zu erkennende Objekt vorkommt, $m = n - l$.
- **Für** $t = 1, ..., T$:

(1) Normalisiere die Gewichte $w_{t,i}$.

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

(2) Trainiere einen einfachen Klassifizierer h_j für jedes Merkmal j. Ermittle den Fehler ϵ_j für jeden Klassifizierer h_j in Abhängigkeit der Gewichte w_t.

$$\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$$

(3) Wähle aus allen einfachen Klassifizierern h_j den Klassifizierer mit dem kleinsten Fehler ϵ_j. $h_t = h_j$, $\epsilon_t = e_j$.

(4) Passe die Gewichte nach Schwierigkeit der Klassifizierung (abhängig von Anzahl der Fehl-Klassifizierungen) der zugehörigen Trainingsbildern an.

· **Für** $i = 1, ..., n$:

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

wobei $e_i = 0$ **falls** das Trainingsbild x_i korrekt klassifiziert wurde, **sonst** 1.

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

- Aus dem AdaBoost Lernalgorithmus resultiert der folgende starke Klassifizierer:

$$h(x) = \begin{cases} 1, \text{falls} \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0, \text{sonst} \end{cases}$$

wobei $\alpha_t = log \frac{1}{\beta_t}$

Tabelle 1
Algorithmus 1: Variante des AdaBoost Lernalgorithmus von Viola et al.

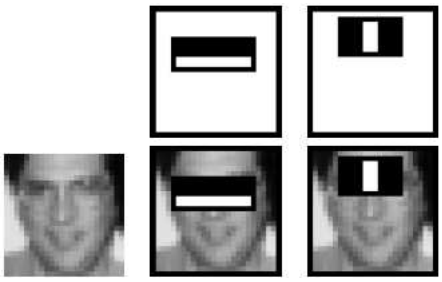

Abbildung 4. Zwei von AdaBoost ausgesuchte Merkmale

3.2 AdaBoost

Adaptive Boosting ist ein Boosting Verfahren, dass die Klassifizierungsfunktion boostet, indem es einfache (schwache) Klassifizierer zu einem starken Klassifizierer kombiniert. Die von Viola et al. präsentierte AdaBoost Variante konstruiert pro Trainings-Runde t einen schwachen Klassifizierer h_t, dessen Eingabe genau ein Merkmal ist. Der schwache Klassifizierer ist typischerweise ein Perzeptron, kann jedoch auch eine Support-Vektor Maschine (SVM) sein. Die einzigen Bedingungen nach Viola et al. [2], die der schwache Klassifizierer erfüllen muss ist, dass die Berechnungszeit und die Erkennungsrate variabel ist. Bei einem Perzeptron kann die Berechnungszeit durch die Anzahl von Eingängen und die Erkennungsrate durch das Anpassen des Schwellkörpers verändert werden. Weiterhin muss der schwache Klassifizierer besser wie der Zufall sein, d.h. Erkennungsrate $> 0,5$. Beim Generieren des schwachen Klassifizierers wird zuerst für jedes zur Verfügung stehende Merkmal j genau ein schwacher Klassifizierer h_j trainiert und der Fehler ϵ_j für jeden Klassifizierer h_j in Abhängigkeit der Gewichte w_t berechnet:

$$\epsilon_j = \sum_i w_i |h_j(x_i) - y_i| \tag{4}$$

Anschliessend wird der Klassifizierer h_j mit dem kleinsten Fehler ϵ_j ausgesucht. h_j wird zu h_t und ϵ_j zu ϵ_t. Dadurch, dass der schwache Klassifizierer auf genau ein Merkmal beschränkt wird, gibt es eine Analogie zwischen Merkmalen und schwachen Klassifizierern die zum Schluss führt, dass pro Runde t nicht nur ein schwacher Klassifizierer erzeugt wird, sondern vielmehr das zu diesem Zeitpunkt t beste Merkmal ausgesucht wird. Am Ende der Runde t werden die Gewichte neu berechnet. Dabei gibt es pro Testmuster genau ein Gewicht, dass bezüglich der Schwierigkeit der Klassifizierung des zugehörigen Musters und somit nach dem Kriterium interessantes / uninteressantes Muster gewichtet wird:

