



Datengenerierung mittels eines neuronalen Netzwerks für eine datengetriebene Gesichtsanimation unter der Verwendung von Facial Action Units

Dominik Walczak
765692

Masterthesis
zur Erlangung des akademischen Grades Master of Science (M.Sc.)
Studiengang Human-centered Computing

Fakultät für Informatik
Hochschule Reutlingen

Abgabe: 29.03.2021

Erstprüfer: Prof. Dr.-Ing. Cristóbal Curio, Hochschule Reutlingen
Zweitprüfer: Prof. Dr. Uwe Kloos, Hochschule Reutlingen

Walczak, Dominik:

Datengenerierung mittels eines neuronalen Netzwerks für eine datengetriebene Gesichtsanimation unter der Verwendung von Facial Action Units –
Masterthesis, Reutlingen: Hochschule Reutlingen, 2021. 76 Seiten.

Walczak, Dominik:

Data Generation through Neural Networks for Data-Driven Facial Animations using Facial Action Units –
Master Thesis, Reutlingen: Reutlingen University, 2021. 76 pages.

Abstract

Datengenerierung mittels eines neuronalen Netzwerks für eine datengetriebene Gesichtsanimation unter der Verwendung von Facial Action Units

In der vorliegenden Master-Thesis wird ein neues System zur Generierung von Gesichtsanimationsdaten mit Bezug zum Facial Action Coding System vorgestellt und kritisch bewertet. Die mit diesem System generierten Daten können dabei für die Animation eines virtuellen Avatars in der Spieleengine Unity verwendet werden. Das entwickelte System basiert auf den Methoden neuronaler Netzwerke, wobei die Kombination aus Long-Short-Term-Memory-Zellen (LSTM) und der Autoencoder-Architektur das Grundgerüst des Systems bilden. Das trainierte System ist in der Lage aus den Daten zweier unterschiedlicher Gesichtsausdrücke die Daten zwischen diesen Gesichtsausdrücken zu generieren und stellt eine einfache Alternative zu herkömmlichen Methoden zur Erstellung von Gesichtsanimationen dar.

Zur Erstellung eines geeigneten Trainingsdatensatzes müssen hierfür zunächst Motion Capture-Aufnahmen unterschiedlicher Gesichtsausdrucksübergänge aufgenommen und verarbeitet werden. Die auf diese Weise gewonnenen Daten können anschließend für das Trainieren, Validieren und Testen des Systems verwendet werden. Um das Lernverhalten des Systems genauer zu untersuchen wird das Modell mit unterschiedlichen Zusammensetzung der Trainingsdaten trainiert. Die Ergebnisse der verschiedenen Modelle liefern wichtige Erkenntnisse über die optimale Zusammensetzung der Beispiele in den Trainingsdaten.

Data Generation through Neural Networks for Data-Driven Facial Animations using Facial Action Units

In this master thesis a new system for the generation of facial animation data with reference to the Facial Action Coding System is presented and critically evaluated. The data generated with this system can be used for the animation of a virtual avatar in the Unity game engine. The developed system is based on the methods of neural networks where the combination of Long Short Term Memory (LSTM) cells and the autoencoder architecture form the basics of the system. The trained system is able to generate the data inbetween two differing facial expressions and therefore presents a simple alternative to conventional methods for creating facial animation.

To create a suitable training dataset motion capture images of different facial expression transitions must first be captured and processed. The data obtained this way can then be used to train, validate and test the system. To further investigate the learning behaviour of the system, the model is trained with different combined training data. The results of those different models provide important insights about the optimal composition of the examples within the training data.

