



Wilhelm Büchner Hochschule
Fachbereich Informatik

Bachelor Thesis im Studiengang Informatik

vorgelegt von:
Sven Kraus
Prischoßstraße 46
63755 Alzenau
Matrikelnummer: 893511

Betreuer: Herr Lachezar Krumov

Abgabetermin 02.08.2019

Inhaltsverzeichnis

1 Einführung	1
1.1 Eingrenzung	2
1.2 Ziel der Untersuchung	2
1.3 Aufbau der Arbeit	3
2 Theoretische Grundlagen	4
2.1 Künstliche neuronale Netze	4
2.1.1 Biologisches Vorbild	5
2.1.2 Formalisiertes Model	7
2.1.3 Aktivierungsfunktionen	8
2.1.4 Aufbau und Funktionsweise	9
2.1.5 Training	11
2.1.6 Faltende Neuronale Netze	16
2.2 Emotionserkennung	18
2.2.1 Einteilung von Emotionen	18
2.2.2 Gesichtserkennung	19
3 Umsetzung & Evaluierung	27
3.1 Beschaffenheit und Einteilung der Daten	27
3.2 Datensätze	28
3.2.1 Beschaffung der Datensätze	28
3.2.2 Einteilung der Datensätze	31
3.3 Datenpräparation	33
3.3.1 Vorverarbeitung	33
3.3.2 Normalisierung	34
3.3.3 Datenmehrung	35

Inhaltsverzeichnis

3.4	Entwurf und Entwicklung neuronaler Netze	36
3.4.1	Entwicklungsumgebung	36
3.4.2	Topologie	36
3.4.3	Hyperparamater	37
3.5	Evaluierung neuronaler Netze?	37
3.5.1	abschnitt 1	37
3.5.2	abschnitt 2	37
3.6	Entwicklung eines Webservice	37
3.6.1	Softwarearchitektur	38
4	Literaturverzeichnis	39

1 Einführung

Mithilfe überwachter Produkttests können zwar schon Heute wertvolle Informationen über den zu erwartenden Erfolg eines Produktes getroffen werden, jedoch werden die zur Verfügung gestellten Informationen nur ineffizient genutzt. Eine Erfassung der Emotionen, welche der Tester während des Interviews empfindet, könnte hier wichtige Erkenntnisse liefern. Dieser Ansatz ist nicht neu, in der Regel werden diese bereits vom Interviewer erfasst. Dieser ordnet subjektiv, aufgrund seiner Erfahrung und empathischen Fähigkeiten, das Verhalten seines Gegenübers bestimmten Emotionen zu. Im Sinne der Vergleichbarkeit, sowie der Effizienz ist es jedoch wünschenswert diese Analysen automatisiert durchzuführen. Die Idee des empathischen Computers fasziniert viele Forscher. Die Arbeiten auf diesem Gebiet erschließen das noch relativ junge Gebiet des Affective Computings, welches sich mit der Erkennung von Benutzergefühlen und anderen Stimuli, sowie adäquaten Reaktionen auf diese beschäftigt.(Schuller 2006)

Die maschinelle Erkennung von Emotionen ist jedoch aufgrund der Komplexität und der Individualität des menschlichen Verhaltens nicht wirklich trivial, da die Körpersprache bei vielen Menschen, zum Beispiel auch kulturell bedingt, unterschiedlich ausfallen kann. Aber auch innerhalb eines Kulturkreises erweist sich eine solide Klassifizierung von Emotionen als schwierig.

Um überhaupt eine solche Klassifizierung zu ermöglichen, müssen Emotionen zunächst in Klassen eingeteilt werden und entsprechende Merkmale gefunden werden. Im Bereich der Bilderkennung, bzw. Klassifizierung haben sich in den

1 Einführung

letzten Jahren künstliche neuronale Netze als führende Technologie durchgesetzt. Diese Arbeit untersucht das Problem der Emotionserkennung mithilfe dieser Netze, genauer gesagt mit gefalteten neuronalen Netze (engl.: convolutional neural network).

1.1 Eingrenzung

Die Erkennung von Emotionen kann mithilfe von Videodaten auf vielfältige Wege geschehen. So bietet ein Video in der Regel Ton-Daten und Bild-Daten. Aus den gewonnenen Ton-Daten können Emotionen anhand von spezifischen Stimmlagen erkannt werden. Des Weiteren kann das Gesprochene mit Sprache-zu-Text (engl. Speech-to-Text) Systemen in Text umgewandelt und kontextuell ausgewertet werden. Aus den Bild-Daten können Emotionen aus Gestik, sowie aus Mimik abgeleitet werden. Aufgrund des begrenzten Zeitraums wird lediglich Letzteres ausführlich in dieser Arbeit behandelt.

1.2 Ziel der Untersuchung

Das Ziel der Arbeit ist es, ein neuronales Netzwerk zu entwerfen und zu trainieren, welches Emotionen von Testern anhand der Mimik “erkennen” kann. Dieses soll von einem Webservice genutzt werden, welcher Videodaten von Probanden entgegen nimmt und eine Zeitleiste mit Emotionen zurück gibt. Der Service soll als Prototyp entwickelt werden.

Zur Realisierung eines solchen Service soll diese Arbeit eine Antwort auf folgende Fragen liefern:

- Wie kann man menschliche Emotionen sinnvoll klassifizieren?

1 Einführung

- Welche Modelle können für diesen Anwendungsfall genutzt und entsprechend spezialisiert werden?
- Woher bekommt man entsprechende Trainingsdaten?

1.3 Aufbau der Arbeit

Die Arbeit gliedert sich in drei Teilbereiche. Zuerst werden die, im Rahmen einer Literatur-Recherche gewonnenen, theoretischen Grundlagen zur Kategorisierung von Emotionen, sowie künstlichen neuronalen Netzen und der Klassifizierung mit deren Hilfe erläutert.

Im darauf Folgenden Teil wird auf Basis der Erkenntnisse des ersten Teils der Entwurf des künstlichen neuronalen Netzes, sowie die Schritte zur Datenbeschaffung und Vermehrung beschrieben. Ferner wird auch auf die gewählte Trainingsmethode, sowie die Implementierung des Webservice eingegangen.

Der dritte Teil befasst sich mit der Evaluierung des entworfenen Klassifizierers anhand einer tiefer gehenden Analyse . Ein Test des Webservices mit Videodaten von verschiedenen Personen wird durchgeführt um die ausreichende Generalisierung des neuronalen Netzes zu überprüfen.

2 Theoretische Grundlagen

2.1 Künstliche neuronale Netze

Künstliche neuronale Netze (KNN), auch kurz einfach neuronale Netze genannt, bezeichnen einen Ansatz zur Modellierung, welcher im Bereich der künstlichen Intelligenz, genauer im Bereich des maschinellen Lernens seinen Einsatz findet. Das Forschungsgebiet des maschinellen Lernens beschäftigt sich mit einer Klasse von Algorithmen, die anhand von Beispielfällen ein Modell erstellen, welches Inputdaten in Sätze aus Attributen und Eigenschaften kategorisiert (News 2018). Ein weiteres Teilgebiet des maschinellen Lernens stellt das tiefe Lernen, engl.: Deep Learning, dar. Abbildung 2.1 zeigt die Einordnung von neuronalen Netzen und Deep Learning in das Gebiet der künstlichen Intelligenz.

2 Theoretische Grundlagen

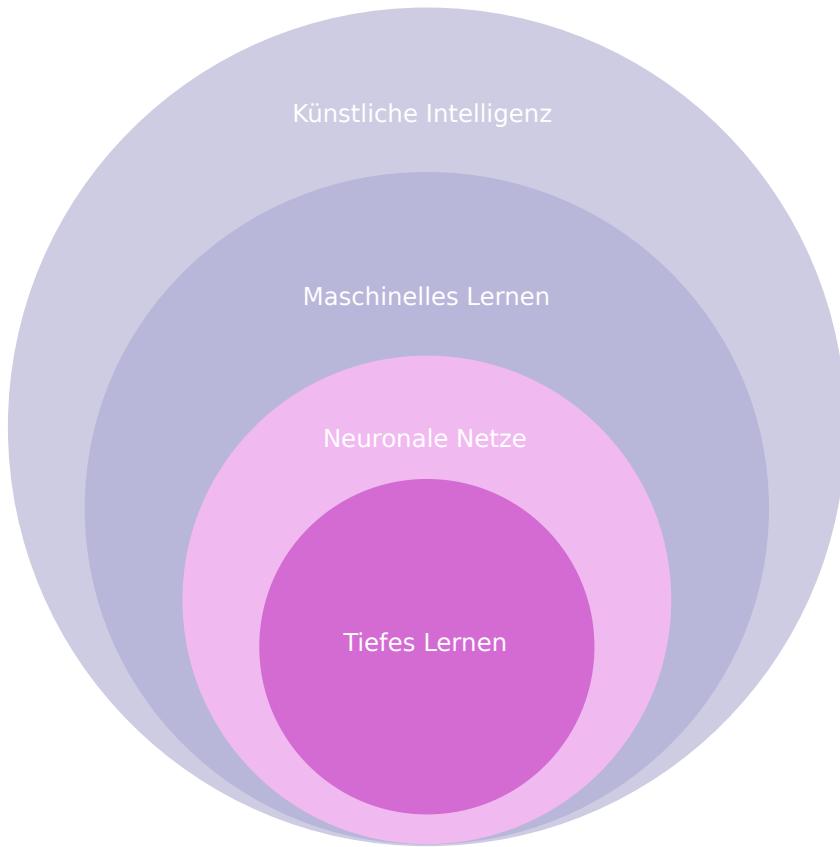


Abbildung 2.1: Einordnung neuronaler Netze in die künstliche Intelligenz.
Quelle: Angelehnt an (News 2018)

2.1.1 Biologisches Vorbild

Der strukturelle Aufbau, sowie die Arbeitsweise von neuronalen Netzen ist der Struktur und Funktionsweise eines Nervensystems, genauer gesagt des menschlichen Gehirns, nachempfunden. Daher wird im Folgenden kurz die Funktionsweise eines biologischen neuronalen Netzwerkes skizziert, wie es zum Beispiel in unserem Gehirn zu finden ist.