$$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i} \tag{5}$$

$$e_i = \begin{cases} 0, \text{falls das Trainingsbild } x_i \text{ korrekt klassifiziert wurde} \\ 1, \text{sonst} \end{cases} \tag{6}$$

$$\beta_t = \frac{\epsilon_t}{1 - \epsilon_t} \tag{7}$$

Die Gewichte der Trainingsbilder, die oft fehl-klassifiziert wurden, werden erhöht, alle anderen Gewichte verringert. Damit wird sichergestellt, dass in zukünftigen Runden t mit einer erhöhten Wahrscheinlichkeit schwache Klassifizierer h_t ausgesucht werden, die dieses oft fehl-klassifizierte Trainingsbild korrekt klassifizieren, da der Fehler ϵ_t von diesem Gewicht abhängt. Nach Ablauf aller T Trainings-Runden stehen T schwache Klassifizierer mit jeweils einem Merkmal als Eingabe zur Verfügung. Diese schwachen Klassifizierer werden nun zu einem starken Klassifizierer kombiniert, der die Form eines Perzeptrons bestehend aus T gewichteten schwachen Klassifizierern und einem Schwellkörper hat:

---

[3] engl., verstärken

[4] geringer Trainingsfehler aber hoher Generalisierungsfehler

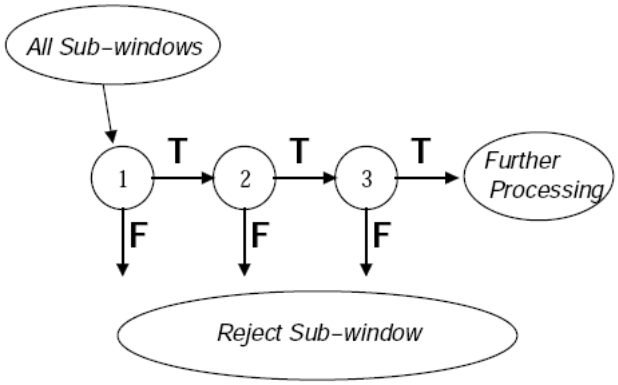

Abbildung 5. Die Klassifizierer-Kaskade bildet einen binären Entscheidungsbaum

$$h(x) = \begin{cases} 1, \text{falls} \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0, \text{sonst} \end{cases} \tag{8}$$

$$\alpha_t = log \frac{1}{\beta_t} \tag{9}$$

α_t stellt ein Gewicht eines schwachen Klassifizierers h_t dar, das invers-proportional zu den Trainingsfehlern β_t ist. Viola et al. haben in einem Testlauf des AdaBoost Algorithmus zur Konstruktion eines starken Klassifizierers die ersten beiden selektierten Merkmale (siehe Abbildung 4) dokumentiert. Das erste Merkmal ist ein Zwei-Rectangle feature und vergleicht die Intensitätsunterschiede der Augen und der oberen Wangenknochen. Das zweite Merkmal ist ein Drei-Rectangle feature das die Intensitätsunterschiede der Augen (graue Region) mit der oberen Hälfte der Nase (weiße Region) vergleicht.

