HOCHSCHULE FURTWANGEN UNIVERSITY | HFU

Masterthesis

im Studiengang

Medieninformatik Master

Nahe Echtzeit-Emotionserkennung mittels Machine-Learning-Verfahren auf High-Framerate-Videodaten im Kontext der Flowmessung

Referent	:	Prof. Dr. Ruxandra Lasowski
Koreferent	:	Ehm Kannegieser - Fraunhofer IOSB
Vorgelegt am	:	31.05.2021
Vorgelegt von	:	Linus Ehmann Matrikelnummer: 263813 Uhlandstraße 23, 70182 Stuttgart linus.ehmann@gmx.de

Abstract

In der vorliegenden Masterarbeit wird ein neuronales Netz dazu genutzt, um aus sehr kurzen und subtilen Gesichtsausdrücken, den sogenannten Micro Expressions, eine dazugehörige Emotion zu prognostizieren. Um an dieses Ziel zu gelangen, wird das neuronale Netz mithilfe mehrerer Micro Expression Datensätze und verschiedener Parametereinstellungen der Schichten und der Inputdaten trainiert. Diese prognostizierten Emotionen sollen in Zukunft weiterverarbeitet werden, um im Rahmen eines Versuchsaufbaus eine Messung von Immersions- und Flowzuständen möglich zu machen. Der Inhalt dieser Arbeit ein Zwischenschritt auf dem Weg dahin. Für den Interessenten ist zu beobachten wie sich verschiedene Einstellungen der Eingangsdaten und Parameter auf das Ergebnis der Prognose auswirken.

In this master thesis, a neural network is used to predict an associated emotion from very short and subtle facial expressions, the so-called micro expressions. To achieve this goal, the neural network is trained using several micro expression datasets and different parameter settings of the layers and the input data. In the future, these predicted emotions will be further processed to make it possible to measure immersion and flow states as part of an experimental setup. This work is therefore an intermediate step on the way to measuring flow. For those interested, it is to be observed how different settings of the input data and parameters affect the result of the prediction.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

AU Action Unit

CASME Chinese Academy of Science Micro-expression

CNN Convolutional Neural Network

CPU Central Processing Unit

CRF Conditional Random Field

CUDA Compute Unified Device Architecture

DNN Deep Neural Network

DSSN-MER Dual-Stream Shallow Networks for Facial Micro-Expression Recognition

DTSCNN Dual Temporal Convolutional Neural Network

EEG Elektroenzephalografie

EKG Elektrokardiogramm

EMG Elektromyografie

FACS Facial Action Coding System

fps frames per second

GSR Galvanic Skin Response

IDGA International Games Developers Association

IOSB Insitut für Optronik, Systemtechnik und Bildauswertung

KI künstliche Intelligenz

LBP Local Binary Pattern

LBP-TOP Local Binary Patter - Three Orthogonal Planes

LSTM Long Short-Term Memory

MKL Multiple Kernel Learning

ReLU Rectified Linear Units

RF Random Forest

SDM Supervised Descent Method

SMIC Spontaneous Micro-expression Database

STCLQP SpatioTemporal Completed Local Quantization Patterns

STSTNet Shallow Triple Stream Three-dimensional CNN

SVM Support Vector Machine

VR Virtual Reality

1. Einleitung

In den letzten Jahren ist die Zahl der Videospielenden weltweit kontinuierlich gestiegen. So haben im Jahr 2017 bereits über 2,8 Milliarden Menschen Videospiele konsumiert. Diese Zahl soll schätzungsweise bis zum Jahr 2025 auf über 4,3 Milliarden ansteigen.[1] Durch die derzeitige Covid-19 Pandemie und den damit verbundenen Ausgangssperren kam es zudem zu einem deutlichen Anstieg an Spielenden.[2] Bei einer Umfrage in den Vereinigten Staaten gaben 48 Prozent der Befragten an, dass sie Videospiele als Flucht aus dem alltäglichen Leben konsumieren.[3] Sie möchten vollkommen im Spiel versinken und die Welt um sich herum vergessen. Dieser Zustand des Eintauchens und von der virtuellen Welt umgeben zu sein, wird als Immersion oder Flow bezeichnet. Bisher ist es noch relativ schwierig, diese immersiven Zustände zu messen.

Eine Methode, die gerne benutzt wird, sind Fragebögen, die die Spielenden nach dem Spielerlebnis ausfüllen. Hier besteht jedoch die Problematik, dass die eigentliche Immersion bereits einige Zeit zurückliegen kann. Die Spielenden können sich oft nicht mehr erinnern, an welchen Stellen sie in immersiven Zuständen waren oder ob sie überhaupt Immersion erlebt haben, da dies für sie schwer einzuschätzen ist.

Das Fraunhofer-Insitut für Optronik, Systemtechnik und Bildauswertung (IOSB) versucht dieses Problem zu lösen, indem es einen Versuchsaufbau realisiert, der es ermöglicht, den Flow messbar zu machen. Hierfür spielen Probanden ein Videospiel für etwa eine halbe Stunde. Während des Spielens sind sie an verschiedene Messinstrumente angeschlossen, um bei-

[1] Statista 2020
[2] GlobalWebIndex 2020
[3] Newzoo 2020

spielsweise den Puls oder Hirnströmungen zu erfassen. Außerdem werden sie von einer Kamera gefilmt. Mithilfe dieser Videoaufnahmen werden die Emotionen der Probanden analysiert. Interessant sind dabei vor allem die sogenannten Micro Expressions.
Dies sind sehr kurze Emotionen, welche nur im Bruchteil einer Sekunde stattfinden und die wahren Emotionen einer Person abbilden. Für das menschliche Auge sind diese Gesichtsausdrücke nur mit viel Übung zu erkennen. Aus diesem Grund soll für die Erkennung der Micro Expressions und der damit verbundenen Emotion ein neuronales Netz genutzt werden. Diese Arbeit befasst sich mit einem Teilaspekt aus dem gesamten Versuchsaufbau der Flowmessung, der Einbindung eines neuronalen Netzes zur Micro Expression und Emotionserkennung. Es werden folgende Ziele verfolgt:

- Präsentation eines Überblicks an Methoden zur Erkennung von Micro Expressions.
- Entwicklung eines Systems, das Micro Expressions mithilfe von Deep Learning aus Videodaten prognostizieren kann.
- Entwicklung dieses Systems so, dass es in den bisherigen Versuchsaufbau integriert und die Prognose aus Daten des Livevideos durchgeführt werden kann.

Dafür werden in einem ersten Schritt die wichtigsten theoretischen Hintergründe erklärt. In Kapitel 2.1 werden die beiden Theorien der Immersion und des Flows und deren Vereinbarkeit erläutert. Danach wird in Kapitel 2.2 geklärt, um was es sich bei Micro Expressions handelt und wie diese entdeckt wurden. In Kapitel 2.3 werden die Unterschiede zwischen künstlicher Intelligenz, Machine Learning und Deep Learing erläutert, sowie verschiedene Arten zum Training eines neuronalen Netzes erklärt. Darauf folgt verwandte Literatur im Kontext der Emotionserkennung mit Machine beziehungsweise Deep Learning (Kapitel 2.4). Den Abschluss von Kapitel 2 bildet der Einblick in den Versuchsaufbau, der zur Zeit am Fraunhofer-IOSB verwendet wird (Kapitel 2.5). In Kapitel 3 wird näher auf die Methodik und die technischen

Bestandteile der Arbeit eingegangen. Die ersten Schritte des Projekts und welche zusätzlichen Ziele daraus abgeleitet wurden, werden in Kapitel 3.1 dargestellt. Im Anschluss wird das schlussendlich verwendete neuronale Netzwerk und dessen Schichten erklärt (Kapitel 3.2). Anschließend wird in Kapitel 3.3 auf die Datensätze eingegangen, die für das Training des Netzwerkes benutzt wurden. Wie die Bilder aus den Datensätzen zu Trainingssets zusammengestellt wurden und welche Ergebnisse das jeweilige Training hatte, wird in Kapitel 3.4 dargestellt. Abschließend wird in Kapitel 3.6 das selbst geschriebene Skript und dessen Funktionsweise anhand einiger Codebeispiele erklärt. Die Arbeit wird im Kapitel 4 mit einer Zusammenfassung der Daten und einem Ausblick auf zukünftige Arbeiten abgeschlossen.

2. Theoretische Grundlagen

2.1. Immersion und Flow

Im Umfeld der Videospiele gibt es zwei mentale Zustände, die in gewisser Weise darstellen, wie sehr die Spielenden in das virtuelle Geschehen involviert sind. Hierbei handelt es sich zum einen um den sogenannten Immersions-Effekt und zum anderen um die Flow-Theorie.

2.1.1. Immersion

Bei der Immersion handelt es sich um einen mentalen Zustand, in welchem Spielende sich völlig der virtuellen Welt hingeben und so komplett in diese eintauchen. Eine Beschreibung der Immersion, die in der Videospieltheorie gerne genutzt wird, ist die Beschreibung der US-Amerikanerin Janet H. Murray.[4]

> *Immersion* is a metaphorical term derived from the physical experience of being submerged in water. We seek the same feeling from a psychologically immersive experience that we do from a plunge in the ocean or swimming pool: the sensation of being surrounded by a completely other reality, as different as water is from air, that takes over all of our attention, our whole perceptual apparatus. We enjoy the movement out of our familiar world, the feeling of alertness that comes from being in this new place, and the delight that comes from learning to move within it.

[4] Murray 2016 S.99

Die Immersion ist allerdings nicht einfach ein Zustand, in dem die Spielenden sich entweder befinden oder nicht, sondern es gibt verschiedene Stufen, die fließend ineinander übergehen. Nach Richard Bartle gibt es vier verschiedene Immersionsstufen.

- Player
- Avatar
- Character
- Persona

Beim Player in der ersten Stufe handelt es sich um die spielende Person selbst. Diese steuert ein Objekt durch die virtuelle Welt. „The way the player regards that object is a measure of their immersion.“[5] Falls das Objekt lediglich als Konstrukt gesehen wird, kommt es zu keiner Identifikation und demnach kann es auch nicht zu einer Immersion führen.
Wird das Objekt jedoch als Repräsentation des Spielenden in der virtuellen Welt gesehen, befindet sich dieser in der zweiten Stufe und das Objekt ist der Avatar. Dieser Repräsentant wird von Spielenden in der dritten Person referenziert.
Auf der nächsten Stufe, der Character-Stufe, befinden sich laut Bartle die meisten Spielenden. „A character is an extension of a player's self [...]“[6]. Üblicherweise wird die Spielfigur nun in der ersten Person referenziert.
Die letzte Stufe ist gleichzeitig auch die höchste Stufe der Immersion. Der Spieler wird auf dieser Stufe zum Charakter selbst. „You're not role-playing a being, you are that being; you're not assuming an identity, you are that identity; you're not projecting a self, you are that self. If you're killed in a fight, you don't feel that your character has died, you feel that you have died[43]. There's no level of indirection, no filtering, no question: You are there.“[7]
Bartle beschreibt diesen Zustand, welche er Persona nennt, als die eigentliche Immersion. Die Avatar und Character-Stufe sind für ihn nur Schritte

[5] Bartle 2003 S. 155
[6] Bartle 2003 S. 155
[7] Bartle 2003 S. 155

auf dem Weg zur vollkommenen Immersion. „ When player and character merge to become a persona, that's immersion; [...] that's when they stop playing the world and start living it.“[8] Neben der Einordnung in Stufen kann die Immersion zudem in verschiedene Typen untergliedert werden. Der Gründer der International Games Developers Association (IDGA) Ernest Adams unterscheidet die Immersion in vier Typen: taktische, strategische, räumliche und erzählerische Immersion. Die **taktische Immersion** tritt vor allem in Spielen mit schnellen Spielabläufen auf. Die Aufgaben, die Spielende erledigen müssen, sind in diesen Spielen so schnell, dass der Spielende an nichts anderes denken kann. Oft wird dieser Typ der Immersion auch als „*Tetris trance*“[9] bezeichnet.

Spielende, die die **strategische Immersion** erleben, versuchen sehr, das Spiel zu gewinnen. Sie sind damit beschäftigt, die nächsten Schritte zu planen und zu optimieren. Adams nennt als Beispiel für die strategische Immersion das Schachspielen.

Wenn Spielende während des Spielens denken sie seien in einer anderen Welt, dann ist die Rede von der **räumlichen Immersion**. Spiele, die mit Virtual Reality (VR)-Ausrüstung gespielt werden, versuchen genau diese Art der Immersion zu nutzen, da dort die Sicht der Spielfigur von der Kopfbewegung des Spielers selbst abhängt.

Neben dem Eintauchen in eine andere Welt, gibt es das Eintauchen in die Handlung des Spiels. Dies ist die sogenannte **erzählerische Immersion**. Sie ist der Immersion, die man erfahren kann, wenn man ein gutes Buch liest und in dessen Geschichte gefangen ist, sehr ähnlich. In Videospielen sind die Spielenden allerdings nicht nur passive Zuschauer, sondern können die Handlung mit ihren Taten selbst beeinflussen.[10]

Es ist zu erkennen, dass vor allem die letzten beiden Typen recht einfach mit den höheren Stufen Bartles in Verbindung gebracht werden können. Das Eintauchen in eine andere Welt oder eine andere Geschichte kann vergleichsweise einfach dazu führen, dass der Spielende sich mit der Spielfigur

[8] Bartle 2003 S.156
[9] Adams 2014 S.20
[10] vgl. Adams 2014 S.21

selbst identifiziert. Bei einem Schachspiel oder bei Tetris wird eine Person vermutlich keine Bindung zu den fallenden Steinen oder den Schachfiguren aufbauen. Allerdings ist es in anderen Spielen dieser Kategorie nicht komplett ausgeschlossen, ebenfalls die höchste Stufe an Immersion zu erfahren. Die Wahrscheinlichkeit ist hier nur geringer. Oftmals können Spielende, je nach Spiel, auf mehrere Arten zur Immersion gelangen. Eine Person erfährt von daher nicht immer nur eine Art der Immersion.
Eine weitere Untergliederung wurde von Jennett et al.[11] definiert. Hierbei handelt es sich um ein dreistufiges Modell. Die erste Stufe ist das Engagement, das jeder automatisch durch die Interaktion mit dem Spiel erlebt. In der zweiten Stufe befindet sich eine Person dann, wenn sie emotional in eine Aktivität involviert ist. Die dritte und letzte Stufe beschreibt wie in der Einordnung Bartles die totale Immersion, bei der eine Person das Raum- und Zeitgefühl verliert.

2.1.2. Flow-Theorie

Neben dem Effekt der Immersion existiert zudem eine verwandte Theorie: Die Flow-Theorie. Sie wurde 1975 vom ungarischen Psychologen Mihály Csíkszentmihályi beschrieben. Er beschreibt den Flow-Zustand als „the holistic sensation that people feel when they act with total involvement“.[12]
Er beschreibt außerdem, dass in diesem Zustand eine Aktion der nächsten gemäß einer internen Logik folgt, welche kein bewusstes Eingreifen der Person selbst benötigt. Wie der Name Flow schon vermuten lässt, fließt man förmlich von einem Moment zum nächsten. Die Unterscheidung zwischen sich selbst und der Umwelt, zwischen Reiz und Antwort oder zwischen Vergangenheit, Gegenwart und Zukunft fällt schwer.[13] Um diesen Flow-Zustand zu erreichen, bedarf es nach Csíkszentmihályi Aktivitäten, die eine perfekte Balance aus Anforderung und den eigenen Fähigkeiten darstellen. Abbildung 1 verdeutlicht dies. Ist die Anforderung aufgrund der eigenen eingeschränk-

[11] Jennett u. a. 2008
[12] Csikszentmihalyi 1975 S.36
[13] vgl. Csikszentmihalyi 1975 S.36

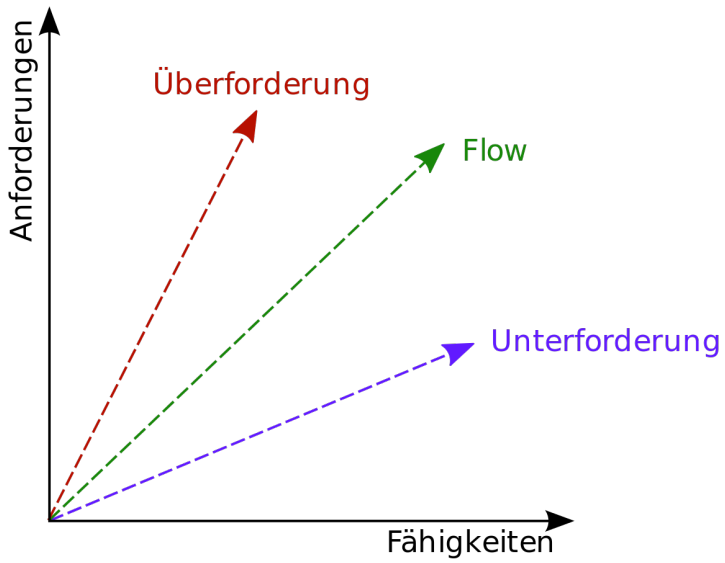

Abbildung 1.: Diagramm zu Flow Aktivitäten; Quelle: Autoren der Wikimedia-Projekte 2003

ten Fähigkeiten zu groß fühlt sich die Person überfordert. Ist jedoch die Anforderung zu niedrig beziehungsweise sind die eigenen Fähigkeiten zu hoch, kommt es ebenfalls zu keinem Flow-Zustand, sondern zu einer Unterforderung.

Der Flow-Zustand selbst ist also ein recht extremer Zustand, welcher entweder vorhanden ist oder nicht. Dies ist der große Unterschied zur Immersion, bei der die betroffene Person auch Abstufungen erfahren kann. Die höchste Stufe der Immersion kann demnach in gewisser Weise als Flow-Zustand betrachtet werden, da sich die betroffene Person vollkommen vom Geschehen umschlossen fühlt und von einem Moment zum nächsten fließt und die reale Welt um sich herum nicht mehr richtig wahrnimmt.

Auch Kennegieser et al.[14] haben eine Kombination der beiden Theorien vorgeschlagen: So wird das dreistufige Modell Jennetts in den Flow Bereich projiziert. Dabei ist die Stufe des Engagement im Bereich von geringer Anforderung und geringer Fähigkeit. Als Gegenstück ist die totale Immersion eine hohe Anforderung bei hoher Fähigkeit (siehe Abbildung 2).

[14]Kannegieser, Atorf und Herold 2021

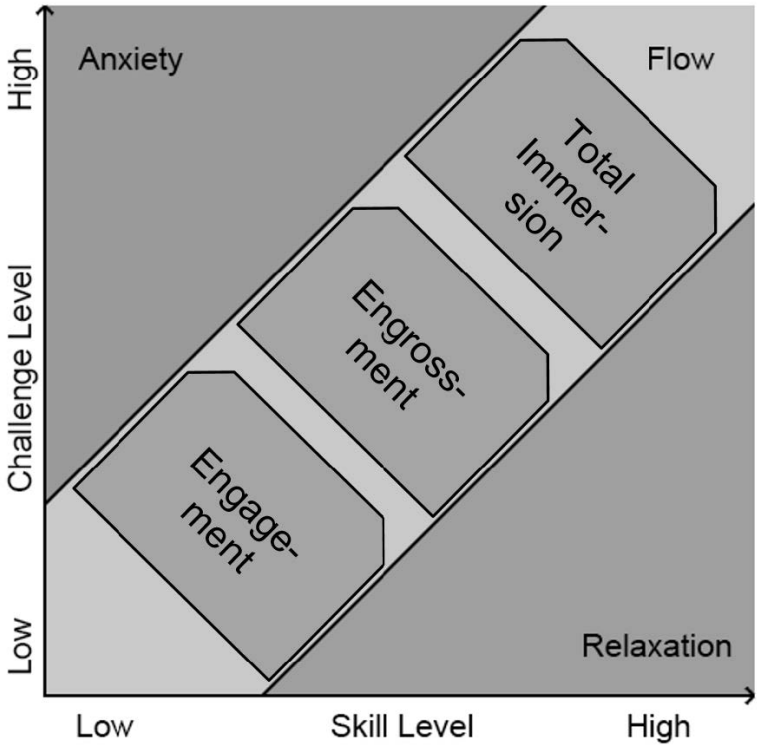

Abbildung 2.: Diagramm zu Kombination von Flow und Immersion; Quelle: Kannegieser, Atorf und Herold 2021

2.2. Micro Expressions

Bei Micro Expressions handelt es sich um sehr kurze, emotionale Gesichtsausdrücke, die meist nur den Bruchteil einer Sekunde andauern. Entdeckt wurden diese „micromomentary expressions“[15] 1966 von Ernest A. Haggard und Kenneth S. Isaacs, als sie Filme von psychotherapeutischen Sitzungen Frame für Frame erforschten. Ihnen fiel auf, dass sich der Gesichtsausdruck der Patienten gelegentlich sehr dramatisch änderte. Innerhalb von drei bis fünf Frames, was in ihrem Film etwa einer Zeit von ⅛ bis ⅕ einer Sekunde entsprach, wechselte der Patient beispielsweise von einem Lächeln auf eine Grimasse und wieder zurück.[16]

Nach weiteren Untersuchungen stellten die beiden die These auf, dass diese kurzen Mikomimiken durch das Unterbewusstsein gesteuert werden. Auch der amerikanische Psychologe Paul Ekman beschreibt Micro Expressions

[15] Haggard und Isaacs 1966
[16] vgl. Haggard und Isaacs 1966

als oftmals unfreiwillige Emotionen, die die wahren Emotionen einer Person offenbaren. Darüber hinaus sagt Ekman, dass die Micro Expressions auch auftreten können, wenn eine Person aktiv versucht, eine Emotion zu unterdrücken. Er spricht hierbei von einem emotionalen Leck[17].

Genau wie Haggard und Isaacs entdeckte Ekman die Micro Expressions beim Erforschen von Zeitlupenaufnahmen. Er fand dies allerdings drei Jahre nach der Entdeckung von Haggard und Isaacs heraus, ohne von deren Ergebnissen zu wissen. Gemeinsam mit seinem Mitarbeiter Wallace Friesen wollte er herausfinden, ob durch Gesichtsausdrücke erkennbar ist, dass eine Person lügt oder nicht.

Ein Erlebnis, das Ekman im Jahr zuvor hatte, stellte sich als glücklicher Zufall heraus: Ekman hatte Gespräche mit psychiatrischen Patienten im Laufe ihres Aufenthaltes in einer Klinik gefilmt. Einmal zu Beginn der Therapie, einmal als sich der Zustand verbessert hatte und einmal kurz vor der Entlassung. Eine Patientin hatte in ihrem mittleren Gespräch erklärt, dass sie nicht mehr depressiv sei und deshalb um eine Ausgangsgenehmigung gebeten. Die Aussage stellte sich jedoch als Lüge heraus, da die Patientin im Anschluss kurz vor ihrem Ausgang zugab, einen Suizid geplant zu haben.

Das Video des Gesprächs analysierten die beiden Bild für Bild. „An einem Punkt im Gespräch fragte der Arzt Mary nach ihren Plänen für die Zukunft. In einem kleinen Augenblick der Pause, bevor sie die Frage beantwortete, sahen wir, wie eine Miene mit starker Angst über ihr Gesicht blitzte. Es ging um eine Sekunde, und es handelte sich nur um zwei Einzelbilder von 24 – ein Zwölftel einer Sekunde –, die rasch von einem Lächeln verdeckt wurden.“[18]

Nachdem die beiden wussten, auf welche Merkmale sie achten mussten, um diese Micro Expressions zu erkennen, fanden sie noch weitere in besagtem Video.

Demzufolge gibt es zwei Arten von Micro Expressions: Die bewusst verschleierten Emotionen, wie dies Ekman und Friesen herausfanden, und die unterbewussten, verdrängten Emotionen, die von Haggard und Isaacs

[17] vgl. *Micro Expressions | Facial Expressions | Paul Ekman Group* 2020 und Ekman und Friesen 2003 S. 151-152

[18] Ekman 2010 S. 296

entdeckt wurden. Im Gegensatz zu Haggard und Isaacs waren Ekman und Friesen in der Lage, die Micro Expressions auch in Echtzeit zu erkennen und waren nicht darauf angewiesen die Videos Frame für Frame zu analysieren. Ein weiteres wichtiges Forschungsergebnis Ekmans ist die Entdeckung der universellen Emotionen. Früher waren die Menschen der Auffassung, dass Emotionen sozial erlernt werden und deswegen auch von Kultur zu Kultur unterschiedlich ausfallen können. Ekman, der das Projekt mit der selben Denkweise begann, widerlegte dies jedoch und stellte fest, dass Emotionen in allen Kulturkreisen universell sind und das selbe bedeuten. In seinen Untersuchungen zeigte er Personen aus unterschiedlichen Kulturkreisen verschiedene Bilder und „bat sie zu beurteilen, was für ein Gefühl jeder einzelne Gesichtsausdruck repräsentierte.“[19]

Die Probanden wählten für verschiedene Gesichtsausdrücke mehrheitlich die selben Emotionen aus, egal welcher Ethnik sie angehörten. Daraufhin kamen erste Zweifel an der These der sozial erlernten Emotionen auf. Um sicher zu gehen, dass die verschiedenen Kulturen die zu beurteilenden Emotionen nicht durch Massenmedien erlernt hatten, führte Ekman zudem Untersuchungen bei indigenen Stämmen in Papua-Neuguinea durch. Auch hier kam er zum selben Ergebnis, wie bei seinen bis dahin durchgeführten Untersuchungen. Im Jahr 1969, noch vor seiner Entdeckung der Micro Expressions, veröffentlichte er seine Ergebnisse.

