Bachelor-Thesis

des Studiengangs Angewandte Informatik
an der

Evaluierung verschiedener Verfahren zur Gesichts- und Texterkennung in TV-Beiträgen und anschliessender Implementierung eines geeigneten Ansatzes

von

Dennis Weyrauch

4. September 2017

Kurs	AI8
Firma	Hochschule Offenburg
Betreuer	Prof. Dr-Ing. Stefan Hensel
	M.Sc Frederik Böhm
Bearbeitungszeitraum	20.04.2017 - 20.10.2017

Bachelorarbeit

von

Dennis Weyrauch

Kurzfassung

Durch die Digitalisierung erschlossen sich in den letzten 15 Jahren nicht nur viele Möglichkeiten neues Bildmaterial oder Videomaterial aufzunehmen und zu verwahren, sondern auch die Zugänglichkeit der breiten Masse zu solcher Technologie. Abseits der Datenmengen wie sie Social Media Plattformen tagtäglich verarbeiten, existieren die Sende- und Rundfunkanstalten mit gigantischen Archiven von Videomaterial. Der Großteil davon ist von dokumentarischer oder szenischer Natur sowie verschiedenste Interviews aus allen Bereichen des öffentlichen Lebens.

Nach dem aktuellen Stand wird das Videomaterial von Hand kategorisiert und zur Indizierung verschlagwortet. Die Aufgabe war es nun, diesen Prozess zumindest teilweise zu automatisieren. Dazu sollten auf dem Markt verfügbare Technologien in Bereich der Gesichtserkennung und Texterkennung auf ihre Nutzbarkeit zu diesem Zweck hin evaluiert werden. Dabei soll mit Hilfe der in Interviews verwendeten **Bauchbinden** das momentan gezeigte Gesicht *gelernt* werden, um es später ohne solche Hilfe wiederzuerkennen.

*Bauchbinden sind Textboxen am unteren Bildschirmrand, die den Namen (und meist etwas Zusatztext) der geraden interviewten Person zeigen. Das Lokalisieren solcher Boxen ist jedoch nicht Teil der Arbeit; Für die Zwecke dieser Arbeit wird eine feste Position im Bild abgesucht.

Eidesstatliche Erklärung

Hiermit versichere ich eidesstattlich, dass die vorliegende Bachelorarbeit von mir selbstständig und ohne unerlaubte fremde Hilfe angefertigt worden ist, insbesondere, dass ich alle Stellen, die wörtlich oder annähernd wörtlich oder dem Gedanken nach aus Veröffentlichungen, unveröffentlichten Unterlagen und Gesprächen entnommen worden sind, als solche an den entsprechenden Stellen innerhalb der Arbeit durch Zitate kenntlich gemacht habe, wobei in den Zitaten jeweils der Umfang der entnommenen Originalzitate kenntlich gemacht wurde. Ich bin mir bewusst, dass eine falsche Versicherung rechtliche Folgen haben wird.

Autor, Dennis Weyrauch

Datum: 4. September 2017

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Allgemeine Abkürzungen

GPU	Graphical Processing Unit
HOG	Histogram of Oriented Gradients
ICR	Intelligent Character Recognition
MPEG-7	Motion Picture Expert Group - 7
OCR	Optical Character Recognition
SIFT	Scale-invariant feature transform
SURF	Speeded up robust features
SVM	Support Vector Machine
TPU	Tensor Processing Unit
OpenCV	Open Source Computer Vision Library
VJ	Viola-Jones

Sonstige Abkürzungen

bzw.	beziehungsweise
d. h.	das heißt
engl.	englisch
fps	Frames-per-second
MHz	Megahertz
px	Pixel
sog.	sogenannte
u. a.	unter anderem
z. B.	zum Beispiel

1 Motivation

Im Zuge der Digitalisierung entstanden eine Vielzahl an neuen Möglichkeiten, Bildmaterial aufzunehmen und zu verwahren. Da die Sende- und Rundfunkanstalten durch ihren Bildungsauftrag dazu verpflichtet sind, ihre täglich angehäuften Materialien zu archivieren[1], ist im Laufe der Zeit einiges zusammengekommen. Doch nicht nur der rein informelle Aspekt spielt eine Rolle: YouTube (300 Stunden Material pro Minute[2]), Facebook (136000 Fotos pro Minute[3]), Twitter (35000 Tweets pro Minute[4]) – sie alle sind schon allein per Gesetz dazu verpflichtet, diese Massen an Information nach illegalen Inhalten zu filtern. Da ein immer größer werdendes Interesse daran besteht, bereits vorhandenes Material wiederzufinden, macht es Sinn es zum Zwecke der Wiederverwendbarkeit zu kategorisieren.

1.1 Zielsetzung und Anforderungen

Das Ziel dieser Arbeit ist die Entwicklung eines Gesichtserkennungsmodules für ein automatisches Schlagwortsystem von TV-Beiträgen. Dabei soll der Name der zu findenden Person durch das Auslesen aus den sog. Bauchbinden gefunden werden. Als Bauchbinden werden dabei die Einblendungen am unteren Bildschirmrand während Interviews bezeichnet (die orange Fläche rechts in Abbildung 1).

Abbildung 1: Beispiel Bauchbinde
siehe Anhang IV für Vergrößerung

Das System soll hierbei folgende Anforderungen erfüllen:

- Erkennen und Extrahieren eines Gesichtes aus Videoframes.
- Auslesen des Names aus Bauchbinden.
- Verbinden der gewonnenen Daten für Trainingszwecke.

Im diesem Sinne sollen aktuell auf dem Markt verfügbare Technologien auf ihre Verwendbarkeit zu oben genannten Zwecken hin evaluiert werden. Abschließend soll dann mit der

[1] etwa 14 Millionen Videominuten pro Jahr → 26.2 pro Minute (Quelle Betreuer)
[2] Quelle: https://fortunelords.com/youtube-statistics/ (Stand: 2017-07-29)
[3] Quelle: https://zephoria.com/top-15-valuable-facebook-statistics/ (Stand: 2017-08-01)
[4] Quelle: http://www.internetlivestats.com/twitter-statistics/ (Stand: 2017-08)

am Ende ausgewählten Technologie das System implementiert werden.

Folgende Anforderungen sind ***nicht*** Teil des Rahmens dieser Arbeit:

- Erstellung eines Frameworks zur Extrahierung von Frames aus einem Videostream.
- Automatische Erkennung der Position und Dimension von Bauchbinden.
  - Zur Erfüllung der oben genannten Texterkennung werden die Bildkoordinaten der Bauchbinden zunächst manuell eingepflegt.
- Aufbau einer Lernsturktur, um die Gesichter zukünftig ohne Bauchbinden wiederzuerkennen.
- Indizierung der Videos mit den gewonnenen Daten.

2 Grundlagen

Im folgenden Kapitel werden einige technische Grundlagen und Basistechnologien zum Thema Objekterkennung allgemein erläutert, da auch Gesichter und Text nur Objekte mit speziellem Muster sind. Im Bereich der automatisierten Bilderkennung gibt es eine große Anzahl an Verfahren und Vorgehensweisen, um ein Bild zu beschreiben oder für eine Suche zu indizieren. Nach dem momentanen Stand der Technik werden diese historisch bedingt in drei Klassen[1] hinsichtlich ihrer Arbeitsweise unterteilt:

Mustererkennung

„Traditionelle“ Bildverarbeitung durch Mustererkennung und manuelles Abstimmen von Parametern, bis ein handgelesener Trainingssatz zufriedenstellend erkannt wird. Die klassische Texterkennung (OCR[5]) war in den 70er Jahren der Startschuss für das Feld der computergestützten Bildverarbeitung allgemein, und das einzige Teilgebiet, das es heute noch in großem Rahmen nutzt.

Machine Learning

Um den Jahrtausendwechsel herum und bedingt durch die zunehmende Digitalisierung entstanden Ansätze, die aus neuen Datensätzen „lernen“, um Abschätzungen oder Vorher-

[5] Optical Character Recognition

berechnungen zuverlässiger aufzustellen. Das Lernverhalten kann dabei in zwei Gruppen unterteilt werden:

Supervised (*Überwacht*): Datensätze bei denen bereits die Antwort bekannt ist, werden zurückgerechnet, um aus den Zusammenhängen der Variablen zueinander eine Lösungsvorschrift zu erarbeiten, die für ungelöste Datensätze genutzt werden kann.

Unsupervised (*Unüberwacht*): Analyse von Zusammenhängen (z. B. Häuserwahl basierend auf Einkommensklasse/Herkunft), um gewisse Vorhersagen zu treffen. Wenn dann neue Datensätze basierend auf den Vorhersagen eintreffen, wird die Erarbeitungsvorschrift dahingehend abgeändert, dass dieser *Weg* bevorzugt für zukünftige ähnliche Entscheidungen genutzt wird.

Die Grenze dazwischen wird mit Fortschreiten der Zeit immer unschärfer, da Beobachtungssätze durch kontinuierliche Korrektur zu einer verlässlichen Lösungsvorschrift werden. Und mit zunehmender Datenmenge entstehen auch kontinuierlich Datensätze deren Antwort (die Zustandsänderung hin zum darauffolgenden Datensatz) bereits bekannt ist.

Deep Learning

In der heutigen Welt des Big Data werden Machine Learning-Verfahren, die mehr als drei Schichten in Neuronalen Netzen verwenden, als ,,Deep Learning“ bezeichnet. Vieles ist hier noch unbekannt, und Forschungsarbeit investiert, um in Zukunft noch besser davon zu profitieren. Eine detailliertere Erklärung folgt in Abschnitt 2.4.

Die meisten traditionellen Verfahren haben im groben den folgenden Ablauf:

- Das Bild wird auf Pixelebene analysiert, um das Vorhandensein und die Position von Objektspezifischen Merkmalen (im Folgenden Features genannt) zu extrahieren.
- Anschließend folgt in einigen Fällen eine Extrahierungsphase, in der die Features in ihrer Anzahl reduziert und formalisiert werden.
- In den meisten Fällen wird die Menge anschließend auf einen gewissen Richtwert hin normalisiert.
- Abschließend wird mit unterschiedlichen Methoden versucht, die enthaltene Information zu beschreiben.

2.1 Feature detection

Definition:

> Unter dem Begriff *Feature Detection* versteht man die Aufgabe, ein Bild auf seine Grundbestandteile zu abstrahieren, um dann im einzelnen zu entscheiden, ob an dieser Stelle ein Feature vorhanden ist oder nicht. Die Erkennung von Features ist meist der erste Schritt in traditionellen Verfahren und arbeitet mit den rohen Intensitätswerten der Pixel selbst.

Die Charakteristik und die Umsetzung dessen, was ein *Feature* ist, hängt von der jeweiligen Problemstellung sowie den darauffolgenden Stufen ab. Die generelle Auffassung ist, dass es einen sog. ,,Point of Interest“ (Blickfang) repräsentiert, auf welchen dann fortführend aufgebaut werden kann. Ein solcher Blickfang kann eine Linie sein (**Edge**, **Ridge**), ein plötzlicher Richtungswechsel (**Corner**), oder etwas größer auch ein sich wiederholendes Muster (**Blob**). Das deterministische Verhalten eines Feature-Detektors ist essentiell für die Qualität der Ergebnisse, da er die rohen Bilddaten vorfiltert um nachfolgenden Abrbeitsschritten die Arbeit zu erleichtern.

Als Beispiel sei die Gesichtserkennung als Zugangskontrolle genannt: In der Detektionsphase wird das Bild abgesucht, ob und wo ein Gesicht zu finden sein könnte; und im Falle eines Treffers wird die aufwändigere ,,Wiedererkennungsstufe“ ausgelöst, die versucht das gefundene Gesicht zu beschreiben und eine Autoritätskontrolle durchzuführen.

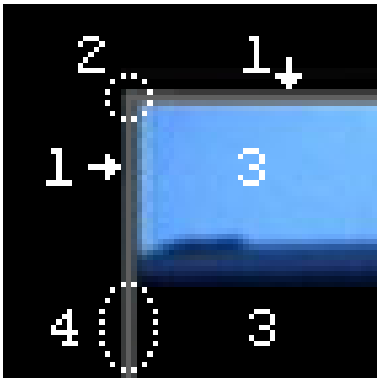


Abbildung 2

1: Edges // 2: Corner
3: Blob // 4: Ridge

Feature Detektoren arbeiten auf den rohen Bilddaten und sind im Vergleich zu den Deskriptoren eher von einfacher Natur. Sie erledigen auch meist eine gewisse Vorarbeit mit Filterung, Skalierung, oder auch dem Berechnen von Differentialen. In den folgenden Unterkapiteln werden die oben erwähnten primitiven Feature-Typen näher erläutert. Abbildung 2 zeigt einen beispielhaften Ausschnitt zur Veranschaulichung.

Abbildung 3: Erzeugung eines Gradientenbildes bei Anwendung des Sobel-Filters.[7]

2.1.1 Edge detection

Edges, engl. für *Kanten*, sind allgemein echte oder wahrscheinliche Abgrenzungen zwischen zwei unterschiedlichen Regionen gleichmäßiger Intensitäten. Sie können beliebige Länge oder Form haben sowie sich überkreuzen. Formell ist eine Kante eine Menge von Pixeln, die aus den umliegenden Intensitäten herausragen, sei es durch Änderung der Tiefe (Abstand des Bildpunktes zum Betrachter), des Materials, oder auch der Beleuchtung [2].

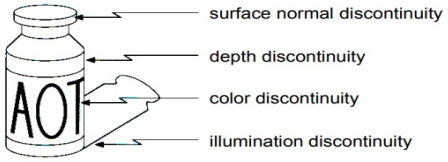


Abbildung 4: Wodurch Kanten entstehen können[6]

Edge-Detektoren werden verwendet, um die Konturen eines Objekts herauszufiltern; Abbildung 3 zeigt ein Beispiel. Gleichzeitig macht es nachfolgende Verarbeitungen weniger anfällig für Farbvariationen oder Helligkeitsschwankungen. Die Arbeiten von John Canny[3] formalisierten die Anforderungen und legten die Grundlagen für weitere Forschungsarbeit. Jedoch ist trotz Fortschritten in der Technik der *Canny Edge Detector* immer noch weitverbreitet und meist die erste Wahl wenn keine problemspezifischen Anforderungen gestellt werden [4].

State-of-the-Art Technologien: Canny Edge Detector[3], Differential-Operatoren wie Roberts [5], Prewit, Sobel[6]

[6]Quelle siehe Anhang 2.
[7]Quelle siehe Anhang 2.

Einschub: *Kernel-Filter*

Unter dem Begriff **Kernel-Matrix** ist hier die Methodik der **Faltunsmatrizen** (engl. *convolution matrix*) zu verstehen und wird in der digitalen Bildverarbeitung für verschiedene lineare Transformationen, meist in Form von Filtern, verwendet. Ein solcher Filter ist eine ein- oder zweidimensionale Matrix, wie z. B. $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ oder $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ (Sobel-Operator).
Das *Falten* ist das gewichtete Addieren der lokalen Nachbarn eines jeden Pixels:

$$\left(\begin{bmatrix} -1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 135 & 75 & 33 \end{bmatrix} \right) = (-1 * 135) + (0 * 75) + (1 * 33)$$

Zu beachten ist, dass eindimensionale Matrizen nur für Edge-Detektoren verwendet werden; Die meisten Filter, wie z. B. ein Gaussischer Weichzeichner, verwenden meist Matrizen der Größe 3x3 oder mehr, um ein Pixel abhängig von all seinen Nachbarn zu verändern.

2.1.2 Corner detection

Corners, engl. für *Ecken*, sind Richtungswechsel von Kanten. Formell ist es der Punkt, in dem sich zwei voneinander verschiedene Kanten (d. h. in unterschiedliche Richtungen verlaufend) überschneiden. In Unterscheidung zu Kanten sind Ecken zweidimensional und besitzen eine feste Position, die über mehrere Bilder hinweg verfolgt werden kann.

Historisch bedingt werden sie auch als *Interest-Point* bezeichnet, hergeleitet durch beiläufige Miterkennung von Endpunkten einer Kante/Linie oder auch isolierten Intensitätsspitzen in mitten einer gleichmäßigen Region. Diese wiederum waren bedingt durch die Entwicklung aus Edge-Detektoren heraus, die durch Nachverarbeitung Überlappungen als Corners identifiziert haben [7].