4 Klassifizierer-Kaskade

Basierend auf der Grund-Idee von Viola et al., dass es möglich ist, einen schnellen Klassifizierer mit einer sehr niedrigen false-negative rate [5] (annähernd 0, durch Anpassung des Schwellwertes des starken Klassifizierers) und einer hohen false-positive rate (ca. $0,5$) zu konstruieren, wurde die Klassifizierer-Kaskade abgeleitet. Die Klassifizierer-Kaskade (siehe Abbildung 5) besteht aus einer selbst-bestimmten Anzahl von Stufen, die jeweils aus einem, mit AdaBoost trainiertem, starken Klassifizierer mit aufsteigender Anzahl von Merkmalen bestehen. In die Klassifizierer-Kaskade werden verschiedene Ausschnitte eines Bildes (das Erkennungsfenster wird 11-Mal skaliert bei einer Anfangsgröße von 24x24 Punkten und über das Testbild geschoben, siehe Kapitel 2) gegeben und diese Bildausschnitte [6] durchlaufen der Reihe nach eine bestimmte Anzahl von Stufen (zwischen 0 bis max. Anzahl von Stufen). Wurde im Bildausschnitt in der Stufe x ein Objekt erkannt (der starke Klassifizierer dieser Stufe hat entschieden, dass es ein Objekt sein könnte und gibt es zur Weiterverarbeitung weiter), so wird der Bildausschnitt an Stufe $x+1$ weitergegeben, ansonsten verworfen. Durchläuft der Bildausschnitt alle Stufen so wurde das Objekt entgültig erkannt.

[5] false negative rate $= \frac{\text{Anzahl der fehlerhaft-negativ erkannten Objekte}}{\text{Anzahl der Bilder mit positiven Objekten}}$

[6] engl. sub-window

- **Gegeben**: Maximale false-positive rate f pro Stufe. Minimum Erkennungsrate d pro Stufe. Gesamte false-positive rate F_{target} der Klassifizierer-Kaskade. Menge der Trainingsbilder P, in denen das zu erkennende Objekt vorkommt. Menge der Trainingsbilder N, in denen das zu erkennende Objekt nicht vorkommt.
- $F_0 = 1,0$. $D_0 = 1,0$
- $i = 0$
- **solange** $F_i > F_{target}$
  - $i = i + 1$
  - $n_i = 0$. $F_i = F_{i-1}$
  - **solange** $F_i > fF_{i-1}$
    - $n_i = n_i + 1$
    - Trainiere mit AdaBoost (siehe Algorithmus 1) einen starken Klassifizierer mit n_i Merkmalen. Benutze dazu Menge P und Menge N.
    - Berechne die false-positive rate F_i (Formel (11)) und die Erkennungsrate D_i (Formel (12)) der aktuellen Klassifizierer-Kaskade. Benutze dazu alle Trainingsbilder P und D und teste die Klassifizierer-Kaskade. Verringere den Schwellwert des i. Klassifizierers solange bis die Erkennungsrate $\geq dD_{i-1}$ ist. Beachte, dass der Wert F_i ebenfalls angepasst wird.
  - $N_{temp} = N$. $N = \{\}$
  - **falls** $F_i > F_{target}$ **dann** teste die aktuelle Klassifizierer-Kaskade mit der Trainingsmenge N_{temp}. Alle Trainingsbilder in N_{temp}, die von der Klassifizierer-Kaskade falsch erkannt wurden, werden der Menge N zugeordnet.

Tabelle 2
Algorithmus 2: Meta-Lernalgorithmus für die Konstruktion einer Klassifizierer-Kaskade

4.1 Lernalgorithmen für die Klassifizierer Kaskade

Die Schwierigkeit der Konstruktion einer Klassifizierer-Kaskade, so von Viola et al. festgestellt, liegt darin, ein gutes Verhältnis zwischen Anzahl von Stufen und insgesamte Erkennungsgeschwindigkeit zu finden ohne die Erkennungsrate zu verschlechtern. Um diese Anforderungen zu überprüfen wird die false-positive rate F, die Erkennungsrate D und die benötigte Anzahl von Merkmalen N der gesamten Klassifizierer-Kaskade berechnet:

$$F = \prod_{i=1}^{K} f_i \tag{10}$$

$$D = \prod_{i=1}^{K} d_i \tag{11}$$

$$N = n_0 + \sum_{i=1}^{K} (n_i \prod_{j<i} p_j) \tag{12}$$

dabei ist K die Anzahl von Stufen = die Anzahl von starken Klassifizierern, f_i die durchschnittliche false-positive rate des i.Klassifizierers aller Beispiele, die die Stufe i durchlaufen sind, d_i die durchschnittliche Erkennungsrate des i. Klassifizierers aller Beispiele, die die Stufe i durchlaufen sind, p_j die positive rate des j. Klassifizierers und n_i die Anzahl

