Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Masterarbeit zur Erlangung des akademischen Grades

Master of Science
Studiengang Informatik

– Sommersemester 2017–

Neuronale Netze und andere Verfahren zur Gesichtserkennung in der Heimautomatisierung

von

Constantin Kirsch

Erstprüfer:	Prof. Dr. Karl Jonas
Zweitprüfer:	Prof. Dr. Marlis von der Hude
Eingereicht am:	29. August 2017

Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbst angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Bonn, den 29. August 2017

Datum Unterschrift

Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich bei der Anfertigung dieser Abschlussarbeit unterstützt und motiviert haben.

Dieser Dank gilt in erster Linie meinem Erstprüfer Prof. Dr. Karl Jonas, der diese Arbeit betreut und ermöglicht hat. Des Weiteren möchte ich den Mitarbeitern und Kommilitonen im Labor C060 der Hochschule Bonn-Rhein-Sieg danken, welche mir stets mit sinnvollen Tipps weiter geholfen haben und zudem als Versuchspersonen bereit standen.

Ganz besonders möchte ich auch Aline Augsburg danken, die viel Zeit in die Korrektur dieser Arbeit gesteckt und mich unermüdlich motiviert sowie moralisch durch alle Tiefphasen hindurch unterstützt hat.

Nicht zuletzt danke ich meinen Eltern, dafür das sie mich während meines gesamten Studiums unterstützt haben.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AI	Artificial Intelligence
AIO	All In One
API	Application Programming Interface
AR	Augmented Reality
BRSU	Bonn-Rhine-Sieg University
CNN	Convolutional Neural Network
CPU	Central Processing Unit
FE	Feature Extractor
FERET	Facial Recognition Technology
GPU	Graphics Processing Unit
HOG	Histogram of Oriented Gradients
HSD	Honestly Significant Difference
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
JPG	Joint Photographic Experts Group
kNN	k-Nearest-Neighbors
LBP	Local Binary Pattern
LDA	Linear Discriminant Analysis
LFW	Labeled Faces in the Wild
MLP	Multi Layer Perceptron
n.a.	nicht anwendbar
ORM	Object Relational Mapper
PCA	Principal Component Analysis
PIE	Pose, Illumination, and Expression
ReLU	Rectified Linear Unit
REST	REpresentational State Transfer
RGB	Rot-Grün-Blau
SGD	Stochastic Gradient Descent
SLP	Single Layer Perceptron
SVD	Singular Value Decomposition
SVM	Support Vector Machine
URL	Uniform Resource Locator

1. Einleitung

Die Fähigkeit einzelne Gesichter erkennen und unterscheiden zu können, ist für die meisten Menschen etwas vollkommen Alltägliches und Unbewusstes. Um Systeme zu entwickeln, die ebenfalls zur Gesichtserkennung in der Lage sind, war jahrzehntelange Forschung nötig, welche noch nicht abgeschlossen ist. Dabei wurden die biologischen Abläufe im Gehirn untersucht und diese als Inspiration zur Entwicklung von Verfahren zur Gesichtserkennung genutzt. Heutzutage nutzen digitale Plattformen sowie Behörden und Unternehmen automatische Gesichtserkennungssysteme für eine Vielzahl an Zwecken. Dazu gehören die Identifizierung und Verifikation an Geldautomaten, Flughäfen, Gebäuden, gegenüber Banken und viele weitere Anwendungszwecke [1]. Anbieter solcher Dienste haben dabei die Möglichkeit, auf umfangreiche Gesichtsdatenbanken und stetig wachsende Rechenleistung zuzugreifen. Beides war lange Zeit ein limitierender Faktor für eine effiziente Gesichtserkennung. Auch heute sind dies die Hauptgründe, welche eine Nutzung dieser Technologie im privaten Umfeld erschweren. Die Heimautomatisierung verbreitet sich aufgrund geringer Kosten und einfacher Nutzung immer mehr und stellt damit einen perfekten Rahmen zur Einbindung von Gesichtserkennungssystemen dar. Es sind zudem automatische Zugangssysteme denkbar, welche das Gesicht einer Person als biometrisches Merkmal nutzen, da es einige Vorteile gegenüber bisherigen biometrischen Merkmalen wie Fingerabdrücke und Iris besitzt [1]. Bis die dabei entstehenden Hindernisse überwunden sind, gibt es dennoch Anreize, eine automatische Gesichtserkennung in der Heimautomatisierung zu ermöglichen. Bewohner könnten über eintreffende Gäste informiert werden oder auch erfahren, ob sich unbekannte Personen vor der Tür befinden. Ein solches System würde nicht nur Komfort bieten. Menschen, die älter sind oder unter bestimmten Krankheiten leiden, können Schwierigkeiten haben, Personen zu erkennen und zu unterscheiden. Für diese Menschen steigert eine automatische Gesichtserkennung, sei es vor der Haustür oder im Alltag, die Lebensqualität, ermöglicht mehr Selbstständigkeit und bietet Sicherheit.

1.1. Problemstellung

Diese Arbeit befasst sich mit der Frage, ob mit heutigen Möglichkeiten eine automatische Gesichtserkennung, also ohne menschliche Interaktion, in der Heimautomatisierung möglich ist. Üblicherweise sind dafür große Mengen an Daten und Rechenleistung nötig. Fortschritte im Bereich der neuronalen Netze, sowie immer günstigere und schneller werdende Rechnersysteme könnten eine Umsetzung dennoch ermöglichen. Zudem besteht für private Wohnräume nicht der Anspruch, beliebig große Mengen an Personen zu erkennen. Viele Dienstleister stellen Schnittstellen im Internet bereit, um eine

Gesichtserkennung von mehreren Personen zu ermöglichen. Die Nutzung solcher Systeme bringt jedoch einige Nachteile mit sich. Zum einen ergibt sich dadurch der Zwang einer Internetverbindung, über welche die Bilder und Ergebnisse gesendet werden müssen. Zum anderen entstehen durch solche Dienste häufig Kosten. Der größte Nachteil besteht jedoch in der Datensicherheit. Es müsste gewährleistet sein, dass der Dienstleister die gespeicherten Personeninformationen, sowie die eventuell erhobenen Metadaten, vertraulich behandelt. Ansonsten besteht die Gefahr, persönliche Informationen der Nutzer und sämtlicher erfasster Personen an Dritte weiterzugeben. Daher besteht eine der Einschränkungen für ein System zur automatischen Gesichtserkennung in der Heimautomatisierung darin, nur auf lokale Ressourcen zurückzugreifen. Dadurch ergeben sich weitere Restriktionen. Die genutzte Hardware sollte dem entsprechen, was aktuell oder in naher Zukunft für die meisten Haushalte als üblich betrachtet werden kann.

Zu diesem Zweck sollen in dieser Arbeit unterschiedliche Verfahren zur Gesichtserkennung betrachtet werden. Diese sollen sowohl aus bereits erprobten als auch moderneren Verfahren bestehen. Im Speziellen soll die Nutzbarkeit von neuronalen Netzen für diesen Anwendungszweck untersucht werden. Die Daten, welche einem Gesichtserkennungssystem zugrunde liegen, sowie die Erfassung dieser, werden ebenso betrachtet. Zuletzt soll die Machbarkeit mit einem Softwareprototyp gezeigt werden, welcher eine Gesichtserkennung in der Heimautomatisierung ermöglicht.

1.2. Aufbau der Arbeit

Zu Beginn dieser Arbeit werden im Stand der Forschung die Grundlagen und Hintergründe der in dieser Arbeit genutzten Technologien und Verfahren erläutert. Im Anschluss werden mehrere Gesichtserkennungsverfahren implementiert und hinsichtlich ihrer Genauigkeit getestet und verglichen. Aufgrund der Ergebnisse dieser Tests werden ein oder mehrere Verfahren ausgewählt, welche im Anschluss zu Implementierung eines Prototyps genutzt werden. Dieser Prototyp soll letztlich die praktische Umsetzbarkeit für ein automatisches Gesichtserkennungssystem zeigen, sowie die nötigen Rahmenbedingungen und eventuellen Einschränkungen aufzeigen. Zuletzt erfolgt eine Zusammenfassung dieser Arbeit und der Ergebnisse, sowie ein Ausblick auf mögliche zukünftige Arbeiten.

2. Stand der Forschung

Um die in der praktischen Umsetzung dieser Arbeit verwendeten Konzepte und Technologien zu verstehen, erfolgt in diesem Kapitel eine Einführung in den Stand der Forschung. Im weiteren Verlauf dieser Arbeit werden zur Abgrenzung mehrschichtige neuronale Netze als tiefe Verfahren bezeichnet, alle weiteren als flache Verfahren. Die aktuelle Forschung im Bereich der Gesichtserkennung schreitet derzeit sehr schnell voran und ist breit gefächert, weswegen eine vollständige Auflistung und Gegenüberstellung aller Möglichkeiten zur Gesichtserkennung den Umfang dieser Arbeit überschreiten würde. Daher wird sich im Folgenden nur auf einige für die vorliegende Arbeit relevanten Grundlagen und Verfahren beschränkt. Eine grundlegende Disziplin für die moderne Gesichtserkennung ist das **Machine Learning**, insbesondere die verschiedenen Lernmethoden, weswegen zuerst näher auf die Konzepte und Inhalte dieses Forschungsgebiets eingegangen wird.

2.1. Machine Learning

Der Prozess des Lernens versucht aus vorhandenen Erfahrungswerten für die Zukunft verwendbare Informationen zu ziehen. Im Kontext des maschinellen Lernens basieren die Erfahrungswerte auf gesammelten Daten, aus denen Vorhersagen für zukünftige Daten getroffen werden. Respektive soll ein System, welches Machine Learning anwendet, die gegebenen Daten nutzen, um eine Generalisierung zu finden, welche auch auf unbekannten Daten (größtenteils) präzise Ergebnisse liefert [2].

Während Speicherplatz in den Anfängen der digitalen Datenverarbeitung noch teuer war, gibt es diesen heutzutage vergleichsweise im Überfluss. Auch aus diesem Grund werden immer mehr Daten in allen Bereichen der Forschung und des alltäglichen Lebens digital erhoben und gespeichert. Um aus diesen Datenmengen Informationen zu ziehen und Zusammenhänge zu erlernen und damit das sogenannte Lernproblem zu lösen, gibt es verschiedene Ansätze, abhängig davon was für Daten vorliegen und welches Ergebnis gewünscht ist.

2.1.1. Lernmethoden

Um ein Lernproblem zu lösen, also aus vorliegenden Daten zu lernen, werden grundsätzlich drei verschiedene Lernmethoden definiert. **Supervised Learning**, **Unsupervised Learning** und **Reinforcement Learning**. Im Folgenden soll näher auf diese Methoden eingegangen werden [3, S. 694ff].

- **Supervised Learning**
  Sind zu den Eingangsdaten die Ausgangsdaten, welche ein System ermitteln soll, bekannt, wird die Lernmethode als Supervised Learning, also überwachtes Lernen, bezeichnet. Das Ziel ist es, eine Funktion h, auch Hypothese genannt, zu finden, welche die Eingangsdaten möglichst gut auf die Ausgangsdaten abbildet. Neben den Trainingsdaten soll diese Funktion selbstverständlich auch auf Testdaten und später im realen Einsatz möglichst genaue Ergebnisse liefern. Ein Trainingsdatensatz N besteht dabei aus Paaren von Eingangs- und Ausgangsdaten $\{(x_1,y_1),(x_2,y_2),...,(x_N,y_N)\}$. Die Ergebnisdatenmenge Y kann als Ergebnis einer unbekannten Funktion $Y = f(X)$ betrachtet werden. Wenn die möglichen Werte für Y aus einer endlichen Menge von festen Werten bestehen, wird der Fall des Lernproblems als Klassifizierungsproblem bezeichnet. Ein einfaches Beispiel ist ein System zur Wettervorhersage. Die Eingangsdaten bestehen aus Informationen über Luftdruck, Windgeschwindigkeit und weiteren Faktoren. Die Ergebnismenge besteht dabei aus konkreten Werten wie sonnig, regnerisch und bewölkt. Diese Zuordnungen werden auch als **Label** oder Kennzeichnung bezeichnet. Sind die Werte aus Y hingegen beliebige Zahlen, beispielsweise aus $\mathbb{R}$, wird von Regression gesprochen. Ein typisches Beispiel hierfür ist die Vorhersage des Kaufpreises einer Wohnung oder eines Hauses anhand der Zimmeranzahl, Grundfläche und weiteren Angaben. Dabei ist zu beachten, dass der ermittelte Wert nicht exakt dem realen Wert entsprechen muss, sondern lediglich einen erwarteten oder durchschnittlichen Wert darstellt.

  Algorithmen zur Gesichtserkennung verwenden üblicherweise ein Bild mit einem Gesicht als Eingangswert und sollen daraus die zugehörige Person aus einer festen Menge von Personen ermitteln. Daher fällt die automatische Gesichtserkennung in den Bereich der Klassifizierung. Wird jedoch nicht nur der Algorithmus, sondern der in Kapitel 1.1 erwähnte und in Kapitel 4 beschriebene Prototyp betrachtet, ist zu beachten, dass für den Endanwender keine in sich geschlossenen Trainings-, Test- und Produktivphasen existieren. Das System soll in Betrieb genommen und durch einen durchgehenden Lernprozess, welcher manuell vom Benutzer unterstützt wird, immer weiter verbessert werden. Diese Anforderungen sind teilweise mit Reinforcement Learning umsetzbar, welches weiter unten erläutert wird.

- **Unsupervised Learning**
  Der Gegensatz zu dem eben beschriebenen überwachten Lernen stellt das unüberwachte Lernen, also Unsupervised Learning dar. Wird ebenfalls ein Trainingsdatensatz N betrachtet, gibt es lediglich die Eingangsdaten $(x_1,x_2,...,x_N)$, die Ausgangsdatenmenge Y ist hier nicht bekannt. Aus diesen Daten müssen Informatio-

nen über die Struktur, beziehungsweise der den Daten zugrunde liegenden Verteilung gezogen werden. Daraus lässt sich ein Modell bilden, welches eine nach Möglichkeit zu den Eingangsdaten nahezu identische Verteilung aufweist. Dies objektiv nachzuweisen ist nicht ohne Weiteres möglich, da keine Informationen über die zu erwartenden Ausgangsdaten vorliegen. Lediglich mit Hilfe von heuristischen Methoden lassen sich Aussagen über die Qualität der Ergebnisse von unüberwachten Lernmethoden ziehen [4, S. 487]. Eines der bekanntesten Aufgabengebiete des unüberwachten Lernens ist das **Clustering**, bei dem eine Menge von Daten in verschiedene Gruppen eingeteilt werden sollen, ohne vorherige Kenntnisse, wie beispielsweise Anzahl, über diese Gruppen. Im Vorfeld können höchstens Annahmen getroffen werden, wie viele unterschiedliche Gruppen zu erwarten sind.

- **Reinforcement Learning**
  Bestärkendes Lernen geht davon aus, dass ein lernendes System ohne jegliche Vorkenntnisse über Eingangs- und Ausgangsdaten in einer unbekannten Umgebung agieren muss. Durch Interaktion mit der Umgebung erhält es Eingangsdaten und kann basierend auf diesen Entscheidungen treffen. Zu Beginn sind diese komplett zufällig. Bei einem Klassifizierungsproblem mit N verschiedenen Klassen beträgt die anfängliche Wahrscheinlichkeit einer richtigen Entscheidung $1/N$. Richtige Entscheidungen haben eine Belohnung zur Folge, falsche eine Bestrafung oder das Auslassen einer Belohnung. Das System lernt in Folge dieser Belohnungen die Entscheidungen zu treffen, mit denen es die erhaltenen Belohnungen maximiert. Der Prototyp, welcher in der vorliegenden Arbeit entwickelt werden soll, könnte je nach Ausprägung in diese Kategorie fallen. Anhand der erfassten neuen Gesichter hat das System die Möglichkeit immer neue Entscheidungen zu treffen. Im Nachhinein kann ein Benutzer das System für richtige und falsche Entscheidungen entsprechend belohnen oder bestrafen. Abhängig davon kann das System intern den zugrunde liegenden Datensatz mit den neuen Daten erweitern und einen neuen überwachten Lernprozess für das eigentliche Modell beginnen.

Zusätzlich kann für jede dieser Methoden zwischen sogenanntem **Online Learning** und **Offline Learning** unterschieden werden. Diese stellen dabei keine eigene Kategorien neben den anderen Lernmethoden dar. Stattdessen bestimmen sie den Ablauf des Lernens und definieren einen konkreten Abschluss eines Lernprozesses. Offline Learning stellt dabei den Lernprozess mit einem statischen Datensatz dar. Mit diesem Datensatz wird das Modell trainiert, validiert und im Anschluss genutzt. Im Gegensatz dazu gibt es bei Online Learning keinen statischen und endlichen Datensatz. Die Daten sind nicht von Anfang an vollständig vorhanden, sondern werden während des Lernprozesses einzeln oder nur in kleinen Mengen verfügbar gemacht.

Der in dieser Arbeit zu entwickelnde Prototyp beinhaltet Aspekte beider Varianten. Während das zugrunde liegende Modell, welches die Klassifizierung durchführt, dem Offline Learning zuzuordnen ist, fällt das übergreifende System eher in das Online Learning. Kommen wie im Abschnitt über Reinforcement Learning und in Kapitel 4.1 beschrieben, neue Daten hinzu, können diese für einen stetigen Lernprozess des Gesamtsystems genutzt werden.

Im Kontext von neuronalen Netzen wird Offline Learning auch als **Batch Learning** bezeichnet, da vor dem Berechnen des Fehlerwertes erst der gesamte Datensatz (**Batch**) einmal zum Lernen genutzt wird [5, S. 156ff]. Danach werden die Gewichtungen im neuronalen Netz entsprechend justiert (in Kapitel 2.5 werden neuronale Netze genauer erklärt).

2.1.2. Validierung

Zu jeder der oben genannten Lernmethoden gibt es für konkrete Verfahren eine Vielzahl von **Hyperparametern**, die das genaue Verhalten und letztlich auch die Leistung des trainierten Modells beeinflussen. Dabei kann es vorkommen, dass ein Modell die Trainingsdaten zu 100 % korrekt erfüllt, auf realen Daten allerdings wesentlich geringere Genauigkeiten erzielt. Eine mögliche Ursache für so ein Verhalten ist das sogenannte **Overfitting** [6, S. 288ff]. Dabei passt sich das Modell den Trainingsdaten perfekt an, ohne letztlich auf das eigentliche Problem zu generalisieren. Um diese Überanpassung im Vorfeld zu erkennen, sollte ein Teil der Trainingsdaten als Testdaten zurückgehalten werden, mit denen nach dem Lernprozess eine Validierung durchgeführt wird. Diese werden dazu genutzt, die eigentliche Leistung des gelernten Modells beurteilen zu können, indem geprüft wird, wie viele Fehlentscheidungen mit den Testdaten getroffen werden. Dabei ist es wichtig mehrere Durchläufe mit verschiedenen Aufteilungen der Daten in Trainings- und Testdaten durchzuführen, um zu verhindern, dass das Modell eine bestimmte Richtung (**Bias**) oder Klassifizierungsentscheidung der Daten bevorzugt [6, S. 288ff]. Ein Verfahren, welches sich besonders gut dazu eignet, ist die Kreuzvalidierung (**Cross-Validation**) [4, S. 242f]. Die Kreuzvalidierung ist auch dann von Vorteil, wenn nicht genügend Daten vorhanden sind, um einen großen Teil davon zur Validierung zurückzuhalten. Das Verfahren liefert Informationen über den zu erwartenden Vorhersagefehler (**Prediction Error**), der zu erwartenden Genauigkeit, sowie Standardabweichungen des Modells. Zur Durchführung wird die Datenmenge X zufällig in k gleich große Blöcke aufgeteilt. Auf $k-1$ dieser Blöcke wird nun der Lernprozess ausgeführt, auf dem verbleibendem Block die Validierung. Dieser Prozess wird k-mal mit unterschiedlichen

Datenblöcken durchgeführt, bis alle k Blöcke zur Validierung genutzt wurden. Bei jedem Durchlauf werden die oben genannten Informationen berechnet und zum Schluss zusammengerechnet.

Mithilfe der Kreuzvalidierung können auch passende Hyperparameter ermittelt werden, indem für jeden Parameterwert ein Durchlauf durchgeführt wird. Am Ende werden die Werte ausgewählt, welche den geringsten Vorhersagefehler oder die höchste Genauigkeit zur Folge haben. Übliche Werte für k sind fünf oder zehn [4, S. 243], wobei die genaue Anzahl abhängig von der Menge der genutzten Daten ist. Wenn N die Anzahl der Trainingsdaten ist und $k = N - 1$ dann wird auch von **Leave One Out Cross Validation** gesprochen.

2.1.3. Ensemble Learning

Eine Möglichkeit, ohne komplexe Algorithmen akzeptable Leistungen zu erhalten, ist das **Ensemble Learning**. Liegen für ein Problem nur sogenannte schwache Lerner vor, deren Entscheidungen nur wenig besser als zufällig sind, können mehrere dieser Lerner genutzt werden, um ein insgesamt wesentlich besseres Ergebnis zu erhalten [3, S. 748ff]. Im einfachsten Fall wird eine Abstimmung (**Voting**) durchgeführt. Bei einer Anzahl von beispielsweise fünf schwachen Lernern müssen demnach mindestens drei in ihrer Annahme falsch liegen, um ein insgesamt falsches Ergebnis zu erzielen. Zusätzlich können die einzelnen Ergebnisse auch gewichtet werden, abhängig davon, wie gut die Leistung des entsprechenden Lerners auf dem Datensatz ist.

Es gibt verschiedene Methoden, um schwache Lerner zu kombinieren. Neben dem bereits erwähntem einfachen Voting gibt es das **Bagging** und **Boosting** [7], wobei Letzteres am weitesten verbreitet ist. Einer der bekanntesten Boosting-Algorithmen ist **AdaBoost** [8], welcher die Trainingsdaten gewichtet. Dieser erzeugt in mehreren Iterationen verschiedene Varianten eines Lerners, wobei bei jedem Durchlauf die Gewichte angepasst werden. Trainingsdaten, die zuverlässig zugeordnet wurden, erhalten schwächere Gewichtungen. Im Gegensatz dazu werden Trainingsdaten, die häufig falsch zugeordnet wurden, mit höheren Gewichten versehen. Dadurch werden die Lerner auf diesen entsprechend gewichteten Daten stärker trainiert. Zum Schluss werden die einzelnen schwachen Lerner gewichtet und zu einem starken Lerner kombiniert.

Das Kombinieren mehrerer schwacher Lerner ist eine effiziente Methode, um mit relativ wenig Aufwand bessere Ergebnisse zu erhalten, als ein einzelner schwacher Lerner erbringen könnte. Daher ist Ensemble Learning mittlerweile weit verbreitet [2].

2.2. Datensätze zur Gesichtserkennung

Äußerst wichtig für Machine Learning und Gesichtserkennung sind die zugrunde liegenden Daten, auf denen die erstellten Modelle basieren. Daher soll im Folgenden näher auf die Thematik an sich, sowie auf einige verfügbare Datensätze eingegangen werden.

Mit der Verfügbarkeit von Datensätzen steht und fällt jedes Verfahren zum Machine Learning. Es ist zu unterscheiden, ob die Daten bereits gekennzeichnet sind (also ein Label besitzen), oder ob die eigentlichen Zuordnungen unbekannt sind. Dies wirkt sich unter anderem auf die Wahl des Lernverfahrens aus (siehe Kapitel 2.1.1). Äußerst wichtig ist zudem die Menge an Daten. Untersuchungen bezüglich der nötigen Datenmenge für Disambiguierungen in natürlichen Sprachen von Banko und Brill von 2001 [9] haben gezeigt, dass eine Steigerung der genutzten Datenmenge eine signifikante Auswirkung auf die Genauigkeit der angewendeten Verfahren hat. Während große Datenmengen sich prinzipiell positiv auf die Genauigkeit auswirken, sind jedoch auch Nachteile zu beachten. Zum einen ist die Verfügbarkeit von so großen Datenmengen[1], die gekennzeichnet sind, nicht selbstverständlich. Dabei ist vor allem die Kennzeichnung problematisch, da diese für die meisten Anwendungsfälle manuell vorgenommen werden muss. Des Weiteren bedeuten mehr Daten auch eine längere Trainingsphase oder erhöhten Bedarf an Rechenleistung und Speicherplatz. Beides ist in Bezug auf die verfügbaren Ressourcen für kleine Projekte oder bestimmte Anwendungszwecke nicht unbedingt möglich.

Zur Gesichtserkennung bestehen die Daten üblicherweise aus Bildern auf denen die Gesichter von einzelnen Personen abgebildet sind. Jeweils eine Person stellt eine einzelne Klasse oder Kennzeichnung dar. Dabei ist zu unterscheiden, ob die Bilder Farbinformationen besitzen oder nur in Graustufen vorliegen. Zudem können Variationen in der Beleuchtung und Pose der Person (beziehungsweise des Gesichts) im Bild vorliegen, welche mit die größten Herausforderungen für Gesichtserkennungsverfahren darstellen [1]. Im Folgenden sollen einige exemplarische Gesichtsdatensätze benannt und genauer erläutert werden.

- **AT&T**
  Ein häufig verwendeter Datensatz ist die **Database of Faces** der *AT&T Laboratories Cambridge*. Die Datenbank wurde zwischen 1992 und 1994 erstellt und beinhaltet Bilder von 40 Personen von denen jeweils 10 Aufnahmen in verschiedenen Sitzungen gemacht wurden. Die Beleuchtung zwischen den einzelnen Aufnahmesitzungen variiert, das Gesicht der Person ist aber immer frontal der Kamera zugewendet. Die Bilder liegen in Graustufen mit einer Größe von 92x112

[1] In [9] wurden drei Größenordnungen von Datenmengen verglichen, von einer Million auf eine Milliarde.

Pixeln vor [10]. Ein Beispiel ist in Abbildung 1 zu sehen. Aufgrund der strikten Bedingungen und geringen Variationen wurden auf diesem Datensatz sehr gute Ergebnisse erzielt, was sich allerdings nicht unbedingt auf reale Problemstellungen übertragen lässt.

Abbildung 1: Beispiel aus dem AT&T-Datensatz.

- **CMU Pose, Illumination, and Expression Database**
  Um eine größere Menge an Daten von Gesichtern für die Erkennung bei unterschiedlichsten Bedingungen zu erhalten, wurden Ende 2000 an der *Carnegie Mellon University* die **Pose, Illumination, and Expression (PIE)**-Datenbank, erstellt [11]. Mit 13 Kameras in verschiedenen Positionen im Raum wurden von 68 Personen über 40.000 Bilder erstellt. Neben den 13 Kameras wurden im Raum 21 Blitzlichter verteilt, um möglichst umfangreiche Beleuchtungsbedingungen darzustellen. Insgesamt liegen pro Person ungefähr 600 Bilder vor, inklusive Farbbildern deren Auflösung 640x486 Pixel beträgt. Ein Ausschnitt einer Aufnahme ist in Abbildung 2 zu sehen.

Abbildung 2: Beispiel aus der PIE-Datenbank [11] mit 13 verschiedenen Posen.

- **FERET**
  Die **Facial Recognition Technology (FERET)**-Datenbank hat das Ziel eine allgemeingültige Datenbank für unabhängige Studien zu neuen Algorithmen zur Gesichtserkennung darzustellen. In [12] erklären die Autoren, dass einige der neu vorgestellten Algorithmen eine hohe Genauigkeit aufweisen, was allerdings damit zu erklären ist, dass diese nur auf geringen Datenmengen getestet wurden. Zudem stammen diese von den gleichen Autoren, die auch den Algorithmus entwickelten. Da diese Daten anderen Arbeitsgruppen nicht direkt zugänglich waren und zudem die Evaluierungsmethoden nicht beschrieben wurden, war keine direkte Vergleichbarkeit gegeben. Um Forschern zu ermöglichen, ihre Algorithmen gegen andere

vergleichbar zu evaluieren, wurde neben dem Datensatz auch ein Testprotokoll entwickelt.

Der Datensatz umfasst für die einzelnen Personen mehrere Posen, unterschiedliche Beleuchtungen und Aufnahmen mit größeren zeitlichen Abständen. Insgesamt enthält der Datensatz ungefähr 14.000 Aufnahmen von 1199 Personen. Die Bilder, von denen Ursprünglich nur Graustufenfassungen mit 256x384 Pixeln zur Verfügung standen, werden in einer neueren Fassung als Farbbilder mit einer Auflösung von 512x768 Pixeln verteilt [13].

- **Labeled Faces in the Wild**
  Als Reaktion auf die vielen verschiedene Datensätze, welche nur Aufnahmen in streng kontrollierten Laborumgebungen beinhalten, liegen in **Labeled Faces in the Wild (LFW)** nur Ausschnitte mit Gesichtern aus bereits vorhandenen Aufnahmen vor. Dies dient dazu, Gesichtserkennungsalgorithmen in vollkommen unkontrollierten Umgebungen zu testen und zu vergleichen [14]. Die Ausschnitte stammen von Bildern aus Nachrichtenartikeln. Mithilfe des Viola-Jones-Detektors (siehe Kapitel 2.3.1) wurden Gesichter in den Bildern ermittelt und diese manuell mit Hilfe der Bildunterschriften gekennzeichnet.

  Als ursprünglicher Verwendungszweck des Datensatzes war die Untersuchung von Verfahren zum **Pair-Matching** vorgesehen. Dabei handelt es sich um Gesichtserkennungsverfahren, welche bei zwei gezeigten Bildern ermitteln, ob auf den Aufnahmen die selbe oder verschiedene Personen zu sehen sind. Neben den reinen Bilddaten gibt es auch ein Testprotokoll und eine Unterscheidung zwischen Trainings- und Testdaten, wodurch Algorithmen, die diesen Datensatz nutzen, gut vergleichbar sind. Der Datensatz enthält 13.233 Bilder von 5.749 verschiedenen Personen, von denen es zu 1.680 mehr als ein Bild gibt. Die Auflösung beträgt 250x250 Pixel. Die meisten Aufnahmen sind farbig, es liegen gemäß der breit gefächerten Herkunft der Bilder allerdings auch einige Graustufenbilder vor. Aufgrund der vielfältigen Aufnahmen ohne jegliche Kontrolle über die Parameter gilt der Datensatz derzeit als einer der anspruchsvollsten für Gesichtserkennungsverfahren. Daher wurde und wird er umfangreich genutzt, um Verfahren zu entwickeln, welche Genauigkeiten erreichen, die zum Teil besser sind als die von Menschen [15].

- **H-BRS Haut-, Gesichts- und Fälschungsdatenbank**
  Im Rahmen von zwei Projekten des Instituts für Sicherheitsforschung der Hochschule Bonn-Rhein-Sieg zur Fälschungserkennung sowie Personendetektion wurde eine Datenbank von Gesichtern angelegt. Da menschliche Haut im nahinfra-

roten Spektralbereich unabhängig von Alter, Geschlecht und Hautfarbe gut unterscheidbar von anderen Materialien ist [16], wurden neben einfachen **Rot-Grün-Blau (RGB)**-Aufnahmen auch Aufnahmen in verschiedenen infraroten Spektralbereichen gemacht.

Der Datensatz, im folgenden als **Bonn-Rhine-Sieg University (BRSU)**-Datensatz bezeichnet, umfasst dabei drei unterschiedliche Teile; Spektrometermessungen an verschiedenen Hautpunkten, Frontalaufnahmen des Gesichts, sowohl in RGB als auch in mehreren Wellenlängenbändern, und beispielhafte Fälschungen in Form von diversen Verfremdungen im Gesicht, ebenfalls in RGB und unterschiedlichen Wellenlängenbändern im Nahinfrarotbereich.

Im Datensatz liegen für jede der 53 Personen (ohne die Spektralaufnahmen) mehrere Aufnahmen, aus verschiedenen Winkeln, sowie mit unterschiedlichen Gesichtsausdrücken, vor. Für jede RGB-Aufnahme gibt es mehrere Aufnahmen mit verschiedenen Wellenlängenbändern. Ein Auszug ist in Abbildung 3 zu sehen. Weitere Informationen zu den Aufnahmen sind in [17] und [18] zu finden.

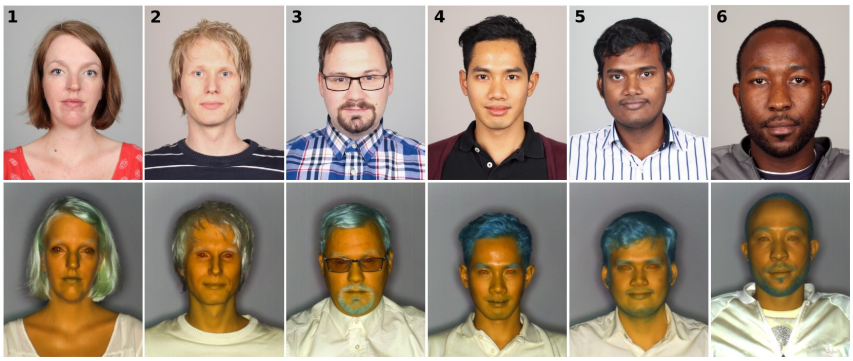


Abbildung 3: Beispiel aus der Haut-, Gesichts- und Fälschungsdatenbank [16] mit verschiedenen Hauttypen und den zugehörigen Spektralaufnahmen aus dem Nahinfrarotbereich [18].

2.2.1. Vorverarbeitung

Bevor eine Aufnahme zur Gesichtserkennung verwendet wird, können diverse Vorverarbeitungsschritte durchgeführt werden, um die Ergebnisse zu verbessern. Die Vorverarbeitung ist vor allem für Anwendungsgebiete sinnvoll, in denen nur wenige Parameter während der Aufnahme kontrolliert werden können. Der Grund dafür ist, vor allem bezogen auf Gesichtserkennung, dass einzelne Bilder von der gleichen Person unterschiedlicher wirken können, als einzelne Bilder zwischen verschiedenen Personen. Ersteres wird auch **Intra-Class-Variation** und letzteres **Inter-Class-Variation** genannt. Vor allem bei

sich ähnelnden Personen stellt dies ein Problem dar [19]. Die Ursachen für solche Unterschiede sind beispielsweise Beleuchtung und Variationen in der exakten Pose während der Aufnahme. Da diese Aspekte auch im Allgemeinen große Probleme für die Gesichtserkennung darstellen, ist es sinnvoll, diese auszugleichen.

Ein typischer Schritt der Vorverarbeitung ist das Angleichen der Kontrastwerte mittels einer sogenannten Histogrammspreizung (**Histogram Equalization**). Dabei werden die Helligkeitswerte aller Pixel über das gesamte Spektrum gleichmäßig verteilt. Schwankungen in der Helligkeit so wie geringe Kontraste in Bildern können somit ausgeglichen werden. Problematisch wird dieser Ansatz, wenn innerhalb eines Bildes starke Beleuchtungsschwankungen vorliegen. Diese können jedoch mithilfe verschiedener Ansätze ausgeglichen werden [20].

Des Weiteren ist in realen Bedingungen die Ausrichtung des Gesichts in einem Bild nicht immer gleich, sondern kann in verschiedenen Ebenen rotiert sein. Dabei muss zwischen zwei Arten von Rotation unterschieden werden. Zum einen die, die außerhalb der sichtbaren Ebene des Bildes statt finden (**out-of-plane rotation**), also im dreidimensionalen Raum der vom Bild nicht erfasst wird. Durch eine so geartete Drehung des Kopfes werden Merkmale, die normalerweise zur Erkennung genutzt werden könnten, verdeckt. Dazu zählen beispielsweise Profilaufnahmen, wenn zur Erkennung Frontalaufnahmen verwendet werden sollen. Zum anderen gibt es Rotationen, die innerhalb der sichtbaren Ebene statt finden (**in-plane rotation**) [21]. Letztere sind dabei einfacher zu korrigieren. Mithilfe bestimmter zu erkennender Merkmale (Mund, Nase, Augen) können sämtliche Aufnahmen so gedreht werden, dass die darauf abgebildeten Gesichter die gleiche Orientierung aufweisen. Dieses Vorgehen wird auch **Alignment** genannt. Wesentlich problematischer gestalten sich out-of-plane Rotationen. Je nach verwendetem Verfahren sind diese in der Lage, Rotationen bis zu einem gewissem Grad auch in zweidimensionalen Bildern auszugleichen. Im Allgemeinen erbringen Verfahren, die auf Aufnahmen mit gleicher Pose gut funktionieren, wesentlich schlechtere Leistung sobald große Unterschiede in den Posen auftreten. Um dieses Problem zu umgehen, gibt es viele Ansätze, die versuchen eine von der Pose unabhängige Gesichtserkennung zu ermöglichen [22].