Inhaltsverzeichnis

1. Einleitung	1
1.1. Motivation	1
1.2. Ziel der Arbeit	3
1.3. Aufbau der Arbeit	4
2. Grundlagen	5
2.1. Künstliche Neuronale Netze	5
2.1.1. Recurrent Neural Network	8
2.1.2. Long Short Term Memory	10
2.1.3. Autoencoder	11
2.2. Blendshape-Modelle	12
2.2.1. Keyframe Animationen	13
2.2.2. Performance-Driven Animationen	14
3. Stand der Technik	15
4. Anforderungen	21
4.1. Vorstellung der Trainingsdaten	21
4.2. Erstellung der Trainingsdaten	22
4.3. Beschreibung des Datensatzes	24
4.4. Das System	26
5. Umsetzung	28
5.1. Datenvorverarbeitung	28
5.2. Entwicklung eines KNN-Systems	32
5.2.1. Vergangenheitssequenz-Vektor x	34
5.2.2. Ziel-Frame x_Z	35
5.2.3. LSTM-Encoder	35
5.2.4. Recurrent-Generator	36
5.2.5. Linearer-Decoder	37
5.3. Training des KNN-Systems	38
5.3.1. Fehlerfunktion	39
5.3.2. Optimierer	40
5.4. Generierung von neuen Animationen	42

5.5. Durchführung der Evaluation	42
6. Ergebnisse	45
6.1. Training	45
6.2. Expression-Generator	47
6.2.1. Absoluter Fehler	47
6.2.2. Datensatzfremde Animationen	53
6.3. Quantitative Evaluation	55
6.3.1. Definition der Kriterien	55
6.3.2. Numerische Analyse	56
7. Diskussion	62
7.1. Datensatz	62
7.2. Expression-Generator	64
7.2.1. Training des Modells	64
7.2.2. Design des Modells	65
7.2.3. Generierte Animationen	66
7.3. Limitierungen des Systems	69
7.4. Quantitative Evaluation	70
8. Fazit	74
Abkürzungsverzeichnis	vi
Tabellenverzeichnis	vii
Abbildungsverzeichnis	viii
Literatur	x
A. Python Code	xiv
B. Videos	xv
C. Numerische Analyse	xvi

Kapitel 1

Einleitung

1.1. Motivation

Die Mimik unserer Gesichter nimmt in unserem Leben eine zentrale Rolle ein, denn sie ist unser stärkstes, nonverbales Mittel zur zwischenmenschlichen Kommunikation und stellt gleichzeitig einen wesentlichen Bestandteil unserer Identität dar [1]. Aus einem einzelnen Gesichtsausdruck lassen sich zahlreiche Informationen, wie beispielsweise der emotionale Zustand einer Person, unabhängig von Alter, Geschlecht, Ethnizität oder Ethik, extrahieren. Es liegen Beobachtungen vor, dass sich die Gesichtsausdrücke eines Menschen in sechs universell geltende Gesichtsausdrücke unterteilen lassen: Angst, Ekel, Freude, Traurigkeit, Überraschung und Wut [2], [3], wobei andere Ansätze mit bis zu 21 universell geltenden Gesichtsausdrücken arbeiten. [4]. Neben psychoanalytischen Forschungen gibt es auch Entwicklungen im Bereich der Informatik, welche sich intensiver mit der Analyse von Gesichtern auseinandersetzt. Neben dem Einsatz animierter und virtueller Avatare in Filmen oder Videospielen in der Unterhaltungsbranche finden solche Avatare auch Anwendungen in der digitale Analyse von Gesichtsausdrücken zur frühzeitigen Depressionserkennung in der Medizin oder auch neuerdings die Synthese von Gesichtern in der Robotikindustrie, in welcher Roboter menschliche Emotionen vermitteln können sollen, Einsatz [5]. Der Fokus dieser Arbeit liegt jedoch klar auf dem Bereich von virtuellen Avataren und deren Animation. Die Ausdrücke eines Avatars können mittels unterschiedlicher Verfahren bearbeitet werden, wobei sich all diese Verfahren in zwei grundlegende Methoden unterteilen lassen. Zum einen die Ausdruckbearbeitung anhand geometrischer Modelle und zum anderen die Ausdrucksbearbeitung mithilfe generativer Modelle. Bei geometrischen Modellen wer-

den bestimmte Gesichts- bzw. Ausdrucksattribute eines Gesichts auf beispielsweise ein 3D Morphable Model (3DMM) übertragen, wohingegen bei generativen Modellen zunächst die Ausdrucksbeschreibungen aus einem umfangreichen Datensatz gelernt werden müssen, bevor ein passender Algorithmus gelernte Gesichtsausdrücke auf ein anderes Gesicht übertragen kann. [6]