von Merkmale des i. Klassifizierers. Beispiel von Viola et al: Eine 10-stufige Klassifizierer-Kaskade $K = 10$ soll eine Erkennungsrate $D \geq 0,9$ und eine false-positive rate $F \leq 1*10^{-5}$ haben. Wie hoch muss dazu die durchschnittliche false-positive rate f sein ? Einsetzen der gegebenen Werte in Formel (10): $0,9 \leq f^10 \rightarrow f \approx 0.99$. Und die durchschnittliche Erkennungsrate d pro Stufe ? Einsetzen der Werte in Formel (11): $1*10^{-5} \geq d^10 \rightarrow d \approx 0.3$. Nicht trivial: Minimieren des Wertes N bei gegebenem Ziel D und F, da ein geringerer N Wert höhere Erkennungsgeschwindigkeit bedeutet. Der AdaBoost Lernalgorithmus, mit dem der starke Klassifizierer jeder Stufe der Klassifizierer Kaskade trainiert wird, konzentriert sich jedoch nur auf die Minimierung der Trainingsfehler, nicht auf die Maximierung der Erkennungsgeschwindigkeit. Mit diesen Problemen haben sich Viola et al. in ihrem Paper [1] beschäftigt und einen Meta-Lernalgorithmus für das Trainieren einer Klassifizierer-Kaskade veröffentlicht (siehe Algorithmus 2). In der Haupt-Schleife $F_i > F_{target}$ wird pro Durchlauf genau eine Stufe zur Klassifizierer-Kaskade mit einem starken Klassifizierer hinzugefügt. Die dabei benötigte Anzahl von Merkmalen für eine Stufe wird in der Unter-Schleife $F_i > fF_{i-1}$ festgelegt und orientiert sich an der gewünschten Erkennungsrate d und der maximalen false-positive rate f pro Stufe. Beide Werte werden einmalig zu Beginn des Algorithmus festgelegt. Die Erkennungsrate D der Klassifizierer-Kaskade kann durch das Verringern des Schwellwertes eines Klassifizierers angehoben werden, bewirkt jedoch auch ein Anstieg der false-positive rate D. Wurde durch das Konstruieren der letzten Stufe die zu erreichende false-positiv rate F_{target} der Klassifizierer-Kaskade noch nicht erreicht, so wird die Menge N vor wiederbetreten der Haupt-Schleife neu sortiert. Dabei werden alle Trainingsbilder aus N entfernt, die die Klassifizierer-Kaskade korrekt klassifiziert. Damit wird sichergestellt, dass sich zukünftige Stufen nur auf die fehl-klassifizierten Trainingsbilder der Menge N konzentrieren, da die vorherige Stufe diese bereits aussortiert hat. Die Reduktion der Trainingsmenge bedeutet weniger benötigte Merkmale für alle weiteren Stufen. P. Viola und M. Jones erreichen mit dieser Vorgehensweise einen Zuwachs in der Erkennungsgeschwindigkeit des Frameworks wie auch in ihren Resultaten ersichtlich wird.

5 Resultate

5.1 Trainingsphase

Viola et al.[1] haben die Klassifizierer-Kaskade mit 4916 Bildern von Gesichtern (siehe Abbildung 6 aus dem WWW[7] mit dem Algorithmus 2 konstruiert und trainiert. Die Bilder wurden zuvor auf die Größe des Erkennungsfensters (24x24 Punkte) heruntersklaliert. 10.000 zufällig selektierte Ausschnitte (von ca. 350 Millionen möglichen) aus 9500 Bildern, die kein Gesicht beinhalten, wurde als negative Trainingsmenge gewählt. Alle Bilder wurden zusätzlich normalisiert, um die negativen Auswirkungen der evtl. schlechten Lichtbedingungen zu kompensieren. Die daraus resultierende Klassifizierer-Kaskade bestand aus 32 Stufen mit insgesamt 4297 Merkmalen, wobei die erste Stufe zwei Merkmale, die zweite Stufe fünf Merkmale und die dritte, vierte und fünfte Stufe 20 Merkmale hat. Schon bereits mit den ersten beiden Stufen, so haben Viola et al. beobachtet, werden 80% aller Bilder die keine Gesichter beinhalten verworfen und fast 100% aller Gesichter

[7] World Wide Web

Abbildung 6. Auszug der verwendeten Trainingsbilder aus dem World Wide Web mit zu erkennenden Gesichtern. Die Bilder wurden bereits auf 24x24 Punkte herunterskaliert.