Corner-Detektoren werden hauptsächlich in Motion-Tracking und 3D-Modellierung verwendet, da sie als Fixpunkte markiert und über einen längeren Zeitraum verfolgt werden können

State-of-the-Art Technologien:
Harris-Operator (als Grundlage für Structure-Tensors)[8, 9], SUSAN[10], FAST; sowie einige Überlappungen mit 2.1.3

2.1.3 Blob detection

Blobs (**B**inary **L**arge **OB**ject), allgemeiner bekannt als Bezeichnung für Generische Binärdaten, bezeichnen die durch **Edges** und **Corners** eingeschlossenen Regionen. In der Bildverarbeitung werden sie definiert als größere zusammenhängende Abschnitte eines Bildes, in denen bestimmte Eigenschaften weitestgehend konstant bleiben.

Durch ihre ursprüngliche Funktion, dem Erkennen von *Interest-Regions* basierend auf Extremwerten, haben sie einige Überlappungen mit der Corner-Detektion. Sie werden meist zur Strukturanalyse von Oberflächen eingesetzt, um Eigenschaften wie Material oder Texturmuster zu extrahieren.

State-of-the-Art Technologien:
Laplacian of Gaussian (LOG), Difference of Gaussian (DoG), Deteminant of Hessian (DoH) ([9])

2.1.4 Ridge detection

Ridges, engl. für *Erhöhung*, sind eine separate Gruppe von Erkennungsmerkmalen mit denen langgestreckte Linien als eigene Elemente erkannt werden sollen. Als Abgrenzung zu Edges (einfacher Intensitätssprung) sind Ridges der Bereich zwischen zwei Intensitätssprüngen in einem ansonsten gleichmäßigen Blob.

Heutzutage wird Ridge-Detektion u. a. als Autoritätskontrolle durch Retinalkapillare oder der automatischen Straßenerkennung in Luftaufnahmen genutzt.

State-of-the-Art Technologien: u. a. Hessian matrix in Scale-Space [11]

2.2 Information reduction

Definition:

Der Oberbegriff der **Information reduction** ist ein Sammelbegriff für Methoden und Verfahren, die eine Grundmenge an Informationen in Größe und Anzahl auf eine bestenfalls nicht-redundante aussagekräftige Menge zu reduzieren, um damit die Effektivität nachfolgender Schritte zu erhöhen.

Die Komplexität eines Datensatzes bestimmt sich durch die Anzahl an variablen Teilen zueinander. Eine hohe Komplexität bedeutet, dass ein Lernalgorithmus Schwierigkeiten bei der Verallgemeinerung haben wird und sich dadurch auf den Trainingssatz überspezialisiert (*overfitting*), was ihn unflexibel auf echte Daten reagieren lässt. Die **Feature Extraktion** bemüht sich daher, den Datensatz auf die aussagekräftigsten Variablen zu reduzieren, die es in Kombination dennoch eindeutig beschreiben können. Im Bereich der Bildverarbeitung beschäftigt sich insbesondere die Texterkennung intensiv mit diesem Thema.

State-of-the-Art Technologien:

- *Principal Component Analysis* (PCA, engl. für Hauptkomponentenanalyse) und dessen Verallgemeinerungen; Verfahren aus der multivariaten Statistik, welche versucht eine hohe Anzahl an Variablen auf eine geringe Anzahl sich nicht überschneidender Eigenschaften zu reduzieren und dabei die größtmögliche Varianz an Eingabewerten abzudecken [12, 13].
- Feature Detektoren (Edge, Corner, Blob, Ridge) sind in dem Sinne ebenfalls Feature Extraktoren, da sie aus rohen Bilddaten die jeweiligen Bildstrukturelemente extrahieren.
- Im Bereich der Neuronalen Netze gibt es *Autoencoder*, ein vorwärtsgekoppeltes unüberwachtes Lernmodel, das versucht einen Datensatz so zu reduzieren, dass aus der Ausgabe trotzdem die ursprüngliche Eingabe rekonstruiert werden kann. Ein solches Beispiel wäre ein Bild, repräsentiert durch Pixel, das in ein Feature-Set umgewandelt wird (z. B. Konturfragmente) [14].

Weitere Bezeichnungen in dem Bereich:

Feature Engineering:

Das Zusammenstellen eines solchen Feature-Sets durch einen oder mehrere Experten, meist spezifisch für einen konkreten Anwendungsfall, wird als **Feature Engineering**

bezeichnet. Dies können entweder vereinfachte Extrahierungsregeln sein (z. B. Netzartiges unterteilen eines Bildes mit den Zellen als Feature) oder sogar konkrete Features selbst (siehe Abschnitt 3.1)

Feature Selektion:

Im Gegensatz zur *Feature Extraktion*, welche Features aus den Rohdaten erzeugt, werden bei der **Feature Selektion** die Features basierend auf einer Erzeugungsvorschrift erstellt. Die Aufgabe ist es nun, eine Untermenge so zu wählen, dass der ursprüngliche Informationsgehalt möglichst komplett erhalten bleibt, d. h. dass redundante oder vernachlässigbare Features aussortiert werden.

2.3 Feature description

Definition:

Als **Feature Deskriptoren** werden meist die Verfahren bezeichnet, die auf den durch obengenannte Methoden erzeugten Ausgaben aufbauen, um schlussendlich das ursprüngliche Bild beschreiben zu können. In den komplexeren Fällen (z. B. Menschen, Autos, etc.) wird meist das komplette Verfahren inklusive Vorverarbeitung und Lernmodel als "der Detektor/Deskriptor"bezeichnet, während der eigentliche Algorithmus nur die problemspezifische Reaktion auf die entstandenen Features ist.

Das Grundproblem bei (Audio-)Visuellen Medien ist, dass deren Inhalte nicht ohne weiteres wie ein Werk in textueller Repräsentation analysiert und indiziert werden können. Hinzukommt, dass ein bestimmtes Objekt nicht nur in Form und Farbe variiert, sondern auch aus unterschiedlichen Winkeln und Entfernungen abgebildet sein kann (ein Problem dass ein simples Dokument nicht aufweist). Das Standardisierungssystem für Metadaten in Audio-Visuellen Mediadateien ist *Motion Picture Expert Group - 7* (MPEG-7) unter ISO/IEC 15938-1.[8][9]

Zu unterscheiden sind dabei **die allgemeinen Deskriptoren**, welche grundlegende Eigenschaften wie Farbe, Umrisse, Texturen, Material, Muster, etc. beschreiben. Sie werden durch einfache Methoden wie Mustererkennung oder Signalverarbeitung aus dem Ursprungsmedium extrahiert. Der Standard definiert jeweils eine Gruppe von Deskriptoren für jede dieser

[8] siehe http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=34228
[9] Die wissenschaftlichen Grundlagen dazu wurden in [15] erarbeitet.

Eigenschaften (siehe unten)[10];
und **problemspezifischen Deskriptoren**, die auf einen Typ von Objekt spezialisiert sind. Diese sind jedoch meist von komplexer Natur, um möglichst viele Variationen des Objekts zu erkennen. Großes Beispiel ist hier die Gesichtserkennung, ebenfalls Teil des Standards.

Farbe: *engl. Color Descriptor*

Die grundlegendste Eigenschaft und die einfachste zu extrahieren. Der Standard definiert Deskriptoren für dominanteste Farbe, Histogramm, Layout, Farbtemperatur, etc.

Textur/Muster: *engl. Texture Descriptor*

Versucht Regionen sich wiederholender Pixelfolgen als Muster zu interpretieren, was unter anderem für Rückschlüsse auf die Oberflächenbeschaffenheit (d. h. Material) verwendet werden kann. Im Standard sind drei Deskriptoren definiert: ein Homogen-Area Descriptor sowie ein Galerie-Wrapper hierfür; und ein Edge Histogram Descriptor für eine Strukturverteilung.

Form: *engl. Shape Descriptor*

Das Herausarbeiten der Umrisse eines oder mehrerer Objekte in einem Bild ist vielleicht die mächtigste aber auch schwierigste Eigenschaft. Die dem Menschen eigene Fähigkeit, beliebige Objekte beim Anblick auf dessen Umrisse zu verallgemeinern und dadurch zu klassifizieren, würde die Indizierung von Bildern fast so effizient wie die eines Dokumentes machen. Es existieren zwar Verfahren diese Umrisse zu erkennen, aber außerhalb des Standards bedarf es zusätzlicher Forschungsarbeit, um diese zusätzliche Verallgemeinerung zu ermöglichen. Der Standard definiert drei Deskriptoren für 2D, zwei für 3D, und einen zusätzlichen zur Umrechnung zwischen 2D und 3D.

Bewegung: *engl. Motion Descriptor*

Um die Bewegung von Objekten zu beschreiben, werden Veränderungen über mehrere Bilder hinweg beobachtet. Im Falle von Kameraaufnahmen jeglicher Art können diese in den Metadaten ihre Eigenbewegung dokumentieren. Zusätzlich dazu sind im Standard drei weitere Deskriptoren definiert: Beschreibung von Bewegungen eines Punktes; Transformationen eines Bildausschnittes über längere Zeit; und intuitive Geschwindigkeit.

Position: *engl. Location Descriptor*

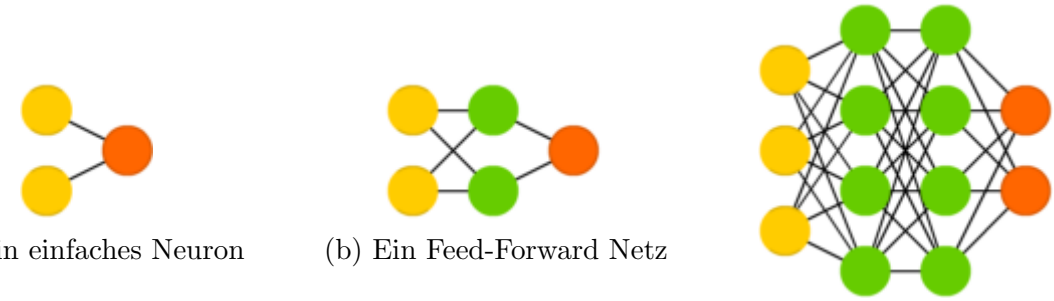
Abschließend enthält der Standard noch zwei Definitionen zur Beschreibung von Posi-

[10] *Anmerkung:* Eine ausführliche Beschreibung der hier aufgelisteten Deskriptoren findet sich in [16].

tion und Lage einer Bildregion in der räumlichen oder zeitlichen Dimension.

Ein kleines Beispiel zur Veranschaulichung wie ein Histogramm eines Bildes erzeugt wird: Das Bild wird zuerst grob in gleichmäßige Teile (z. B. 4x4=16) zerlegt, welche wiederum in mehrere Pixel große Zellen unterteilt werden. Innerhalb dieser Zellen werden dann abhängig von der Zieleigenschaft Durchschnitte berechnet oder Detektoren verwendet. Im Falle der Textur, die mit Edges arbeiten, werden basierend auf Intensitätsverteilungen in der Zelle die Ausrichtung einer Kante, wenn es sie denn gibt, in Toleranzgrenzen auf Richtungskanäle verteilt. Der oben erwähnte Edge Histogram Descriptor verwendet 5 Kanäle (0°/180°, 90°, 45°, 135°, und unbestimmt) bei einer Auflösung von 4x4 Pixeln. Dieses simple Verfahren ist zudem ein effizienter Weg die Hauptsymmetrien/Objektkonturen eines Bildes herauszuarbeiten und wird unter anderem in HOG[11] verwendet.

[11] Histogram of Oriented Gradients; siehe Abschnitt 4.



(a) Ein einfaches Neuron (b) Ein Feed-Forward Netz (c) **Deep** Feed-Forward

Abbildung 5: Schematische Darstellung simpler Neuronaler Netze.[12]

Gelb: Eingabe, Grün: Zwischenschicht, Rot: Ausgabe

2.4 Deep Learning

Wie bereits in Abschnitt 2 erwähnt ist **Deep Learning** eine Vertiefung der klassischen Machine Learning Thematik und verwendet mehrschichtige neuronale Netze, um komplexere Modelle mit höherer Genauigkeit zu evaluieren. Ein Neuronales Netzwerk ist ein Netz aus Neuronen, von denen jeder eine simple Abbildungsfunktion $In \times weight \rightarrow Out$ repräsentiert.

Anders ausgedrückt, nimmt ein Neuron eine Menge an Eingabewerten entgegen, gewichtet sie unterschiedlich, und berechnet eine Ausgabe daraus. Um nun unterschiedliche Entscheidungsfälle abbilden zu können, werden dementsprechend viele Neuronen eingesetzt, wobei jedes Neuron die Eingaben unterschiedlich stark gewichtet. Deren Ausgaben können nun erneut auf verschiedene Weise ausgewertet werden, bis am Schluss ein Ergebnis feststeht. Abbildung 5b zeigt ein Beispiel mit einfacher Abstraktion, während das Beispiel in Abbildung 5c bereits zwei Schichten besitzt, was die Verarbeitung deutlich komplexerer Eingabemuster ermöglicht. Bei einer Anzahl von 3 oder mehr Schichten spricht man von einem **Deep Neural Net** (*Tiefen Neuronalen Netz*).

Werden nun Bilder in einheitlicher Größe als Eingabe verwendet, so erhöht sich die Größe dementsprechend auf die Anzahl der Pixel, da jedes Pixel des Bildes nun einen eigenen Eingangswert darstellt (z. B. 20x20px entsprechen 400 Eingabeknoten). Da Bilder nur zum Zwecke der Objekterkennung verwendet werden, existiert eine ähnliche Kategorisierung wie zuvor in VJ und HOG : Ein Teil der Knoten bestimmt die Wahrscheinlichkeit, dass es Objekt X ist bzw. enthält; und der andere Teil das Gegenteil, ob dies nicht der Fall ist.

[12] Quelle siehe Anhang 2.

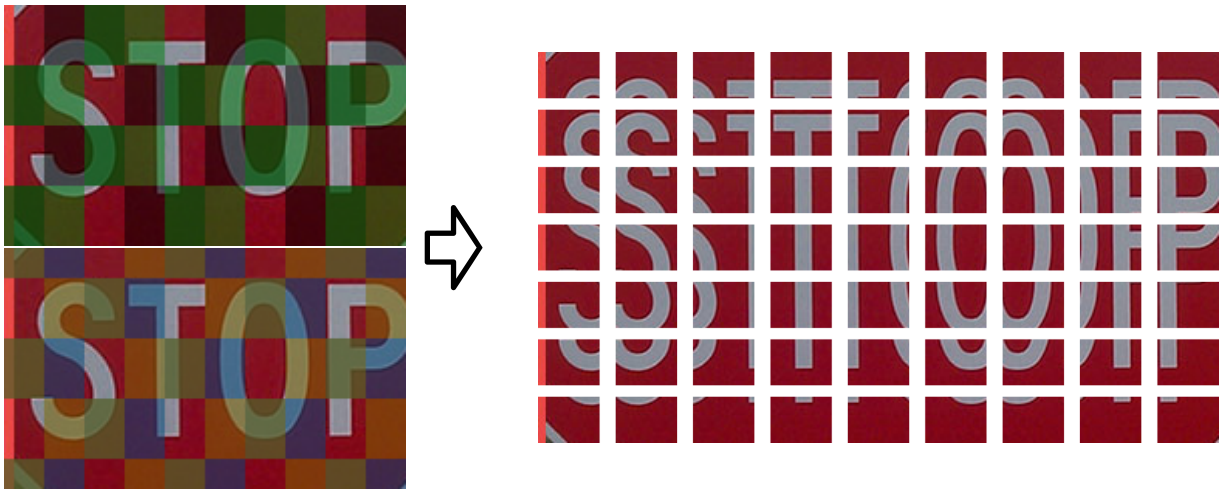


Abbildung 6: Beispiel-Ausschnitt für Convolution.[13]

(200x120 in 63 Teilen von 40x30; 50% Versatz, d. h. 20 bzw. 15px)

Bedingt durch den Aufbau eines neuronalen Netzes ist es normalerweise überspezialisiert auf die Ausrichtung der Trainingsdaten; Ist das Zielobjekt nicht in etwa an derselben Stelle wie der Durchschnitt des Trainingssets, dann wird das neuronale Netz Probleme haben dies als solches zu erkennen. Das Problem lässt sich teilweise durch das Erzeugen neuer aber abweichender Bilder beheben, was im Anschluss die Komplexität des Netzes durch Hinzufügen weiterer Schichten erfordert. Aber schlussendlich sind zwei identische Objekte an zwei verschiedenen Stellen schon allein dadurch zwei komplett verschiedene Objekte für das neuronale Netz, da sie ja auch als solche trainiert werden mussten. Sie sind *nicht kontextfrei*.