“Ein Neuron ist eine Zelle, die elektrische Aktivität sammelt und weiterleitet.” (Kruse et al. 2015) Ein stark vereinfachtes Modell eines Neurons ist in

2 Theoretische Grundlagen

Abbildung 2.2 zu sehen.

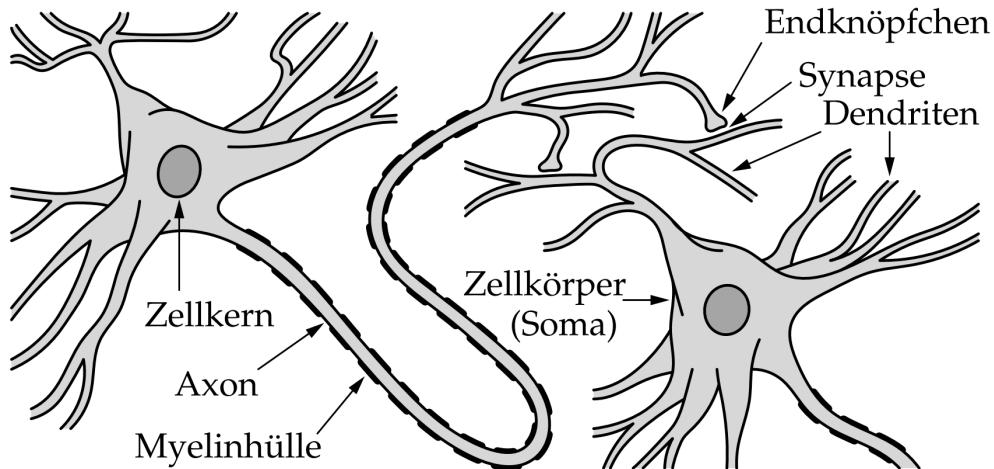


Abbildung 2.2: Darstellung eines biologischen Neurons. Quelle: (Kruse et al. 2015)

Hier sieht man den Zellkörper, der auch Soma genannt wird. Von ihm aus gehen mehrere Dendriten, sowie das Axon ab. Der Zellkörper ist in der Lage eine interne elektrische Spannung zu speichern. Dabei laden elektrische Signale, die über die Dendriten zum Soma transportiert werden, den Zellkörper auf. Ab einem gewissen Schwellwert jedoch entlädt sich dieser wieder über das Axon, welches mit Dendriten von anderen Neuronen über die Synapsen verbunden ist und lädt diese dadurch auf. So entsteht ein größeres Netzwerk aus Neuronen.

Die Verbindung zwischen Synapsen und Dendriten ist jedoch nicht perfekt leitend, da es einen "kleinen Spalt" zwischen ihnen gibt, welchen die Elektronen nicht ohne Weiteres überwinden können. Dieser ist mit chemischen Substanzen, den sogenannten Neurotransmittern, gefüllt. Diese können durch eine anliegende Spannung ionisiert werden, sodass sie eine Ladung über den Spalt transportieren. (Heinsohn et al. 2012) (News 2018)

2 Theoretische Grundlagen

Die Synapsen spielen also in diesem neuronalen Netz eine sehr wichtige Rolle. Sie können ihre Leitfähigkeit verändern, wodurch ein neuronales Netz mithilfe der Anpassung der Leitfähigkeit (der Gewichte) der einzelnen Verbindungen (Synapsen) zwischen den Neuronen lernfähig wird. Denn abhängig von der Leitfähigkeit der einzelnen Synapsen, verändert sich die Reaktion des Netzwerkes auf bestimmte Eingabeinformationen.

2.1.2 Formalisiertes Model

Aufgrund dieser vereinfacht beschriebenen Funktionsweise lag es zunächst nahe, ein Neuron formal als Schwellenwertelement zu modellieren. Bereits 1943 untersuchten McCulloch und Pitts ein solches Modell, weshalb man Schwellenwertelemente auch McCulloch-Pitts-Neuronen nennt. Oft werden Schwellenwertelemente auch als Perzepron bezeichnet, obwohl dieses von Rosenblatt entworfene Modell eigentlich noch etwas komplexer ist. Der Aufbau eines solchen künstlichen Neurons wird in Abbildung 2.3 gezeigt(Kruse et al. 2015).

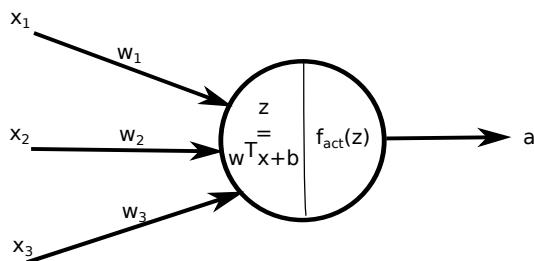


Abbildung 2.3: Darstellung eines Perzeprons - Angelehnt an (Kruse et al. 2015)

Den Ausgabewert a eines Neurons erhält man durch Anwendung der Aktivierungsfunktion f_{act} auf die interne Ladung des Neurons. Bei klassischen Schwellenwertelementen ist die Aktivierungsfunktion typischerweise die Sprungfunktion, welche 1 annimmt sobald die interne Ladung ϵ eines Neurons den definierten Schwellenwert θ überschreitet.

2 Theoretische Grundlagen

$$f_{\text{act}}(\varepsilon, \theta) = \begin{cases} 1, & \text{wenn } \varepsilon \geq \theta, \\ 0, & \text{sonst.} \end{cases}$$

Die interne Ladung ε erhält man, indem man eine gewichtete Summe der Eingabeparameter berechnet. Also das Skalarprodukt eines Eingabevektors \vec{x} mit den Eingabewerten x_1, \dots, x_n und dem Gewichtsvektor \vec{w} mit den jeweiligen Gewichten w_1, \dots, w_n . Zu diesem wird vor Anwendung der Aktivierungsfunktion noch ein sogenannter Bias b addiert.

$$\varepsilon = \sum_{i=1}^n w_i x_i + b$$

Der berechnete Ausgabewert a dient wiederum als ein Eingabewert x_i für eines oder mehrere weitere Neuronen im künstlichen neuronalen Netzwerk.

2.1.3 Aktivierungsfunktionen

Die Wahl der Aktivierungsfunktion spielt eine wichtige Rolle bei der Modellierung eines KNN's, denn sie bringt die Eingabewerte in Relation mit dem späteren Ausgabewert des Neurons. Die Aktivierungsfunktion soll eine nicht-lineare Komponente in das neuronale Netzwerk bringen, da es ansonsten ausschließlich möglich wäre, linear lösbar Probleme zu lösen (Gupta & Reddi 2013). Einige Beispiele für im Umfeld von maschinellen Lernen häufig verwendete Aktivierungsfunktionen sind in Abbildung 2.4 zu sehen.

2 Theoretische Grundlagen

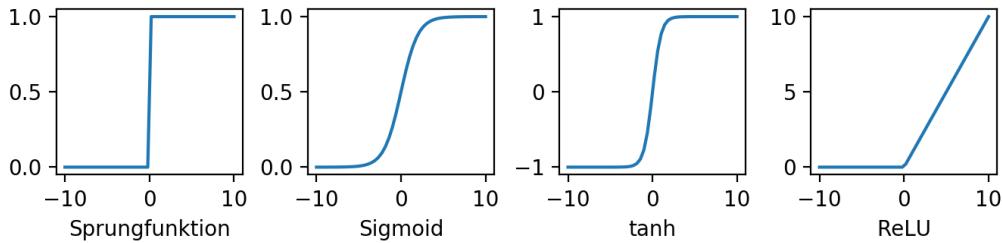


Abbildung 2.4: Beispiele für Aktivierungsfunktionen.

2.1.4 Aufbau und Funktionsweise

Bei der Betrachtung neuronaler Netze werden diese typischerweise als gerichtete Graphen dargestellt. Ein Graph besteht im Allgemeinen aus einem oder mehreren Knoten und Kanten. Die Kanten verbinden die einzelnen Knoten. Die Kanten eines Graphen können ungerichtet oder gerichtet sein. Bei einer ungerichteten Kante existiert eine Verbindung zwischen den Knoten in beide Richtungen, bei einem gerichteten jedoch immer nur in eine. Man spricht auch von einem gerichteten oder ungerichteten Graphen, wenn dieser nur gerichtete oder ungerichtete Kanten enthält (Goodfellow et al. 2016).

Bei der Darstellung eines neuronalen Netzes symbolisieren die Knoten die einzelnen Neuronen und die gerichteten Kanten die Synapsen bzw. die Verbindungen. Ein neuronales Netz wird in der Regel aufgeteilt in eine Eingabeschicht, sowie eine Ausgabeschicht (engl.: Input-/Output-Layer) und optional eine oder mehrere verdeckte Schichten (engl.: hidden Layer). Jede dieser Schichten (engl.: Layer) kann ein oder mehrere Neuronen enthalten. Man bezeichnet die Anzahl der Schichten als die Tiefe L des Netzwerkes, wobei die Eingabeschicht nicht berücksichtigt wird.