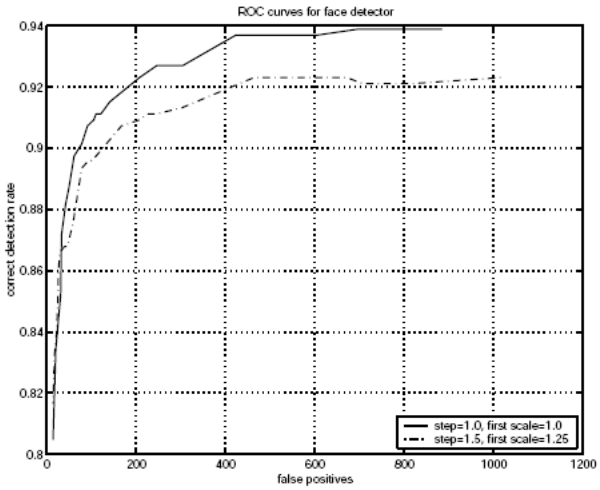

Abbildung 7. ROC Kurve von 32 stufiger Klassifizierer-Kaskade resultierend von MIT+CMU Testbildern

erkannt. Die Wahl der Merkmale pro Stufe, so gibt Viola et al. zu, war ein trial and error Verfahren, in dem die Anzahl der Merkmale n_i solange erhöht wurde bis eine signifikante Reduktion der false-positive rate F_i erreicht ist. Anschließend wurden noch weitere Merkmale hinzugefügt, bis die false-positive rate F_i fast 0 und die Erkennungsrate D_i noch akzeptabel war. Das Training dauerte auf einem 466 MHz Alpha Rechner mehrere Wochen.

Detector, False detections	10	31	50	65	78	95	110	167	422
Viola-Jones	78,3%	85,2%	88,8%	89,8%	90,1%	90,8%	91,1%	91,8%	93,7%
Rowley-Baluja-Kanade	83,2%	86,0%	-	-	-	89,2%	-	90,1%	89,9%
Schneiderman-Kanade	-	-	-	94,4%	-	-	-	-	-
Roth-Yang-Ahuja	-	-	-	-	(94,8%)	-	-	-	-

Tabelle 3
Erkennungsraten verschiedener Objekterkennungssysteme in % bei gegebener Anzahl von fehlerhaft als Gesicht-erkannte Objekte

5.2 Testphase

Das im Kapitel 5.1 trainierte Framework wurde von Viola et al. anhand der MIT+CMU[4] Bilder getestet und mit anderen Objekterkennungssystemen verglichen. Es sind 130 Bilder mit 507 zu erkennenden Gesichtern. Die Ergebnisse des Vergleiches befinden sich in Tabelle 3. Abbildung 7 zeigt die ROC[8] Kurve des Frameworks. Die Kurve zeigt die Abhängigkeit der Erkennungsrate von der Anzahl der dabei gemachten Fehler. Ab einer Erkennungsrate von $0,93$ (feste Linie) wurden mit dem MIT+CMU Testset mehr als 200 Erkennungsfehler (false-positiv + false-negative) gemacht. Die gestrichelte Linie spiegelt die Ergebnisse bei einem Skalierungsfaktor des Erkennungsfensters von $1,25$ statt $1,0$ wieder. Viola et al. kommen zu dem Schluss, dass Vergleiche zwischen allen Objekterkennungssystemen aufgrund des unterschiedlich verwendeten Bildmaterials (das CMU Bildmaterial stand nicht allen zur Verfügung) erschwert durchzuführen waren, sind jedoch der Meinung, dass ihr Objekterkennungssystem eine ähnlich gute, teils bessere Objekterkennungsqualität im Vergleich zu anderen aufweist und die bessere Objekterkennungsgeschwindigkeit hat (um Faktor 15 schneller wie das Rowley-Baluja-Kanade Erkennungssystem und um Faktor 500.wie Schneiderman-Kanade).