Eine Lösung für dieses Problem wird **Convolutional Neural Network**, engl. für *Gefaltetes Neuronales Netz*, genannt. Ziel ist es zu lernen, dass die Position eines Objekts nicht Teil der Kategorisierung ist. Dafür wird in einem ersten Schritt das Bild in kleine sich überlappende Teile gleicher Größe unterteilt; Abbildung 6 zeigt ein Beispiel. Jedes dieser Teile wird dann durch ein kleines aber für alle identisches neuronale Netz verarbeitet, welches auf dieselbe Art versucht das Objekt X (bzw. Teile davon) zu erkennen (vgl. Abschnitt 4). Die Ergebnisse werden anschließend in derselben Reihenfolge und Anordnung wie deren Bildteile abgespeichert, um ausnutzen zu können, dass benachbarte „Felder“ jeweils 50% ihrer Ursprungsdaten teilen.

Um diesen Überschuss nun zu reduzieren, folgt eine sog. *Pooling-Schicht*, in welcher typischerweise das *Max Pooling* angewendet wird: In einem Gitter aus sich nicht überlappenden

[13] Quelle: Ausschnitt aus Abbildung 12.

2x2 Zellen wird jeweils der maximal Wert gespeichert und der Rest verworfen. Der Hintergrund ist, dass bei Makrostrukturen die (pixel-)exakte Position einzelner Strukturelemente meist weniger wichtig ist als die Information, dass an dieser Stelle das Element überhaupt vorhanden ist.

Um graduell komplexere Muster erkennen zu können, werden diese beiden Schritte nach Bedarf wiederholt. Abschließend befinden sich dann ein oder mehrere **vollständig vermaschte Neuronale Netze** (engl. *fully-connected neural network*), welche die eigentliche Klassifizierung vornehmen können. Da die letzte Output-Schicht für jede Kategorie einen Knoten besitzt, in dem die Wahrscheinlichkeit eines Treffers ihrer Kategorie enthalten ist, wäre es vorteilhaft dieses Ergebnis in einen einzelnen Wert (die wahrscheinlichste Kategorie) herunterzureduziert. Hierfür wird in der Regel eine **Softmax**-Funktion verwendet.

Was man sonst noch über das Thema wissen sollte:

- Das Verwenden von **GPUs** (*Graphical Processing Unit*) ist zum quasi Standard im Bereich Deep Learning geworden. Mit dem Vordergrund der effizienten Bildmanipulation sind sie darauf spezialisiert, Matrixmultiplikationen, wie sie in Form von Kernel-Filtern zu finden sind, effizient durchzuführen. Etabliert durch [17], Referenzwerk [18]
- Der Vorreiter im Gebiet der CNN ist (wieder mal) Google mit der Cloud Schnittstelle **TensorFlow**[14]. Ursprünglich entwickelt um *Google Translator* basierend auf Nutzerfeedback intelligenter arbeiten zu lassen, wurde es 2016 unter einer OpenSource Lizenz veröffentlicht, um Forschung und Entwicklung überall auf der Welt zu fördern. Um die Leistung weiter zu steigern, wurden speziell für CNN bzw. Tensorflow allgemein eigene Prozessorchips, genannt **TPU**(*Tensor Processing Unit*), entwickelt[15]. TPUs sind spezielle Berechnungseinheiten, die auf Kosten einiger Nachkommastellen die Rechengeschwindigkeit erhöhen.
- Eine Erwähnung wissenschaftlicher Arbeiten zum Thema ,,Go“ wie z. B. AlphaGo[19][16] darf ebenfalls nicht fehlen.

[14] https://www.tensorflow.org/
[15] https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html
[16] https://deepmind.com/research/alphago/

3 Gesichtserkennung mit Viola-Jones

In der Objekterkennung kann man zwischen zwei Arten des „Erkennens“ unterscheiden: Auf der einen Seite die kategorische Erkennung, das *„was“* in dem Bild zu sehen ist; Hierfür sind Verallgemeinerungen wie Form oder Strukturelemente (z. B. ein Tisch hat vier Beine und eine Tischplatte) ausreichend. Auf der anderen Seite gibt es das *„wer“*, also das Unterscheiden innerhalb derselben Kategorie. Wenngleich dies bei den meisten Kategorien vernachlässigbar erscheint (eine Indizierung aller möglichen Tische oder Stühle bringt kaum einen Mehrwert abseits der Marke), so hat dieser Schritt bei Gesichtern einen weit höheren Informationswert, wenn auch nur durch das Verbinden mit dem Profil der Person.

Ein solches zwei-stufiges Erkennungsverfahren verläuft in mehreren Schritten:

1. Finde alle Vorkommnisse des Zielobjekts im Bild.
2. Fall bedingt die Ausrichtung/Helligkeit normalisieren; z. B. Gesichter so ausrichten, dass Augen auf einer geraden Linie sind.
   - Zu beachten sei, dass ein Objekt X immer noch zu Kategorie Y gehört selbst wenn es in einem unvorteilhaften Winkel gezeigt oder schlecht beleuchtet wird
3. Zerlege das Objekt in Einzelteile mit deren Hilfe es von anderen seiner Kategorie unterschieden werden kann.
   - Im Falle des Gesichts wären das Augenfarbe, Hautfarbe, Gesichtsgeometrie (wie Abstände & Größen), etc.
4. Zuletzt, kombiniere diese Einzelteile zu einem Entscheidungsträger, mit welchem eine Datenbank in der Lage ist, das Objekt eindeutig zu identifizieren.

Im Rahmen einer Machbarkeitsstudie im Vorfeld der Arbeit wurde der Feature Descriptor *Histogram of Oriented Gradients* (HOG) von Dalal & Triggs[20] verwendet, welcher etwa zwei Minuten für eine Minute Videomaterial benötigte. Im Folgenden wird allerdings auf den Vorreiter, das *Viola-Jones Framework* nach Viola *et al.*[21], eingegangen und daraufhin untersucht, inwieweit sich die Bestandteile nutzen lassen, um bestehende Verfahren zu verbessern. Zur Veranschaulichung wird zwar HOG zur Verwendbarkeit herangezogen; dies soll sich jedoch nicht nur auf diesen beschränken.

Das Verfahren nach Viola-Jones, im englischen Original *Viola-Jones object detection framework*, ist deshalb von so großer Bedeutung, da es das erste Verfahren war, das Gesichter in Echtzeit verarbeiten und erkennen konnte. Als Referenzwert: Sie arbeiteten auf einem 700

MHz Intel Pentium III und es wurden Gesichter mit einer Verarbeitungsgeschwindigkeit von 15fps in Bildern der Größe 384x288px erkannt[21]. Im Folgenden werden die einzelnen Schritte analysiert, in welchem Umfang sie sich zur Geschwindigkeitssteigerung des im Anschluss an die Evaluation entscheidenden Verfahrens eignen könnten.

3.1 Haar-Feature Selection

Der erste Teil des Verfahrens, die sogenannten **Haar-Features**, basiert zum Teil auf Arbeiten von Papageorgiou *et al.*[22] über Haar-Basis Funktionen, die mit Hilfe einer SVM[18] bereits erste Erfolge in der (frontalen) Gesichtserkennung vorzeigen konnten.

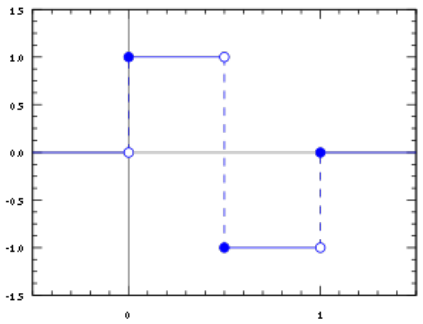


Abbildung 7: Ein Haar-Wavelet

Haar-Transformationen (vgl. Fourier mit orthogonal Basis) werden unter anderem in der JPEG-Kompression verwendet.[17]

Die Funktionsweise der Haar-Features basiert auf der Schattengeometrie von Gesichtern. Durch eine Reihe von Beobachtungen lässt sich feststellen, dass z. B. der Nasenrücken in der Regel heller ist als die tiefer liegenden Augen. Um diese Bereiche erkennen zu können, werden die drei verschiedenen Typen von Haar-Features aus Abbildung 8 in beliebigen Ausrichtungen und Größen verwendet. Der Wert eines solchen Features berechnet sich dabei aus der Differenz der Pixelsummen der dunklen minus der hellen Flächen. Zuletzt werden diese mit Hilfe eines Sliding Windows[19] über das Bild gefahren, um alle möglichen Treffer abzudecken; Die Features, die jedoch am Ende dafür verwendet werden, sind in den nachfolgenden Schritten auf eine überschaubare Menge konzentriert, von anfangs über 150000 pro Sliding Window-Frame, auf etwas unter 500 insgesamt.

Bewertung: Die Tatsache, dass die Haar-Features stark von der Beleuchtung abhängig sind, und, wie sich später zeigen wird, auch auf eine bestimmte Orientierung des Gesichtes, macht das Verfahren unter nicht-gestellten Situationen sehr schwach. Abseits davon, sind die Features selbst der Kern eines jeden Verfahrens; Es wäre nicht mehr derselbe Algorithmus wenn man diese frei austauschen würde.

[17] Bildquelle: https://commons.wikimedia.org/wiki/File:Haar_wavelet.svg
[18] Support Vector Machine
[19] Grundauflösung 24x24 px, siehe Glossar

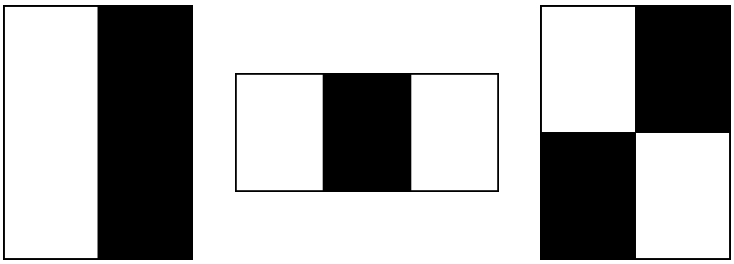


Abbildung 8: Die drei Arten von Haar-Features (HF2, HF3, HF4).

3.2 Integral Images

Der folgende Abschnitt beschäftigt sich mit der effizienten Berechnung von Flächensummen, wie sie von den oben genannten Haar-Features verwendet werden. Unter dem alternativen Namen *Summed Area Table* wurde es 1984 von Frank Crow zur Beschleunigung von Texture-Mapping auf 3D-Modellen erdacht [23]. Der hier verwendete Begriff *Integral Images* stammt, wie das Prinzip dahinter, ebenfalls von Viola *et al.*[21].

Der Wert jeder Zelle (x, y) dieser Tabelle beträgt dann

$$ISum(x,y) := \sum_{x'<=x,y'<=y}^{x',y'=0} i(x',y') \text{ mit } i(x,y) := Intensität$$

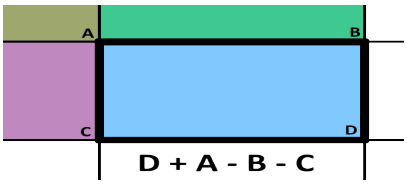


Abbildung 9: Schematische Darstellung des Zugriffs

Durch Ausnutzen der Rekursivität ($ISum(x,0) - ISum(x-1,0) = i(x,0)$) lässt sich diese Formel verallgemeinern auf $I(x,y) = i(x,y) - I(x-1,y-1) + I(x,y-1) + I(x-1,y)$[20], was es ermöglicht, die Tabelle in einem Durchgang zu füllen. Dies erlaubt es, ein Rechteck beliebiger Größe und Position in konstanter Zeit mit nur vier Zugriffen berechnen zu können. Bedingt durch die Geometrie der Haar-Features können dementsprechend HF2 mit sechs; HF3 mit acht; HF4 mit neun Zugriffen berechnet werden.

Alternative: *Pyramiden*

Abseits des obengenannten sind **Pyramiden** die verbreitetste Art und Weise für den Umgang mit Skalierung. Die Schichten einer solchen Pyramide enthalten dabei die Grafik vorausberechnet in mehreren Zoomstufen. Bei facettenreichen 3D-Modellen, z. B. in Videospielen, erhöht dies den Realitätsgrad und steigert die Verarbeitungsgeschwindigkeit, auf Kosten des Speicherbedarfs. Es existieren zwei gebräuchliche Formate dieser Pyramide: *Gaußsche*

[20] $I(x,y) = 0$ für $x, y < 0$

Pyramiden verwenden hochauflösende Bilder um sie auf die gewünschte Größe herunterzuskalieren; *Laplace-Pyramiden* hingegen werden in die andere Richtung zum Hochskalieren verwendet. In Kombination mit Differenzbildern (Zwischenstufen einer Gaußschen Pyramide) kann das Verfahren zur Bildkompression eingesetzt werden[24]. Bei der Bildverarbeitung bedeutet dies nun, dass jedes Bild zuerst in eine vorbestimmte Anzahl an Zoomstufen[21] vorausberechnet wird, um anschließend einen Monoskalaren Detektor darauf anzuwenden. In Viola *et al.*[21] wird beschrieben, dass allein das Berechnen einer solchen Pyramide länger dauert als ihr Verfahren für alle Zoomstufen desselben Bildes benötigt.

Bewertung: Integral Images sind eine effiziente und schnelle Methode zur Flächensummenberechnung mit variablen Abmessungen, was bei Multiskalaren Detektionsverfahren einen immensen Vorteil gegenüber den Pyramiden mit sich bringt. Bei Moloskalaren Verfahren, wie HOG sie anwendet, wären Pyramiden sinnlos, wohingegen Integral Images keinen Vorteil bei einheitlich großen Features bieten. Porikli[25] beschreibt eine Methode, bei der eine angepasste Version zur Vorberechnung der Orientations-Kanäle verwendet wurde. Aufbauend auf den Ergebnissen von Dalal & Triggs[20], dass mit Hilfe verschieden großer Blockgrößen eine geringe Verbesserung auf Kosten der Rechenzeit erziehlt werden kann, zeigt Zhu *et al.*[26], dass in Kombination mit einer Degenerativen Kaskade (siehe 3.4) durchaus eine Performanz-Steigerung um das 70-fache erreicht werden kann.

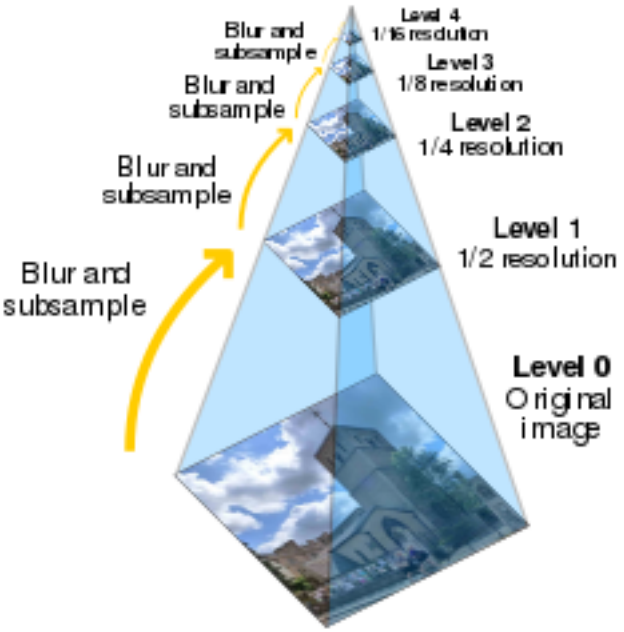


Abbildung 10: Beispiel einer Pyramide[22]

[21] 11 bei VJ; jede Stufe etwa 25% kleiner als die vorherige
[22] Quelle siehe Anhang 2.

3.3 Adaboost Training

Das in Viola-Jones eingesetzte Lernverfahren ist **Adaboost**, kurz für ,,Adaptive Boosting“[27]. Bezogen auf die in Abschnitt 2 beschriebenen Unterteilungen ist es als *Machine Learning* einzuordnen. Die grobe Funktionsweise ist, dass es jeden Entscheidungsträger auf seine Aussagekraft hin überprüft und daraus einen schwach-gewichteten Klassifizierer erstellt. Die N besten werden dann am Ende zu einem stark-gewichteten Klassifizierer verkettet. Im Gegensatz zu anderen Trainingsarten, wie z. B. SVMs[23], werden dabei nur jene Features ausgewählt die auch eine eindeutige Verbesserung des Ergebnisses erzielen können; dies wird im Englischen auch *Predictive Power* (Vorhersagekraft) des Models genannt.