Laut Ekman gibt es demnach sieben universelle Emotionen (Wut, Verachtung, Ekel, Freude, Angst, Trauer und Überraschung). Dies ist im Kontext der Emotionsdetektion besonders wichtig, da somit eine Methode für alle Ethniken genutzt werden kann. Seine Ergebnisse warfen daraufhin viele weitere Fragen auf. „Über wie viele Gesichtsausdrücke verfügt der Mensch? Liefert ein Gesichtsausdruck eine korrekte Information oder ist er manchmal irreführend? Ist jede Gesichtsbewegung Ausdruck eines Gefühls? Können Menschen mit ihrem Gesicht genauso lügen wie mit Worten?“[20]

Die letzte Frage führt dann zu der bereits erläuterten Forschung der psych-

[19] Ekman 2010 S. 4
[20] Ekman 2010 S. 19

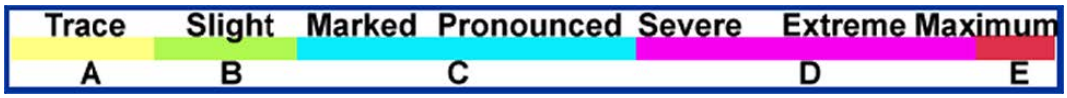

Abbildung 3.: Ausprägungsstärke der Action Units; Quelle: Ekman, Friesen und Hager 2002 S.8

iatrischen Patienten. In den folgenden Jahrzehnten nach seiner Veröffentlichung befasste sich Ekman intensiv mit weiteren dieser Fragen und fand heraus, dass ein Gesicht über 10000 Ausdrücke annehmen kann. Zusammen mit Friesen entwickelte er außerdem ein System, um Gesichtsbewegungen zu kodieren: das Facial Action Coding System (FACS).

Dieses System wurde 1978 veröffentlicht und besteht aus verschiedenen, sogenannten Action Units (AUs). Eine AU repräsentiert dabei eine sichtbare, muskuläre Bewegung im Gesicht. Darüber hinaus wird jeder AU ein Wert für dessen Ausprägung angegeben. Die Ausprägung kann fünf Stärken von A bis E annehmen. In der niedrigsten Stufe *A* ist nur eine geringe Spur der Veränderung wahrnehmbar (vgl. Abbildung 3).

Die AUs wurden im FACS-Handbuch in Bewegungen der oberen und unteren Gesichtshälfte unterteilt. AUs der oberen Hälfte umfassen demnach Bewegungen in der Augen-, Augenbrauen- und Stirnregion. Die AUs der unteren Hälfte umfassen Bewegungen der Mund- und Nasenregion. Darüber hinaus gibt es noch AUs, die für eine bestimmte Kopf- beziehungsweise Augenbewegung stehen. In Abbildung 4 sind die verschiedenen AUs mit ihrer Kennnummer und ihrem Namen dargestellt. Durch Kombination mehrerer dieser AUs können auch Emotionen definiert werden. So charakterisiert Ekman das Gesicht der Trauer folgendermaßen:

1. Innere Ecken der Augenbrauen werden nach oben und zusammen gezogen (AUs 1+4)

2. Obere Augenlider sind hängend und Augen schauen nach unten (AU 64)

3. Mundwinkel sind nach unten gezogen (AU 15)[21]

[21] *What is Sadness?* | *Feeling Sadness* | *Paul Ekman Group* 2021

Upper Face AUs			Lower Face AUs		
AU	**Name**	**Starting on**	**AU**	**Name**	**Starting on**
1	Inner Brow Raise	page 20	**9**	Nose Wrinkle	
2	Outer Brow Raise	page 22	**10**	Upper Lip Raiser	
4	Brow Lowerer	page 17	**11**	Nasolabial Furrow Deepener	
5	Upper Lid Raise	page 24	**12**	Lip Corner Puller	
6	Cheek Raise	page 31	**13**	Sharp Lip Puller	
7	Lids Tight	page 28	**14**	Dimpler	
43	Eye Closure	page 36	**15**	Lip Corner Depressor	
45	Blink	page 39	**16**	Lower Lip Depress	
46	Wink	page 40	**17**	Chin Raiser	
70	Brows Not Visible		**18**	Lip Pucker	
71	Eyes Not Visible		**20**	Lip Stretch	
	Head Positions		**22**	Lip Funneler	
51	Turn Left		**23**	Lip Tightener	
52	Turn Right		**24**	Lip Presser	
53	Head Up		**28**	Lips Suck	
54	Head Down		**72**	Lower Face Not Visible	
55	Tilt Left			**Miscellaneous AUs**	
56	Tilt Right		**8**	Lips Toward Each Other	
57	Forward		**19**	Tongue Show	
58	Back		**21**	Neck Tightener	
	Eye Positions		**29**	Jaw Thrust	
61	Eyes Left		**30**	Jaw Sideways	
62	Eyes Right		**31**	Jaw Clencher	
63	Eyes Up		**32**	Bite	
64	Eyes Down		**33**	Blow	
65	Walleye		**34**	Puff	
66	Crosseye		**35**	Cheek Suck	
	Lip Parting and Jaw Opening		**36**	Tongue Bulge	
25	Lips Part		**37**	Lip Wipe	
26	Jaw Drop		**38**	Nostril Dilate	
27	Mouth Stretch		**39**	Nostril Compress	

Abbildung 4.: Beschreibung der Action Units; Quelle: Ekman, Friesen und Hager 2002 S.514

FACS wird heutzutage vor allem von Geheimdiensten oder polizeilichen Behörden benutzt, da so die Micro Expressions, also die wahren Emotionen, der Befragten identifiziert werden können. Mithilfe der AUs konnten außerdem viele Emotionsdatensätze angefertigt werden (mehr zu solchen Datensätzen in Kapitel 3.3).

2.3. Neuronale Netze

Um verstehen zu können, um was es sich bei Deep Learning beziehungsweise bei neuronalen Netzen handelt, ist es wichtig sich bewusst zu werden, dass Deep Learning eine Teilmenge des Machine Learning ist. Machine Learning selbst ist wiederum eine Teilmenge der künstlichen Intelligenz.(siehe Abbildung 5). Aus diesem Grund macht es Sinn, auf der obersten Ebene, der künstliche Intelligenz (KI) zu beginnen und dann tiefer in die einzelnen Teilmengen zu blicken.

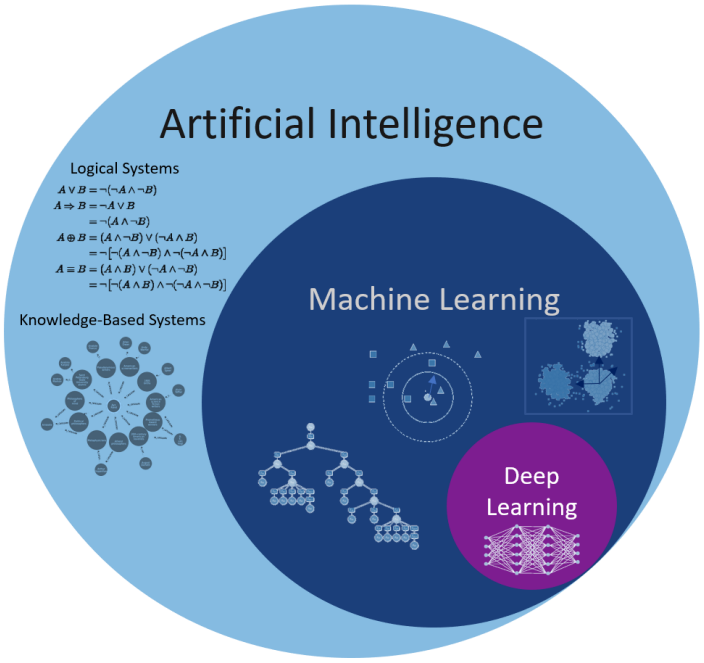

Abbildung 5.: Übersicht über Zusammenhang von künstlicher Intelligenz, Machine Learning und Deep Learning; Quelle: *Machine Learning vs Deep Learning – Wo liegt der Unterschied? – Data Science Blog* 2021

2.3.1. Künstliche Intelligenz

In den Medien werden die KI und Machine Learning oft als Synonyme verwendet. Jedoch ist Machine Learning, wie bereits erwähnt, nur eine Teilmenge der KI. Unter KI versteht man ein Programm oder eine Maschine, die möglichst menschlich und intelligent handelt. Dabei müssen neben dem Aspekt des Lernens noch andere Aspekte erfüllt werden. Zudem ist es wichtig, die erlernten Informationen in einer bestimmten Art und Weise zu verarbeiten und zu verstehen. Autonomes Fahren, Spracherkennung oder auch personalisierte Werbung sind Beispiele für KI. Die erlernten Informationen wie Kaufverhalten oder die Stimme werden mithilfe von anderen Systemen der KI so verarbeitet, dass den Nutzenden ein besseres Erlebnis geboten werden kann.

2.3.2. Machine Learning

Der Begriff Machine Learning beschreibt eine Anwendung „that can automatically learn and improve from experience without being explicitly programmed to do so. The learning occurs as a result of analyzing ever increasing amounts of data, so the basic algorithms don't change, but the code's internal weights and biases used to select a particular answer do.“[22] Die Algorithmen erkennen bestimmte Muster und Zusammenhänge zwischen Input- und Output-Paaren. Gibt man einem Algorithmus beispielsweise die Zahlen $1, 4, 5, 8$ und 10 als Input mit dem dazugehörigen Output $7, 13, 15, 21$ und 25, dann wird dieser zu dem Schluss kommen, dass der Zusammenhang zwischen den Zahlen $2x + 5$ sein muss. Wenn man nun noch eine 6 als Input gibt, wird der Algorithmus als Ergebnis die Zahl 17 prognostizieren.[23] Die Merkmale der Daten werden meistens durch von Hand entworfenen Algorithmen extrahiert und anschließend werden von der Maschine Zusammenhänge gelernt. Für den Lernprozess selbst wird eine große Menge an Daten benötigt.

[22] Mueller und Massaron 2019 S.26
[23] vgl. Mueller und Massaron 2019 S.26

2.3.3. Deep Learning

Da Deep Learning eine Teilmenge des Machine Learning ist, werden auch hier große Datenmengen zum Lernen benötigt. Anders als das klassische Machine Learning, bei dem die Algorithmen auf mathematischer Logik und statistischer Analyse beruhen, werden beim Deep Learning die Daten mithilfe von Recheneinheiten (Neuronen), die in verschiedenen Schichten angeordnet sind, verarbeitet. Diese Technik ergibt ein sogenanntes neuronales Netz. Ebenso müssen beim Deep Learning, dank der vielen Schichten, im Voraus keine Merkmale erstellt werden, wie dies beim Machine Learning der Fall ist. Das neuronale Netz lernt diese Merkmale von alleine.
Der Aufbau eines neuronalen Netzes ist im Grunde immer gleich. Es gibt eine Eingangsschicht, mehrere versteckte Schichten sowie eine Ausgangsschicht (siehe Abbildung 6). Die Neuronen in den versteckten Schichten haben alle eine gewisse Gewichtung, die beim Training des Netzes verändert werden, um bestimmte Ergebnisse zu erhalten. Je nachdem, welche Art von Input dem Netz geboten wird und welcher Output erwartet wird, können verschiedene Lernarten definiert werden.
Es gibt hier die Unterscheidung zwischen Supervised (deutsch: überwacht), Unsupervised (unüberwacht) und Reinforcement (verstärkend) Learning.

Supervised Learning:
Beim Supervised Learning bekommen die Inputdaten ein Label, wie beispielsweise eine bestimmte Klasse. Es wird also schon ein gewisser Output vom Netzwerk erwartet. Wenn man beispielsweise Bilder von verschiedenen Tieren hat, kann man jedem Bild als Label die Art des Tieres geben. Das Netzwerk passt das Modell an die Daten an und die Prognosen werden mit der Zeit immer genauer.
Wenn dem Netz nun ein neues, unbekanntes Bild vorgelegt wird, versucht dieses eine Prognose zu treffen, in welche Kategorie das Tier gehört. Grundsätzlich gibt es zwei Arten von Problemen, die mithilfe von Supervised Learning gelöst werden können. Dies sind zum einen Regressionsprobleme, bei

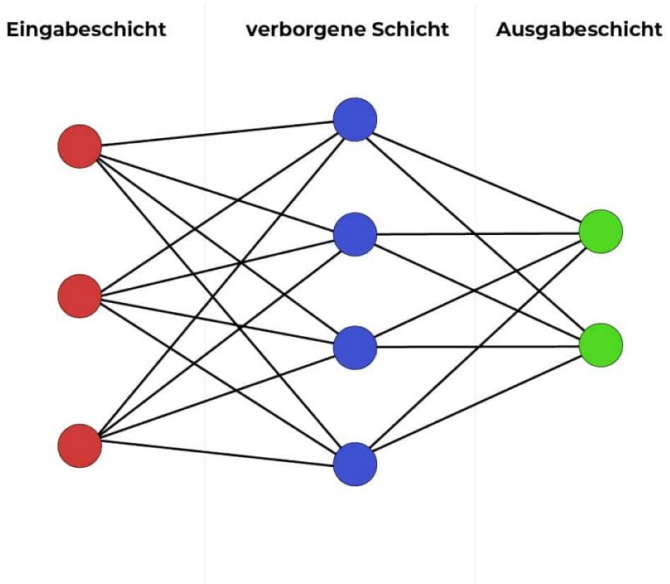

Abbildung 6.: Darstellung eines einfachen neuronalen Netzes; Quelle: *Deep Learning 2021: Was ist es und warum wird es eingesetzt?* 2021

denen als Output ein numerischer Wert erwartet wird, wie zum Beispiel bei einer Preisentwicklung. Zum anderen sind dies Klassifikationsprobleme, die eine Klasse prognostizieren, wie im Falle der Tierebilder.[24] Diese Methode wird es auch für das Problem der Emotionserkennung zum Einsatz kommen, da es dabei um ein Klassifikationsproblem handelt. Als Output wird am Ende eine Prognose gewünscht, die aussagt, um welche Art der Emotion es sich handelt.

Unsupervised Learning:

Im Gegensatz zum Supervised Learning sind die Daten beim Unsupervised Learning nicht gelabelt. Der Output, den das Netzwerk liefert, ist nicht bekannt. Die Eingangsdaten werden analysiert und das Netz findet daraufhin selbst interessante Muster in diesen Daten. Das Unsupervised Learning ist nicht dazu geeignet, um in irgendeiner Form eine Prognose zu treffen, sondern um große Datenmengen in bestimmte Gruppen einzuordnen. Wenn man dem Netz zum Beispiel erneut Tierbilder als Input gibt, wird es versuchen, diese in passende Gruppen einzuordnen. Als Output kann so ebenfalls

[24] vgl. *Machine Learning: Algorithmen, Methoden und Beispiele* 2021; Mueller und Massaron 2019 S.28f

	Unsupervised learning	Supervised learning
Prozess	Nur Inputdaten sind gegeben	Input- und Outputdaten sind vorgegeben
Inputdaten	Beispieldaten ohne Zielvaribale	Beispieldaten mit Zielvaribale
Echtzeiteinsatz	Kann in Echtzeit genutzt werden	Das Lernen passiert vor dem Deployment
Anzahl der Features	Anzahl ist unbekannt	Anzahl ist bekannt
Einsatzgebiete	Generierung von Wissen und Mustern aus großen Datenmengen: z.B. Clustering von Kundenmerkmalen, Dimensionsreduktion von großen Datensätzen oder Extraktion von einem Regelwerk.	Vorhersagen von Werten und Klassen: z.B. Vorhersage von einer Kündigung, Kaufwahrscheinlichkeiten oder den Stromverbrauch.

Abbildung 7.: Unterschiede zwischen Supervised und Unsupervised Learning; Quelle: *Supervised Learning: Definition, Arten & Beispiele - datasolut Wiki* 2021

eine Einordnung nach Tierart vorkommen, es kann allerdings auch sein, dass das Netz die Tiere nach ihrer Größe oder Farbe ordnet.[25] Ein weiterer Unterschied ist, dass das Unsupervised Learning, im Gegensatz zum Supervised Learning, in Echtzeit lernt. Das heißt, dass das Netzwerk nicht im Voraus trainiert werden muss. In Abbildung 7 sind die Unterschiede zwischen Supervised und Unsupervised Learning noch einmal dargestellt.

Reinforcement Learning:

Beim Reinforcement Learning lernt das Netzwerk selbständig und ohne gelabelte Daten. Allerdings gehört beim Reinforcement Learning auch immer eine Feedback-Schleife dazu. Wenn eine Aufgabe gut erledigt wurde, bekommt es positives Feedback. Macht es eine Aufgabe falsch erhält es entsprechend negatives Feedback. So lernt das Netz, welche Verbindungen zwischen Input und Output gut sind und welche nicht. In einer gewissen Weise ist das Reinforcement Learning ähnlich zum Lernen beim Menschen oder Tieren. Wenn wir Menschen eine erfolgreich gelöst haben und dafür positives Feedback erhalten, werden wir diese Aufgabe vermutlich öfters

[25] vgl. *Machine Learning: Algorithmen, Methoden und Beispiele* 2021; Mueller und Massaron 2019 S.29

so erledigen. Bei negativem Feedback werden wir, die Aufgabe vermutlich danach versuchen anders zu lösen.

Neben diesen Arten gibt es auch noch Sonder- und Mischformen, wie beispielsweise das Semi-Supervised oder das Self-Supervised Learning, auf die an dieser Stelle aber nicht näher eingegangen wird.

2.4. Verwandte Literatur zur Emotionserkennung

Die Methoden, die in den letzten Jahren genutzt wurden, um Emotionserkennung durchzuführen, können in zwei Gruppen eingeteilt werden: Zum einen gibt es die von Hand entworfenen Methoden und zum anderen die Methoden, welche neuronale Netze benutzen.

2.4.1. Von Hand erstellte Methoden

Bereits im Jahr 2007 wurde von Zhao et al.[26] das Local Binary Pattern (LBP) auf drei orthogonalen Ebenen angewendet. LBPs sind Muster, die beispielsweise für die Gesichtserkennung oder Altersbestimmung verwendet werden können. Alle umliegenden Pixel um einen anderen Pixel werden bei diesem Verfahren entweder zu einer 0 oder einer 1, je nachdem, ob der Wert des Pixels größer oder kleiner des ausgewählten Pixels in der Mitte ist. Danach wird aus diesen Nullen und Einsen eine Binärzahl gebildet, indem ein Auslesen im Uhrzeigersinn stattfindet. Die Umrechnung dieser Binärzahl in eine Dezimalzahl wird daraufhin der neue Wert des mittleren Pixels. Die Abbildung 8 demonstriert dieses Verfahren. Diese Methode wird auf jedem Pixel eines Bildes angewandt. Dadurch können bestimmte Muster erkannt werden. Durch das somit neu entstandene Local Binary Patter - Three Orthogonal Planes (LBP-TOP) wurde das LBP auf den dreidimensionalen Raum erweitert, sodass neben den räumlichen auch die zeitlichen Informationen erfasst

[26] G. Zhao und Pietikäinen 2007

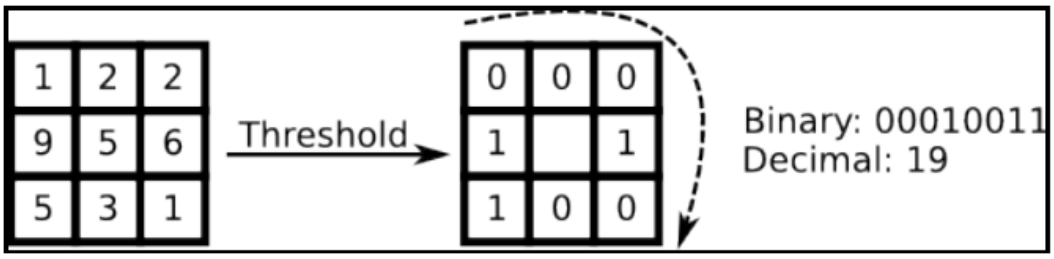

Abbildung 8.: Darstellung der Funktionsweise von LBP; Quelle: Fh-Bielefeld-Mif-Sw-Engineering-2017 2021

werden können. Polikovsky et al.[27] haben mit einer High-Speed-Kamera Bilder mit 200 frames per second (fps) aufgenommen. Auf diesen Bildern wurden die Gesichter anschließend in zwölf Regionen aufgeteilt, welche mit Blick auf das FACS ausgewählt wurden. Über diese Regionen wurde dann ein 3D-Gradient Histogram Descriptor benutzt, um Bewegungsinformationen zu erhalten.

Shreve et al.[28] haben eine Methode entwickelt, um anhand der Belastung der Gesichtshaut während einer Mimik die damit verbundene Emotion zu erkennen. Die Stärke der Belastung wurde mithilfe des Optical Flow in acht verschiedenen Unterregionen des Gesicht berechnet. Im Jahr 2011 wurde von Pfister et al.[29] ein Datensatz von spontanen Micro Expression entwickelt (mehr zum Datensatz selbst in Kapitel 3.3). Auf diesen Datzensatz wendeten sie dann einen Spatio-Temporal Local Texture Descriptor an. In diesem Fall wurde LBP-TOP benutzt, welche im Zusammenspiel mit verschiedenen Klassifikatoren wie Support Vector Machine (SVM), Multiple Kernel Learning (MKL) oder Random Forest (RF) zu einer Micro-Expression-Erkennung geführt haben. Auch Wu et al.[30] haben sich den SVM-Klassifikator zu Nutze gemacht, um Micro Expression zu erkennen. Zur Merkmalsextraktion kam in diesem Fall der Gabor-Filter zum Einsatz.

Wang et al.[31] nutzten ebenfalls SVM-Klassifikatoren. Zur Extraktion wurden hier mithilfe von Supervised Descent Method (SDM) zuerst 49 Referenzpunk-

[27] Polikovsky, Kameda und Ohta 2010
[28] Shreve u. a. 2011
[29] Pfister u. a. 2011
[30] Wu, Shen und Fu 2011
[31] Y. Wang u. a. 2015

te auf dem Gesicht lokalisiert. Dadurch konnten kleine Bereiche um diese Refernzpunkte extrahiert werden. Auf diese Bereiche wurde dann LBP-TOP angewendet.
Huang et al.[32] waren mit LBP-TOP jedoch nicht zufrieden. Sie waren der Meinung, dass bisher wichtige Informationen nicht berücksichtigt werden und dass oftmals klassische Mustertypen benutzt werden, die aber in manchen Anwendungssituationen als nicht optimal gelten. Aus diesem Grund schlugen sie SpatioTemporal Completed Local Quantization Patterns (STCLQP) für die Analyse von Micro Expressions vor. STCLQP extrahiert wichtige Informationen wie „sign, magnitude and orientation components“[33]. Darüber hinaus wird mit Vektorquantisierung und Codebüchern gearbeitet, um klassische Mustertypen zu generalisieren. Duan et al. [34] haben eine Framework entwickelt, um Micro Expression nur anhand der Augenregion zu erkennen. Hierfür wurden mit LBP-TOP Merkmale der Augenregion extrahiert. Danach wurden mehr als 20 Klassifikatoren mit diesen Merkmalen getestet und miteinander verglichen.