6 Asymmetric AdaBoost

In einem weiteren aktualisierten Paper [2] haben P. Viola und M. Jones die Beobachtung gemacht, dass es eine Einschränkung des AbaBoost Verfahrens gibt, die im Zusammenhang mit der Klassifizierer-Kaskade auftritt. Es wurde festgestellt, dass mit diesem Verfahren nur der Klassifizierungsfehler als ganzes minimiert wird und nicht die Anzahl von fehlerhaft erkannten Nicht-Objekten. Die Auswahl der Merkmale orientiert sich somit nur am Klassifizierungsfehler und die selektierten Merkmale selbst sind dementsprechend nicht optimal für das Verwerfen der Bilder, die keine zu erkennenden Objekte beinhalten, geeignet. Da es jedoch die nötige Anforderung gibt in jeder Stufe der Kaskade soviele Nicht-Objekte wie möglich auszusortieren, haben sich Viola et al. mit diesem Problem beschäftigt und der erste Ansatz war die initialen Werte der Gewichtungen anzupassen. Die Gewichte der Testbilder, die zu erkennende Objekte beinhalten, wurden im Vergleich zu den anderen Gewichten erhöht. Hinweis: Die folgenden Formeln variieren im Vergleich zu den im Algorithmus 1 (AdaBoost Lernalgorithmus) verwendeten wie folgt. Z_t ist der Trainingsfehler des schwachen Klassifizierers h_t und entspricht ϵ_t. $D_t(i)$ ist das Gewicht