Im Allgemeinen ist ein solcher Klassifizierer als die Summe der dazugehörenden schwach-gewichteten Klassifizierer definiert $[F_T(x) = \sum_{t=1}^{T} f_t(x)]$; ein schwacher Klassifizierer wiederum ist definiert als eine Zustandsfunktion, die zu einem Objekt x einen Vektor zurückliefert, der die vermutete Kategorie repräsentiert. In einem 2D-Raum kann beispielsweise das Vorzeichen die Klassifizierung sein, während der Absolutwert die Verlässlichkeit darstellt. Dies wird im Folgenden als $h(x_i)$ bezeichnet.
Des weiteren wird, pro Klassifizierer, für jedes Objekt eine Gewichtung a_t gesucht, die zusammen mit $h(x_i)$ den gewichteten Fehlerkoeffizienten E_t des vorherigen Klassifizierers verringert.

$$E_T = \sum_i E[F_{t-1}(x_i) + a_t * h(x-1)]$$

Abschließend wird jedem Objekt ein eigenes Gewicht $w_t = E(F_{t-1}(x)$ zugewiesen, basierend darauf, ob sie richtig erkannt wurden oder nicht (Dafür ist die vorherige Markierung des Trainingssets erforderlich). Je kleiner, desto mehr haben dieses Objekt erkannt, was die entsprechenden Klassifizierer-Gewichte ,,glaubwürdiger“ macht.

Viola *et al.*[21] verwendete eine leicht modifizierte Version von Adaboost die auf die Haar-Features angepasst ist. Die Entscheidungsträger sind in diesem Fall die Haar-Features und die Zustandsfunktion ist eine Grenzwertfunktion, die entscheidet, wie sicher das Feature als Gesicht bzw. nicht als Gesicht zu bewerten ist. Zudem reduziert Adaboost die Anzahl der Features von ca. 162000[24] auf einige hundert bis tausend.

[23] Support Vector Machines
[24] basierend auf der Grundauflösung des Sliding Windows von 24x24px

Der Ablauf ist wie folgt:

Init Gegeben sei eine Menge an Trainingsdaten der Größe N mit den Eigenschaften (x^i, y^i); mit $y^i \in \{1, -1\}$ abhängig ob es ein Bild enthält oder nicht.

1. Initialisiere Gewichtung $w_1^i = \frac{1}{N}$
2. Für jedes mögliche Feature f_j in $j = 1, ..., M$
   (a) Renormalisiere $\sum_{i=1}^{N} w_j^i = 1$
   (b) Teste f für alle Bilder i und finde Grenzwert θ_j + Kategorie s_j, sodass der gewichtete Fehlerkoeffizient minimal ist.

$$\theta_j, s_j = \arg\min_{\theta,s} \sum_{i=1}^{N} w_j^i \varepsilon_j^i \text{ mit } \varepsilon_j^i = \begin{cases} 0 & y^i = h_j(x^i, \theta_j, s_j) \\ 1 & \text{ansonsten} \end{cases}$$

   (c) Setze die Gewichtung $\alpha_j = 1 - \text{Fehlerrate}$.
   (d) Reduziere die Gewichtung w_{j+1}^i für jene Bilder, die korrekt erkannt wurden.
3. Kombiniere die Resultate zu einem stark-gewichteten Klassifizierer $h(x)$:

$$h(x) = \text{sign}\left(\sum_{j=1}^{M} \alpha_j h_j(x)\right) \text{ mit } h_j(x) = \begin{cases} -s_j & f_j < \theta_j \\ s_j & \text{ansonsten} \end{cases}$$

Bewertung: AdaBoost war damals und ist immer noch einer der besten Lernalgorithmen wenn eine große Menge an Entscheidungsträgern in einen Entscheidungsbaum umgewandelt werden soll. Kégl[28] stellt eine Erweiterung vor, die auf einer Höhe mit Support Vector Machines (SVMs) stehen. Wie bereits im vorherigem Kapitel erwähnt, wird Adaboost von Zhu *et al.*[26] verwendet, um eine effiziente Auswahl aus den nun in verschiedenen Größen vorhandenen HOG-Features[25] zu bewerkstelligen. Zum Vergleich: Bei Dalal & Triggs[20] sind es 105 Blöcke, während Zhu aus 5031 Blöcken auswählt.

3.4 Cascading Classifiers

Als letztes wird eine Entscheidungskaskade, die **Cascading Classifiers**, eingesetzt. Betrachtet man die Menge aller Frames, die ein Sliding-Window aus einem Bild extrahiert, dann wird der Anteil der Frames, die tatsächlich ein Zielobjekt erhalten *können*, relativ gering

[25] siehe Abschnitt 4.

sein. Ohne die Kaskade würden für jedes Frame alle (bzw. bis zu einem gewissen Fehlergrenzwert) Entscheidungsträger ausgewertet werden, was in 99% der Fälle erfolglos bleiben wird. Dalal & Triggs[20],

Die Idee ist nun, dass jede Schicht einen Entscheidungsträger so trainiert, dass er nur einen bestimmten Prozentsatz der falschen Bilder aussortiert und alle anderen akzeptiert. Der Unterschied ist jedoch, dass die nachfolgenden Schichten nur mit den zuvor akzeptierten Bildern trainiert werden, was immer komplexere Entscheidungsträger erfordert, um die Fehlerrate weiterhin einzuhalten. Im späteren Einsatz werden abgelehnte (Sliding-Window)Frames sofort abgebrochen, was bei einer effektiven Eingangsschicht eine Beschleunigung von bis zu 50% erbringen kann. Im Folgenden wird der grobe Ablauf veranschaulicht:

Als Eingabe dienen:

1. f = maximal akzeptierte Fehlerrate pro Schicht (*False Positive Rate*)
2. d = minimal akzeptierte Erkennungsrate pro Schicht (*True Positive Rate*)
3. $Ftarget$ = maximale Fehlerrate der Kaskade mit $F_{ges} = \prod_{i=1}^{K} F_i$
4. P, N als Menge der positiven bzw. negativen Bilder.[26]

```
init: F, D, n → Liste
      F[0] = 1.0, D[0] = 1.0, i = 0
while F[i] > Ftarget do // Füge neue Schicht hinzu:
    i++; n[i] = 0; F[i] = F[i-1]
    while F[i] > f × F[i-1] do
        n[i]++ // Anzahl der Features
        Trainiere Klassifizierer aus n[i]-Features mit P & N
        Teste Klassifizierer und setze F[i], D[i] entsprechend
        if D[i] < d × D[i-1] then loop
            Verringere Grenzwert θ des Klassifizierers
        until D[i] >= d × D[i-1]
    N = ∅ // Leere NegativMenge
    if F[i] > Ftarget then
        Evaluiere Schicht mit OriginalMenge von N
        Füge alle Treffer in N ein.
```

Listing 1: Cascading Pseudocode

[26] K = Anzahl der erzeugten Schichten

Eine Schicht wird den Klassifizierer solange erweitern bis eine messbare Verbesserung der Fehlerrate erzielt werden kann; umgekehrt werden Schichten hinzugefügt solange die Testmenge keine zufriedenstellend geringe Fehlerrate feststellt. Die Experimente von Viola *et al.*[21] brachten eine Kaskade mit 32 Schichten hervor, bestehend aus insgesamt 4297 features. Die ersten beiden Schichten, bestehend aus 2 und 5 jeweils, sortierten bereits 80% der Negativen Frames aus. Die Kaskade als Ganzes steigert die Effizienz um ca. 1000%.[27]

Bewertung: Kombiniert mit einem der inzwischen zahlreichen Varianten der Boost-Familie erzeugen Cascading Classifiers immer noch eine der besten Entscheidungsbäume im Bereich der Bildverarbeitung. Wie in den vorherigen zwei Abschnitten hat Zhu *et al.*[26] auch die Cascading Classifiers verwendet, um die von Adaboost erstellten SVMs als Klassifizierer zu trainieren und in einem Entscheidungsbaum abzubilden. Abschließend berichten sie, dass im direkten Vergleich mit dem Original nach Dalal & Triggs[20] nicht nur ähnliche Resultate produziert werden, sondern diese auch um bis zu 70fach schneller berechnet wurden.[28]

3.5 Evaluation

Nun, stellt sich die Frage: Wenn die Einzelteile so erfolgreich sind, wieso wird der Algorithmus als Ganzes nicht mehr breitflächig verwendet? Das Hauptproblem sind wie bereits erwähnt die Haar-Features. Der Ursprung lag darin, eine effiziente Gesichtserkennung für Umgebungen zu bauen, in denen CPU-Leistung Luxusgut ist, beispielsweise in Digitalkameras. Dies mag zwar um die 2000er herum Gültigkeit gehabt haben, jedoch wie alles andere sind auch Kameras heutzutage Hochleistungsgeräte. Nichtsdestotrotz hat der daraus entstandene Autofokus überlebt[29]. Haar-Features sind zu abhängig von den Optimalbedingungen um effizient zu sein. Die Nachteile im Detail:

1. Die Abhängigkeit von Licht: Die Evaluierung funktioniert nur mit Licht aus einer bestimmten Richtung, welche annehmbarer Weise oben durch die Sonne ist; kommt das Licht jedoch von der Seite, müsste entsprechend ein neues Trainingsset gebaut werden. Unzureichende Beleuchtung (oder zu viel) könnte jedoch in gewissem Rahmen durch Gammakorrektur behoben werden.
2. Da Haar-Features auf Rechtecke beschränkt sind, sind Drehungen oder Neigungen des

[27] Quelle: [VJ] s.15 & fig.7
[28] Hinweis: Diese Algorithmus ist geschützt unter US-Patent 20070237387.
[29] siehe 3 für ein Beispiel

Zielobjekts meist unmöglich zu erkennen (Überspezialisierung auf Trainingsbilder), abseits der Tatsache, dass ein eigenes Trainingsset erforderlich wäre.

Wie bereits oben erwähnt, das Austauschen der Features durch andere Informationseinheiten, wie es Zhu *et al.*[26] beschreibt, zeigt, dass dieser Ansatz noch lange nicht ins Archiv gehört. Die Vorteile im Einzelnen:

1. Inverted Images macht die Features unabhängig von Skalierung und Position des Zielobjekts. Dies ist auch der Hauptvorteil gegenüber den verbreitet eingesetzten Pyramiden.
2. AdaBoost war schon damals etabliert, und das hat sich bis heute nicht verändert. Es gibt verschiedenste Varianten für unterschiedliche Problemstellungen, um aus einer Menge an Entscheidungsträgern die Besten auszuwählen.
3. Und zuletzt, die Cascading Classfiers, welche durch einen Degenerativen Entscheidungsbaum effiziente Vorsortierung von zu evaluierenden Bildausschnitten eines Sliding-Windows vornehmen.

4 Stand der Technik

Wie bereits in Abschnitt 3 erwähnt, gibt es drei Schritte ein Objekt, hier ein Gesicht, zu identifizieren:

1. Finden aller Vorkommnisse
2. Ausrichten auf einen Normwinkel
3. Vektorisieren und mit Datenbank vergleichen

4.1 Erkennen der Gesichter

Die aktuellen *state-of-the-Art* Technologien in diesem Bereich sind *Histogram of Oriented Gradients* (HOG) nach Dalal & Triggs[20] und *Scale-invariant feature transform* (SIFT), wobei im Folgenden näher auf HOG eingegangen wird.

Zuerst wird das Bild angepasst. Die gängisten Varianten sind:

1. Anwenden eines Gaußschen Weichzeichners, um Kompressions-Rauschen zu entfernen; dies ist nur ratsam wenn die Hauptgradienten stärker hervorgehoben werden sollen.
2. Transformieren der Farbkanäle in ein Graustufen-Bild;
3. Anwenden der Gammakorrektur, meist um die allgemeine Helligkeit zu normalisieren.
4. Skalieren auf ein volles Vielfaches der Sliding-Window Schrittweite oder eines anderen Richtwertes.

Es ist implementierungsabhängig, welche davon ausgeführt werden oder ob das Bild in Originalform verarbeitet wird. Die verbreitetsten Implementierungen befinden sich in den Frameworks *Open Source Computer Vision Library* (OpenCV) und *dlib*[30]. Beide beinhalten zudem eine große Bandbreite an Funktionalität für den Bau eigener Verfahren im Bereich der digitalen Bildverarbeitung.

Als nächstes wird ein Edge-Detektor genutzt um die Gradienten des Bildes herauszuarbeiten. Es hat sich herausgestellt, dass die effektivsten Filter hierfür simple eindimensionale Ableitungen, wie die 1x3 Variante des Sobel-Operators[31], sind. Die eigentlichen Gradienten entstehen dann durch das Durchlaufen in beide Richtungen. Seien die Gradienten in x-Richtung gx und die in y-Richtung gy, dann lassen sich zwei Metriken daraus berechnen:

[30] http://opencv.org und http://dlib.net/
[31] $[-1, 0, 1]$

Die Intensität(en) mit $\sqrt{gx^2 + gy^2}$; und die Orientierung mit $arctan\left(\frac{gx}{gy}\right)$. OpenCV bietet hierfür die Funktion $cartToPolar(gx, gy)$ an. Dalal & Triggs[20] weisen darauf hin, dass bei Farbbildern die Gradienten für jeden Farbkanal separat berechnet und anschließend nur der höchste Wert zur Berechnung der Metriken genutzt wird.

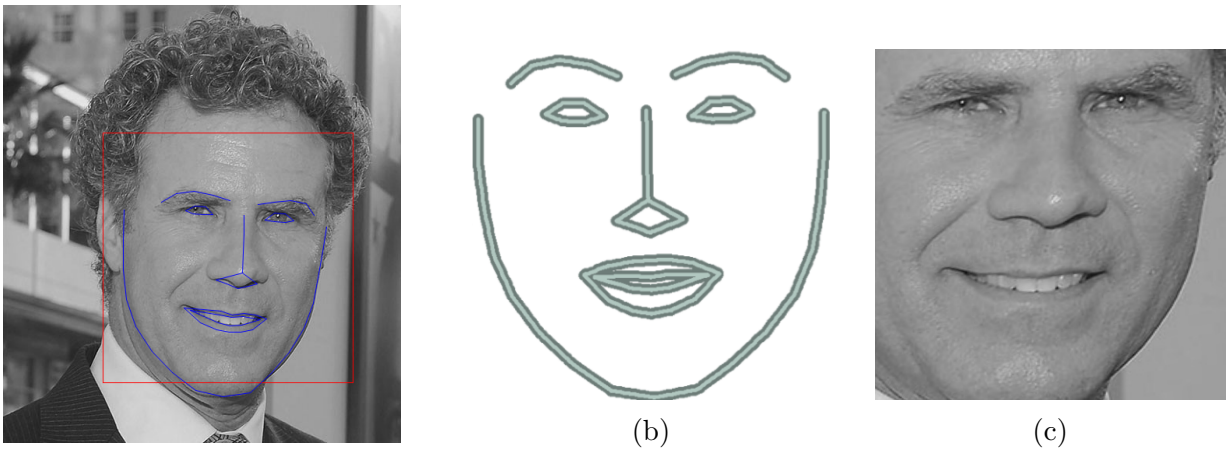
Da ein einzelner Gradient jedoch an sich nur geringen Informationsgehalt besitzt, werden sie in quadratische „Zellen“ der Größe 8, 16, oder 32 zusammengefasst. Innerhalb dieser Zellen werden die Intensitäten nach ihren Orientierungen gruppiert und zusammengezählt. Anhang V zeigt ein Beispiel. Zuletzt wird mit einem Sliding-Window dann über das Feature gefahren, um nach das Zielobjekt identifizierenden Mustern zu suchen. Es bietet sich an, Zellen zuvor zu Blöcken zusammenzufassen, um eventuelle Intensitätsspitzen wegzunormalisieren; einige Implementierungen verwenden stattdessen eine Vektormaske, die aus dem Durchschnitt aller positiven Trainingsbilder erzeugt wurde. In beiden Fällen wird die Evaluierung durch eine SVM durchgeführt.