Zusätzlich werden die Bilder häufig auf die gleiche Größe skaliert. Für einige Verfahren ist es eine technische Voraussetzung, allgemein verbessert es aber auch die Erkennungsraten. Grund dafür ist, dass verschiedene Merkmale in allen Bildern unterschiedlich groß erscheinen und damit nicht unbedingt ein Zusammenhang erkannt wird. Zudem liefern moderne Kameras Aufnahmen mit Auflösungen im Megapixelbereich, was eine enorme Datenmenge darstellt. Hier ist es sinnvoll, die Bilder auf wesentlich geringere Auflö-

sungen zu skalieren. Ein positiver Nebeneffekt davon ist die Reduktion von Rauschen im Bild. Dabei ist eine Erkennung selbst dann noch möglich, wenn die Auflösung eines Gesichts 12x11 Pixel beträgt, wohingegen für menschliche Beobachter eine minimale Auflösung von ungefähr 18x24 Pixel nötig ist [21].

2.2.2. Augmentierung

Der Erfolg von Gesichtserkennungsverfahren in den letzten Jahren ist vor allem den immer umfangreicheren Datensätzen zu verdanken. Große Datensätze anzulegen ist jedoch ein sehr zeitaufwendiger Prozess. Um mehr Trainingsdaten zu erhalten, ohne neue Daten manuell zu erfassen, können aus bestehenden Daten neue generiert werden. Üblich im Rahmen der Gesichtserkennung und neuronaler Netze sind dafür die Begriffe **Augmentation** oder auch **Data Jittering** und **Virtual Sampling**.

Da vor allem neuronale Netze von großen Datenmengen profitieren, wird Augmentierung häufig im Bereich des **Deep Learning**, der Arbeit mit tiefen neuronalen Netzen, eingesetzt. Untersuchungen haben allerdings ergeben, dass auch flache Verfahren durch künstlich vergrößerte Datenmengen bessere Leistungen erzielen können [23]. Dabei gibt es klassischerweise zwei verschiedenen Arten, um aus einzelnen Bildern mehrere Bilder einer Klasse, also der gleichen Person, zu erzeugen.

- **Ausschnitte erzeugen**
  Aus einem Bild wird ein Ausschnitt gebildet, der kleiner ist als das Original. Die genauen Maße hängen dabei von der Ausgangsgröße der Bilder, beziehungsweise der Bereiche mit Gesichtern, sowie der gewünschten oder benötigten Eingangsgröße[2] ab. Die Ausschnitte können aus der Mitte des Bildes erfolgen oder zusätzlich aus allen vier Ecken oder auch jeder beliebigen Verschiebung von Pixeln. Wichtig ist, dass genügend Informationen über das abgebildete Gesicht erhalten bleiben.

- **Spiegelung**
  Wesentlich simpler ist es, ein Bild zu Spiegeln, um ein weiteres Bild für den Datensatz zu erhalten. Für Aufnahmen von Gesichtern, die üblicherweise aufrecht vorliegen und aufgenommen werden, bietet sich eine horizontale Spiegelung an. Sollten vorher mehrere Ausschnitte mit der oben genannten Augmentierungsmethode erzeugt worden sein, können diese jeweils gespiegelt werden.

[2] Neuronale Netze, im konkreten **Convolutional Neural Networks (CNNs)**, setzen eine feste Eingangsgröße voraus, weiteres dazu in Kapitel 2.5.

Durch die Kombination dieser beiden Schritte lässt sich die Anzahl der verfügbaren Bilder schnell vervielfachen. Zuerst können aus einem Bild mit beispielsweise 500x500 Pixeln fünf einzelne Ausschnitte mit jeweils 450x450 Pixeln erzeugt werden. Ein Ausschnitt aus der Mitte, die anderen aus den vier Ecken. Danach wird jeder Ausschnitt horizontal gespiegelt, wodurch sich insgesamt 10 zusätzliche Aufnahmen ergeben.

Eine weitere Nutzungsmöglichkeit ist nicht nur das künstliche Vervielfachen von Daten, sondern auch der Ausgleich von ungleich verteilten Datensätzen [24]. Liegen beispielsweise für eine Person nur zwei Bilder vor, für eine andere jedoch 20, ist die Wahrscheinlichkeit groß, dass das Modell sich auf die zweite Person überanpasst. Mithilfe der Augmentierung lässt sich dieser Mengenunterschied bis zu einem gewissen Grad ausgleichen.

Neben den oben genannten gibt es noch viele weitere Möglichkeiten, um Bilder leicht zu verändern und somit für Trainingszwecke zu vervielfachen [25–27]. Generell ist zu beobachten, dass die Augmentierung der Daten positive Auswirkungen hat. Jedoch sind dem Autor zum Zeitpunkt der Erstellung dieser Arbeit keine Veröffentlichungen bekannt, welche sich mit den genauen Auswirkungen im Detail, sowie einem konkreten Vergleich der verschiedenen Augmentierungsmethoden beschäftigt.

2.3. Gesichtsdetektion

Um eine Gesichtserkennung durchführen zu können, muss zuerst ermittelt werden, ob und wo in einem Bild ein Gesicht vorhanden ist. Die Gesichtsdetektion steht dabei vor ähnlichen Problemen wie die Gesichtserkennung und -identifizierung an sich und wird daher schon ebenso lange erforscht. Verschiedene Posen, Beleuchtungen, Skalierungen und Variationen in verschiedenen Gesichtern stellen große Herausforderungen dar.

Durch die vielfältigen Anwendungsmöglichkeiten und einem entsprechenden Interesse aus Forschung und Industrie haben sich im Lauf der Jahrzehnte viele unterschiedliche Ansätze entwickelt, um die Gesichtsdetektion in Bildern zu ermöglichen. Eine Kategorisierung dieser Ansätze wurde 2002 in [28] vorgenommen und soll hier, inklusive der Bezeichnungen, übernommen werden. Eine Überlappung der Verfahren und Kategorien ist dabei nicht ausgeschlossen

- **Knowledge-based**
  Bei wissensbasierten Ansätzen wird die Gesichtsdetektion durch fest definierte Regeln ermöglicht. Diese Regeln beschreiben was ein Gesicht ausmacht, beispielsweise zwei Augen, eine Nase und ein Mund. Diese Merkmale in einem Ge-

sicht werden auch **Features** genannt und sind auch für die Gesichtserkennung in Kapitel 2.4 relevant. Zusätzlich können die Positionen und Distanzen dieser Merkmale zueinander genutzt werden.

Um den Rechenaufwand gering zu halten, wird die Erkennung üblicherweise in hierarchisch geschichteten Teilschritten durchgeführt. Dabei wird ein Bild zuerst auf grobe Gesichtszüge hin untersucht, indem ein sich verschiebender Teilbereich des Bildes betrachtet wird, ein sogenanntes Fenster (**Window**). Eine bildliche Darstellung solch einer Regel ist in Abbildung 4 zu sehen.

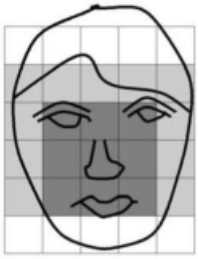


Abbildung 4: Beispiel für eine Regel in einer wissensbasierten Gesichtserkennung. Unterschiede und Verteilung der Helligkeitswerte sind ausschlaggebend. Bild aus [28], ursprüngliche Beschreibung der Regel in [29].

In dem gezeigten Beispiel ist ein Gesicht durch mehrere Zellen gleicher Helligkeit in der Mitte, sowie darüber und daneben liegenden Zellen mit gleicher Helligkeit definiert. Die durchschnittliche Helligkeit dieser Teilbereiche muss sich zusätzlich signifikant unterscheiden. Auf Bereiche mit vielversprechenden Kandidaten werden detailliertere Regeln angewendet, welche beispielsweise Augen und ähnliche Merkmale beschreiben [29].

Die meisten dieser Ansätze erzielen gute Ergebnisse, wenn nur ein Gesicht mit einem relativ gleichmäßigem Hintergrund abgebildet ist. Bei Aufnahmen mit mehreren Gesichtern, die zudem nah beieinander sind, sinkt die Erkennungsrate stark. Ein weiterer Nachteil ist der hohe Aufwand, welcher nötig ist, um aus menschlichem Wissen gut funktionierende Regeln zu abstrahieren. Sind diese zu strikt, werden Gesichter eventuell nicht erkannt, sind sie zu locker erkennt das System Gesichter in Regionen in denen keine vorhanden sind. Zusätzlich erfassen solche Regeln Gesichter nur in einer bestimmten Orientierung. Augen müssen sich demnach über einer Nase und diese wiederum über einem Mund befinden. Um rotierte Gesichter zu erkennen, müssten viele einzelne Regeln für unterschiedliche Rotationen erfasst werden, was den Aufwand zusätzlich erhöht.

- **Feature invariant**
  Im Kontrast zu den eben genannten Nachteilen sind menschliche Betrachter durchaus in der Lage auch dann Gesichter zu erkennen, wenn diese sich in verschiedensten Posen und Beleuchtungen befinden. Aufgrund dieser Beobachtung ergibt sich der Ansatz, Eigenschaften oder Merkmale zu ermitteln, welche unabhängig von den oben genannten möglichen Unterschieden sind.

  Ähnlich wie bei den wissensbasierten Verfahren können Merkmale in einem Gesicht als Grundlage genommen werden. Anstatt allerdings feste Positionierungen der Merkmale anzunehmen, sind relative Positionierungen und Distanzen zueinander ausschlaggebend, was auch die Erkennung von rotierten Gesichtern ermöglicht. Die Merkmale an sich werden über Kantendetektion und **Blob Detection**[3] ermittelt. Im Anschluss wird durch verschiedene statistische Verfahren entschieden, ob die gefundenen Bereiche ein Gesicht enthalten.

  Neben der Abhängigkeit von konkreten Merkmalen gibt es auch Ansätze, die sich auf das Vorhandensein von bestimmten Texturen stützen. Durch Texturmodelle von Haut und Haaren lässt sich ermitteln, ob und wo diese in bestimmten Verhältnissen zueinander vorhanden sind. Ein Vorteil von diesem Vefahren ist die komplette Unabhängigkeit von Posen, sowie eine Robustheit gegenüber Verdeckungen wie Brillen oder Gesichtsbehaarung.

  Nicht nur die Textur sondern auch die Farbe von Haut kann genutzt werden, um auf Gesichter zu schließen. Frühe Ansätze haben versucht ein Modell von menschlicher Hautfarbe in diversen Farbräumen zu definieren. Es hat sich gezeigt, dass trotz verschiedener Hautfarben die größeren Unterschiede in der Helligkeit anstatt in konkreten Farbwerten liegen. Trotz verschiedener Ansätze reicht das Erkennen von Bereichen mit Hautfarbe allerdings nicht aus, um zuverlässig Gesichter zu erkennen. Vielmehr werden häufig kombinierte Ansätze verwendet, welche aus Erkennung von Farbe, Form, Größe und weiteren der oben genannten Methoden bestehen. Dabei können erst Kandidaten durch Erkennung von Hautfarbe ermittelt werden. Im Anschluss wird jeder Kandidat auf das Vorhandensein von konkreten Merkmalen wie beispielsweise Augen untersucht.

- **Template matching**
  Ähnlich wie bei wissensbasierten Ansätzen beruht das **Template matching** (oder auch Mustervergleich) auf abstrahiertem Wissen darüber, was ein Gesicht ausmacht. Dabei werden allerdings keine Regeln vorgegeben, sondern bestimmte

[3] Bereiche im Bild welche sich beispielsweise durch ihre Helligkeit von umliegenden Regionen unterscheiden.

Muster, die vorgeben, wie ein Gesicht aussieht. Ein zu untersuchendes Bild wird auf Korrelation zu dem Standardmuster hin überprüft. In seiner grundlegenden Form ist dieser Ansatz leicht umzusetzen, allerdings ist er sehr anfällig gegenüber Änderungen in Pose, Form und Skalierung von Gesichtern. Um diesen Defiziten entgegen zu wirken, wurden diverse Verbesserungen entwickelt.

Unter anderem können statt eines festen Musters mehrere kleine Muster, sogenannte **Subtemplates**, beschrieben werden, welche Augen, Nase, Mund, Kontur und ähnliche Merkmale eines Gesichts beschreiben. Konkret liegen diese Muster als Linienabschnitte vor, welche aus einem Bild anhand der Gradientenänderungen[4] ermittelt werden. Ist die grobe Kontur eines Gesichts gefunden, wird innerhalb dieser nach den einzelnen Subtemplates gesucht. Ausgehend von diesem Ansatz haben sich viele weitere Methoden mit schwankenden Ergebnissen entwickelt. Diese haben allerdings alle den Nachteil, dass zu Beginn aus Expertenwissen heraus ein Muster festgelegt werden muss, selbst wenn dieses in der Lage ist, sich später in gewissen Grenzen anzupassen.

- **Appearance-based**
  Um die Voraussetzung eines menschlichen Experten der Regeln vorgibt zu eliminieren, wurden Ansätze entwickelt, welche selbständig lernen können, woran ein Gesicht in einem Bild zu erkennen ist. Diese Ansätze nutzen Machine Learning und statistische Methoden, welche auch häufig zur Gesichtserkennung im Rahmen der holistischen Methoden (siehe Kapitel 2.4.2) genutzt werden.

  Anhand von Beispieldaten lernen diese Ansätze, was ein Gesicht in einem Bild ausmacht. Verfahren aus dieser Kategorie zeigen aufgrund der steigenden Größe von Datensätzen sowie immer mehr verfügbarer Rechenleistung bessere Leistungen als Ansätze aus den anderen Kategorien [30]. Aus diesem Grund wurden viele Ansätze entwickelt, die in diese Kategorie fallen, wie beispielsweise **Eigenfaces** und neuronale Netze, auf welche später noch genauer eingegangen wird.

  Grundlegend versuchen diese Verfahren entweder die Wahrscheinlichkeit zu berechnen, mit der ein bestimmter Bereich eines Bildes als Gesicht einzuordnen ist, oder es wird versucht eine Diskriminante, also eine Funktion, zu finden welche entscheiden kann, ob ein Gesicht vorhanden ist oder nicht.

Zwei Verfahren, welche für die praktische Umsetzung dieser Arbeit (siehe Kapitel 3.1.1) relevant sind, sollen im Folgenden näher beschrieben werden. Für weitere Informationen

[4] In der Bildverarbeitung entspricht dies in der Regel einer Kantendetektion, welche durch verschiedene Methoden durchgeführt werden kann.

zur Gesichtsdetektion und der Entwicklung in diesem Forschungsgebiet wird auf [28,30, 31] verwiesen.

2.3.1. Viola-Jones-Verfahren

Ein Meilenstein der erstmals Gesichtsdetektion in Echtzeit ermöglichte, wurde 2004 von [32] erreicht. Aufgrund der weiten Verbreitung und Bedeutung des Viola-Jones-Verfahrens für die Gesichtsdetektion soll im Folgenden auf dieses näher eingegangen werden. Das Verfahren ist prinzipiell in die Kategorie **Appearance-based** einzuordnen, da es letztlich einen Klassifizierer bildet, der mithilfe von Trainingsdaten lernt, zwischen Gesichtern und Nicht-Gesichtern zu unterscheiden. Seitdem gab es mehrere Weiterentwicklungen sowie ähnliche Ansätze, welche die Gesichtsdetektion weiterhin verbessert und beschleunigt haben. Einen guter Überblick über diese Entwicklungen wird in [30] wiedergegeben.

Das Verfahren von Viola und Jones besteht aus drei Komponenten, die eine Gesichtsdetektion in Graustufenbildern in Echtzeit ermöglichen. Die erste ist eine spezielle Art von Merkmalen, abgeleitet von den sogenannten Haar-Funktionen. Von diesen Merkmalen gibt es drei verschiedene Sorten, welche alle aus verschiedenen Kombinationen aneinander angrenzender Rechtecke bestehen. Diese besitzen immer die gleiche Form und Größe und können sowohl vertikal als auch horizontal aneinander angrenzen. In Abbildung 5 sind diese Merkmale beispielhaft dargestellt.

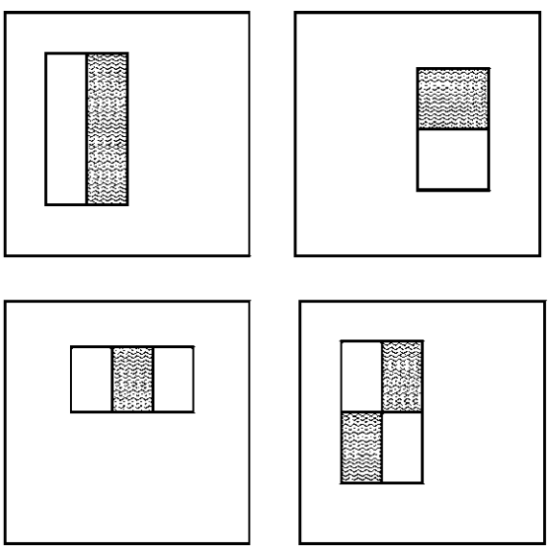


Abbildung 5: Beispiele für Merkmale im Viola-Jones-Verfahren. Die obere Reihe zeigt die erste Art von Merkmalen mit zwei Rechtecken, die zweite Reihe die beiden anderen mit drei und vier Rechtecken. Abbildung aus [32].

Die erste Sorte von Merkmalen besteht aus zwei Rechtecken, wobei sich der Wert des Merkmals aus der Differenz der Summe der Pixel[5] in den beiden Rechtecke ergibt. Ein weiteres Merkmal besteht aus drei Rechtecken, wobei die beiden äußeren Rechtecke eine Summe ergeben und von der Summe des mittleren Rechtecks abgezogen werden. Das letzte Merkmal ergibt sich aus vier Rechtecken, welche zwei diagonale Paare bilden. Um die Berechnung der Merkmale zu beschleunigen, wird in einem Zwischenschritt ein sogenanntes Integralbild erzeugt. In diesem wird für jeden Pixel die Summe der Pixel oberhalb und links berechnet. Dieses Integralbild kann in einem Durchlauf erzeugt werden. Danach kann die Summe jedes beliebigen Rechtecks in konstanter Zeit berechnet werden. Trotz dieser Einsparungen ergeben sich für jede mögliche Position und Größe der Merkmalsorten innerhalb eines Bildes oder Bildausschnitts wesentlich mehr Merkmale als es Pixel gibt[6]. Aus diesem Grund wurde als zweite Neuerung die Selektion der Merkmale zur Klassifizierung mithilfe des AdaBoost-Algorithmus durchgeführt (siehe Kapitel 2.1.3). Der Algorithmus wird dazu genutzt, diejenigen Merkmale auszuwählen, welche am besten ein Gesicht beschreiben. Dazu werden viele einzelne schwache Lerner gebildet, welche jeweils auf einem einzelnen Merkmal basieren. Diejenigen mit der geringsten Fehlerrate und höchsten Genauigkeit werden am Ende zu einem starken Lerner zusammengefasst und bilden den Klassifizierer, von Viola und Jones auch Detektor genannt.

In der konkreten Anwendung besitzt dieser Detektor eine Größe von 24x24 Pixeln und wird Schrittweise über das gesamte Bild geschoben. Für jeden Teilausschnitt gibt dieser an, ob ein Gesicht vorhanden ist oder nicht. Um eine zuverlässige Aussage treffen zu können, wären hierfür ungefähr 200 Merkmale nötig. Um die Detektion von Gesichtern in Echtzeit zu ermöglichen, ist der Aufwand alle Merkmale zu überprüfen jedoch zu groß[7]. Daher wurde als drittes eine Kaskade von Klassifizierern eingeführt. Ziel ist es, einen Bildausschnitt ohne ein Gesicht so schnell wie möglich zu verwerfen und nur gute Kandidaten näher zu betrachten. Die ersten Stufen filtern dabei sehr grob und benutzen nur sehr wenige Merkmale, wodurch sie schnell zu berechnen sind. Insgesamt besteht die letztlich genutzte Kaskade aus 38 Stufen mit 6.060 Merkmalen. Der Vorteil des Verfahrens liegt eindeutig in der schnellen und relativ genauen Detektion von Gesichtern. Nachteile sind, dass nur mehr oder weniger frontale Gesichter gut erkannt werden und auch diese nur mit begrenzten Rotationswinkeln. Letzteres lässt sich allerdings durch den zum Training verwendeten Datensatz beeinflussen. Zudem sinkt die Detektionsrate stark, wenn sich die Beleuchtungsbedingungen ändern, oder wenn Teile des Gesichts einer Person, vor allem einzelne Augen, verdeckt sind.

[5] Beziehungsweise der Helligkeitswert des Pixels, da mit Graustufen gearbeitet wird.

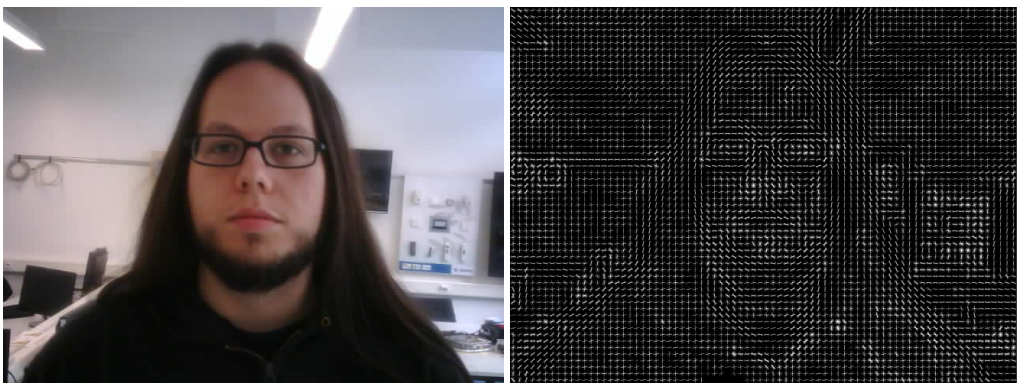
[6] Viola und Jones gehen von einer Basisgröße von 24x24 Pixel aus, damit ergeben sich mehr als 160.000 Merkmale.

[7] Zumindest auf der damals genutzten Hardware, einem Intel PIII 700MHz Prozessor.

2.3.2. Histogram of Oriented Gradients

Ein weiteres Verfahren zur Gesichtserkennung basiert auf der Ausrichtung von Kanten in einem Bild. Zur Personenerkennung wurde es 2005 von [33] verwendet und konnte dabei sehr gute Ergebnisse erzielen. Die Idee selbst wurde schon 1995 von [34] genutzt, konnte sich jedoch erst durch die Neuerungen von [33] durchsetzen. In der Grundidee werden Histogramme von Gradienten für jeden Pixel im Bild verwendet, um einen Deskriptor für das Bild zu erzeugen. Daraus leitet sich der Name **Histogram of Oriented Gradients (HOG)** ab.

Nach einer optionalen Gammakorrektur wird im ersten Schritt eine Kantendetektion auf dem Bild durchgeführt. Das kann beispielsweise mit einem Sobel-Filter geschehen. Dadurch entsteht ein Gradientenbild, aus dem sich für jeden Pixel der Betrag und die Richtung des Gradienten berechnen lassen. Im Anschluss wird das Bild in verschiedene Zellen unterteilt, für die Personenerkennung werden dabei Zellengrößen von 8x8 Pixel empfohlen. Für jede Zelle wird dann ein Histogramm der Gradienten erstellt, eingeteilt in neun Winkel von $0\,^\circ$ bis $180\,^\circ$ wobei der Gradientenbetrag als Wert im Histogramm hinzugefügt wird. Danach werden im Bild Blöcke von 2x2 Zellen gebildet, deren Histogramme konkateniert und normalisiert werden. Das Fenster zur Normalisierung wird dabei jeweils 8 Pixel, also eine Zelle breit, weiter geschoben. Die hier wiedergegeben Angaben entsprechen dabei den ermittelten besten Werten aus [33]. Ein so erzeugter Deskriptor ist in Abbildung 6 visualisiert.



(a) (b)

Abbildung 6: Darstellung eines Originalbilds (a) und Visualisierung der Gradientenhistogramme (b).

Der Deskriptor kann anschließend mit einem zuvor aus vielen Deskriptoren erlernten Modell verglichen werden, um zu bestimmen ob und wo in einem Bildbereich ein Gesicht vorliegt. Basierend auf diesem Ansatz wurde von [35] ein erweiterter Deskriptor

vorgestellt, welcher das oben erwähnte Gradientenbild in verschiedene Teile trennt. Wird ein Bild mit einem erlernten Modell zur Objektdetektion verglichen, können sich diese unabhängig voneinander bewegen, um besser dem Modell zu entsprechen. Durch diese erlaubte Deformierung können erlernte Objekte auch dann erkannt werden, wenn diese rotiert oder aus verschiedenen Winkeln aufgenommen sind. Im Kontext der Personendetektion ergeben sich Modelle welcher invariant gegenüber verschiedenen Posen von Personen sind. Allgemein eignet sich das HOG-Verfahren gut zur Objekterkennung und kann entsprechend auch zur Gesichtsdetektion verwendet werden.

2.4. Verfahren zur Gesichtserkennung

Nach der Gesichtsdetektion soll im folgenden Unterkapitel näher auf die konkreten Methoden zur Gesichtserkennung und eine Unterscheidung zwischen diesen eingegangen werden. Aufgrund der vielfältigen Anwendungsmöglichkeiten bei einem interdisziplinärem Forschungsfeld, welches in den letzten Jahren einen großen Aufschwung erfährt[8], ist der Umfang an wissenschaftlichen Arbeiten zur automatischen Gesichtserkennung enorm. Jedes Jahr werden hunderte neue Publikationen veröffentlicht und berichten von immer höheren Erkennungsraten und Leistungen bei Verfahren und Algorithmen. Dabei werden häufig bereits vorhandene Methoden abgewandelt, mit anderen kombiniert oder anderweitig verbessert. Durch diesen stetigen Wandel ist es nur sehr schwer möglich, die verschiedenen Arten von Verfahren zur Gesichtserkennung eindeutig zu klassifizieren und zu unterscheiden, da gerade moderne Verfahren sich häufig Vorgehensweisen aus verschiedenen Bereichen bedienen. Der Einfachheit halber soll daher im Folgenden eine Unterscheidung getroffen werden, welche für diese Arbeit in sich stimmig ist, aber keinesfalls den Anspruch erhebt allgemeingültig oder eindeutig zu sein. Diese orientiert sich dabei an der in [1] und [21] genutzten Unterscheidung.

- **Merkmalbasierte Verfahren**
  Einzelne Merkmale eines Gesichts werden aus einem Bild extrahiert und dazu genutzt um Vergleiche durchzuführen. Frühe Algorithmen basierten auf diesem Verfahren. Dabei werden die Merkmale häufig manuell extrahiert, was sehr zeitaufwändig ist. Daher wurde viel Forschungsaufwand betrieben, um robuste Algorithmen zur automatischen Extraktion von Merkmalen zu entwickeln.

- **Holistische Verfahren**
  Im Gegensatz zur Nutzung einzelner Merkmale versuchen holistische Verfahren

[8] In den letzten Jahrzehnten gab es immer wieder Hochphasen in der Gesichtserkennung, wenn durch neue Technologien oder Verfahren sprunghafte Fortschritte möglich wurden.

die Informationen aus dem Gesicht als ganzes verfügbar zu machen. Dadurch entstehen große Datenmengen mit hoher Dimensionalität, weswegen viele Algorithmen in dieser Kategorie statistische Verfahren zur Dimensionsreduktion nutzen. Neuronale Netze werden ebenfalls bei den holistischen Verfahren eingeordnet.

Prinzipiell kann die Identifizierung einer Person anhand des Gesichts in mehrere Teilschritte unterteilt werden [21]. Jeder dieser Schritte für sich hat diverse Anwendungsgebiete und bringt eigene Herausforderungen für die genutzten Verfahren und Algorithmen mit. Zuerst muss ermittelt werden, ob überhaupt ein Gesicht in dem vorliegenden Bild vorhanden ist. Daher wird zuerst eine Gesichtsdetektion (siehe Kapitel 2.3) durchgeführt. Im nächsten Schritt werden aus dem Bildbereich mit dem erkannten Gesicht Merkmale extrahiert. Der Begriff „Merkmale“ meint nicht unbedingt konkrete Gesichtsmerkmale wie Nase, Mund und Augen. Vielmehr bezieht er sich allgemein auf extrahierte Informationen zum Vergleich beziehungsweise zur Einordnung des Gesichts[9]. Je nach verwendetem Verfahren überschneiden sich Gesichtsdetektion und Extraktion der Merkmale oder werden gemeinsam ausgeführt. Da längst nicht alle gefundenen Merkmale für eine erfolgreiche Gesichtserkennung notwendig sind - zu viele können sich sogar nachteilig auswirken - wird zudem häufig noch eine Selektion durchgeführt. Dabei werden nur die Merkmale beibehalten, welche gewisse Kriterien erfüllen. Extraktion und Selektion können auch in einem einzelnen Schritt gemeinsam ausgeführt werden. Zuletzt erfolgt die eigentliche Gesichtserkennung anhand der zuvor ausgewählten Merkmale. In Abbildung 7 ist dieser Ablauf nochmal dargestellt.

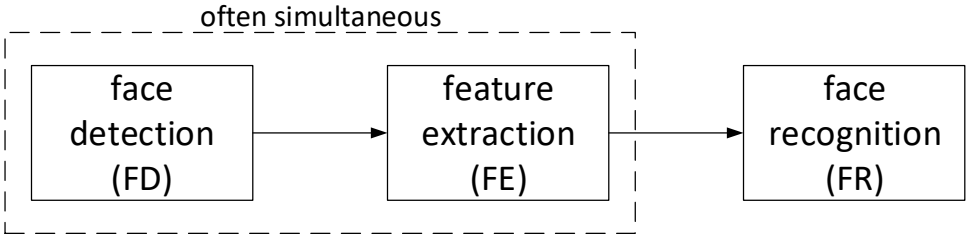


Abbildung 7: Grober Ablauf der Gesichtserkennung, zuerst die Gesichtserkennung, dann die Extraktion der Merkmale (wobei diese beiden Schritte häufig gleichzeit ablaufen) und zuletzt die eigentliche Erkennung. Bild von [36].

2.4.1. Merkmalbasierte Verfahren

Frühe Verfahren haben sich von neurologischen Erkenntnissen über die Gesichtserkennung im menschlichen Gehirn inspirieren lassen. Die Prozesse, welche dabei im Gehirn

[9] Üblicherweise repräsentiert durch einen Zahlenvektor.

ablaufen, und vor allem wo sie statt finden, waren und sind immer noch Forschungsgegenstand der Neurologie und Psychologie.

Mithilfe von modernen Verfahren, wie beispielsweise funktioneller Magnetresonanztomographie (fMRT), können die Aktivitäten einzelner Bereiche im Gehirn in Echtzeit ermittelt werden. In verschiedenen Experimenten wurden Probanden einzelne Teile eines Gesichts oder auch komplette Gesichter gezeigt. Dabei stellte sich heraus, dass spezielle Bereiche im menschlichen Gehirn zusammen arbeiten, um Gesichter zu erkennen und zu identifizieren. Am Anfang wurde angenommen, dass konkrete Merkmale eines Gesichts ermittelt werden, wie beispielsweise Augen, Nase, Mund und Ohren. Diese Merkmale werden wiederum kombiniert und von anderen Bereichen verarbeitet, bis eine Repräsentation des Gesichts im Gehirn vorliegt und identifiziert werden kann. Neben der Gesichtserkennung an sich laufen parallel dazu ähnliche Vorgänge ab, welche sich mit der Ermittlung von bestimmten Gesichtsausdrücken beschäftigen, was für automatische Gesichtserkennungssysteme ebenfalls interessant sein kann. Aufgrund dieser Erkenntnisse wurden in frühen Verfahren bestimmte Merkmale (sogenannte **Landmarks** oder auch **Fiducial Points**) aus einem Gesicht automatisch oder manuell extrahiert. Aus den Positionen und Relationen dieser Punkte zueinander ergeben sich Vektoren, welche mit einfachen Methoden zum **Pattern Matching** verglichen werden können. Hier liegt auch eine große Ähnlichkeit zu den in Kapitel 2.3 beschriebenen Verfahren zur Gesichtsdetektion mittels Template Matching vor. Ein verbreiteter Ansatz [37] aus dieser Kategorie bildet aus den Merkmalen einen Graph und erlaubt es durch Vergleichen der Graphen eine Gesichtserkennung durchzuführen. Im Allgemeinen ist das automatische ermitteln der Merkmale äußerst kompliziert und fehleranfällig [38], sowie sehr rechenaufwändig [39], weswegen aktuellere Verfahren andere Ansätze als Grundlage wählen.

2.4.2. Holistische Verfahren

Holistische - oder auch ganzheitliche - Verfahren beschränken sich nicht auf einzelne festgelegte Merkmale, sondern versuchen aus dem gesamten Bild, beziehungsweise Gesicht, relevante Informationen zu beziehen. Dieses Vorgehen stimmt mit aktuelleren Erkenntnissen aus der Neurologie über die Gesichtserkennung im menschlischen Gehirn überein [40]. Dazu werden häufig statistische Methoden genutzt, die auch im Machine Learning und der allgemeinen Datenanalyse Anwendung finden. Holistische Verfahren erzielen in der Regel höhere Genauigkeiten bei gleichem Rechenaufwand als merkmalbasierte Verfahren, zumindest wenn bei beiden die Extraktion der Merkmale automatisch abläuft. Relevante Ansätze sind hier:

Eigenfaces

Einer der erfolgreichsten frühen Ansätze zur automatischen Gesichtserkennung ohne die Nutzung konkreter Merkmale, wurde 1991 von Turk und Pentland [41] entwickelt. Das Verfahren liefert dabei letztlich eine Menge von Vektoren, welche durch Musterabgleich oder ähnliche Methoden verglichen werden können, um Gesichter einzelner Personen zu erkennen und zuzuordnen. Neben der Problematik Merkmale wie Augen, Nase, Mund und ähnliche, zuverlässig automatisch zu erkennen, waren diese für Turk und Pentland nicht ausreichend, um Gesichter effektiv zu erkennen und unterscheiden. Daher sollten alle vorhandenen Informationen aus dem Bild genutzt werden.

Das Verfahren arbeitet mit Graustufenbildern, wobei jedes Pixel als Dimension in einem Vektor angesehen wird. Ein Bild der Höhe N und Breite N wird dabei zu einem Vektor der Länge N^2. Dadurch wird das gesamte Bild als Punkt in einem NxN-Dimensionalem Raum beschrieben. Die Grundidee des Verfahrens ist es, dass Bilder der gleichen Person in diesem hochdimensionalem Raum nicht zufällig verteilt sind, sondern sich relativ nah beieinander befinden. Es sind also längst nicht alle Dimensionen nötig, um ein Gesicht ausreichend zu beschreiben. Demnach gibt es Dimensionen die wesentlich mehr zur Varianz in den Daten, also den Unterschieden zwischen Gesichtern, beitragen als andere. Um diese zu ermitteln, wird die sogenannte Hauptkomponentenanalyse, oder auch **Principal Component Analysis (PCA)** genutzt. Das Verfahren wird in [42] genauer beschrieben. Die dabei ermittelten Eigenvektoren werden auch **Eigenfaces** genannt, da sie grafisch dargestellt Gesichtern ähneln. Von diesen Eigenvektoren werden k Vektoren mit den höchsten Eigenwerten, die am meisten Varianz in den Daten erklären, ausgewählt. Diese spannen einen Unterraum auf, den sogenannten **face space**, in dem sich idealerweise alle Bilder mit Gesichtern einordnen lassen und in dem Gesichter der selben Person nah beieinander liegen. In Abbildung 8 ist dieser Unterraum vereinfacht dargestellt.

Entsprechend werden alle bekannten Gesichter in den von diesen Eigenvektoren aufgespannten Unterraum projiziert und die dadurch ermittelten Vektoren gespeichert. Um eine Person auf einem Bild zu ermitteln, wird dieses ebenfalls in den face space projiziert. Dadurch wird der N^2-Dimensionale Vektor auf k Dimensionen reduziert. Turk und Pentland haben für 16 Gesichter (also bei 16 möglichen Hauptkomponenten beziehungsweise Eigenfaces) sieben ausgewählt. Mit diesem wesentlich kleineren Vektor kann dann die eigentliche Klassifizierung durchgeführt werden. Der einfachste Ansatz ist hier die euklidische Distanz zu den anderen bekannten Vektoren. Befindet sich die geringste Distanz unter einem vorher bestimmten Grenzwert, wird angenommen, dass die durch diese beiden Vektoren beschriebenen Gesichter zur gleichen Person gehören. Falls die Distanz über dem Grenzwert liegt, gilt die Person als unbekannt. Tests mit 2.500

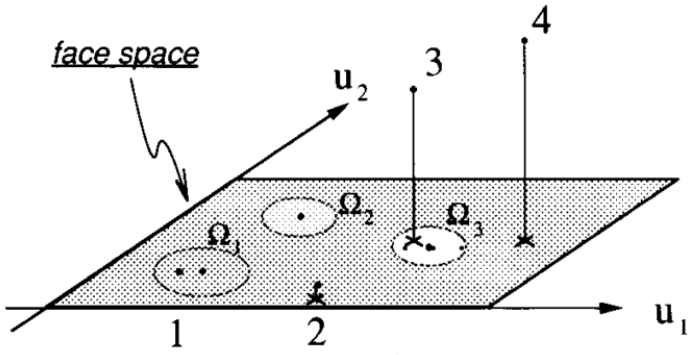


Abbildung 8: Vereinfachte Darstellung des face space. $\mathbf{u}_1$ und $\mathbf{u}_2$ sind die aufspannenden Eigenfaces, $\Omega 1 - 3$ sind bekannte Gesichter und die Punkte 1-4 sind Projektionen von Bildern in den face space. Die Punkte von Bildern auf denen keine Gesichter abgebildet sind (3 und 4), befinden sich weiter entfernt vom face space als andere Punkte (1 und 2). Bild von [41].