2.4.2. Deep-Learning-Methoden

In den letzten fünf Jahren wurde der Fokus allerdings immer mehr auf Deep-Learning-Methoden gerichtet. Die neuronalen Netze wurden dank der stetigen technischen Weiterentwicklung immer besser und schneller. Eine der ersten, die Deep-Learning- und Micro-Expression-Erkennung miteinander kombinierten, waren Patel et al.[35] Sie benutzten Transfer Learning von objekt- und mimikbasierten Convolutional Neural Networks (CNNs). Dies bedeutet, dass sie CNNs nutzten, die bereits für andere Aufgaben entwickelt worden waren. In diesem Fall für die Objekt- und Macro-Expression-Erkennung. Auch Peng et al.[36] haben ihr Netzwerk mit Transfer Learning trainiert. Das ResNet10 wurde mit dem ImageNet-Datensatz vortrainiert und anschließend mit

[32] Huang u. a. 2015
[33] Huang u. a. 2015
[34] Duan u. a. 2016
[35] Patel, Hong und G. Zhao 2016
[36] Peng, Zhan u. a. 2018

Micro-Expression-Datensätzen ein Feintuning durchgeführt. Madhumita et al. haben- um ihr CNN trainieren zu können- eine Data Augemntation der beiden Datensätze Chinese Academy of Science Micro-expression (CASME) und CASME II durchgeführt und so die Anzahl der Bilddaten verdoppelt.
Guo et al.[37] haben ein multi-modality CNN vorgestellt, das auf visuellen und geometrischen Informationen basiert. Dem CNN werden hier am Ende noch zusätzliche Daten aus einer parallel ablaufenden Facial-Landmark-Erkennung übergeben. Peng et al.[38] haben ein Dual Temporal Convolutional Neural Network (DTSCNN) vorgeschlagen, welches ein Zwei-Stream-Netzwerk darstellt. Diese Zwei-Stream-Methode wird in diesem Fall benutzt um sich an unterschiedliche Framerate des Inputs anzupassen.
Hasani et al.[39] haben ein Deep Neural Network (DNN) entwickelt, das aus zwei Teilen besteht. Im ersten Teil werden die räumlichen Relationen mit Convolutional-Schichten und drei Inception-ResNet Modulen verarbeitet. Der zweite Teil findet mithilfe von Linear Chain Conditional Random Field (CRF) die zeitliche Relation der Bilder. Auch die Arbeiten von Kim et al.[40], Khor et al.[41], Wang et al.[42] und erneut Kim et al.[43] machen sich eine zweiteilige Architektur zu Nutze. Im ersten Teil werden mithilfe eines CNN die räumlichen Informationen extrahiert. Diese Informationen werden anschließend als Featurevektor an ein Long Short-Term Memory (LSTM)-Modul weitergegeben, um daraus die zeitlichen Informationen zu gewinnen.
Liong et al.[44] haben ihrem CNN Optical-Flow-Daten eingespeist. Diese Optical-Flow-Daten bestehen aus der Zeit zwischen dem Beginn einer Micro Expression (dem Onset-Frame) und dem Höhepunkt des Ausdrucks (dem Apex-Frame). Zhao et al.[45] entwickelten einen neuen Datensatz, der eine Verbindung von mehreren Emotionen, wie beispielsweise glücklich überrascht,

[37] Guo u. a. 2017
[38] Peng, C. Wang u. a. 2017
[39] Hasani und Mahoor 2017
[40] Kim, Baddar und Ro 2016
[41] Khor u. a. 2018
[42] S.-J. Wang u. a. 2018
[43] Kim, Baddar, Jang u. a. 2017
[44] S.-T. Liong u. a. 2018
[45] Y. Zhao und Xu 2019

enthält. Aus diesen Daten wurden dann ebenfalls Optical-Flow-Merkmale zwischen Onset- und Apex-Frame extrahiert und von einem CNN verarbeitet. Auch Liong et al.[46] arbeiten mit den Optical-Flow-Daten, die dann an ein Shallow Triple Stream Three-dimensional CNN (STSTNet) weitergegeben werden. Jedes dieser drei Streams besitzt in diesem Fall unterschiedliche Kernelgrößen. Shahar et al.[47] gehen in ihrer Arbeit einen anderen Weg und bestimmen die Emotionen mithilfe der Farbänderung des Gesichts, die während einer Micro Expression auftritt. Diese Daten werden dann mit einem LSTM-Model verarbeitet, um die Emotion zu prognostizieren.

2.5. Bereits vorhandener Versuchsaufbau

Der Versuchsaufbau, der gegenwärtig beim Fraunhofer IOSB vorzufinden ist, wurde konzipiert, um in naher Zukunft Immersion messbar zu machen. Zum jetzigen Zeitpunkt kann der Zustand der Immersion nur mit Fragebögen erfasst werden. Da diese Fragebögen aber nicht während der eigentlichen Aktivität ausgefüllt werden können, sondern erst im Anschluss ausgehändigt werden, kann es bei der Beantwortung durchaus zu Abweichungen kommen. Denn es kann vorkommen, dass sich die Probanden nicht mehr daran erinnern können, ob sie in einem immersiven Zustand waren oder nicht.
Aus diesem Grund kam die Idee auf, Immersion und Flow mithilfe von physiologisch messbaren Daten sichtbar zu machen, um somit auch die Immersion selbst messen zu können. Solche physiologischen Daten sind zum Beispiel der Puls, die Leitfähigkeit der Haut, Muskelkontraktionen, Hirnströmungen oder auch die Mimik.
Bei der Mimik interessieren vor allem die Micro Expressions, da diese die wahren Emotionen eines Menschen offenbaren, wie bereits in Kapitel 2.1 erläutert wurde. Für Daten wie Puls und Muskelkontrationen gibt es bereits seit vielen Jahren geeignete Messinstrumente wie beispielsweise Elektromyografie (EMG), Elektroenzephalografie (EEG), Galvanic Skin Re-

[46] S. Liong u. a. 2019
[47] Shahar und Hel-Or 2019

sponse (GSR) und Elektrokardiogramm (EKG). In den letzten Jahren haben sich auch einige neue Möglichkeiten ergeben, um die Emotionen eines Menschen zu erkennen. Im Gebiet des Deep Learning gibt es einige passende Netzwerke, um eine Emotionserkennung durchzuführen. Allerdings sind diese Netzwerke meist nur auf die länger andauernden Macro Expressions ausgelegt worden. Im Gebiet zur Erkennung von Micro Expressions gibt es nach wie vor recht wenig neuronale Netze.

Auch beim neuronalen Netz, welches beim IOSB bereits im Einsatz ist, handelt es sich um ein Netzwerk, dass nur auf normale Emotionsausdrücke ausgelegt ist. Beim neuronalen Netz, welches von Gil Levi und Tal Hassner entwickelt wurde, handelt es sich um ein CNN, das abgebildete LBPs als Input verwendet. Im Grunde handelt es sich bei der Arbeit von Levi und Hassner nicht nur um ein Netzwerk, sondern um mehrere CNNs, die alle jeweils eine Emotion ermitteln und das Ergebnis den Durchschnitt der ermittelnden Emotionen darstellt. Als Input werden zugeschnittene und in Graustufen umgewandelte Bilder LBP codiert. Diese Codes werden wiederum auf verschiedene Weise im 3D-Raum abgebildet und zusammen mit den RGB-Daten als Input an die vier Netzwerke übergeben. Im Versuchsaufbau wird jedoch nur das RGB-Netzwerk verwendet.

Ein typischer Versuchsablauf besteht derzeit aus drei Phasen, wobei die erste Phase eine Briefing- und Setup-Phase ist. Die Testperson bekommt in dieser Phase eine genaue Erklärung über die Aufgabenstellung und wird an die verschiedenen Messgeräte angeschlossen. In der zweiten Phase spielt die Testperson für 30 Minuten ein, aus einer Auswahl frei wählbares Videospiel. Dies soll es dem Probanden ermöglichen, schneller in einen Immersions- oder Flowzustand zu gelangen, da ein ihm vertrautes Genre gespielt werden kann. Die dritte Phase ist die Beurteilungsphase. Die Testperson schaut sich ihre aufgezeichnete Sitzung im Nachhinein an und beantworten regelmäßig Fragebögen zur Flow- und Immersionsbestimmung. Abbildung 9 visualisiert diesen Aufbau. Mithilfe eines Programms zur Evaluation der Daten kann durch Korrelation der Emotionsdaten mit den physiologischen Daten und den Antworten der Fragebögen eine Vermutung angestellt werden, an welchen

Stellen im Video ein immersiver Zustand stattfindet.

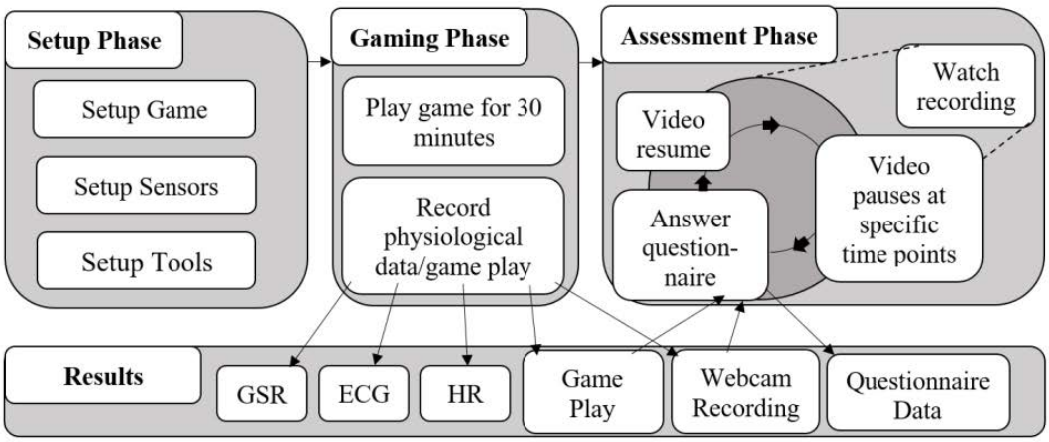

Abbildung 9.: Darstellung der drei Phasen des Versuchsaufbaus; Quelle: Kannegieser, Atorf und Herold 2021

3. Vorgehensweise

3.1. Erste Schritte

Zu Beginn der Arbeit bestanden die ersten Arbeitsschritte darin, den bereits vorhanden Versuchsaufbau zu verstehen und sich mit den Arbeitsabläufen vertraut zu machen. An dieser Stelle fielen schon die ersten Schwierigkeiten in der Handhabung auf. Der Ablauf der Emotionserkennung ist nicht leicht zu verstehen. Um zu einem Ergebnis zu kommen, sind drei verschiedene Programme zu benutzen. Zuerst werden die Daten durch die Benutzung des ersten Programms aufgezeichnet.
Die für die Emotionserkennung wichtigen Videoaufnahmen werden im Anschluss mithilfe des zweiten Programmes in Einzelbilder umgewandelt. Gleichzeitig findet an dieser Stelle eine Gesichtserkennung statt und das gefundene Gesicht wird ausgeschnitten. Um nun eine Vorhersage zur Emotion zu erhalten, werden diese ausgeschnittenen Gesichtsbilder an ein drittes Programm übergeben, welches dann die eigentliche Vorhersage tätigt. Die sequentielle Abarbeitung macht den Prozess relativ langwierig. Außerdem ist für das dritte Programm, das Netzwerk, eine virtuelle Maschine nötig, da das Netzwerk auf einer Linuxdistribution erstellt wurde.
Ein weitere Schwierigkeit war, dass die Dokumentation für die Nutzung des Netzwerks von Levi und Hassner sowie vom Programm zum Ausschneiden der Gesichter unvollständig ist. Um die jeweiligen Programme nutzen zu können, muss eine bestimmte Ordnerstruktur eingehalten werden, welche aus der Dokumentation nicht eindeutig hervorgeht. Diese Struktur wurde erst durch einen Blick in den Quellcode erkennbar. Eine weitere Herausforderung ergab sich, als das vorhandene Netzwerk mit Bildern aus Datensets an

Micro Expressions getestet wurde. Es wurde in den meisten Fällen nur eine neutrale Emotion vorhergesagt, da die Bewegungen bei Micro Expressions sehr kurz und subtil sind. Das Netzwerk hätte also von Grund auf neu trainiert werden müssen. Nach Rücksprache mit dem Autor des Netzwerkes riet dieser dazu, ein anderes Netz zu nutzen, da das verwendete Netz schon ein paar Jahre alt ist und nicht ohne weiteres auf Micro Expressions trainiert werden könne.

Bei der Suche nach einem neuen Netzwerk wurde vor allem darauf geachtet, welche Genauigkeit diese in ihrer Vorhersage haben. Dafür wurden die Werte betrachtet, die die Autoren selbst in ihren Papern angaben. Ein Netzwerk, das sehr vielversprechend aussah war *EmotionNet2*. Dieses Netzwerk baut auf der *ResNet*-Architektur auf und wurde durch einen Gesichtserkennungsalgorithmus erweitert. Nachdem das Netzwerk und die nötigen Voraussetzungen aufgesetzt waren, stellte sich allerdings heraus, das dieses Netz das selbe Problem wie das zuvor genutzte Netz von Levi und Hassner hatte. Auch hier wurden die Micro Expressions in den meisten Fällen als neutral bewertet.

Es musste erneut ein anderes Netzwerk gefunden werden. Das Hauptaugenmerk wurde darauf gelegt, dass das Netz bereits eine gute Performance bei der Erkennung von Micro Expressions vorweisen konnte. Das gewählte Netzwerk liefert laut dem Veröffentlichungspaper genau dies.

Dual-Stream Shallow Networks for Facial Micro-Expression Recognition (DSSN-MER)[48] besteht aus Zwei-Streams, die mit Optical FLow Daten als Input arbeiten. Aus dem Paper konnte entnommen werden, dass anhand der Micro-Expression-Datensätze hier eine bessere Performance geliefert wurde, als dies bei anderen Methoden der Fall war.

Obwohl der Quellcode des Netzes sehr lückenhaft und undurchsichtig war und nicht ersichtlich war, welche der vielen darin enthaltenen Modelle und Methoden vom Autor genutzt wurden, um die im Paper erwähnten Ergebnisse zu erzielen, wurde der erwartete zeitliche Mehraufwand der Implementierung an das Netzwerk in Kauf genommen. Die erwartete Performance schien dies zu rechtfertigen.

[48] IcedDoggie 2019

Als das Netzwerk dann implementiert war, ergab sich die nächste Herausforderung: Das Netzwerk konnte auf dem verwendeten Computer nicht genutzt werden, da die Grafikkarte keine Compute Unified Device Architecture (CUDA) besitzt. Bei CUDA handelt es sich um eine Schnittstelle, mit deren Hilfe sich auch die Rechenleistung der Grafikkarte für bestimmte Anwendung nutzen lässt. Zwar lassen sich die Netzwerke auch nur mit dem Prozessor trainieren, wenn diese passend umgebaut werden. Da der Code aber so undurchsichtig war und dies erneut sehr viel Zeit in Anspruch genommen hätte, fiel die Entscheidung, erneut ein neues Netzwerk zu suchen. Neben der guten Performance bei Micro Expressions war dieses Mal auch eine einfache Handhabung wichtig.
Das Netzwerk, das nun gefunden wurde liefert laut eigenen Aussagen eine sehr gute Performance und die Bedienung schien auf den ersten Blick einfach. Auf Grund der gemachten Erfahrungen haben sich zusätzliche Ziele ergeben: Zum einen soll das finale Produkt mit nur einem Programmaufruf funktionieren, um dem Anwender die Nutzung zu erleichtern. Außerdem soll das neue Skript Kommentare erhalten, die die verschiedenen Funktionen und Methoden erklärt. Eine einfache Anpassung der wichtigen Parameter soll dadurch ebenso ermöglicht werden, damit Nachfolgeprojekte besser weiterarbeiten können. Außerdem war ein Ziel, dass das Programm auch auf einer Windowsdistribution funktioniert und nicht in einer virtuellen Linux Umgebung laufen muss. Dies erleichtert vor allem die Eingliederung in das vorhandene System.

3.2. Das Netzwerk

Beim Netzwerk, das zuletzt gefunden wurde, handelt es sich um eine abgewandelte Form des MicroExpSTCNN[49]. Der Name bedeutet ausgeschrieben Micro Expression Spatiotemporal Convolutional Neural Network. Dieses macht sich ein dreidimensionales CNN zu Nutze, um neben den räumlichen Informationen zusätzlich noch zeitliche Informationen zu verarbeiten.

[49] Reddy u. a. 2019

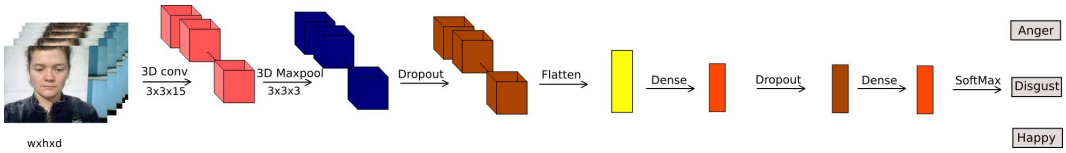

Abbildung 10.: Architektur des MicroExpSTCNN; Quelle: Reddy u. a. 2019

Das vorliegende Netzwerk ist ebenfalls darauf ausgelegt, dass es mit einer CUDA-fähigen Grafikkarte trainiert und genutzt wird. Da das vorhandene Skript jedoch deutlich leichter zu verstehen war, konnte das Netzwerk so umgebaut werden, dass es auch ohne eine solche Grafikkarte trainiert und benutzt werden kann. Hierfür musste am Modell selbst nur die Eingabereihenfolge für die Convolution3D-Schicht geändert werden. In Abbildung 10 ist die Architektur des Netzwerkes zu sehen.
Nachfolgend werden die einzelnen Schichten mit ihren Parametern kurz erklärt, um ein besseres Verständnis zu bekommen, wie das Netzwerk arbeitet. Im Quellcode 1 ist zu sehen, wie das Modell im Programm aufgebaut ist und welche Parameter vorhanden sind. In Zeile 155 des Quellcodes ist

```
184     model = Sequential()
185     model.add(Convolution3D(32, (3, 3, 15),
                input_shape=(image_rows, image_columns,
                image_depth, 1), activation='relu'))
186     model.add(MaxPooling3D(pool_size=(3, 3, 3)))
187     model.add(Dropout(0.5))
188     model.add(Flatten())
189     model.add(Dense(128, activation='relu'))
190     model.add(Dropout(0.5))
191     model.add(Dense(numberOfClasses))
192     model.add(Activation('softmax'))
```

Quellcode 1.: Das Modell

zu erkennen, dass es sich bei dem Modell um ein **sequenzielles** Modell handelt. Dies bedeutet, dass die Schichten nacheinander ausgeführt werden, wobei der Output der vorherigen Schicht immer der Input der nächsten Schicht ist. Zeile 156 zeigt, dass die erste Schicht des Modells eine **dreidi-**

mensionale Convolution-Schicht (deutsch: Faltung) ist. Ein Convolutional Neural Network ist ein neuronales Netzwerk, dass besonders gut bei Bilddaten funktioniert. Andere neuronale Netze erkennen beispielsweise in einem Datensatz aus Zahlen, die Verbindung aus dunklen Pixeln und der gezeigten Zahl. Wenn dann zu einem späteren Zeitpunkt ähnliche Werte beim neuronalen Netz ankommen, weiß dieses, dass es sich vermutlich um die selbe Zahl handelt. Bei Bildern ist dies allerdings im Normalfall ein wenig anders. „In an image, it is simply much less likely that an interesting feature will result from a combination of 9 randomly selected pixels throughout the image than from a 3 × 3 patch of adjacent pixels.“[50] Genau hier setzt die Faltung an. Man stelle sich vor, dass man ein Bild der Größe 5 × 5 Pixel hat. Die Bildmatrix würde dann wie folgt aussehen:

$$I = \begin{bmatrix} i_{11} & i_{12} & i_{13} & i_{14} & i_{15} \\ i_{21} & i_{22} & i_{23} & i_{24} & i_{25} \\ i_{31} & i_{32} & i_{33} & i_{34} & i_{35} \\ i_{41} & i_{42} & i_{43} & i_{44} & i_{45} \\ i_{51} & i_{52} & i_{53} & i_{54} & i_{55} \end{bmatrix}$$

Um aus diesem Bild bestimmte Eigenschaften zu erhalten ist zusätzlich noch eine zweite Matrix notwendig. Um einen 3 × 3 Bereich zu analysieren, benötigt man auch eine 3 × 3 Matrix.

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

Möchte man die Eigenschaft von Pixel i_{33} erhalten, so muss dafür einfach das Skalarprodukt der beiden Matrizen berechnet werden. Dies sieht folgendermaßen aus.

$o_{33} = w_{11} \cdot i_{22} + w_{12} \cdot i_{23} + w_{13} \cdot i_{24} + w_{21} \cdot i_{32} + w_{22} \cdot i_{33} + w_{23} \cdot i_{34} + w_{31} \cdot i_{42} + w_{32} \cdot i_{43} + w_{33} \cdot i_{44}$

[50] Weidman 2019 S. 128

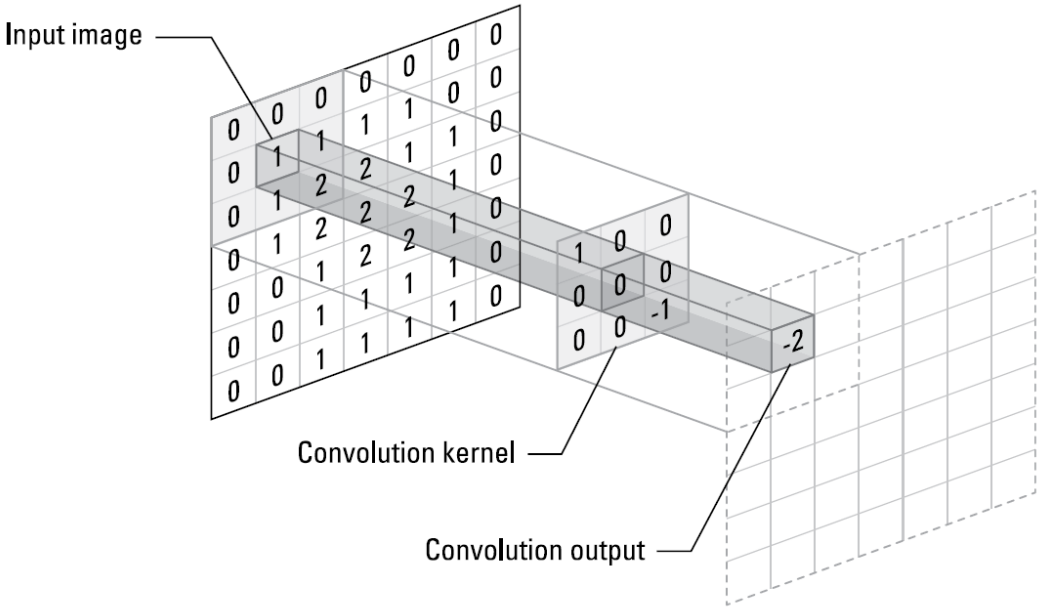

Abbildung 11.: Veranschaulichung einer Faltung; Quelle: Mueller und Massaron 2019 S.185

Abbildung 11 verdeutlicht diesen Schritt noch einmal grafisch. Aus der Computer Vision ist bekannt, dass man durch das Skalarprodukt spezieller Matritzen mit Bildpunkten erkennen kann, ob sich an dieser Stelle beispielsweise eine Kante oder Ecke befindet. Dies macht sich ein CNN zu Nutze und führt dieses Skalarprodukt an jedem Pixel des Ausgangsbildes aus. Am Ende erhält man so eine neue Matrix mit beinahe der selben Größe wie das Ausgangsbild. Diese Matrix ist eine sogenannte Featuremap, also eine Karte von Eigenschaften des Bildes. In diesem Beispiel würde die Featuremap wie folgt aussehen.

$$O = \begin{bmatrix} o_{22} & o_{23} & o_{24} \\ o_{32} & o_{33} & o_{34} \\ o_{42} & o_{43} & o_{44} \end{bmatrix}$$

In einer Convolution-Schicht gibt es allerdings nicht eine Matrix, die eine solche Featuremap erstellt, sondern eine beliebige Anzahl an Matritzen. Die Matrix, die das Skalarprodukt ausführt, wird auch Filter oder Kernel genannt (in Beispiel W). Im gefundenen Modell von Reddy et al. handelt es sich um eine dreidimensionale Convolution-Schicht, das bedeutet, dass sich der Ker-

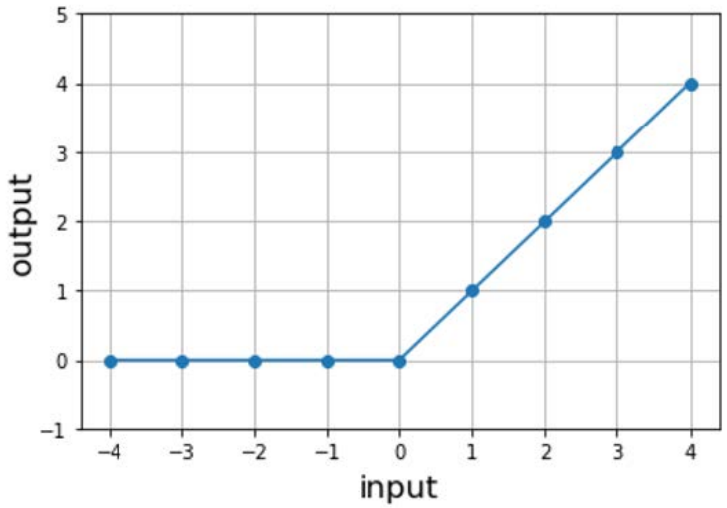

Abbildung 12.: Die ReLU-Funktion; Quelle: Mueller und Massaron 2019 S.174

nel nicht nur von links nach rechts und von oben nach unten durch die Input Matrix bewegt, sondern auch in die Tiefe geht. In Zeile 156 im Quellcode 1 kann erkannt werden, dass hier 32 Kernel mit einer Größe von 3 × 3 × 15 über die Inputmatrix bewegt werden.

Diese Inputmatrix hat, wie im Code zu sehen ist, die Form *image_rows, image_columns, image_depth, 1*. Image_rows und image_columns stehen hier für die Höhe und Breite des Bildes, welches in diesem Fall weiter oben im Skript auf 64 × 64 Pixel festgelegt wurde. Image_depth steht hier für die Anzahl an Bildern und somit auch für die Anzahl an Frames, die hier auf 96 festgelegt wurde. Die Zahl eins steht für die Anzahl an Bildkanälen. Bei einem Farbbild wäre dies auf Grund der drei Farbkanäle die Zahl drei. Hier wird also ein Schwarz-Weiß-Bild erwartet.

Der nächste Parameter ist der Parameter der Aktivierungsfunktion. Eine Aktivierungsfunktion hat die Aufgabe, einem Neuron zu sagen, wann ein Inputwert hoch oder niedrig genug ist, um einen Impuls abzugeben. Bei der Aktiverungsfunktion Rectified Linear Units (ReLU) handelt es sich um die Funktion $f(x) = max(x, 0)$, welche in Abbildung 12 zu sehen ist. Wie man sieht, ist die Funktion 0, solange x kleiner als 0 ist. Danach gleicht sie einer linearen Funktion. Die nächste Schicht im Modell ist eine **Pooling-Schicht**. Diese Schicht ist dazu da, den Datenfluss des Netzwerkes zu minimieren.