Abbildung 2.5 zeigt ein einfaches neuronales Netz mit 3 Eingängen, einem hidden Layer und einem Output Layer.

2 Theoretische Grundlagen

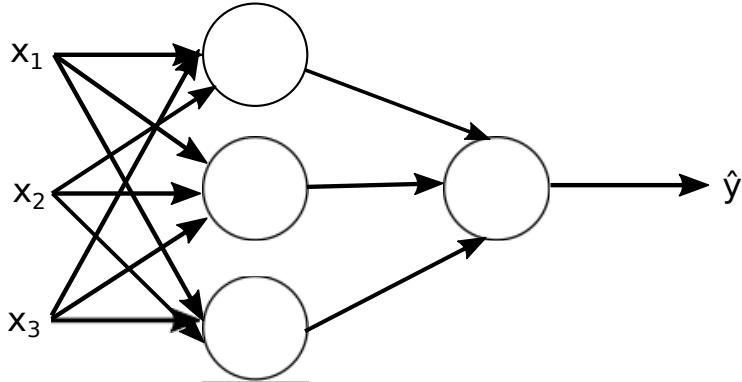


Abbildung 2.5: neuronales Netz mit 3 Eingängen, 1 versteckten Schicht und einem Ausgangsknoten - Angelehnt an (Ng 2017)

Anhand des Beispiels eines neuronalen Netzes zur Klassifizierung von Hundebildern soll im Folgenden die grundsätzliche Funktionsweise der einzelnen Schichten beschrieben werden. Zunächst nimmt die Eingabeschicht die benötigten Informationen von außen entgegen, zum Beispiel die numerisch dargestellten Pixel eines Hundebildes. Die Eingabedaten werden durch die versteckten Schichten geleitet und entsprechend verändert, bis sie zur Ausgabeschicht gelangen, welche nun ein Ergebnis anhand der Eingabewerte liefert. In unserem einfachen Beispiel zur Feststellung, ob es sich um ein Hundebild handelt oder nicht (binäre Klassifikation), würde eine Zahl zwischen 0 und 1 ausgegeben, welche der Wahrscheinlichkeit entspricht, dass das eingegebene Bild einen Hund darstellt.

Bei einer Klassifizierung mit mehr als 2 Klassen (z.B. Hund, Katze oder keines von beidem) entspricht das Ergebnis einem Ausgabevektor aus Wahrscheinlichkeiten für jede Klasse. Die Summe der ausgegebenen Wahrscheinlichkeiten entspricht stets 1. Letzteres Beispiel ist in Abbildung 2.6 vereinfacht dargestellt.

2 Theoretische Grundlagen

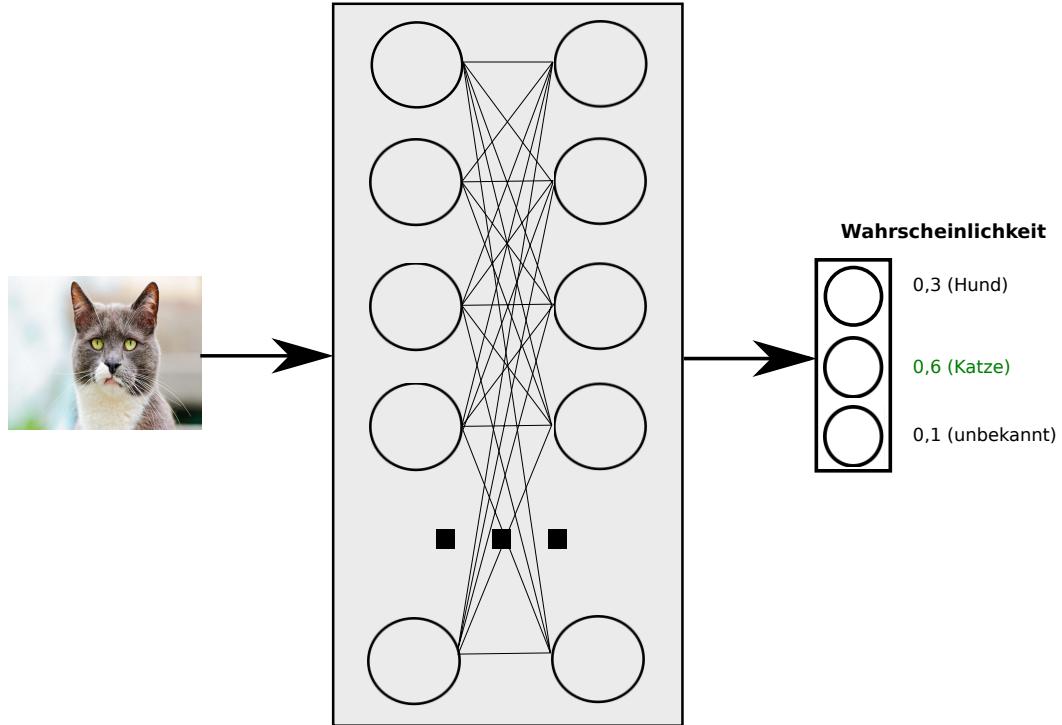


Abbildung 2.6: Hunde-/Katzen-Klassifizierer - Angelehnt an (Kirste & Schürholz 2018)

2.1.5 Training

Eine grundlegende Eigenschaft eines KNN ist, dass man es trainieren kann. Während der Trainingsphase lernt das neuronale Netz anhand von Eingabedaten passende Ausgabedaten zu liefern.

Das Wort lernen ist ein starker Begriff, da man leicht auf die Idee kommen könnte, die Maschine (oder das KNN) würde analog zum Menschen eine neue Fertigkeit, wie zum Beispiel Zeichnen oder das Verstehen einer fremden Sprache, erlernen. Bei der herkömmlichen Entwicklung von Programmen ist der Großteil des Programmverhaltens durch den Programmierer klar vorgegeben. Das bedeutet, der Entwickler setzt klare Regeln für die Lösung eines entsprechenden Problems.

2 Theoretische Grundlagen

Beim maschinellen Lernen dagegen verwendet man bestimmte Regeln zur Anpassung von Parametern anhand gegebener Daten (Gupta & Reddi 2013).

Genauer bedeutet das, dass je nach Art der verfügbaren Daten innerhalb eines Trainingsprozesses die einzelnen Gewichte und Biase des neuronalen Netzes anhand von bestimmten Regeln angepasst werden. Man unterscheidet bei den Trainings- bzw. Lernverfahren im Allgemeinen zwischen überwachtem, unüberwachtem und reinforcement learning.

2.1.5.1 Überwachtes Lernen (*supervised learning*):

Das überwachte Lernen ist die einfachste Trainingsmethode für neuronale Netze. Hierzu benötigt man einen Datensatz, in welchem sich sowohl die Eingangsdaten, als auch die dazu passende Ausgabe befindet. Hierbei werden in mehreren Durchläufen (Epochen) die Eingabewerte dem neuronalen Netzwerk präsentiert und der Netzwerkfehler berechnet. Der Trainingsprozess hat das Ziel den Netzwerkfehler mit jeder Epoche zu verringern und diesen dadurch zu minimieren. Der Netzwerkfehler kann mit Hilfe von unterschiedlichen Kostenfunktionen berechnet werden. Typische Beispiele für Probleme, welche mit supervised learning gelöst werden, sind Probleme der Regression oder Klassifizierung.

2.1.5.2 Unüberwachtes Lernen (*unsupervised learning*):

Das Gegenstück zum supervised learning ist das unsupervised learning. Hierbei werden dem Netzwerk in jedem Schritt Trainingsdaten gezeigt, ohne jedoch den Zielausgabewert zu kennen. Das Netz lernt in diesen Daten bestimmte Strukturen oder Muster zu erkennen.

Beispiele für Probleme des unüberwachten Lernens sind die Erkennung von

2 Theoretische Grundlagen

Ausreißern oder generative Modelle, welche neue Daten nach Art der Trainingsdaten generieren.

2.1.5.3 Reinforcement learning:

Beim reinforcement learning wird dem neuronalen Netz, ähnlich wie beim überwachten Lernen, während des Lernprozesses ein Feedback gegeben. Die Ausgabe eines Reinforcement learning Models wird als action bezeichnet. Das Label (Zielwert beim überwachten Lernen) für einen Eingabewert wird als reward bezeichnet. Das Netz erhält also vereinfacht gesagt für jede Eingabe eine Belohnung oder eine Bestrafung. Ein reward muss sich nicht immer direkt auf eine Eingabe beziehen, sondern kann sich auch auf mehrere Eingaben, oder eine Eingabe der Vergangenheit beziehen.

Typische Anwendungsfelder sind zum Beispiel das Spielen eines Spiels (z.B. Go) oder auch die Steuerung von Robotern, bei denen es kein richtiges Ergebnis im eigentlichen Sinne gibt, sondern nur Konsequenzen welche geringfügig mit bestimmten Aktionen in Verbindung stehen (Gupta & Reddi 2013).

Im weiteren Verlauf wird zunehmend auf überwachtes Lernen eingegangen, da die Methode auch in dieser Arbeit verwendet wird. Nach der Trainingsphase ist das neuronale Netzwerk im Idealfall in der Lage, anhand von ungewohnten Eingaben, das heißt solchen, welche nicht im Trainingsdatensatz vorhanden waren, den richtigen Ausgabewert zu ermitteln. Man nennt das die Generalisierungsfähigkeit des Netzes. Es kann passieren, dass während des Trainings eine Überanpassung (engl.: overfitting) an die Daten aus dem Trainingsdatensatz stattgefunden hat. Das bedeutet das Netzwerk kennt die Daten so gut, dass es diese perfekt zuordnen kann, kann jedoch keine brauchbaren Ergebnisse für neue Daten liefern.