Bilder von 16 Personen mit unterschiedlichen Beleuchtungsbedingungen und Kopforientierungen haben ergeben, dass das Verfahren relativ robust gegenüber Änderungen der Beleuchtung ist. Bei unterschiedlichen Größenskalierungen verschlechtert sich die Leistung hingegen drastisch.

Eigenfaces bildet aufgrund seiner einfachen Umsetzung mit akzeptabler Leistung (Erkennungsraten von über 90 % bei kontrollierten Bedingungen) und Geschwindigkeit die Basis für viele Erweiterungen und Verbesserungen. Dabei stellt das Verfahren wie bereits erwähnt nur eine Selektion von relevanten Merkmalen dar, welche auf unterschiedliche Arten klassifiziert werden können.

Fisherfaces

Ähnlich wie bei Eigenfaces, wurde von Belhumeur *et al.* [43] ein Verfahren entwickelt, bei dem auf Bilddaten eine Dimensionsreduktion vorgenommen wird. Die Motivation war hierbei, ein Verfahren zu entwickeln, welches robuster gegenüber starken Änderungen in den Beleuchtungsbedingungen und Gesichtsausdrücken ist. Wie auch bei Eigenfaces ist die Grundannahme, dass sich Aufnahmen von Gesichtern in einem Unterraum mit wesentlich weniger Dimensionen befinden[10] als der gesamte Bildraum. Eine weitere Grundannahme besagt, dass manche Bereiche in einem Gesicht starke Varianzen zwischen verschiedenen Aufnahmen aufweisen, die allerdings von Beleuchtungsunterschieden und unterschiedlichen Gesichtsausdrücken stammen. Diese sind daher für die Gesichtserkennung ungeeignet. Das Verfahren zielt daher darauf ab, bei der Dimensi-

[10] Beziehungsweise durch diesen beschrieben werden können.

onsreduktion und sich daraus ergebenden Projektion diese Bereiche nicht mit einzubeziehen. Der von Belhumeur *et al.* entwickelte Ansatz basiert auf der Fisher'schen Diskriminanzfunktion, welche Teil der linearen Diskriminanzanalse (**Linear Discriminant Analysis (LDA)**) ist, woraus sich der Name **Fisherfaces** herleitet. Dabei wird versucht, das Verhältnis von Unterschieden innerhalb einer Klasse (Bilder einer einzelnen Person) zu den Unterschieden zwischen den Klassen (Bildern von verschiedenen Personen) zu maximieren. In Kapitel 2.2.1 wurde auf diese Begriffe bereits als Intra- und Inter-Class-Variation eingegangen. Im Vergleich dazu, die bei Eigenfaces genutzte PCA erzeugt eine Projektion, welche die Unterschiede zwischen sämtlichen Bildern maximiert. Dadurch fließen Variationen, die durch Beleuchtungsunterschiede verursacht wurden, ebenso mit in die Projektion ein wie auch die eigentlich relevanten Unterschiede zwischen den Gesichtern. Durch Nutzung von klassenspezifischen Informationen kann ein Unterraum gebildet werden, welcher genauere Klassifizierungen mithilfe des Fisher'schen Diskriminanzkriterium erlaubt. Dies dient als Kriterium zur Ermittlung der Projektion. Auch hier stellt das Verfahren lediglich die Merkmale zum Vergleich zur Verfügung, indem eine Dimensionsreduktion in einen Unterraum durchgeführt wird. Die Klassifizierung selbst kann, wie auch bei Eigenfaces, über eine einfache euklidische Distanz oder andere Metriken und Methoden erfolgen.

Für weitergehende Informationen bezüglich Verfahren die mit LDA arbeiten, sowie ein allgemeiner Vergleich zwischen Verfahren, die Projektionen in Unterräume zur Gesichtserkennung nutzen, sei auf [44–46] verwiesen. Im Allgemeinen hat sich ergeben, dass Verfahren auf Basis von LDA in der Regel besser Ergebnisse liefern als PCA-Basierte Verfahren, wie beispielsweise Eigenfaces. Letztere können allerdings bei kleineren Datensätzen bessere Erkennungsraten aufweisen [47].

Local Binary Pattern

Eine weitere Art von Merkmalen zur Gesichtserkennung sind **Local Binary Pattern (LBP)**, welche mit dem gleichnamigen Verfahren erzeugt werden. Ursprünglich von [48] zur rotationsunabhängigen Klassifizierung von zweidimensionalen Texturen gedacht, wurden LBPs von [49] erfolgreich zur Gesichtserkennung eingesetzt.

Ein Gesicht wird für das Verfahren als eine Ansammlung von Mikrostrukturen angesehen, bestehend aus Kanten, Ecken, ebenen Flächen und Punkten. Diese werden durch einen speziellen Operator zu LBPs kodiert. Dafür wird für jedes Pixel betrachtet welche Graustufen die umliegenden Pixel, auch Nachbarschaft genannt, haben. Ist der Graustufenwert niedriger als das Pixel im Zentrum, wird es zu einer Null, sonst zu einer Eins. Es wird also eine Grenzwertprüfung vorgenommen, anhand derer das Bild binarisiert wird. In Abbildung 9 ist dieser Vorgang dargestellt.

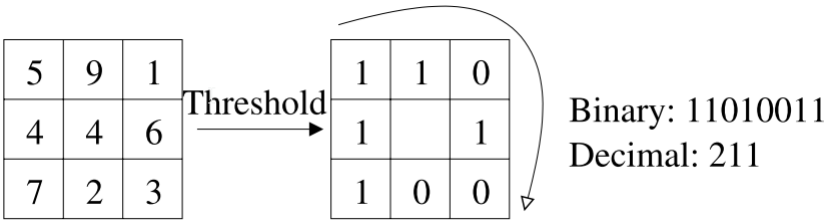


Abbildung 9: Darstellung des Vorgangs, um ein LBP aus einem Pixel und seiner Nachbarschaft zu bilden. Bild von [49].

Ein kompletter Deskriptor für ein Bild wird über ein Histogramm sämtlicher ermittelter LBPs gebildet und zur Klassifizierung genutzt. Um durch das Histogramm keine räumlichen Informationen über ein Gesicht zu verwerfen, wird das Bild zudem in mehrere gleich große Rechtecke unterteilt, von denen jeweils getrennte Histogramme berechnet werden. Um Bereichen wie Augen und Mund mehr Bedeutung beizumessen, können diese entsprechend gewichtet werden. Zusätzlich kann der Radius und die Anzahl der genutzten Pixel in der Nachbarschaft variiert werden. In [49] wurde ein Radius von zwei gewählt, sowie 16 Pixel aus der Nachbarschaft mit einer Unterteilung in 7x7 quadratische Bereiche genutzt. Mithilfe von entsprechenden Gewichtungen konnten Erkennungsraten von über 90 % erreicht werden. Neben dem Vorteil der Rotationsinvarianz ist die Extraktion der Merkmale zudem sehr schnell. Zur Klassifizierung können hier für Histogramme ausgelegte Distanzmaße genutzt werden.

Support Vector Machines

Die bisherigen Verfahren sind alle zur Extraktion und Selektion von aussagekräftigen und effizienten Merkmalen gedacht. Während diese meistens über einfache **Nearest Neighbor**-Verfahren miteinander verglichen wurden, gibt es noch eine Vielzahl weiterer Möglichkeiten zur Klassifizierung. Einer weit verbreiteter Ansatz sind **Support Vector Machines (SVMs)**, welche sich auch im Rahmen der Gesichtserkennung als sehr erfolgreich erwiesen haben [50]. Daher sollen SVMs im folgenden kurz aus [6, S. 281ff] und [3, S. 744ff] erläutert werden.

Gibt es in einem Datensatz zwei linear trennbare Klassen, gibt es eine Hyperebene, die den Datensatz perfekt trennt. Idealerweise ist der Abstand zwischen den Datenpunkte und der Hyperebene so groß wie möglich, wodurch eine gute Generalisierung des Problems stattfinden kann. Diese Ebene ist durch einige Punkte in der Nähe definiert, die sogenannten Support-Vektoren. Um eine optimale Ebene zu finden, müssen die besten Support-Vektoren gesucht werden. Ein Beispiel ist in Abbildung 10 abgebildet.

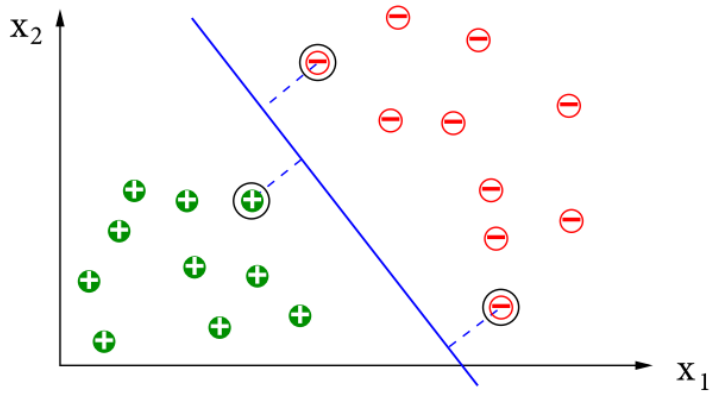


Abbildung 10: Die Trennlinie stellt die Hyperebene dar, die eingekreisten Punkte sind Support-Vektoren. Bild aus [6].

Entgegen des dargestellten Beispiels kann allerdings häufig keine lineare Trennung in den Daten vorgenommen werden. Um diese trotzdem zu klassifizieren, kann eine nichtlineare Transformation vorgenommen werden, wobei die transformierten Daten im Anschluss wieder linear trennbar sind. Zu beachten ist, dass durch die Transformation neue Dimensionen hinzukommen, durch die eine lineare Trennung wieder möglich wird. Es gilt allgemein, dass durch die Transformation in eine beliebig hohe Dimension so gut wie alle[11] Datensätze linear trennbar sind. Die Methoden welche Daten in einen höherdimensionalen Raum übertragen, werden Kernel-Methoden genannt. Durch ihre Nutzung kann die Berechnung der Support-Vektoren implizit in den höheren Dimensionen stattfinden, ohne wesentlich größeren Rechenaufwand zu betreiben. Dieser Prozess wird auch Kernel-Trick genannt. Es gibt verschiedene Kernel mit unterschiedlichen Auswirkungen auf die dadurch gefundenen Hyperebenen. Daher sind manche Kernel für bestimmte Anwendungen eher geeignet als andere. Umfassendere Informationen und mathematische Hintergründe zu SVMs und Kernel-Methoden sind in [51] zu finden.

Aufgrund des großen Erfolgs von SVMs wurden diese auch frühzeitig zur Gesichtserkennung genutzt. Prinzipiell können SVMs nur zwischen zwei Klassen unterscheiden, was für den Anwendungsfall der Gesichtserkennung jedoch nicht ausreichend ist. Um dennoch beliebig viele Klassen unterscheiden zu können, gibt es verschiedene Ansätze [52]. Beispielsweise kann für jede Klasse ein Vergleich mit dem Rest der Daten durchgeführt werden (**One-Against-All**). Beispielsweise ein Gesicht wird dann von jeder erzeugten SVM klassifiziert und das am häufigsten auftretende Ergebnis ausgewählt. Hierbei können allerdings ambivalente Ergebnisse entstehen. Ein anderer Ansatz wurde daher von [53] gewählt. Hier wird für jedes Klassenpaar eine SVM erzeugt (**One-Against-One**). Diese SVMs bilden dann die Blätter eines binären Baums. Ein Testbild

[11] Bis auf einige Ausnahmen, beispielsweise wenn Widersprüche in den Daten vorliegen.

wird zuerst in den Blättern und dann den Baum entlang an jedem Knoten erneut klassifiziert. An der Wurzel wird die endgültige Klasse ermittelt. Zudem wurde in dieser Arbeit ein direkter Vergleich von SVMs als Klassifizierer und einer klassischen euklidischen Distanz auf Eigenfaces vollzogen. Dabei zeigte sich, dass SVMs wesentlich geringere Fehlerraten und bessere Erkennungsraten liefern. Noch umfangreichere Vergleiche wurden in [54] durchgeführt. Es stellte sich heraus, dass SVMs zwar bessere Ergebnisse erzielen als klassische Verfahren, der Abstand zu LDA-basierten Methoden wie zum Beispiel Fisherfaces allerdings relativ gering ausfällt. Der Grund dafür ist, dass sowohl bei der LDA als auch in SVMs nach diskriminatorischen Eigenschaften in den Daten gesucht wird. Als Klassifizierer sind SVMs neben Neuronalen Netzen derzeit eines der leistungsfähigsten Werkzeuge zur Gesichtserkennung.

2.5. Neuronale Netze

Aufgrund der Relevanz von neuronalen Netzen zur Gesichtserkennung sollen diese in einem eigenem Unterkapitel genauer beschrieben werden. Von besonderer Bedeutung sind hier die **Convolutional Neural Networks (CNNs)**. Neuronale Netze erleben in den letzten Jahren einen enormen Aufschwung, sind aber ein lange bekanntes Konzept. Selbst die ersten Veröffentlichungen, die sich mit künstlicher Intelligenz [12] befassen, hatten neuronale Netze zur Grundlage [55]. Zwar stand die Definition einfacher logischer Operationen mithilfe von speziell angeordneten Netzen aus Neuronen im Vordergrund, jedoch wurde in der selben Veröffentlichung schon vorgeschlagen, dass lernende neuronale Netze eine Möglichkeit sind. Die meisten Algorithmen, die neuronale Netze zur Gesichtserkennung nutzen, sind dem holistischen Ansatz zuzuordnen, jedoch gibt es auch hier Ausnahmen.

Die Inspiration für neuronale Netze kommt von biologischen Neuronen, wie sie im zentralen Nervensystem von Menschen und allgemein in allen Gewebetieren zu finden sind. Neuronen, oder auch Nervenzellen, sind spezielle Zellen, welche durch chemische Reaktionen untereinander Informationen in Form elektrischer Signale austauschen. Diese sind direkt für das Denken und Handeln zuständig und können sich zudem neuen Gegebenheiten anpassen. Anhand dieses biologischen Beispiels orientieren sich künstliche neuronale Netze, indem Neuronen vereinfacht als formales Modell angesehen werden. Ein einzelnes Neuron bildet dabei die kleinste Einheit, von denen viele miteinander verknüpft werden um unterschiedliche Aufgaben abzubilden[13].

[12] Wobei der Begriff der künstlichen Intelligenz - **Artificial Intelligence (AI)** - erst 1956 erstmals verwendet wurde [3, S. 17].

[13] Im menschlichen Gehirn befinden sich etwa 10^{11} Neuronen mit jeweils 1.000 bis 10.000 Verbindungen zu anderen Neuronen [6, S. 248].

Üblicherweise hat jedes von k verschiedenen Neuronen eine Anzahl von j gewichteten Eingängen sowie einem Bias-Wert[14] b, deren Summe

$$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k \tag{1}$$

an eine Aktivierungsfunktion übergeben wird, wobei $(x_1, x_2, ..., x_j)$ die Eingangsdaten sind. Diese Aktivierungsfunktion entscheidet, ob das Neuron „feuert“, also sich das Ausgangssignal ändert. Sofern die Aktivierungsfunktion nichtlinear ist, kann das neuronale Netz auch nichtlineare Probleme lösen [3, S. 729]. Die einfachste Aktivierungsfunktion, welche schon von McCulloch und Pitts verwendet wurde [55], ist eine einfache Schwellwertfunktion,

$$y_k = \begin{cases} 1 & \text{if } v_k \geq 0, \\ 0 & \text{if } v_k < 0 \end{cases} \tag{2}$$

wobei y_k das Ausgangssignal des Neurons ist, welches an eine beliebige Anzahl von Neuronen weitergeleitet werden kann. Diese Schwellwertfunktion wird auch **Heaviside-Funktion** genannt. In Abbildung 11 ist eine Darstellung eines künstlichen Neurons zu sehen.

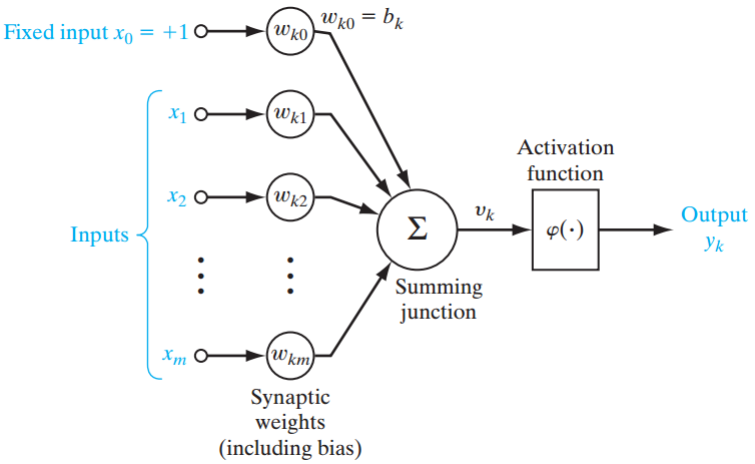
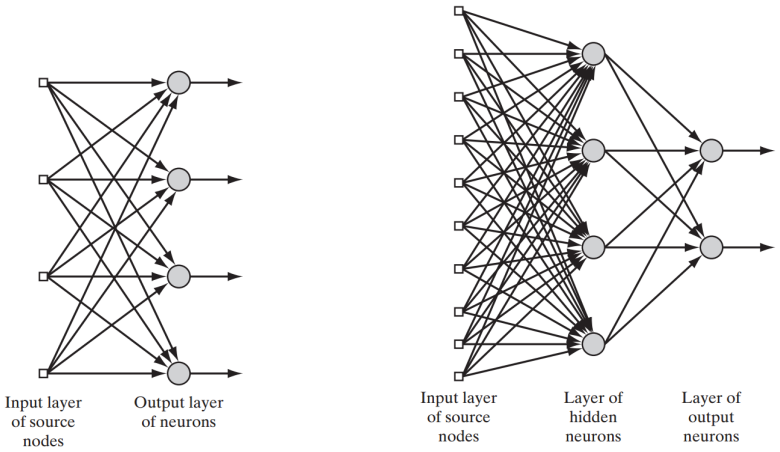


Abbildung 11: Darstellung eines Neurons, wobei y_k das Ergebnis der Aktivierungsfunktion φ, angewendet auf die Summe v_k der gewichteten Eingangssignale, ist [5, S. 42].

Ein Problem der Schwellwertfunktion ist, dass sie nicht differenzierbar ist. Erst durch differenzierbare Aktivierungsfunktionen, welche weiter unten beschrieben werden, sind effiziente Lernalgorithmen für neuronale Netze möglich.

[14] Der Bias-Wert stammt von einem impliziten Bias-Neuron und beeinflusst den summierten Wert, welcher nötig ist um das Neuron zu aktivieren [56].

Es gibt verschiedene Arten von Topologien oder Architekturen, die den Aufbau von neuronalen Netzen bestimmen. Dabei wird grundlegend unterschieden, ob die Neuronen und ihre Ausgangssignale im Netzwerk strikt nach vorne weitergeleitet werden (**Feed-Forward-Network**), oder ob sie Signale auch an zurückliegende Neuronen leiten können (**Recurrent-Neural-Network**) [5, S. 51ff]. Für Mustererkennung und in diesem Fall auch Gesichtserkennung sind hauptsächlich Feed-Forward-Netzwerke relevant, weswegen sich im Folgenden ausschließlich auf diese bezogen wird. Üblicherweise werden solche Netzwerke in einzelnen Schichten, beziehungsweise **Layer**, aufgebaut, von denen jeweils eine Schicht ihre Ausgangssignale an die nächste weiterleitet. Der einfachste Aufbau besteht dabei aus einer einzelnen Schicht von einem oder mehreren Neuronen. Die Eingangsknoten werden dabei nicht als eigene Schicht gezählt. Sollte ein Netzwerk aus mehreren Schichten bestehen, werden die mittleren (welche keine Eingangs- oder Ausgangsneuronen des Netzes enthalten) als versteckte Schichten (**Hidden Layer**) bezeichnet. Von historischer Bedeutung ist hier der **Perceptron**, welcher in den 1950ern komplett in Hardware gebaut wurde und in der Lage war, Bilder von einer Matrix aus 20x20 Photorezeptoren zu klassifizieren [57]. Aus diesem Grund werden ein- und mehrschichtige neuronale Netze häufig auch als **Single Layer Perceptron (SLP)** und **Multi Layer Perceptron (MLP)** bezeichnet. Ein Vergleich dieser beiden Netzwerkarten ist in Abbildung 12 dargestellt.



(a) Single Layer Perceptron (b) Multi Layer Perceptron

Abbildung 12: Vergleich von SLPs und MLPs [5, S. 51f].

Ein SLP ist dazu in der Lage, Daten durch eine Hyperebene linear zu trennen. Mit nur einem Neuron können zwei Klassen in den Daten unterschieden werden. Mit mehreren Neuronen können entsprechend viele Klassenunterscheidungen vorgenommen werden. Sind die Daten nicht linear trennbar, kann der **Perceptron-Convergence**-Algorithmus [5, S. 84] keine Lösung finden, also nicht konvergieren. Durch eine Kritik von [58] an der fehlenden Fähigkeit von Perceptrons und den mit ihnen eingeführten Lernalgorithmus

auch nichtlineare Probleme zu lösen, geriet die Forschung an neuronalen Netzen lange in den Hintergrund. Erst durch ein Zusammenspiel von mehreren Komponenten konnte ihre Entwicklung, vor allem von mehrschichtigen Netzen, wieder voran getrieben werden. Grundlegend dafür war die Einführung des **Back-Propagation**-Algorithmus [59] in den 1980ern. Seine Entwicklung war nötig, weil der Perceptron-Convergence-Algorithmus nicht dazu in der Lage ist, Gewichtskorrekturen auch an Gewichten von versteckten Schichten durchzuführen. Dadurch können mehrschichtige Netzwerke mit diesem Algorithmus nicht effizient trainiert werden. Die Idee des Back-Propagation-Algorithmus ist es, zu betrachten wie sehr sich eine Veränderungen der Gewichtung eines Neurons auf die Fehlerrate des Netzwerks auswirkt. Dazu wird zuerst eine Funktion benötigt, um den Gesamtfehler als Summe der Fehler der Ausgangsneuronen zu berechnen. Diese wird auch als Fehlerfunktion bezeichnet[15]. Die einfachste Methode hierbei ist der quadratische Fehler, auch **Square Loss** genannt,

$$E = \frac{1}{2}\sum_{c}\sum_{j}(y_{j,c} - d_{j,c})^2 \tag{3}$$

wobei c der Index über ein Eingabebeispiel ist, j der Index über die Ausgabeneuronen, y der Ausgangswert des Neurons sowie d als der gewünschte Ausgabewert. Um eine Fehlerfunktion für den Back-Propagation-Algorithmus nutzen zu können, müssen zwei Eigenschaften erfüllt sein:

1. Die Fehlerfunktion E muss als Durchschnitt über die einzelnen Fehler der Ausgangsneuronen für ein Trainingsbeispiel x definierbar sein.

$$E = \frac{1}{n}\sum_{x} E_x \tag{4}$$

2. Die Kostenfunktionen darf nur direkt abhängig von Ausgangswerten der Ausgabeneuronen, also der letzten Schicht, sein.

In der Literatur gibt es verschiedene Fehlerfunktionen, welche unterschiedliche Auswirkungen auf das Lernverhalten haben. Eine der am weitesten verbreiteten ist die Kreuzentropie, oder auch **Cross Entropy**,

$$E = -\frac{1}{n}\sum_{x}\sum_{j}[d_j \ln y_j + (1-d_j)\ln(1-y_j)] \tag{5}$$

[15] In der Fachliteratur sind sowohl die Begriffe **Loss Function**, **Error Function** oder auch **Cost Function** zu finden.

wobei y wieder der letzten Neuronenschicht entspricht. Durch partielle Ableitungen dieser Kostenfunktionen in Bezug auf die Gewichtungen [16], lässt sich so ein Gradient berechnen, welcher angibt in welche „Richtung“ die einzelnen Gewichte angepasst werden müssen, um die Fehlerfunktion zu minimieren. Wichtig ist hierbei, dass aufgrund der partiellen Ableitungen die Aktivierungsfunktion der Neuronen differenzierbar sein muss. Dies ist für die weiter oben beschriebene Schwellwertfunktion 2 nicht gegeben. Stattdessen wird von [59] für eine gewichtete Eingabesumme v_k des Neurons k (siehe Gleichung 1) die Sigmoid-Funktion

$$y_k = \frac{1}{1+e^{-v_k}} \tag{6}$$

verwendet, welche auch aktuell noch Verwendung findet. Ähnlich dazu ist der Tangens Hyperbolicus:

$$y_k = \tanh(v_k) = \frac{e^{v_k} - e^{-v_k}}{e^{v_k} + e^{-v_k}} \tag{7}$$

Während die Sigmoid-Funktion Werte zwischen 0 und 1 ausgibt nimmt der Tangens Hyperbolicus Werte zwischen -1 und +1 an. In aktuelleren Arbeiten wird häufig eine weitere Aktivierungsfunktion verwendet, die sogenannte **Rectified Linear Unit (ReLU)**

$$y_k = \max(0, v_k) \tag{8}$$

welche einfacher zu berechnen und näher am biologischen Vorbild ist. Zudem liefert sie bessere Ergebnisse bei typischen Bilderkennungsproblemen liefert, zu denen auch die Gesichtserkennung gehört [25, 61–63]. In Abbildung 13 ist ein Vergleich der verschiedenen Aktivierungsfunktionen dargestellt.

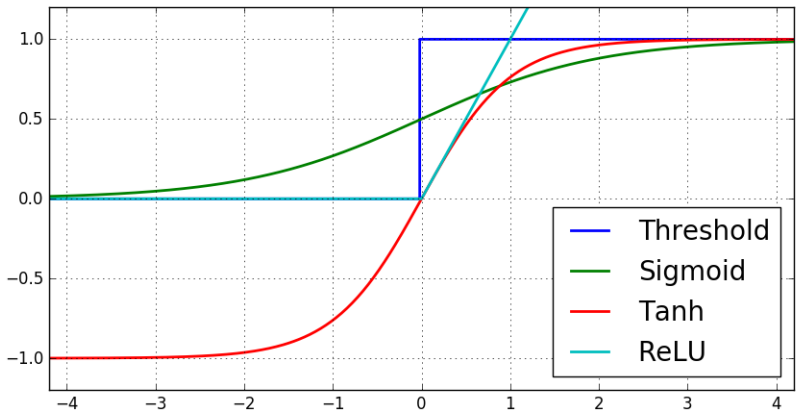


Abbildung 13: Vergleich der verschiedenen Aktivierungsfunktionen.

[16] Für die genauen mathematischen Hintergründe sei auf [3, 5, 6, 56, 60] verwiesen.

Weiter oben wurde bereits beschrieben, wie mithilfe der Fehlerfunktion und des Back-Propagation-Algorithmus ein Gradient berechnet werden kann. Dieser gibt an, in welche Richtung die Gewichte angepasst werden müssen, um die Funktion zu minimieren. Es ist zum Verständnis hilfreich, sich eine Fläche vorzustellen, welche durch die Fehlerfunktion definiert wird. Eine bestimmte Kombination von Gewichtungen gibt einen Punkt auf dieser Fläche an. Um die Fehlerfunktion zu minimieren, muss ein Minimum in dieser Fläche gefunden werden. Der Gradient gibt die Richtung des steilsten Anstiegs an. Werden die Gewichte in die entgegengesetzte Richtung mit dem Betrag des Gradienten bewegt, nähert sich die Fehlerfunktion automatisch einem Minimum. Daher wird dieses Verfahren auch Gradientenabstieg (**Gradient Descent**) genannt. Wichtig ist, dass dabei nicht unbedingt ein globales Minimum gefunden wird. Das Verfahren kann auch auf einem lokalen Minimum konvergieren. Um die Lernrate zu beeinflussen, also die Größe der Schritte mit der die Gewichte angepasst werden, wird der Betrag des Gradienten zudem mit einem Wert η multipliziert. Zur Veranschaulichung ist der Ablauf des Verfahrens in Abbildung 14 visualisiert.

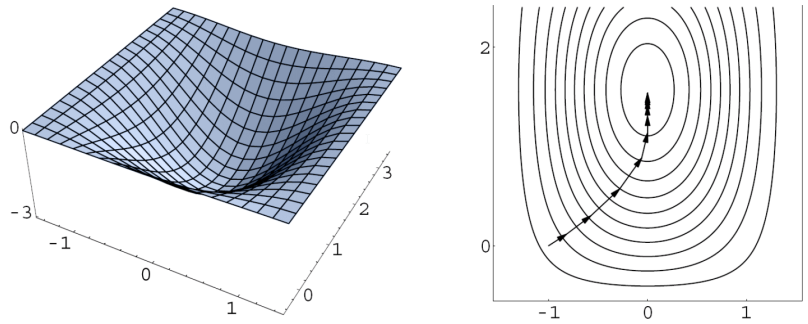


Abbildung 14: Visualisierung des Gradientenabstiegs. Links die Fläche einer zweidimensionalen Fehlerfunktion. Rechts ist der Abstieg anhand von Höhenlinien dargestellt. Umso steiler der Anstieg des Gradienten, umso größer ist die Schrittweite mit der sich das Verfahren dem Minimum nähert. Bild aus [60].

Mit diesen Grundlagen lassen sich bereits neuronale Netze konstruieren, die mit entsprechenden Trainingsdaten Ergebnisse erreichen, die gleich oder besser als von flachen Verfahren (unabhängig von der Thematik) erzielte Ergebnisse. Mit einigen Verbesserungen an den Fehlerfunktionen, dem Back-Propagation-Algorithmus und Gradientenabstieg können neuronale Netze auf vielen Gebieten geringere Fehlerraten und bessere Leistung erbringen als die meisten Algorithmen oder sogar Menschen. Eine Erweiterung von neuronalen Netzen speziell für Probleme der Bild- und Mustererkennung sind CNNs.

2.5.1. Convolutional Neural Networks

Bisher wurden neuronale Netze losgelöst vom Kontext der Gesichtserkennung betrachtet. Speziell in dem Gebiet der Bilderkennung können diese dabei als reine Klassifizierer genutzt werden. Die in Kapitel 2.4 beschriebenen Verfahren können dafür zur Extraktion von Merkmalen genutzt werden. Alternativ dient das gesamte Netz sowohl zur Extraktion als auch zur Klassifizierung. Üblicherweise werden dafür mehrschichtige neuronale Netze genutzt. Um dabei akzeptable Erkennungsraten zu erreichen, sind entsprechend große und tiefe Netzwerke nötig, welche viel Rechenzeit zum Trainieren brauchen. Zusätzlich sind solche umfangreichen Netze anfällig für Überanpassung und langsame Lernraten aufgrund von ungünstigen Eigenschaften des Gradientenabstiegs [64]. Um diese Probleme zu umgehen und die Geschwindigkeit des Lernprozesses zu steigern, wurden CNNs entwickelt. Sehr einflussreich ist die Arbeit von [65]. Die Grundidee ähnelt dabei wieder den biologischen Abläufen bei der Bilderkennung. Speziell die Forschung von [66] aus den 1960ern bezüglich des visuellen Cortex von Katzen diente bei der Konzipierung als Vorbild.

Es gibt drei Konzepte bezüglich der Architektur von CNNs, die sie von normalen neuronalen Netzen unterscheiden:

- **Lokale rezeptive Felder**
  Anstatt wie üblich einem Neuron in einer versteckten Schicht sämtliche Ausgangsdaten der Neuronen in der davor liegenden Schicht als Eingangssignal zu liefern, werden nur Signale von lokal begrenzten Neuronen weitergeleitet. Dadurch können Informationen über räumliche Begebenheiten in einem Bild erlernt werden. Zu beachten ist, dass diese rezeptiven Felder in einem zwei- oder mehrdimensionalem Raum liegen, also nicht mehr wie bisher auf einem Vektor beschränkt sind. An der technischen Umsetzung mithilfe von Vektoren ändert sich dadurch jedoch nichts. Abbildung 15 verdeutlicht den Aufbau eines rezeptiven Feldes.

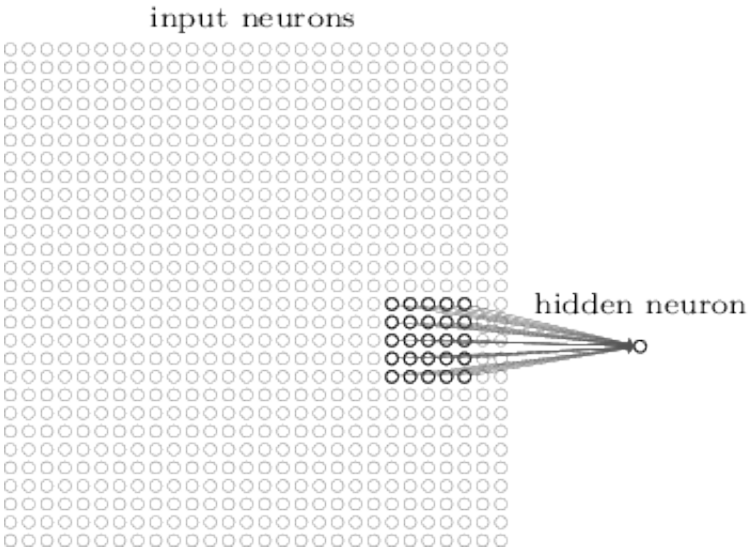


Abbildung 15: Darstellung eines lokalen rezeptiven Feldes. Im Beispiel umfasst das Feld 5x5 Pixel, in der Praxis können die Felder auch größer ausfallen, je nach Größe der Eingangsdaten. Bild aus [56].

- **Geteilte Gewichte**

  Eine weitere Besonderheit von CNNs ist, dass sich die Neuronen in den einzelnen Schichten ihre Gewichtungen teilen. Es gibt also, aufgrund der eben beschriebenen rezeptiven Felder eine feste Matrix an Gewichten, die für jedes einzelne Neuron in den versteckten Schichten angewendet wird. Die Neuronen einer einzelnen Schicht erkennen, beziehungsweise reagieren, somit immer auf das gleiche Muster in den Eingangsdaten, nur an jeweils verschiedenen Positionen. Die mathematische Operation zur Berechnung des Eingangssignals aus den geteilten Gewichten wird auch als Konvolution bezeichnet, wodurch sich der Name Convolutional Neural Networks ergibt. Neben der Reduktion von benötigtem Speicherplatz für die Gewichte hat dies zur Folge, dass das neuronale Netz robust gegenüber Verschiebungen innerhalb der Eingangsdaten ist. Das erkannte Muster kann im Kontext der Gesichtserkennung auch als Merkmal betrachtet werden. Die versteckte Schicht, welche das Merkmal repräsentiert, wird als **Feature Map** bezeichnet [56]. Die Matrix von geteilten Gewichten wird häufig auch als **Kernel** oder Filter bezeichnet. Um mehr als nur ein Merkmal erkennen zu können, müssen mehrere Schichten mit unterschiedlichen Gewichtungsmatrizen auf eine Eingangsschicht folgen. In Abbildung 16 ist dieser Vorgang dargestellt.

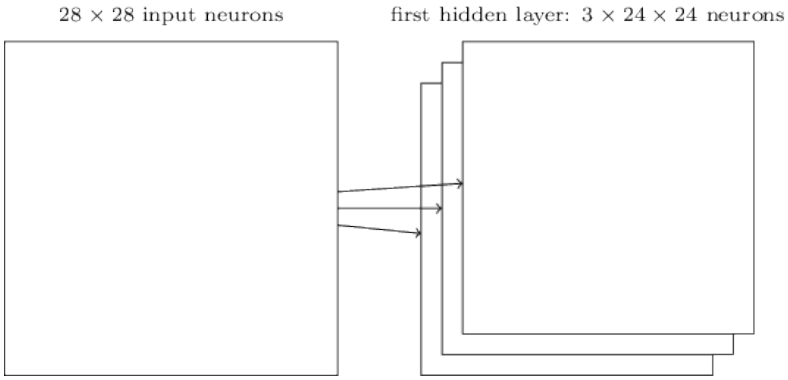


Abbildung 16: Darstellung der verschiedenen versteckten Schichten in CNNs, die es ermöglichen unterschiedliche Merkmale zu erkennen. Bild aus [56].

- **Pooling**

  Nach der Berechnung der einzelnen Feature Maps wird das **Pooling** durchgeführt. Dabei werden konkrete Informationen über lokale Gegebenheiten in den Eingangsdaten zusammengefasst. Grund dafür ist zum einen der verminderte Rechenaufwand in den folgenden Schichten, sowie weitere Robustheit gegenüber lokalen Verschiebungen [67, S. 339]. Es gibt verschiedene Arten von Pooling, die populärste davon ist das **Max-Pooling**. Dabei wird ein 2x2 (oder auch mehr) Pixel großes Fenster über die Ausgangssignale einer Konvolutionsschicht geschoben. Der jeweils größte Wert ist dann der Wert für ein Pixel der Pooling-Schicht (siehe Abbildung 17).