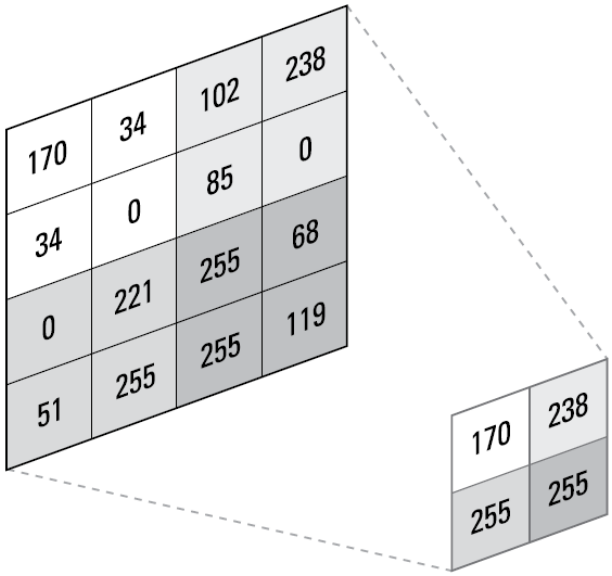

Abbildung 13.: Veranschaulichung der MaxPooling-Schicht; Quelle: Mueller und Massaron 2019 S.187

Wie schon bei der Faltung wird eine Art Fenster über die Inputdaten bewegt. Im Falle einer **MaxPooling-Schicht** wird in diesem Fenster die größte Zahl gesucht. Dadurch wird die Outputmatrix deutlich kleiner und die Daten werden signifikant verringert. In Abbildung 13 ist dieser Prozess dargestellt. Dort besteht die Datenmenge nach dem Pooling aus nur noch einem Viertel der vorherigen Schicht. Im Modell von Reddy et al. ist dieses Fenster der MaxPooling-Schicht 3 × 3 × 3 groß.
Die nächste Schicht ist die **Dropout-Schicht**. Hier werden je nach Parameter eine gewisse prozentuale Anzahl an Neuronen abgeschaltet. Bei einem Wert von 0.5 werden 50 Prozent der Neuronen auf 0 gesetzt. Dies hilft dem Netzwerk dabei, ein Overfitting zu verhindern. Overfitting bedeutet, dass ein Netzwerk zwar sehr gut mit den trainierten Daten funktioniert, aber mit neuen unbekannten Daten Probleme hat. Das Netzwerk lernt hier also Eigenschaften, die nur im gegebenen Datensatz vorhanden und in der eigentlichen Anwendung gar nicht von Bedeutung sind. Nach dieser Dropout-Schicht folgt die **Flatten-Schicht**. Hier wird eine mehrdimensionale Inputmatrix in einen eindimensionalen Outputvektor umgewandelt. In der nächsten Schicht, der **Dense-Schicht**, werden nun alle Neuronen des Inputs (der eben erzeugte Vektor) mit allen Neuronen des Outputs verbunden. Aus diesem Grund

wird diese Schicht auch als fully-connected bezeichnet. Die Anzahl der Outputneuronen wird im Quellcode in der Parameterklammer definiert. Beim vorgestellten Modell beträgt diese Zahl 128 Neuronen mit der Aktivierungsfunktion ReLU. Durch diese vollständige Verknüpfung lernt das Netzwerk Verbindungen zwischen den verschiedenen Eigenschaften, welche durch die Faltung erhalten wurden, zu finden. Die Faltung ist also in gewisser Weise für die Extraktion der Eigenschaften zuständig und die Dense-Schichten für das eigentliche Lernen. Nach der ersten Dense-Schicht folgt ein erneuter Dropout mit 50 Prozent. Auf diese Schicht wird eine zweite Dense-Schicht benutzt, welche als Anzahl der Outputs die Anzahl an möglichen Klassen haben muss. Im Fall der Emotionserkennung sind dies drei beziehungsweise vier Klassen (siehe hierzu die einzelnen Trainingssets in Kapitel 3.4). Damit muss auch die letzte Dense-Schicht als Output drei beziehungsweise vier Neuronen besitzen. Als letzte Schicht folgt eine **Softmax-Aktivierungs-Schicht**. Es handelt sich hier um die Aktivierungsfunktion Softmax. Diese Funktion gibt die Wahrscheinlichkeit an, die den jeweiligen Klassen am Ende zugeschrieben wird.

3.3. Datensets

Um ein Netzwerk trainieren zu können, benötigt man vor allem eines: Daten. Aus diesem Grund wurden mehrere Micro-Expression-Datensätze ausfindig gemacht, welche nachfolgend erklärt werden. Dies sind zum einen die drei chinesischen Datensätze CASME, CASME II sowie CAS(ME)^2 und zum anderen der finnische Datensatz Spontaneous Micro-expression Database (SMIC) beziehungsweise SMIC-E

3.3.1. SMIC

Im Jahr 2011 wurde der Datensatz SMIC entwickelt, um mit Machine-Learning-Algorithmen ein System zu bauen, das Micro Expressions entdecken kann.

Dieser Datensatz bestand allerdings aus nur sechs Teilnehmern[51], sodass im Jahr 2013 eine neue Version des Datensatzes veröffentlicht wurde. Der aktualisierte Datensatz enthielt 164 Micro Expressions von 16 verschiedenen Teilnehmern[52]. Der ursprüngliche SMIC Datensatz war einer der ersten Micro-Expression-Datensätze. Zuvor gab es zwar schon etliche Datensätze für normale Gesichtsausdrücke, den Micro Expressions wurde allerdings lange keine große Beachtung geschenkt. Ein Problem, was die Autoren von SMIC in den bisher vorhanden Datensätzen sahen, war, dass die Emotionen oft gestellt und somit nicht natürlich waren. Die Teilnehmer wurden dazu aufgefordert, bestimmte Emotionen nachzuahmen.

Wie in Kapitel 2.2 bereits erklärt wurde, handelt es sich bei Micro Expressions vor allem um Emotionen, die zum einen sehr kurz auftreten und zum anderen, entweder bewusst oder unterbewusst, unterdrückt werden. Um dies zu erreichen, wurden 16 Videoclips ausgesucht, die verschiedene Emotionen in den Teilnehmern auslösen sollten. Außerdem wurde ihnen gesagt, dass sie während dem Betrachten der Videoclips versuchen sollen, ihre Emotionen zu verstecken. Man erzählte den Teilnehmer zudem, dass falls ihnen dies nicht gelingen sollte, ein langweiliger, 500 Seiten langer Fragebogen auf sie warten würde. Die Videoaufnahmen der Teilnehmer wurden anschließend von zwei Annotatoren im Einklang mit den, von den Teilnehmern selbst zurückgemeldeten Emotionen gelabelt. Nur Videos, bei denen sich die beiden Annotatoren einig waren, wurden am Ende in den Datensatz aufgenommen. Außerdem durfte die Gesamtlänge der Emotion nicht länger als eine halbe Sekunde sein, da es sich sonst nicht mehr um eine Micro Expression handeln würde.

Die Videos wurden in drei Klassen eingeteilt: positiv, negativ und überrascht. Der Datensatz besteht dadurch aus 164 Micro-Expression-Clips von 16 verschiedenen Teilnehmern, welche mit 100 fps aufgenommen und anschließend in drei Gruppen aufgeteilt wurden. Acht Teilnehmer (71 Videoclips) wurden zusätzlich noch mit einer 25 fps Kamera und einer Nahinfrarotkamera

[51] Pfister u. a. 2011
[52] Li, Pfister u. a. 2013

aufgenommen. Der SMIC Datensatz besteht also aus drei verschiedenen Datensätzen. 2015 wurde zudem SMIC-E veröffentlicht[53]. Hierbei handelt es sich im Grunde um die selben Clips, die auch im SMIC Datensatz zu finden sind. Jedoch sind die Clips in einer längeren Variante vorhanden. Zuvor waren nur die eigentlichen Emotionen selbst vorhanden. In SMIC-E sind zudem die Frames vor und nach jeder Emotion enthalten. Dies soll es Algorithmen ermöglichen, eine bessere Emotionsdetektion durchzuführen.

3.3.2. CASME

Auch Wu et al., die Autoren hinter dem CASME-Datensatz, waren von der Situation der verfügbaren Datensätze nicht überzeugt. Sie sahen vor allem drei Probleme: Wie auch schon die Autoren des SMIC-Datensatzes waren sie der Auffassung, dass gestellte Emotionen sich deutlich von spontanen Emotionen unterscheiden. Außerdem enthalten die anderen Datensätze auch unemotionale Bewegungen wie beispielsweise Schlucken oder Augenrollen. Als Drittes sahen sie es als problematisch an, dass die Labels der Daten unpräzise waren. Aus diesem Grund veröffentlichten sie im Jahr 2013 den CASME-Datensatz[54]. Der verbesserte SMIC-Datensatz war zu diesem Zeitpunkt noch nicht veröffentlicht worden.
Der Datensatz selbst besteht aus 195 Micro Expressions von 35 Teilnehmern und ist in zwei Teile untergliedert. In Part A wurden die Teilnehmer mit 60 fps in natürlichem Licht gefilmt. In Part B wurde ebenfalls mit 60 fps gefilmt, jedoch kam das Licht dieses Mal von zwei LED-Lichtern. Die Teilnehmer schauten sich, wie auch schon im SMIC-Datensatz, mehrere Videos an, welche bestimmte Emotionen hervorrufen sollten. Beim anschließenden Labeln der Emotionen wurde sich in diesem Datensatz anhand der AUs des FACS orientiert. Darüber hinaus wurde auch die Rückmeldung der Teilnehmer sowie der Inhalt des geschauten Videos in die Beurteilung miteinbezogen. In Abbildung 14 ist zu sehen, welche AUs für die jeweilige Emotion vorhanden sein muss. Die dritte Spalte zeigt zudem, wie oft diese Emotion im Datensatz

[53] Li, Hong u. a. 2015
[54] Yan, Wu u. a. 2013

Emotion	Criteria	N
Amusement	Either AU6 or AU12 must be present	5
Sadness	AU1 must be present	6
Disgust	At least one of AU9, AU10, AU4 must be present	88
Surprise	AU1+2, AU25 or AU2 must be present	20
Contempt	Either unilateral AU10 or unilateral AU12 be present	3
Fear	Either AU1+2+4 or AU20 must be present	2
Repression	AU14, AU15 or AU17 is presented alone or in combination	40
Tense	Other emotion-related facial movements	28

Abbildung 14.: Kriterien für die Beschriftung von AUs im CASME-Datensatz; Quelle: Yan, Wu u. a. 2013

vorkommt. Um die unemotionalen und irrelevanten Bewegungen zu entfernen, wurden die Teilnehmer nach ihren gewohnheitsmäßigen Bewegungen befragt. Da Micro Expressions bewusst oder unterbewusst unterdrückte Emotionen sind, wurde den Teilnehmern gesagt, dass sie versuchen sollen, keine Emotionen zu zeigen. Falls dies doch passiere, würde ein Teil ihrer Entlohnung einbehalten.

3.3.3. CASME II

Im Jahr 2014 wurde der Datensatz CASME II veröffentlicht.[55] Grund für die Veröffentlichung eines weiteren Datensatzes war, dass den Autoren die bisher vorhandenen Datensätze immer noch eine zu kleine Menge an Daten enthielt. Zudem wurde die Qualität der bisherigen Videoaufnahmen als nicht den Maßstäben entsprechend angesehen. Deshalb wurden die Videoaufnahmen im CASME II-Datensatz mit 200 fps und einer höheren Auflösung aufgenommen. Der Versuchsaufbau selbst war dem des Vorgängers sehr ähnlich. Dieses Mal gab es insgesamt 35 Teilnehmer, wovon 18 dazu aufgefordert wurden zu versuchen, einen neutralen Gesichtsausdruck zu bewahren. Die übrigen 17 Teilnehmer sollten versuchen, ihre Gesichtsbewegungen zu unterdrücken, wenn sie das Gefühl hatten, dass ein Gesichtsausdruck aufkommt. Grund hierfür ist, dass man versucht, beide Arten von Micro Expressions zu erhalten. Der Beschriftungsprozess ist im Grunde der selbe wie schon bei

[55] Yan, Li u. a. 2014

Emotion	Criteria	N
Happiness	either AU6 or AU12	33
Disgust	one of AU9, AU10 or AU4+AU7	60
Surprise	AU1+2, AU25 or AU2	25
Repression	AU15 or AU17 alone or in combination	27
Others	Other emotion-related facial movements	102

Abbildung 15.: Kriterien für die Beschriftung von AUs im CASME II-Datensatz; Quelle: Yan, Li u. a. 2014

CASME. Lediglich die Einteilung der Klassen unterscheidet sich. In diesem Datensatz gibt es nur fünf Klassen, wie man auch in Abbildung 15 erkennen kann. Insgesamt ergaben sich so 247 Micro Expressions.

3.3.4. CAS(ME)^2

Drei Jahre nach der Veröffentlichung des CASME II-Datensatzes wurde der dritte Datensatz CAS(ME)^2 publiziert[56]. Dieser Datensatz enthält wie auch SMIC-E längere Videos, um eine bessere Detektion zu gewährleisten. Neben Micro Expressions befinden sich auch Macro Expressions in diesem Datensatz. Dies soll ebenfalls bei der Entwicklung von passenden Algorithmen helfen. Die Daten sind in CAS(ME)^2 in zwei Bereiche unterteilt. In Teil A befinden sich 87 lange Videos, die sowohl Micro als auch Macro Expressions enthalten. Im zweiten Teil, Teil B, befinden sich schon zugeschnittene Bilder. Von den 22 Teilnehmern wurden hier insgesamt 300 Macro und 57 Micro Expressions gesammelt. Im Gegensatz zu den anderen beiden vorherigen Datensätzen war die Abtastrate mit 30 fps deutlich geringer. Zudem gibt es nur noch vier Emotionskategorien. Wie schon beim Datensatz SMIC gibt es positiv, negativ, überrascht und zusätzlich noch die Kategorie Andere (siehe hierzu auch Abbildung 16).

[56] Qu u. a. 2017

Emotion Category	Criteria	Number	Macro-expression	Micro-expression
Positive	AUs needed for Happiness, At least AU6 or AU12 was present	124	116	8
Negative	AUs needed for Anger, Disgust, Sadness, Fear	126	105	21
Surprise	At least AU 1+2, AU 25, or AU 2 was present	25	16	9
Others	Other facial movements*	82	63	19

Abbildung 16.: Kriterien für die Beschriftung von AUs im CAS(ME)^2 Datensatz; Quelle: Qu u. a. 2017

3.4. Trainingssets

Wie bereits in Kapitel 3.2 erwähnt, musste das ursprüngliche Netzwerk leicht umgebaut werden, um mit einer Central Processing Unit (CPU) genutzt werden zu können. Um das Netzwerk zu trainieren, wurden zudem noch weitere Änderungen vorgenommen, damit es mit dem Aufbau des Skriptes in Kapitel 3.6 im Einklang ist. Bevor es zum Training des Netzes kommt, müssen zuerst ein oder auch mehrere Trainingssets aus Daten erstellt werden. Im Gegensatz zu den Autoren des Netzwerkes bestehen die in dieser Arbeit erstellten Trainingssätze nicht aus den Videos selbst, sondern aus den in den Datensätzen bereitgestellten einzelnen Frames. In der Readme-Datei des ersten CASME-Datensatzes wird davor gewarnt, dass es zu unvorhergesehenen Problemen kommen kann, wenn man die Videos selbst in einzelne Bilder konvertiert. Im Grunde wurden aus den zur Verfügung stehenden Daten aus den Datensätzen vier verschiedene Trainingssets abgeleitet.

3.4.1. Trainingsset 1

Während das Netzwerk ursprünglich nur mit dem CAS(ME)^2-Datensatz trainiert wurde, besteht das erste Trainingsset aus allen CASME-Datensätzen, die in Kapitel 3.3 aufgeführt sind. Um die Daten über alle Datensätze konsistent zu halten, wurden jedoch nur die Videos beziehungsweise Bilder

ausgewählt, bei denen 96 oder mehr Frames vorhanden waren. Ebenso wurden die Klassen nicht mehr mit *Anger, Disgust* und *Happy* gelabelt, sondern mit *Positive, Negative* und *Surprise*, da auch hier jeweils unterschiedliche Kodierungen vorhanden waren. Im CASME- und CASME II-Datensatz wurden die Emotionen *Disgust, Sadness* und *Fear* zur negativen Klasse zusammengefasst. Die Emotionen *Repression, Tense* und *Others* wurden nicht für das Trainingsset berücksichtigt. Mit diesen Rahmenbedingungen ergaben sich so im ersten Trainingsset insgesamt 429 Emotionen. Die genaue Aufteilung ist in Tabelle 1 dargestellt.

	Negative	Positive	Surprise
CASME	30	10	12
CASME II	66	28	26
CAS(ME)²	119	114	24

Tabelle 1.: Anzahl der Emotionen im ersten Trainingsset

3.4.2. Trainingsset 2

Im zweiten Trainingsset ist den Daten aus dem ersten Set eine zusätzliche vierte Klasse mit dem Namen *Neutral* hinzugefügt worden. Da in den Datensätzen keine explizite Kodierung für neutrale Ausdrücke vorhanden ist, wurden Frames genommen, denen keine Emotionen zugeordnet wurde. Meist wurden dafür die Frames am Anfang des Videos verwendet. Es wurden außerdem nicht alle möglichen Frames in das Trainingsset aufgenommen, da dies eine sehr große Menge an Frames ergeben hätte. Zusätzlich kamen durch die Addition einer vierten Klasse noch 110 neutrale Gesichtsausdrücke hinzu, sodass das Trainingsset im zweiten Durchgang insgesamt aus 539 Emotionen beziehungsweise Gesichtsausdrücken besteht. In Tabelle 2 ist die genaue Verteilung zu erkennen.

3.4.3. Trainingsset 3

Das dritte Trainingsset beinhaltet neben den Bildern aus dem ersten Set zudem die Bilder aus dem SMIC-Datensatz. Aus den Daten ließen sich 173

	Negative	Positiv	Surprise	Neutral
CASME	30	10	12	0
CASME II	66	28	26	33
CAS(ME)²	119	114	24	77

Tabelle 2.: Anzahl der Emotionen im zweiten Trainingsset

zusätzliche Emotionen gewinnen. Insgesamt befinden sich dadurch 602 Gesichtsausdrücke in diesem Set. Das Set wurde nur in der Variante mit drei Klassen erstellt, da sich bei den ersten Versuchen die Variante mit vier Klassen als deutlich schlechter herausgestellt hat. Die genaue Verteilung der jeweiligen Gesichtsausdrücke ist in Tabelle 3 zu sehen

	Negative	Positiv	Surprise
CASME	31	9	15
CASME II	66	28	26
CAS(ME)²	120	121	24
SMIC	70	50	42

Tabelle 3.: Anzahl der Emotionen im dritten Trainingsset

3.4.4. Trainingsset 4

Das letzte Set ist ein Set, welches nur aus den Daten aus dem CAS(ME)²-Datensatz besteht, wie dies auch im Paper des Netzwerkes der Fall war, da dies der größte der vorhandenen Datensätze war. So ergaben sich in diesem Trainingsset nur noch 265 Gessichtsausdrücke wie in Tabelle 4 zu erkennen ist.

	Negative	Positiv	Surprise
CAS(ME)²	120	121	24

Tabelle 4.: Anzahl der Emotionen im vierten Trainingsset

3.5. Versuche

Mit den verschiedenen Trainingssets wurden verschiedene Versuche beziehungsweise Trainings durchgeführt mit dem Ziel eine möglichst genaue

Prognose der Micro Expressions zu treffen. Dazu wurden unterschiedliche Parameter, wie beispielsweise die Abtastrate oder Anzahl der Klassen verändert. Ein wichtiger Befehl aus dem Skript ist der Befehl *model.compile*.

```
193       model.compile(loss =
                'categorical_crossentropy', optimizer =
                'SGD', metrics = ['accuracy'])
```

Quellcode 2.: Der model.compile-Befehl

An dieser Stelle wird das Modell für das Training kompiliert. In der Parameterklammer wird angegeben, welche Verlustfunktion, welchen Optimierer und welche Metriken verwendet werden sollen. Bei einer Verlustfunktion handelt es sich um eine Funktion, die angibt, wie weit ein vom Netzwerk prognostizierter Wert vom tatsächlichen Wert entfernt ist. Bei der Categorical-Crossentropy-Verlustfunktion wird dabei nur der Verlust der tatsächlichen Klasse betrachtet. Der Optimierer reagiert auf diesen Verlust und optimiert das Modell, indem es Änderungen an der Gewichtung vornimmt. Mit dem Metrik-Parameter wird festgelegt, anhand welcher Werte das Modell bewertet werden soll. Im vorgeschlagenen Modell geschieht dies anhand der Genauigkeit der Validierungsbilder, der sogenannten Validierungsgenauigkeit.
Vom Trainingsset werden achtzig Prozent der Daten für das eigentliche Training verwendet. Mit den übrigen Daten wird dann am Ende eine Validierung durchgeführt. Insgesamt wurde das Netz 100 Epochen lang traininert. Dies bedeutet, dass über die gesamten Inputdaten 100 Mal iteriert wird. Am Ende einer solchen Epoche wird mit den Validierungsdaten eine Prognose durchgeführt. Output eines solchen Trainingsvorgangs ist eine Datei, welche die ermittelten Gewichte des besten Durchgangs beinhaltet.
Zudem wird am Ende des Trainings eine Confusionmatrix erstellt. Diese gibt an, wie oft das Netzwerk richtig lag und wie oft es sich falsch entschieden hat, indem die wahren Labels mit den Prognosen des Netzes verglichen werden. Aus dieser Matrix lassen sich dann weitere Daten berechnen, um

ein Netzwerk zu evaluieren. Eine dieser Daten ist der F1-Score. Dieser wird mit der Formel $2 * \frac{presicion*recall}{presicion+recall}$ berechnet. Die Presicion gibt an, wie viel Prozent der Prognosen tatsächlich diese Klasse sind. In einer Formel lässt sich dies folgendermaßen festhalten: $\frac{TruePositives}{TruePositives+FalsePositives}$. Recall dagegen beschreibt, wie viel Prozent der Prognoses aus der Gesamtzahl an möglichen richtigen Prognosen stimmen. Die Formel $\frac{TruePositives}{TruePositives+FalseNegatives}$ beschreibt dies. Der F1-Score ist das harmonische Mittel dieser beiden Formeln.

3.5.1. Versuch 1

Der erste Versuch wurde mit dem ersten Trainingsset abgehalten. Im Vorverarbeitungsprozess der Bilder wurde neben der Umwandlung in Graustufen zudem eine Gesichtserkennung durchgeführt. Die erkannten Bilder wurden auf die Größe 64 × 64 Pixel skaliert und in die Inputmatrix überführt. Die Gesichtserkennung erfolgte mit dem Haar-Cascade-Classifier. Dieser erste Versuch besteht somit aus insgesamt 429 Emotionen mit je 96 Bildern, welche zugeschnitten und in drei Klassen untergliedert wurden. Das Ergebnis des ersten Versuchs hat eine Validierungsgenauigkeit von 79 Prozent. Dies entspricht allerdings einer Differenz von acht Prozent zu den Ergebnissen des Autors. Wie an der Confusionmatrix (Abbildung 17) zu erkennen ist, prognostiziert das Netzwerk die meisten Daten richtig. Bei der Klasse 0 handelt es sich um die negative Emotion, Klasse 1 ist die positive Emotion und Klasse 2 steht für überrascht. Es ist zu sehen, dass das Netz vor allem bei der Erkennung von Überraschung Schwierigkeiten hat. Der Recallwert beträgt hier nur 50 Prozent. Aber auch bei den positiven Emotionen ist Verbesserungspotenzial vorhanden. Der F1-Score dieses Versuchs liegt insgesamt bei 71,7 Prozent.

3.5.2. Versuch 2

Der zweite Versuch wurde mit den selben Parametern und dem selben Vorverarbeitungsprozess durchgeführt. Allerdings wurde nun auch das zweite Trainingsset benutzt, das heißt, dass nun vier anstatt drei Klassen als Mög-

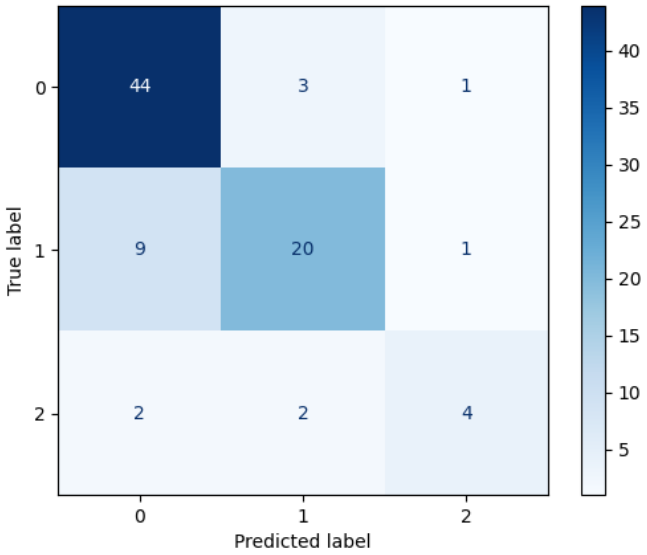

Abbildung 17.: Confusion Matrix - Versuch 1

lichkeit zu Verfügung standen. Dies war aber der einzige Unterschied der beiden Versuche. Insgesamt schneidet dieses Set deutlich schlechter ab als das erste Set mit nur drei Klassen. In der Variante mit vier Klassen gab es nur eine 61-prozentige Validierungsgenauigkeit. Ein Grund hierfür kann sein, dass die neutralen Bilder nicht sorgfältig genug gewählt wurden und nicht zu hundert Prozent neutral sind. Da aber keine neutralen Labels vorhanden waren, war dies vorerst nicht zu ändern. Betrachtet man wieder die Confusionmatrix (Abbildung 18), so ist gut zu erkennen, dass die eigentlich neutralen Emotionen (Klasse 3) vom Netz sehr häufig als negative Emotion angesehen wurden. Und auch die Überraschung ist nach wie vor ein Problem dieses Netzes. Der F1-Score beträgt in diesem Fall nur 55 Prozent.