Daher ist es ein weiteres Ziel der Trainingsphase auch ein overfitting zu ver-

2 Theoretische Grundlagen

hindern und somit eine gute Generalisierungsfähigkeit zu erhalten (Kruse et al. 2015).

Nach jeder Epoche des Trainings werden die Gewichte anhand einer sogenannten Lernregel angepasst. Im Folgenden wird auf einige bekannte Lernregeln kurz eingegangen.

2.1.5.4 Hebb-Regel

Die Hebb-Regel stellt eine der einfachsten Lernregeln dar. Sie weist eine große biologische Plausibilität auf und wurde 1949 vom Psychologen Donald Olding Hebb aufgestellt. Auf das Thema der neuronalen Netze bezogen, lässt sich die Regel wie folgt formulieren:

Das Gewicht zwischen zwei Knoten wird dann verändert, wenn beide Knoten gleichzeitig aktiv sind (Anon 2019).

Als Formel lässt sie sich wie folgt beschreiben:

$$\Delta w_{ij} = \alpha a_i a_j$$

Δw_{ij} beschreibt die Größe der Gewichtsanpassung zwischen den beiden Knoten n_i und n_j . Die Lernrate α stellt einen sogenannten Hyperparameter dar, der bereits vor dem Trainingprozess definiert wird und die gesamte Trainingsphase unverändert bleibt. Sie hat einen direkten Einfluss darauf, wie stark die Gewichte nach jeder Epoche angepasst werden. a_i und a_j stehen für das Aktivitätsniveau des empfangenden und des sendenden Knotens.

2 Theoretische Grundlagen

2.1.5.5 Delta-Regel

Die Delta Regel, auch Windroff-Hoff-Regel genannt, basiert auf dem Vergleich einer zu erwartenden und der tatsächlichen Ausgabe eines Knotens. Folgende Formel beschreibt die Funktionsweise der Delta-Regel.

$$\Delta w_{ij} = \alpha \delta_i a_j$$

Die Bedeutung von Δw_{ij} , α , sowie a_j in dieser Formel ist identisch zur Hebb-Regel. δ_i bezeichnet hier die Differenz zwischen dem erwarteten und dem tatsächlichen Aktivitätsniveau des sendenden Knotens n_i . Die folgende Formel veranschaulicht die Berechnung von δ_i .

$$\delta_i = a_i(\text{erwartet}) - a_i(\text{tatsächlich})$$

Wie man sieht, ist für die Ermittlung von δ_i , und somit für die Anwendung der Delta-Regel, die Kenntnis des zu erwartenden (korrekten) Ausgabewertes des Knotens n_i erforderlich.

Beim supervised learning liegt der korrekte Ausgabewert des gesamten Netzwerks vor. Das bedeutet die Delta-Regel ist ausschließlich für einschichtige neuronale Netze, also Netze ohne versteckte Schichten, einsetzbar, da nur hier die Ausgabe des Netzes direkt auf die Ausgabe der einzelnen Knoten zurückzuführen ist.

2.1.5.6 Backpropagation

Der Backpropagation Algorithmus soll dieses Problem lösen, sodass man die Grundidee der Delta-Regel auch auf tiefe neuronale Netze, also Netze mit mehreren versteckten Schichten anwenden kann. Damit eine Berechnung möglich

2 Theoretische Grundlagen

wird unterteilt der Algorithmus die Anpassung der Gewichte jeweils in die 3 Schritte forward pass, Fehlerermittlung und backward pass.

1. Forward pass Hier werden dem Netz entsprechende Eingabedaten aus dem Trainings-Datensatz präsentiert und von der Eingabe- bis hin zur Ausgabeschicht die Werte aller Knoten berechnet.
2. Fehlerermittlung Hier werden die Fehler der Ausgabeknoten ermittelt. Der Fehlerwert wird nun mit einem definierten Schwellwert verglichen. Ist der Fehler kleiner als der Schwellwert, oder die definierte Anzahl an Epochen bereits erreicht, wird der Algorithmus abgebrochen, falls nicht erfolgt der 3. Schritt.
3. Backward pass Dieser Schritt, stellt die innovative Neuerung des Algorithmus dar. Die zuvor ermittelten Fehler breiten sich jetzt von der Ausgabeschicht, bis hin zur Eingabeschicht rückwärts aus und die Gewichte der einzelnen Knoten werden entsprechend angepasst. Zur Bestimmung der Gewichtsanpassung kommt das Gradientenabstiegsverfahren zum Einsatz. Nach der Anpassung der Gewichte startet der Algorithmus mit der nächsten Trainingsepoke erneut mit dem Forward pass.

Auf eine genaue Beschreibung und mathematische Definition des Gradientenabstiegsverfahrens soll aufgrund der Komplexität in dieser Arbeit verzichtet werden. Typischerweise greifen heutige neuronale Netze auf den Backpropagation Algorithmus zurück (Neuronalesnetz.de n.d.)

2.1.6 Faltende Neuronale Netze

Bei faltenden neuronale Netzen (CNN, convolutional neural networks) handelt es sich um eine Sonderform von KNN, welche vor allem bei Daten verwendet werden, welche eine Raster-artige Struktur aufweisen. Beispiele dafür sind zum

2 Theoretische Grundlagen

Bilder, welche man sich als ein zweidimensionales Raster von Pixel-Werten vorstellen kann.

Ein typisches CNN besteht aus einer oder mehreren convolutional Schichten gefolgt von einer oder mehreren fully-connected Schichten, wie wir Sie bereits aus den klassischen neuronalen Netzen kennen.

Eine convolutional Schicht besteht aus einem oder mehreren Filtern gleicher Größe. Man kann diesen Filter als eine Art Fenster vorstellen, welches über die Daten „geschoben wird“. Dabei entstehen aus den meist größeren Rastern der Eingabedaten, neue Raster mit kleineren Dimensionen (siehe 2.7). Gibt es mehrere Filter, werden die entstehenden Ausgabeschichten aufeinander gestapelt (Goodfellow et al. 2016).

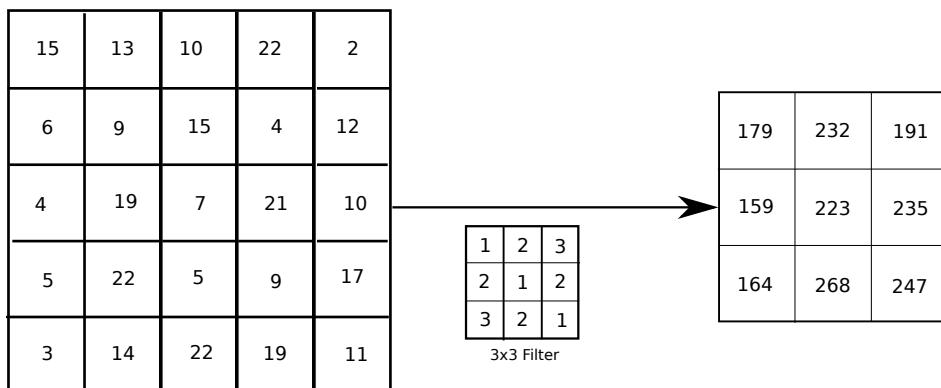


Abbildung 2.7: vereinfacht dargestelltes faltendes neuronales Netzwerk

Jede dieser convolutional Schichten hat mehrere (Hyper-)Paramenter, die beeinflussen welche Dimensionen das nachfolgende Daten-Raster erhält. Diese wären zum Beispiel die Filtergröße, sowie die Anzahl der Filter. Daneben ist die Schrittweite ein weitere Parameter, welche beeinflusst, wie groß die Sprünge sind, in welchen der Filter über die Daten „geschoben“ wird. Als letzter Parameter wäre noch ein mögliches padding (Füllung) zu nennen. Hierbei wird eine definierte Anzahl an zusätzlichen Zeilen an jeder Seite des Rasters mit Nullen aufgefüllt. Diese Methode dient dazu, dass der Filter auch die äußerer

2 Theoretische Grundlagen

Werte mit einer Ähnlichen Gewichtung berücksichtigen kann.

2.2 Emotionserkennung

Bevor man sich mit der automatisierten Erkennung von Emotionen befasst sollte zunächst die Frage nach der Definition des Begriffs Emotion geklärt werden. Eine einheitliche Definition ist im Bereich der Psychologie sehr umstritten. Im Allgemeinen beschreiben Emotionen jedoch subjektive Empfindungen kürzerer Zeiträume. In der Wissenschaft haben sich vier verschiedene Ansätze zur Entstehung dieser herauskristallisiert. So unterscheidet man zumeist zwischen dem evolutionstheoretischen, dem stimulativen, dem kognitiven und dem sozial konstruktiven Ansatz (Schuller 2006).

Nachfolgend wird der evolutionstheoretische Ansatz kurz beschrieben, da diese Arbeit sich vorwiegend auf diesen bezieht. Dieser stützt sich auf die Erkenntnisse von Charles Darwin (Charles Darwin et al. 1872), der die Ansicht vertrat, dass die Emotionen des Menschen ein Ergebnis der Evolution sind. Jede Emotion impliziert ein bestimmtes Verhalten, welches sich auf das Aussterben oder Überleben einer Art auswirkt.