[8] Receiver Operating Characteristic

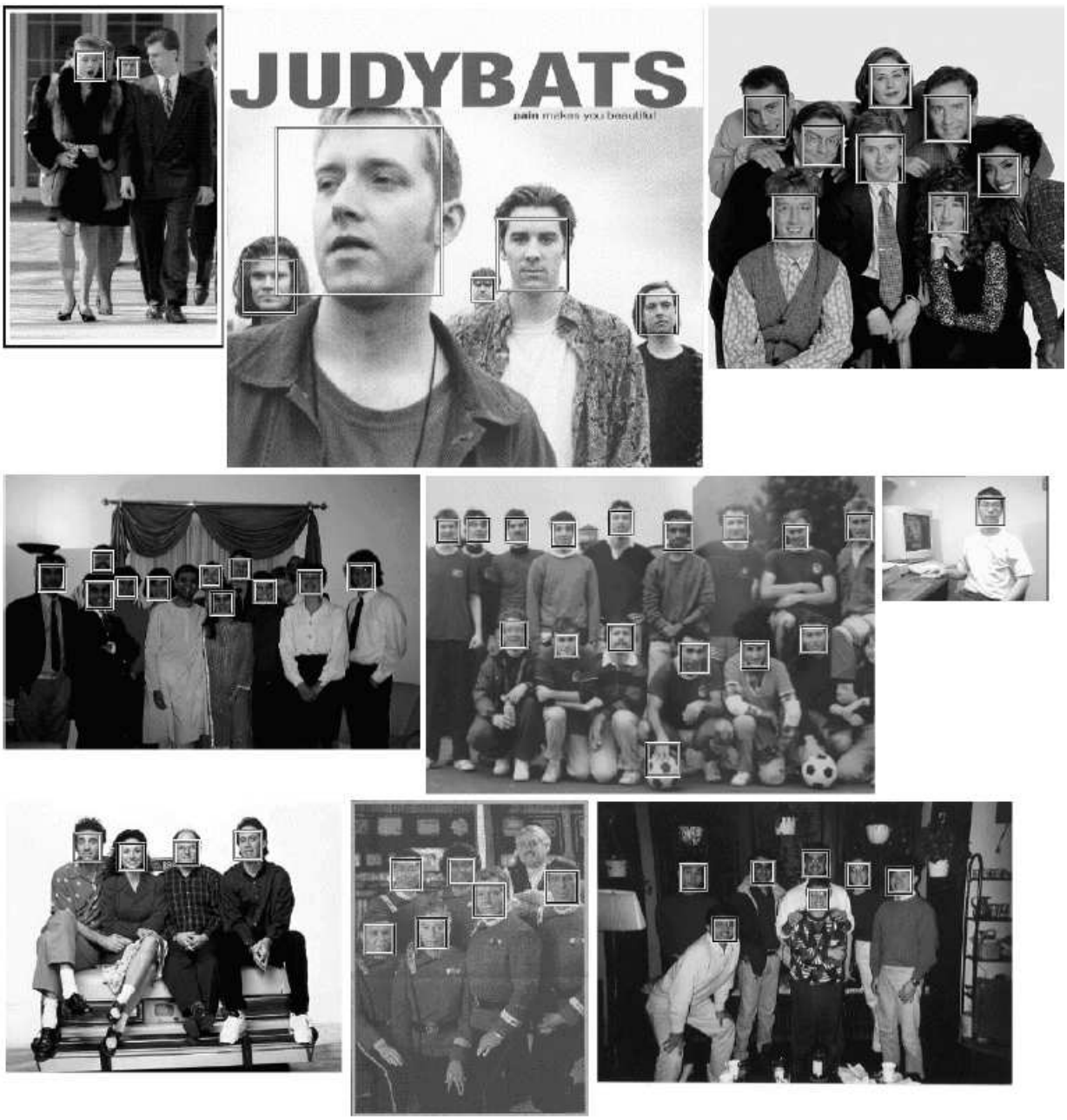

Abbildung 8. MIT+CMU Testbilder. Das graue Viereck um die Gesichter stellt das Erkennungsfenster dar. Gesichter über die kein Erkennungsfenster gelegt wurde, wurden mit der Klassifizierer-Kaskade von Viola et al. nicht erkannt

des i. Muster-Gewichtes in Trainingsrunde t und entspricht $w_{t,i}$. Der Trainingsfehler Z_t des schwachen Klassifizierers h_t ist nicht mehr $\epsilon_j = \sum_i w_i |h_t(x_i) - y_i|$ sondern

$$Z_t = \sum_i D_t(i) \exp(-y_i h_t(x_i)) \tag{13}$$

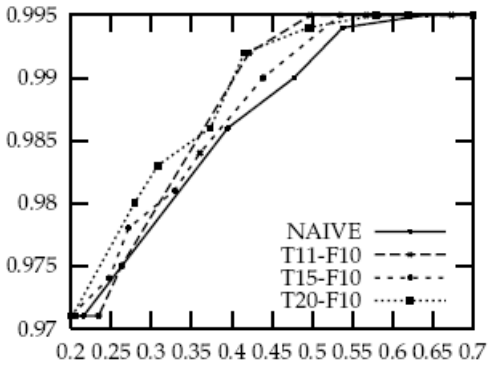

Abbildung 9. ROC Kurve die das einfache AdaBoost Verfahren mit dem asymmetrischen vergleicht. Asymmetric AdaBoost hat bei einer Erkennungsrate von $0,99$ ca. 20% weniger fehl-Erkennungen von Nicht-Objekten.