4.2 Ausrichten des Gesichts

Es hat sich herausgestellt, dass das Ausrichten des Gesichts auf einen gewissen *Mittelwert* (z. B. die Augen immer auf einer Linie, Nase zentriert, etc.) die Qualität der Ergebnisse steigern kann. Um die erhaltenen Gesichter nun auf diesen Mittelwert hin zu transformieren wird eine Technik names **Face landmark estimation** (etwa *Gesichtslandkarte*) verwendet. Es existieren viele verschiedene Implementierungen hiervon [29, 30, 31, 32, 33, 34]; im Folgenden wird sich auf [34] in der Form in welcher es bereits in dlib vorhanden ist, bezogen.

Laut [35] ist die Implementierung von dlib theoretisch in der Lage die Facial Landmarks in etwa einer Millisekunde zu finden; Da dem jedoch die Gesichtserkennung voran geht, dauert der ganze Prozess etwa 30ms. Mit etwas Optimierung kann dies jedoch immer noch eine Verarbeitungsgeschwindigkeit von etwa 60 bis 70 fps schaffen.[32] Anschließend müssen diese Landmarks dann noch auf die zuvor erwähnte Durchschnittsform transformiert werden. Da Parallelitäten erhalten bleiben sollen, werden ausschließlich affine Transformationen verwendet. Ziel ist es, die nun gefundenen Landmarks auf die in Abbildung 11b gezeigte Form zu transformieren, damit die Augen und Mundwinkel auf parallel zueinander verlaufenden Linien liegen. Abbildung 11a und 11c zeigen ein Beispiel.

[32] Werte stammen aus eingebettetem Video in [35].
[33] Siehe Anhang 2für Quelle des Orginalbilds von (a)/(c) sowie von (b).



(a) (b) (c)

Abbildung 11: Vorher-nachher Bild von Will Ferrell bei Ausrichten auf die zentrierte Maske.[33]

4.3 Vektorisieren / Encodieren

Als letztes müssen dann noch die Gesichter zu den Personen zugeordnet werden. Ein trivialer Ansatz wäre es, die unbekannten mit allen bereits bekannten Gesichtern zu vergleichen, was aber aus offensichtlichen Gründen nicht praktikabel ist. Ein anderer Ansatz wäre es, einige definierende Merkmale und/oder Größenverhältnisse dafür heranzuziehen, wie Augenfarbe, Länge des Nasenrückens, Stirnhöhe, etc. Es hat sich jedoch herausgestellt, dass solche Merkmale von geringem Nutzen für den Algorithmus sind; Stattdessen ist es das Beste, die für ihn nützlichen Angaben durch den Algorithmus selbst bestimmen zu lassen. Zu diesem und inzwischen vielen weiteren Zwecken wird *Deep Learning* verwendet. Anzumerken sei, dass neuronale Netze auch dafür verwendet werdenm dessen biologisches Vorbild, das Gehirn, besser zu verstehen.

Genauer, es wird in den meisten Fällen ein *Deep Convolutional Neuronal Network*[34] verwendet, um diese Abmessungen zu erhalten. Als Trainingsmethode wird dabei sogenanntes **Triplet Sampling** (engl. *Drillingsbeprobung*) eingesetzt: Es werden zuerst zwei Bilder des Gesichts der gewünschten Person eingespeist und dann mit einem dritten, fremden Gesicht überlagert. Das neuronale Netz wird nun versuchen, dass die Differenz der Abmessungen der ersten beiden Bilder zueinander kleiner ist als zum dritten Bild. Wird dies nun Millionen mal wiederholt, sollte das neuronale Netz in der Lage sein, zuverlässige Abmessungen für jedes Gesicht zu erzeugen [36]. Ein Auszug aus einer auf diese Weise generierten Liste ist in

[34] Wie in Abschnitt 2.4, s.13 erläutert.

Tabelle 1 zu sehen.

Auch wenn das Training mit sehr hohem Rechenaufwand verbunden ist, so muss es nur einmal am Anfang durchgeführt werden. OpenFace bietet ein bereits ausreichend trainiertes Netz für genau diesen Zweck an[35], worauf sich nachfolgende Arbeiten stützen werden.

0.097496084868908
0.12529824674129
0,030809439718723
...

Tabelle 1: Beispiel für Abmessungen

In einem letzten Schritt müssen diese Abmessungen dann mit dem gewünschten Namen zusammen in ein simples Klassifizierungssystem (z. B. eine SVM) eintrainiert werden, damit zukünftige Abmessungen (aus unbekannten Gesichtern) korrekt zugeordnet werden können.

[35] https://github.com/cmusatyalab/openface/tree/master/models/openface

5 Methodiken der Evaluation

Dieser Abschnitt befasst sich mit Methodiken, wie verschiedene Objekterkennungsverfahren miteinander verglichen werden können. Zu unterscheiden sind dabei solche, die nur eine räumliche Erkennung oder Abgrenzung vornehmen (meist Detektoren); und solchen, die sich bemühen Informationen zu extrahieren oder zu klassifizieren (Deskriptoren).

5.1 Intersection over Union

Der **Jaccard Index**[37], auch genannt *Jaccard Koeffizient* oder **Intersection over Union**, ist eine Methodik die Ähnlichkeit zweier Datensätze zueinander zu bestimmen. Er ist definiert als das Ergebnis der Schnittmenge (*Intersection*) geteilt durch die Vereinigungsmenge (*Union*) beider Datensätze:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Daraus leitet sich ab, dass, je geringer die nicht miteinander geteilte Menge ist, desto größer muss die Ähnlichkeit der beiden Datensätze zueinander sein.

Umgekehrt existiert die **Jaccard Distance** als Maß für die Unterschiedlichkeit zweier Datensätze und ist damit das Komplementär zum Jaccard Index. Sie berechnet sich entweder aus der Differenz des Indexes zu 100% $[1 - J(A,B)]$ oder alternativ aus der Differenz der Vereinigungsmenge und Schnittmenge geteilt durch die Schnittmenge:

$$d_J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

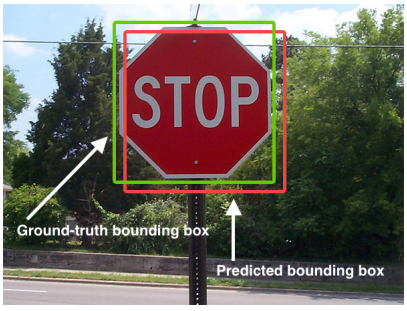


Abbildung 12: Intersection-over-Union Beispiel[36]

Angewandt auf die Bildverarbeitung wird er dazu verwendet, die Qualität eines Trainings mit Hilfe der tatsächlichen Positionen zu bewerten. Es kann dabei jeder Algorithmus getestet werden, solange dessen Ergebnisse eine *Bounding Box* um die erkannte Bildregion wiedergeben. Voraussetzung ist

[36] Bildquelle: https://commons.wikimedia.org/wiki/File:Intersection_over_Union_-_object_detection_bounding_boxes.jpg

natürlich, dass die Trainingsdaten ebenfalls (manuell) mit solchen Rahmen versehen wurden, um besagte Überschneidung feststellen zu können (siehe Abbildung 12).

Eine solche Qualitätskontrolle wird in drei Schritten durchgeführt: 1. Trainieren des Detektors mit einem Trainingssatz; 2. Anwenden des Detektors auf einen *Testsatz*; 3. Vergleichen der Testergebnisse mit den manuell markierten Bounding Boxen. Um die Überspezialisierung gering zu halten, kann die Menge der Trainingsbilder und der Testbilder per Zufall (in einem festgelegten Verhältnis) aus einem kombinierten Satz ausgewählt werden.

Listing 2 zeigt in Pseudocode wie sich der IoU-Wert aus einem Detektionsergebnis und dem korrespondierenden Trainingsrahmen berechnen lässt:

```
calc_IoU(bA, bB) {
  // Bestimme den Bereich der Schnittmenge
  xA, yA = max(bA.x1, bB.x1), max(bA.y1, bB.y1)
  xB, yB = min(bA.x2, bB.x2), max(bA.y2, bB.y2)
  // Berechne Rechtecksfläche
  areaA, areaB, areaIoU = (.x2 - .x1 +1) * (.y2 - .y1 +1)
  // Berechne IoU
  iou = areaIoU / (areaA + areaB - areaIoU)
  return iou
}
```

Listing 2: Berechnung Intesection of Union

5.2 Confusion matrix

Im Bereich der Klassifizierungssysteme werden hingegen meist **Wahrheitsmatrizen** (engl. *confusion matrix, error matrix*) verwendet. Eine solche Tabelle kreuzt die tatsächliche Anzahl an positiv und negativ markierten Trainingsdaten mit denen, die das System als solche erkannt hat. Typischerweise wird es für binäre Klassifizierungen nach dem Ja-Nein Schema verwendet; Es lässt sich jedoch beliebig auf n-dimensionale Klassifizierungen erweitern. Anzumerken sei, dass diese Methodik allgemein für alle Arten von statistischen Testverfahren verwendet wird, allen voran in der Medizin für symptomatische Schnelltests, aber eben auch in der Informatik.

		Wahrheitswert	
		Positiv	Negativ
Vermutung	Positiv	TP	FP
	Negativ	FN	TN

TruePositive = Treffer
FalsePositive = Falscher Alarm
FalseNegative = Übersehen
TrueNegative = Verworfen

Abbildung 13: Confusion Matrix

Abbildung 13 zeigt den allgemeinen Aufbau einer solchen Tabelle. Wie der Tabelle zu entnehmen ist, gibt es zwei Fälle von Fehlern zu unterscheiden: Im ersten Fall werden Unwahrheiten akzeptiert (z. B. Verhaftung eines Unschuldigen); und im zweiten werden Wahrheiten abgelehnt bzw. nicht erkannt (z. B. Freispruch eines Kriminellen). Alternativ werden sie auch FehlerTyp I und II genannt. Es lassen sich verschiedenste Metriken daraus ableiten, aber die Wichtigsten davon sind:

- die **Sensibilität** (engl. *Sensitivity*), die Wahrscheinlichkeit, dass ein Vorhandensein auch erkannt wird; ($\rightarrow TP/\sum Positiv$)
- die **Spezifität** (engl. *Specificity*), die Wahrscheinlichkeit, dass Abwesenheit als solche erkannt wird; ($\rightarrow TN/\sum Negativ$)[37]
- sowie die Präzision (alternativ PPV[38]) bzw. Verlässlichkeit, dass ein gegebenes positives Ergebnis auch tatsächlich positiv ist;
- und dessen Antithesis NPV[38], für negative Ergebnisse.

Weiterhin wichtig ist die **Prävalenz** (engl. *Prevalence*), der Prozentsatz des Datensatzes der das Zielobjekt beinhaltet. Eine Vorhersage wäre schließlich selbst dann 100% korrekt, wenn alle Daten im Satz der selben Wahrheitsklasse angehören würden. Das ist auch der Grund, weshalb PPV und NPV keine verlässlichen Metriken sind, da sie nicht selbst-definierend sind. Zudem sind sie zueinander nahezu ausschließend, da entweder ein verlässlicher Ausschluss ($NPV \uparrow = \ FN \downarrow, \ FP \uparrow$) mit anschließend eingehender Untersuchung der (meist wenigen) positiven Ergebnisse erfolgt; oder umgekehrt ein verlässlicher Treffer ($PPV \uparrow = \ FP \downarrow, \ FN \uparrow$) und Untersuchung der negativen Ergebnisse.

Eine weitere Maßzahl, welche vor allem im Bereich des Machine Learning verwendet wird, ist der Matthews Kooeffizient[39]. Er wird folgendermaßen aus den vier Grundwerten der

[37] Sensibilität & Spezifität sind normiert unter EUR 17538
[38] Positive Predictive Value & Negative Predictive Value[38]

Tabelle berechnet:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Dabei entsteht eine Zahl im Bereich $[-1, 1]$, wobei 0 für „nicht besser als zufälliges Raten“ steht und positive bzw. negative Werte den Grad der Verbesserung/Verschlechterung gegenüber diesem Nullpunkt aufzeigen.

6 Texterkennung

Im Vergleich zur Objekterkennung und im Speziellen zur Gesichtserkennung sind die Zielobjekte bei der Texterkennung, also die Zeichen eines Textes, von theoretisch begrenzter Anzahl und wohldefiniert in ihrer Form. Generell gibt es zwei Richtungen zu unterscheiden:

- *Optical Character Recognition* (OCR) für gedruckten bzw. machinell gefertigten Text
- *Intelligent Character Recognition* (ICR) umfasst zusätzlich handschriftlich verfassten Freitext

Der Hauptunterschied zwischen der statischen OCR und dynamischer ICR ist, dass in der Echtzeiterfassung der Bewegungsablauf für das Schreiben des Zeichens als zusätzlicher Eingabeparameter zur Verfügung steht[40]. Ursprünglich enstanden aus dem Bedarf Lesegeräte für Blinde zu bauen, wurde es zuerst hauptsächlich zur Digitalisierung von von Hand ausgefüllten Dokumenten, wie Formulare, verwendet. Erst allmälich, mit Fortschritten in der Technik, kam die etwas aufwändigere Handschriftenerkennung hinzu, was die Digitalisierung alter Bücher und Manuskripte erlaubt[39]

6.1 Allgemeines

Wie zuvor bei der Objekterkennung gibt es einige gängige Aufbereitungsschritte, um die Qualität des Ergebnisses zu steigern:

Normalisierung
Sicherstellen einer geraden Ausrichtung, Größe, etc. und eventuelle Störflächen (versuchen) auszugleichen; ein Beispiel wäre ein Kaffeefleck auf einem eingescannten Dokument oder das Rauschen der Papierstruktur.

Binarisation *Zwei-Farben-Bild*
Da für die Erkennung eines Zeichens effektiv nur dessen Form zu erkennen sein muss, hat es sich etabliert, ein solches Dokument in ein binäres Schwarz-Weiß Bild zu transformieren. Effektiv soll dabei eventueller Hintergrund weg reduziert werden (Weiß/0), während alles andere eingeschwärzt wird (Schwarz/1)[41]. Es hängt natürlich von der Problemstellung ab, welche Teile als unwichtig markiert werden sollen und welche nicht. Insbesondere Formulare

[39] Projekt Gutenberg beschreibt den Nutzen davon für ihre Zwecke seit 1989

sind darauf ausgelegt, schnell und effizient ausgelesen zu werden, und beinhalten daher meist eine kontrastreiche, vorgedruckte Segmentierung, die das Ausfüllen in Druckbuchstaben erleichtert.

Layout-Analyse
Um insbesondere bei mehrspaltigen Dokumenten die Reihenfolge des Textflusses zu erhalten, muss eine Strukturanalyse des Dokuments durchgeführt werden, um die verschieden Inhaltstypen wie Überschriften, Blocktext, Aufzählungen, Tabellen, Bilder, etc. sauber voneinander zu trennen. Der exakte Ablauf sowie Durchführung wird meist auf den zu erwarteten Eingabetyp angepasst. Bei einem Freitext Bild, d. h. einem Bild, in dem zufällig Text enthalten ist, wird sicherlich keine Layout-Analyse stattfinden, und man würde eher eine normale Objekterkennung bemühen, um die Textregion einzugrenzen.

Der eigentliche Verarbeitungsschritt besteht dann meist in der Segmentierung von zuerst Linien, Wörtern, und zuletzt Zeichen; das Erkennen der Schriftart, wenn trainiert, kann den Prozess beschleunigen. Abseits der primitiven auf Pixelmuster basierenden Vergleiche aus den Anfängen der Textverarbeitung wird heutzutage durch sog. **Nearest Neighbour classifiers** (engl. *Nächster Nachbar Klassifizierer*) aus teilweise mehrfach unterteilten Zeichen der bestmögliche Treffer gesucht. Der Prozess ist stark rekursiv, da in einer bestimmten Sprache bestimmte Zeichenfolgen häufiger zusammen auftauchen als andere, was zu einem selbst-korrigierenden Prozess führt. Typische Beispiele wären $c| <-> d$, oder $rn <-> m$. Unterstützt durch Wörterbücher können dadurch auch ganze Wörter zumindest als Korrekturvorschlag angegeben werden.