Das letztere Animationsverfahren bedient sich dabei zumeist an Methoden von so-nannten *neuronalen Netzen*, auf welche in einem späteren Teil dieser Arbeit genauer eingegangen wird. Ein wesentliches Problem der verfügbaren, generativen Methoden ist, dass sich die Ausdrucksbeschreibungen zumeist auf nur sehr oberflächliche Beschreibungen wie „Freude“ oder „Traurigkeit“ beziehen. Das Lernen von präziseren und objektiveren Beschreibungen, wie zum Beispiel Beschreibungen nach Facial Action Coding System (FACS) [7], findet nur selten statt. Die Vorteile von Beschreibungen nach FACS sind beispielsweise die Vermeidung des subjektiven Empfindens sowie die eindeutige Beschreibung eines Gesichtsausdrucks, da nach FACS jeder Gesichtsausdruck durch die Kombination von elementaren Komponenten, den Facial Action Units (AUs), beschrieben werden [7]. AUs repräsentieren dabei Hautdeformationen, bei welchen mehrere Muskel beteiligt sein können. Dabei sind zum einen die eingesetzten AUs mit ihrer Nummerierung von Bedeutung, zum anderen auch die Intensitäten, welche mit den Buchstaben von minimal *A* bis maximal *E* definiert sind [7]. Durch diese Taxonomie lässt sich beispielsweise ein Gesicht, bei dem lediglich der Mund maximal geöffnet ist, durch die *AU27E* objektiv codieren, ohne dass subjektive Eindrücke die Beschreibung verfälschen. Daraus folgt, dass die Codierung eines Gesichtsausdrucks mit FACS allgemeingültig ist. Die visuellen Darstellungen aller in dieser Thesis verwendeten AUs sind in Kapitel 4.1 in den Abbildung 4.1 und 4.2 zu finden.

Die detaillierten Beschreibungen nach FACS bringen jedoch neue Herausforderungen mit sich, wie etwa die Notwendigkeit von Expertenwissen um entsprechende Bilder oder Videos zu annotieren sowie den hohen Zeitaufwand für diesen Prozess [8]. Folglich stehen Datensätze mit qualitativen Annotationen nur selten zur Verfügung [9] und sind darüber hinaus aufgrund fehlender AU-Kombinationen nur bedingt brauchbar [10]. Die Möglichkeit qualitative Daten mit FACS-Bezug generieren zu können, wäre daher ein großer Schritt für die Wissenschaft. Solche Datensätze könnten beispielsweise in der psychoanalytischen Forschung Einsatz finden, bei welcher anhand der Analyse von Gesichtsausdrücken Depressionen frühzeitig festgestellt werden können, aber auch in der Informatik könnten mithilfe großer

Datensätze Systeme weiterentwickelt werden, in denen Maschinen versuchen AU-Aktivierungen zu erkennen [10]. Nicht nur der aufwändige Aufnahme- bzw. Annotationsprozess würde umgangen werden, es würden sich zudem auch ganz neue Möglichkeiten für die Gesichtsanimation bieten. Im Bereich von virtuellen Avataren in Videospielen könnte dies bedeuten, dass ein computergesteuerter Avatar seine Animationen dem Verhalten des Spielers anpasst, ohne dass hierfür jegliche Kombinationen von Gesichtszügen von Hand modelliert werden müssen. Im Bereich der Bewegungssynthese von ganzen Körpern ist die Wissenschaft hingegen deutlich weiter. Hier kommen generative neuronale Netzwerkarchitekturen bereits vermehrt zum Einsatz [11], [12], [13]. Netzwerke, welche mit großen, hochdimensionalen Datensätzen trainiert werden, können bereits eine Vielzahl von nutzerbestimmten, komplexen Bewegungsabläufen generieren [14]. Dadurch stellt sich die Frage, inwiefern sich die Techniken der Bewegungssynthese von Körpern auf die Synthese von Gesichtsausdrücken übertragen lassen, um qualitative Daten (bspw. mit FACS-Bezug) für die Gesichtsanimation generieren zu können.