3.5.3. Versuch 3 und 4

Versuche drei und vier wurden ebenfalls mit den Trainingssets eins und zwei durchgeführt. Im Gegensatz zu den vorherigen Versuchen wurden dieses Mal keine zugeschnittenen Gesichtsbilder als Input weitergeben, sondern das komplette Bild. Dieses wurde dann auch in Graustufen umgewandelt und auf die Größe von 64 × 64 Pixeln skaliert, sodass es den Anforderungen

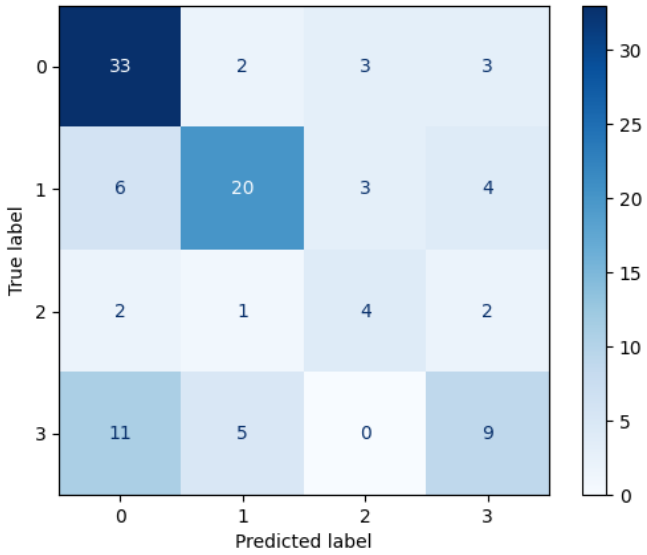

Abbildung 18.: Confusion Matrix - Versuch 2

des Modells entspricht. Im Falle des Versuchs mit drei Klassen brachte diese Umstellung eine Steigerung von vier Prozent ein. Bei der Variante mit vier Klassen blieb die Validierungsgenauigkeit bei 61 Prozent. Aufgrund der Erkenntnisse dieser beiden Versuche wurden die Nächsten nur noch mit drei Klassen durchgeführt und auch der Zuschnitt auf die Gesichter wurde nicht mehr verwendet. Bei Versuch drei ist anhand der Confusionmatrix (Abbildung 19) zu sehen, dass im Vergleich zum ersten Versuch nun die positiven Emotionen besser erkannt wurden. Der Recallwert stieg dadurch auch auf 83 Prozent an. Die überraschende Emotion konnte dafür aber noch schlechter erkannt werden und kommt nur auf einen Recallwert von 37 Prozent. Dies spiegelt sich auch im F1-Score wieder, der mit 71,4 Prozent leicht unter Versuch eins liegt. Auch die Confusionmatrix des vierten Versuchs (Abbildung 20 zeigt, dass sich die Erkennung von positiven Emotionen gesteigert hat. In diesem Fall wurden die überraschten Emotionen allerdings besser erkannt als in Versuch zwei. Allerdings hat sich die Erkennung der neutralen Emotion deutlich verschlechtert, sodass der F1-Score bei 55 Prozent blieb

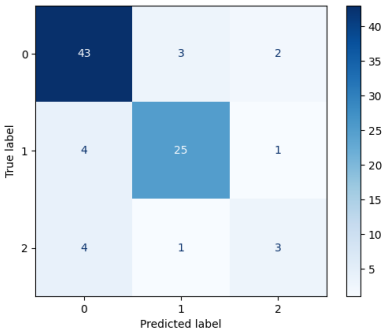

Abbildung 19.: Confusion Matrix - Versuch 3

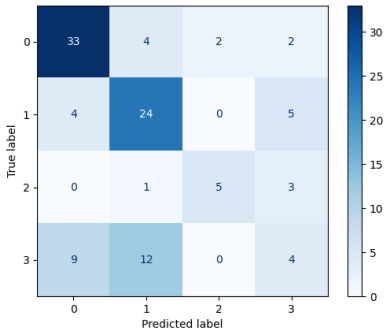

Abbildung 20.: Confusion Matrix - Versuch 4

3.5.4. Versuch 5

Da die CASME-Datensätze bereits ordentliche Ergebnisse eingebracht hatten, wurde nun auch mit Trainingsset drei ein Versuch durchgeführt. Durch das Hinzufügen des SMIC-Datensatzes wurde keine Verbesserung erzielt. Das Ergebnis verschlechterte sich auf eine 69-prozentige Validierungsgenauigkeit. Die Problematik hierbei lag vermutlich daran, dass die Datensätze alle unterschiedliche Parameter besitzen. Die Auflösung ist bei allen Datensätzen unterschiedlich und auch die Framerate ist nicht überall die selbe. Dies spiegelt sich auch in der Confusionmatrix wieder (Abbildung 22). Im Vergleich zu Versuch eins und drei gab es eine Verschlechterung in allen drei Emotionsklassen, sodass der F1-Score nur noch 65 Prozent beträgt.

3.5.5. Versuch 6

Eine weitere Überlegung war es, die Länge der Sequenz zu verringern, um im späteren Anwendungsfall kürzere Zeitabstände einzufangen. Die Menge der Daten wurde von 96 auf 60 Bilder verringert, da dies im Falle einer Aufnahme von 120 oder 240 fps einer Dauer von einer viertel oder halben Sekunde entspricht. Diese halbe Sekunde ist wie in Kapitel 2.2 beschrieben, die Länge bis zu der eine Emotion noch als Micro Expression zählt. Jede Emotion, die länger ist, wird als Macro Expression angesehen. Dieser Versuch wurde wie die vorherigen Tests auch mit den selben Parametern des Netzwerks durchgeführt. Hier kam es im Vergleich zur Variante mit 96 Frames zu keiner

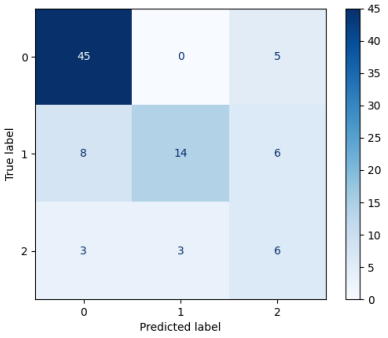

Abbildung 21.: Confusion Matrix - Versuch 6

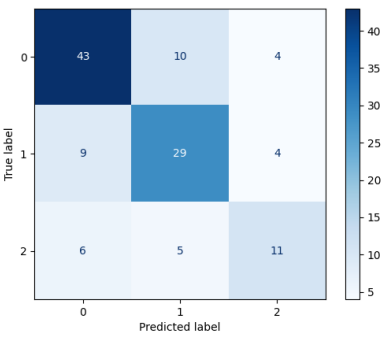

Abbildung 22.: Confusion Matrix - Versuch 5

Verbesserung. Die Validierungsgenauigkeit liegt nur noch bei 72 Prozent. Woran diese Verschlechterung liegt, kann man an der Confusionmatrix in Abbildung 21 erkennen. Die negative Emotion wurde immer noch sehr gut prognostiziert. Der Recallwert liegt bei 90 Prozent. Allerdings hat das Netz nun öfter die überraschte Emotion vorhergesagt. Leider war dies nur in 35 Prozent der Fälle richtig, wie der Precisionwert zeigt. Der F1-Score beträgt bei diesem Versuch 62,8 Prozent.

3.5.6. Versuch 7 und 8

Da die Inputmatrix im vorherigen Versuch durch das Verringern der dritten Dimension kleiner war, kam der Gedanke auf, auch die Kernelgröße anzupassen. In einem weiteren Versuch wurde die Kernelgröße der Pooling-Schicht von 3 × 3 × 3 auf 2 × 2 × 2 reduziert. Tatsächlich lieferte dies eine Verbesserung von vier Prozent. Trotzdem war der neue Wert von 76 Prozent unter der bisherigen Bestmarke von 83 Prozent in Versuch drei. Die Verbesserung von vier Prozent zwischen Versuch sechs und sieben, durch die Verringerung der Kernelgröße, führte dazu, dass in Versuch acht noch einmal der dritte Versuch wiederholt wurde. Dieses Mal jedoch ebenfalls mit der verringerten Kernelgröße von 2 × 2 × 2. Es kam jedoch zu keiner Verbesserung und die Validierungsgenauigkeit blieb bei 83 Prozent. In Abbildung 23 ist zu sehen, wo die Verbesserung in Versuch acht auf einen F1-Score von 67,4 Prozent stattfand. Wenn die überraschte Emotion prognostiziert wurde,

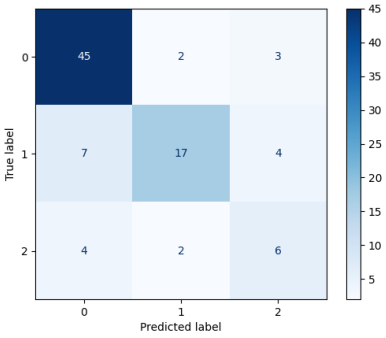

Abbildung 23.: Confusion Matrix - Versuch 7

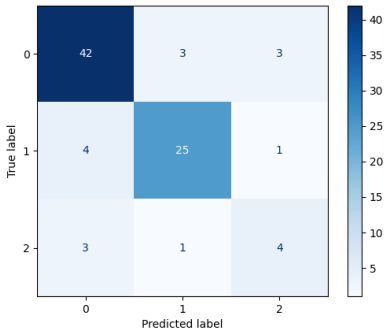

Abbildung 24.: Confusion Matrix - Versuch 8

war dies nun um zehn Prozent genauer. Der Presicionwert beträgt nun 46 Prozent. Auch der Recallwert der positiven Emotion ist um zehn Prozent auf 60 gestiegen. Die Confusionmatrix von Versuch sieben sieht der Matrix der dritten Versuchs sehr ähnlich (Abbildung 24) Der F1-Score stieg aber leicht auf 73,7 Prozent, was an der leichten Verbesserung in der Erkennung der überraschten Emotion liegt.

3.5.7. Versuch 9

Ein letzter Versuch bestand darin, wie auch im Paper des Netzwerkes, nur mit einem Datensatz zu trainieren, da so die Bilddaten identische Rahmenbedingungen, wie Auflösung und fps, haben. Hierfür wurde das Trainingsset 4 verwendet. Das Ergebnis dieses Versuchs war das bisher beste mit einer Validierungsgenauigkeit von beinahe 89 Prozent. Damit wurde sogar der Wert des Papers knapp geschlagen. Auch die Confusionmatrix spiegelt dieses starke Ergebnis wider (Abbildung 25). Insgesamt wurde nur sechs Mal falsch getippt, sodass es nicht verwundert, dass der F1-Score mit 84,5 Prozent ebenfalls der höchste aller Versuche ist.

3.6. Das Skript

Neben dem Finden eines neuen Netzwerkes war es wichtig, dass die Vorhersage schon direkt, quasi in Echtzeit, während der Aufnahme des Videos

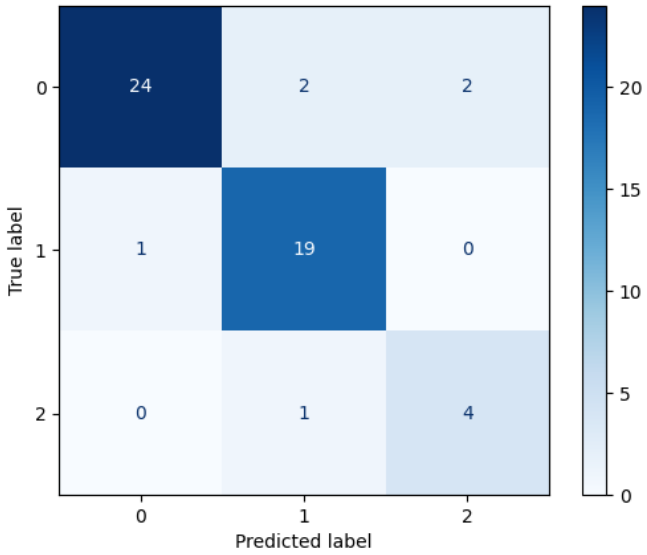

Abbildung 25.: Confusion Matrix - Versuch 9

getätigt werden kann. Das Skript, das hierfür entwickelt wurde, wird im Folgenden anhand des Quellcodes erklärt. Insgesamt besteht das Skript aus sechs Methoden. Die erste Methode ist dafür da, die Eingaben, welche beim Aufrufen des Skriptes im Terminal mitgegeben werden, an die jeweiligen Variablen zu übergeben (Quellcode 3). Der Nutzer kann so definieren, wie der Nutzer heißt und auch das Verzeichnis festlegen, in das die Ergebnisse gespeichert werden sollen.

```
34  def parseArguments():
35      global PARTICIPANT_NAME
36      global OUTPUT_DIR
37      if len(sys.argv) >= 2:
38          PARTICIPANT_NAME = sys.argv[1]
39      if len(sys.argv) >= 3:
40          OUTPUT_DIR = sys.argv[2]
```

Quellcode 3.: Die parseArguments-Methode

Die zweite Methode ist eine Methode, die überprüft, ob die benötigten Verzeichnisse schon vorhanden sind. Falls dies nicht der Fall ist, werden

diese neu erstellt. Somit kann nicht vorkommen, dass das Programm nicht funktioniert, falls kein Outputverzeichnis mitgegeben wird oder dieses noch nicht vorhanden ist (siehe hierzu Quellcode 4).

Eine dritte Methode ist die Initialisierung des Modells, welche bereits in Quellcode 1 zu sehen war. Auch die vortrainierten Gewichte werden hier ins Modell geladen.

Methode vier ist dazu da, eine Liste zu erstellen, in der später die Wahrscheinlichkeiten der jeweiligen Vorhersagen gespeichert werden. In dieser Methode werden dabei direkt die Überschriften der jeweiligen Spalten festgelegt. Es handelt sich hierbei um die Namen der drei beziehungsweise vier Klassen sowie dem Timestamp, an dem die Vorhersage stattfindet, welcher in Frames angegeben wird.

Die Methode *predictParticipant* führt anhand der Inputmatrix die eigentliche Vorhersage durch und schreibt das Ergebnis dieser Vorhersage auch in die Konsolenausgabe, sodass jederzeit direkt ersichtlich ist um welche Emotion es sich handelt. Außerdem werden hier auch die Wahrscheinlichkeitswerte jeder Klasse in die bereits erwähnte Liste geschrieben.

Die letzte Methode ist die Hauptmethode, die alles steuert. Anfangs werden hier die Methoden *checkDirectory*, *createEmptyPredictionList*, sowie *initModel* aufgerufen. Der nächste Schritt ist nun einen Videofeed zu erhalten. Dies geschieht mithilfe der Open Source Computer Vision und Machine Learning Software Library *OpenCV*, welche auf eine Webcam oder ein Video im Dokumentenverzeichnis zugreift. Anschließend führt OpenCV, falls gewünscht, innerhalb jedes Frames des Videofeeds eine Gesichtserkennung durch. Hierbei kommt der Haar-Cascade-Klassifikator zum Einsatz. In OpenCV stehen für die Erkennung des Gesichts vier dieser Klassifikatoren zur Verfügung[57]. Nach einem Test über das erste Trainingsset (siehe Kapitel 3.4.1) stellte sich der Standard Klassifikator als der beste heraus. Dieser erkannt nur in 14 Fällen nicht die richtige Anzahl an Gesichtern. Der Vergleich der verschiedenen Klassifikatoren ist in Tabelle 5 zu erkennen. Falls eine Gesichtserkennung gewünscht ist, wird, wie in Quellcode 5 zu sehen ist, um

[57] opencv 2021

	Anzahl an Bildfolgen mit weniger/mehr als 96 Gesichtern
frontalface_default	14
frontalface_alt	85
frontalface_alt2	58
frontalface_alt_tree	428

Tabelle 5.: Vergleich der Klassifikatoren zur Gesichtserkennung

jedes der erkannten Gesichter ein Rechteck gezogen (Zeile 86). Dieser Bildausschnitt wird daraufhin ausgeschnitten und auf die Maße 64 × 64 skaliert und in ein Schwarz-Weiß-Foto umgewandelt, um den Eingabedimensionen des CNNs zu entsprechen. Der somit neu entstandene Frame wird sowohl für die spätere Weiterverarbeitung in einer Liste als auch auf dem System als Foto gespeichert. Falls keine Gesichtserkennung gewünscht ist und das gesamte Bild genutzt werden soll, wird dieses ebenfalls auf die Maße 64 × 64 skaliert und in ein Schwarz-Weiß-Foto umgewandelt sowie auf dem auf dem System gespeichert.

Anschließend kommt es alle 96 Frames, des Inputvideos, zu einer Vorhersage des Netzwerks. Vorher müssen diese Frames aber noch in das richtige Format gebracht werden, um vom Netzwerk verarbeitet werden zu können. Dies geschieht in Quellcode 6. Die Frames werden an dieser Stelle in mehreren Schritten in eine fünfdimensionale Matrix umgewandelt, welche dann in Zeile 113 an die Methode *predictParticipant* übergeben wird. Am Ende der *startEmotionRecognition*-Methode findet sich außerdem noch Code, um das Inputvideo in einem Fenster anzuzeigen und das Programm mit einem Druck der Taste *Q* zu beenden. Wenn das Programm beendet wird oder das Inputvideo zu Ende ist (im Falle eines vorher aufgenommenen Video), wird die Liste mit den Vorhersagen noch in eine CSV-Datei gespeichert.

```
201 def checkDirectory():
202     global OUTPUT_DIR
203     global PARTICIPANT_NAME
204     scriptDir =
          os.path.dirname(os.path.abspath(__file__))
205     print("Skript Directory: " + scriptDir)
206     dirFace = 'cropped_face'
207
208     #If no Output directory is given - Set
          directory to be the same as the scripts
          location
209     if OUTPUT_DIR is None:
210         dirOutput =
              os.path.join(scriptDir,PARTICIPANT_NAME)
211     else:
212         dirOutput =
              os.path.join(OUTPUT_DIR,PARTICIPANT_NAME)
213
214     # Create if there is no output directory
215     if not os.path.exists(dirOutput):
216         os.mkdir(dirOutput)
217         print("Directory " , dirOutput ,  "
              Created ")
218     else:
219         print("Directory " , dirOutput ,  " has
              been found.")
220
221     #create path for the CSV-file
222     csvPath = os.path.join(dirOutput,
          PARTICIPANT_NAME+"_emotions.csv")
223
224     croppedFolder =
          os.path.join(dirOutput,dirFace)
225
226     #Create if there is no Folder for extracted
          Images
227     if not os.path.exists(croppedFolder):
228         os.mkdir(croppedFolder)
229         print("Directory " , croppedFolder ,  "
              Created ")
230     else:
231         print("Directory " , croppedFolder ,  "
              has been found.")
232
233     return croppedFolder, csvPath
```

Quellcode 4.: Die checkDirectory-Methode

```
110             if croppImages:
111                 for (x, y, w, h) in faces:
112                     # Draw rectangles around each face
113                     cv2.rectangle(im, (x, y), (x + w,
                            y + h),(0,0,255),thickness=2)
114                     # saving faces according to
                            detected coordinates
115                     sub_face = im[y:y+h, x:x+w]
116                     #Resize
117                     sub_face_resize =
                            cv2.resize(sub_face,
                            (image_rows,image_columns),
                            interpolation= cv2.INTER_AREA)
118                     #Convert RGB to Grayscale
119                     greyImage =
                            cv2.cvtColor(sub_face_resize,
                            cv2.COLOR_BGR2GRAY)
120                     frames.append(greyImage)
121                     faceCount += 1
122                     cv2.imwrite(FaceFileName,
                            sub_face)
```

Quellcode 5.: Gesichtserkennung mit OpenCV

```
125             if faceCount == framerate:
126                 framesArray = numpy.asarray(frames)
127                 videoarray =
                        numpy.rollaxis(numpy.rollaxis
                        (framesArray,2,0),2,0)
128                 frameList.append(videoarray)
129                 frameList = numpy.asarray(frameList)
130                 samples = len(frameList)
131                 frameset =
                        numpy.zeros((samples,1,image_rows,
                        image_columns, image_depth))
132                 for h in range(samples):
133                     frameset[h][0][:][:][:] =
                            frameList[h,:,:,:]
134                 frameset = frameset.astype('float32')
135                 frameset -= numpy.mean(frameset)
136                 frameset /= numpy.max(frameset)
137                 frameset = tf.transpose(frameset,
                        [0,2,3,4,1])
```

Quellcode 6.: Verarbeitung der Videoframes

4. Fazit

4.1. Ergebnisse

Betrachtet man alle Ergebnisse der verschiedenen Versuche und vergleicht diese miteinander, so erkennt man deutlich, dass Versuch neun mit Abstand das beste Ergebnis geliefert hat. In Tabelle 6 werden die Ergebnisse dargestellt. Der F1-Score liegt mit 84,5 Prozent knapp zehn Prozent über der vorherigen Bestmarke von 74,7 Prozent des Versuchs acht. Auch die Validierungsgenauigkeit ist um sechs Prozent höher als dies bei Versuch acht und drei der Fall ist.
Um zu testen, wie die jeweiligen Netzwerke beziehungsweise Gewichte im finalen Skript abschneiden, wurden jedem Netz vier Videos als Input übergeben. Diese Videos waren jedoch auch aus den jeweiligen Datensätzen, sodass ein eventuelles Overfitting nicht direkt ausgeschlossen werden konnte.
Das erste Video war aus dem CASME II-Datensatz und wurde von den Autoren als überraschte Emotion eingestuft. Nahezu jedes Netz erkannte

	F1-Score	Validierungsgenauigkeit
Versuch 1	71,7 %	79 %
Versuch 2	55 %	61 %
Versuch 3	71,4 %	83 %
Versuch 4	55 %	61 %
Versuch 5	65 %	69 %
Versuch 6	62,8 %	72 %
Versuch 7	67,4 %	76 %
Versuch 8	67,4 %	83 %
Versuch 9	**84,5 %**	**89 %**

Tabelle 6.: Vergleich der Ergebnisse der verschiedenen Versuche

dies auch eindeutig als die überraschte Emotion an. Das Netz aus Versuch vier hat in diesem Fall sogar vor der eigentlichen Emotion einen neutralen Zustand prognostiziert. Netz zwei war sich nicht ganz sicher, ob es sich um eine neutrale oder überraschte Emotion handelt. Mit einer Prognose von 49,96 Prozent für die überraschte Emotion und 49,86 Prozent für den neutralen Zustand wurde sich aber dennoch richtig entschieden. Netzwerk sechs, das die zusätzlichen SMIC-Bilder im Trainingsset hatte, prognostizierte hier stattdessen eine negative Emotion. Und auch der Versuch, der laut den Scores am besten abgeschlossen hat, war sich sehr sicher, dass es sich bei dieser Emotion nicht um die überraschte Emotion handelte.
Das zweite Video war aus dem ersten CASME-Datensatz und stellte eine negative Emotion dar. Auffallend ist, dass hier fast alle Netze die richtige Prognose abgeliefert haben und nur das letzte Netz erneut falsch lag und eine positive Emotion erkannt hat.
Im dritten Video aus dem CAS(ME)^2-Datensatz sind mehrere Emotionen vorhanden, der Inhalt des von den Probanden geschauten Videos war jedoch ein Video um positive Emotionen hervorzurufen. Im Datensatz sind insgesamt zwölf Emotionen innerhalb des Videos erkannt und die Zeitpunkte, an denen diese vorkommen, notiert worden. Emotionen, die vom Netzwerk zwischen diesen Zeitpunkten erkannt werden, werden deshalb ignoriert beziehungsweise sie werden nicht so stark gewichtet. Bei den wahren Emotionen handelt es sich hauptsächlich um positive Emotionen, aber auch die überraschte und die negative Emotion sind im Video vorhanden.
Das Netzwerk des ersten Versuchs hat am Anfang des Videos mehrmals die negative Emotionen prognostiziert. Nach der fünften Prognose wurde daraufhin fast immer die positive Emotion erkannt. Auch die überraschte Emotion wurde an drei Stellen prognostiziert, jedoch waren dies nicht die Stellen, an denen diese laut Labels der Daten stattfinden. Wenn nur die Prognosen an den Zeitpunkten der gelabelten Emotionen betrachtet werden, so ist zu sehen, dass in acht von elf Prognosen richtig getippt wurde. Es handelt sich hier nur um elf und nicht um zwölf Prognosen, da zwei der gelabelten Emotionen sehr kurz aufeinander folgen und diese dann in die

selben 96 Frames fallen, welche während einer Prognose betrachtet werden. Netzwerk zwei schließt schlechter ab und erzielt nur fünf aus elf richtigen Prognosen. Auch hier wurde anfangs noch die negative Emotion prognostiziert. In diesem Fall wurde aber deutlich öfter eine überraschte Emotion vom Netz erkannt, sodass zwar auch die wahre überraschte Emotion erkannt werden konnte, aber dies auch oft an falschen Stellen getan wurde. Alle anderen Versuche haben das identische Ergebnis geliefert und zwar eine Prognose der positiven Emotion an jeder Stelle, sodass hier ebenfalls acht von elf Emotionen richtig erkannt wurden.