6.2 Tesseract

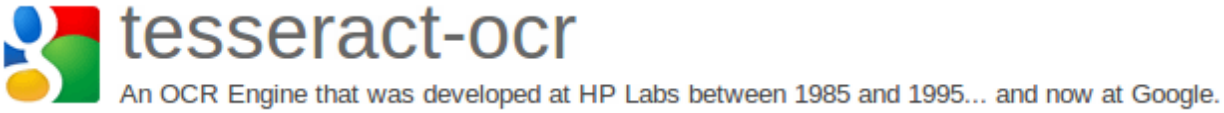


Abbildung 14: Tesseract-Logo[40]

Main Repository: https://github.com/tesseract-ocr/tesseract

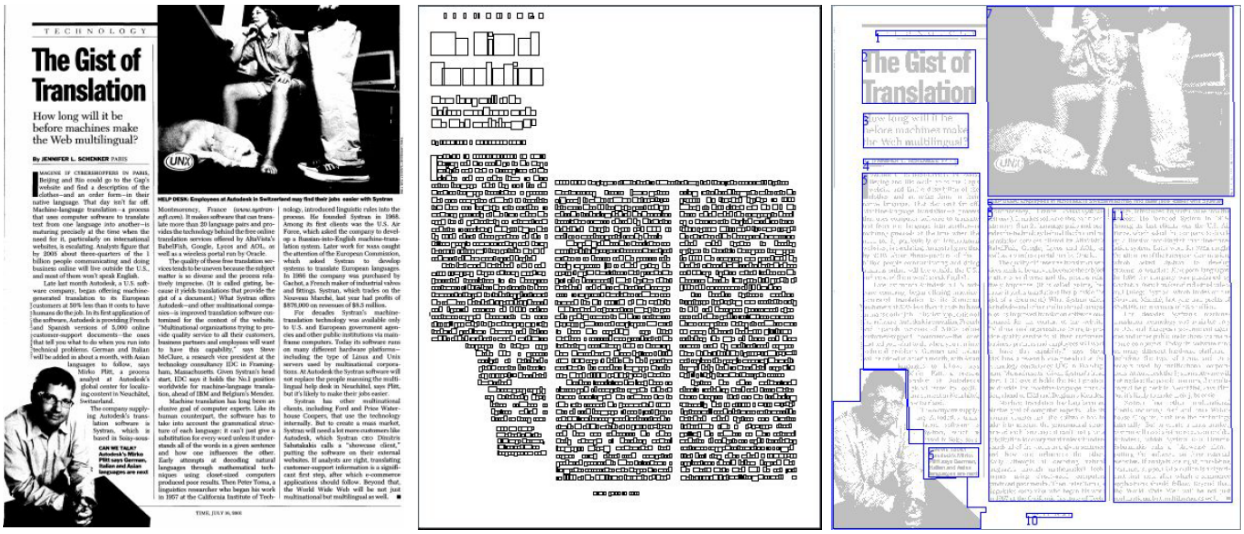
Das am weitesten verbreitete und auch etablierteste System zur Texterkennung ist Tesseract (siehe Abbildung 14) welches seit 2008 von Google gesponsert wird. [41]

Dessen grundlegender Ablauf ist wie folgt:

1. Allgemeine Normalisierung des Eingabebildes
2. Page Layout Analyse: Zeilen (und eventuelle Spalten) erkennen, um Linien zu extrahieren
3. Linien normalisieren und in Wörter zerteilen.
4. „Chopping“ des Wortes in Glyphe-Kandidaten.
5. Identifizieren der Segmente; eventuell unter Einbindung von Nachbarn
6. Besten Treffer für Wort als ganzes suchen
7. Schritte 5-6 wiederholen bis eine zufriedenstellende Kombination aller Segmente gefunden wurde.
8. Evtl. umgebrochene Wörter verbinden & Kreuz-Vergleich auf oft zusammen vorkommende Wortfolgen (wie *New York City*)
9. Nachbereitung und Ausgabe

Im Folgenden wird etwas genauer auf die Schritte 2, 5, und 6 eingegangen[42]

[40] Bildquelle: https://commons.wikimedia.org/wiki/File:TesseractLogo.png
[41] https://web.archive.org/web/20061026075310/http://google-code-updates.blogspot.com/2006/08/announcing-tesseract-ocr.html
[42] basierend auf den Unterlagen zu https://github.com/tesseract-ocr/docs/tree/master/das_tutorial2016/ und [42].



(a) (b) (c)

Abbildung 15: Schematische Darstellung simpler Neuronaler Netze.[43]

Layout-Analyse

Da für den Algorithmus ein Bild unabhängig vom Inhalt nur eine Ansammlung von Pixeln ist, muss in jedem Bild wie zuvor bei der normalen Objekterkennung der Text zuerst gefunden werden. Unter der Annahme einer eingescannten Seite eines Journals, Zeitung, etc., ist die Herausforderung, die unterschiedlichen Teilelemente nicht miteinander zu vermischen. Abbildung 15 zeigt ein kleines Beispiel: Es muss erkannt werden, dass so etwas wie ,,Spalten“ existiert, um sie nicht mit einem Wortzwischenraum zu verwechseln. Um die einzelnen Regionen zu klassifizieren existieren zwei Varianten: *Top-down* wird in der Regel für vortrainierte/vordefinierte Muster, wie z. B. Formulare, genutzt; *Bottom-Up* hingegen ist freier und ähnelt mehr einer Blob-Detektion. Es wird versucht zusammenhängende Pixel in Regionen zusammenzufassen, welche in bestimmten Abständen zu rechteckigen Boxen verallgemeinert werden. Darauf aufbauend muss natürlich die Flussrichtung erkannt werden, da u. a. einige ostasiatische Sprachen wie Japanisch, Chinesisch, und Koreanisch in horizontaler wie vertikaler Richtung geschrieben werden können. Zuletzt müssen mit Hilfe der Fluchtlinien, die sich zwischen den Boxen kenntlich machen (Abbildung 15b), die verschiedenen Layoutelemente als Ganzes verpackt werden (Abbildung 15c).[44]

[43] Bildquelle siehe Anhang 2; Original aus https://dataset.primaresearch.org
[44] Quelle: Tesseract[5]

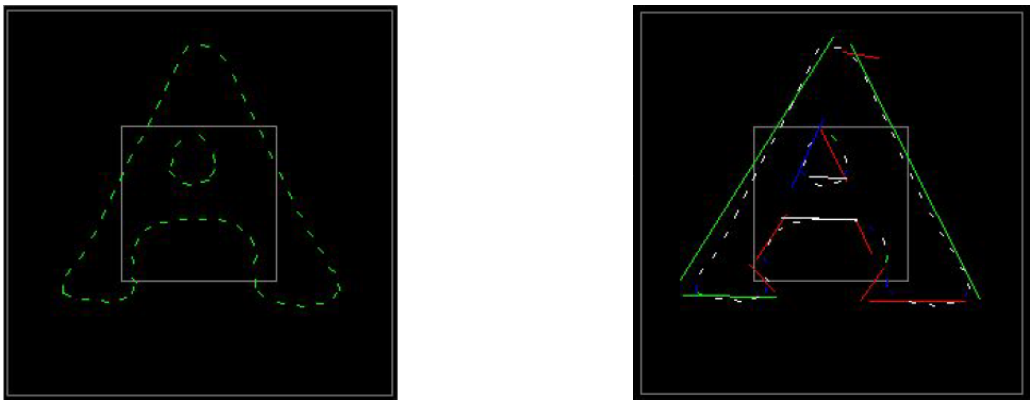


Abbildung 16: Beispiel zur Zerlegung eines Segments in viele kleine (3D-)Features von einheitlicher Länge (links), welche dann mit der besten Übereinstimmung überlappt dargestellt werden (rechts).[46]

Glyphen-Erkennung

Nach der Lokalisierung des Textes im Eingabemedium und formatgerechter Zerteilung in Wörtern, folgt die eigentliche Erkennungsphase der einzelnen Zeichen. Zu allererst wird die Proportionalität der Zeichen überprüft. Im Falle einer sog. **Festschrittschrift** (*Monospace*) kann bereits zu Beginn eine nahezu perfekte Segmentierung vorgenommen werden; andernfalls wird der Umriss an dünnen Stellen oder Berührpunkten zerteilt.

Ein neu hinzuzufügendes Zeichen wird für jede Schriftart zuerst durch eine polygonale Annäherung beschrieben, mit welcher dann das Neuronale Netz trainiert wird. Die so extrahierten Features werden dabei durch einen 4D-Vektor[45] repräsentiert, bestehend aus der Position (x, y), der Richtung (θ) und zuletzt der Länge (als Vielfaches der Feature-Größe). Für aus unbekannten Glyphen extrahierte Features wird hingegen nur ein 3D-Vektor aus Position und Richtung verwendet, wobei diese jedoch von geringer Größe im Vergleich zur vollen Glyphe sind.

Mit Hilfe von Vektorgeometrie und der Länge des Umrisses wird dann die Wahrscheinlichkeit für eine Übereinstimmung berrechnet. Aus Geschwindigkeitsgründen wird wie bei den Histogrammen auf einen segmentierten Feature-Raum der Größe 24^3 zurückgegriffen, wobei jede Zelle eine Menge an Gewichten fürsprechend für ein bestimmtes Zeichen stehen. Annehmbarer Weise beschleunigt es sich, sobald ein bestimmtes Alphabet sowie eine Schriftart als konstant angesehen werden können. Abbildung 16 zeigt ein Beispiel.[46]

[45] Dargestellt als Grund für den Namen *Tesseract*
[46] Quelle: Tesseract[3]

Worterkennung

Die so zunächst ,,vermuteten“ Zeichen werden anschließend in einem Segmentation-Graph, wie er in Anhang VI zu sehen ist, angeordnet, wobei die Zeichen absteigend nach ihrer Wahrscheinlichkeit angeordnet sind. Zudem handelt Tesseract nach dem Prinzip *chop on demand, merge on demand* (teile wenn nötig, verbinde wenn nötig) was so viel bedeutet wie, dass miteinander verbundene Segmente nur zerteilt (oder umgekehrt verbunden) werden, wenn das Resultat einen gewissen Grenzwert nicht unterschreitet. Unterstützt wird dieser Prozess von einem parallel abgesuchten Sprachenmodell, welches aus einem Wörterbuch, NGramm-Parser und Symbolparser besteht. Bestimmte Zeichen, wie Ligaturen oder Diakritika, können ausschlaggebend für die Auswahl des richtigen Modells sein. Andernfalls muss inkrementell über Treffer in den beteiligten Wörterbüchern die Auswahlpriorität schrittweise angepasst werden. Die Bewertung, und damit Verlässlichkeit, eines Wortes hängt nun von mehreren Faktoren ab:

- Zuversichtlichkeiten der Segmente
- Zeichenabstände: Vereinzelt zu kleiner/großer Segmentabstand
- Abweichungen in der Höhe verglichen mit der Vermutung; z. B. ein ‘i’-Punkt in einer ‘m’-Glyphe (siehe rechts)
- Länge des Wortes in einem Einheitsmaß
- Gesamtlänge der Umrisse aller Segmente

In mehreren Schritten wird so nach und nach das jeweils schlechteste Segment geteilt oder verbunden, wie in den Abbildungen 17 & 18 gezeigt.[48]

Abbildung 17
Mounm → Mountain[47]

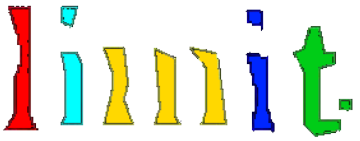


Abbildung 18
Wort ,,limit“

[47] Entnommen aus Tesseract[4], leicht modifiziert für Kontrast.
[48] Quelle: Tesseract[4]

7 Abschluss-Evaluation

```
def show_box(img):
  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  faces = face_cascade.detectMultiScale(gray, 1.3, 5)
  for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
  pos_frame = str(cap.get(cv2.CAP_PROP_POS_FRAMES))
  tstr = faceStore + pos_frame + '.jpg'
  cv2.imwrite(tstr,img)
```

	Pos.	Neg.
Hit	73 /0	7/0
No Hit	188/47	XXX

Abbildung 19: Confusion matrix für Haar-Kaskaden: Ein kurzer Testlauf über 100 Frames (in 25fps = 4s) des Testvideos, mit einer Unterscheidung nach 53 Frames, da dort ein Szenenwechsel stattfand.[49]Die Zahlen in der Tabelle stehen für ,,Anzahl Rahmen“, daher ist TN mit XXX markiert, da die Zahl der korrekterweise verworfenen Stellen unbekannt ist.

```
faces = facer.getAllFaceBoundingBoxes(img)
for face in faces:
    cv2.rectangle(img,(face.left(),face.top()),
        (face.right(),face.bottom()),(255,0,0),2)
```

	Pos.	Neg.
Hit	224/47	2/0
No Hit	38/0	XXX

Abbildung 20: Confusion matrix für HOG-Features: Identische Bedingungen wie zuvor. Beide haben Schwierigkeiten eine Person zu erkennen, wenn das zweite Auge verdeckt ist (siehe Anhang VIII)

Die Gründe weshalb Viola-Jones nicht zur Auswahl steht, wurden bereits ausführlich in Abschnitt 3.5 erläutert. Obgleich die Haar-Kaskaden eine moderne Implementierung in OpenCV besitzen, ist die Durchführung bedingt durch den Algorithmus immer noch schwammig.

Wie in Abbildung 19 zu sehen ist, sind die Resultate nicht verwertbar. Die Trefferrate ($[TP/Pos]$) ist mit 23% sehr gering, dafür ist die Analyse in etwas unter sechs Sekunden beendet. Im Vergleich dazu, HOG aus dlib in Abbildung 20, benötigt zwar fünfzig Sekunden, liefert dafür aber akkurate Ergebnisse mit einer Trefferrate von 89%. Wenn man nun ausnutzt, dass

1. Aufnahmegeräte schneller aufnehmen als sich Menschen aus eigener Kraft bewegen und wahrnehmen können;
2. Menschen und Aufnahmegerät nicht in ununterbrochener Bewegung sein werden;
3. der ursprüngliche Verwendungszweck des Videomaterials gewisse Qualitätsmerkmale garantiert, u. a. Reportagen, Interviews, etc.;

[49]Ein exemplarisches Bild aus beiden Szenen ist in Anhang VII zu sehen.

4. und sich die Umgebung daher in einer für Menschen erträglichen Geschwindigkeit ändert (langsam, kurz und schnell, selten abrupt);
5. eine Person nur einmal erkannt werden muss, um sie als ,,im Video vorhanden“ zu markieren.

. . . dann kann es als ausreichend betrachtet werden, wenn nur jedes fünfte Frame analysiert wird. Zudem lässt sich mit einer einfachen Bildsumme die Veränderung des Frames zum zuletzt betrachteten Bild vergleichen, sodass mit einem geeigneten Grenzwert die Anzahl weiter reduziert werden kann. Bei einem (für das Testsegment ausreichenden) Wert von 1.500.000 wurden nur 10 der 100 Frames in $\tilde{4}.24$ s[50] analysiert, wobei immer noch jede Person mindestens einmal erkannt wurde. Das Verarbeiten des vollständigen Videos (2:25 min) benötigte hierbei nur 2:50 min, ebenfalls ohne dedizierte Hardware zu nutzen. All dies hängt natürlich von der zu Grunde liegenden Detektions-Technologie ab, daher folgen noch einige Alternativen zu (reinem) HOG.[51]

Die Entdeckung nach Zhu *et al.*[26] bewies zwar, dass HOG in Kombination mit Integral Images, AdaBoost und Cascading Classfiers eine Geschwindigkeitssteigerung von 70% bei gleichbleibender Qualität hervorbringen kann. Da dieses Verfahren jedoch patentiert ist[52][53], ist dies keine verwendbare Alternative.

Anzumerken wäre, dass die Algorithmen *Scale-invariant feature transform* (SIFT)[44] und dessen Erweiterung *Speeded up robust features* (SURF)[45] zwar ebenfalls zu den besten des Bereichs zählen, aber eher im 3D-Bereich anzuwenden sind. SIFT arbeitet mit Punktwolken, während SURF die Haar-Wavelets und Integral Images von VJ wieder aufleben lässt.

Im Bereich der Texterkennung wird schnell deutlich, dass es keine vergleichbare Alternative zu Google's Tesseract gibt. Das Projekt ist OpenSource und in C++ geschrieben. Da es jedoch eine Konsolen-Applikation ist, kann es von jeder Programmiersprache angesprochen und genutzt werden. Zudem besitzt es eine ausgiebige Datenbank gespeist von jahrelangen Nutzen, nicht zuletzt durch Google's weitere Webservices.

[50] Die Geschwindigkeit läse sich sicherlich noch weiter steigern wenn es auf einem Server und mit Parallelität verarbeitet wird.

[51] Der Autor von dlib wünscht bei Nutzung seiner Arbeit folgende Referenz [43] einzubinden, was hiermit getan wurde.