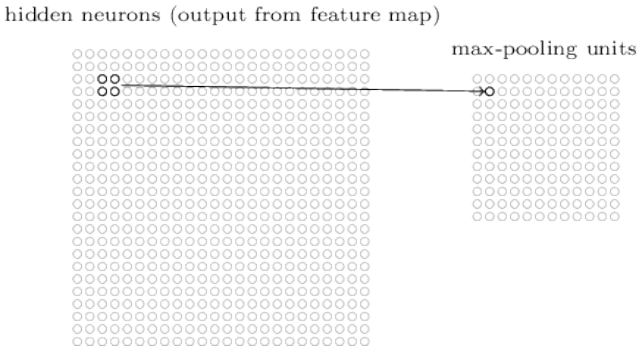


Abbildung 17: Darstellung des Poolings. Bei Max-Pooling wird der größte Wert aus den vier markierten Pixeln aus der linken Schicht in der rechten Schicht übernommen. Bild aus [56].

Diese drei Grundbausteine lassen sich auf verschiedene Arten kombinieren und variieren, wodurch das Verhalten eines CNN angepasst werden kann. Ein Beispiel, welches alle eben erklärten Konzepte beinhaltet, ist in Abbildung 18 dargestellt. Die letzte Schicht besteht dabei aus normalen Neuronen und gibt die Klassifikation des Netzwerks wieder.

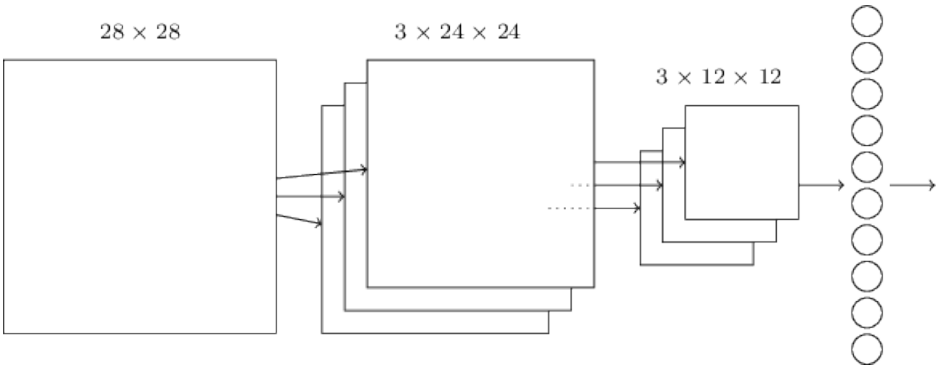


Abbildung 18: Darstellung eines einfachen CNN. Zuerst die Eingangsschicht, danach erfolgt eine Konvolutionsschicht, dann eine Pooling-Schicht und zuletzt die Ausgangsschicht mit normalen Neuronen. Bild aus [56].

Seit der Einführung von CNNs wurde der Ansatz vielfältig erweitert und ausgebaut. Eine komplette Übersicht dieser Techniken würde weit über den Rahmen der vorliegenden Arbeit hinausgehen. Neben den bereits genannten Quellen sei daher als aktuelles und umfassendes Werk das Buch **Deep Learning** genannt [67].

2.5.2. Aktuelle Entwicklung

Der Einsatz von CNNs zur effizienten Lösung relevanter Aufgaben begann 1998 als [65] ein CNN namens **LeNet-5** beschrieben, welches den MNIST-Datensatz mit einer Fehlerrate von 0,8 % richtig klassifizierte. Ein davon leicht abgewandeltes System wurde im Anschluss von Banken genutzt, um handgeschriebene Schecks maschinell zu erfassen. Ansätze, um CNNs zur Gesichtserkennung zu nutzen gab es sogar schon 1997, wenn auch unter den relativ einfachen Bedingungen des in Kapitel 2.2 beschriebenen AT&T-Datensatzes[17] [39]. In den breiten Fokus der Öffentlichkeit gerieten CNNs allerdings erst wieder ab 2012, als Forscher von Google ein stark modifiziertes CNN entwickelten, welches eine enorme Verbesserung der Erkennungsrate auf den **ImageNet**-Daten mit sich brachte [68]. ImageNet stellt eine Datenbank von mehreren Millionen Bildern in tausenden Kategorien zur Verfügung [69]. Dieses CNN erreichte dabei einer Erkennungsrate von 15,8 %, was zwar wenig scheint, im Vergleich zum vorherigen Rekord von 9.3 % jedoch eine enorme Steigerung darstellt. Darauf aufbauend entwickelten [25] ein ähnliches Netzwerk und testeten es unter den offiziellen Testbedingungen der jährlich abgehaltenen **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)**, und konnten dort wieder den Rekord brechen. Dabei wurden Erkennungsraten von 84,7 % im Vergleich zum vorherigen Rekordwert von 73,8 % erreicht. Seitdem dominieren CNNs den

[17] Früher war dieser auch als ORL-Datensatz bekannt.

jährlich abgehaltenen Wettbewerb und erreichen jedes mal neue Höchstwerte. Im Jahr 2014 konnten Forscher von Google mit einem CNN, genannt **GoogLeNet** in Anlehnung an LeNet, eine Erkennungsrate von 93,33 % erreichen [70]. Dabei ist anzumerken, dass selbst Menschen nur mit hohem Zeit- und Lernaufwand ähnliche Erkennungsraten erreichen können [71].

Auch in der Gesichtserkennung und -detektion haben CNNs in den letzten Jahren große Fortschritte erzielt und die meisten flachen Verfahren überholt. Bereits im Jahr 2004 wurde von [72] ein CNN zur Gesichtsdetektion in Bildern entwickelt. Dieses ist zwar etwas langsamer als beispielsweise das Verfahren von Viola und Jones (siehe Kapitel 2.3), weist dafür jedoch bessere Detektionsraten auf. Ein CNN zur Gesichtserkennung wurde 2005 von [73] entwickelt. Als Datensatz wurde die PIE-Datenbank verwendet, speziell die Frontalaufnahmen. Unter diesen strikten Bedingungen zeigte sich, dass bereits 25 Bilder pro Person ausreichen, um eine Erkennungsrate von 98 % zu erreichen. Ab dem Jahr 2014 nimmt die Anzahl der Forschungsarbeiten zu dem Thema stark zu, eine gute Zusammenfassung bietet [74]. Im Jahr 2015 haben [75] mit einem vergleichsweise simplen aber dafür tiefen CNN mit 37 Schichten eine Erkennungsrate von 98,95 % auf dem LFW-Datensatz erreicht. Ebenso wurde ein Verfahren entwickelt, um halbautomatisch extrem große Datensätze zu erzeugen, welche ähnlich umfangreich sind wie die nicht öffentlich verfügbaren Datensätze von Facebook oder Google. Des Weiteren ist das **OpenFace**-Projekt erwähnenswert [76], welches neben dem Modell eine Softwarebibliothek zum Trainieren von neuronalen Netzen bereitstellt. Eine Forschungsgruppe von Facebook entwickelte mit **DeepFace** ein CNN, welches den Datensatz von per Hand markierten Gesichtern im eigenen sozialen Netzwerk ausnutzt. Auf dem LFW-Datensatz erreicht DeepFace eine Genauigkeit von bis zu 97,35 % [77]. Die nach aktuellem Stand besten Ergebnisse liefert ein CNN von Google, genannt **FaceNet** [78]. Auf dem anspruchsvollem LFW-Datensatz erreicht es eine Erkennungsrate von 99,63 %. Dazu muss erwähnt werden, dass zum Training über 200 Millionen Bilder verwendet wurden. Die Trainingsphasen wurden auf einem CPU-Cluster durchgeführt und dauerten 1.000 bis 2.000 Stunden.

Während viele Forschungsgruppen eigene **Frameworks** zum Training und Benutzen von neuronalen Netzen entwickelten, wächst die Anzahl von frei verfügbaren Softwareprojekten stetig. Da eine effiziente Umsetzung der benötigten Algorithmen für eine Nutzung von CNNs oder auch klassischen flachen Verfahren unter realen Bedingungen unabdingbar ist, sollen im Anschluss einige der verfügbaren Frameworks vorgestellt werden.

2.6. Frameworks

Zum Abschluss dieses Kapitels werden verschiedene Frameworks aufgelistet und betrachtet, welche Verfahren zur Gesichtserkennung und Bildverarbeitung bereitstellen. Da das im Rahmen dieser Arbeit entwickelte System zur Gesichtserkennung quelloffen und kostenlos sein soll, werden keine kommerziellen Produkte betrachtet. Zudem ist eines der Ziele unabhängig von externen Schnittstellen zu sein. Deswegen werden Dienste, die auf ihren Servern im Internet eine Gesichtserkennung ermöglichen, ebenfalls nicht vorgestellt oder genutzt.

2.6.1. OpenCV

Eines der bekanntesten Frameworks für allgemeine Bildverarbeitung ist **OpenCV** [79]. Das Framework steht unter der BSD-Lizenz und ist somit quelloffen und kostenlos nutzbar. Es bietet umfangreiche Methoden zur Bildverarbeitung und kann sowohl in C, C++, Java als auch Python genutzt werden. Zur Gesichtsdetektion steht ein Viola-Jones-Detektor zur Verfügung. Zudem sind die Verfahren Eigenfaces, Fisherfaces und Local Binary Pattern implementiert. Allerdings sind diese in sich geschlossen, es ist also nicht möglich beispielsweise die extrahierten Features mit anderen Distanzmaßen abgesehen von der euklidischen Distanz zu vergleichen ohne den Quelltext zu ändern.

2.6.2. Dlib

Dlib ist ein Framework, welches Algorithmen zur Bildverarbeitung und für Machine Learning bereitstellt [80]. Es enthält im speziellen einige auf Gesichtserkennung ausgerichtete Methoden. Diese umfassen einen Gesichtsdetektor auf Basis des HOG, sowie eine Implementierung eines neuronalen Netzes zur Gesichtserkennung. Zu diesem werden auch bereits trainierte Gewichte bereitgestellt. Dabei ist dieses Netzwerk nicht zur Klassifizierung gedacht, sondern extrahiert Merkmale, welche zum Trainieren von Klassifizierern genutzt werden können. Das Framework steht unter der `Boost Software License`.

2.6.3. Scikit-Learn

Ein weiteres quelloffenes und kostenloses Framework ist **Scikit-Learn** [81], wobei der Fokus im Gegensatz zu OpenCV allgemein auf Machine Learning liegt. Sämtliche Bausteine, um beispielsweise Eigenfaces oder Fisherfaces mit wenig Aufwand zu implementieren, liegen vor. Zusätzlich gibt es eine große Menge an Klassifizierern sowie

Werkzeuge, um automatisch die besten Parameter für ein Verfahren zu finden. Außerdem bietet Scikit-Learn die Möglichkeit verschiedenste Arten von Evaluierungen und Vorverarbeitungen von Daten vorzunehmen. Es stellt keine Methoden zur direkten Gesichtsdetektion oder Erkennung bereit. Dennoch lohnt sich die Integration von Scikit-Learn in das zu entwickelnde System da es viele nützliche Werkzeuge zur Verfügung stellt. Die Entwicklung erfolgt wie bei den meisten hier vorgestellten Frameworks in der Programmiersprache Python.

2.6.4. FaceRec

Vom selben Entwickler, welcher auch die Algorithmen zur Gesichtserkennung in OpenCV entwickelt hat, gibt es ein eigenständiges Framework, welches diverse Verfahren implementiert [82]. Speziell gibt es hier eine Unterscheidung zwischen der Extraktion von Merkmalen und der eigentlichen Klassifizierung. Für ersteres stehen unter anderem Eigenfaces, Fisherfaces und LBPs zur Verfügung. Als Klassifizierer stehen **k-Nearest-Neighbors (kNN)** mit verschiedenen Distanzmaßen, sowie SVMs zur Verfügung. Ebenfalls gibt es Methoden zur Validierung der gelernten Modelle. Da diese beliebig kombiniert werden können, bietet FaceRec mehr Freiheiten als OpenCV, nutzt selbiges aber zur Gesichtsdetektion. Das Framework kann sowohl mit Python als auch mit **MATLAB** genutzt werden.

2.6.5. OpenFace

Wie bereits oben erwähnt, bietet OpenFace neben einer Architektur für ein neuronales Netz auch ein Framework um CNNs zu trainieren [76]. Zudem sind ebenfalls Werkzeuge zur Evaluierung vorhanden. Außerdem gibt es viele Beispiele und mehrere Modellbeschreibungen. Einzig die offizielle Dokumentation ist nicht sehr umfangreich. Die Entwicklung erfolgt ebenfalls in Python, wobei das Projekt selber auf einem anderen Framework zur Implementierung von neuronalen Netzen basiert. Intern wird **Torch** genutzt, welches ein Framework speziell zur effizienten Verarbeitung von Daten auf Grafikkarten ist.

2.6.6. Sonstige

Neben den bisher erwähnten Projekten gibt es insbesondere zur Entwicklung von neuronalen Netzen mehrere Frameworks. Da sie sich in den grundlegenden Eigenschaften und Zielsetzung nicht groß unterscheiden, sollen diese hier zusammen genannt werden. Das **Caffe**-Projekt wird seit 2014 am **Berkeley Artificial Intelligence Research Lab**

entwickelt [83]. Zur Definition von neuronalen Netzen wurde eine eigene Definitionssprache eingeführt. Erwähnenswert ist hier der sogenannte **Model Zoo**, eine Sammlung von vortrainierten Modellen, die von Benutzern erweitert werden kann. Ein weiteres Framework mit dem neuronale Netze trainiert werden können, ist **Theano**. Der grundlegende Anwendungszweck des Frameworks ist in erster Linie als Werkzeug zur effizienten, **Graphics Processing Unit (GPU)**-unterstützten Berechnung von mathematischen Ausdrücken, die mit Matrizen arbeiten. Da dies auf die meisten mathematischen Operationen von neuronalen Netzen zutrifft, ist dieses Framework dafür gut geeignet [84]. Zum Schluss soll noch **TensorFlow** erwähnt werden, welches aktiv im Umfeld von Google entwickelt wird [85]. Die beiden vorher genannten Frameworks arbeiten mehr oder weniger direkt mit Matrizen, beziehungsweise Tensoren. Auf diesen aufbauend existiert ein weiteres Framework namens **Keras** [86]. Dieses abstrahiert die Arbeit mit den Tensoren und stellt einfache Methoden zur Verfügung, um schnell neuronale Netze zu entwickeln und zu testen. Alle der hier genannten Frameworks besitzen von Grund auf Unterstützung zur Berechnung mithilfe einer oder mehrerer GPUs, was den Zeitaufwand zum Trainieren von neuronalen Netzen enorm verringern kann. Zudem können sie mit Python genutzt werden, was eine nahtlose Integration von anderen Frameworks wie beispielsweise OpenCV oder Scikit-Learn ermöglicht.

3. Vergleich der Gesichtserkennungsverfahren

Im vorhergehenden Kapitel wurden mehrere Verfahren zur Gesichtserkennung beschrieben, die in dieser Arbeit miteinander hinsichtlich ihrer Eignung für ein Gesichtserkennungssystem im Heimautomatisierungsumfeld verglichen werden sollen. In diesem Kapitel soll zunächst näher auf die Implementierung der Verfahren eingegangen werden. Im Anschluss erfolgt die Vorstellung der für den Vergleichen genutzten Datensätze. Aus diesen ergeben sich Vorbereitungen und Annahmen hinsichtlich der Daten auf denen eine Testreihe durchgeführt werden sollen. Des Weiteren soll kurz die eigentliche Durchführung der Testreihe und die dafür genutzte Hardware beschrieben werden. Danach erfolgt die Auswertung der erfassten Testergebnisse, auf deren Basis zuletzt eine Auswahl erfolgen soll. Dabei gilt zu beachten, dass nicht unbedingt ein einzelnes Verfahren in allen Fällen die besten Ergebnisse erreicht. Sollten für unterschiedliche Bedingungen verschiedene Verfahren besser geeignet sein, können auch mehrere für den Prototyp verwendet werden. Wie bereits in Kapitel 2.4 erläutert, lassen sich Verfahren zur Gesichtserkennung in verschiedene Teilschritte gliedern. Der Schritt der Gesichtsdetektion wird innerhalb der Vorverarbeitung durchgeführt. Die Gesichtserkennung besteht damit zum einen aus der Extraktion von Merkmalen, diese Verfahren werden im Folgenden auch **Feature Extractor (FE)** genannt. Zum anderen erfolgt im Anschluss die Klassifizierung. Verfahren zur Feature Extraction und Klassifizierung lassen sich beliebig kombinieren und werden im folgenden als **Kombinationsverfahren** bezeichnet. Neben diesem zweiteiligen Ablauf sind neuronale Netze in der Lage, beide Schritte direkt auszuführen. Dabei werden durch die ersten Schichten Merkmale aus den Bildern extrahiert und in den letzten klassifiziert. Diese Verfahren werden im Folgenden als **All In One (AIO)**-Verfahren bezeichnet. Für diese Arbeit soll ein möglichst breites Spektrum aus flachen Verfahren und moderneren tiefen Ansätzen untersucht werden. Dazu zählen unter anderem FEs, welche aus bereits trainierten neuronalen Netzen bestehen. Tabelle 1 zeigt alle in dieser Arbeit verglichenen Verfahren.

Tabelle 1: Zu vergleichende Verfahren

FE	**Klassifizierer**	**AIO**
Eigenfaces	kNN	Einfaches CNN (SCNN)
Fisherfaces	SVM	Abgestimmtes CNN (FCNN)
Local Binary Pattern	Neuronales Netz	
Dlib-Deskriptor*		
Openface-Deskriptor*		
VGG19-Deskriptor*		

* Diese Verfahren nutzen neuronale Netze um Deskriptoren aus Bildern zu erzeugen.

3.1. Implementierung

Im folgenden Abschnitt wird die Implementierung der Vorverarbeitung, Gesichtsdetektion und einzelnen Verfahren zur Gesichtserkennung genauer beschrieben. Um Zeit bei der Implementierung zu sparen, wurden dabei nach Möglichkeit vorhandene Software-Bibliotheken und Frameworks verwendet (siehe Kapitel 2.6). Anfangs war vorgesehen größtenteils auf OpenCV zurückzugreifen, da sowohl allgemeine Bildverarbeitungsschritte, als auch Eigenfaces, Fisherfaces und LBP damit abgedeckt werden. Durch einige ungünstige Eigenschaften (siehe 3.1.3) mussten für die Gesichtserkennungsverfahren jedoch andere Frameworks genutzt werden. Die dazu nötigen Schritte und getroffenen Entwicklungsentscheidungen werden im weiteren Verlauf dieses Kapitels näher erklärt. Sämtliche Implementierungen dieser Arbeit wurden mit der Programmiersprache Python vorgenommen. Der Hauptgrund dafür ist, dass Python eine von nahezu allen Frameworks für maschinelles Lernen und Bildverarbeitung unterstützte, wenn nicht sogar geforderte Sprache ist. Sofern möglich, sollen alle Verfahren parallelisierbar sein, also alle vorhandenen **Central Processing Unit (CPU)**-Kerne ausnutzen. Im Fall der neuronalen Netzen soll die Ausführung auf einer GPU möglich sein. Einige der im Folgenden verwendeten Verfahren von Scikit-Learn sind von sich aus parallelisiert. Dies liegt an der internen Nutzung von **Numpy** [87], einem Framework zur effizienten Verarbeitung von Matrizen, da dieses, wenn möglich, immer parallel arbeitet.

Zur Modellierung und zum Training von neuronalen Netzen wurde Keras [86] verwendet, ein Framework welches einfache Schnittstellen für diese Aufgaben bereitstellt. Es kann in Verbindung mit verschiedenen Frameworks genutzt werden, welche die nötigen Algorithmen bereitstellen. Im Fall dieser Arbeit wird dafür TensorFlow [85] genutzt.

3.1.1. Vorverarbeitung

Bevor die Gesichtserkennung stattfinden kann, müssen die Bilder erst im Rahmen der Vorverarbeitung angepasst werden (siehe 2.2.1). Beim Einlesen der Bilder wird bereits entschieden, ob diese als Graustufen- oder RGB-Bilder weiter verarbeitet werden sollen. Das Einlesen kann wahlweise mit OpenCV oder der Python-Bibliothek **Pillow** [88] geschehen. Die optionale Umwandlung in Graustufen-Bilder erfolgt in beiden Fällen mithilfe der Formel zur Luminanzberechnung aus [89]. Die Bilder liegen intern als mehrdimensionale Matrizen von einzelnen Pixeln vor, wobei die ersten beiden Dimensionen immer die Breite und Höhe des Bildes darstellen. Die dritte ist optional und enthält, falls vorhanden, drei Farbkanäle für Rot, Grün und Blau. Während die in dieser Arbeit verwendete Software immer diese Anordnung der Dimensionen verwendet, ist diese nicht allgemeingültig. Andere Frameworks verwenden unterschiedliche Reihenfolgen, nicht

nur für die Dimensionen selbst sondern auch für die Reihenfolge der Farbkanäle. Zur Gesichtsdetektion wurde anfangs das Verfahren von Viola-Jones (siehe Kapitel 2.3.1) verwendet. Dazu wurde die vorhandene Implementierung aus OpenCV genutzt. Das dafür notwendige Kaskadenmodell wird direkt mit dem Framework ausgeliefert. Alternative Modelle können als Parameter übergeben werden. Eine weitere Möglichkeit zur Gesichtserkennung bietet das HOG-Verfahren, welches robuster gegenüber Schwankungen in der Beleuchtung im Bild ist. Eine fertige Implementierung, inklusive eines Modells, bietet Dlib, ein weiteres Framework für maschinelles Lernen und Bildverarbeitung. Bei ersten Vergleichen hat sich gezeigt, dass die Anzahl der korrekt detektierten Gesichter höher ist, als mit der Viola-Jones-Methode. Vor allem werden weniger falsch-positive Ergebnisse erzeugt, also Detektionen von Gesichtern, wenn keine sichtbar sind. Die Gesichtsdetektion liefert dabei für ein Bild die Ergebnisse in Form von quadratischen Bildbereichen, einer sogenannten **Bounding-Box**, in denen sich gefundene Gesichter befinden. Die Bereiche des Detektors von Dlib umfassen dabei nur einen kleinen Bereich in der Mitte eines Gesichts. Daher wurde für diese Arbeit eine Methode implementiert, um die Bounding-Box über einen variablen Parameter zu vergrößern. Für diese Arbeit wurden die Bildbereiche um 80 % vergrößert, um möglichst den gesamten Kopf einer Person zu erfassen. Dadurch fallen die Gesichtserkennungsraten im Vergleich zur normalen Größe für alle Verfahren insgesamt höher aus. Die Auswirkungen von verschiedenen Größenunterschieden können aus Zeitgründen nicht näher untersucht werden. Der Unterschied zwischen den ausgeschnitten Bereichen ist in Abbildung 19 dargestellt.

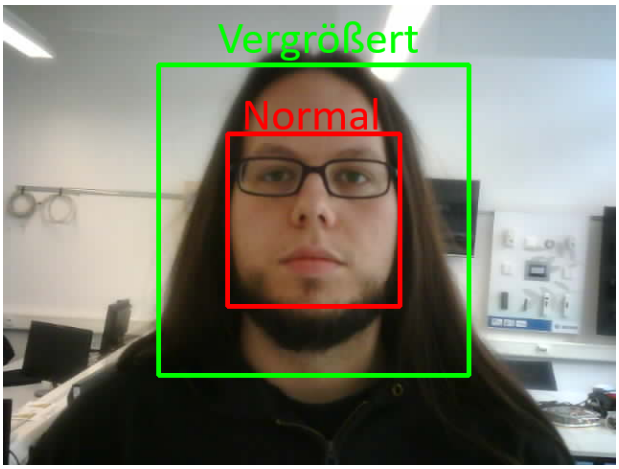


Abbildung 19: Unterschied zwischen der normalen und vergrößerten Bounding-Box von Dlib.

Durch die Nutzung der normalen Bounding-Box würden Teile des oberen und unteren Kopfbereichs abgeschnitten und könnten damit nicht zur Gesichtserkennung genutzt werden. Durch die vergrößerte Bounding-Box wird hingegen nahezu der gesamte Kopf erfasst.

Die gesamte Vorverarbeitung wurde in einer parametrisierten Methode realisiert, welche neben der Gesichtsdetektion weitere Schritte ermöglicht. Dazu zählen eine Helligkeitsanpassung mithilfe der sogenannten Histogram-Equalization, welche von OpenCV bereitgestellt wird. Des Weiteren können die Bilder nach der Detektion automatisch rotiert werden, damit das detektierte Gesicht mittig und aufrecht im Bild sichtbar ist (Alignment). Die Vorverarbeitung kann sowohl mit Graustufen- als auch mit RGB-Bildern durchgeführt werden. Für die Gesichtsdetektion mit OpenCV werden Farbbilder in Graustufen-Bilder umgewandelt. Die Histogram-Equalization wird für Farbbilder komplett übersprungen, da diese sonst wesentlich mehr Aufwand in Anspruch nimmt. Zusätzlich können die Bilder auf eine vorgegebene Größe skaliert werden. Während der folgenden Implementierung der Gesichtserkennungsverfahren hat sich gezeigt, dass alle Vorverarbeitungsschritte, außer der Detektion, die Ergebnisse der einzelnen Verfahren eher verschlechtern als verbessern. Durch umfangreiches Ermitteln der korrekten Parameter für die einzelnen Schritte könnten einzelne Verfahren bessere Ergebnisse bringen. Das ginge aber zulasten der anderen Gesichtserkennungsverfahren. Um eine für alle gleiche Grundlage zu schaffen, wird daher im weiteren Verlauf lediglich die Gesichtsdetektion, Vergrößerung der Bounding-Box sowie Skalierung durchgeführt. Nach der Vorverarbeitung können die Bilder für die eigentliche Gesichtserkennung genutzt werden. Abbildung 20 zeigt die einzelnen Verarbeitungsschritte der Daten, im Rahmen des Vergleichs der Verfahren.

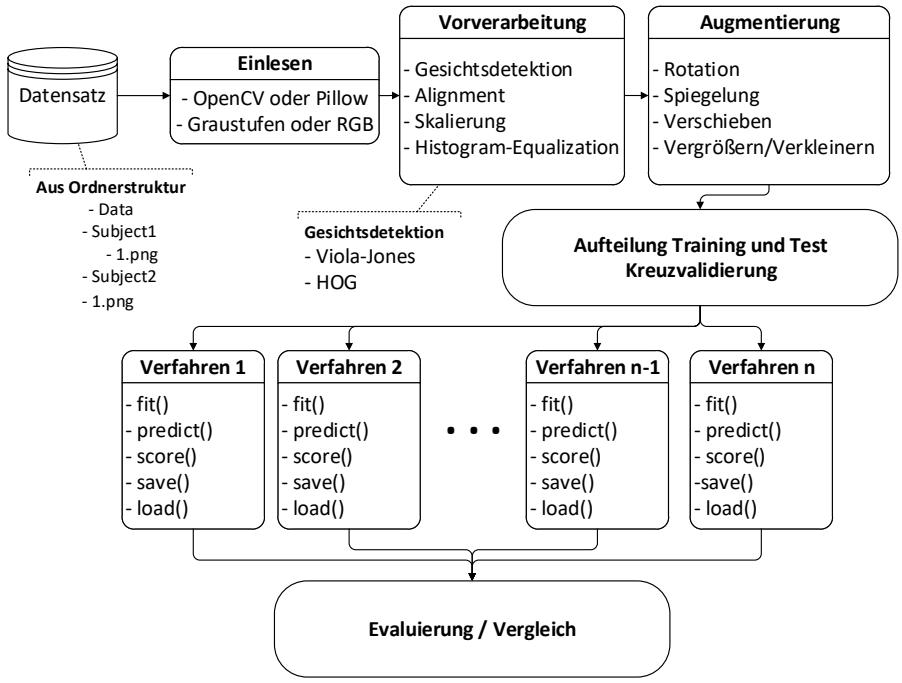


Abbildung 20: Verarbeitungsschritte der Daten im Ablauf der Tests zum Vergleich der Verfahren.

Nach der Vorverarbeitung der Daten erfolgt die Augmentierung, welche im nächsten Kapitel erläutert wird.

3.1.2. Augmentierung

Der in dieser Arbeit durchgeführte Vergleich soll auch unterschiedlich große Datensätze mit einbeziehen. Da nur begrenzte Datenmengen vorhanden sind, werden diese mittels Augmentierung vervielfältigt (siehe Kapitel 2.2.2). Das zur Entwicklung von neuronalen Netzen ausgelegte Framework Keras bietet hierfür bereits Methoden. Die Operationen um Bilder zufällig zu verändern sind über Parameter steuerbar. Für diese Arbeit werden folgende Einstellungen gewählt.

- Horizontale Spiegelung
- Rotation um bis zu 10°
- Verschiebung in der Breite um bis zu 5 %
- Verschiebung in der Höhe um bis zu 5 %
- Vergrößerung/Verkleinerung um bis zu 10 %

In welchem Maße und ob die Operationen angewendet werden, wird zufällig vom Framework während der Ausführung ermittelt. Die für diese Arbeit implementierte Augmentierungsfunktion nimmt zudem einen Augmentierungsfaktor an, der bestimmt wie viele Ausgabebilder für jedes Eingabebild erzeugt werden. Ein Beispiel für die Vervielfältigung ist in Abbildung 21 zu sehen.

3.1.3. Eigenfaces

Das Eigenfaces-Verfahren wird von OpenCV bereits implementiert. Ursprünglich sollte der Einfachheit halber diese Implementierung genutzt werden. Allerdings ist dieser Ansatz nicht flexibel genug, da die Klassifizierung mithilfe einer einfachen euklidischen Distanz durchgeführt wird. Die C++-Schnittstelle von OpenCV bietet zusätzlich zur Klassifizierung auch die Möglichkeit, ein Bild nur in den durch die PCA erzeugten Unterraum zu projizieren. In der für diese Arbeit genutzten Python-Schnittstelle ist diese Funktion jedoch nicht vorhanden, weswegen die Eigenfaces-Implementierung von OpenCV nicht genutzt wird. Alternativ steht mit FaceRec ein weiteres Framework für Gesichtserkennungsalgorithmen zur Verfügung. Dieses verfolgt ebenfalls den Ansatz verschiedene FEs und Klassifizierer zu kombinieren. Einige Verfahren werden dort direkt implementiert, für manche Funktionalitäten werden wiederum weitere Frameworks

Abbildung 21: Beispiel für eine 3-fache Augmentierung.

genutzt (beispielsweise Scikit-Learn), obwohl diese dort bereits umgesetzt sind. Das betrifft unter anderem die PCA. Aufgrund dieser Inkonsistenz wird im weiteren Verlauf auf die Nutzung dieses Frameworks verzichtet. Ein weiterer Grund ist der Verzicht auf zu viele Abhängigkeiten von Projekten die in Zukunft eventuell nicht weiterentwickelt werden. Bei Scikit-Learn ist die zukünftige Entwicklung und Unterstützung durch die Größe und Bedeutung des Projekts eher gesichert als bei FaceRec.

Aus diesen Gründen wurde in dieser Arbeit das Eigenfaces-Verfahren auf Basis der PCA-Implementierung von Scikit-Learn umgesetzt. Zusätzlich zur eigentlichen Extraktion der Merkmale kann der Implementierung ein Klassifizierer übergeben werden, welcher bei Bedarf intern trainiert werden kann. Das soll die spätere Benutzung in einem produktiven Umfeld vereinfachen und ist bei allen folgenden FEs der Fall.

3.1.4. Fisherfaces

Das Fisherface-Verfahren ist ebenfalls in OpenCV und FaceRec implementiert. Es gelten dabei die gleichen Einschränkungen und Probleme wie auch beim Eigenfaces-Verfahren.

Aus diesen Gründen wurde hier ebenfalls eine Implementierung auf Basis der vorhandenen PCA und LDA-Methoden von Scikit-Learn bevorzugt. Zusätzlich bietet die LDA-Implementierung von Scikit-Learn die Möglichkeit, verschiedene Methoden zum erzeugen des Unterraums zu nutzen. Diese werden intern als **Solver** bezeichnet. Einer dieser Solver ist der **Eigen-Solver**, welcher eine Kovarianzmatrix berechnet, um möglichst große Interklassen- und geringe Intraklassen-Abstände zu erzeugen. Da dieses Verfahren jedoch sehr hohe Speicheranforderungen hat, wird im Vorfeld eine PCA durchgeführt, um die Anzahl der Merkmale zu reduzieren. Ab einer gewissen Datenmenge können die Speicheranforderungen ein sinnvolles Maß überschreiten, weswegen alternativ ein anderer Solver, die **Singular Value Decomposition (SVD)**, genutzt werden kann. Diese berechnet keine Kovarianzmatrix und kann auch ohne vorhergehende PCA durchgeführt werden. Ab einer empirisch bestimmten Datenmenge wird automatisch auf diese Variante gewechselt, um schwerwiegende und im schlimmsten Fall nicht auffangbare Fehler durch mangelnden Arbeitsspeicher zu vermeiden. Die Schwelle wird auf dem für die Testreihe verwendeten System bei 4.500 Bildern mit 96x96 Pixeln erreicht. Eine dynamische Ermittlung dieser Schwelle, abhängig vom verfügbaren Arbeitsspeicher, ist im Zeitrahmen dieser Arbeit nicht möglich. Für Details zur Funktionsweise der SVD wird auf [90] verwiesen. Eine Untersuchung von [91] zeigt, dass eine reine SVD für den Anwendungsfall der Gesichtserkennung unzulänglich ist. Ob dies für diese Arbeit relevant ist, wird in der Auswertung in Kapitel 3.6 näher betrachtet.

3.1.5. Local Binary Pattern

Auch für die Gesichtserkennung auf Basis von LBPs gibt es Implementierungen in OpenCV und FaceRec. Wie bei den beiden vorhergehenden Verfahren wird auf die Verwendung dieser Frameworks diesbezüglich verzichtet. Da die Berechnung von LBPs kein typischer Arbeitsschritt des maschinellen Lernens ist, gibt es keine Implementierung in Scikit-Learn. Ein weiteres quelloffenes Framework, welches das Verfahren zur Gesichtserkennung nach [49] implementiert, wurde während der Erstellung dieser Arbeit nicht gefunden. Aus diesem Grund wird das Verfahren für diese Arbeit selbst implementiert. Zur Erzeugung der LBPs kommt das Framework **Mahotas** [92] zum Einsatz. Dieses stellt eine Methode bereit, um aus einem Bild ein LBP-Histogramm zu erzeugen. Um das Verfahren von [49] zu erfüllen, wird das zu verarbeitende Bild vorher in mehrere Teilbilder aufgetrennt. Die Form des Rasters kann dabei frei eingestellt werden. Wie in [49] angegeben, wird für die Testreihe ein Aufteilung in 7x7 Teilbilder verwendet. Die Transformation der Bilder in LBP-Histogramme läuft parallel ab, wobei die Anzahl der Prozesse als Parameter übergeben werden kann.

3.1.6. Dlib-Deskriptor

Das Framework Dlib bietet neben der Gesichtsdetektion auch ein Verfahren um aus einem Gesicht einen Zahlenvektor zu extrahieren, welcher für Vergleiche genutzt werden kann. Die Basis dafür ist ein CNN, dessen Topologie auf dem von [93] entwickelten **ResNet-34** basiert. Im Gegensatz zur ursprünglichen Variante wird das CNN nicht zur direkten Klassifizierung genutzt. Stattdessen erzeugt das CNN aus einem Gesicht einen Vektor von 128 Zahlen. Ähnlich wie zum Beispiel bei Eigenfaces findet eine Dimensionsreduktion in einen Unterraum statt. Diese Vektoren, auch Deskriptoren oder **Embeddings** genannt, haben die Eigenschaft, dass Bilder der gleichen Person in einem euklidischen Raum möglichst nah beieinander liegen. Bilder von verschiedenen Personen hingegen besitzen einen größeren Abstand. Das von Dlib gelieferte neuronale Netz ist darauf trainiert, dass Deskriptoren von Bildern der gleichen Person einen euklidischen Abstand von weniger als 0,6 aufweisen. Einen ähnlichen Ansatz verfolgt auch das in Kapitel 2.5.2 erwähnte Projekt DeepFace [77]. Bei der Implementierung ist zu beachten, dass die Eingangsdaten für das CNN nicht die Bilder sind, sondern vorher aus diesen extrahierte Landmarks. Dlib bietet auch dafür eine Methode, um effizient 68 Gesichtsmerkmale zu extrahieren, basierend auf der Arbeit von [94].