2.2.1 Einteilung von Emotionen

Um eine Vorhersage, bzw. Erkennung der aktuellen Emotion zu bewerkstelligen ist es nötig diese sinnvoll zu unterscheiden. Für die Einteilung oder Kategorisierung von menschlichen Emotionen gibt es ebenfalls unterschiedliche Ansätze welche verfolgt werden können.

Im Allgemeinen unterscheidet man zwischen der kategorischen und der dimensionalen Einteilung. Bei letzterer werden die Emotionen auf einem Spektrum

2 Theoretische Grundlagen

dargestellt. Es wird also niemals eine konkrete Emotion zugeordnet, sondern immer ein Punkt auf der Skala. (POSNER et al. 2005)

Bei der kategorischen Einteilung geht man davon aus, dass es eine endliche Anzahl an wohl definierten menschlichen Emotionen gibt. Insbesondere Vertreter des evolutionstheoretischen Ansatzes von Emotionen gehen auch von einer kategorischen Einteilung aus. Zur Untermauerung des Darwinschen Ansatzes untersuchten einige Forscher unter der Leitung von Dr. Ekman (Ekman 2004) die Gesichtsausdrücke für bestimmte Situationen in einen Eingeborenen-Stamm in Neu Guinea. Dieser hatte zuvor vollkommen isoliert von anderen Gesellschaften gelebt. Somit waren die Reaktionen der Menschen dort nicht auf gesellschaftliche Einflüsse zurück zu führen. Ekman konnte damals bereits anhand des Gesichtsaudrucks vier universelle Emotionen ableiten. Diese waren Wut, Trauer, Ekel und Fröhlichkeit.

In weiterführenden Forschungen konnte Ekman die Erkenntnisse vertiefen und entwickelte zusammen mit einigen anderen Wissenschaftlern das Facial Acting Coding System (FACS) welche die menschlichen Emotionen in insgesamt sieben Basisemotionen einteilt, welche unabhängig vom gesellschaftlichen Einfluss vorhanden sind. Zusätzlich zu den vier zuvor abgeleiteten Emotionen beinhaltet das FACS noch die Emotionen Überraschung, Verachtung und Angst.

Die Einteilung nach dem FACS bildet die Basis für die Klassifizierung von Emotionen in dieser Arbeit.

2.2.2 Gesichtserkennung

Da sich das FACS auf Gesichtszüge bezieht und daher auch die Emotionserkennung in dieser Arbeit anhand von Gesichtsausdrücken stattfindet, ist es sinnvoll das zu erstellende neuronale Netz mit Bildern von Gesichtern als

2 Theoretische Grundlagen

Eingabedaten zu versorgen. Das heißt, dass von einem Bild im Idealfall immer nur der relevante Teil (das Gesicht) ausgeschnitten und dem neuronalen Netz präsentiert wird. Das gilt sowohl für die Trainingsphase, sowie für die spätere Anwendung.

Um aus einem größeren Bildausschnitt automatisiert den relevanten Teil, also den Gesichtsausschnitt, zu extrahieren ist es notwendig innerhalb des Bildes das Gesicht und dessen Rahmen (engl. bounding box) zu erkennen.

Die aktuell gängigste Methode zur Gesichtserkennung ist der sogenannte Viola-Jones Detektor(Shen et al. 1997). Diese Methode gibt nach Eingabe eines größeren Bildes, die genaue Position des Gesichtes in diesem wieder. Dieser Bereich kann dann ausgeschnitten und weiter verwendet werden. Viola und Jones nutzen in Ihrem Algorithmus 3 unterschiedliche Techniken, die ihn besonders effizient machen. Im Folgenden sollen diese kurz beschrieben werden.

2.2.2.1 integral Image

Zur weiteren einfachen Verarbeitung der Eingabebilder sollen nur bestimmte Merkmale der Bilder extrahiert werden. Anstelle eines pixelbasierten Ansatzes führten Viola und Jones hierzu den Begriff des Integralbildes (engl. integral image) ein. Die dazu extrahierten Features ähneln den Haar-Basis Merkmalen, wie Sie zuvor auch in anderen Arbeiten verwendet wurden (vergleich (Papageorgiou n.d.).

Die Wertigkeit eines Pixels im Integralbild steht immer im Zusammenhang mit den Pixeln in der unmittelbaren Umgebung. Desto höher der Wert ist, desto heller ist der betreffende Bildausschnitt. Auf Basis dieser Informationen haben Viola und Jones drei, aufgrund Ihrer Ähnlichkeit sogenannte, Haar-like features definiert, welche ein Gesicht anhand der entsprechenden Helligkeits-

2 Theoretische Grundlagen

unterschiede beschreiben.

“Ein two-rectangle feature wird beschrieben durch die Differenz aus der Summe der Pixel zwischen zwei rechteckigen Bildausschnitten. Die Ausschnitte haben die selbe Größe und Form und Grenzen entweder horizontal oder vertikal aneinander an. Ein three-rectangle feature wird berechnet durch die Differenz, der Summe aus zwei äußeren Rechtecken und der Summe eines zentrierten Rechteckes. Ein four-rectangle feature wird letzten Endes durch die Differenz zwischen Diagonalen Paaren von Rechtecken beschrieben.” (Shen et al. 1997)

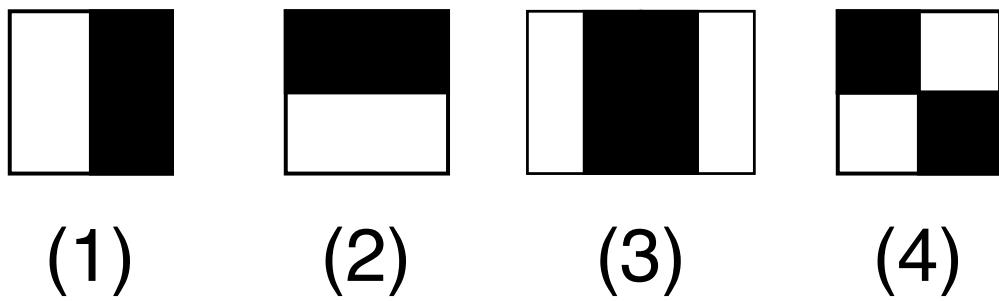


Abbildung 2.8: Haar-like features, two-rectangle (1, 2), three-rectangle (3), four-rectangle (4) - Quelle: https://commons.wikimedia.org/wiki/File:VJ_featureTypes.svg

Zur Berechnung des Integralbildes wird das Ursprungsbild zunächst in ein Graustufen-Bild umgewandelt. Dies ist ein übliches Vorgehen in der automatisierten Bildverarbeitung, da ein solches in der Regel noch alle benötigten Informationen beinhaltet, aber massiv weniger Daten beinhaltet (zB. 8 Bit pro Pixel statt 24 Bit pro Pixel). Ein Pixel des Integralbildes an der Stelle (x,y) errechnet sich nun aus der Summe aller Pixelwerte eines Rechtecks des Graustufen-Bildes vom Ursprung $(0,0)$ des Bildes bis zum besagten Punkt (x,y) (Shen et al. 1997). Die Formel zur Berechnung des Pixelwertes an der Stelle (x,y) Integralbildes i aus dem Ursprungs-Graustufenbild i lautet also

2 Theoretische Grundlagen

wie folgt.

$$ii(x,y) = \sum_{n=0,m=0}^{n \leq x, m \leq y} i(n,m)$$

In Abbildung 2.9 sieht man die Berechnung eines Pixelwertes anhand eines Beispiels. Man sieht das hier $ii(1,1) = 255 + 222 + 220 + 205 = 902$

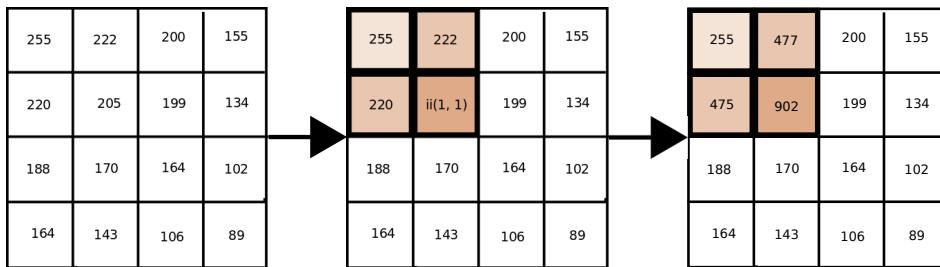


Abbildung 2.9: Berechnung eines Integralbildes aus einem Graustufenbild - Angelehnt an (Schmidt & A 2014)

Die Vorteile der Nutzung des Integralbildes sind die einfache, und damit effiziente, Berechnungsmethode, sowie die einfache Adaptierbarkeit des Prozesses auf einzelne Bildausschnitte. Des Weiteren ist es möglich, zur Berechnung von Pixelwerten auf zuvor bereits bestimmte Werte zurückzugreifen. Das ist in Abbildung 2.10 veranschaulicht. Hier sieht man, dass $ii(2,0) = i(0,0) + i(1,0) + i(2,0) = 255 + 222 + 200$ gleichzusetzen ist mit $ii(2,0) = ii(1,0) + i(2,0) = 277 + 200$.