Das vierte Video war aus einem kleinen Datensatz von Husak et al.[58], der aus kurzen Ausschnitten von Pokerspielen bestand. Die meisten der Video konnten für diese Zwecke und zum Training nicht verwendet werden, da in den meisten Videos sehr viel Bewegung stattfindet und die Videos eigentlich auch kürzer als 96 Frames sind. Um das zweite Problem zu lösen, wurden deshalb vor und nach das Video noch eine Zeit lang ein Standbild des ersten beziehungsweise letzten Frames geschnitten. Die Emotion, die im Video zu sehen ist, wurde von den Autoren als überraschte Definition gelabelt. Die Netze aus Versuch eins, drei, vier und acht prognostizierten hier ausschließlich negative Emotionen. Das Netz des zweiten Versuches erkannte zudem eine positive Emotion. Die beiden Netzwerke, die alle 60 Frames eine Prognose durchführen, haben neben zwei negativen Emotionen zudem eine überraschte erkannt. Auch das Netzwerk aus Versuch fünf lieferte eine überraschte Emotion und außerdem eine positive Emotion. Netz neun erkannte beide Male eine positive Emotion.

Kombiniert man nun all diese Ergebnisse, wie dies in Tabelle 7, dann ist zu erkennen, dass das Netzwerk aus Versuch neun nicht besonders gut abschneidet. Stattdessen schneidet das Netzwerk mit der höheren Abtastrate am besten ab, obwohl es zuvor eines der schlechtesten war. Jedoch ist das Ergebnis bei allen Netzwerken nicht sonderlich zufriedenstellend. Mit einem Blick auf die visualisierten Daten, die während des Trainings erstellt wurden (Abbildung 26 und Abbildung 27), ist zu erkennen, dass der Validierungsver-

[58] Husak, Cech und Matas 2017

	Anzahl der richtig erkannten Emotionen
Versuch 1	10,5 von 14
Versuch 2	7 von 14
Versuch 3	10 von 14
Versuch 4	10 von 14
Versuch 5	9,5 von 14
Versuch 6	**11 von 14**
Versuch 7	10,5 von 14
Versuch 8	10 von 14
Versuch 9	8 von 14

Tabelle 7.: Anzahl der richtig erkannten Testemotionen

lust nicht, wie es eigentlich zu erwarten wäre, sinkt, sondern im Laufe des Trainingsprozesses immer weiter steigt. Dies deutet auf ein Overfitting des Netzwerks hin. Leider findet sich dieses Verhalten des Validierungsverlust in jedem der durchgeführten Trainings wieder. Betrachtet man die Daten genauer, dann ist zu erkennen, dass die Zunahme dieses Verlustes bereits im Bereich der Epochen 15-20 stattfindet. An diesen Stellen wäre das Netzwerk also schon fertig trainiert gewesen.

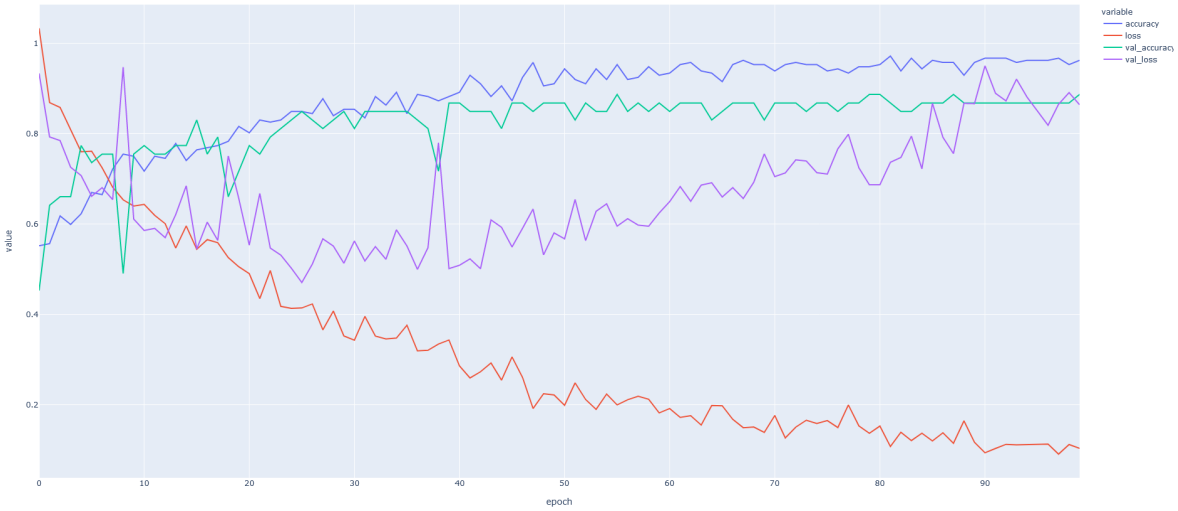

Abbildung 26.: Visualisierung der Metriken - Versuch 9

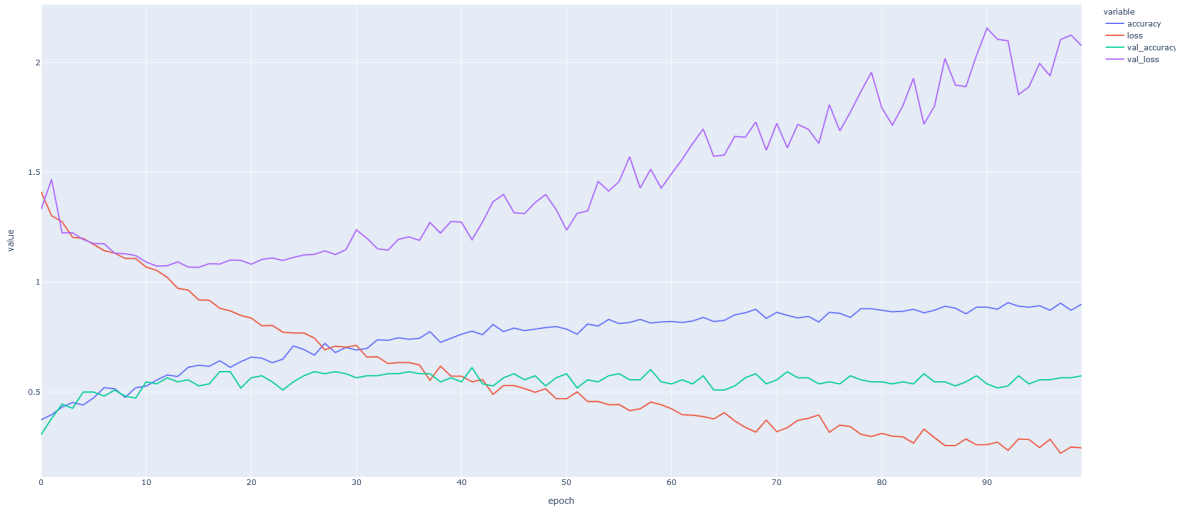

Abbildung 27.: Visualisierung der Metriken - Versuch 4

4.2. Diskussion der erhaltenen Ergebnisse

Anhand der Ergebnisse ist zu erkennen, dass bei allen trainierten Netzwerken vermutlich ein Overfitting stattgefunden hat. Bei Betrachtung der Scores und den Ergebnissen aus den anschließenden Test lässt sich erkennen, dass das Netzwerk aus Versuch eins insgesamt am besten abgeschnitten hat, da es recht solide Werte im F1-Score, der Valididierungsgenauigkeit, sowie den anschließenden Test erzielt hat. Wichtig ist hierbei zu sehen, dass nicht gesagt ist, dass die verschiedenen Netzwerke im Versuchsaufbau des Fraunhofer IOSB schlechte Ergebnisse liefern werden. Mit Blick auf die vorhandenen Daten ist es aber eventuell besser, das Netzwerk noch weiter zu trainieren. Auch sollten weitere Test durchgeführt werden, um zu ermitteln, ob eine höhere Abtastrate doch besser zum gegebenen Versuchsaufbau passt. Auch ob der Input mithilfe der Gesichtserkennung zugeschnitten werden soll oder nicht kann in weiteren Versuchen untersucht werden.
Ein Netzwerk besitzt sehr viele Parameter an denen kleine Veränderungen schon große Auswirkungen auf das Netzwerk haben können. Aus diesem Grund ist das Finden der passenden Parameter für die gegebene Herausforderung ein langer Prozess. Ich denke, dass das hier benutzte Netzwerk durchaus dazu in der Lage ist, auch ohne Overfitting trainiert zu werden.

Leider ist dieses Problem erst nach Durchführung aller Trainings aufgefallen, da als Metrik nur der F1-Score sowie die Validierungsgenauigkeit genutzt wurden. Eine Herausforderung kann jedoch sein, dass die vorhandenen Datensätze noch immer zu wenig Daten liefern, um das Netzwerk effizient zu trainieren. Eventuell ist eine Erweiterung durch synthetische Daten, wie beispielsweise 3D-Bilder, nötig. Auch kann es von Vorteil sein, selbst einen Datensatz aus den bereits durchgeführten Versuchen zur Flowmessung herzustellen. Dies ist allerdings ein langer Prozess, da die Videoaufnahmen Frame für Frame durchgeschaut werden müssen und eine Änderung der Emotion notiert werden muss. Mithilfe des FACS könnten dann die richtigen Emotionen erkannt und gelabelt werden.
Da das Netzwerk recht unkompliziert aufgebaut ist, dauert der Trainingsprozess über 100 Epochen auch nicht allzu lange. Wenn das Netzwerk zusätzlich auf einem Computer mit CUDA-Grafikkarte betrieben wird, verringert sich die Dauer des Trainings und der Prognose nochmals deutlich. Ein positiver Aspekt des Netzwerks beziehungsweise des Skriptes ist, dass es die Daten sehr gut in Echtzeit verarbeiten kann. Es kommt zu keiner Verzögerung während das Netzwerk eine Emotion prognostiziert. Dies liegt auch an der eher flachen Architektur.
Um das Ganze nun noch im Kontext der Flowmessung zu betrachten, bin ich der Meinung, dass eine Micro-Expression-Erkennung alleine nicht unbedingt zu Verbesserung der Messung führen wird. Ein Immersions- oder Flowzustand kann vorkommen, wenn man positive Emotionen verspürt, aber auch wenn man traurig oder überrascht ist. Eine Überlegung meinerseits war es, dass es hilfreich sein kann, alle Daten, sowohl physiologische als auch das Ergebnis der Emotionsprognose, an ein neuronales Netz zu geben und dieses dann bestimmen zu lassen, ob es sich zum gegebenen Zeitpunkt um einen Immersions- oder Flowzustand handelt.
Abschließend wird nochmals ein Blick auf die Ziele dieser Arbeit geworfen, wie sie in der Einleitung definiert wurden: Eine Präsentation der Methoden und Systeme zur Micro-Expression-Erkennung wurde in Kapitel 2.4 dargelegt. Dort war zu sehen, dass seit einigen Jahren zunehmend auf neuronale

Netze, im besonderen auf CNNs, gesetzt wird. Auch wurde deutlich, dass für eine Erkennung der Expressions immer eine zeitliche Information vorhanden sein muss. Bei der Entwicklung des Systems wurde deshalb auch auf ein CNN gesetzt. Dieses schafft es, aus den Daten eines Livevideos eine Emotion zu prognostizieren. Inwiefern die trainierten Gewichte dazu in der Lage sind, dies auch im gegebenen Versuchsaufbau zu tun, bleibt abzuwarten. Weitere Tests und Evaluationen sind hierfür nötig. Somit wurden die Ziele alle erreicht. Zudem wurden auch die zusätzlichen Ziele erreicht. Das ganze System besteht nur aus einem Skript. Um eine Prognose anhand der Livedaten zu starten, muss dieses nur aufgerufen werden. Für das Training ist jedoch ein anderes Skript notwendig Lediglich ein paar Anpassungen sind je nach Anforderung nötig. Da in den Skripten aber sehr viel mit Kommentaren gearbeitet wurde, sollte dies auch für einen unerfahrenen Anwender gut möglich sein. Ebenso läuft das Skript auch in einer Windowsdistribution, was die gesamte Einbindung an das bestehende System erleichtern sollte.

Literatur

Adams, Ernest (2014). *Fundamentals of game design*. 3rd ed. Berkeley, Calif.: New Riders.

Autoren der Wikimedia-Projekte (Okt. 2003). *Flow (Psychologie) – Wikipedia*. [Online; accessed 21. May 2021]. URL: https://de.wikipedia.org/w/index.php?title=Flow_(Psychologie).

Bartle, Richard (2003). *Designing Virtual Worlds*. New Riders Games. ISBN: 0131018167.

Fh-Bielefeld-Mif-Sw-Engineering-2017 (Mai 2021). *LBPH · Script*. [Online; accessed 29. May 2021]. URL: https://fh-bielefeld-mif-sw-engineerin.gitbooks.io/script/content/embedded-computing/gesichtserkennung/lbph.html.

Csikszentmihalyi, Mihaly (1975). *Beyond boredom and anxiety*. 1st ed. The Jossey-Bass behavioral science series. San Francisco: Jossey-Bass Publishers. ISBN: 9780875892610.

Deep Learning 2021: Was ist es und warum wird es eingesetzt? (Jan. 2021). [Online; accessed 21. May 2021]. URL: https://datasolut.com/was-ist-deep-learning.

Duan, Xiaodong u. a. (2016). „Recognizing spontaneous micro-expression from eye region“. In: *Neurocomputing* 217. SI: ALLSHC, S. 27–36. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2016.03.090. URL: https://www.sciencedirect.com/science/article/pii/S0925231216306087.

Ekman, Paul (2010). *Gefühle lesen*. Springer Berlin Heidelberg. DOI: 10.1007/978-3-662-53239-3. URL: https://doi.org/10.1007/978-3-662-53239-3.

Ekman, Paul und Wallace V. Friesen (2003). *Unmasking the face*. eng. OCLC: 253914548. Cambridge,MA: Malor Books. ISBN: 9781883536367.

Ekman, Paul, Wallace V. Friesen und Joseph C. Hager (2002). *Facial action coding system: the manual*. eng. OCLC: 178927696. Salt Lake City, Utah: Research Nexus. ISBN: 9780931835018.

GlobalWebIndex (März 2020). *Media usage during COVID-19 by country | Statista*. [Online; accessed 30. May 2021]. URL: https://www.statista.com/statistics/1106498/home-media-consumption-coronavirus-worldwide-by-country.

Guo, Jianzhu u. a. (Apr. 2017). „Multi-modality Network with Visual and Geometrical Information for Micro Emotion Recognition“. In: DOI: 10.1109/FG.2017.103.

Haggard, Ernest A. und Kenneth S. Isaacs (1966). „Micromomentary facial expressions as indicators of ego mechanisms in psychotherapy“. In: *Methods of Research in Psychotherapy*. Springer US, S. 154–165. DOI:

10.1007/978-1-4684-6045-2_14. URL: https://doi.org/10.1007/978-1-4684-6045-2_14.

Hasani, Behzad und Mohammad Mahoor (März 2017). „Spatio-Temporal Facial Expression Recognition Using Convolutional Neural Networks and Conditional Random Fields“. In:

Huang, Xiaohua u. a. (Okt. 2015). „Spontaneous Facial Micro-expression Analysis using Spatiotemporal Completed Local Quantized Patterns“. In: *Neurocomputing* 175. DOI: 10.1016/j.neucom.2015.10.096.

Husak, Petro, Jan Cech und J. Matas (2017). „Spotting Facial Micro-Expressions “ In the Wild ““. In:

IcedDoggie (Mai 2019). *DSSN-MER*. [Online; accessed 28. May 2021]. URL: https://github.com/IcedDoggie/DSSN-MER.

Jennett, Charlene u. a. (Sep. 2008). „Measuring and defining the experience of immersion in games“. en. In: *International Journal of Human-Computer Studies* 66.9, S. 641–661. ISSN: 10715819. DOI: 10.1016/j.ijhcs.2008.04.004. URL: https://linkinghub.elsevier.com/retrieve/pii/S1071581908000499 (besucht am 29. 05. 2021).

Kannegieser, Ehm, Daniel Atorf und Joachim Herold (2021). *Measuring Flow, Immersion and Arousal/Valence for application in Adaptive Learning Systems*.

Khor, Huai-Qian u. a. (Mai 2018). „Enriched Long-Term Recurrent Convolutional Network for Facial Micro-Expression Recognition“. In: S. 667–674. DOI: 10.1109/FG.2018.00105.

Kim, Dae, Wissam Baddar, Jinhyeok Jang u. a. (Apr. 2017). „Multi-Objective Based Spatio-Temporal Feature Representation Learning Robust to Expression Intensity Variations for Facial Expression Recognition“. In: *IEEE Transactions on Affective Computing* PP, S. 1–1. DOI: 10.1109/TAFFC.2017.2695999.

Kim, Dae, Wissam Baddar und Yong Ro (Okt. 2016). „Micro-Expression Recognition with Expression-State Constrained Spatio-Temporal Feature Representations“. In: S. 382–386. DOI: 10.1145/2964284.2967247.

Li, Xiaobai, Xiaopeng Hong u. a. (Nov. 2015). „Reading Hidden Emotions: Spontaneous Micro-expression Spotting and Recognition“. In: *Arxiv*.

Li, Xiaobai, Tomas Pfister u. a. (Apr. 2013). „A Spontaneous Micro-expression Database: Inducement, collection and baseline“. In: S. 1–6. ISBN: 978-1-4673-5545-2. DOI: 10.1109/FG.2013.6553717.

Liong, Sze-Teng u. a. (2018). *OFF-ApexNet on Micro-expression Recognition System*. arXiv: 1805.08699 [cs.CV].

Liong, Sze u. a. (Mai 2019). „Shallow Triple Stream Three-dimensional CNN (STSTNet) for Micro-expression Recognition“. In: S. 1–5. DOI: 10.1109/FG.2019.8756567.

Machine Learning vs Deep Learning – Wo liegt der Unterschied? – Data Science Blog (Mai 2021). [Online; accessed 21. May 2021]. URL: https://data-science-blog.com/blog/2018/05/14/machine-learning-vs-deep-learning-wo-liegt-der-unterschied.

Machine Learning: Algorithmen, Methoden und Beispiele (Jan. 2021). [Online; accessed 21. May 2021]. URL: https://datasolut.com/was-ist-machine-learning/#ueberwachtes-lernen.

Micro Expressions | Facial Expressions | Paul Ekman Group (Feb. 2020). [Online; accessed 17. May 2021]. URL: https://www.paulekman.com/resources/micro-expressions.

Mueller, John und Luca Massaron (2019). *Deep learning*. For dummies. Hoboken, NJ: John Wiley und Sons, Inc. ISBN: 9781119543039.

Murray, Janet Horowitz (2016). *Hamlet on the Holodeck. The Future of Narrative in Cyberspace*. New York: Free Press. ISBN: 978-1-439-13613-3.

Newzoo (Jan. 2020). *Motivation for playing video games U.S. 2020 | Statista*. [Online; accessed 30. May 2021]. URL: https://www.statista.com/statistics/239310/reasons-why-female-online-gamers-play-games-in-the-united-states.

opencv (Mai 2021). *opencv*. [Online; accessed 26. May 2021]. URL: https://github.com/opencv/opencv/tree/master/data/haarcascades.

Patel, Devangini, Xiaopeng Hong und Guoying Zhao (Dez. 2016). „Selective deep features for micro-expression recognition“. In: S. 2258–2263. DOI: 10.1109/ICPR.2016.7899972.

Peng, Min, Chongyang Wang u. a. (Okt. 2017). „Dual Temporal Scale Convolutional Neural Network for Micro-Expression Recognition“. In: *Frontiers in Psychology* 8, S. 1745. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2017.01745. URL: http://journal.frontiersin.org/article/10.3389/fpsyg.2017.01745/full (besucht am 26. 05. 2021).

Peng, Min, Wu Zhan u. a. (Mai 2018). „From Macro to Micro Expression Recognition: Deep Learning on Small Datasets Using Transfer Learning“. In: S. 657–661. DOI: 10.1109/FG.2018.00103.

Pfister, Tomas u. a. (2011). „Recognising spontaneous facial micro-expressions“. In: *2011 International Conference on Computer Vision*, S. 1449–1456. DOI: 10.1109/ICCV.2011.6126401.

Polikovsky, Senya, Yoshinari Kameda und Yuichi Ohta (Jan. 2010). „Facial micro-expressions recognition using high speed camera and 3D-gradient descriptor“. In: S. 1–6. DOI: 10.1049/ic.2009.0244.

Qu, Fangbing u. a. (Jan. 2017). „CAS(ME)2): A Database for Spontaneous Macro-Expression and Micro-Expression Spotting and Recognition“. In: *IEEE Transactions on Affective Computing* 9, S. 424–436. DOI: 10.1109/TAFFC.2017.2654440.

Reddy, Sai u. a. (Juli 2019). „Spontaneous Facial Micro-Expression Recognition using 3D Spatiotemporal Convolutional Neural Networks“. In: S. 1–8. DOI: 10.1109/IJCNN.2019.8852419.

Shahar, Hadas und Hagit Hel-Or (2019). „Micro Expression Classification using Facial Color and Deep Learning Methods“. In: *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, S. 1673–1680. DOI: 10.1109/ICCVW.2019.00207.

Shreve, Matthew u. a. (Apr. 2011). „Macro- and micro-expression spotting in long videos using spatio-temporal strain“. In: S. 51–56. DOI: 10.1109/FG.2011.5771451.

Statista (Aug. 2020). *Video Games - Users worldwide 2025 | Statista*. [Online; accessed 30. May 2021]. URL: https://www.statista.com/forecasts/456610/video-games-users-worldwide-forecast.

Supervised Learning: Definition, Arten & Beispiele - datasolut Wiki (Jan. 2021). [Online; accessed 21. May 2021]. URL: https://datasolut.com/wiki/supervised-learning.

Wang, Su-Jing u. a. (Juni 2018). „Micro-expression Recognition with Small Sample Size by Transferring Long-term Convolutional Neural Network“. In: *Neurocomputing* 312. DOI: 10.1016/j.neucom.2018.05.107.

Wang, Y. u. a. (2015). „Dynamic facial expression recognition using local patch and LBP-TOP“. In: *2015 8th International Conference on Human System Interaction (HSI)*, S. 362–367.

Weidman, Seth (2019). *Deep learning from scratch: building with Python from first principles*. First edition. Sebastopol, CA: O'Reilly Media, Inc. ISBN: 9781492041412.

What is Sadness? | Feeling Sadness | Paul Ekman Group (Mai 2021). [Online; accessed 19. May 2021]. URL: https://www.paulekman.com/universal-emotions/what-is-sadness.

Wu, Qi, Xunbing Shen und Xiaolan Fu (Jan. 2011). „The Machine Knows What You Are Hiding: An Automatic Micro-expression Recognition System“. In: S. 152–162. ISBN: 978-3-642-24570-1. DOI: 10.1007/978-3-642-24571-8_16.

Yan, Wen-Jing, Xiaobai Li u. a. (Jan. 2014). „CASME II: An Improved Spontaneous Micro-Expression Database and the Baseline Evaluation“. In: *PloS one* 9, e86041. DOI: 10.1371/journal.pone.0086041.

Yan, Wen-Jing, Qi Wu u. a. (Apr. 2013). „CASME Database: a dataset of spontaneous micro-expressions collected from neutralized faces“. In: DOI: 10.1109/FG.2013.6553799.

Zhao, Guoying und Matti Pietikäinen (Juli 2007). „Dynamic Texture Recognition Using Local Binary Patterns with an Application to Facial Expressions“. In: *IEEE transactions on pattern analysis and machine intelligence* 29, S. 915–28. DOI: 10.1109/TPAMI.2007.1110.

Zhao, Yue und Jiancheng Xu (Dez. 2019). „A Convolutional Neural Network for Compound Micro-Expression Recognition“. In: *Sensors* 19.24, S. 5553. DOI: 10.3390/s19245553. URL: https://doi.org/10.3390/s19245553.

Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbständig verfasst und hierzu keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Stellen der Arbeit die wörtlich oder sinngemäß aus fremden Quellen entnommen wurden, sind als solche kenntlich gemacht.
Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt oder an anderer Stelle veröffentlicht.
Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

Stuttgart, den 31.05.2021 Linus Ehmann

A. Quellcodes

Quellcode 1: Trainingsskript

```
1  import os
2  import sys
3  import cv2
4  import numpy
5  import tensorflow as tf
6  import matplotlib.pyplot as plt
7  from keras.models import Sequential
8  from keras.layers.core import Dense, Dropout,
       Activation, Flatten
9  from keras.layers.convolutional import
       Convolution3D, MaxPooling3D
10 from keras.optimizers import SGD, RMSprop
11 from tensorflow.keras.callbacks import
       ModelCheckpoint, CSVLogger
12 from keras.utils import np_utils, generic_utils
13 from sklearn.model_selection import
       train_test_split
14 from sklearn.metrics import confusion_matrix,
       ConfusionMatrixDisplay, f1_score,
       precision_score, recall_score, accuracy_score
15 from keras import backend as K
16 from ctypes import wintypes, windll
17 from functools import cmp_to_key
18
19
20
21 K.set_image_data_format('channels_last')
22 #classifier for face detection
```

```
23  classifier =
        cv2.CascadeClassifier(cv2.data.haarcascades+
        "haarcascade_frontalface_default.xml")
24
25  image_rows, image_columns, image_depth = 64, 64,
        96
26
27  numberOfClasses = 3
28
29  training_list = []
30
31  croppImage = False
32
33  def startTrain():
34      global training_list
35      global numberOfClasses
36
37      #--------------Pfade anpassen--------------
38      negativepath = 'D:\Desktop\Train/negative'
39      positivepath = 'D:\Desktop\Train\positive'
40      surprisepath = 'D:\Desktop\Train\surprise'
41      neutralpath = 'D:\Desktop\Train/neutral'
42      #------------------------------------------
43
44      #Number of Emotions - Addition for later use
            in Label list
45      negativeAmount = len(os.listdir(negativepath))
46      positiveAmount = len(os.listdir
            (positivepath))+negativeAmount
47      surpriseAmount = len(os.listdir
            (surprisepath))+positiveAmount
48      neutralAmount = len(os.listdir(neutralpath))+
            surpriseAmount
49
50      #process all Images into a numpy Array
51      processImages(negativepath)
52      processImages(positivepath)
53      processImages(surprisepath)
54      if numberOfClasses == 4:
```

```
55          processImages(neutralpath)
56
57      #make list a numpy Array and crate List for
            Labels in same length
58      training_list = numpy.asarray(training_list)
59      trainingsamples = len(training_list)
60
61      traininglabels = numpy.zeros((trainingsamples,
            ), dtype = int)
62
63      #fill list with regarding the number of
            classes and the amount of emotions
64      traininglabels[0:(negativeAmount-1)] = 0
65      traininglabels[(negativeAmount):␣
            (positiveAmount-1)] =
            1
66      traininglabels[positiveAmount:␣
            (surpriseAmount-1)] =
            2
67      if numberOfClasses == 4:
68          traininglabels[surpriseAmount:␣
                (neutralAmount-1)] =
                3
69
70      #make it categorigal - needed for
            categorical-crossentropy
71      traininglabels =
            np_utils.to_categorical(traininglabels,
            numberOfClasses)
72
73      #create the 5D Array needed as Input of the
            model
74      training_data = [training_list,
            traininglabels]
75      (trainingframes, traininglabels) =
            (training_data[0], training_data[1])
76      training_set = numpy.zeros((trainingsamples,1
            , image_rows, image_columns, image_depth))
77      print(training_set.shape)
```

```
78      print(trainingframes.shape)
79      for h in range(trainingsamples):
80              training_set[h][0][:][:][:] =
                    trainingframes[h,:,:,:]
81
82      training_set = training_set.astype('float32')
83      training_set -= numpy.mean(training_set)
84      training_set /= numpy.max(training_set)
85
86      # Save training images and labels in a numpy
            array
87      numpy.save('D:\Desktop\Train/
            microexpstcnn_images.npy',
            training_set)
88      numpy.save('D:\Desktop\Train/
            microexpstcnn_labels.npy',
            traininglabels)
89
90      """
91      # Load training images and labels that are
        stored in numpy array
92      training_set = numpy.load('D:\Desktop\Train/
        microexpstcnn_images.npy')
93      traininglabels =numpy.load('D:\Desktop\Train/
        microexpstcnn_labels.npy')
94      """
95
96      # MicroExpSTCNN Model
97      model = Sequential()
98      model.add(Convolution3D(32, (3, 3, 15),
            input_shape=(image_rows, image_columns,
            image_depth, 1), activation='relu'))
99      model.add(MaxPooling3D(pool_size=(3, 3, 3)))
100     model.add(Dropout(0.5))
101     model.add(Flatten())
102     model.add(Dense(128, activation='relu'))
103     model.add(Dropout(0.5))
104     model.add(Dense(numberOfClasses))
105     model.add(Activation('softmax'))
```

```
106         model.compile(loss =
                'categorical_crossentropy', optimizer =
                'SGD', metrics = ['accuracy'])
107
108         model.summary()
109
110         # Load pre-trained weights
111         #model.load_weights('D:\Desktop\Train/
                weights-improvement-97-0.69.hdf5')
112
113         #set save path for the weights generated in
                the training
114         filepath="D:\Desktop\Train\weights-
                improvement-{epoch:02d}-
                {val_accuracy:.2f}.hdf5"
115         checkpoint = ModelCheckpoint(filepath,
                monitor='val_accuracy', verbose=1,
                save_best_only=True, mode='max')
116
117         # Spliting the dataset into training and
                validation sets
118         train_images, validation_images, train_labels,
                validation_labels =
                train_test_split(training_set,
                traininglabels, test_size=0.2,
                random_state=4)
119
120         #transpose for cpu usage
121         train_images = tf.transpose(train_images,
                [0,2,3,4,1])
122         validation_images =
                tf.transpose(validation_images,
                [0,2,3,4,1])
123
124
125         # Save validation set in a numpy array
126         numpy.save('D:\Desktop\Train/
                microexpstcnn_val_images.npy',
                validation_images)
```

```
127         numpy.save('D:\Desktop\Train/microexpstcnn_val_labels.npy',
                validation_labels)
128
129         # Load validation set from numpy array
130         """
131         validation_images =
            numpy.load('D:\Desktop\Train/microexpstcnn_val_images.npy')
132         validation_labels =
            numpy.load('D:\Desktop\Train/microexpstcnn_val_labels.npy')
133         """
134
135         #Save training history in csv file
136         csv_logger =
                CSVLogger('D:/Desktop/Train/log.csv',
                append=True, separator=';')
137         callbacks_list = [checkpoint, csv_logger]
138         # Training the model
139         hist = model.fit(train_images, train_labels,
                validation_data = (validation_images,
                validation_labels),
                callbacks=callbacks_list, batch_size = 16,
                epochs= 100, shuffle=True)
140
141         # Finding Confusion Matrix using pretrained
                weights and plot it
142         predictions = model.predict(validation_images)
143         evaluation(predictions, validation_labels)
144     #Sorts like Windows explorer 1,2,3,... instead of
        1,10,11,2,...
145     def winsort(data):
146         _StrCmpLogicalW =
                windll.Shlwapi.StrCmpLogicalW
147         _StrCmpLogicalW.argtypes = [wintypes.LPWSTR,
                wintypes.LPWSTR]
148         _StrCmpLogicalW.restype  = wintypes.INT
149
```

```
150         cmp_fnc = lambda psz1, psz2:
                _StrCmpLogicalW(psz1, psz2)
151         return sorted(data, key=cmp_to_key(cmp_fnc))
152
153     def processImages(directory):
154         global classifier
155         global training_list
156         global image_rows, image_columns, image_depth
157         counter = 0
158         for folder in winsort(os.listdir(directory)):
159             if os.path.isdir(os.path.join
                    (directory,folder)):
160                 folder =
                        os.path.join(directory,folder)
161                 counter += 1
162                 frames = []
163                 faceCount=0
164                 for image in
                        winsort(os.listdir(folder)):
165                     path = os.path.join(folder,image)
166                     print(path)
167                     im = cv2.imread(path)
168                     imageresize = cv2.resize(im,
                            (image_rows, image_columns),
                            interpolation =
                            cv2.INTER_AREA)
169                     grayimage =
                            cv2.cvtColor(imageresize,
                            cv2.COLOR_BGR2GRAY)
170                     frames.append(grayimage)
171
172                     if croppImage:
173                         faces =
                                classifier.detectMultiScale
                                (
174                         im, # stream
175                         scaleFactor=1.10, # change
                                these parameters to
                                improve your video
                                processing performance
```

```
176                         minNeighbors=20,
177                         minSize=(30, 30) # min image
                                detection size
178                         )
179                         for (x, y, w, h) in faces:
180                             faceCount += 1
181                             sub_face = im[y:y+h,
                                    x:x+w]
182                             imageresize =
                                    cv2.resize(im,
                                    (image_rows,
                                    image_columns),
                                    interpolation =
                                    cv2.INTER_AREA)
183                             grayimage = cv2.cvtColor
                                    (imageresize,
                                    cv2.COLOR_BGR2GRAY)
184                             frames.append(grayimage)
185
186                         """
187                         #DEBUG: Are there enough Face
        Frames?
188                         if faceCount != image_depth:
189                             with
        open('D:/Desktop/Train/log.txt', 'a') as f:
190                                 print('Faces: ' +
        str(faceCount) + ' in Folder ' + folder + ' :
        ' + str(images), file=f)
191                         """
192                 frames = numpy.asarray(frames)
193                 videoarray = numpy.rollaxis
                        (numpy.rollaxis(frames, 2, 0), 2,
                        0)
194                 training_list.append(videoarray)
195
196     def evaluation(predictions, val_labels):
197         #get the labels
198         predictions_labels = numpy.argmax(predictions,
                axis=1)
```

```
199         val_labels = numpy.argmax(val_labels, axis=1)
200         #create Confusion matrix
201         cfm = confusion_matrix(val_labels,
                predictions_labels)
202         #Label names used for confusion matrix
203         ls = [0,1,2]
204         #get f1 score, accuracy, presicion and recall
205         f1 = f1_score(val_labels, predictions_labels,
                average='macro')
206         presicion = precision_score(val_labels,
                predictions_labels, average='macro')
207         recall = recall_score(val_labels,
                predictions_labels, average='macro')
208         accuracy = accuracy_score(val_labels,
                predictions_labels)
209         #Plot the Confusion matrix
210         disp = ConfusionMatrixDisplay␣
                (confusion_matrix=cfm,
                display_labels=ls)
211         disp.plot(cmap='Blues')
212         plt.show()
213         #save scores to file
214         with open('D:/Desktop/Train/log.txt', 'a') as
                f:
215             print('Accuracy: ' + str(accuracy) + ';
                    Presicion: ' + str(presicion) + ';
                    Recall: ' + str(recall) + '; F1: ' +
                    str(f1), file=f)
216
217 startTrain()
```

Quellcode 2: Erkennung aus Videodaten

```
1 import cv2
2 import os
3 import sys
4 import csv
5 import numpy
```

```
6  import tensorflow as tf
7  from sklearn.metrics import confusion_matrix
8  from goprocam import GoProCamera
9  from goprocam import constants
10 from keras.preprocessing import image as img
11 from tensorflow.keras.models import Sequential
12 from keras.layers.core import Dense, Dropout,
       Activation, Flatten
13 from keras.layers.convolutional import
       Convolution3D, MaxPooling3D
14 from keras import backend as K
15
16
17
18 OUTPUT_DIR = None
19 #Default Participant name for folder creation
20 PARTICIPANT_NAME = 'John Doe'
21
22 predictionList = None
23 #dimension of the input - depth equals number of
       frames
24 image_rows, image_columns, image_depth = 64,64,60
25 #set True if you want to cropp the images to the
       face region
26 croppImages = False
27
28 # number of Emotion classes you want to predict
29 numberOfClasses = 3
30 #path to weights
31 weightsPath = "D:\Desktop\Train/weights-
       improvement-54-0.72.hdf5"
32 _net = None
33
34 def parseArguments():
35     global PARTICIPANT_NAME
36     global OUTPUT_DIR
37     if len(sys.argv) >= 2:
38         PARTICIPANT_NAME = sys.argv[1]
39     if len(sys.argv) >= 3:
```

```
40          OUTPUT_DIR = sys.argv[2]
41
42  def startEmotionRecognition():
43      global predictionList
44      global _net
45      global numberOfClasses
46      frameCount=0
47      framerate = image_depth
48      faceCount = 0
49      frames = []
50      frameList = []
51
52      #classifier for face Detection
53      classifier = cv2.CascadeClassifier(
            (cv2.data.haarcascades+
            "haarcascade_frontalface_default.xml")
54
55      #where you want the channel last or first
56      K.set_image_data_format('channels_last')
57
58      #check if all directory exist
59      croppedFolder, outputPath = checkDirectory()
60
61      #if the participant was already processed stop
            the script
62      if os.path.isfile(outputPath):
63          print("Already calculated. Skipping.")
64          return
65
66      #Create empty List
67      predictionList = createEmptyPredictionList()
68
69      #initialize the model
70      _net = initModel()
71
72      # Choose right option accordingly to your
            needs
73      #gpCam = GoProCamera.GoPro()        # GoPro
            functionality not yet implemented see
            https://github.com/KonradIT/gopro-py-api
            for usage with OpenCV
```

```
74      capture = cv2.VideoCapture(0)       # Camera 0
            according to USB port
75      #capture = cv2.VideoCapture(r"D:\Downloads/
            Thesis/Master/Datasets/CASME/Casme^2/
            rawvideo/s24/24_0507climbingthewall.avi")
            #use Video from path
76
77      while (True):
78          f, im = capture.read()          # f
                returns only True, False according to
                video access
79
80          if f != True:
81              break
82
83          # im=cv2.flip(im,1,0)           #if you
                would like to give a mirror effect
84
85          # detectfaces
86          if croppImages:
87              faces = classifier.detectMultiScale(
88                  im,                         #
                        stream
89                  scaleFactor=1.10,           #
                        change these parameters to
                        improve your video processing
                        performance
90                  minNeighbors=20,
91                  minSize=(30, 30)            # min
                        image detection size
92                  )
93
94          #incremet Counter
95          frameCount += 1
96          FaceFileName = os.path.join(croppedFolder,
                "face_" + str(frameCount) + ".jpg")   #
                folder path and counter name image
97
98          #if you dont want to use cropped Images
```

```
99          if not croppImages:
100             #resize
101             sub_face_resize = cv2.resize(im,
                    (image_rows,image_columns),
                    interpolation= cv2.INTER_AREA)
102             #Convert RGB to Grayscale
103             greyImage =
                    cv2.cvtColor(sub_face_resize,
                    cv2.COLOR_BGR2GRAY)
104             frames.append(greyImage)
105             faceCount += 1
106             #Image saved on your System
107             cv2.imwrite(FaceFileName, im)
108
109         #if you want cropped images
110         if croppImages:
111             for (x, y, w, h) in faces:
112                 # Draw rectangles around each face
113                 cv2.rectangle(im, (x, y), (x + w,
                        y + h),(0,0,255),thickness=2)
114                 # saving faces according to
                        detected coordinates
115                 sub_face = im[y:y+h, x:x+w]
116                 #Resize
117                 sub_face_resize =
                        cv2.resize(sub_face,
                        (image_rows,image_columns),
                        interpolation= cv2.INTER_AREA)
118                 #Convert RGB to Grayscale
119                 greyImage =
                        cv2.cvtColor(sub_face_resize,
                        cv2.COLOR_BGR2GRAY)
120                 frames.append(greyImage)
121                 faceCount += 1
122                 cv2.imwrite(FaceFileName,
                        sub_face)
123
124         #start Prediction every 96 frames and
                convert array to match input dimension
                of model
```

```
125             if faceCount == framerate:
126                 framesArray = numpy.asarray(frames)
127                 videoarray =
                        numpy.rollaxis(numpy.rollaxis␣
                        (framesArray,2,0),2,0)
128                 frameList.append(videoarray)
129                 frameList = numpy.asarray(frameList)
130                 samples = len(frameList)
131                 frameset =
                        numpy.zeros((samples,1,image_rows,
                        image_columns, image_depth))
132                 for h in range(samples):
133                     frameset[h][0][:][:][:] =
                            frameList[h,:,:,:]
134                 frameset = frameset.astype('float32')
135                 frameset -= numpy.mean(frameset)
136                 frameset /= numpy.max(frameset)
137                 frameset = tf.transpose(frameset,
                        [0,2,3,4,1])
138                 predictParticipant␣
                        (frameset,frameCount)
139                 faceCount = 0
140                 frames = []
141                 frameList = []
142
143             # Video Window
144             cv2.imshow('Video Stream',im)
145             key = cv2.waitKey(1) & 0xFF
146             #press q for exit
147             if key == ord('q'):
148                 break
149         capture.release()
150         cv2.destroyAllWindows()
151
152         #write list in csv file at the end
153         outputfile = open(outputPath, 'w')
154         outputfile.write(predictionList)
155         outputfile.close()
156         return
```

```
157
158 def predictParticipant(input_dir, frame):
159     global predictionList
160
161     timestamp = str(frame)
162     #predict the emotion
163     predictions = _net.predict(input_dir)
164     #write predicted emotions in list
165     print('Predicted emotion: ' +
            str(numpy.argmax(predictions, axis=1)))
166     if  numberOfClasses == 3:
167         predictionList +=
                "{0},{1},{2},{3}\n".format(timestamp,
168         predictions[0][0], predictions[0][1],
                predictions[0][2])
169     elif numberOfClasses == 4:
170         predictionList +=
                "{0},{1},{2},{3},{4}\n".format␣
                (timestamp,
171         predictions[0][0], predictions[0][1],
                predictions[0][2], predictions[0][3])
172
173 def createEmptyPredictionList():
174     global numberOfClasses
175     if numberOfClasses==3:
176         return
                "timestamp,negativ,positiv,surprise\n"
177     elif numberOfClasses == 4:
178         return
                "timestamp,negativ,positiv,surprise,neutral\n"
179
180 def initModel():
181     global numberOfClasses
182     global weightsPath
183
184     model = Sequential()
185     model.add(Convolution3D(32, (3, 3, 15),
            input_shape=(image_rows, image_columns,
            image_depth, 1), activation='relu'))
```

```
186         model.add(MaxPooling3D(pool_size=(3, 3, 3)))
187         model.add(Dropout(0.5))
188         model.add(Flatten())
189         model.add(Dense(128, activation='relu'))
190         model.add(Dropout(0.5))
191         model.add(Dense(numberOfClasses))
192         model.add(Activation('softmax'))
193         model.compile(loss =
                'categorical_crossentropy', optimizer =
                'SGD', metrics = ['accuracy'])
194
195         model.summary()
196
197         model.load_weights(weightsPath)
198
199         return model
200
201     def checkDirectory():
202         global OUTPUT_DIR
203         global PARTICIPANT_NAME
204         scriptDir =
                os.path.dirname(os.path.abspath(__file__))
205         print("Skript Directory: " + scriptDir)
206         dirFace = 'cropped_face'
207
208         #If no Output directory is given - Set
                directory to be the same as the scripts
                location
209         if OUTPUT_DIR is None:
210             dirOutput = os.path.join␣
                    (scriptDir,PARTICIPANT_NAME)
211         else:
212             dirOutput = os.path.join␣
                    (OUTPUT_DIR,PARTICIPANT_NAME)
213
214         # Create if there is no output directory
215         if not os.path.exists(dirOutput):
216             os.mkdir(dirOutput)
217             print("Directory " , dirOutput ,  "
                    Created ")
```

```
218         else:
219             print("Directory " , dirOutput ,  " has
                    been found.")
220 
221         #create path for the CSV-file
222         csvPath = os.path.join(dirOutput,
                PARTICIPANT_NAME+"_emotions.csv")
223 
224         croppedFolder =
                os.path.join(dirOutput,dirFace)
225 
226         #Create if there is no Folder for extracted
                Images
227         if not os.path.exists(croppedFolder):
228             os.mkdir(croppedFolder)
229             print("Directory " , croppedFolder ,  "
                    Created ")
230         else:
231             print("Directory " , croppedFolder ,  "
                    has been found.")
232 
233         return croppedFolder, csvPath
234 
235 parseArguments()
236 startEmotionRecognition()
```