3.1.7. Openface-Deskriptor

Openface ist ein Projekt, welches ebenfalls mittels eines neuronalen Netzes Deskriptoren von Gesichtern erzeugt. Die Software stellt eine Python-Schnittstelle bereit, sowie die trainierten Modelle. Intern baut das Projekt auf Torch auf, einem in Lua geschriebenen Framework zur effizienten Verarbeitung von neuronalen Netzen auf Grafikkarten. Aus diesem Grund besteht die Python-Schnittstelle lediglich aus dem Aufruf eines Lua-Skripts. Dieses nimmt eine einzelne Bilddatei entgegen und erzeugt daraus den Deskriptor, welcher aus 128 Zahlen besteht. Das hat zur Folge das jedes zu verarbeitende Bild, obwohl es bereits im Speicher liegt, nochmal auf Festplattenspeicher geschrieben wird. Des Weiteren nehmen die Modelle ausschließlich Bilder im Format von 96x96 Pixeln entgegen. In Abbildung 22 ist ein schematischer Überblick über das von OpenFace bereitgestellte System dargestellt.

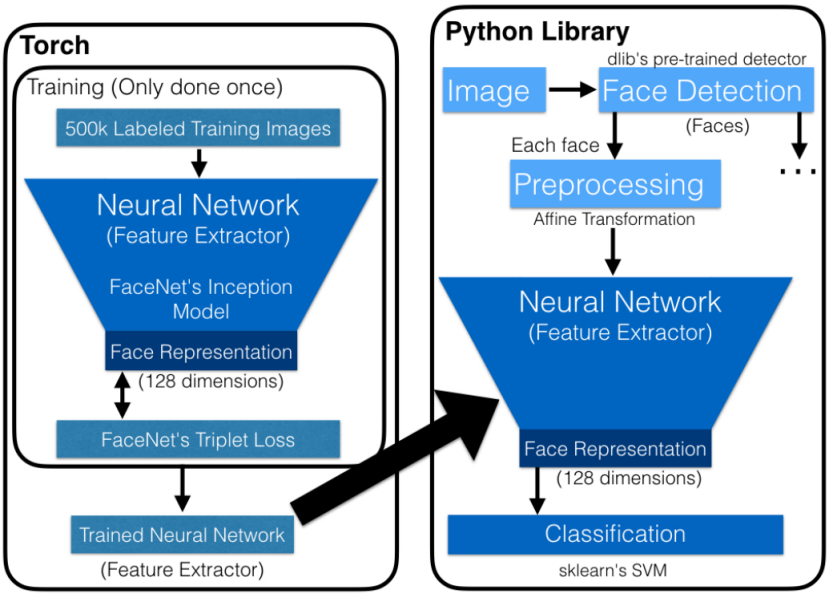


Abbildung 22: Schematischer Überblick der Struktur von OpenFace [76].

Der für diese Arbeit genutzte Teil beginnt mit dem neuronalen Netz auf der rechten Seite. Dort werden Merkmale aus den Bildern extrahiert und der Deskriptor erzeugt. Dieser wird im Anschluss zur Klassifizierung genutzt.

3.1.8. VGG19-Deskriptor

Auf der Grundlage eines Artikels von Francois Chollet auf dem Blog von Keras [95], soll noch ein weiterer Ansatz für Deskriptoren auf Basis von CNNs verwendet werden. Keras stellt mehrere bereits trainierte Modelle von verschiedenen Topologien zur Verfügung. Als Trainingsdatensatz dient dabei immer ImageNet. Chollet beschreibt, wie die Ausgabe der letzten Konvolutionsschicht als Eingang für ein einfaches, voll vernetztes, neuronales Netz genutzt werden kann. Diese durch die letzte Konvolutionsschicht erzeugten Daten werden im Artikel als **Bottleneck-Features** bezeichnet. Ein kleines (ein bis zwei Schichten tief), voll vernetztes neuronales Netz kann diese Bottleneck-Features nutzen, um andere Klassifizierungsaufgaben durchzuführen, als ursprünglich vorgesehen[18]. Die Idee dabei ist, dass das bereits trainierte CNN genügend Informationen und Abstraktionsfähigkeit besitzt, um auch für andere Aufgaben der Bilderkennung sinnvolle Ergebnisse zu liefern. Für diesen Zweck ermöglicht Keras es, ein Modell auch ohne die letzten Schichten zu initialisieren, die üblicherweise die Klassifizierung durchführen.

[18] Im Fall dieser Arbeit die Klassifizierung der Gesichter von verschiedenen Personen, anstatt Bilder aus den 1.000 Kategorien aus dem ImageNet-Wettbewerb.

Zusammengefasst dient das CNN als FE, dessen erzeugte Merkmale wiederum zur Klassifizierung genutzt werden können. Daher beschränkt sich diese Arbeit nicht nur darauf, Bottleneck-Features durch ein einfaches neuronales Netz zu klassifizieren. Stattdessen ist die Kombination mit beliebigen Klassifizierern möglich. Das von Keras bereitgestellte Modell des VGG19-Netzes [96] liefert im Vergleich zu den anderen verfügbaren Modellen die besten Ergebnisse, weswegen in dieser Arbeit für die Testreihe dieses Modell genutzt wird. Die Topologie ist in Abbildung 23 dargestellt. Die Bottleneck-Features ergeben sich als Ausgabe der letzten **Maxpool**-Schicht. Zu beachten ist, dass die Features zur weiteren Verarbeitung aus einer mehrdimensionalen Matrix in einen eindimensionalen Feature-Vektor umgewandelt werden müssen.

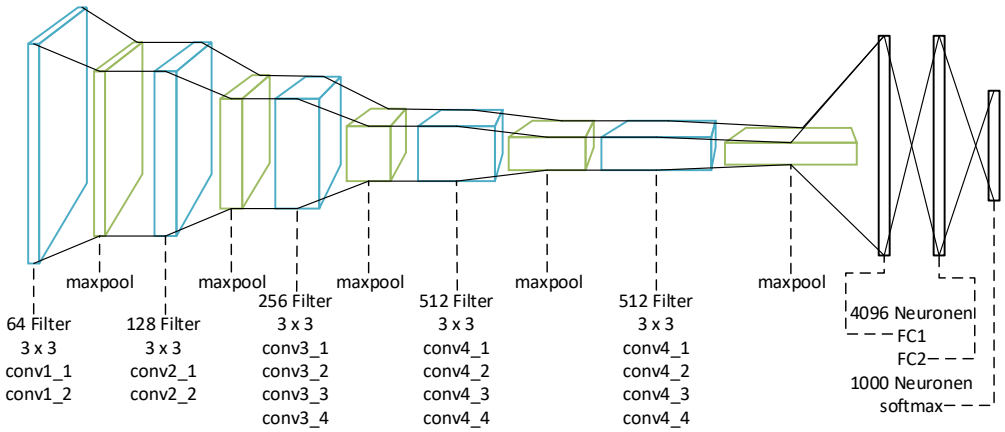


Abbildung 23: Schematische Darstellung der Topologie des von Keras bereitgestellten VGG19-Netzes.

3.1.9. k-Nearest-Neighbors

Alle beschriebene FEs haben das Ziel Deskriptoren, oder auch Merkmale, aus Gesichtern zu erzeugen. Neben der Dimensionsreduktionen und damit vereinfachten weiteren Verarbeitung, haben diese die Eigenschaft, bestimmte Abstände in den jeweiligen Unterräumen zueinander zu haben. Um eine Klassifizierung vorzunehmen, wird ein Bild mit einem Gesicht in diesen Unterraum projiziert, oder einfach gesagt, es werden Merkmale extrahiert. Mithilfe des kNN-Verfahrens wird ermittelt, welche k bekannten Feature-Vektoren dem Vektor des zu identifizierenden Gesichts am nächsten sind. Der am häufigsten auftretende Nachbar gilt als Klassifizierungsergebnis. In Scikit-Learn ist ein Klassifizierer auf Basis von kNN bereits implementiert und kann direkt genutzt werden. In der Standardkonfiguration beträgt $k = 5$, es werden also die fünf nächsten Nachbarn in Betracht gezogen. Als Distanzmaß wird die euklidische Distanz genutzt. Zusätzlich zum Klassifizierungsergebnis können auch Wahrscheinlichkeiten zurückgegeben werden. Dabei wird zu jedem möglichen Ergebnis (allen bekannten Klassen oder

Personen), eine Wahrscheinlichkeit ermittelt, die sich aus der Anzahl entsprechenden Nachbarn ergibt, die in der nähe des zu klassifizierenden Bildes sind. Für eine noch genauere Kontrolle über das Ergebnis, können auch die konkreten Distanzen zu einer bestimmten Anzahl an Nachbarn ermittelt werden. Während für den Vergleich des Verfahrens nicht auf diese Möglichkeit zurückgegriffen wird, ist diese für die Implementierung des Prototyps relevant, sofern das kNN-Verfahren ausgewählt wird.

3.1.10. Neuronales Netz

Um ein neuronales Netz zur Klassifizierung zu nutzen, wird eine einfache Topologie mit zwei versteckten Schichten gewählt. Die Anzahl der Eingangsneuronen wird dynamisch anhand der Größe der vom genutzten FE berechneten Merkmale ermittelt. Die Anzahl der Ausgangsneuronen ist identisch mit der Menge der zu klassifizierenden Klassen, beziehungsweise Personen in diesem Fall. Das Netz besteht aus zwei voll vernetzten versteckten Schichten, sowie einer Ausgangsschicht. Nach den beiden versteckten Schichten liegt jeweils eine **Dropout**-Schicht [97]. Diese wählen zufällig eine bestimmte Anzahl an Ausgangssignalen der vorherigen Schicht und setzen diese auf die Zahl Null. Durch dieses simple, aber sehr effiziente Regularisierungsverfahren wird eine Überanpassung an die Daten verhindert [98]. Die erste Dropout-Schicht verwirft 50 % aller Neuronen, die Zweite 20 %. Anfänglich wurden wie in der Arbeit von [97] empfohlen 50 % verwendet, dabei konnte das Netz während des Trainings die Genauigkeit nicht verbessern. Die Dropout-Rate wurde in 10 %-Schritten verringert, bis die Genauigkeit während des Trainings anfing zu steigen. Ein anschließender Test mit nur 10 % oder ohne Dropout bewirkte, dass das Netzwerk sich an die Daten überanpasste.

Als Aktivierungsfunktion werden für die versteckten Schichten ReLUs genutzt. Die Ausgabeschicht verwendet eine **Softmax**-Aktivierung, wodurch die Summe der Aktivierungen der Ausgabeneuronen $1,0$ ergibt [56]. Die Werte der einzelnen Neuronen spiegeln somit Wahrscheinlichkeiten für die jeweiligen Klassen wieder. Abbildung 24 zeigt die Topologie des neuronalen Netzes.

3.1.11. Support Vector Machines

Die Möglichkeit SVMs zur Klassifizierung zu benutzen wurde bereits in Kapitel 2.4.2 beschrieben. Eine effiziente Implementierung ist in Scikit-Learn vorhanden. Um eine Klassifizierung für mehrere Klassen zu ermöglichen, wird der One-Against-One-Ansatz

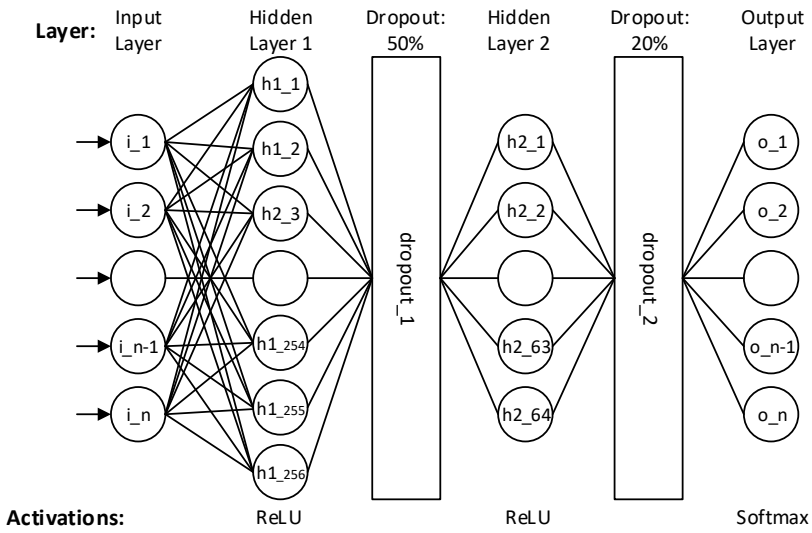


Abbildung 24: Die Topologie des neuronalen Netzes, dass als Klassifizierer genutzt wird.

verwendet. Um gute Ergebnisse mit SVMs zu ermöglichen, ist es häufig nötig mehrere Parameter anzupassen. Üblicherweise wird dafür eine Rastersuche, oder auch **Gridsearch** in Scikit-Learn, durchgeführt. Es werden sämtliche anzupassende Parameter in mehreren gewünschten Stufen in allen Kombinationen auf den zu klassifizierenden Daten durchprobiert. Die Parameter mit dem besten Ergebnis werden dann im weiteren Verlauf verwendet. Da der Vorgang sehr Zeitaufwändig ist, wird er im Vorfeld für jeden FE einzeln durchgeführt. Beim Erzeugen einer SVM kann somit jeder FE die für das jeweilige Verfahren besten Parameter bereitstellen. Sämtliche gewählten Parameter sind in Tabelle 2 aufgelistet[19].

Tabelle 2: Durch Rastersuche ermittelte SVM-Parameter

Feature-Extractor	**Kernel**	**C**	**Gamma**
Eigenfaces	linear	1×10^{-4}	n.a.
Fisherfaces	linear	1×10^{-4}	n.a.
Local Binary Pattern	linear	1×10^{1}	n.a.
Dlib-Deskriptor	rbf	1×10^{1}	1
Openface-Deskriptor	rbf	1×10^{2}	1
VGG19-Deskriptor	linear	1×10^{-3}	n.a.

Dabei ist zu beachten, dass je nach Ausprägung des Datensatzes andere Parameter bessere Ergebnisse liefern können. Daher sollte bei großen Änderungen der zugrunde liegenden Daten eine erneute Rastersuche durchgeführt werden.

[19]Für eine genaue Bedeutung der einzelnen Parameter sei auf die Dokumentation von Scikit-Learn unter `http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html` verwiesen.

Neben der Zuordnung eines Bildes zu einer Klasse, können zudem auch Wahrscheinlichkeiten ermittelt werden. Diese geben für jede bekannte Klasse an, wie wahrscheinlich es ist, dass das übergebene Bild zu dieser Klasse gehört. Die Wahrscheinlichkeiten ergeben in der Summe immer $1,0$, ähnlich der bei neuronalen Netzen (siehe Kapitel 3.1.10) verwendeten Softmax-Funktion. Das SVM-Verfahren kann daher nicht von sich aus Personen als unbekannt einstufen.

3.1.12. Einfaches Convolutional Neural Network

In den vorherigen Abschnitten sind mehrere FEs und Klassifizierer aufgelistet, welche beliebig kombiniert werden können. Drei der FEs basieren auf neuronale Netze, im speziellen auf CNNs. Wie bereits in Kapitel 2.5.1 beschrieben, sind diese Art von neuronalen Netzen darauf spezialisiert Bilder zu verarbeiten. Ein CNN mit einer oder mehreren voll vernetzten Schichten zum Schluss ist zudem in der Lage Klassifizierungen durchzuführen. Damit erledigt solch ein Modell beide Schritte, Merkmalsextraktion und Klassifizierung, auf einmal und zählt zu den AIO-Verfahren. Um diesen Ansatz umzusetzen, wird mittels Keras eine einfache Topologie modelliert. Die genauen Parameter werden dabei durch experimentelles Anpassen ermittelt. In Abbildung 25 ist das Modell dargestellt.

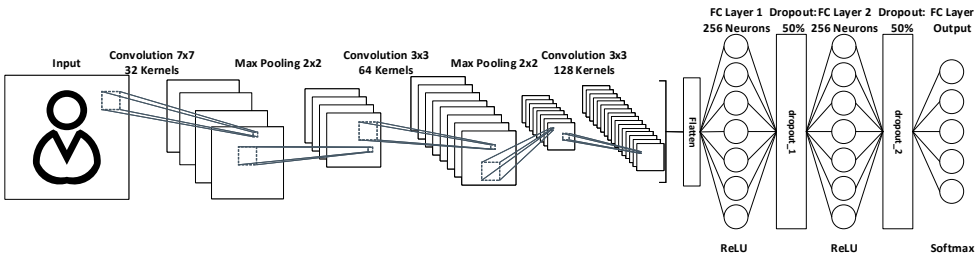


Abbildung 25: Die Topologie des einfachen CNN.

Zu sehen sind die Konvolutionsschichten, welche visuelle Informationen aus dem Bild extrahieren. Nach der dritten Konvolutionsschicht werden die Ausgaben sämtlicher Feature-Maps konkateniert und in zwei versteckten Schichten weiterverarbeitet. Die Ausgabeschicht stellt das Ergebnis der Klassifizierung dar. Eines der Ziele ist es, das neuronale Netz und damit die Anzahl der zu trainierenden Parameter möglichst klein zu halten, wodurch sich auch der Name einfaches CNN ergibt. Um dennoch eine Überanpassung an die Daten zu vermeiden, wird neben Dropout auch L2-Regularisierung genutzt. Dadurch werden kleine Gewichtungen im Netzwerk bevorzugt. Für die genauen Funktionsweisen und Hintergründe bezüglich Regularisierungs-Methoden wird aus Platzgründen nochmal auf [56] verwiesen.

Als Optimierungsverfahren wurde **AdaDelta** [99] ausgewählt, da das Netzwerk dadurch am schnellsten, wenn auch nicht immer, konvergierte. Während der Implementierungsphase ist es möglich, experimentell eine sinnvolle Anzahl an Epochen zu finden. Während der automatisierten Testreihe muss automatisch ermittelt werden, wann die Trainingsphase beendet wird. Dadurch wird zum einen eine Überanpassung an die Trainingsdaten verhindert, zum anderen wird Zeit gespart. Dieses Vorgehen wird **Early Stopping** genannt. Wenn sich der Fehler des Netzwerks in Form der Kreuzentropie nach sechs Epochen nicht um 0,01 Punkte verbessert hat, wird das Training unterbrochen. Die Kreuzentropie wird dabei auf den Testdaten ermittelt. Die zum Schluss verwendeten Gewichte des Modells stammen aus der Epoche mit der besten Genauigkeit auf den Testdaten.

3.1.13. Abgestimmtes Convolutional Neural Network

Als letztes Verfahren wird ein vortrainiertes CNN verwendet, um, ähnlich wie beim VGG19-Deskriptor, ein voll vernetztes neuronales Netz zu trainieren. Dabei werden zuerst mit dem CNN, auch Base-Model genannt, Bottleneck-Features aus den Trainingsdaten erstellt. Im Anschluss wird das voll vernetzte Netz, oder auch Top-Model, mit diesen Features trainiert. Im nächsten Schritt wird dieses Netz auf das erste aufgesetzt. Im Anschluss wird eine weitere Trainingsphase mit diesem zusammengesetzten Netz durchgeführt. Um zu verhindern, dass dabei bereits im Netzwerk gespeicherte Informationen verloren gehen, werden alle Schichten bis zum letzten Block von Konvolutionen für Gewichtsänderungen gesperrt. Lediglich das voll vernetzte Netz und die letzten Konvolutionsschichten werden angepasst. Einen Überblick über das zusammengesetzte Netz und die gesperrten Schichten ist in Abbildung 26 dargestellt.

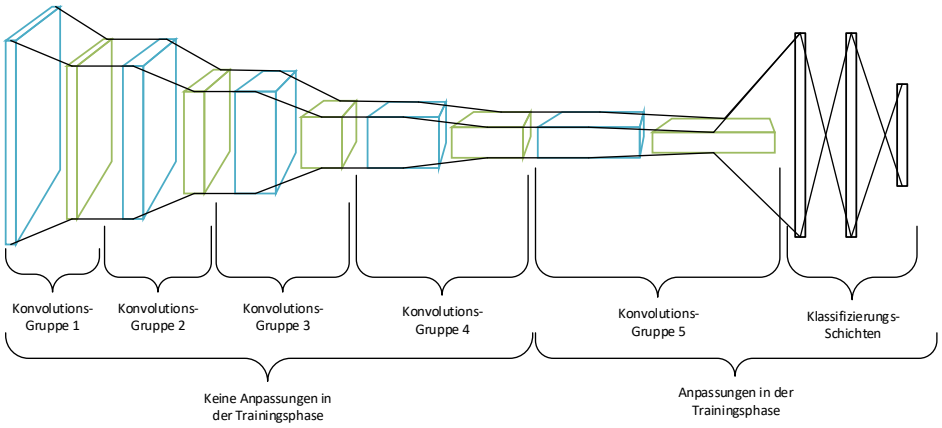


Abbildung 26: Die Topologie des abgestimmten CNN.

Das Base-Model in Form der Konvolutionsgruppen stammt aus dem VGG19-Netz, das Top-Model entspricht den Klassifizierungsschichten. Ziel ist es, das neu gebildete Netzwerk auf das neue Klassifikationsproblem abzustimmen, und dabei die bereits vorhandenen Informationen weiterhin zu nutzen. Durch diese Feinabstimmung ergibt sich der Name des Verfahrens. Die Gewichte im Netz dürfen nur in sehr geringem Maße angepasst werden. Aus diesem Grund wird als Optimierungsverfahren das **Stochastic Gradient Descent (SGD)**-Verfahren mit einer sehr geringen Lernrate von 1×10^{-4} verwendet (siehe Kapitel 2.5). Dieses gesamte Vorgehen wird auch als **Fine Tuning**, beziehungsweise **Transfer Learning** bezeichnet [100, 101]. In experimentellen Versuchen im Rahmen dieser Arbeit hat sich gezeigt, dass mit der richtigen Auswahl an Bottleneck-Features das Top-Model bereits 97 % Genauigkeit erreichen kann. Durch die anschließende Feinabstimmung konnten zum Teil 100 % Genauigkeit auf Testdaten erreicht werden. Während der Vorbereitungsphase für die in Kapitel 3.5 durchzuführenden Tests zeigte sich jedoch, dass in einer automatisierten Testumgebung diese Ergebnisse nicht immer reproduzierbar sind. Zum Teil erreicht das Top-Model in der ersten Trainingsphase schon höhere Genauigkeiten als gemeinsam mit dem Base-Model in der zweiten Trainingsphase. Aus diesem Grund wurden Überprüfungen eingebaut, die in einem solchen Fall ein zweites Training verhindern. Zudem werden die Trainingsphasen, wie auch beim oben beschriebenen einfachen CNN, nach einer bestimmten Anzahl an Epochen ohne Verbesserung unterbrochen, sowie die jeweils besten Gewichte gespeichert und behalten. Sämtliche Modelle und Abläufe sind auf Basis von Keras implementiert.

3.2. Ausgewählte Datensätze

Um zu prüfen, ob und wie sich unterschiedliche Ausprägungen von Datensätzen auf die Leistung der Verfahren auswirken, sollen zwei verschiedene ausgewählt werden. Dabei ist zu beachten, dass sich Eigenschaften wie Auflösung, Bildqualität und Beleuchtungsumgebungen unterscheiden. Weitere Unterschiede können auch in der Gesamtzahl an Bildern, sowie der Anzahl von einzelnen Bildern pro Person liegen.

3.2.1. H-BRS Haut-, Gesichts- und Fälschungsdatenbank

Der BRSU-Datensatz ist bereits in Kapitel 2.2 näher beschrieben. Zusammenfassend ist die Qualität der Bilder sehr hoch, die Gesichter sind in den Bildern bereits gut zentriert und sehr gleichmäßig beleuchtet. Im Datensatz gibt es Aufnahmen von 53 Personen mit jeweils 5-6 Bildern pro Person bei einer Auflösung von 426x640 Pixeln. Es liegen auch Aufnahmen im Nahinfrarotbereich vor, diese sind jedoch für diese Arbeit nicht relevant

und werden daher nicht genutzt. Es ist jedoch gut vorstellbar, diese Aufnahmen, zusätzlich zu den normalen Bildern, zu nutzen, um ein Netzwerk nicht nur Personen erkennen zu lassen, sondern zugleich Fälschungen in Form von Masken oder ausgedruckten Bildern zu entdecken.

3.2.2. Labordatensatz

In einem realen Anwendungsfall ist es unwahrscheinlich, dass Personen perfekt frontal vor einer Kamera stehen. Auch die Beleuchtung kann sich unterscheiden. Aus diesem Grund sollte für diese Arbeit ein Datensatz erzeugt werden, welcher realen Bedingungen entspricht. Grundlage dafür ist ein digitaler Türspion, welcher von Hendrik Linka im Rahmen einer Bachelorarbeit entwickelt wurde [102]. Dieser wurde seitdem weiterentwickelt und besteht aus einer Arduino-Cam und einem WLAN-Chip. Die Auflösung beträgt 640x480 Pixel, wobei die Qualität schlechter ist als bei den Aufnahmen aus dem BRSU-Datensatz. Es ist teilweise deutliches Bildrauschen erkennbar, sowie Artefakte aufgrund der Kompression des **Joint Photographic Experts Group (JPG)**-Formats.

Auf Knopfdruck werden innerhalb weniger Sekunden mehrere Bilder aufgenommen und an einen Server übertragen, welcher diese abspeichert. Der Aufbau wurde im Labor C060 der Hochschule Bonn-Rhein-Sieg angebracht. Dort herrschen Beleuchtungsbedingungen aus einem alltäglichem Umfeld mit wechselndem natürlichen und künstlichen Lichtquellen. Zu unterschiedlichen Zeiten wurden verschiedene Personen gebeten Aufnahmen von sich erstellen zu lassen. Aus diesen Bildern werden für den Datensatz die Aufnahmen ausgewählt, welche frontale Ansichten auf Gesichter darstellen. Zum Zeitpunkt der Erfassung der Testergebnisse umfasst dieser sogenannte Labordatensatz 189 Bilder von 14 Personen. Dabei schwankt die Anzahl an Aufnahmen pro Person zwischen 5 und 31, um ein realistisches Szenario nachzubilden bei dem manche Personen häufiger einen Wohnort einer anderen Person aufsuchen.

3.3. Beschreibung der Testreihe

Im folgenden Abschnitt wird das Konzept beschrieben, mit dem die zum Vergleich der Gesichtserkennungsverfahren nötigen Daten erhoben werden. Aus Gründen der Reproduzierbarkeit und Einfachheit erfolgt dieser Ablauf möglichst automatisiert. Aus den sechs Verfahren zur Merkmalsextraktion sowie drei zur Klassifizierung ergeben sich 18 Kombinationsmöglichkeiten. Zusammen mit den beiden AIO-Verfahren gibt es 20 Möglichkeiten eine Gesichtserkennung durchzuführen. Für jedes Verfahren stehen unterschiedliche Parameter zur Verfügung. Die Untersuchung der Auswirkungen und die

entsprechenden Vergleiche würden jedoch über den Rahmen dieser Arbeit hinausgehen. Daher werden, sofern im Abschnitt 3.1 nicht anders beschrieben, Parameterwerte gewählt, welche durch die genutzten Frameworks oder die zugrunde liegenden wissenschaftlichen Arbeiten vorgegeben sind. Für das oder die Verfahren, welche die besten Ergebnisse liefern, können im Anschluss die einzelnen Parameter betrachtet und optimiert werden, sofern dadurch Verbesserungen möglich sind.

Jede der Kombinationsmöglichkeiten wird mit den beiden Datensätzen, sowie unterschiedlich stark augmentierten Versionen davon, trainiert und getestet. Um die Genauigkeit eines Verfahrens, beziehungsweise Modells, zu ermitteln, werden die Daten in einen Trainings- und einen Testdatensatz aufgeteilt. Der Trainingsdatensatz wird während der Trainingsphase zum Lernen genutzt. Im Anschluss werden die Bilder aus dem Testdatensatz durch das trainierte Modell klassifiziert. Die Anzahl der korrekt klassifizierten Bilder ergibt die Genauigkeit, als Wert zwischen 0,0 und 1,0. Ein Typischer Anteil an Trainingsdaten ist 80 % [67, S. 118]. Problematisch ist dieser Ansatz bei relativ kleinen Datensätzen. Mit dem im Labor erhobenen Datensatz würde der Anteil an Testdaten 38 der insgesamt 189 Bilder betragen. Zudem ist die Aufteilung zufällig, wodurch manche Bilder aus einer Klasse gar nicht im Testdatensatz vorhanden sein können. Dadurch wären die Testdaten nicht repräsentativ [103, S. 152]. Der umgekehrte Fall ist ebenso möglich. Um diese und weitere Probleme zu umgehen, werden die Testergebnisse mittels der Kreuzvalidierung (siehe Kapitel 2.1.2) erhoben. Dabei werden die Daten in k Blöcke aufgeteilt, wobei jeder einzelne Block einmal als Testdatensatz zählt, die jeweils restlichen als Trainingsdatensatz. In der Literatur wird für k häufig fünf oder zehn vorgeschlagen [4, S. 243]. Eine hohe Anzahl an Blöcken bringt den Nachteil mit sich, dass für jeden Block ein Modell berechnet werden muss. Dadurch steigt insgesamt die Zeit für die Auswertung stark an. Deshalb wird für diese Arbeit $k = 5$ gewählt. Um sicher zu stellen, dass die Verteilung der einzelnen Klassen innerhalb der Blöcke der Verteilung der Klassen im gesamten Datensatz entspricht, wird die stratifizierte Kreuzvalidierung verwendet [104]. Eine Implementierung dieses Verfahrens wird von Scikit-Learn bereitgestellt. Für die spätere Auswertung werden für jeden Durchlauf der Kreuzvalidierung folgende Daten erfasst:

- Name des FE
- Name des Klassifizierers
- Name des Datensatzes
- Anzahl der Bilder im Datensatz
- Augmentierungsfaktor
- Nummer des Durchlaufs in der Kreuzvalidierung
- Trainingsdauer des FE in Millisekunden

- Trainingsdauer des Klassifizierers in Millisekunden
- Trainingsdauer insgesamt in Millisekunden
- Verarbeitungsdauer des FE für ein Bild in Millisekunden
- Verarbeitungsdauer des FE für zehn Bilder in Millisekunden
- Verarbeitungsdauer des Klassifizierers für ein Bild in Millisekunden
- Verarbeitungsdauer des Klassifizierers für zehn Bilder in Millisekunden
- Genauigkeit der Klassifizierung
- Verwendung des SVD-Solver für Fisherfaces
- Verwendung der GPU

Da für die AIO-Verfahren eine Unterteilung in FE und Klassifizierer nicht möglich ist, wird in allen entsprechenden Feldern der gleiche Wert eingetragen. Um für eine Verfahrenskombination auf einem Datensatz mit einem bestimmten Augmentierungsfaktor die Genauigkeit zu bestimmen, wird der Mittelwert der entsprechenden Kreuzvalidierungsergebnisse gebildet.

3.4. Vorbereitungen und Annahmen für die Testreihe

Die Erhebung der Testergebnisse erfolgt mithilfe eines in Python geschriebenen Skripts. In diesem Skript werden sämtliche Kombinationen von Verfahren, Datensätzen und Augmentierungsfaktoren gebildet und getestet. Damit die Ergebnisse reproduzierbar sind, dürfen zwischen den einzelnen Durchläufen keine Zufallsfaktoren den Ablauf beeinflussen. Das hat zur Folge, dass zum einen die Augmentierung nicht für jedes Verfahren erneut durchgeführt werden kann, da die einzelnen Operationen vom Zufall beeinflusst sind. Um Speicherplatz zu sparen, können auch nicht sämtliche Kombinationen von Datensätzen und Augmentierungsfaktoren auf einmal berechnet und durchlaufen werden. Stattdessen wurde ein Skript geschrieben, welches die Datensätze vorher auf verschiedenen Faktorstufen augmentiert und die eventuellen Vorverarbeitungsschritte durchführt. Im Anschluss werden die so erzeugten Datensätze als separate Dateien abgespeichert.

Die Augmentierungsfaktoren betragen 5, 10, 15, 20, 25 und 30. Mit den beiden Basisdatensätzen ergeben sich so insgesamt 14 Datensätze, mit Datenmengen zwischen wenigen hundert und mehreren tausend Bildern. Der Name dieser Dateien besteht aus diversen Angaben bezüglich des Inhalts. Im Folgenden ein Beispiel: `Labor_AUGMENTED_1_-RGB_96_X.npy`. Zuerst steht Name des Datensatzes. Danach folgt die Angabe ob es sich um den Basisdatensatz oder um eine augmentierte Version handelt. Ist das der Fall, steht danach der Augmentierungsfaktor. Im Anschluss folgt die Angabe des Farbmodus, RGB für Farbbilder und GRAY für Graustufen. Danach folgt die Seitenlänge der Bilder

in Pixel. Am Ende steht ein `X` für die eigentlichen Bilddaten, oder ein `y` für die dazugehörigen Personen. Die Dateiendung steht für in Binärform abgespeicherte NumPy-Arrays, da sämtliche Daten intern in dieser Form verarbeitet werden. Es ist anzumerken, dass die Kennzeichnung der jeweiligen Person bei der internen Verarbeitung durch eine Zahl ersetzt wird, die der Reihenfolge beim einlesen der Bilder entspricht. Dadurch wird die weitere Nutzung für die einzelnen Verfahren einfacher. Um später bei Bedarf dennoch auf die ursprünglichen Label zurückzugreifen, wird bei der eben beschriebenen Vorbereitung ein **Dictionary** erzeugt, eine Datenstruktur von Python die einer Hashtabelle entspricht. In dieser wird jeder Nummer das entsprechende Label zugeordnet. Dieses Dictionary wird serialisiert und im gleichen Ordner wie die vorbereiteten Datensätze abgespeichert.

Zusätzlich zu der Augmentierung wird in dem Vorbereitungsskript auch die Gesichtsdetektion auf Basis des HOG-Verfahrens von Dlib durchgeführt. Dadurch wird die mehrfache Durchführung im eigentlichen Testskript verhindert. Dabei werden die Bilder, beziehungsweise die Ausschnitte mit den Gesichtern, auf eine bestimmte Größe skaliert. Dadurch arbeiten alle Verfahren mit den exakt gleichen Bildern. Der limitierende Faktor sind hierbei die neuronalen Netze. Bei größer werdenden Bildern steigt die Anzahl der nötigen Eingangsknoten quadratisch an, was sich negativ auf die Verarbeitungsgeschwindigkeit und den Speicherverbrauch auswirkt. Daher sind geringe Bildgrößen von unter 100x100 Pixel üblich. Während die meisten hier genutzten neuronale Netze größere Bilder verwenden könnten, abhängig von der verfügbaren Menge an Grafikkarten- beziehungsweise Arbeitsspeicher, gibt das vom Openface-Deskriptor genutzte neuronale Netz eine feste Größe von 96x96 Pixel vor. Wie in [21] beschrieben, ist diese Größe mehr als ausreichend zur Gesichtserkennung[20], weswegen sich dadurch keine praktischen Auswirkungen ergeben. Daher wurde als Seitenlänge für sämtliche Bilder 96 Pixel gewählt.

Für die in dieser Arbeit durchgeführte Testreihe werden RGB-Bilder verwendet. Allerdings können nicht alle Verfahren diese auch nutzen. Lediglich die FEs, welche Deskriptoren von Bildern durch neuronale Netze erzeugen, können RGB-Bilder verarbeiten. Eigenfaces, Fisherfaces und LBP arbeiten mit nur einem Farbkanal, den Graustufen. Bei den AIO-Verfahren können prinzipiell beide mit Farbbildern arbeiten, jedoch ergibt sich durch die Nutzung von drei Farbkanälen die dreifache Menge an Eingangsneuronen. Durch diese Anzahl an zusätzlichen Parametern würde das Modell nicht mehr in den Grafikkartenspeicher der genutzten Grafikkarte passen. Daher wäre ein Geschwindigkeitsvergleich mit anderen neuronalen Netzen die mithilfe der Grafikkarte arbeiten

[20] Zumindest für holistische Verfahren. Konkret hat sich gezeigt, dass solange die Bildgröße die Dimensionalität des zu bildenden Unterraums überschreitet, eine Gesichtserkennung möglich ist.

nicht mehr möglich. Aus diesem Grund wird an dieser Stelle auf Farbinformationen für das einfache neuronale Netz verzichtet. Die genauen Auswirkungen auf die Genauigkeit durch verschiedene Farbmodi wird aus Zeitgründen in dieser Arbeit nicht näher untersucht.

3.5. Durchführung der Testreihe

Wie bereits beschrieben, durchläuft das Testskript alle Kombinationen von Datensätzen und Verfahren. Algorithmus 1 zeigt den Ablauf des Testskripts in Pseudocode.