[52] siehe https://www.google.com/patents/US20070237387

[53] Eine kurze Kontextsuche beim Patentamt ergibt Zusatzpatente in China, Japan, EU(EPO), und zuletzt weltweit (WIPO).

8 Implementierung

Wie bereits in Abschnitt 1.1 erwähnt, ist die Arbeit Teil eines größeren Projektes, mit dem ein Video verschlagwortet werden soll. Zu diesem Zweck existiert ein Rahmenprogramm, bei welchem ein eingelesenes Video Frame für Frame angefragt werden kann. Weiterhin ist es zuständig für das Zusammentragen der Informationen aus den verschiedenen Analysemodulen. Eines dieser Module soll die Gesichtserkennung, welche Bestandteil dieser Arbeit ist, sein.

Basierend auf der Evaluation in Abschnitt 7 wurde sich für HOG[54] entschieden, da es eine zufriedenstellende Implementierung in dlib[43] besitzt. Für die Texterkennung wurde Tesseract verwendet, da es, wie bereits erwähnt, das momentan beste auf dem Markt hierfür ist. Zudem ist die Verwendung kostenlos und OpenSource. Im Folgenden werden die einzelnen Schritte des Programmablaufs näher beschrieben:

1. Das Video wird eingelesen, woraufhin die einzelnen Frames von dem Rahmenprogramm angefragt werden können.
2. Alle fünf Frames wird eine Pixelsumme berechnet. Ist die absolute Differenz dieses Wertes von dem letzten gespeicherten Wert unter einem festen Grenzwert (hier: 1.5 mio durch Testen), wird es übersprungen. Die Gründe hierfür wurden bereits in Abschnitt 7 erläutert.
3. Im Falle, dass es über dem Grenzwert liegt (oder dies das erste Frame des Videos ist), wird die Pixelsumme intern gesichert und die eigentliche Analyse startet.
4. Es wird eine Funktion des Rahmenprogramms aufgerufen, die feststellt, ob eine Bauchbinde (siehe Anhang IV) vorhanden ist. Aktuell ist diese Funktion nur ein Mock[55], welcher auf einer voreingestellten Stelle nach einer solchen Bauchbinde sucht. Diese Funktion gibt entweder die Position der Box oder einen Null-Wert zurück, woraufhin zu Schritt 2 gesprungen wird.
5. Falls eine Bauchbinde gefunden wurde, wird die Box an der angegebenen Stelle ausgeschnitten und an Tesseract übergeben, welches den Vor- und Nachnamen daraus extrahiert.
6. Als Nächstes wird die Gesichtserkennung von HOG aufgerufen, welche zuerst alle Gesichter sucht und anschließend aus den gefundenen Bounding-Boxen die größte

[54] Histogram of Oriented Gradients
[55] siehe Glossary

zurückliefert. Diese wird dann zusammen mit dem oben gefundenen Namen als Trainingsdatum abgespeichert.

7. Die Schritte 2-6 werden wiederholt bis das Ende des Videos erreicht wurde.

Abschließend zeigt Abbildung 21 das Rahmenprogramm mit dem in dieser Arbeit entwickelten Gesichtserkennungsmodul in Aktion. Wie oben beschrieben wird nur das größte Gesicht angezeigt, während die Person im Hintergrund zwar erkannt aber unmarkiert bleibt.

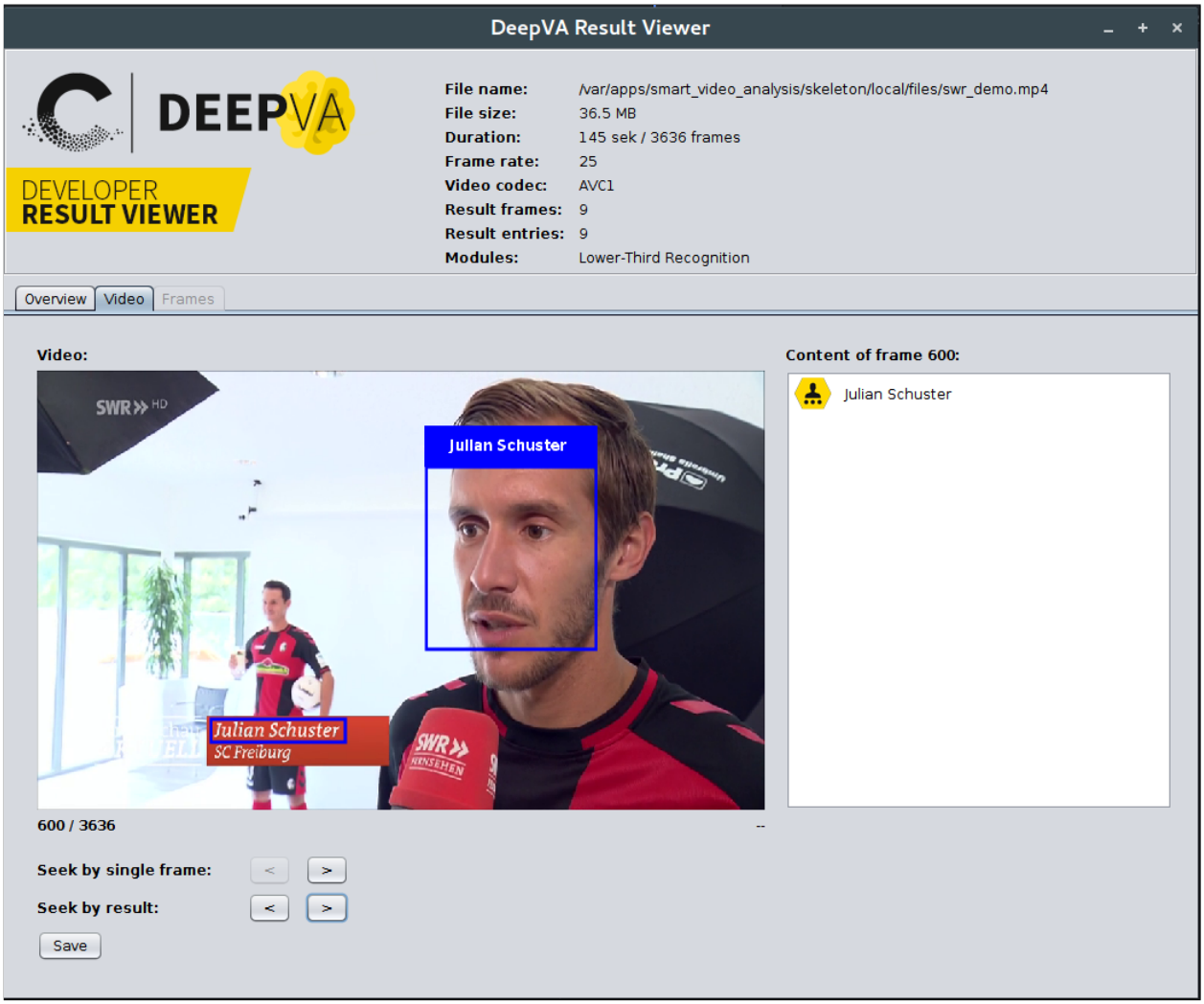


Abbildung 21: Rahmenprogramm mit fertigem Modul

8.1 Fazit

Das in dieser Arbeit erstellte Modul für die Verschlagwortung von Videos im Bereich der Gesichts- und Texterkennung erfüllt seine wesentlichen Aufgaben.

8.2 Ausblick

Das Teilgebiet der Objekterkennung wird auch in den nächsten Jahren ein vielversprechendes Forschungsthema bleiben. Insbesondere Deep Learning verspricht noch einige potenzielle Durchbrüche in der Zukunft, da es bereits jetzt als der nächste ,,große Durchbruch“ bezeichnet wird. Die aktuellen Ausblicke in der Gesichtserkennung richten sich darauf, weiterreichende Informationen, u. a. Emotionen oder Herkunft, auszulesen.[56]

Über die Bachelorarbeit hinaus wird es sich anbieten, die so gesammelten Trainingsdaten zum Aufbau eines neuronalen Netzes zu verwenden. Des Weiteren soll der aktuell durch ein Mock repräsentierte Ansatz zum Finden der Bauchbinden durch einen konkreten Detektionsalgorithmus ersetzt werden. Ein Ziel des Endproduktes wäre es, die eingelesenen Videos nicht länger als ihre eigentliche Laufzeit zu verarbeiten, was sich als Herausforderung bei höheren Frameraten erweist.

[56] http://blog.affectiva.com/announcing-inaugural-emotion-ai-summit-event-at-mit-media-lab

Quellenverzeichnis

[1] Tombone's Computer Vision Blog. *Deep Learning vs Machine Learning vs Pattern Recognition*. März 2015. URL: http://www.computervisionblog.com/2015/03/deep-learning-vs-machine-learning-vs.html.

[2] H. G. Barrow und J. M. Tenenbaum. "Interpreting Line Drawings As Three-dimensional Surfaces". In: *Artif. Intell.* 17.1-3 (Aug. 1981), S. 75–116. ISSN: 0004-3702. DOI: 10.1016/0004-3702(81)90021-7. URL: http://dx.doi.org/10.1016/0004-3702(81)90021-7.

[3] J. Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), S. 679–698. ISSN: 0162-8828. DOI: 10.1109/TPAMI.1986.4767851.

[4] Linda G. Shapiro und George C. Stockman. *Computer Vision*. Pearson, 2001. ISBN: 0130307963.

[5] Lawrence G Roberts. "Machine perception of three-dimensional solids". Diss. Massachusetts Institute of Technology, 1963.

[6] Irwin Sobel. *History and Definition of the so-called SSobel Operator"*. Feb. 1968,2015. URL: https://www.researchgate.net/publication/239398674_An_Isotropic_3_3_Image_Gradient_Operator.

[7] Linda G. Shapiro und George C. Stockman. *Computer Vision*. Pearson, 2001. ISBN: 0130307963. URL: https://www.amazon.com/Computer-Vision-Linda-G-Shapiro/dp/0130307963?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0130307963.

[8] Chris Harris und Mike Stephens. "A combined corner and edge detector". In: *In Proc. of Fourth Alvey Vision Conference*. 1988, S. 147–151.

[9] Tony Lindeberg. *Scale-Space*. John Wiley & Sons, Inc., 2007. ISBN: 9780470050118. DOI: 10.1002/9780470050118.ecse609. URL: http://dx.doi.org/10.1002/9780470050118.ecse609.

[10] Stephen M. Smith und J. Michael Brady. "SUSAN—A New Approach to Low Level Image Processing". In: *International Journal of Computer Vision* 23.1 (1997), S. 45–78. ISSN: 1573-1405. DOI: 10.1023/A:1007963824710. URL: http://dx.doi.org/10.1023/A:1007963824710.

[11] Tony Lindeberg. "Edge Detection and Ridge Detection with Automatic Scale Selection". In: *International Journal of Computer Vision* 30.2 (1998), S. 117–156. ISSN: 1573-1405. DOI: 10.1023/A:1008097225773. URL: http://dx.doi.org/10.1023/A:1008097225773.

[12] Karl Pearson F.R.S. "LIII. On lines and planes of closest fit to systems of points in space". In: *Philosophical Magazine* 2.11 (1901), S. 559–572. DOI: 10.1080/14786440109462720. eprint: http://dx.doi.org/10.1080/14786440109462720. URL: http://dx.doi.org/10.1080/14786440109462720.

[13] Harold Hotelling. "Analysis of a complex of statistical variables into principal components." In: *Journal of educational psychology* 24.6 (1933), S. 417.

[14] H. Bourlard und Y. Kamp. "Auto-association by multilayer perceptrons and singular value decomposition". In: *Biological Cybernetics* 59.4 (1988), S. 291–294. ISSN: 1432-0770. DOI: 10.1007/BF00332918. URL: http://dx.doi.org/10.1007/BF00332918.

[15] Mrinal K Mandal, F Idris und Sethuraman Panchanathan. "A critical evaluation of image and video indexing techniques in the compressed domain". In: *Image and Vision Computing* 17.7 (1999), S. 513–529.

[16] Moving Picture Experts Group. *MPEG-7: Visual Descriptors*. Apr. 2008. URL: http://mpeg.chiariglione.org/standards/mpeg-7/visual.

[17] Patrice Y. Simard, Dave Steinkrau und Ian Buck. "Using GPUs for Machine Learning Algorithms". In: *2013 12th International Conference on Document Analysis and Recognition* 00 (2005), S. 1115–1119. ISSN: 1520-5263. DOI: doi.ieeecomputersociety.org/10.1109/ICDAR.2005.251.

[18] Dan C Ciresan u. a. "Flexible, high performance convolutional neural networks for image classification". In: *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*. Bd. 22. 1. Barcelona, Spain. 2011, S. 1237.

[19] Chris J. Maddison u. a. "Move Evaluation in Go Using Deep Convolutional Neural Networks". In: *CoRR* abs/1412.6564 (2014). URL: http://arxiv.org/abs/1412.6564.

[20] Navneet Dalal und Bill Triggs. "Histograms of oriented gradients for human detection". In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Bd. 1. IEEE. 2005, S. 886–893.

[21] Paul Viola und Michael Jones. "Robust Real-time Object Detection". In: *International Journal of Computer Vision*. 2001.

[22] C. Papageorgiou, M. Oren und T. Poggio. “A General Framework for Object Detection”. In: *International Journal of Computer Vision*. 1998.

[23] Franlin C. Crow. “Summed-Area Tables for Texture Mapping”. In: Bd. 18. 3. Juli 1984, S. 207–212. URL: https://classes.soe.ucsc.edu/cmps160/Fall05/papers/p207-crow.pdf.

[24] P. Burt und E. Adelson. “The Laplacian Pyramid as a Compact Image Code”. In: *IEEE Transactions on Communications* 31.4 (1983), S. 532–540. ISSN: 0090-6778. DOI: 10.1109/TCOM.1983.1095851.

[25] Fatih Porikli. “Integral histogram: A fast way to extract histograms in cartesian spaces”. In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Bd. 1. IEEE. 2005, S. 829–836.

[26] Qiang Zhu u. a. “Fast human detection using a cascade of histograms of oriented gradients”. In: *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. Bd. 2. IEEE. 2006, S. 1491–1498.

[27] Yoav Freund und Robert E. Schapire. *A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting*. 1997.

[28] Balázs Kégl. “The return of AdaBoost.MH: multi-class Hamming trees”. In: *CoRR* abs/1312.6086 (2013). URL: http://arxiv.org/abs/1312.6086.

[29] Jason M Saragih, Simon Lucey und Jeffrey F Cohn. “Face alignment through subspace constrained mean-shifts”. In: *Computer Vision, 2009 IEEE 12th International Conference on*. Ieee. 2009, S. 1034–1041.

[30] Xiangxin Zhu und Deva Ramanan. “Face detection, pose estimation, and landmark localization in the wild”. In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, S. 2879–2886.

[31] Peter N Belhumeur u. a. “Localizing parts of faces using a consensus of exemplars”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.12 (2013), S. 2930–2940.

[32] Tadas Baltrusaitis, Peter Robinson und Louis-Philippe Morency. “Constrained local neural fields for robust facial landmark detection in the wild”. In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2013, S. 354–361.

[33] Shaoqing Ren u. a. “Face Alignment at 3000 FPS via Regressing Local Binary Features”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.

[34] Vahid Kazemi und Josephine Sullivan. “One millisecond face alignment with an ensemble of regression trees”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, S. 1867–1874.

[35] Satya Mallick. *Facial Landmark Detection*. Okt. 2015.

[36] Florian Schroff, Dmitry Kalenichenko und James Philbin. “Facenet: A unified embedding for face recognition and clustering”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, S. 815–823.

[37] Paul Jaccard. *Étude comparative de la distribution florale dans une portion des Alpes et des Jura*. 1901.

[38] Douglas G Altman und J Martin Bland. “Statistics Notes: Diagnostic tests 2: predictive values”. In: *BMJ* 309.6947 (1994), S. 102. ISSN: 0959-8138. DOI: 10.1136/bmj.309.6947.102. eprint: http://www.bmj.com/content/309/6947/102.1.full.pdf. URL: http://www.bmj.com/content/309/6947/102.1.

[39] Brian W Matthews. “Comparison of the predicted and observed secondary structure of T4 phage lysozyme”. In: *Biochimica et Biophysica Acta (BBA)-Protein Structure* 405.2 (1975), S. 442–451.

[40] C. C. Tappert, C. Y. Suen und T. Wakahara. “The state of the art in online handwriting recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.8 (1990), S. 787–808. ISSN: 0162-8828. DOI: 10.1109/34.57669.