1.2. Ziel der Arbeit

Das höhergestellte Ziel, dem diese Arbeit zuträgt, ist die Unterstützung der Gesichtsausdrucksforschung sowie dem Bereich der Informatik, welcher sich mit der Animation von Gesichtern beschäftigt. Hierbei steht nicht zwingend die Animation selbst im Vordergrund, sondern vielmehr die Möglichkeit neue, plausible Daten für Gesichtsausdrücke generieren zu können, um so die wenigen FACS-annotierten Datenbanken [9], [10], zum einen für die psychoanalytische Forschung und zum anderen für die Erstellung von Animationen in der Unterhaltungsbranche, zu erweitern. Hierfür werden mithilfe von [15] zunächst AU-Intensitäten aus Motion Capture-Aufnahmen für Ausdrucksübergänge gewonnen. Daraus ergibt sich die folgende Hauptforschungsfrage, mit welcher sich die vorliegende Arbeit beschäftigt:

Wie können Animationsdaten mit FACS-Kodierung mittels eines generativen Netzwerks generiert werden?

Zur Beantwortung dieser Forschungsfrage wurden zudem folgende Teilfragen definiert, welche signifikant zur Problemlösungsfindung beitragen und deren Beantwortung einen wesentlichen Bestandteil dieser Arbeit darstellen:

1. Wie werden Animationsdaten im Bereich der Ganzköpersynthese generiert?

2. Wie können die in 1. ermittelten Architekturen auf die eigene Problemstellung und die eigene Datenstruktur angepasst werden?
3. Wie können generierten Daten überprüft werden?

Durch die Beantwortung dieser Teilfragen lassen sich somit folgende Ziele festhalten: Zu Beginn müssen geeignete Methoden der Ganzkörpersynthese recherchiert und anschließend evaluiert werden, darauf folgt eine Anpassung der ermittelten Architektur auf die eigene Problemstellung und deren Code-Implementierung. Zuletzt müssen die generierten Daten kritisch evaluiert werden, um die Eignung einer solchen Architektur zur Gesichtsausdrucksdatengenerierung begutachten zu können.

1.3. Aufbau der Arbeit

Die für die Klärung der gestellten Forschungsfrage notwendigen Grundlagen werden in Kapitel 2 beschrieben, bevor in Kapitel 3 Technologien und Methoden aus relevanten Arbeiten vorgestellt werden. Anschließend folgt in Kapitel 4 eine Beschreibung der für das zu entwickelnde System geltenden Anforderungen sowie eine Vorstellung der vorhandenen Daten. In Kapitel 5 wird die Umsetzung eines in dieser Arbeit entwickelten Systems zur Generierung von Animationsdaten beschrieben, dessen Ergebnisse in Kapitel 6 zunächst vorgestellt und anschließend in Kapitel 7 ausführlich bewertet und diskutiert werden. In Kapitel 8 wird daraufhin ein Fazit gezogen, indem die Forschungsfrage beantwortet wird.

Kapitel 2

Grundlagen

In diesem Kapitel werden die grundlegenden Technologien beschrieben, die für ein besseres Verständnis der in dieser Arbeit eingesetzten Methoden erforderlich sind. Hierbei handelt es sich vorwiegend um Künstliche Neuronale Netze (KNNs) und Blendshape-Modelle wie sie in dieser Arbeit Anwendung finden.