Algorithmus 1 : Ablauf des Testskripts

```
begin
    datasetnames ← [...]
    augmentationfactors ← [1,5,10,15,20,25,30]
    folds ← 5
    filepaths ← constructFilepaths(datasetnames, augmentationfactors)
    fes ← constructFeatureExtractors()
    foreach path in filepaths do
        X, y ← loadDataset(path)
        splits ← splitDataset(X, y, folds)
        foreach fe in fes do
            // Classifiers can be dependent on FEs
            clfs ← constructClassifiers(fe)
            foreach clf in clfs do
                foreach split in splits do
                    XTrain, yTrain, XTest, yTest ← prepareData(X, y, split)
                    // Measure Timings for each following step
                    fitExtractor(fe, XTrain, yTrain)
                    featuresTrain, featuresTest ← extractFeatures(fe, XTrain, XTest)
                    trainClassifier(clf, featuresTrain, yTrain)
                    evaluateClassifier(clf, featuresTest, yTest)
                    writeResults()
                end
            end
        end
    end
end
```

Die AIO-Verfahren werden ähnlich verarbeitet, es ergeben sich allerdings weniger Verschachtelungen. Während der praktischen Umsetzung haben sich einige Probleme mit dem verfügbaren Arbeitsspeicher ergeben. Um diese zu umgehen war es letztlich nötig, die AIO-Verfahren separat auszuführen. Das Testskript wurde auf einem Desktop-PC ausgeführt. Dieser beinhaltet einen Prozessor mit vier Kernen a 3,4 GHz. Dem System stehen 12 GB Arbeitsspeicher zur Verfügung. Der PC beinhaltet zudem eine GPU in Form einer NVidia GTX660. Diese Grafikkarte wird, wenn möglich, zur Beschleunigung der Berechnung der neuronalen Netze genutzt. Bei vorläufigen Tests hat sich ergeben, dass dadurch die Trainingsdauer um einen Faktor 6-7 kürzer ist, als wenn nur die CPU genutzt wird. Als Betriebssystem kommt ein Ubuntu 16.04 zum Einsatz. Diese Einschränkung ergibt sich durch die Installation von **CUDA**, einem Framework von NVidia für parallele Berechnungen auf GPUs [105]. Zum Zeitpunkt der Erstellung dieser Arbeit ist Ubuntu 16.04 die aktuellste unterstützte Betriebssystemversion. Eine vollständige Anleitung zur Installation aller benötigten Softwarepakete, sowie sämtlicher Quelltexte zur Erhebung der Daten sind unter [106] einsehbar. Das Sammeln aller Daten nimmt ungefähr 30 Stunden in Anspruch. Die Daten liegen im Anschluss als **Dataframe** vor, einer Datenstruktur für tabellarische Daten der Python-Bibliothek **Pandas**. Nach einer Konvertierung in ein CSV-Format werden diese Daten im Anschluss analysiert und ausgewertet.

3.6. Auswertung

Im Folgenden Kapitel werden die gesammelten Testdaten ausgewertet und die Ergebnisse ausführlich beschrieben. Die Auswertung erfolgt mithilfe von R, einer Skriptsprache für statistische Untersuchungen [107]. Alle Grafiken im folgenden Kapitel wurden, sofern nicht anders gekennzeichnet, in R mit dem Paket **ggplot2** [108] erzeugt. Die Daten liegen in zwei separaten Dateien vor. Eine enthält die Ergebnisse der Verfahrenskombinationen von FE und Klassifizierer, eine weitere die der AIO-Verfahren. Zuerst werden beide getrennt betrachtet und anschließend die jeweils vielversprechendsten Verfahren noch einmal gegenüber gestellt. Sowohl das Skript als auch die Dateien mit den Rohdaten sind ebenfalls unter [106] einsehbar.

3.6.1. Kombinationsverfahren

In diesem Kapitel werden die Ergebnisse der Kombinationsverfahren, welche aus einem FE und einem Klassifizierer bestehen, ausgewertet. Zu Beginn ist es sinnvoll, einen Überblick über sämtliche Verfahren und ihre erzielten Genauigkeiten zu erhalten. Dafür wird ein Boxplot-Diagramm, zu sehen in Abbildung 27, erstellt.

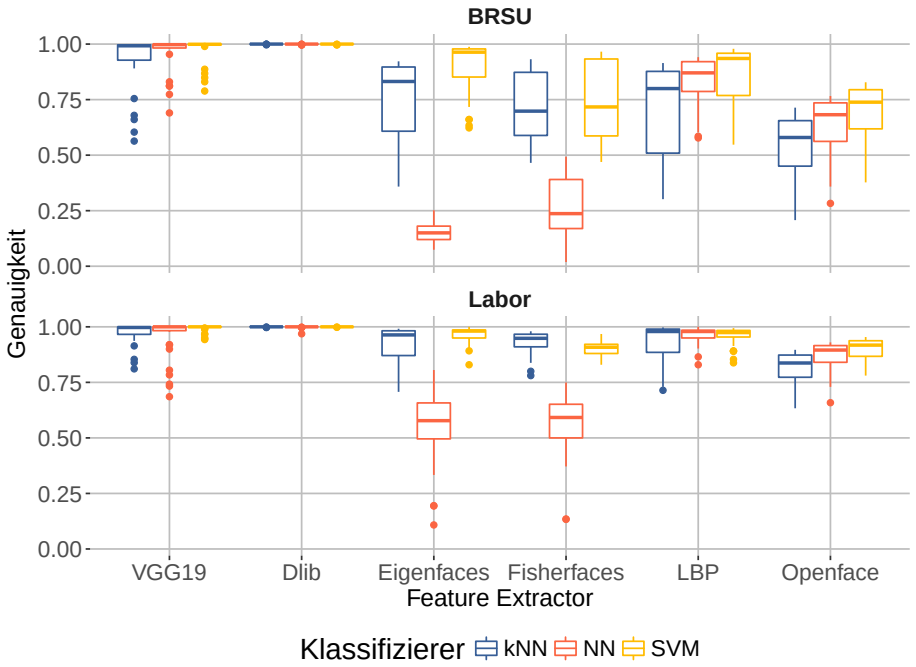


Abbildung 27: Genauigkeit in Abhängigkeit des verwendeten Datensatzes, FE und Klassifizierer.

Die Genauigkeit sämtlicher Verfahrenskombinationen wird dabei in Abhängigkeit des FE und Klassifizierers, sowie aufgeteilt nach Datensatz dargestellt. Ein Boxplot zeigt dabei den Median, das obere sowie untere Quartil, sowie die Antennen welche maximal das 1,5-Fache des Interquartilabstands lang sind. Die Punkte stellen Werte dar, welche nicht mehr in diesen Bereich fallen und weisen auf mögliche Ausreißer hin. In diesem Fall sind die Streuungen darauf zurückzuführen, dass die Darstellung sämtliche Augmentierungsfaktoren beinhaltet. Bei geringer oder nicht durchgeführter Augmentierung liegen nur wenig Daten vor, wodurch einige Verfahren nur sehr schlechte Ergebnisse liefern. Dadurch ergeben sich Datenpunkte, welche als Ausreißer nach unten erscheinen, sowie die Streuung erhöhen. Im weiteren Verlauf des Kapitels werden die Auswirkungen des Augmentierungsfaktors näher betrachtet. Dennoch gibt die Grafik einen Überblick darüber, welche Kombinationen bessere Ergebnisse als andere liefern. Sehr auffällig sind die durchgängig hohen Genauigkeiten des Dlib-Deskriptors, gefolgt vom VGG19-Deskriptor und LBPs. Es ist erkennbar, dass SVMs die geringsten Streuungen und höchsten Genauigkeiten erreichen, gefolgt von neuronalen Netzen und zum Schluss folgt kNN. Eine Ausnahme bildet hier die Kombination von Eigenfaces und Fisherfaces mit neuronalen Netzen, welche durchweg schlechte Ergebnisse erreichen. Die genauen Gründe dafür konnten im Rahmen dieser Arbeit nicht mehr ermittelt werden. Auffällig ist auch, dass die Streuungen im Labordatensatz teilweise wesentlich geringer sind, sowie die Genauigkeiten bei manchen Verfahren höher ausfallen, als im BRSU-Datensatz.

Mögliche Gründe hierfür sind die geringere Anzahl an Personen im Datensatz, sowie die größere durchschnittliche Anzahl an Bildern pro Person. Die Verfahren haben demnach mehr Informationen zur Verfügung, um zwischen weniger Klassen zu entscheiden. In Abbildung 28 wird die Abhängigkeit der Genauigkeit zu den FEs und dem Augmentierungsfaktor dargestellt.

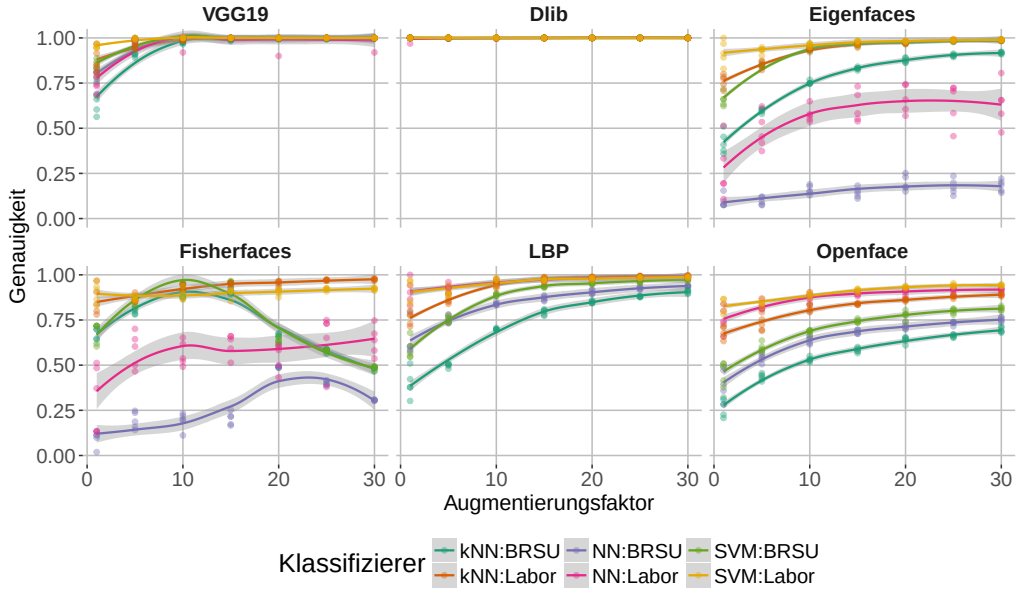


Abbildung 28: Genauigkeit der einzelnen Verfahren in Abhängigkeit zum Augmentierungsfaktor mit Regressionsmodell.

In der Abbildung ist für jeden Kreuzvalidierungseintrag ein Punkt eingetragen. Über alle Punkte sowie Augmentierungsfaktoren wird mittels lokaler Regression eine Linie gelegt. Die Schattierung der Linie stellt ein Konfidenzintervall über den Standardfehler an diesem Punkt dar[21]. Die Klassifizierer werden dabei abhängig vom jeweiligen Datensatz betrachtet. Ein größerer Augmentierungsfaktor ist gleichzusetzen mit einer größeren Datenmenge. Diese führt zu höheren Genauigkeiten was bekräftigt, dass die Augmentierung ein hilfreicher oder sogar notwendiger Schritt ist, um gute Ergebnisse zu erzielen. Ausnahmen bilden hier zum einen der Dlib-Deskriptor, welcher unabhängig vom Augmentierungsfaktor gute Ergebnisse liefert. Zum anderen zeigt der FE Fisherfaces starke Schwankungen. Erklärbar sind diese durch den Wechsel auf den SVD-Solver ab einer bestimmten Datenmenge. Interessant ist hierbei, dass im Labordatensatz dennoch gute Ergebnisse von den Klassifizierern kNN und SVM erzielt werden. Für den BRSU-Datensatz scheint sich die Vorhersage aus Kapitel 3.1.4 durch [91] zu bestätigen, dass der SVD-Solver bei der Gesichtserkennung Unzulänglichkeiten aufweist. Allge-

[21] Für mehr Informationen bezüglich der Erzeugung der Regressionsmodelle sei auf die Pakete „ggplot2“ und „stats“ von R verwiesen.

mein zeigt sich dennoch eine Steigerung der Genauigkeiten durch größere Datenmengen. Bezüglich der Genauigkeit zeigt sich, dass die Dlib- und VGG19-Deskriptoren die besten Ergebnisse erzielen. Mit entsprechenden Datenmengen und dem richtigen Klassifizierer erreichen auch LBP und Eigenfaces Genauigkeiten über 90 %.

Im nächsten Schritt wird in Abbildung 29 die Genauigkeit der Klassifizierer selbst betrachtet.

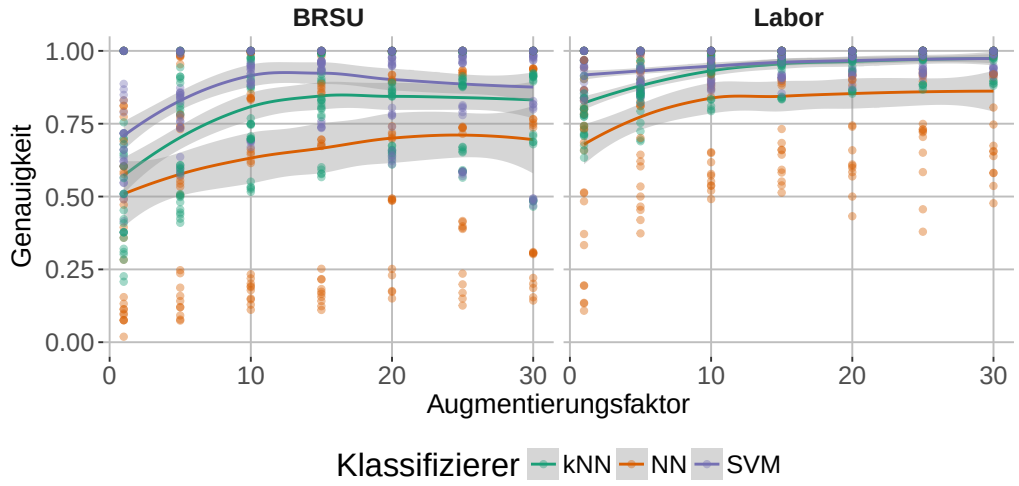


Abbildung 29: Genauigkeit der einzelnen Klassifizierer in Abhängigkeit zum Augmentierungsfaktor und getrennt nach Datensatz.

Da die Ergebnisse sämtlicher FEs gemeinsam betrachtet werden, sind starke Streuungen zu sehen. Dennoch bietet das Diagramm eine Übersicht, wie sich die Klassifizierer im Allgemeinen bei steigender Datenmenge verhalten. Auf dem BRSU-Datensatz, welcher insgesamt mehr Klassen und Bilder enthält, liefern SVM und kNN die besten Ergebnisse. Ab einer bestimmten Datenmenge nimmt die Genauigkeit wieder etwas ab und schwankt stärker. Eine Ursache dafür könnte sein, dass die Modelle aufgrund der großen Datenmenge überangepasst werden. Auf dem Labordatensatz, welcher wie oben beschrieben weniger Klassen und mehr Bilder pro Person aufweist, tritt dieses Verhalten nicht auf. Bei beiden Datensätzen treten starke Streuungen auf, wenn ein neuronales Netz zur Klassifizierung genutzt wird. Ein Grund könnte die mangelnde manuelle Anpassung der Parameter des neuronalen Netzes an die Gegebenheiten der Daten sein.

Neben der Genauigkeit ist auch die Trainingsdauer ein wichtiger Aspekt. Im Testskript wurden dafür die Zeiten für FE und Klassifizierer getrennt betrachtet. Da allerdings lediglich die FEs Fisherfaces und Eigenfaces trainiert werden müssen, wird im Folgenden nur die Trainingsdauer der einzelnen Klassifizierer verglichen. Abbildung 30 zeigt diese für alle FEs in Abhängigkeit zum Augmentierungsfaktor.

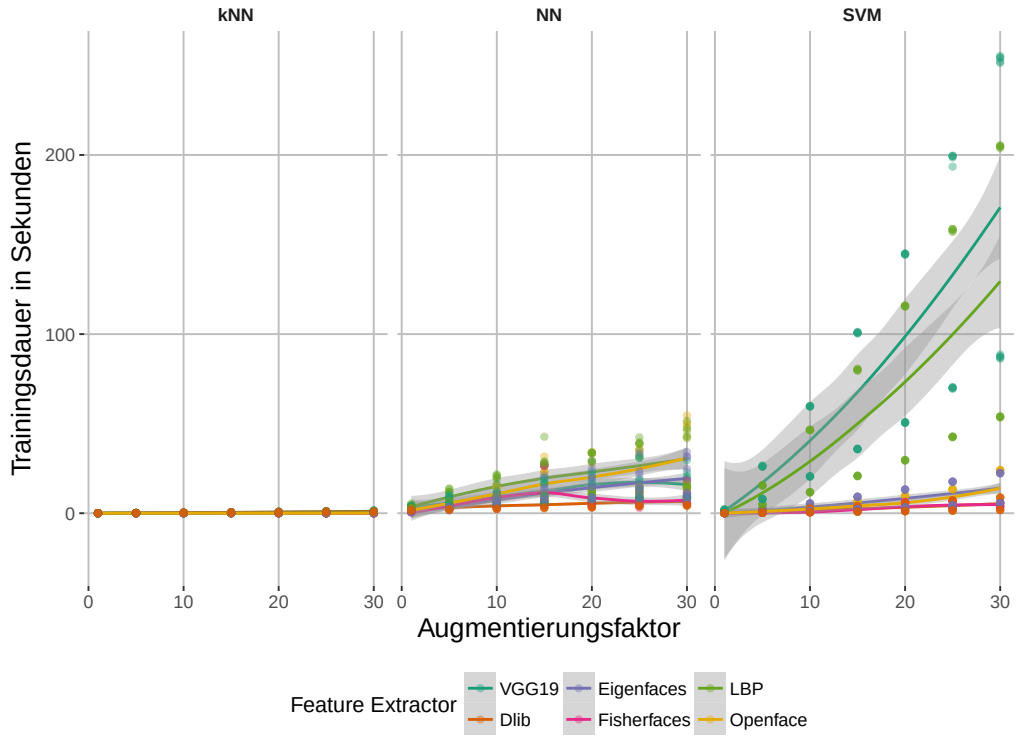


Abbildung 30: Trainingsdauer der einzelnen Klassifizierer, abhängig vom Augmentierungsfaktor und FE, in Sekunden.

Die Darstellung verdeutlicht die Unterschiede in der Trainingsdauer der verschiedenen Klassifizierer. Unabhängig von der Länge der Deskriptoren, ist die Trainingsdauer für kNN sehr gering und beträgt nie mehr als zwei Sekunden. Grund dafür ist, dass keine mathematischen Operationen nötig sind, es wird lediglich eine interne Kopie der zum Training verwendeten Deskriptoren erstellt. Neuronale Netze zeigen hingegen eine stärkere Steigung durch größere Datenmengen. Am deutlichsten ist eine Abhängigkeit bei SVMs zu sehen, vor allem wenn LBP oder der VGG19-Deskriptor als FE genutzt werden. Die Feature-Vektoren von diesen sind mit 1.764 und 4.608 Zahlen pro Vektor am längsten von allen FEs, was die längere Trainingsdauer erklärt.

Als nächstes wird in Abbildung 31 die Trainingsdauer auf beiden Datensätzen verglichen, um Unterschiede zwischen verschiedenen Datenmengen während des Trainings darzustellen.

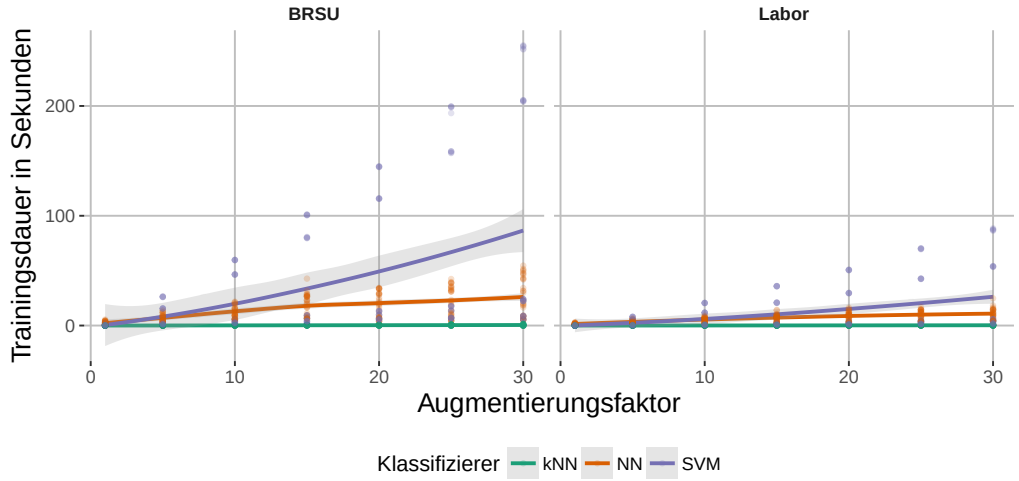


Abbildung 31: Trainingsdauer der Klassifizierer auf den verwendeten Datensätzen in Sekunden.

Dabei wird deutlich, dass der SVM-Klassifizierer auf dem BRSU-Datensatz eine größere Steigung abhängig zum Augmentierungsfaktor aufweist. Das zeigt, dass die Anzahl der Klassen sowie die Datenmenge einen Einfluss auf die Trainingsdauer hat, was typisch für SVMs ist.

Des Weiteren ist auch die Zeit wichtig, die ein FE benötigt, um Merkmale aus einem zu klassifizierendem Bild zu extrahieren. In Abbildung 32 sind diese sogenannten Durchlaufzeiten dargestellt. Diese bestehen dabei aus dem Durchschnitt von jeweils zehn Einzelmessungen, um eventuelle Schwankungen durch Übertragungszeiten zwischen Arbeitsspeicher, CPU und GPU auszugleichen.

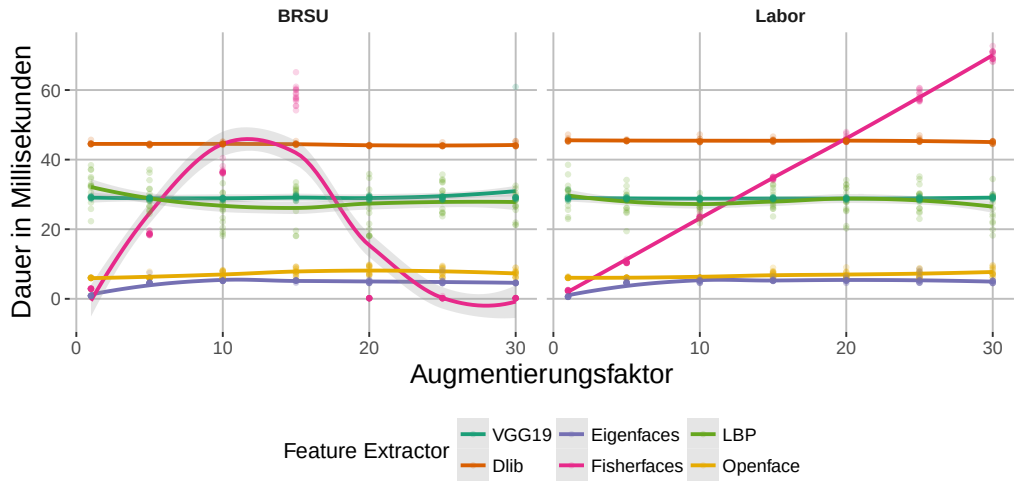


Abbildung 32: Durchlaufzeiten der einzelnen FEs in Millisekunden, abhängig vom Augmentierungsfaktor und getrennt nach Datensatz.

Die Zeiten bleiben nahezu konstant, lediglich bei Fisherfaces ist ein Anstieg zu sehen. Im BRSU-Datensatz ist deutlich sichtbar, dass der Wechsel auf den SVD-Solver die Durchlaufzeit verringert. Abgesehen davon ist der Anstieg, wie auch im Labor-Datensatz sichtbar, linear. Die durchschnittliche Durchlaufzeit beträgt 43 ms wobei der Median mit 5 ms wesentlich geringer ist. Ebenso wichtig sind die Durchlaufzeiten der Klassifizierer, welche in Abbildung 33 dargestellt sind.

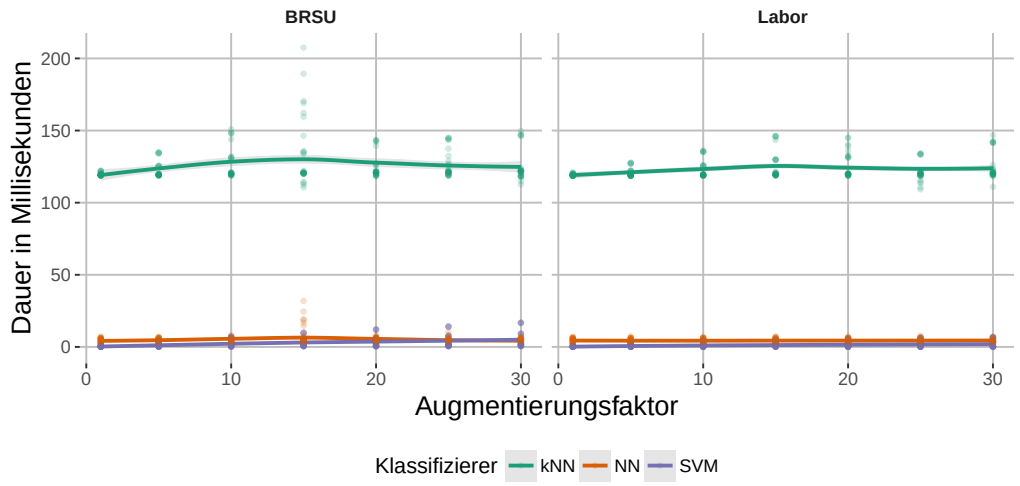


Abbildung 33: Durchlaufzeiten der einzelnen Klassifizierer in Millisekunden, abhängig vom Augmentierungsfaktor und getrennt nach Datensatz.

Die Zeiten sind durchgehend konstant, wobei das kNN-Verfahren mit durchschnittlich 124 ms mehrere Größenordnungen länger dauert, als die anderen Klassifizierer, welche nur wenige Millisekunden benötigen. Diese großen Unterschiede sind in der Auswahl entsprechend zu berücksichtigen. Eine weitere Möglichkeit, die Unterschiede in der Genauigkeit zwischen den Verfahren zur verdeutlichen, ist der Tukey-**Honestly Significant Difference (HSD)**-Test. Dabei werden Konfidenzintervalle über die Mittelwertdifferenzen der verschiedenen Verfahren erzeugt, was einem paarweisen Vergleich entspricht. In Abbildung 34 ist das Ergebnis des Tukey-Tests für die Genauigkeit in Abhängigkeit der einzelnen FEs dargestellt.

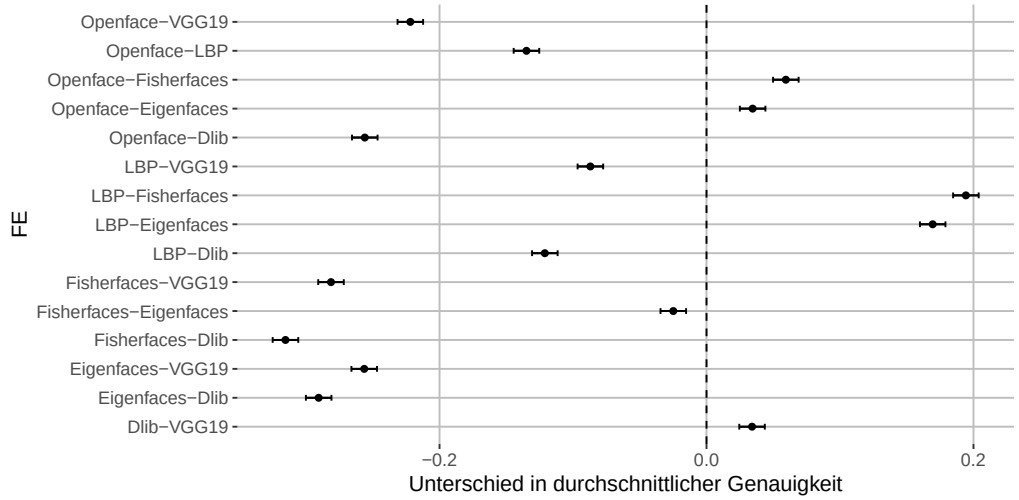


Abbildung 34: Eine Darstellung der vom Tukey-HSD-Test erzeugten Konfidenzintervalle für alle FEs.

Grundlage für die Berechnung des Tukey-HSD-Tests ist eine Varianzanalyse auf den Daten, in welche sämtliche Wechselwirkungen zwischen den Einflussfaktoren (FE, Klassifizierer, Datensatz, Augmentierungsfaktor) mit einbezogen werden. Umfasst eines der Konfidenzintervalle die Nulllinie, bedeutet dies, dass die Mittelwertdifferenz zwischen diesen beiden Verfahren nicht signifikant ist. Zudem gibt die Lage links oder rechts von der Nulllinie an, welches der beiden verglichenen Verfahren einen höheren Mittelwert aufweist und, bezogen auf die Genauigkeit, besser ist. Für die FEs zeigt sich, dass sämtliche Differenzen signifikant sind. Die besten Verfahren basieren auf den VGG19- und Dlib-Deskriptoren. Die Ergebnisse des Tukey-HSD-Tests für die Klassifizierer sind in Abbildung 35 dargestellt.

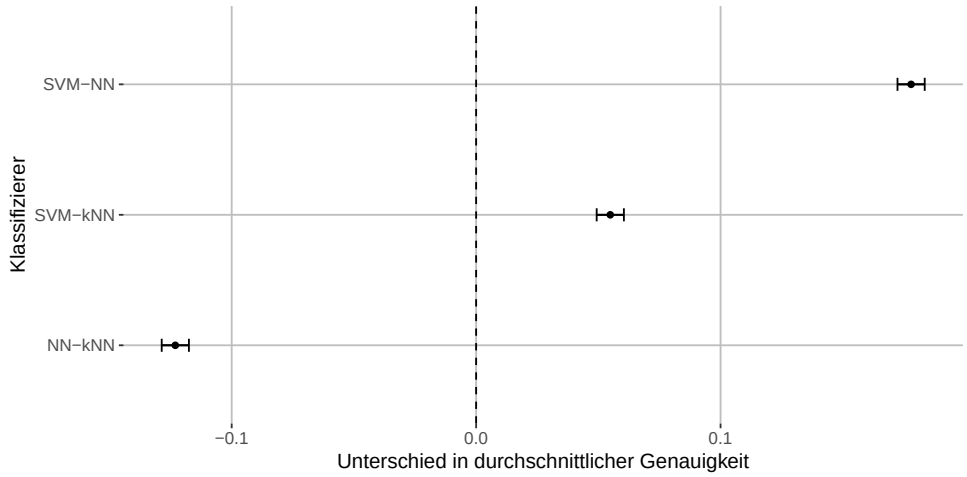


Abbildung 35: Eine Darstellung der vom Tukey-HSD-Test erzeugten Konfidenzintervalle für alle Klassifizierer.

Die Mittelwertdifferenzen sind ebenfalls alle signifikant, da keines der Konfidenzintervalle über der Nulllinie liegt. Zudem ist die Differenz der SVMs zu den anderen Klassifizierern am höchsten. Die Auswertung der Kombinationsverfahren bis zu diesem Punkt hat ergeben, dass die VGG19- und Dlib-Deskriptoren in Kombination mit SVMs und dem kNN-Verfahren die höchsten Genauigkeiten ergeben. Der einzige Nachteil von kNN ist die vergleichsweise lange Verarbeitungsdauer um Merkmale zu klassifizieren. Im folgenden werden die Ergebnisse der beiden AIO-Verfahren ausgewertet.

3.6.2. All-In-One-Verfahren

Die AIO-Verfahren werden ähnlich wie die Kombinationsverfahren zuerst mit einem Boxplot in Abbildung 36 verglichen.

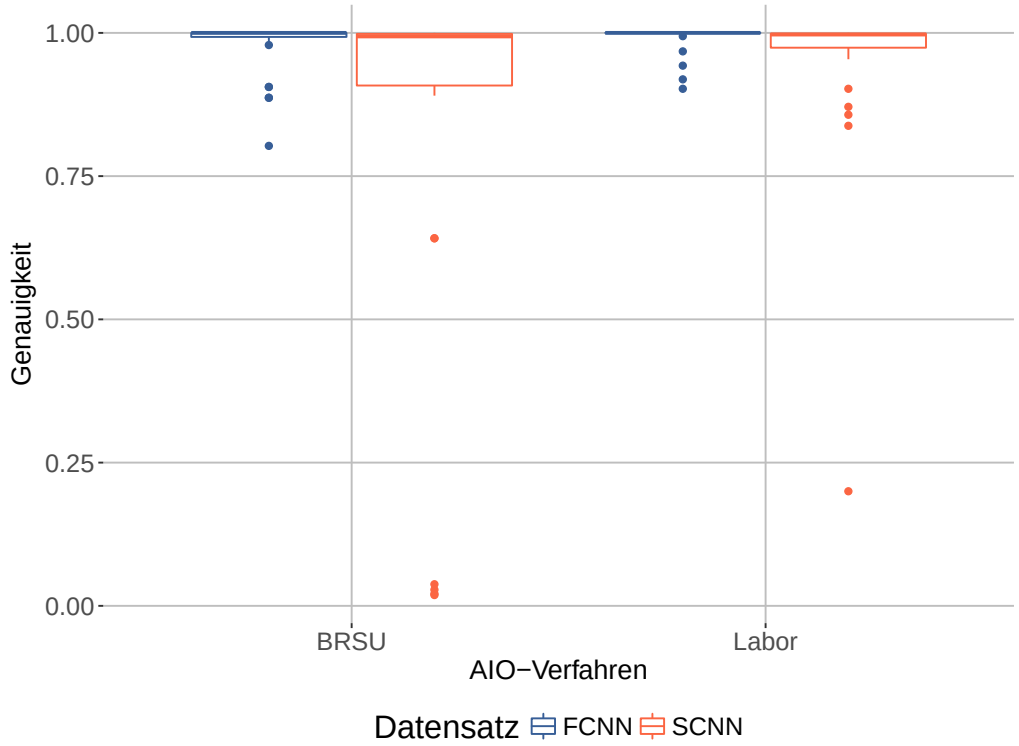


Abbildung 36: Vergleich der Genauigkeiten der AIO-Verfahren auf beiden Datensätzen.

Das einfache CNN (SCNN) streut auf beiden Datensätzen stark, jedoch weniger auf dem Labordatensatz. Die teilweise niedrigen Genauigkeiten kommen durch die geringe Datenmenge in den nicht augmentierten Datensätzen zustande. Das Netzwerk erhält nicht genügend Informationen um gute Ergebnisse zu liefern. Das abgestimmte CNN (FCNN), welches intern auf dem VGG19-CNN aufbaut, weist wesentlich bessere Ergebnisse auf. Grund dafür ist, dass es vortrainiert ist und damit bereits Informationen beinhaltet, auf

die zurückgegriffen werden kann. Um den Einfluss der Datenmenge besser einzuschätzen, ist in Abbildung 37 ein lokales Regressionsmodell der Genauigkeit in Abhängigkeit zum Augmentierungsfaktor dargestellt.

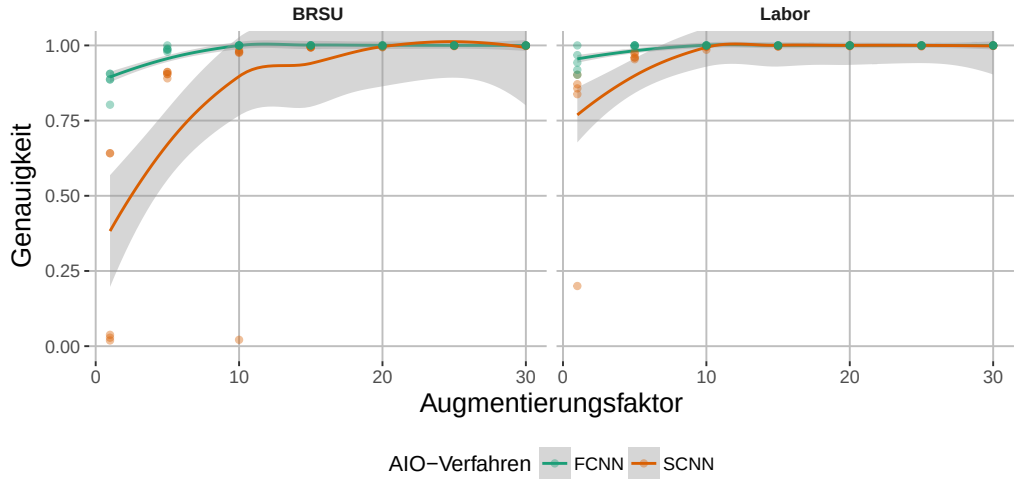


Abbildung 37: Vergleich der Genauigkeiten der AIO-Verfahren auf beiden Datensätzen, abhängig vom Augmentierungsfaktor.

Dabei ist gut zu erkennen, das beide Verfahren auf dem Labor-Datensatz mit weniger Daten gute Ergebnisse erreichen, vor allem das einfache CNN. Das abgestimmte CNN konvergiert auf beiden Datensätzen bei geringeren Augmentierungsfaktoren früher. Im Folgenden wird die Dauer der Trainings- und Klassifizierungs-Phasen betrachtet. In Abbildung 38 ist die Trainingsdauer der beiden Verfahren zu sehen.