2 Theoretische Grundlagen

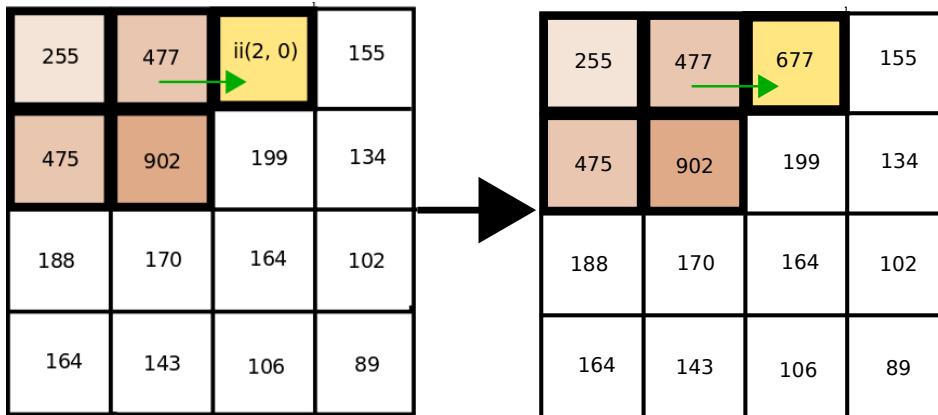


Abbildung 2.10: Berechnung der nächstfolgenden Pixelsumme des Integralbildes - Angelehnt an (Schmidt & A 2014)

2.2.2.2 modifizierter AdaBoost Algorithmus

In seiner ursprünglichen Form dient der AdaBoost Algorithmus dazu die Performance eines Klassifizierers im Bereich des maschinellen Lernens zu optimieren (engl. boost). Der Netzwerkfehler eines so optimierten Algorithmus (engl. boosted algirthm) konvergiert mit der Anzahl der Trainingsdurchläufe exponentiell gegen Null.

In Viola und Jones' Arbeit ergaben sich aus jedem 24x24 Pixel Fenster über 180000 rectangle-fatures. Zwar kann jedes einzelne dieser Merkmale sehr effizient berechnet werden, allerdings wäre es ziemlich aufwendig die gesamte Menge an Merkmalen zu berechnen. Sie stellten die Vermutung auf, dass bereits eine geringe Anzahl dieser Merkmale für einen effektiven Klassifizierer genutzt werden können. Viola und Jones nutzen AdaBoost daher an 2 Stellen. Zum einen zum Auswählen der zu nutzenden Merkmale und zum anderen zum eigentlichen Trainieren des Klassifizierers. Dazu wurde der weak classifier, so angepasst, dass er das Rechteck-Merkmal findet, welches die positiven Beispiel-Daten am besten von den negativen Beispiel-Daten separiert (Shen et al. 1997). In folgender Abbildung 2.11 ist der modifizierte AdaBoost Algo-

2 Theoretische Grundlagen

rithmus nach Viola und Jones kurz skizziert.

- Given example images $(x_1, y_1), \dots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \dots, T$:
 1. Normalize the weights,
$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^n w_{t,j}}$$

so that w_t is a probability distribution.

 2. For each feature, j , train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t , $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$.
 3. Choose the classifier, h_t , with the lowest error ϵ_t .
 4. Update the weights:

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

where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.

- The final strong classifier is:

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

$$\text{where } \alpha_t = \log \frac{1}{\beta_t}$$

Abbildung 2.11: modifizierter AdaBoost Algorithmus nach Viola und Jones.
Quelle: (Shen et al. 1997)

2 Theoretische Grundlagen

2.2.2.3 cascade of classifiers

Die dritte von Viola und Jones eingeführte Technik zur Gesichtserkennung sind die kaskadierenden Klassifizierer. Die Idee hierbei besteht darin mehrere optimierte Klassifizierer für nur ein bestimmtes Merkmal hintereinander zu schalten. Die einzelnen Klassifizierer haben dabei eine Negativ-Erkennungs-Rate (engl. false positive rate) gegen 0. So können diese eine große Anzahl an nicht passenden Bildausschnitten aussortieren und nur die passenden an den entsprechenden nächsten Klassifizierer weitergeben. Da diese einfachen Klassifizierungen sehr effizient berechnet werden können, wird verhindert, dass für nicht passende Ausschnitte zu viel Rechenaufwand betrieben wird. Der Begriff Kaskade wurde gewählt um zu veranschaulichen, dass jeder Klassifizierer nur anhand eines positiven Ergebnisses des vorgeschalteten Klassifizierers weiter arbeitet.

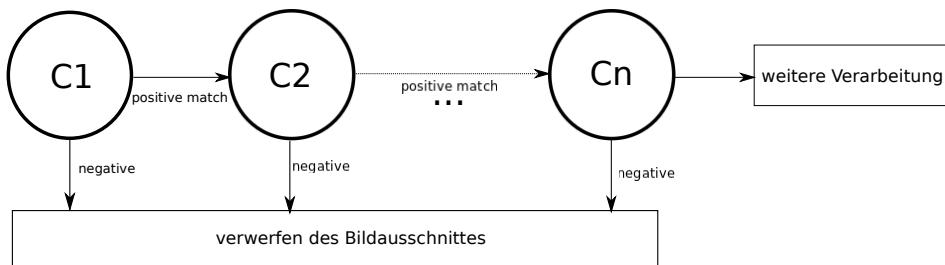


Abbildung 2.12: kaskadierende Klassifizierer C1 bis Cn - Angelehnt an (Shen et al. 1997)

3 Umsetzung & Evaluierung

In diesem Teil wird die Umsetzung der Arbeit eingegangen. im ersten Abschnitt wird auf die Art und Einteilung der zu klassifizierenden Daten eingegangen. Im Weiteren Verlauf wird die Beschaffung der verschiedenen Datensätze behandelt mit welchen gearbeitet werden soll. Anschließend werden auf diverse Verfahren der Datenpräparation wie Vorbereitung, Normalisierung und Datenvervielfältigung thematisiert. Letzten Endes werden verschiedene neuronale Netze entworfen, trainiert und getestet, bis dann im letzten Abschnitt die Realisierung des zu erstellenden Webservice näher betrachtet wird.

3.1 Beschaffenheit und Einteilung der Daten

Wie bereits beschrieben, sollen für den späteren Webservice Videodaten als Eingabe dienen. Das neuronale Netz wird jedoch nicht direkt die Videodaten, sondern aus dem Video extrahierte Einzelbilder als Eingabe erhalten. Diese Eingabedaten werden anhand später beschriebener Verfahren noch weiter vorbereitet und optimiert. Die Einzelbilder sollen als Array von Pixelwerten eines Graustufenbildes an das neuronale Netzwerk übergeben werden, siehe Kapitel Datenpräparation . Die Bilder sollen in 7 verschiedene Klassen. Die verschiedenen Emotionen bilden den Zielklassenvektor Z , es gilt also $|Z| = 7$. Jedes Bild welches der selben Emotion des FACS entspricht ist Mitglied der selben Klasse aus Z .

3.2 Datensätze

Ein Teil der Arbeit bestand darin geeignete Datensätze für das Training und die Evaluierung des neuronalen Netzes zu finden und diese später in entsprechende Trainings-, Evaluierungs- und Test-Datensätze für das neuronale Netzwerk zu unterteilen.

3.2.1 Beschaffung der Datensätze

Dazu sollen 2 verschiedene Ansätze unterschieden werden. Zum einen die Beschaffung eines vorhandenen freien Datensatzes von Gesichtsbildern inklusive der Zuordnung zu einer der entsprechenden FACS Emotionen, sowie zum anderen die Generierung von eigenen Daten mithilfe eines Webservice und freiwilligen Probanden. In dieser Arbeit wurden beide Ansätze in Kombination verwendet. Es wurden also 2 verschiedene Datenquellen herangezogen, was bei der Aufteilung der Datensätze noch eine wichtige Rolle spielt (siehe Kapitel Einteilung der Datensätze).

3.2.1.1 FER+

Als großer frei Verfügbarer Datensatz wurde der Facial Expression Recognition+ (FER+) (Barsoum et al. 2016) Datensatz verwendet. Bei den Eingangsdaten des FER+ handelt es sich um die selben Bilder, wie auch beim FER2013, welcher Teil der International Conference for Machine Learning (ICML) Challenge 2013 war, und danach der Öffentlichkeit zur Verfügung gestellt wurde. Bei FER+ wurden jedoch alle Label, mithilfe von Crowdsourcing neu erstellt, um eine bessere Datenqualität zu erreichen. (vgl. (Barsoum et al. 2016)). Der Datensatz besteht aus 34034 48x48 Graustufen Bilder von Gesichtern. Jedes dieser Bilder von je 10 Freiwilligen mithilfe von Crowdsourcing bewertet. Der

3 Umsetzung & Evaluierung

Datensatz enthält für jede Klasse (Emotionen des FACS und “kein Gesicht”) die Anzahl an Freiwilligen, welche das Bild entsprechend bewertet haben. Ein Beispiel für ein einzelnes Datum des Datensatzes ist in Abbildung ?? zu sehen

TODO: Abbildung FER+ Single row image

Das Team von Microsoft Research (Barsoum et al. 2016) beschreibt mehrere Variationen, wie die mehrfach gelabelten Daten verwendbar sind. In dieser Arbeit wird jedoch ausschließlich der einfache Mehrheits-Ansatz verfolgt. Es wird also jedes Bild der Klasse zugeordnet, welche die meisten Stimmen erhalten hat.

3.2.1.2 selbsterstellte Daten

Zum selbst erstellen von Daten wurde im Rahmen der Arbeit eine einfache Website erstellt, welche mithilfe von WebRTC Zugriff auf die Kamera bekommt. Auf dieser Website haben freiwillige Probanden, und auch der Autor dieser Arbeit, die Möglichkeit nacheinander für jede Emotion des FACS ein 15 sekündiges Video aufzunehmen. Dieses wird anschließend direkt auf dem Server gespeichert. Die Webseite ist in Abbildung 3.1 zu sehen.