2.1. Künstliche Neuronale Netze

KNNs sind einem dem menschlichen Gehirn nachgebildete Informationsverarbeitungsstruktur, welche während eines Trainingsvorgangs umstrukturiert werden können [16]. Dieser Vorgang bildet den Ausgangspunkt des Lernens eines KNNs. Ein solches KNN besteht, wie der Name bereits sagt, aus mehreren miteinander verbundenen künstlichen Neuronen, welche je nach Netzart unterschiedlich miteinander verbunden sein können [16], [17]. Durch die verschiedenen Verbindungsarten entwickelt ein Netzwerk spezielle Eigenschaften, von welchen in unterschiedlichen Anwendungsgebieten Gebrauch gemacht wird. Neben der Fähigkeit aus Informationen zu lernen, haben die meisten dieser Netzarten gemein, dass sie auf drei verschiedene Arten von Neuronen basieren [16], [17]:

1. **Input-Units** nehmen Signale von der Außenwelt auf.
2. **Hidden-Units** sind Neuronen, welche in keiner Form mit der Außenwelt kommunizieren.
3. **Output-Units** geben Signale an die Außenwelt ab.

KNNs werden oftmals aufgrund ihrer Eigenschaften dem Bereich Künstliche Intelligenz (KI) aus dem Gebiet *Machine Learning* zugeordnet. Mehrere Neuronen gleicher Art werden als Schicht (englisch: layer) zusammengefasst. Ein KNN besitzt in der Regel jeweils eine Input- als auch eine Output-Schicht und kann zudem eine beliebige Anzahl an versteckten Schichten besitzen.

Elementar für das Verständnis von KNNs ist die Eigenschaft, dass die Informationsverarbeitung als mathematisches System angegeben werden kann. Das Ziel hinter diesem System ist die Approximation einer Funktion f , welche einen zuvor bestimmten Output y_i bestmöglich aus dem verfügbaren Input x_i abbildet. Die gesuchte Funktion wird dabei aus einer passende Funktionsklasse F ermittelt und bestimmt gleichzeitig die Eigenschaften des Netzwerks. Hierunter fallen z.B. die Anzahl der Hidden-Layers oder die Anzahl der einzelnen Neuronen pro Layer. [16], [18]

Die Implementierung eines einzelnen Neurons kann durch folgende von McCulloch und Pitts formulierte Funktionen beschrieben werden [19]:

$$u = \sum_{k=1}^n w^l x^l - \theta \quad (2.1)$$

und

$$y = g(u) \quad (2.2)$$

Wobei $x = (x_1, \dots, x_n)$ die Eingangssignale in das Neuron sind, während $w = (w_1, \dots, w_n)$ die Gewichtungen der Eingangssignale x , θ die Verzerrung (englisch: bias) des jeweiligen Neurons und u das Aktivierungspotenzial angeben. In der zweiten Formel von McCulloch und Pitts beschreibt y das Ausgangssignal des Neurons, welches aus dem Aktivierungspotenzial u und einer Aktivierungsfunktion $g()$ berechnet wird [20] [18] [17]. Ein vollständiges KNN lässt sich durch die Komposition mehrerer Funktionen (aufgeführt durch \circ) $g^{(L)}$, wobei L die Anzahl der Hidden-Schichten angibt, beschreiben [18]:

$$h^{(L)} = g^{(L)} \circ g^{(L-1)} \circ \dots \circ g^{(1)}(x) \quad (2.3)$$

$h^{(0)}$ beschreibt den Input in das Netzwerk. $h^{(l)}$ kann rekursiv folglich durch $h^{(l)} = g^{(l)}(h^{(l-1)})$ für alle $l = (1, 2, \dots, L)$ bestimmt werden.

Damit das KNN wie oben angegeben lernen kann, muss darüber hinaus eine Metrik definiert werden, anhand derer bestimmt wird, wie gut das KNN beispielsweise $x_i \mapsto y_i$ abbilden kann. Dabei besteht die Annahme der Existenz einer Funktion $f(x_i) = y_i$. Konkret bedeutet dies, dass eine solche Metrik beispielsweise die Distanz $D = \hat{f}(x_i) - y_i$, wobei \hat{f} die unangepasste Ausgangsfunktion des KNNs beschreibt [18]. In der Fachsprache wird von einem Verlust oder verallgemeinert von einer Verlustfunktion gesprochen. Das eigentliche Lernen findet durch die zwei letzten Konzepte von KNNs statt, der *Differenzierung der Verlustfunktion* sowie der *Rückpropagierung* (englisch: backpropagation) [16].