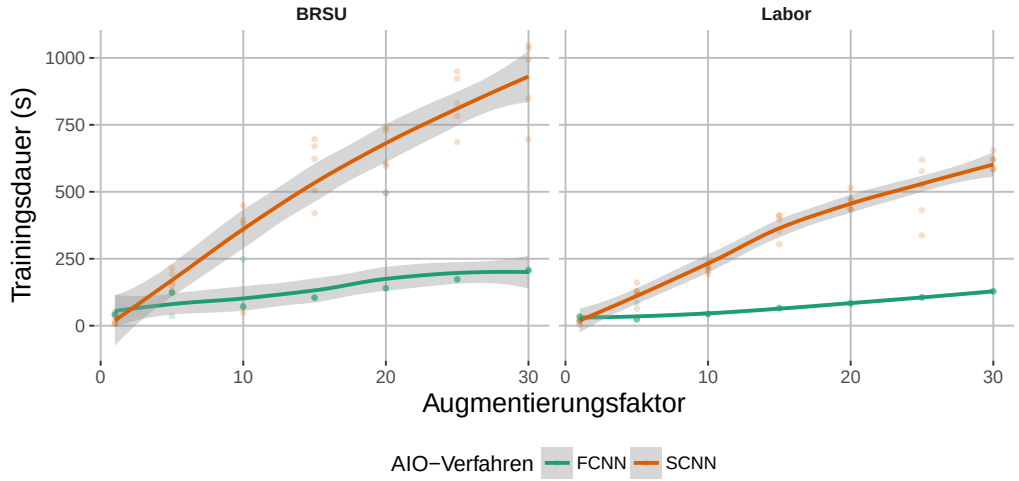


Abbildung 38: Vergleich der Trainingsdauer der AIO-Verfahren auf beiden Datensätzen, abhängig vom Augmentierungsfaktor.

Beide Verfahren zeigen einen linearen Zusammenhang zwischen Trainingsdauer und Datenmenge. Das abgestimmte CNN skaliert jedoch wesentlich besser, was daran liegt, dass erst Merkmale aus dem VGG19-Netz extrahiert werden, ähnlich wie im VGG19-Deskriptor. Danach erfolgt die Trainingsphase auf einem im Vergleich zum einfachen CNN kleinen neuronalen Netz. Wichtig für den praktischen Einsatz ist die Zeit, innerhalb der ein Bild klassifiziert werden kann. Diese wird in Abbildung 39 dargestellt.

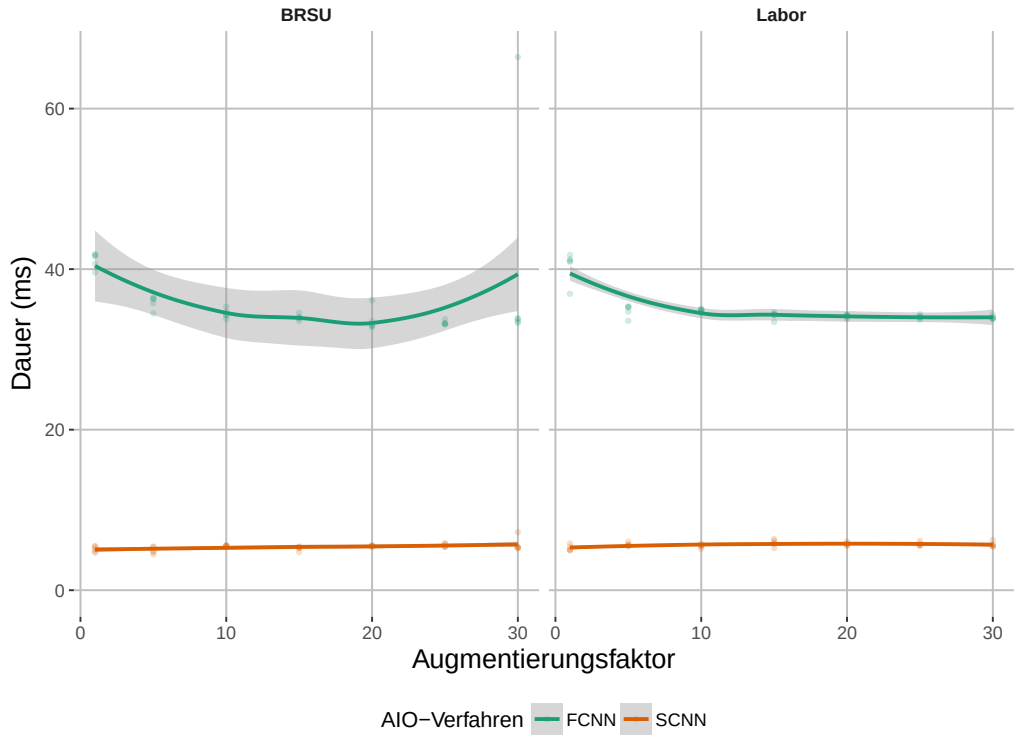


Abbildung 39: Vergleich der Klassifizierungsdauer der AIO-Verfahren auf beiden Datensätzen, abhängig vom Augmentierungsfaktor.

Abgesehen von einigen Schwankungen für das abgestimmte CNN sind die Klassifizierungszeiten konstant, ohne signifikante Zusammenhänge zu den Augmentierungsfaktoren. Die Klassifizierung mit dem einfachen CNN ist sechs mal schneller als mit dem abgestimmten, was auf die insgesamt geringere Modellgröße zurückzuführen ist. Die durchschnittliche Klassifizierungsdauer von 35 ms für das abgestimmte CNN ist allerdings immer noch kurz genug für eine praktische Anwendung. Da es zudem wesentlich bessere Genauigkeiten mit weniger Daten erreicht, ist es bei der Auswahl zu bevorzugen. Im Anschluss werden nochmal die bisher besten Verfahrenskombinationen und das abgestimmte CNN gegenübergestellt, um daraufhin eine endgültige Auswahl zu treffen.

3.6.3. Abschließender Vergleich der Verfahren

Im folgenden Abschnitt werden die besten Kombinationen von FEs und Klassifizierern sowie das abgestimmte CNN miteinander verglichen. Von den FEs erzielen die VGG19- und Dlib-Deskriptoren die besten Ergebnisse bezüglich der Genauigkeit. Als Klassifizierer werden SVMs und kNN ausgewählt. Zwar benötigt das neuronale Netz weniger Zeit für die Klassifizierung als kNN, die Genauigkeit ist jedoch schlechter. Das abgestimmte CNN wird ausgewählt, weil es bis auf die Klassifizierungsdauer in allen belangen bessere Ergebnisse erzielt, als das einfache CNN. Aus den vorherigen Untersuchungen ist bereits hervorgegangen, dass keines der Verfahren für eine Klassifizierung mehr als 200 ms benötigt. Dadurch können pro Sekunde fünf Bilder klassifiziert werden, was für die praktische Nutzung durch den zu entwickelnden Prototyp ausreichend ist. Daher werden nur die erzielten Genauigkeiten betrachtet. Abbildung 40 stellt das Ergebnis eines Tukey-HSD-Tests mit allen verbleibenden Verfahren dar.

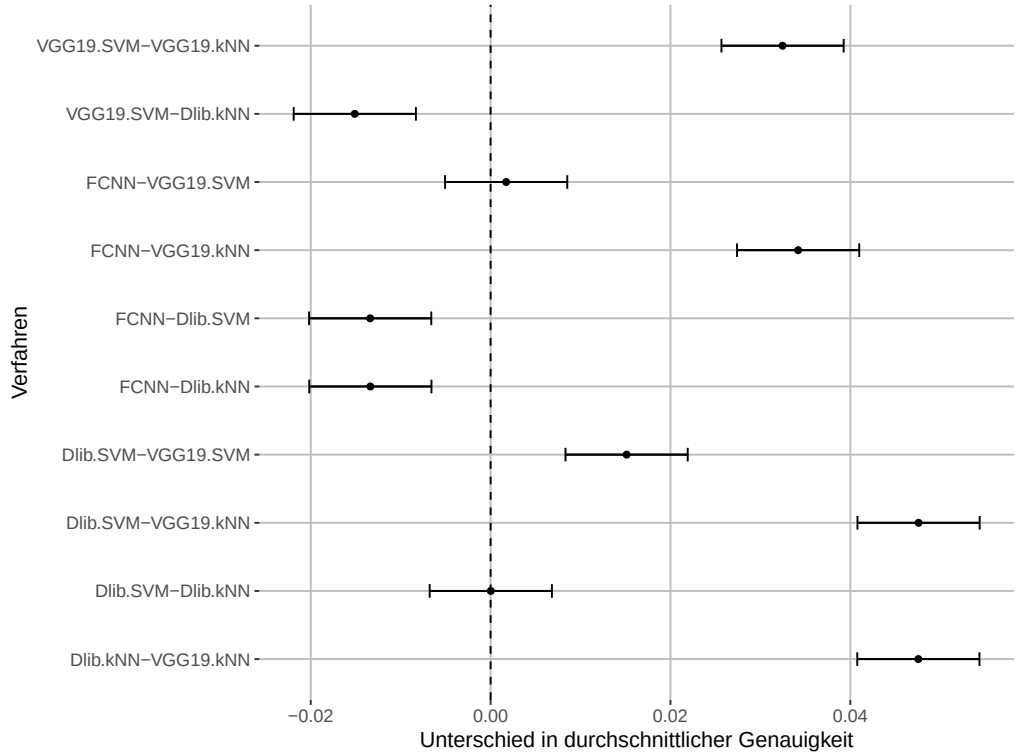


Abbildung 40: Konfidenzintervalle über die Mittelwertdifferenzen der Genauigkeiten aller verbleibenden Verfahren.

Die Darstellung verdeutlicht, dass der Dlib-Deskriptor die besten Genauigkeiten erzielt, unabhängig vom verwendeten Klassifizierer. An zweiter Stelle stehen das abgestimmte CNN und der VGG19-Deskriptor mit einer SVM als Klassifizierer. Der Unterschied

zwischen diesen beiden ist nicht signifikant. Da beide Verfahren auf dem vortrainierten VGG19-Netzwerk aufbauen, ist dieses Ergebnis naheliegend. Im Allgemeinen sind die Unterschiede zwischen den Verfahren jedoch sehr gering.

3.7. Auswahl/Begründung

Die Auswertungen aus dem vorherigen Kapitel zeigen, dass der Dlib-Deskriptor die beste Wahl für einen FE darstellt. Er weißt von allen Verfahren die höchste Genauigkeit auf, was sich aus der Spezialisierung auf Gesichtserkennung und die große Datenmenge auf der er basiert ergibt. Die Wahl des Klassifizierers ist abhängig davon, ob das Training oder die Klassifizierung schnell ablaufen soll. Im praktischen Einsatz ist eine kurze Klassifizierungszeit zu bevorzugen. Jedoch könnte die Notwendigkeit Personen auch als unbekannt einstufen zu können (siehe Kapitel 4.1) die Verwendung von kNN begünstigen. Das Verfahren ermöglicht es, die Distanzen zu den umliegenden k Nachbarn zu ermitteln. Sind diese größer als ein zu bestimmender Grenzwert, gilt die abgebildete Person als unbekannt. Sollten die konstant hohen Genauigkeiten des Dlib-Deskriptors nicht reproduzierbar sein, sollte das abgestimmte CNN verwendet werden.

4. Implementierung des Prototyps

Der Vergleich der Gesichtserkennungsverfahren im vorherigen Kapitel hat ergeben, dass die Kombination des Dlib-Deskriptors mit einer SVM als Klassifizierer sehr gute Ergebnisse liefert. Darauf aufbauend soll im folgenden Kapitel die praktische Nutzbarkeit durch einen Prototypen untersucht werden. Durch die Nutzung in einem Heimautomatisierungsumfeld ergeben sich einige Anforderungen, welche im ersten Abschnitt betrachtet werden. Im Anschluss erfolgt die Beschreibung der Implementierung des Prototyps, sowie Erläuterungen zu den Entscheidungen welche aufgrund der Anforderungen während der Entwicklung getroffen wurden. Es ist anzumerken, dass es der Zweck des Prototyps ist, die Machbarkeit zu zeigen, nicht ein fertiges Produkt zu entwickeln.

4.1. Anforderungen an den Prototyp

Das Anwendungsszenario für den Prototyp ist ein typisches Heimautomatisierungsumfeld. Dabei existieren in einem Wohnhaus oder einer Wohnung mehrere physische Komponenten, welche als Aktoren und Sensoren dienen. Die Steuerung und Verarbeitung der gesammelten Informationen wird, je nach Ausprägung, von einem oder mehreren Systemen übernommen. Ein Hauptaugenmerk liegt immer auf der Autonomie und Ressourcensparsamkeit der Komponenten. Sie sollen möglichst wenig Platz, Strom, Kosten und Wartungsaufwand in Anspruch nehmen.

Ein Gesichtserkennungssystem in diesem Umfeld sollte ebenfalls diesen Ansprüchen gerecht werden. Die Nutzung von mehreren Rechnern oder Grafikkarten zur Berechnung, sowie großen Datenmengen ist dadurch nicht möglich. Dazu kommt, dass wie in Kapitel 1.1 beschrieben, aus Gründen des Datenschutzes und der Unabhängigkeit von einer Internetverbindung, sämtliche Daten lokal vorliegen sollen. Dadurch kann das Gesichtserkennungssystem auch genutzt werden, wenn keine Internetverbindung besteht. Aus diesen Gründen ergeben sich folgende grundlegende Anforderungen.

Das System soll möglichst wenige Daten benötigen. Konkret betrifft das die Anzahl der Bilder von Personen. Mit immer günstigerem Speicherplatz und Festplatten im Terabyte-Bereich dient diese Anforderung lediglich der Abgrenzung zu kommerziellen Nutzern und Anbietern von Gesichtserkennungssoftware, welche häufig Millionen von Bildern als Datenbasis nutzen. Zudem soll der Prototyp auch ohne Internetverbindung und damit nur im lokalen Netzwerk arbeiten. Die benötigte Rechenleistung sollte nicht übersteigen, was aktuell oder in naher Zukunft in Haushalten üblich ist. Zum Zeitpunkt der

Erstellung der Arbeit wurden keine aktuellen Statistiken diesbezüglich gefunden, weswegen der PC, mit dem die Testreihen durchgeführt wurden, als Standard angesehen wird (siehe Kapitel 3.5). Dieses System ist wesentlich leistungsstärker als beispielsweise ein Raspberry Pi, welcher in der Heimautomatisierung weit verbreitet ist. Dennoch sollte die Gesichtserkennung auch auf schwächeren Systemen lauffähig sein, zulasten der Trainings- und Klassifizierungsdauer.

Neben den oben beschriebenen Hardwareanforderungen werden an den Prototyp auch Anforderungen gestellt, welche sich auf die Benutzbarkeit beziehen. Diese soll vor allem möglichst einfach sein und kein Fachwissen bezüglich der Gesichtserkennung erfordern. Zudem sollte die nötige Benutzerinteraktion minimal gehalten werden. Der Prototyp muss Aktionen bereitstellen, welche das Zuordnen von Bildern zu bestimmten Personen, das Training von Modellen, die Korrektur von Klassifizierungen sowie eine Live-Ansicht von hinzukommenden Bildern ermöglichen. Neben der Aussage, welche Person in einem Bild zu sehen ist, soll der Prototyp auch in der Lage sein zu entscheiden, ob eine Person unbekannt ist. Damit einher geht die Aussage über eine Wahrscheinlichkeit, mit der das zugrunde liegende Modell ein Bild einer Person zuordnet. Das Training von neuen Modellen wird der zeitaufwändigste Vorgang sein, weswegen dieser im Hintergrund durchführbar sein muss. Während des Trainings muss der Prototyp weiterhin benutzbar sein. Ist ein Trainingsvorgang abgeschlossen, sollen Informationen bezüglich der Genauigkeit eines Modells abrufbar sein. Schlussendlich müssen alle Vorgänge im Prototyp über eine Schnittstelle, auch **Application Programming Interface (API)** genannt, einsehbar und steuerbar sein, damit eine Integration in bestehende Heimautomatisierungssysteme möglichst einfach ist. Diese Anforderungen sind in Tabelle 3 zusammengefasst.

Tabelle 3: Anforderungen an den Prototyp

Nr.	Anforderung
1	Wenig Bilder pro Person
2	Keine Internetverbindung nötig
3	Einfache Benutzbarkeit
4	Manuelle Zuordnung von Bildern zu Personen
5	Manuelles Starten des Trainings
6	Korrektur von durchgeführten Klassifizierungen
7	Live-Ansicht von Klassifizierungen
8	Einordnung einer Person im Bild als unbekannt
9	Durchführung des Trainings im Hintergrund
10	Anzeige von Informationen über trainierte Modelle
11	Steuerung über eine API

4.2. Umsetzung des Prototyps

Im folgenden Abschnitt wird die Umsetzung des Prototyps und die dabei getroffenen Entscheidungen erläutert. Auch wenn der Prototyp kein fertiges Produkt darstellt, können die bei der Entwicklung gesammelten Informationen in der Weiterentwicklung hilfreich sein. Der Quelltext des Prototyps ist öffentlich auf GitHub verfügbar [109], einer Plattform zur Quelltextverwaltung. In den vorhergehenden Anforderungen werden sowohl eine einfache Benutzung des Prototyps, sowie eine Schnittstelle für andere Anwendungen genannt. Um diese beiden Anforderungen zu erfüllen, bietet sich eine Webanwendung an. Diese läuft auf einem Server im lokalen Netzwerk und ermöglicht jedem Benutzer ohne weitere Anwendungsinstallationen den Prototyp zu nutzen. Zudem kann über Aufrufe von bestimmten **Uniform Resource Locators (URLs)** eine Schnittstelle definiert werden. Diese **REpresentational State Transfer (REST)**-API bietet allen Systemen, welche URLs aufrufen können, die Möglichkeit, Informationen vom Prototyp zu erhalten und diesen zu steuern. Zudem ergibt sich dadurch eine Unabhängigkeit von einer bestimmten Oberfläche. Für den Prototyp wird diese zwar bereitgestellt, kann jedoch ohne Änderungen an der Servertechnologie ausgetauscht werden. Die für diese Arbeit erstellte Oberfläche des Prototyps ermöglicht es dem Benutzer sämtliche oben beschriebenen Funktionen zu nutzen. Die Benutzung ist möglichst einfach und interaktiv gehalten, ohne den Benutzer mit technischen Details zu überfordern. Er kann neue Personen anlegen und diesen Bildern zuordnen. Auf einer Live-Ansicht werden aktuell klassifizierte Bilder dargestellt. Die Darstellung umfasst alle gefundenen Personen, die Wahrscheinlichkeit der Zuordnung, sowie die fünf ähnlichsten Personen und die zugehörigen Wahrscheinlichkeiten. Bereits durchgeführte Klassifizierungen können vom Benutzer nachträglich korrigiert werden. Zusätzlich wird eine Übersicht mit Informationen zu allen bisher trainierten Modellen bereitgestellt. Zudem wird eine sich automatisch aktualisierende Anzeige über eventuell laufende Trainingsvorgänge angezeigt. Abbildungen dieser Oberfläche sind als Screenshots im Anhang A.3 hinterlegt.

Als Programmiersprache für den Prototyp wird Python verwendet, um eine möglichst einfache Weiterverwendung der bereits entwickelten Komponenten zu ermöglichen. Für die Entwicklung einer Webanwendung mit Python gibt es verschiedene Möglichkeiten. In dieser Arbeit wird das Web-Framework Flask [110] verwendet, welches aktiv entwickelt wird und weit verbreitet ist. Zudem ermöglicht die Verwendung von Erweiterungen weitere Funktionen hinzuzufügen. Während der Entwicklung wird von Flask ein interner Webserver genutzt, was den Entwicklungsaufwand nochmal verringert. Von einem produktiven Einsatz mit diesem eingebauten Webserver wird allerdings abgeraten. Wie bereits erwähnt, werden einige der im Verlauf dieser Arbeit entwickelten Komponenten in den Prototyp integriert. Das betrifft die Methoden zur Gesichtsdetektion,

Augmentierung und Umwandlung von Bildern in den Dlib-Deskriptor sowie der dazugehörigen Hilfsmethoden. Bis auf geringe Anpassungen wurden diese Komponenten vollständig übernommen. Zum Speichern von Informationen wird eine Datenbank verwendet. Für diese Arbeit wird PostgreSQL als Datenbanksystem genutzt. Dieses kann aber durch beliebige andere ausgetauscht werden. Ermöglicht wird dies durch die Verwendung von SQLAlchemy [111], einem **Object Relational Mapper (ORM)** für Python, sowie der entsprechenden Flask-Erweiterung Flask-SQLAlchemy [112]. Das Datenbankmodell wird mit Python in Form von Klassen definiert, die entsprechenden SQL-Skripte werden im Anschluss automatisch generiert und auf der jeweiligen Datenbank ausgeführt. Die Python-Anwendung selbst verwendet intern nur die vorher definierten Klassen. SQL-Queries müssen keine geschrieben werden, da SQLAlchemy diese automatisch abhängig von der jeweiligen Datenbankoperation erzeugt und ausführt. Die Ergebnisse der Queries werden zur weiteren Verwendung in Instanzen der jeweiligen Klasse umgewandelt. Das vom Prototyp verwendete Datenmodell ist in Abbildung 41 dargestellt.

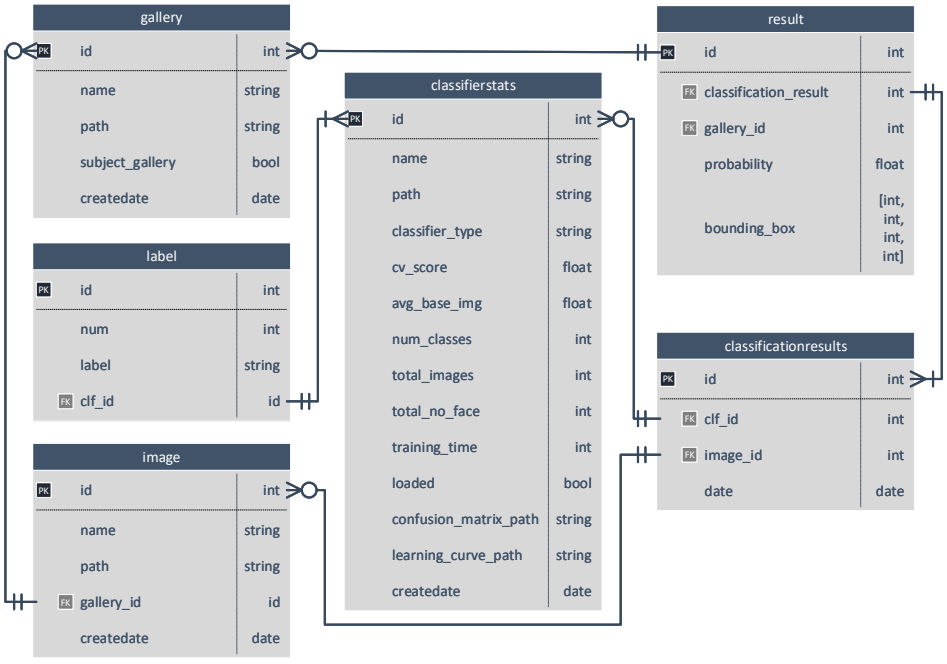


Abbildung 41: Datenbankmodell des Prototyps.

Die Tabellen **gallery** und **image** enthalten Informationen zu den Bildern der jeweiligen Personen. Eine Galerie beinhaltet sämtliche Bilder einer bestimmten Person, die dieser zugeordnet wurden. Dabei werden nicht die Bilder selbst, sondern die Dateipfade gespeichert. Dieses Konzept ist etwas aufwändiger, da die Struktur der Bilder und Galerien sowohl in der Datenbank als auch auf dem Dateisystem verwaltet werden muss. Dadurch wird allerdings die Wartbarkeit und Benutzbarkeit durch den Benutzer vereinfacht. Für

diesen lassen sich Bilder in einer Ordnerstruktur einfacher verwenden als in einer Datenbank, falls Probleme auftreten sollten. Für den Prototyp ist daher primär die Struktur auf dem Dateisystem ausschlaggebend. Um die Datenbank zu befüllen, wird eine Methode zur Synchronisierung bereitgestellt, welche die Ordnerstruktur in die Datenbank überführt. In den Tabelle **classifierstats** und **labels** werden Informationen zu trainierten Modellen, wie die Anzahl der Personen, wie viele Bilder insgesamt erzeugt wurden, die Genauigkeit und Weitere gespeichert. Wie das Training abläuft und diese Werte ermittelt werden, wird im weiteren Verlauf des Kapitels näher erläutert. Um dem Benutzer Informationen über durchgeführte Klassifizierungen verfügbar zu machen, werden diese in den Tabellen **classificationresults** und **result** gespeichert. Erstere enthält dabei Verweise auf das Bild welches klassifiziert wurde, sowie den genutzten Klassifizierer. Letztere beinhalten für jedes erkannte Gesicht in einem Bild einen Verweis auf die entsprechende Galerie, die Wahrscheinlichkeit, sowie die Bildkoordinaten für ein Rechteck, in dem das Gesicht gefunden wurde. Diese Informationen können von einer Oberfläche (unabhängig von der Plattform) zur Anzeige der erkannten Gesichter in einem Bild genutzt werden.

Wie bereits beschrieben, liegen die Bilder der Personen in einer Ordnerstruktur vor. Dabei gibt es zwei vom Prototyp vorgeschriebene Ordner, welche nicht gelöscht oder umbenannt werden dürfen. Das ist zum einen der Ordner für neue, noch nicht einer Person zugeordneten Bilder. Zum anderen gibt es einen festen Ordner für Personen welche explizit als unbekannt eingeordnet sind. Die Benutzung dieses Ordners ist optional, da eine Klassifizierung als unbekannt auch anderweitig möglich ist (siehe unten). Er wird vom Prototyp allerdings erwartet und muss daher vorhanden sein. Die Ordnerstruktur ergibt sich wie folgt:

- images
  - new
  - persons
    - unknown
    - person1
    - person2
    - ...
    - personX

Alle Bilder, bis auf die nicht zugeordneten, bilden den Trainingsdatensatz. In Kapitel 3.6 hat sich gezeigt, dass die Augmentierung, das Vervielfachen der einzelnen Bilder, eine wichtige Voraussetzung für eine hohe Genauigkeit der Gesichtserkennung ist. Daher

werden die Bilder während des Trainings zuerst diesem Schritt unterzogen. Da die Unterschiede zwischen echt verschiedenen Bildern der einzelnen Personen zu bevorzugen ist, soll nur eine bestimmte Anzahl an Augmentierungen durchgeführt werden. Dafür wird eine Zahl als Ziel definiert, die für jede Person erreicht werden soll. Die Differenz zwischen dieser Anzahl und den bereits vorhandenen Aufnahmen ist die Menge der zu erzeugenden Bilder. Diese Zahl an zu erreichenden Bildern kann in der Konfiguration des Prototyps geändert werden.

Nach dieser dynamischen Augmentierung wird auf jedem Bild eine Gesichtsdetektion durchgeführt. Wird in einem Bild kein Gesicht gefunden, wird es verworfen. Bei allen anderen Bildern werden die gefundenen Bildbereiche in Dlib-Deskriptoren überführt (siehe Kapitel 3.1.6). Diese dienen im Anschluss als Trainingsdaten für den Klassifizierer. Als Klassifizierer wird aufgrund der in Kapitel 3 ermittelten Ergebnisse eine SVM mit den für die Tests verwendeten Parametern verwendet. Zusätzlich unterstützt der Prototyp die Verwendung des kNN-Verfahrens.

Während des Trainings wird, wie auch in der Testreihe, eine 5-Fach Kreuzvalidierung durchgeführt, aus der sich die Genauigkeit ergibt. Für zusätzliche Informationen wird sowohl eine Konfusionsmatrix als auch eine Lernkurve erstellt. Die Konfusionsmatrix stellt dar, wie häufig jede Klasse mit einer anderen verwechselt wurde. Eine Lernkurve zeigt an, wie gut die Genauigkeiten mit bestimmten Datenmengen sind. Beide Grafiken können dem Benutzer Aufschluss darüber geben, wie gut der Prototyp funktioniert, ob Konfigurationsparameter angepasst oder mehr Bilder benötigt werden. Zum Schluss wird das Modell serialisiert und mit einem einzigartigen Namen in einem Ordner abgelegt. Alle ermittelten Informationen werden in der Datenbank gespeichert. Da das Modell statt mit Namen der Personen mit Zahlen arbeitet, wird eine Zuordnung dieser zu den entsprechenden Namen erzeugt und ebenfalls in der Datenbank abgespeichert.

Weil der Trainingsprozess einige Zeit benötigt, muss dieser im Hintergrund ablaufen, um die Webanwendung nicht zu blockieren. Dafür wird das Framework Celery verwendet [113]. Dieses stellt ein asynchrones System zur Bearbeitung von beliebigen Aufgaben zur Verfügung. Der Trainingsvorgang stellt die zu bearbeitende Aufgabe dar. Die Bearbeitung der Aufgabe läuft in einem eigenen Prozess und stellt Statusaktualisierungen während des Trainings bereit. Diese Statusmeldungen können über die REST-Schnittstelle abgerufen und in der Applikation angezeigt werden. Während des Trainings kann ein bereits trainiertes Modell weiterhin vom Prototyp zur Klassifizierung genutzt werden. Ist das Training abgeschlossen, wird stattdessen das neue Modell geladen und verwendet.

Neben der Klassifizierung einzelner Personen soll der Prototyp in Bildern erkannte Gesichter auch als unbekannt einstufen können. Dies wird über die Wahrscheinlichkeiten realisiert, die bei einer Klassifizierung vom Modell errechnet werden. Diese betragen in der Summe über sämtliche bekannte Klassen 1,0. Die Person mit der höchsten Wahrscheinlichkeit wird als Ergebnis der Klassifizierung angesehen. Beträgt die maximale Wahrscheinlichkeit weniger als ein konfigurierbarer Grenzwert, wird die abgebildete Person als unbekannt eingestuft. Wurden dem Modell im Training Bilder in der Galerie für unbekannte Personen gezeigt, kann es in diesem Fall direkt eine Zuordnung als unbekannt durchführen. Eine Standardkonfiguration ist dem Prototyp beigefügt, sowie im Anhang A.1 aufgelistet.

Um dem Benutzer Klassifizierungen zu zeigen, sobald diese abgeschlossen sind, wird SocketIO verwendet [114]. Über das WebSocket-Protokoll, oder alternativ sogenanntes Long-Polling, wird ein durchgehender Kommunikationskanal zwischen einem Server und mehreren Clients ermöglicht. Ruft ein Benutzer in der Webanwendung die Live-Ansicht auf, wird automatisch eine Verbindung zum Server hergestellt. Der Browser des Benutzers stellt den Client dar. Erhält dieser vom Server eine Nachricht, dass ein neues Bild klassifiziert wurde, wird es an dieser Stelle mit weiteren Informationen dargestellt. Da die Webanwendung komplett über eine REST-API gesteuert werden kann, erfolgt auch das Hinzufügen neuer Bilder über einen URL-Aufruf. Erhält der Prototyp darüber ein Bild, wird es zunächst sowohl auf dem Dateisystem, als auch in der Datenbank gespeichert und der Galerie für noch nicht zugeordnete Bilder hinzugefügt. Im Anschluss wird eine Gesichtsdetektion durchgeführt. Ist diese erfolgreich, wird ein Deskriptor von jedem gefundenem Gesicht erzeugt und mithilfe des derzeit geladenen Modells klassifiziert. Die Bildbereiche in denen Gesichter gefunden wurden, werden im Bild markiert und mit dem Namen der wahrscheinlichsten Person annotiert. Im Anschluss wird das Bild inklusive aller Klassifizierungsergebnisse an alle über SocketIO verbundenen Clients gesendet und dargestellt. Eine Auflistung sämtlicher API-Aufrufe befindet sich im Anhang A.2.

Um den Prototyp zu testen, wurde er mit dem Labordatensatz (siehe Kapitel 3.2.2) trainiert. Im Verlauf von mehreren Tagen wurden sowohl Personen, welche bereits im Labordatensatz vorhanden sind, als auch fremde Personen gebeten, die Kamera zu aktivieren. Dabei traf der Prototyp in sämtlichen beobachteten Fällen die korrekte Entscheidung. Dies stimmt auch mit den errechneten Genauigkeiten der Modelle überein, welche in den getesteten Fällen immer ~99,9 % erreichen. Es konnten auch nicht mit dem System vertraute Personen dieses bedienen und neue Galerien anlegen, Trainingsvorgänge starten, Informationen abrufen und durchgeführte Klassifizierungen korrigieren. Zuvor unbekannte Personen konnten nach der Zuordnung von im Durchschnitt fünf Bildern

zuverlässig erkannt werden. Verwechslungen von Personen traten während der Testphase nicht auf. Lediglich bekannte Personen wurden als unbekannt eingestuft, wenn diese Grimassen zogen oder Accessoires trugen, welche in den Trainingsdaten nicht vorhanden sind. Umfassendere Tests bezüglich der Benutzbarkeit für technische Laien und in verschiedenen Umgebungen sind aus zeitlichen Gründen im Rahmen dieser Arbeit nicht möglich. Diese sollten aber in der Zukunft durchgeführt werden. Die Ergebnisse der Umsetzung und Tests zeigen, dass der Prototyp die in Tabelle 3 gestellten Anforderungen erfüllt. Abschließend ergibt sich, dass eine Gesichtserkennung mit vergleichsweise geringen Ressourcen und unabhängig von externen Diensten im Internet umsetzbar und benutzbar ist.

5. Zusammenfassung und Ausblick

Verfahren zur Gesichtserkennung werden aktiv erforscht und auch bereits praktisch eingesetzt. Häufig setzt dies allerdings große Datenmengen und Rechenleistung voraus. Beides ist in einem Heimautomatisierungsumfeld selten vorhanden. Aus diesem Grund wurden in dieser Arbeit mehrere Verfahren zur Gesichtserkennung betrachtet und miteinander verglichen. Diese erzeugen in einem ersten Schritt einen Deskriptor, welcher das Gesicht in einem im Vergleich zum Ursprungsbild niedrigdimensionalen Raum repräsentiert. Für den ersten Schritt wurden zum einen seit Jahren erprobte Methoden wie Eigenfaces, Fisherfaces und LBPs verwendet. Zum anderen wurden auch neuere Ansätze, basierend auf zur Bildverarbeitung spezialisierten neuronalen Netzen, in den Vergleich einbezogen. Diese stammen aus den Projekten Openface und Dlib, welche speziell zur Unterscheidung von Gesichtern trainiert wurden. Des Weiteren wurde ein auf Bildern des ImageNet-Wettbewerbs [69] trainiertes neuronales Netz auf Basis des VGG19-Modells genutzt [96].

Im zweiten Schritt wurden zur Klassifizierung SVMs, kNN und ein voll vernetztes neuronales Netz verwendet. Jede Kombinationsmöglichkeit wurde als ein einzelnes Verfahren betrachtet. Da sowohl die Extraktion der Merkmale eines Gesichts sowie die Klassifizierung von einem neuronalen Netz auf einmal durchgeführt werden kann, wurden auch hier zwei Ansätze in den Vergleich mit aufgenommen. Ein vergleichsweise kleines CNN, welches vollständig trainiert werden muss und ein ebenfalls auf dem VGG19-Modell basierendem Netz, welches auf bereits im Netz vorhandene Informationen aufbaut. Der Vergleich wurde auf zwei verschiedenen Datensätzen durchgeführt, welche unterschiedliche Bildqualitäten, Beleuchtungsbedingungen, Bilder pro Person und Personen insgesamt aufweisen. Zudem wurden die Daten künstlich vervielfacht. Diese sogenannte Augmentierung der Daten diente dazu, die geringe Datenmenge auszugleichen. In der Auswertung wurden sowohl die Genauigkeit, als auch die Dauer des Trainings und der Klassifizierung untersucht. Dabei zeigte sich, dass die Verwendung des Dlib-Deskriptors die beste Variante darstellt. Grundlage für dieses Verfahren ist ein bereits trainiertes neuronales Netz, welches für Gesichter der gleichen Person möglichst identische Zahlenvektoren erzeugt. Diese Vektoren dienen als Trainingsgrundlage für einen Klassifizierer. Selbst bei geringen Datenmengen werden somit hohe Erkennungsraten von ~99,9 % erreicht. Als Klassifizierer sind sowohl SVMs als auch kNN geeignet, wobei erstere wesentlich weniger Zeit bei der Klassifizierung benötigen. Aus einem Bildausschnitt mit Gesicht kann innerhalb von ~50 ms ein Zahlenvektor erzeugt und die Klassifizierung durchgeführt werden. Auf Basis dieser Erkenntnisse wurde im weiteren Verlauf der Arbeit ein Prototyp entwickelt, der zeigen sollte, dass eine praktische Verwendung im Heimautomatisierungsumfeld möglich ist. Dieser Prototyp besteht aus

einem Webserver, welcher eine REST-API zur Verfügung stellt. Diese ermöglicht eine Integration in viele bestehende Systeme. Alle Funktionen des Prototyps sind über diese API steuerbar.