Der Fehler selbst (ohne Multiplikation der Gewichte) wird in einer Euler-Funktion modelliert. Die Anpassung der Gewichte wird entsprechend des Fehlers Z_t angepasst:

$$D_{t+1}(i) = \frac{D_t(i)\exp(-y_i h_t(x_i))}{Z_t} \tag{14}$$

Beim Minimieren des Fehlers Z_t über T Runden gibt es eine obere Schranke $\prod_t Z_t$ des Trainingsfehlers des resultierenden starken Klassifizierers.

$$\prod_t Z_t = \sum_i \exp(-y_i \sum_t h_t(x_i)) \tag{15}$$

Wird diese obere Schranke modifiziert, so dass jeder Term der Summe für jedes i durch eine Loss Funktion eingeschränkt wird, so bewirkt das Minimieren der oberen Schranke des Trainingsfehlers auch das Minimieren des einfachen Verlustes.

$$\exp(-y_i \sum_t h_t(x_i)) \geq Loss(i) = \begin{cases} 1, \text{falls } y_i \neq C(x_i) \\ 0, \text{sonst} \end{cases} \tag{16}$$

$C(x_i)$ ist die Klasse, die durch den starken Klassifizierer ermittelt wurde (bei Zweiklassen-Klassifizierer $\epsilon\{-1,+1\}$). Da der Klassifizierungsfehler und die Anzahl von Nicht-Objekten nicht voneinander abängig sind, wird von einem asymmetrischen Ungleichgewicht zwischen diesen beiden zu minimierenden Faktoren gesprochen. Viola et al. führen im folgenden eine ALoss Funktion ein, die den asymmetrischen Verlust zurückgibt.

$$ALoss(i) = \begin{cases} \sqrt{k}, \text{falls } y_i = 1 \text{ und } C(x_i) = -1 \\ \frac{1}{\sqrt{k}}, \text{falls } y_i = -1 \text{ und } C(x_i) = 1 \\ 0, \text{sonst} \end{cases} \tag{17}$$

In der Verlustrechnung kosten falsch erkannte Nicht-Objekte das k fache wie falsch erkannte Objekte. Wird die Loss Funktion und die ALoss Funktion zusammen betrachtet und werden beide Seiten der Vergleichsrechnung mit $\exp(y_i\sqrt{k})$ multipliziert, so ergibt

das die asymmetrische Schranke:

$$\exp(-y_i \sum_t h_t(x_i)) \exp(y_i \sqrt{k}) \geq ALoss(i) \tag{18}$$

Die Minimierung dieser Schranke, um den asymmetrischen Verlust (Fehler) zu minimieren, kann erreicht werden, indem im AdaBoost Lernverfahren die Gewichtsanpassung durch eine Voranpassung jedes Trainingsmusters mit $\exp(y_i \log \sqrt{k})$ verändert wird.

$$D_{t+1}(i) = \frac{\exp(-y_i \sum_t D_t(i) h_t(x_i)) \exp(y_i \log \sqrt{k})}{\prod_t Z_t} \tag{19}$$

Die neue obere Schranke zur Minimierung des asymmetrischen Verlustes lautet:

$$\prod_t Z_t = \sum_i (\exp(-y_i \sum_t h_t(x_i)) \exp(y_i \log \sqrt{k})) \tag{20}$$

Viola et al. haben im Laufe der Testphase des neuen Verfahrens zur Minimierung des asymmetrischen Verlustes festgestellt, dass das AdaBoost Verfahren bereits nach der ersten Runde die asymmetrisch verteilten Gewichte verlieren, da das AdaBoost Verfahren bei der Wahl der schwachen Klassifizierer h_t zu gierig [9] vorgeht. Dies kann kompensiert werden, indem jeder schwache Klassifizierer in seiner Sicherheit bezüglich der Klassifizierung modifiziert wird:

$$h'_t = h_t() - \frac{1}{N} \log \sqrt{k} \tag{21}$$

Literatur

[1] Paul Viola and Michael Jones, Robust Real-time Object Detection, *Second International workshop on statistical and computational theories of visio.* (Vancuver,Canada,2001).

[2] Paul Viola and Michael Jones, Fast and Robust Classification using Asymmetric AdaBoost and a Detector Cascade, (2002).

[3] Robert E. Schapire, The Boosting Approach to Machine Learning - An Overview, *Nonlinear Estimation and Classification, Springer* (Florham Park,NJ/USA,2001).

[4] CMU Face Detection Project, http://vasc.ri.cmu.edu/idb/html/face/frontal_images.

[9] engl. greedy