[41] M. Sezgin und B. Sankur. “Survey over image thresholding techniques and quantitative performance evaluation”. In: *Journal of Electronic Imaging* 13 (Jan. 2004), S. 146–165. DOI: 10.1117/1.1631315.

[42] R. Smith. “An Overview of the Tesseract OCR Engine”. In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Bd. 2. 2007, S. 629–633. DOI: 10.1109/ICDAR.2007.4376991.

[43] Davis E. King. “Dlib-ml: A Machine Learning Toolkit”. In: *Journal of Machine Learning Research* 10 (2009), S. 1755–1758.

[44] D. G. Lowe. “Object recognition from local scale-invariant features”. In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Bd. 2. 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[45] Herbert Bay, Tinne Tuytelaars und Luc Van Gool. “Surf: Speeded up robust features”. In: *Computer vision–ECCV 2006* (2006), S. 404–417.

Anhang

— Glossar —

Big Data ist die Denkschule des Messens und Analysierens großer Datenmengen, um auf umfangreiche Datenbanken für MachineLearning zurückzugreifen.

Intensität Schwarz-Weiß Wert eines Pixels.

Edge (*Kante*) Intensitätsspitze; siehe Seite 5 (2.1.1).

Corner (*Ecke*) Richtungswechsel einer Kante; siehe Seite 6 (2.1.2).

Ridge (*Klippe*) siehe Seite 7 (2.1.4).

Blob (*Blob*) "Area of interest"; siehe Seite 7 (2.1.3).

Kernel (*Faltungsmatrix*) siehe Seite 6 (4).

Glyph (*Glyphe*)

Die graphische Darstellung eines bestimmten typographischen Elements mit eigener Bedeutung. Dabei beschränkt sich dessen Definition auf die Repräsentation eines bestimmten *Schriftzeichens* in einer bestimmten Schriftart.

Sliding Window

Ein Suchfenster, dass in einer festen Größe über ein Bild fährt, um es ausschnitsweise zu analysieren (bzw. ein Algorithmus der das SW verwendet.)

SVM

Eine **Support Vector Machine** (*Support Vektor Maschiene*) ist ein simples Klassifizierungssystem, welches genutzt wird, um Mehrdimensionale Entscheidungsprobleme durch Vektorgeometrie zu lösen. Anfangs müssen die gewünschten Fixpunkte mit den hervorzubringenden Resultaten trainiert werden. Im Anwendungsfall wird zu jeder weiteren Eingabe dann das Resultat mit der geringsten geometrischen Distanz zurückgegeben.

Mock (*Attrape*)

Ein Programmteil der zu Testzwecken als Platzhalter für das echte Objekt (oder Codeteil) dient.

— Bildquellen —

Bild 3 : http://www.dummies.com/photography/cameras/canon-camera/live-mode-face-detection-settings-on-a-canon-eos-60d

Bild 4 : https://cs.stanford.edu/people/eroberts/courses/soco/projects/1997-98/computer-vision/edges.html

Bild 5 : https://becominghuman.ai/cheat-sheets-for-ai-neural-networks-machine-learning-deep-learning-big-data-678c51b4b463

Bild 10 : https://upload.wikimedia.org/wikipedia/commons/thumb/4/43/Image_pyramid.svg/240px-Image_pyramid.svg.png

Bild 11 : https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78

Bild 15 : https://github.com/tesseract-ocr/docs/blob/master/das_tutorial2016/5LayoutAnalysis.pdf

Hinweis

Alle Bilder, deren Quellenangabe nicht vorliegt, stammen aus eigener Produktion oder dem vom Betreuer zur Verfügung gestellten Testvideo.

— Internetquellen —

Dieser Abschnitt ist für lose Internetquellen gedacht, die keine spezielle Referenz im Dokument beinhalten.

OpenCV http://docs.opencv.org/master/ [und] https://github.com/opencv/opencv

dlib http://dlib.net/imaging.html [und] https://github.com/davisking/dlib

IoU http://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

NN & DL https://medium.com/@ageitgey/ [57]

Patente https://depatisnet.dpma.de/DepatisNet/depatisnet?action=einsteiger

Tesseract https://github.com/tesseract-ocr/docs/blob/master/das_tutorial2016/

[57] Neuronale Netze & Deep Learning

— Übergroße Bilder —

Im folgenden Abschnitt finden sich Bilder die anderweitig zu viel Platz einnehmen würden.

Anhang IV: Vollbild für Bauchbindenbeispiel

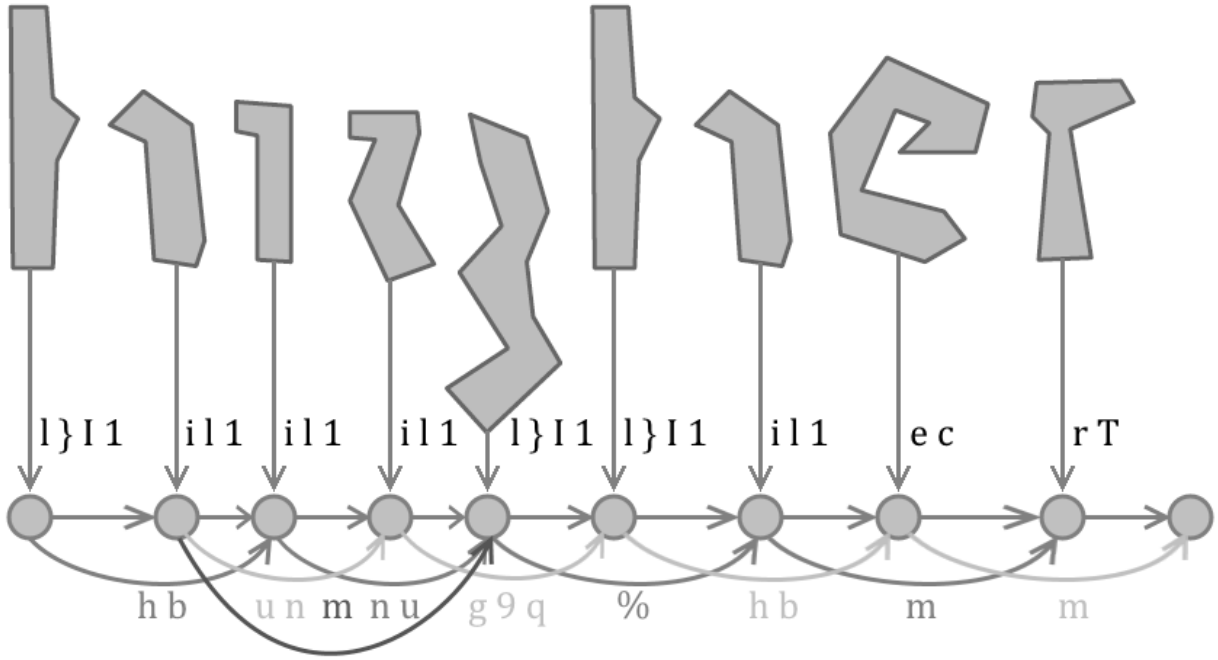
Anhang V: HOG-Features

(a) Vorher

(b) Nachher

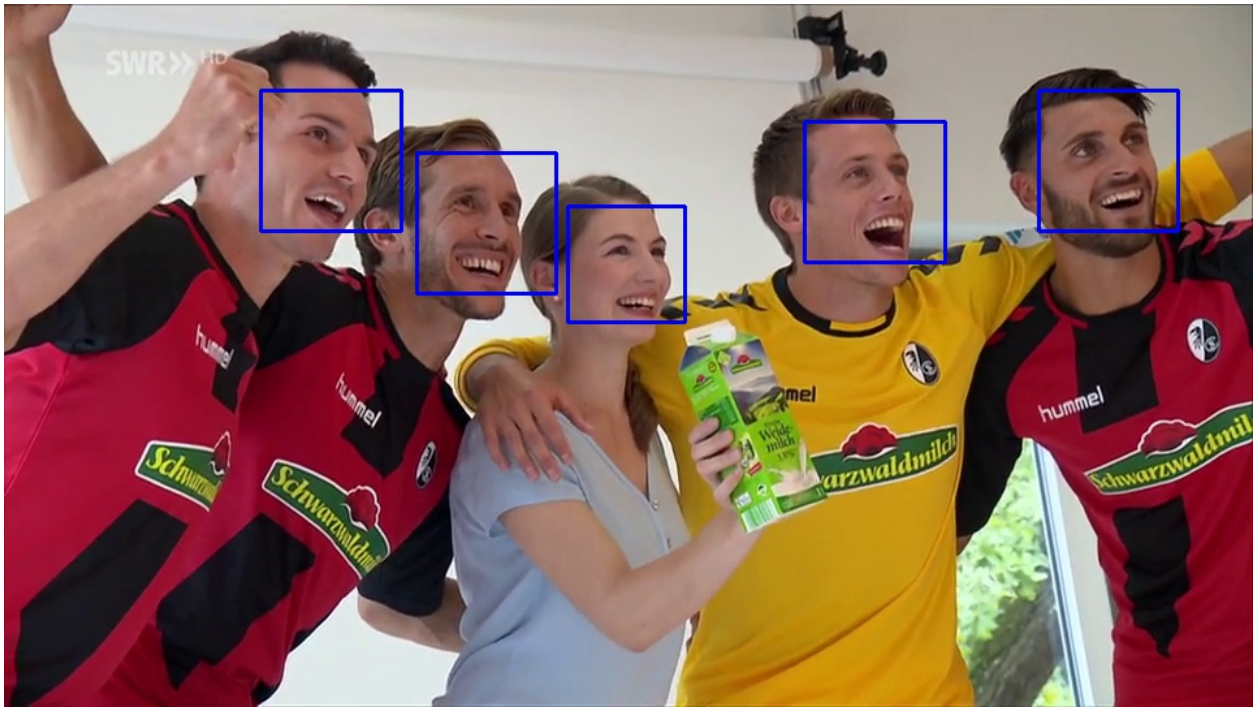
Bildquelle: https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78

Anhang VI: Segmentation-Graph für das Wort **higher** [2-1-2-2-1-1].



Bildquelle: Bild modifiziert von
https://github.com/tesseract-ocr/docs/blob/master/das_tutorial2016/4CharSegmentation.pdf, s.4

Anhang VII: Beispiel-Frames für Abbildung 19 & 20.

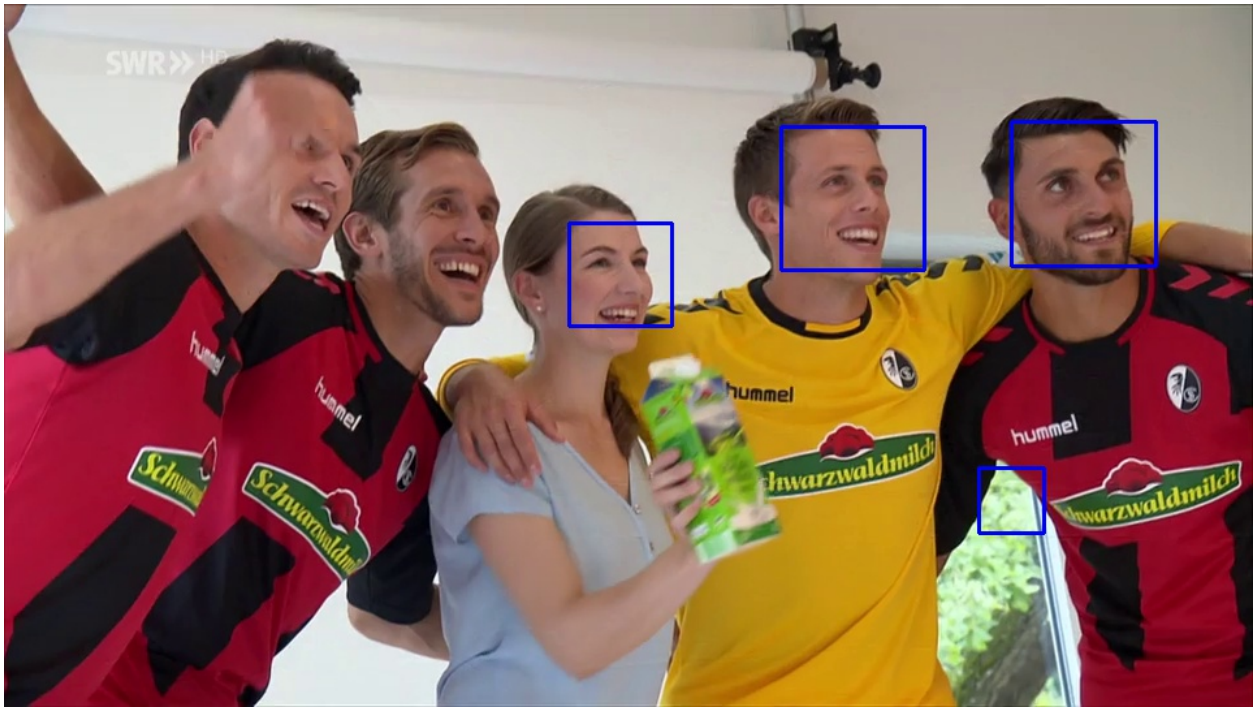


(a) Bild aus Szene 1 (Frames 1-53)

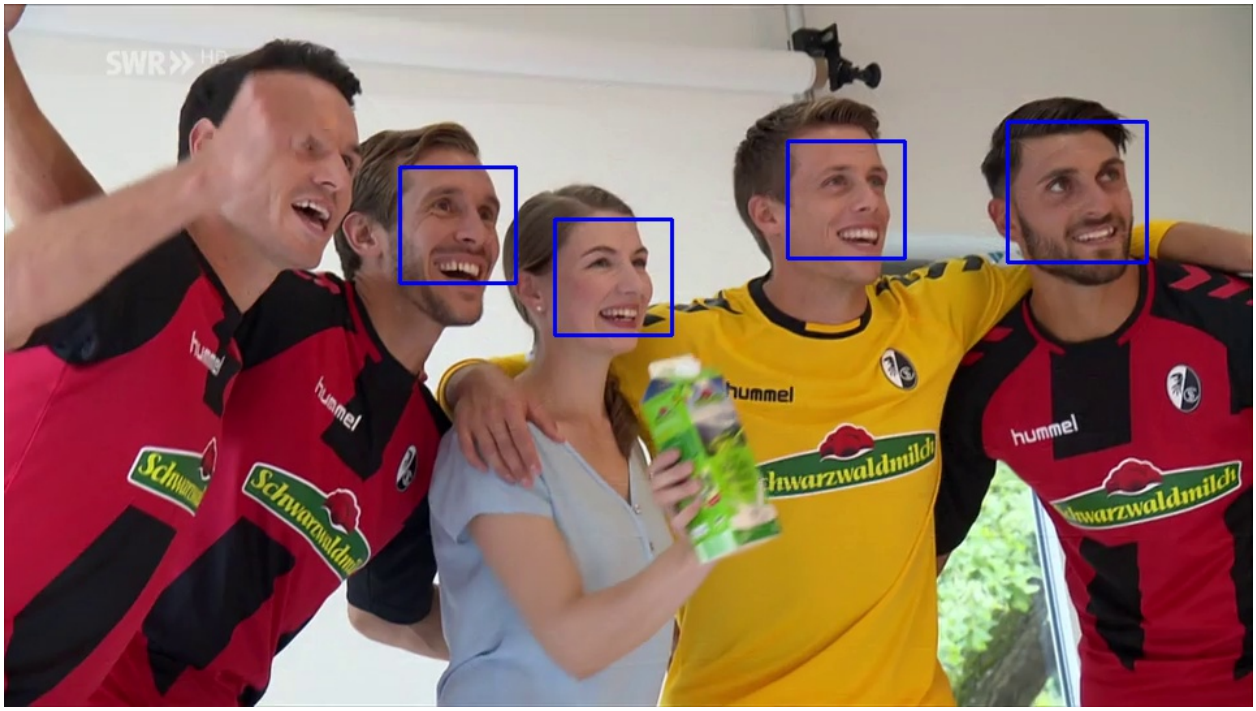
(b) Bild aus Szene 2 (Frames 54-100). Das unscharfe Gesicht der Frau wird ignoriert

Anmerkung: Die Gesichter in (a) und (b) wurden durch HOG erkannt.

Anhang VIII: Beispiel Gesichtserkennung



(a) Beispiel mit Viola-Jones



(b) Beispiel mit Histogram of Oriented Gradients