Der Prototyp verwendet Erkenntnisse aus dem Verfahrensvergleich und nutzt diese für eine dynamische Augmentierung. Dadurch kann auch mit geringer Anzahl vorhandener Bilder schnell eine hohe Genauigkeit im praktischen Einsatz erreicht werden. Neben der Erkennung bekannter Personen, können Personen auch als unbekannt eingestuft werden. Neben der API wird eine Schnittstelle bereitgestellt, auf der sich Systeme als Clients registrieren können. Darüber erhalten diese Informationen über durchgeführte Klassifizierungen, sobald das System neue Bilder erhält. Auf dieser Basis wurde eine Webanwendung erstellt, welche einem Benutzer sämtliche Funktionalitäten des Prototyps bereitstellt und Informationen bezüglich der Genauigkeit des zugrunde liegenden Modells anzeigt. Dieser Prototyp wurde unter Alltagsbedingungen in einem Laborraum der Hochschule Bonn-Rhein-Sieg erfolgreich getestet und zeigt, dass eine Gesichtserkennung in einem Heimautomatisierungsumfeld möglich ist. Für die Zukunft ergeben sich einige Fragestellungen, welche sowohl die Verfahren zur Gesichtserkennung, als auch die Weiterentwicklung des Prototypen betreffen.

Die Vergleiche der einzelnen Verfahren werden größtenteils hinsichtlich ihrer Abhängigkeit zu den jeweiligen Gesichtserkennungsverfahren und der Menge an Trainingsdaten verglichen. In der Auswertung zeigte sich ansatzweise, dass zusätzlich auch die Anzahl der Klassen, sowie die durchschnittliche Bilderanzahl pro Klasse, Auswirkungen auf die Genauigkeit der Verfahren haben. Dabei erzielte der Labordatensatz mit weniger Personen und mehr Bildern pro Person bessere Ergebnisse. Hier wären genauere Untersuchungen interessant. Erkenntnisse aus diesen könnten vor allem für die dynamische Augmentierung der Bilder im Prototyp genutzt werden.

Des Weiteren besteht die Fragestellung, ob sich die Nutzung einer bestimmten Anzahl augmentierter Bilder im Vergleich zur gleichen Anzahl tatsächlich unterschiedlicher Bilder auf die Genauigkeit auswirkt. Sollte dies der Fall sein, muss ermittelt werden, wie stark diese Auswirkung ist.

Vor der Testphase war vorgesehen, unterschiedliche Vorverarbeitungsschritte durchzuführen. Da diese die Genauigkeit jedoch beeinträchtigen, wurden bis auf die Gesichtsdetektion und Größenanpassungen keine weiteren Schritte durchgeführt. Es wäre zu klären, warum die Genauigkeit der einzelnen Verfahren durch die Vorverarbeitungen geringer wurde. Grund könnten falsche Parameter sein, welche zudem für einzelne Verfahren unterschiedlich sein können. Zudem könnten für manche Gesichtserkennungsverfahren nur einige oder auch nur einer der Schritte sinnvoll sein.

Der Prototyp erhält seine Bilder im Labor von einer Kamera mit normaler Linse. Überwachungskameras und Türspione verfügen jedoch häufig über Fischaugenobjektive, um ein möglichst großes Sichtfeld zu bieten. Dadurch werden die Bilder deutlich verzerrt. Das neuronale Netz, welches die Deskriptoren im Prototyp erzeugt wurde mit normalen Bildern trainiert, weshalb die Verzerrungen zu Problemen führen könnten. Ob und wie stark sich die Verzerrungen durch Fischenaugenobjektive auswirken, sollte näher betrachtet werden. Bei starken Auswirkungen könnte eine automatische Entzerrung durchgeführt werden.

Der Prototyp wurde bisher lediglich im Labor in relativ kurzen zeitlichen Abständen getestet. Es müssen umfangreichere Tests in verschiedenen Szenarien durchgeführt werden. Diese umfassen die Anbringung einer Kamera außerhalb von geschlossenen Räumen mit unterschiedlichen Wetter- und Beleuchtungsbedingungen.

Die Gesichtsdetektion erfolgte bisher mit einem bereits trainiertem HOG-Modell. Dieser Ansatz weißt gute Detektionsraten auf, solange die Gesichter der Personen frontal und aufrecht der Kamera zugewandt sind. Drehungen oder die Augen verdeckende Accessoires bereiten Probleme und führen zu nicht erkannten Gesichtern. Hier sollte nach robusteren Ansätzen gesucht werden. Neben der Gesichtserkennung werden auch für dieses Problem bereits erfolgreich neuronale Netze eingesetzt. Eine höhere Detektionsrate könnte dabei allerdings auf Kosten der Geschwindigkeit gehen, mit der ein Bild untersucht wird. Die Detektion sollte auch ohne dedizierte Grafikkarte in nahezu Echtzeit funktionieren.

Viele mögliche Verbesserungen beziehen sich auf die Funktionalität des Prototyps. Bisher ist dieser in der Lage, Personen zuverlässig zu erkennen und, wenn nötig, als unbekannt einzustufen. Wurden mehrere Bilder einer Person zugeordnet, muss das Training bisher jedoch manuell gestartet werden. Hier wäre eine Mechanik wünschenswert, welche in bestimmten Zeitabständen einen neuen Trainingsvorgang startet, sofern neue Daten vorhanden sind. Zudem kann neben dem Grenzwert, ab dem eine Person als unbekannt gilt, ein weiterer Grenzwert eingeführt werden. Ist eine Person mit einer Wahrscheinlichkeit höher als dieser Wert erkannt, wird sie automatisch der entsprechenden Galerie zugeordnet. Dadurch wird der Lernvorgang unabhängiger von menschlicher Interaktion.

Durch den eben beschriebenen Mechanismus können für manche Personen schnell mehr Bilder vorhanden sein, als vom Zielwert für die dynamische Augmentierung vorgegeben sind. Ab diesem Punkt muss die Entscheidung getroffen werden, sämtliche Bilder zum Training zu nutzen, oder nur eine zufällig ermittelte Teilmenge mit der Anzahl des Zielwerts. Für die zweite Variante ist ein automatisches Training vorstellbar, welches erneut durchgeführt wird, wenn eine bestimmte Teilmenge der Bilder ein schlechteres Ergebnis

als vorher erzielt hat. Auf Basis dieses Vorgehens sind noch viele weitere Auswahlmöglichkeiten für die zu verwendeten Bilder vorstellbar.

Aufgrund der Verwendung einiger neuronaler Netze auf Basis von Tensorflow und Keras in dieser Arbeit, basiert die Implementierung der Augmentierung auf den dafür bereitgestellten Methoden von letzterem Framework. Der Prototyp selber nutzt zwar die Augmentierung, ist aber unabhängig von den genannten Frameworks. Wenn möglich sollten die Augmentierungsfunktionen daher direkt im Prototyp implementiert werden. Alternativ kann auch ein auf diesen Anwendungszweck spezialisiertes Framework genutzt werden.

Eine wichtige Arbeit für die Zukunft ist die vollständige Umwandlung des Prototyps in eine fertige Anwendung. Neben der Benutzbarkeit sollte die Sicherheit betrachtet werden, da mit den zugeordneten Bildern sensible Daten vorliegen. Sämtliche Anforderungen für Webanwendungen fallen ebenfalls an. Dazu zählt auch eine simple Installationsroutine die möglichst wenig technische Kenntnisse erfordert. Zusätzlich sollte auch die Erstellung eines Docker-Containers in betrachtet gezogen werden.

Neben einer Webbasierten Anwendung sind auch native Anwendungen auf mobilen Geräten vorstellbar. In Verbindung mit **Augmented Reality (AR)**-Brillen kann eine Gesichtserkennung für Personen hilfreich sein, welche alters- oder krankheitsbedingte Schwierigkeiten mit der Erkennung bekannter Personen haben.

Diese und weitere Fragestellungen können untersucht werden, um die Verwendung von Gesichtserkennungsverfahren in der Heimautomatisierung weiter zu vereinfachen. Um als Basis dafür zu dienen, sind sämtliche in dieser Arbeit ermittelten Daten, Ergebnisse und Quelltexte frei verfügbar.

Literaturverzeichnis

[1] R. Jafri and H. R. Arabnia, "A Survey of Face Recognition Techniques," *Journal of Information Processing Systems*, vol. 5, no. 2, pp. 41–68, 2009.

[2] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78 – 87, 2012.

[3] S. J. Russell and P. Norvig, *Artificial intelligence : a modern approach*, 3rd ed. Pearson, 2010.

[4] T. Hastie, R. Tibshirani, and J. H. J. H. Friedman, *The elements of statistical learning : data mining, inference, and prediction*. Springer-Verlag New York Inc, 2009.

[5] S. S. Haykin, *Neural networks and learning machines*, 3rd ed. Prentice Hall/-Pearson, 2009.

[6] W. Ertel, *Grundkurs Künstliche Intelligenz*. Wiesbaden: Springer Fachmedien Wiesbaden, 2013.

[7] E. Bauer and R. Kohavi, "An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants," *Machine Learning*, vol. 36, no. 1/2, pp. 105–139, 1999.

[8] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[9] M. Banko and E. Brill, "Scaling to very very large corpora for natural language disambiguation," in *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics - ACL '01*. Morristown, NJ, USA: Association for Computational Linguistics, 2001, pp. 26–33.

[10] F. Samaria and A. Harter, "Parameterisation of a stochastic model for human face identification," in *Proceedings of 1994 IEEE Workshop on Applications of Computer Vision*. IEEE Comput. Soc. Press, 1994, pp. 138–142.

[11] T. Sim, S. Baker, and M. Bsat, "The CMU Pose, Illumination, and Expression (PIE) Database," in *Proceedings of the Fifth IEEE International Conference on Automatic Face and Gesture Recognition*. IEEE Computer Society, 2002, p. 436.

[12] P. J. Phillips, H. Wechsler, J. Huang, and P. J. Rauss, "The FERET database and evaluation procedure for face-recognition algorithms," *Image and Vision Computing*, vol. 16, pp. 295–306, 1998.

[13] R. Gross, "Face Databases," in *Handbook of Face Recognition*. Springer-Verlag, 2005, pp. 301–327.

[14] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," University of Massachusetts, Amherst, Tech. Rep. 07-49, 2007.

[15] E. Learned-Miller, G. B. Huang, A. RoyChowdhury, H. Li, G. Hua, and G. B. Huang, "Labeled Faces in the Wild: A Survey," in *Advances in Face Detection and Facial Image Analysis*. Springer International Publishing, 2016, pp. 189–248.

[16] Institut für Sicherheitsforschung. (2015) H-BRS Haut-, Gesichts- und Fälschungs-Datenbank. Accessed 01.03.2017. [Online]. Available: http://isf.h-brs.de/skin-db/

[17] H. Steiner, A. Kolb, and N. Jung, "Reliable face anti-spoofing using multispectral SWIR imaging," in *2016 International Conference on Biometrics (ICB)*. IEEE, 2016, pp. 1–8.

[18] H. Steiner, A. Kolb, N. Jung, and S. Sporrer, "Design of an Active Multispectral SWIR Camera System for Skin Detection and Face Verification," *Journal of Sensors*, vol. 2016, pp. 1–16, 2016.

[19] H. Lamba, A. Sarkar, M. Vatsa, R. Singh, and A. Noore, "Face recognition for look-alikes: A preliminary study," in *2011 International Joint Conference on Biometrics (IJCB)*. IEEE, 2011, pp. 1–6.

[20] Shiguang Shan, Wen Gao, Bo Cao, and Debin Zhao, "Illumination normalization for robust face recognition against varying lighting conditions," in *2003 IEEE International SOI Conference. Proceedings (Cat. No.03CH37443)*. IEEE Comput. Soc, 2003, pp. 157–164.

[21] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, "Face recognition: A literature survey," *ACM Computing Surveys (CSUR)*, vol. 35, no. 4, pp. 399–458, 2003.

[22] C. Ding and D. Tao, "A Comprehensive Survey on Pose-Invariant Face Recognition," *CoRR*, vol. abs/1502.0, 2015.

[23] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," in *British Machine Vision Conference*, 2014.

[24] B. Leng, K. Yu, and Q. Jingyan, "Data augmentation for unbalanced face recognition training sets," *Neurocomputing*, vol. 235, pp. 10–14, 2017.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[26] J.-J. Lv, X.-H. Shao, J.-S. Huang, X.-D. Zhou, and X. Zhou, "Data augmentation for face recognition," *Neurocomputing*, 2016.

[27] J.-J. Lv, C. Cheng, G.-D. Tian, X.-D. Zhou, and X. Zhou, "Landmark perturbation-based data augmentation for unconstrained face recognition," *Signal Processing: Image Communication*, vol. 47, pp. 465–475, 2016.

[28] M.-H. Yang, D. J. Kriegman, and N. Ahuja, "Detecting Faces in Images: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 1, pp. 34–58, 2002.

[29] G. Yang and T. S. Huang, "Human face detection in a complex background," *Pattern Recognition*, vol. 27, no. 1, pp. 53–63, 1994.

[30] C. Zhang and Z. Zhang, "A Survey of Recent Advances in Face Detection," Microsoft Research, Tech. Rep., 2010. [Online]. Available: https://www.microsoft.com/en-us/research/publication/a-survey-of-recent-advances-in-face-detection/

[31] S. Zafeiriou, C. Zhang, and Z. Zhang, "A survey on face detection in the wild: past, present and future," *Computer Vision and Image Understanding*, vol. 138, pp. 1–24, 2015.

[32] P. Viola and M. J. Jones, "Robust Real-Time Face Detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[33] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1. IEEE, 2005, pp. 886–893.

[34] W. T. Freeman and M. Roth, "Orientation histograms for hand gesture recognition," in *International workshop on automatic face and gesture recognition*, vol. 12, 1995, pp. 296–301.

[35] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object Detection with Discriminatively Trained Part-Based Models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[36] P. Campadelli, R. Lanzarotti, and G. Lipori, "Automatic Facial Feature Extraction for Face Recognition," in *Face Recognition*. I-Tech Education and Publishing, 2007, ch. 3.

[37] L. Wiskott, J.-M. Fellous, N. Kuiger, and C. von der Malsburg, "Face recognition by elastic bunch graph matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 775–779, 1997.

[38] I. Cox, J. Ghosn, and P. Yianilos, "Feature-based face recognition using mixture-distance," in *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 10. IEEE Comput. Soc. Press, 1996, pp. 209–216.

[39] S. Lawrence, C. Giles, Ah Chung Tsoi, and A. Back, "Face recognition: a convolutional neural-network approach," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 98–113, 1997.

[40] D. Y. Tsao and M. S. Livingstone, "Mechanisms of face perception," *Annu. Rev. Neurosci.*, vol. 31, pp. 411–437, 2008.

[41] M. Turk and A. Pentland, "Face recognition using eigenfaces," in *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*. IEEE, 1991, pp. 586–591.

[42] ——, "Eigenfaces for Recognition," *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–86, 1991.

[43] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. Fisherfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 711–720, 1997.

[44] W. Zhao, R. Chellappa, and P. J. Phillips, "Subspace Linear Discriminant Analysis for Face Recognition," Tech. Rep., 1999.

[45] X.-Y. Jing, H.-S. Wong, and D. Zhang, "Face recognition based on 2D Fisherface approach," *Graph-based Representations*, vol. 39, no. 4, pp. 707–710, 2006.

[46] J. Li, S. Zhou, and C. Shekhar, "A comparison of subspace analysis for face recognition," in *2003 International Conference on Multimedia and Expo. ICME '03. Proceedings (Cat. No.03TH8698)*. IEEE, 2003, pp. III–121.

[47] A. M. Martinez and A. C. Kak, "PCA versus LDA," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2001.

[48] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987, 2002.

[49] T. Ahonen, A. Hadid, and M. Pietikäinen, "Face recognition with local binary patterns," *Computer vision-eccv 2004*, pp. 469–481, 2004.

[50] H. Li, S. Wang, and F. Qi, "Automatic Face Recognition by Support Vector Machines," in *Combinatorial Image Analysis*. Springer, Berlin, Heidelberg, 2004, pp. 716–725.

[51] N. Cristianini and J. Shawe-Taylor, *An introduction to support vector machines : and other kernel-based learning methods*. Cambridge University Press, 2000.

[52] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.

[53] G. Guo, S. Li, and K. Chan, "Face recognition by support vector machines," in *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)*. IEEE Comput. Soc, 2000, pp. 196–201.

[54] K. Jonsson, J. Matas, J. Kittler, and Y. P. Li, "Learning Support Vectors for Face Verification and Recognition," in *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)*. IEEE Comput. Soc, 2000, pp. 208–213.

[55] W. S. Mcculloch and W. Pitts, "A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY," *BULLETIN OF MATHEMATICAL BIOPHYSICS*, vol. 5, 1943.

[56] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. [Online]. Available: http://neuralnetworksanddeeplearning.com/

[57] F. Rosenblatt, "PRINCIPLES OF NEURODYNAMICS. PERCEPTRONS AND THE THEORY OF BRAIN MECHANISMS." DTIC Document, 1961.

[58] M. Minsky and S. Papert, *Perceptrons; an introduction to computational geometry*. MIT Press, 1969.

[59] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[60] D. Kriesel, *Ein kleiner Überblick über Neuronale Netze*, 2007. [Online]. Available: http://www.dkriesel.com/science/neural_networks

[61] K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *2009 IEEE 12th International Conference on Computer Vision*. IEEE, 2009, pp. 2146–2153.

[62] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[63] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *Proceedings of the 27 th International Conference on Machine Learning*, 2010.

[64] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 9, pp. 249–256, 2010.

[65] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[66] D. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex." *The Journal of physiology*, vol. 160, no. 1, pp. 106–54, 1962.

[67] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[68] Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng, "Building high-level features using large scale unsupervised learning," in *International Conference in Machine Learning*, 2012.

[69] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[70] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," *CoRR*, vol. abs/1409.4, 2014.

[71] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[72] C. Garcia and M. Delakis, "Convolutional face finder: a neural architecture for fast and robust face detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 11, pp. 1408–1423, 2004.

[73] J. H. Saito, T. V. de Carvalho, M. Hirakuri, A. Saunite, A. N. Ide, and S. Abib, "Using CMU PIE Human Face Database to a Convolutional Neural Network - Neocognitron," in *ESANN*, 2005, pp. 491–496.

[74] G. Hu, Y. Yang, D. Yi, J. Kittler, W. Christmas, S. Z. Li, and T. Hospedales, "When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2015, pp. 142–150.

[75] O. M. Parkhi, A. Vedaldi, A. Zisserman *et al.*, "Deep face recognition," in *BMVC*, vol. 1, no. 3, 2015, p. 6.

[76] B. Amos, B. Ludwiczuk, and M. Satyanarayanan, "OpenFace: A general-purpose face recognition library with mobile applications," CMU-CS-16-118, CMU School of Computer Science, Tech. Rep., 2016.

[77] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2014, pp. 1701–1708.

[78] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 815–823.

[79] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[80] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[81] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[82] P. Wagner. (2017) Face recognition algorithms for matlab/gnu octave and python. Accessed 22.08.2017. [Online]. Available: https://github.com/bytefish/facerec

[83] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[84] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, Y. Bengio, A. Bergeron, J. Bergstra, V. Bisson, J. Bleecher Snyder, N. Bouchard, N. Boulanger-Lewandowski, X. Bouthillier, A. de Brébisson, O. Breuleux, P.-L. Carrier, K. Cho, J. Chorowski, P. Christiano, T. Cooijmans, M.-A. Côté, M. Côté, A. Courville, Y. N. Dauphin, O. Delalleau, J. Demouth, G. Desjardins, S. Dieleman, L. Dinh, M. Ducoffe, V. Dumoulin, S. Ebrahimi Kahou, D. Erhan, Z. Fan, O. Firat, M. Germain, X. Glorot, I. Goodfellow, M. Graham, C. Gulcehre, P. Hamel, I. Harlouchet, J.-P. Heng, B. Hidasi, S. Honari, A. Jain, S. Jean, K. Jia, M. Korobov, V. Kulkarni, A. Lamb, P. Lamblin, E. Larsen, C. Laurent, S. Lee, S. Lefrancois, S. Lemieux, N. Léonard, Z. Lin, J. A. Livezey, C. Lorenz, J. Lowin, Q. Ma, P.-A. Manzagol, O. Mastropietro, R. T. McGibbon, R. Memisevic, B. van Merriënboer, V. Michalski, M. Mirza, A. Orlandi, C. Pal, R. Pascanu, M. Pezeshki, C. Raffel, D. Renshaw, M. Rocklin, A. Romero, M. Roth, P. Sadowski, J. Salvatier, F. Savard, J. Schlüter, J. Schulman, G. Schwartz, I. V. Serban, D. Serdyuk, S. Shabanian, E. Simon, S. Spieckermann, S. R. Subramanyam, J. Sygnowski, J. Tanguay, G. van Tulder, J. Turian, S. Urban, P. Vincent, F. Visin, H. de Vries, D. Warde-Farley, D. J. Webb, M. Willson, K. Xu, L. Xue, L. Yao, S. Zhang, and Y. Zhang, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, 2016.

[85] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: http://tensorflow.org/

[86] F. Chollet *et al.* (2015) Keras. https://github.com/fchollet/keras. Accessed 22.08.2017.

[87] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[88] A. Clark. (2017) Pillow, python imaging library (fork). Accessed 22.08.2017. [Online]. Available: https://github.com/python-pillow/Pillow

[89] International Telecommunication Union, "Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios ," ITU, Tech. Rep., 2011. [Online]. Available: https://www.itu.int/rec/R-REC-BT.601-7-201103-I/en

[90] V. Klema and A. Laub, "The singular value decomposition: Its computation and some applications," *IEEE Transactions on Automatic Control*, vol. 25, no. 2, pp. 164–176, 1980.

[91] Y. Tian, T. Tan, Y. Wang, and Y. Fang, "Do singular values contain adequate information for face recognition?" *Pattern Recognition*, vol. 36, no. 3, pp. 649–655, 2003.

[92] L. P. Coelho, "Mahotas: Open source software for scriptable computer vision," *Journal of Open Research Software*, vol. 1, 2012.

[93] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[94] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 1867–1874.

[95] F. Chollet. (2016) Building powerful image classification models using very little data. Accessed 22.08.2017. [Online]. Available: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

[96] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, vol. abs/1409.1, 2014.

[97] I. J. Goodfellow, "Piecewise Linear Multilayer Perceptrons and Dropout," *CoRR*, vol. abs/1301.5, 2013.

[98] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[99] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012.

[100] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3320–3328.

[101] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: an astounding baseline for recognition," *CoRR*, vol. abs/1403.6382, 2014.

[102] H. Linka, "Entwicklung einer intelligenten, selbstversorgenden türüberwachung mit hilfe von modernen single-board computern," Bachelorarbeit, Hochschule Bonn-Rhein-Sieg, Sankt Augustin, 2015.

[103] I. H. Witten, E. Frank, and M. A. Hall, *Data mining : practical machine learning tools and techniques*. Morgan Kaufmann, 2011.

[104] R. Kohavi, “A study of cross-validation and bootstrap for accuracy estimation and model selection,” in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI’95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 1137–1143.

[105] J. Nickolls, I. Buck, M. Garland, and K. Skadron, “Scalable parallel programming with cuda,” *Queue*, vol. 6, no. 2, pp. 40–53, 2008.

[106] C. Kirsch. (2017) Skripte zum Vergleich von Gesichtserkennungsverfahren. Accessed 24.08.2017. [Online]. Available: https://github.com/kircon/facerec_comparison

[107] R Core Team. (2017) R: A language and environment for statistical computing. R Foundation for Statistical Computing. Vienna, Austria. Accessed 22.08.2017. [Online]. Available: https://www.R-project.org/

[108] H. Wickham, *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. [Online]. Available: http://ggplot2.org

[109] C. Kirsch. (2017) Heimdall - Prototyp Webanwendung. Accessed 24.08.2017. [Online]. Available: https://github.com/kircon/heimdall

[110] A. Ronacher. (2017) Flask (A Python Micoframework). Accessed 01.08.2017. [Online]. Available: http://flask.pocoo.org/

[111] SQLAlchemy. (2017) SQLAlchemy The Python SQL Toolkit and Object Relational Mapper. Accessed 01.08.2017. [Online]. Available: https://www.sqlalchemy.org

[112] A. Ronacher. (2017) Flask-SQLAlchemy. Accessed 01.08.2017. [Online]. Available: http://flask-sqlalchemy.pocoo.org/2.1/

[113] Celery. (2017) Celery: Distributed Task Queue. Accessed 01.08.2017. [Online]. Available: http://www.celeryproject.org/

[114] SocketIO. (2017) Socket.IO. Accessed 01.08.2017. [Online]. Available: https://socket.io/

A. Anhang

A.1. Prototyp Standardkonfiguration

```
1  # -*- coding: utf-8 -*-
2  import os
3
4  BASEDIR = os.path.abspath(os.path.dirname(__file__))
5
6  SERVER_NAME = '<ip/hostname>:<port>'
7
8  CAMERA_SOCKET_HOST = '<host>'
9  CAMERA_SOCKET_PORT = 8001
10
11 SQLALCHEMY_DATABASE_URI = 'postgresql://<db_user>:<password>@<host
       >/<db_name>'
12 SQLALCHEMY_TRACK_MODIFICATIONS = False
13
14 CELERY_BROKER_URL = 'redis://<host>:<port>/0'
15 CELERY_RESULT_BACKEND = 'redis://<host>:<port>/0'
16 CELERYD_MAX_TASKS_PER_CHILD = 1
17
18 IMAGE_FOLDER = 'images'
19 NEW_IMAGES_FOLDER = os.path.join(IMAGE_FOLDER, 'new')
20 SUBJECT_IMAGES_FOLDER = os.path.join(IMAGE_FOLDER, 'subjects')
21 PLOTS_FOLDER = os.path.join(IMAGE_FOLDER, 'model_plots')
22 UNKNOWN_IMAGES_FOLDER = os.path.join(SUBJECT_IMAGES_FOLDER, '
       unknown')
23
24 ML_MODEL_PATH = os.path.join(BASEDIR, 'ml_models')
25 IMAGE_BASE_PATH = os.path.join(BASEDIR, IMAGE_FOLDER)
26 PLOTS_BASE_PATH = os.path.join(BASEDIR, PLOTS_FOLDER)
27 NEW_IMAGES_PATH = os.path.join(IMAGE_BASE_PATH, 'new')
28 SUBJECTS_BASE_PATH = os.path.join(IMAGE_BASE_PATH, 'subjects')
29 UNKNOWN_IMAGES_PATH = os.path.join(SUBJECTS_BASE_PATH, 'unknown')
30
31 NUM_TARGET_IMAGES = 50
32 PROBABILITY_THRESHOLD = 0.4
33
34 DLIB_SHAPE_PREDICTOR_MODEL = 'shape_predictor_68_face_landmarks.dat
       '
35 DLIB_FACE_RECOGNITION_MODEL = '
       dlib_face_recognition_resnet_model_v1.dat'
36
37 DLIB_SHAPE_PREDICTOR_MODEL_URL = 'http://dlib.net/files/
       shape_predictor_68_face_landmarks.dat.bz2'
38 DLIB_FACE_RECOGNITION_MODEL_URL =
39 'http://dlib.net/files/dlib_face_recognition_resnet_model_v1.dat.
       bz2'
40
41 DLIB_SHAPE_PREDICTOR_PATH = os.path.join(ML_MODEL_PATH,
       DLIB_SHAPE_PREDICTOR_MODEL)
42 DLIB_FACE_RECOGNITION_MODEL_PATH = os.path.join(ML_MODEL_PATH,
       DLIB_FACE_RECOGNITION_MODEL)
```

A.2. REST-API

Tabelle 1: REST-API

URL	Request-Methode	URL-Parameter	Parameter	Beschreibung
/api/resync/	GET	n.a.	n.a.	Leert die Datenbank und legt sie entsprechend der Struktur auf dem Dateisystem neu an
/api/gallery/	POST	n.a.	name	Legt eine Galerie an
/api/gallery/[id]	GET	Galerie-ID	n.a.	Liefert Details zur angegebenen Galerie
	DELETE	Galerie-ID	n.a.	Löscht die angegebene Galerie
/api/gallery/[id]/images	GET	Galerie-ID	n.a.	Liefert Details zu den Bildern in der Galerie
/api/gallery/[id]/clear	POST	Galerie-ID	n.a.	Löscht alle Bilder aus der Galerie
/api/galleries/	GET	n.a.	n.a.	Liefert Details zu allen vorhanden Galerien
/api/images/	GET	n.a.	n.a.	Liefert Details zu allen vorhandenen Bildern
	PUT	n.a.	gallery_id image_ids	Verschiebt Bilder in eine andere Gallerie
/api/classifier/	GET	n.a.	n.a.	Liefert Details zu den gespeicherten Modellen
/api/classifier/[id]	GET	Modell-ID	n.a.	Liefert Details zum angegebenen Modell
/api/classifier/load/	POST	n.a.	n.a.	Lädt das aktuellste Modell
/api/classifier/load/[id]	POST	Modell-ID	n.a.	Lädt das angegebene Modell
/api/classifier/delete/[id]	DELETE	Modell-ID	n.a.	Löscht das angegebene Modell
/api/recognizer/train/	GET	n.a.	n.a.	Startet einen Trainingsvorgang, liefert eine Trainings-ID
/api/recognizer/train/status/[id]	GET	Trainings-ID	n.a.	Liefert Details zum Trainingsvorgang
/api/recognizer/classify/[id]	GET	Bild-ID	n.a.	Klassifiziert das angegebene Bild
/api/live/	POST	n.a.	image	Nimmt ein Bild als Base64-String an, klassifiziert es und sendet das Ergebnis an die Live-Ansicht.
/api/tasks/	GET	n.a.	n.a.	Liefert Informationen zu aktuell laufenden Aufgaben/Trainingsvorgängen
/connect/	GET	n.a.	n.a.	Verbindet einen Client mit der Live-Ansicht
/disconnect/	GET	n.a.	n.a.	Trennt einen Client von der Live-Ansicht

A.3. Prototyp-Oberfläche

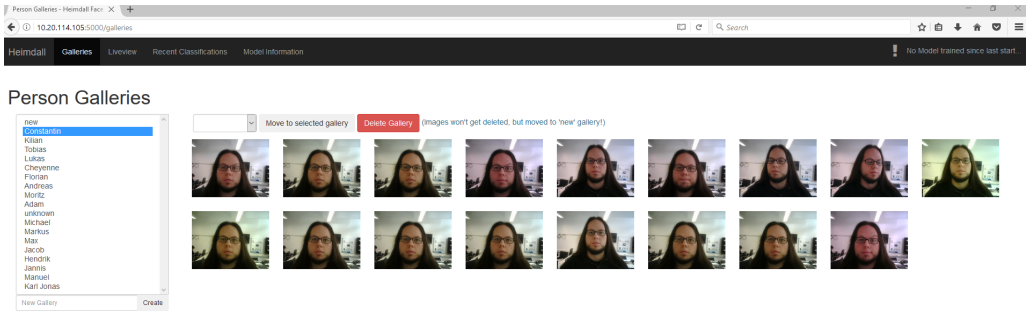


Abbildung 42: Screenshot der Übersicht aller Galerien.

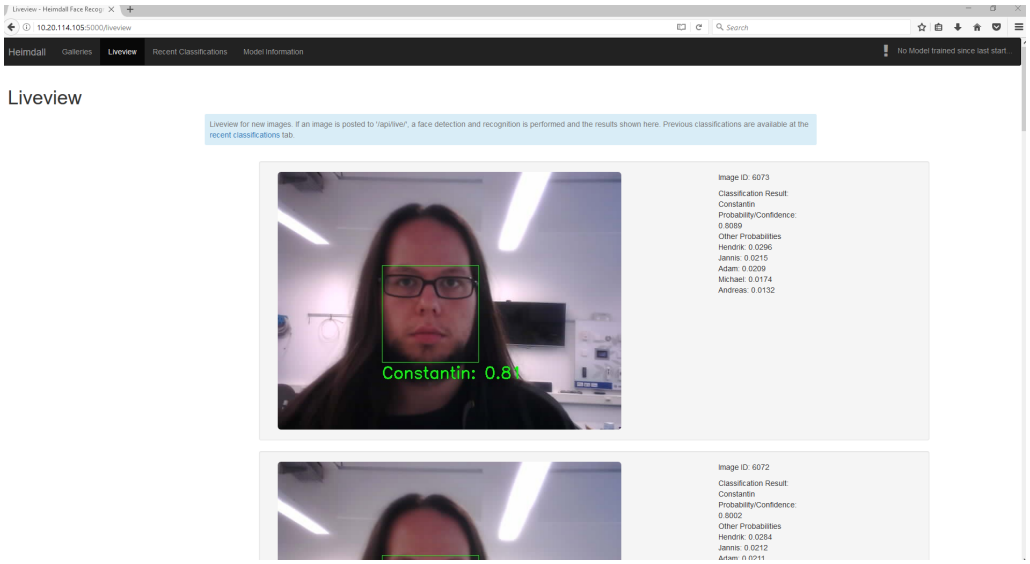


Abbildung 43: Screenshot der Live-Ansicht.

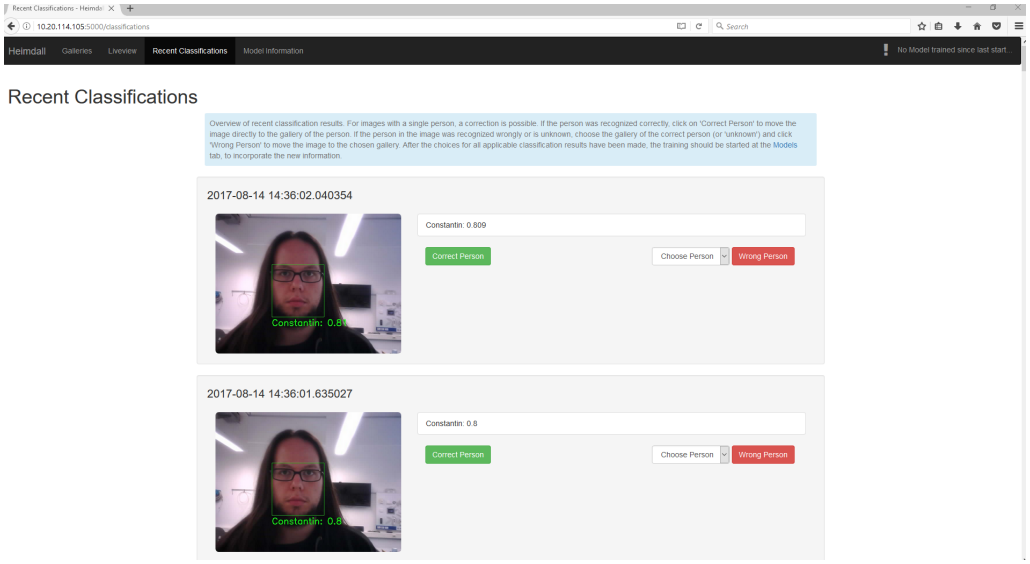


Abbildung 44: Screenshot der Übersicht von kürzlich vorgenommen Klassifizierungen.

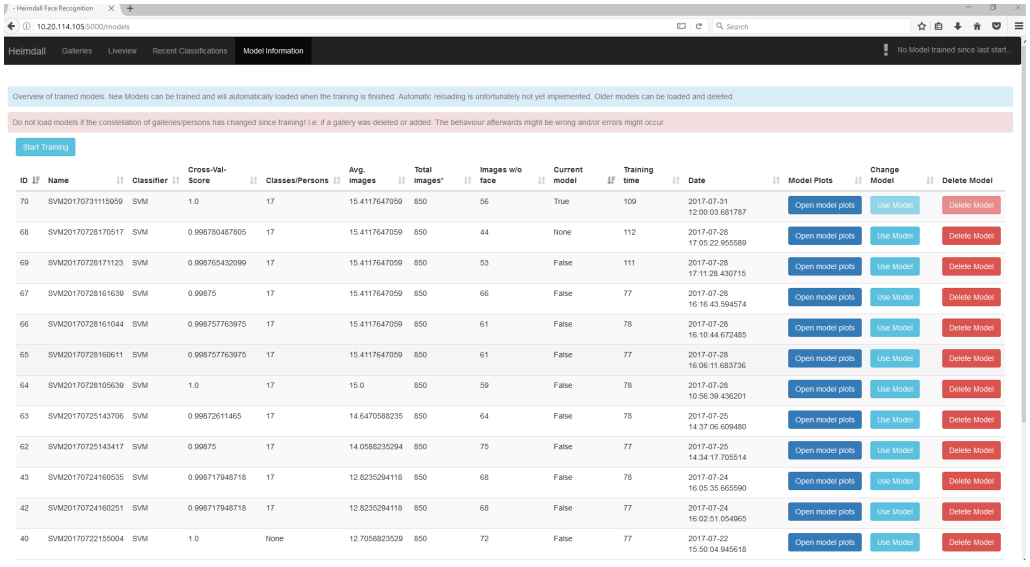


Abbildung 45: Screenshot der Übersicht der trainierten Modelle.