DigitalLabs / Emotion Video Recorder

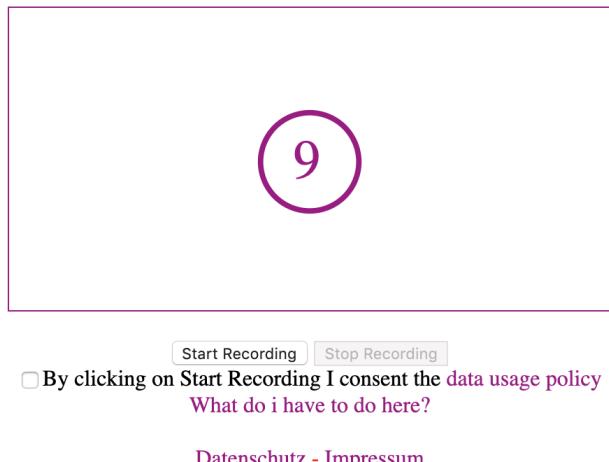


Abbildung 3.1: Screenshot der Video-Recording Website

Mithilfe dieser Webseite wurden insgesamt 15 Sätze an 15 sekündigen Videos von 8 verschiedenen freiwilligen Probanden (den Autor dieser Arbeit eingeschlossen) gesammelt. Aus diesen Videos wurde anschließend mit Hilfe des folgenden Python-Skripts pro Sekunde ein Einzelbild extrahiert, und mit dem Namen der entsprechenden Klasse abgespeichert. Somit wurden also $15 * 15 = 225$ Einzelbilder pro Klasse generiert.

```
def extract_video_frames(prefix, videofile,
    targetdir = "../data/extracted"):

    vidcap = cv2.VideoCapture(videofile)
    fps = min((50, int(vidcap.get((cv2.CAP_PROP_FPS)))))

    print ("extracting every {} frame from {}".format(fps,videofile))
    success,image = vidcap.read()
```

3 Umsetzung & Evaluierung

```
count = 0
while success:
    if (count % fps) == 0:
        # save frame as JPEG file
        cv2.imwrite("{}_frame{}.jpg".format(targetdir,
                                             prefix , count) , image)
    success,image = vidcap.read()
    print('Read a new frame: ', success)
    count += 1
```

Die Einzelbilder wurden anschließend manuell auf Korrektheit, das heißt Zuordnung zur Klasse, geprüft. Dabei wurden insgesamt 200 Bilder wieder aus sortiert (TODO: Verify numbers).

3.2.2 Einteilung der Datensätze

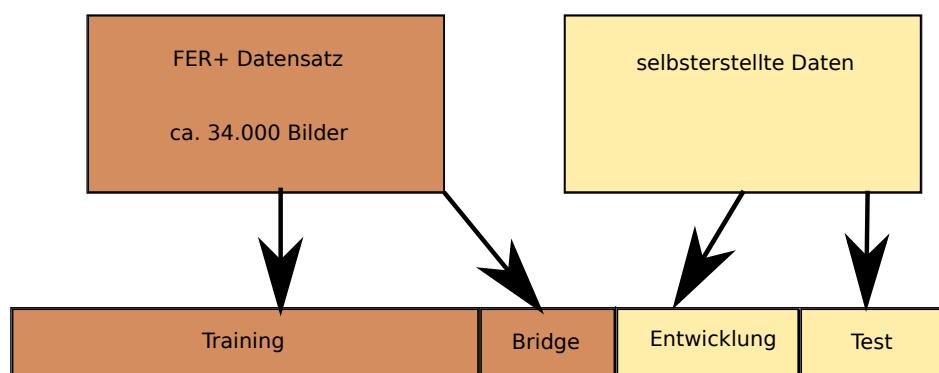
Beim maschinellen Lernen ist es üblich den vorhandenen Datensatz, bzw. die vorhandenen Datensätze in verschiedene Verwendungszwecke einzuteilen. Klassisch spricht man hier immer vom train/test-Split, also einer Aufteilung der Daten in einen Trainings- und einen Test-Datensatz. In modernen Projekten, welche sich mit maschinellen Lernen beschäftigen spricht man jedoch zumeist von einem train/dev/test-split. Die Daten werden also in einen Trainings-, einen Entwicklungs- und einen Test-Datensatz eingeteilt. Als Entwicklungs-Datensatz bezeichnet man, jene Daten, welche während der Entwicklung, also dem Anpassen bestimmter (Hyper-)Parameter, des neuronalen Netzes zur Evaluierung verwendet werden. Der Test-Datensatz ist in diesem Szenario ein Satz aus Daten, welches das neuronale Netz vor der Fertigstellung noch nicht “zu sehen” bekommen hat. Beim klassischen train/test-split ist der Test-Satz also eigentlich, das was wir Heute als Entwicklungs-Datensatz bezeichnen, und es gibt keinen wirklichen Test-Datensatz. Bei der Wahl der Datenquellen, ist es wichtig, dass die Test-Daten

3 Umsetzung & Evaluierung

möglichst ähnlich, zu den später erwarteten Eingangsdaten sind, und dass Entwicklungs- und Test-Datensatz aus der selben Quelle stammen sollten.

Für diese Arbeit bedeutet das, dass die Entwicklungs- und Test-Daten aus den selbstgenerierten Daten stammen, da diese bereits von aufgenommenen Videos stammen, was den Zeildaten sehr nahe kommt. Als Trainingsdaten wird entsprechend der FER+ Datensatz verwendet. Ein Problem bei einer solchen Aufteilung, wenn also die Trainingsdaten aus einem anderen Datensatz stammen als die Entwicklungs und Test-Daten, ist, dass man gewisse Probleme, wie zum Beispiel eine Überanpassung teilweise nur schwer erkennen kann. Deshalb ist es in einem solchen Fall sinnvoll noch einen vierten Datensatz einzuführen, welcher aus der selben Quelle wie die Trainingsdaten stammt (hier FER+). Man spricht hier vom dev_train oder auch bridge Datensatz. Dieser wird im Prinzip analog zum Entwicklungsdatensatz behandelt und dient zum Testen der Parameter des neuronalen Netzwerkes nach jeder Änderung. Anhand der Unterschiedlichen Ergebnisse für den bridge und den dev Datensatz kann man nun schnell bestimmte Probleme des neuronalen Netzwerks erkennen.

In dieser Arbeit wurde daher auch die Einteilung in 4 Datensätze gewählt. Die Daten wurden dabei wie folgt aufgeteilt: Die selbsterstellten Daten wurden in zu je 50% in den Entwicklungs- und Test-Datensatz aufgeteilt. Vom FER+ Datensatz wurden 10% Der Bilder für den Bridge Datensatz verwendet und 90% als Trainingsdaten. Die Aufteilung ist in Abbildung ?? veranschaulicht.



3 Umsetzung & Evaluierung

Um eine Eingangs zufällige, jedoch immer gleich reproduzierbare Aufteilung zu erzielen wurde das folgende Python Skript verwendet.

TODO: Code Listing Data_split.py

3.3 Datenpräparation

Um die Effizienz, sowie die Genauigkeit der Vorhersage des neuronalen Netzwerkes zu steigern werden alle Daten, bevor Sie dem KNN präsentiert werden auf diverse Arten präpariert. Im Folgenden wird auf die angewandten Methoden genauer eingegangen.

3.3.1 Vorverarbeitung

Um ein möglichst gutes Ergebnis zu erzielen wurden die selbst erstellten Daten, aber auch die frei verfügbaren Daten aus dem FER+ Datensatz auf diverse Weise vorverarbeitet. Bei den selbst aufgezeichneten Daten, wurde wie bereits erwähnt eine abschließende manuelle Sichtung vorgenommen, um möglichst alle falsch markierten Daten auszusortieren. Des weiteren werden alle der selbst aufgezeichneten Bilder auf den Ausschnitt des Gesichtes beschränkt, bevor sie dem selbst entworfenen KNN präsentiert werden. Dazu wird der bereits beschriebene Viola-Jones-Detektor (Shen et al. 1997) verwendet um das Gesicht im jeweiligen Bild zu detektieren. Nachdem der, das Gesicht enthaltende, Teil des Bildes ausgeschnitten wurde, wird dieser auf 48x48 Pixel skaliert und in ein Graustufenbild umgewandelt. Damit haben letzten Endes alle Eingangsdaten des neuronalen Netzes die gleiche Struktur. Der Python Code für diese Vorverarbeitung ist im folgenden Listing zu sehen.

TODO: Listing, preprocess selfrecorded data.