B. Visualierung der Trainingsmetriken

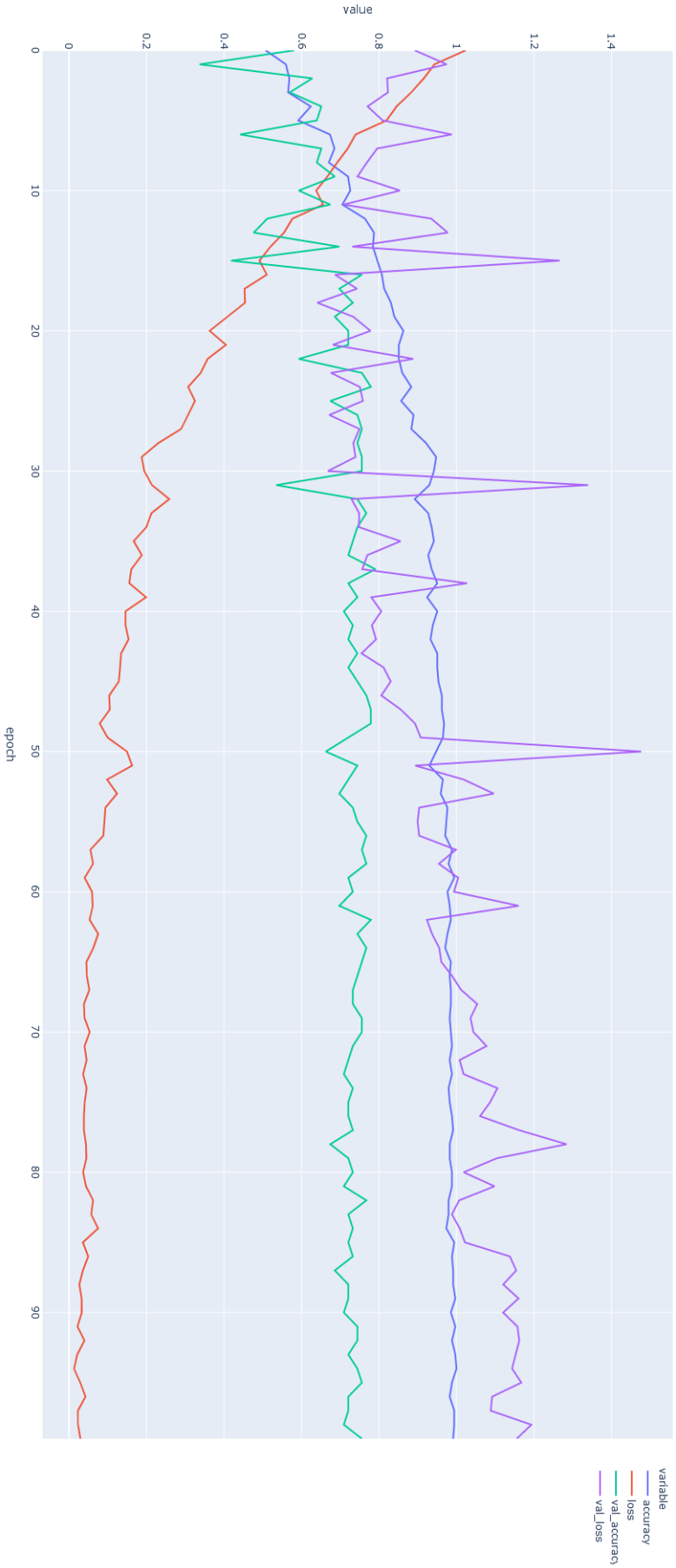

Abbildung 28.: Visualisierung der Metriken - Versuch 1

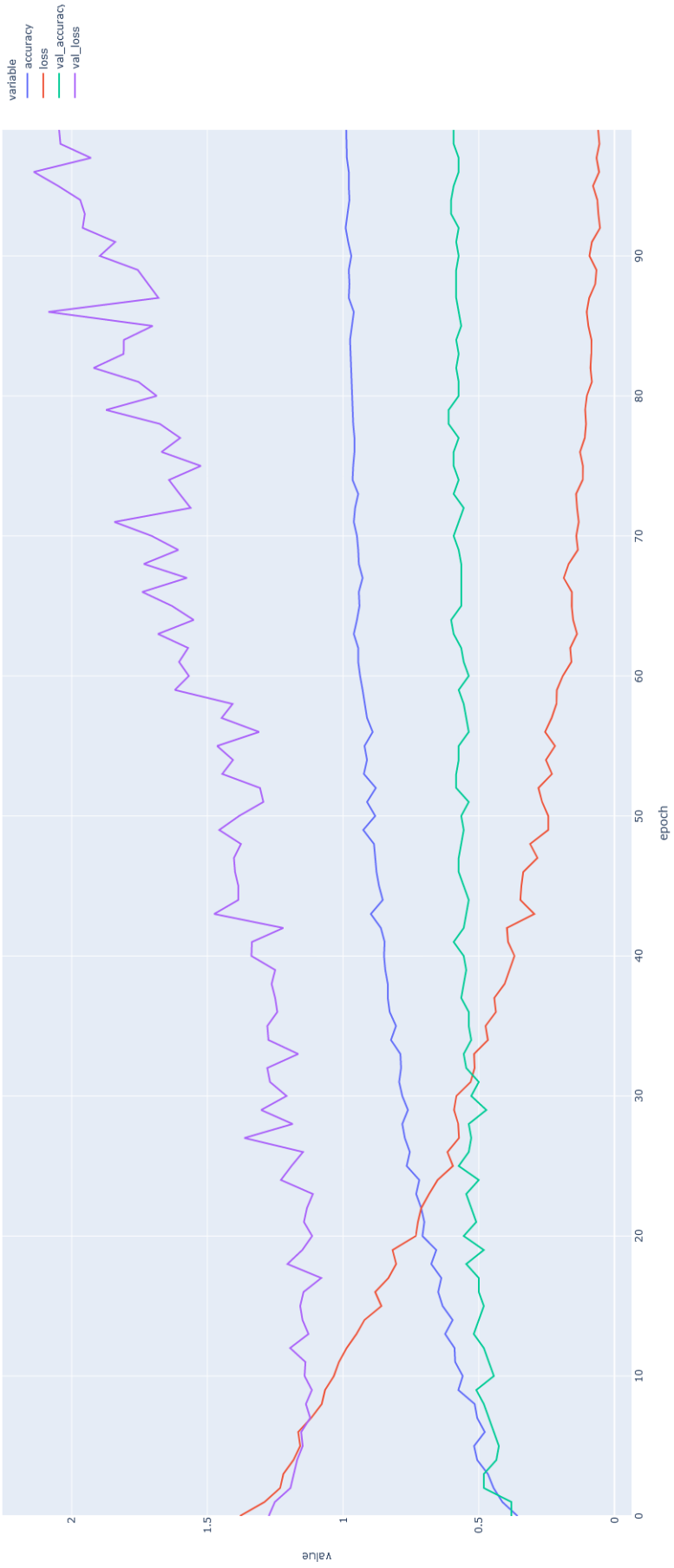

Abbildung 29.: Visualisierung der Metriken - Versuch 2

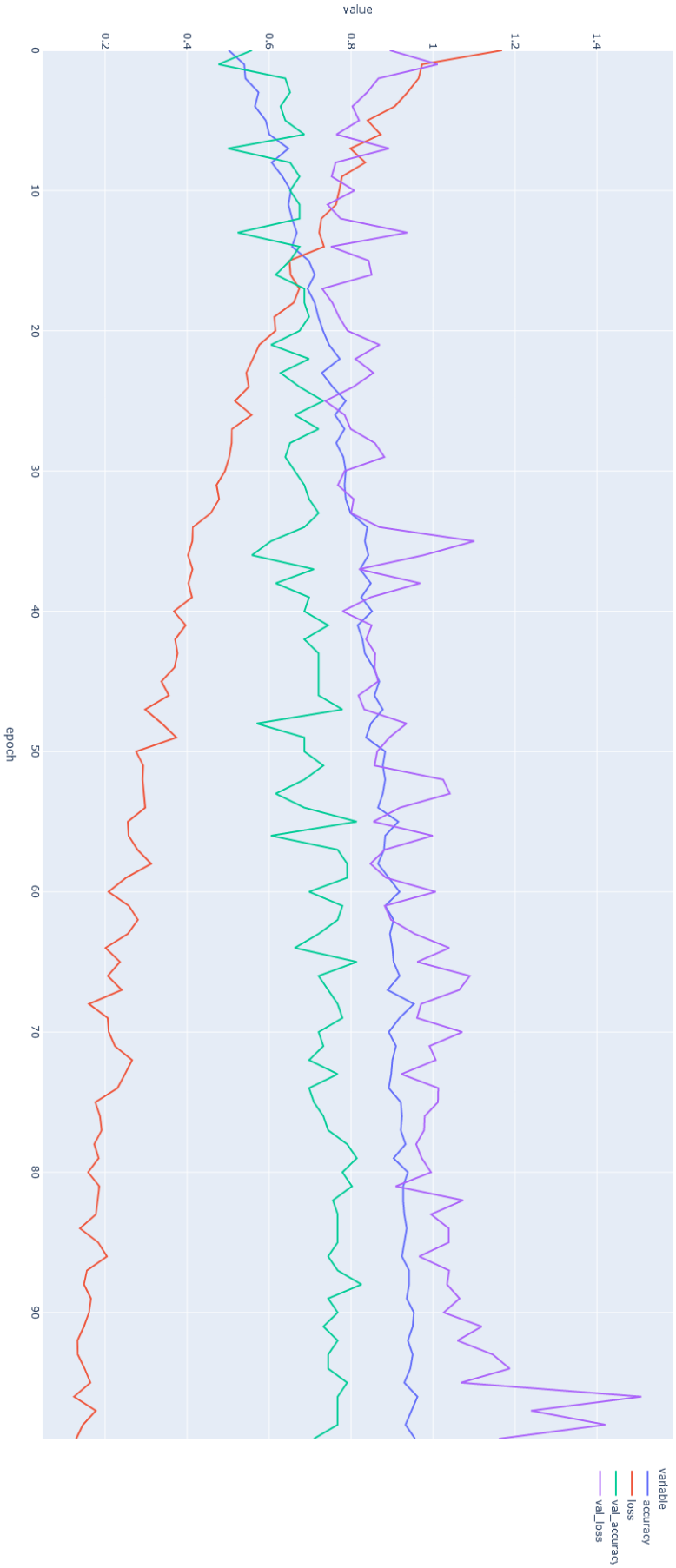

Abbildung 30.: Visualisierung der Metriken - Versuch 3

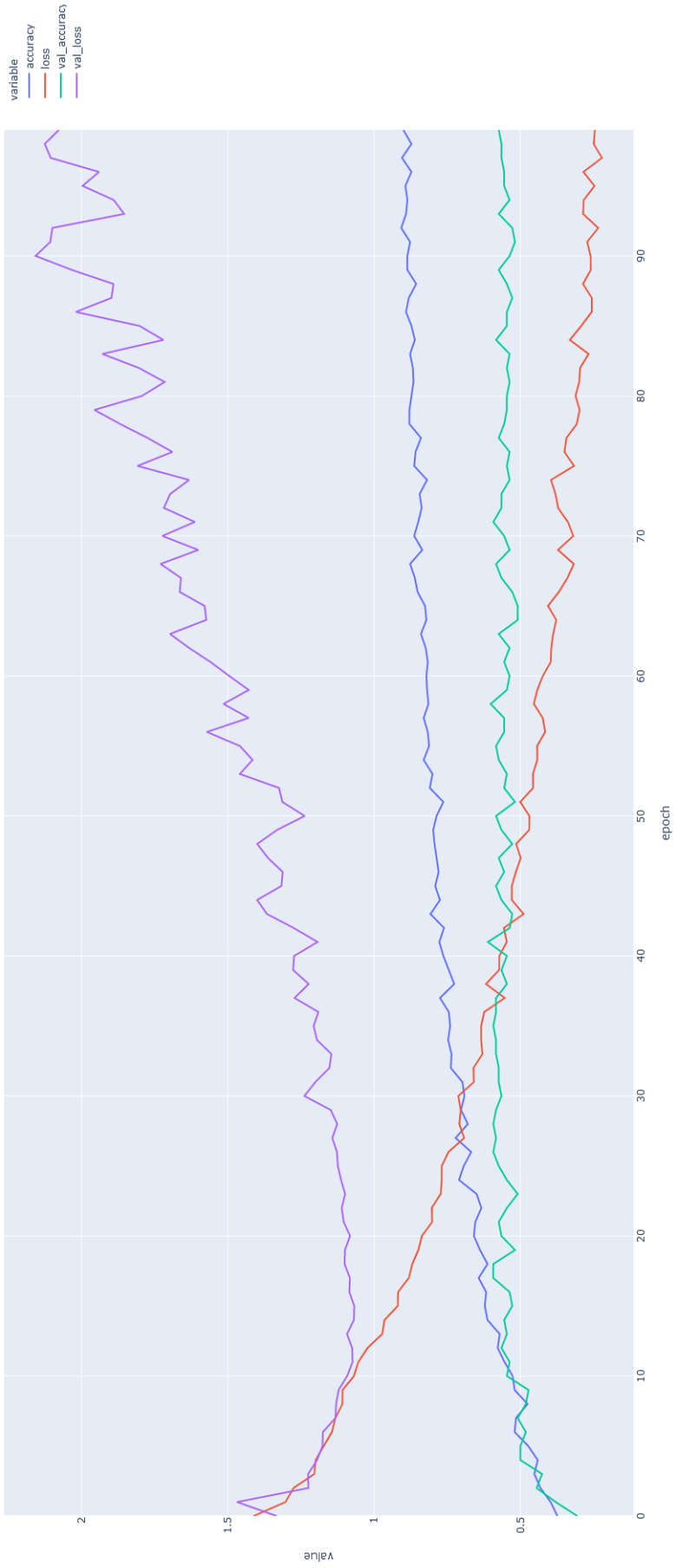

Abbildung 31.: Visualisierung der Metriken - Versuch 4

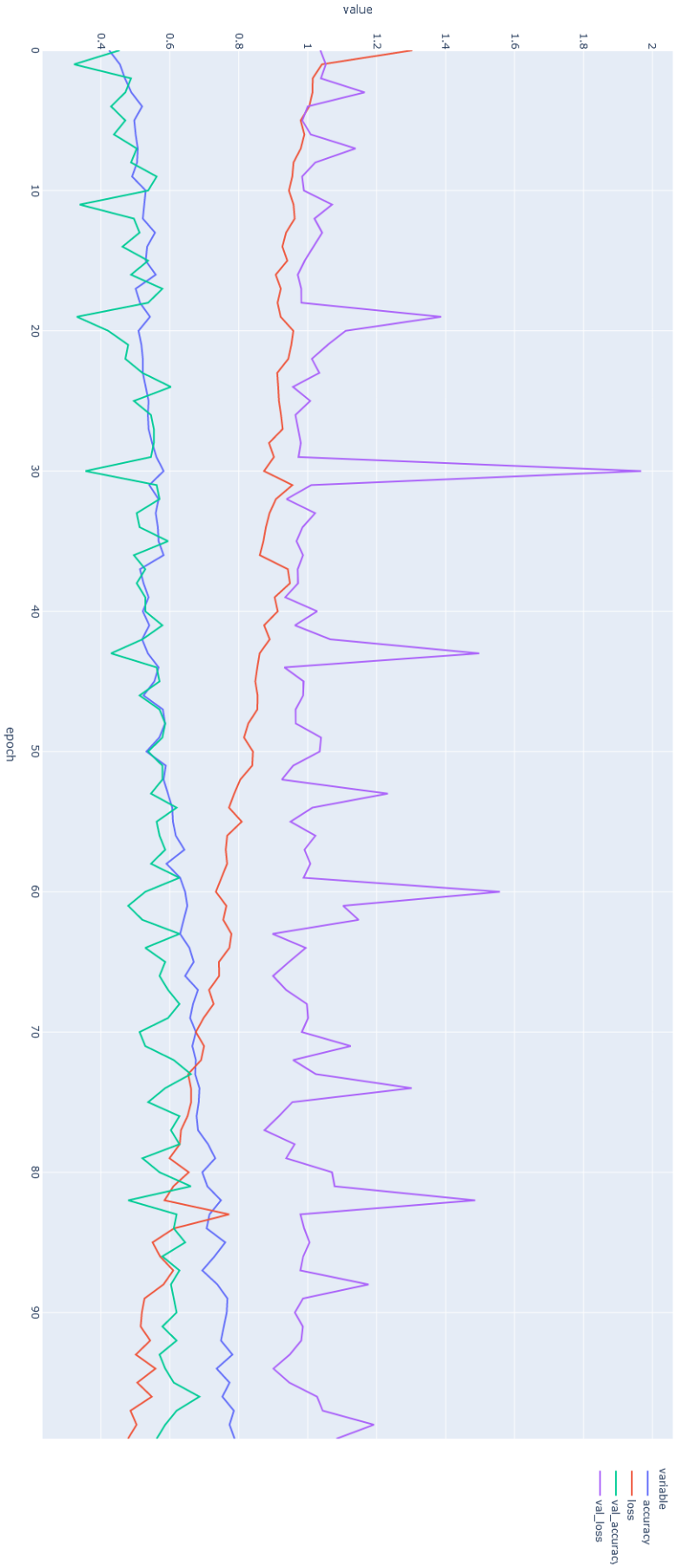

Abbildung 32.: Visualisierung der Metriken - Versuch 5

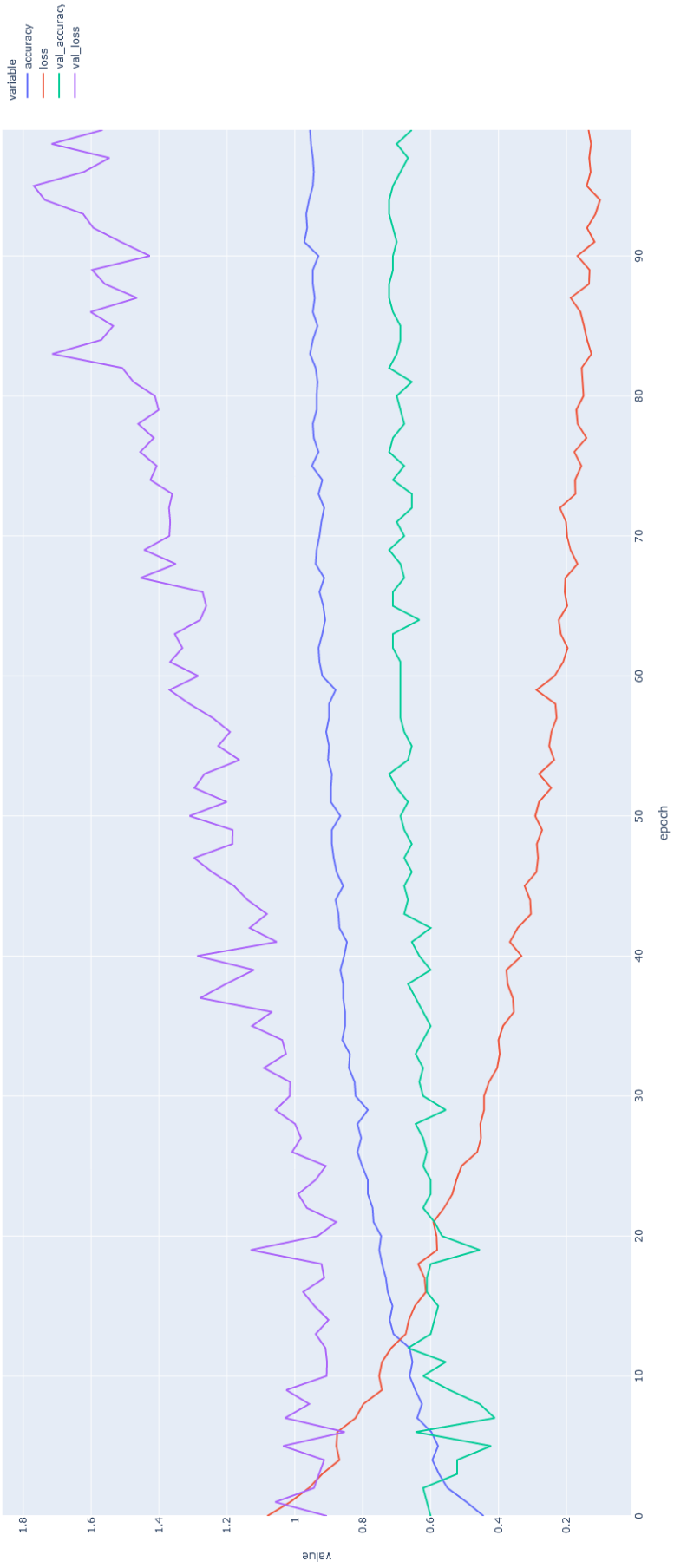

Abbildung 33.: Visualisierung der Metriken - Versuch 6

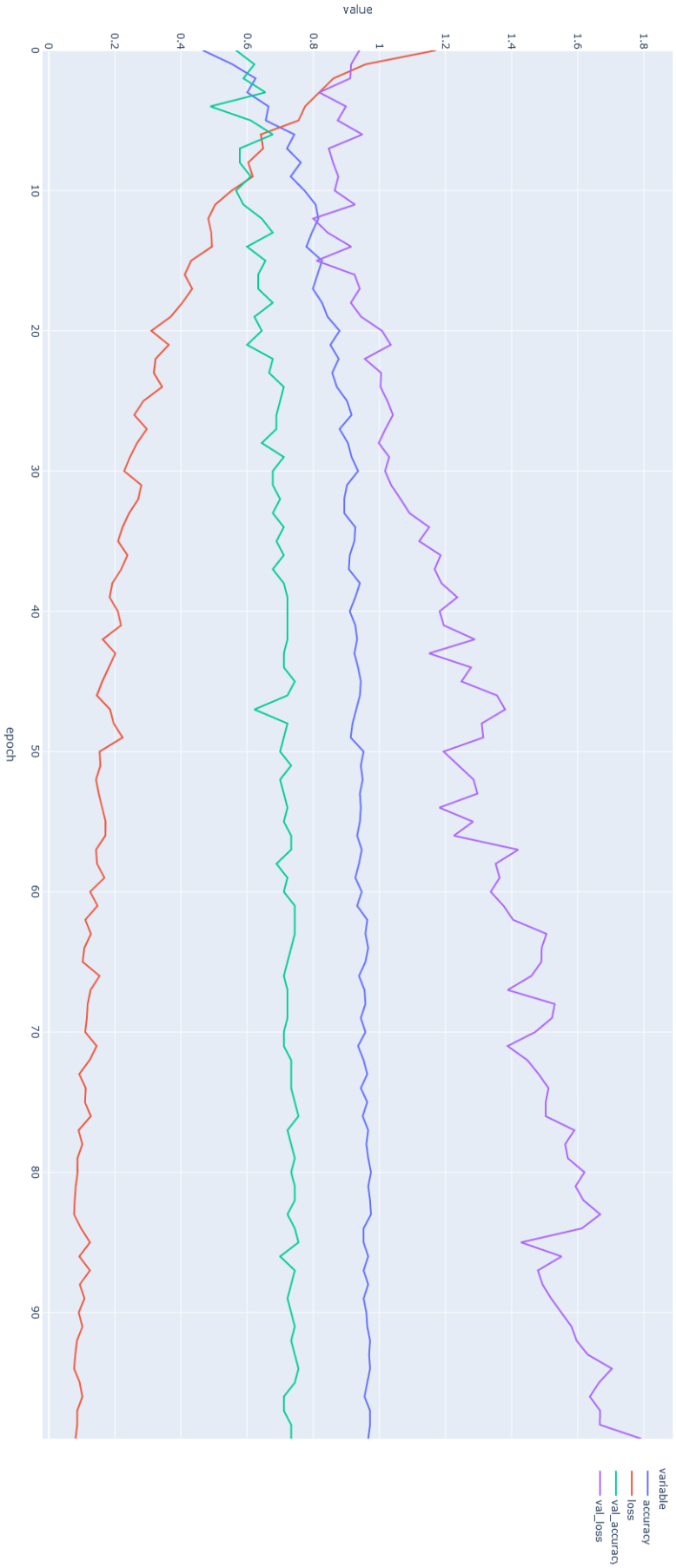

Abbildung 34.: Visualisierung der Metriken - Versuch 7

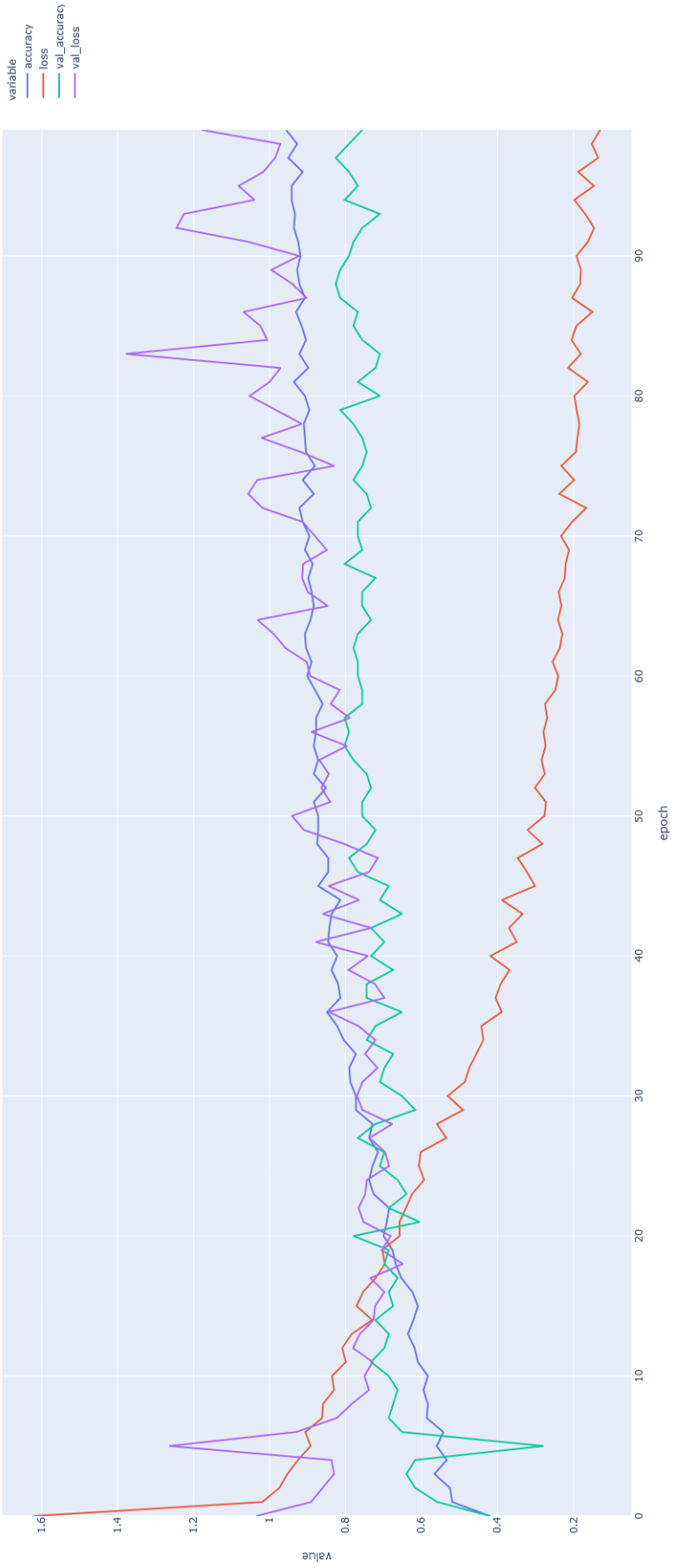

Abbildung 35.: Visualisierung der Metriken - Versuch 8

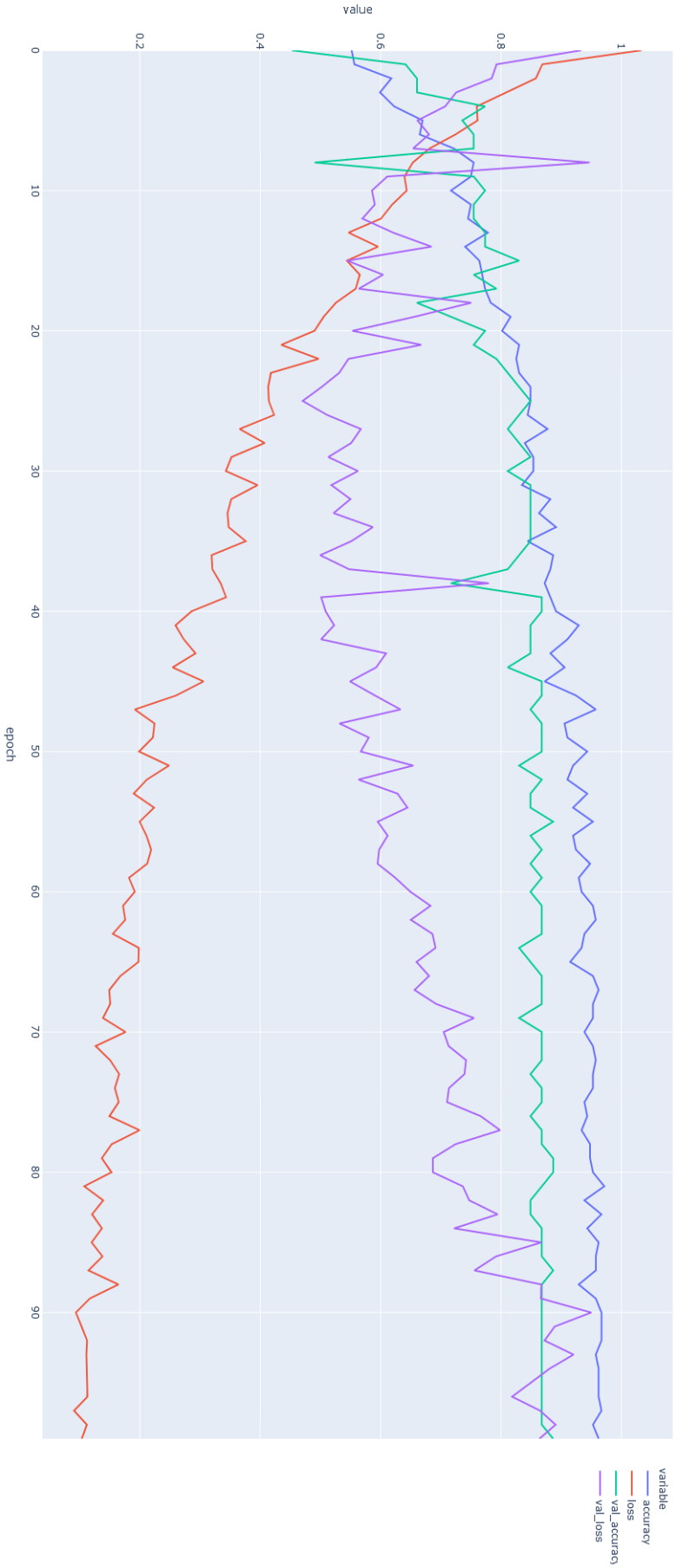

Abbildung 36.: Visualisierung der Metriken - Versuch 9