3 Umsetzung & Evaluierung

Bei den Daten aus dem FER+ Datensatz ist etwas weniger Vorverarbeitung nötig. Die Vorverarbeitung dieser Daten besteht im wesentlichen darin, die Pixelwerte welche als ein eindimensionales Array vorliegen in eine 48x48 Matrix umzuwandeln. Des weiteren wird aus den mehrstimmigen Labels des FER+ Datensatzes mithilfe des einfachen Mehrheitsprinzips das hier genutzte extrahiert. In einem letzten Schritt werden dann alle Datensätze, welche mit “not a face” markiert sind, aussortiert. (siehe Listing TODO:)

TODO: CODELISTING PROCESS FER+

3.3.2 Normalisierung

“Data normalization has been proposed to address the aforementioned challenge by reducing the training space and making the training more efficient.”
(Zhang et al. 2018)

Ein üblicher Schritt, um die Trainingsphase im maschinellen Lernen zu beschleunigen ist es die Eingabedaten zu normalisieren. Ziel ist es die Eingabedaten, welche auf einem sehr breiten Spektrum liegen so zu normalisieren um das Spektrum zu verkleinern. Im vorliegenden Fall geht es um die Graustufenbilder. Im Generellen kann man Normalisierung von solchen Bildern wie folgt beschreiben: Ein n-dimensionales Graustufenbild $I : \{X \subseteq \mathbb{R}^n\} \rightarrow \{\text{Min}, \dots, \text{Max}\}$ mit den Pixelwerten zwischen Min und Max wird in ein neues Graustufenbild $I_N : \{X \subseteq \mathbb{R}^n\} \rightarrow \{\text{newMin}, \dots, \text{newMax}\}$ mit Pixelwerten zwischen newMin und newMax überführt. (Gonzalez & Woods 2008) Die lineare Normalisierung eines Graustufenbildes berechnet sich wie folgt:

$$I_N = (I - \text{Min}) \frac{\text{newMax} - \text{newMin}}{\text{Max} - \text{Min}} + \text{newMin}$$

Im vorliegenden Beispiel sind die Ausgangswerte für $\text{Min} = 0$ und $\text{Max} =$

3 Umsetzung & Evaluierung

255 und für eine einfache Normalisierung werden die Werte $newMin = 0$ und $newMax = 1$ gewählt, damit alle Datenwerte zwischen 0 und 1 liegen. Damit ergibt sich die vereinfachte Formel:

$$I_N = \frac{I}{\text{Max}} \Rightarrow I_N = \frac{I}{255}$$

Zur Normalisierung der Daten werden dementsprechend alle Pixelwerte durch 255 dividiert, bevor das Bild dem neuronalen Netz gezeigt wird.

3.3.3 Datenmehrung

Je mehr Trainingsdaten für das KNN vorhanden sind, desto besser kann es auch mit ungesehenen Daten umgehen. Da nur begrenzt viele Daten zur Verfügung stehen werden in dieser Arbeit einige Methoden der künstlichen Datenvermehrung angewandt. Dazu werden die Bilder der Eingangsdaten zum Beispiel gespiegelt, verzerrt oder gedreht. In dieser Arbeit wurden die Methoden der Spiegelung, sowie des zufälligen drehens einiger Bilder angewandt. Durch die Spiegelung, bzw. Drehung eines Bildes entsteht wieder ein neues Bild, welches zum Training des neuronalen Netzes verwendet werden kann. So kann mit dieser relativ einfachen Methode die Anzahl der Trainingsdaten sehr einfach verdoppelt werden. Die Methode die zur Vermehrung der Bilder des FER+ Datensatzes verwendet wurde ist nachfolgende abgebildet.

TODO: Code Listing spiegeln/drehen

3.4 Entwurf und Entwicklung neuronaler Netze

Im Rahmen dieser Arbeit sollen 3 verschiedene neuronale Netzwerke entworfen, trainiert und evaluiert werden. Alle 3 sollen gefaltete neuronale Netze sein, sich jedoch in der Topologie und den (Hyper-) Parametern unterscheiden.

3.4.1 Entwicklungsumgebung

Die Entwicklung der KNN's wurde in Python mithilfe der Machine-Learning Erweiterung Keras vorgenommen. Keras ist eine vereinfachte Schnittstellenimplementierung zur einfachen Verwendung von verschiedenen Machine Learning-Schnittstellen. In dieser Arbeit wurde Keras mit der Tensorflow-Schnittstelle verwendet. Als Entwicklungsumgebung wurde hierzu ein Jupyter Notebook verwendet. Der Vorteil eines Jupyter Notebook liegt darin, dass sehr einfach Text und Programmierabschnitte, sowie deren Ausgabe nebeneinander visualisiert werden können.

3.4.2 Topologie

Unter der Topologie des KNN versteht man die Architektur im Zusammenhang mit den (Hyper-)Parametern. Die Architektur beschreibt den Aufbau, oder die Struktur, des Netzwerkes. In einem CNN also im wesentlichen die Art und Reihenfolge der einzelnen Netzwerkschichten.

Als Hyperparameter hingegen bezeichnet man weitere Rahmenparameter, welche unabhängig vom grundlegenden Aufbau des Netzes verändert werden können. Einige solcher Hyperparameter wurden in Kapitel 2 bereits vorgestellt.

In dieser Arbeit sollen drei verschiedene Netzwerktopologien für das Problem

3 Umsetzung & Evaluierung

der Emotions-Klassifizierung entworfen werden.

3.4.2.1 Einfaches faltendes neuronales Netz

Als erstes Model soll ein sehr einfaches faltendes neuronales Netz entworfen werden. Die Architektur des Netzes sieht dabei wie folgt aus:

3.4.2.2 abgewandeltes Xception Net

3.4.2.3 zeitabhängiges faltendes neuronales Netz

3.4.3 Hyperparameter

3.5 Evaluierung neuronaler Netze?

3.5.1 abschnitt 1

3.5.2 abschnitt 2

3.6 Entwicklung eines Webservice

Nachdem sich für das passende neuronale Netz entschieden wurde wird dieses einem Webservice zur Verfügung gestellt. Der Webservice hat die Aufgabe Videodaten entgegen zu nehmen und sekundenweise Einzelbilder an den

3 Umsetzung & Evaluierung

Klassifizierer zu übergeben und anhand der Ausgabe eine Zeitleiste mit den erkannten Emotionen zurückliefern.

3.6.1 Softwarearchitektur

Der Webservice wurde mithilfe der Python Erweiterung Flask realisiert. Flask ist eine schlanke Erweiterung zur einfachen Erstellung von Web Diensten in Python. Der Werbservice, sowie der Klassifizierer sind als sogenannte Microservices aufgebaut welche in separaten Containern laufen. (siehe Abbildung 3.6.1)

TODO: Abbildung Architektur

4 Literaturverzeichnis

Anon, 2019. neuronalesnetz.de - Hebb Regel., p.1. Available at: <http://www.neuronalesnetz.de/hebb.html> [Accessed June 3, 2019].

Barsoum, E. et al., 2016. Training Deep Networks for Facial Expression Recognition with Crowd-Sourced Label Distribution. Available at: <http://arxiv.org/abs/1608.01041>.

Charles Darwin, B., Society Hardwicke, R. & Fritz Muller, B., 1872. With illustrations. 2 vols. 8vo. 24s. Murray, 1871. THE VARIATION OF ANIMALS AND PLANTS UNDER DOMESTICATION. Third Thousand. With Illustrations. 2 vols. 8vo. 28s. Murray, 1868. Effects of Crossing. With Woodcuts., p.1862. Available at: https://pure.mpg.de/rest/items/item%7B/_%7D2309885/component/file%7B/_%7D2309884/content.

Ekman, P., 2004. Ekman-ShouldWeCallIt., 10(4), pp.1–15. Available at: [papers2://publication/uuid/2979D086-4479-4CC8-B669-E4B8E5FC3F74](http://publications/uuid/2979D086-4479-4CC8-B669-E4B8E5FC3F74).

Gonzalez, R.C. & Woods, R.E., 2008. Digital image processing, Pearson/Prentice Hall. Available at: <https://books.google.de/books?id=8uG0njRGEzoC>.

Goodfellow, I., Bengio, Y. & Courville, A., 2016. Deep learning, MIT Press.

Gupta, M.S. & Reddi, V.J., 2013. Synthesis Lectures on Computer Architecture.

Heinsohn, J., Boersch, I. & Socher, R., 2012. Wissensverarbeitung : eine Einführung in die Künstliche Intelligenz für Informatiker und Ingenieure 2. Aufl.,

Kirste, M. & Schürholz, M., 2018. Einleitung: Entwicklungswege zur KI,

4 Literaturverzeichnis

Kruse, R. et al., 2015. Computational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze, Available at: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84858020615%7B/&%7DpartnerID=tZ0tx3y1>.

Neuronalesnetz.de, Backpropagation.

News, 2018. KI - Künstliche Intelligenz. <Http://Link.Springer.Com/Article/10.1007/S13218-011-0122-Y>, (25), pp.10.1007/s13218-011-0122-y.

Ng, A., 2017. Neural Networks and Deep Learning. Available at: https://www.youtube.com/watch?v=fXOsFF95ifk%7B/&%7Dlist=PLkDaE6sCZn6Ec-XTbcX1uRg2%7B/_%7Du4x0Eky0%7B/&%7Dindex=26%7B/&%7Dt=0s.

Papageorgiou, C.P., Haar-Like _eng.pdf., pp.555–562.

POSNER, J., RUSSELL, J.A. & PETERSON, B.S., 2005. The circumplex model of affect: An integrative approach to affective neuroscience, cognitive development, and psychopathology. *Development and Psychopathology*, 17(3), pp.715–734.

Schmidt, P.A. & A, O.K.M., 2014. Prototypische Realisierung mit Betrachtung der ethischen Dimension Prüfer : Betreuer : begonnen am : beendet am : Inhaltsverzeichnis. PhD thesis.

Schuller, B., 2006. Automatische Emotionserkennung aus sprachlicher und manueller Interaktion., p.243.

Shen, Y.G. et al., 1997. Rapid Object Detection using a Boosted Cascade of Simple Features. *Surface Science*, 394(1-3).

Zhang, X., Sugano, Y. & Bulling, A., 2018. Revisiting data normalization for appearance-based gaze estimation., pp.1–9.