Nachdem der Verlust der Funktion \hat{f} berechnet wurde, findet zunächst die Differenzierung der Verlustfunktion statt. Dadurch kann berechnet werden, welche Parameter welchen Einfluss auf $\hat{f}(x_i)$ haben. Im Falle einer eindimensionalen Funktion würde diese Formel die Steigung angeben, da KNNs jedoch zumeist mehrdimensionale Funktionen beschreiben, werden folglich Gradienten berechnet. Der Gradient liefert Informationen darüber, wie sehr ein Parameter des KNNs die Verlustfunktion tatsächlich beeinflusst und auf welche Art (Erhöhung, Verringerung oder keine Veränderung) dieser angepasst werden sollte um den Verlust zu verringern.

Während der Rückpropagierung werden die ermittelten Parameter rückwärts, also entgegen der eigentlichen Berechnungsrichtung des KNNs, angepasst. Dadurch dass sich die Funktion des KNNs in seine einzelnen Schichten und diese wiederum in seine einzelnen Neuronen aufteilen lässt, lässt sich gleichzeitig die Differenzialgleichung der Verlustfunktion in ihre Einzelteile zerlegen [17], [20], sodass jeder einzelne Parameter des gesamten Netzes individuell angepasst werden kann.

Insgesamt gibt es drei Haupttypen für das Lernen von KNNs:

(1) **Überwachtes Lernen** (englisch: supervised learning):

Beim überwachten Lernen sind sowohl die Eingabedaten, als auch die gewünschten Ausgabedaten bekannt. Solche Datensätze werden als *labeled* (deutsch: beschriftet, annotiert) bezeichnet. Ziel des überwachten Lernens ist es, ein Modell zu entwickeln, welches für unbekannte Eingabedaten die richtige beziehungsweise gewollte Ausgabe ausgeben kann [16] [17].

(2) **Unüberwachtes Lernen** (englisch: unsupervised learning):

Im Gegensatz zum überwachten Lernen stehen beim unüberwachten Lernen nicht zwingend der Ausgang, sondern vielmehr die Datenstruktur oder die Gemeinsamkeiten der Eingabedaten im Vordergrund [18]. Des Weiteren sind die Daten nicht

beschriftet oder annotiert. Ein bekanntes Ziel des unüberwachten Lernens ist beispielsweise das Clustern von Daten [17].

(3) **Bestärkendes Lernen** (englisch: reinforcement learning):

Bestärkendes Lernen unterscheidet sich deutlich von den zwei bisher genannten Lerntypen, da beim bestärkenden Lernen das KNN auf Basis der Eingabedaten bestimmte Aktionen ausführt und dadurch einen intelligenten Agenten darstellt. Dieser Agent wird durch einen dafür festgelegten Lernalgorithmus für zielbringende Aktionen belohnt, wohingegen er für falsche Aktionen bestraft wird. [19]

Aufgrund der Tatsache, dass KNNs durch ihre Architekturen unterschiedliche Eigenschaften besitzen, können sie in unterschiedlichen Anwendungen verwendet werden [16].

2.1.1. Recurrent Neural Network

Eine der wesentlichen in dieser Arbeit verwendeten KNN-Architekturen ist das Recurrent Neural Network (RNN). Bei dieser Art von Netzwerk werden entweder der ganze (klassisches RNN) oder auch nur bestimmte Teile des Neuronen-Outputs (Long Short Term Memory (LSTM) oder Gated Recurrent Units (GRU)), welcher zumeist als *Hidden-Output* bezeichnet wird, als Feedbackmechanismus für die daraufliegenden Neuronen wiederverwendet [17]. Dadurch sind RNNs in der Lage sequenzielle Informationen zu verarbeiten und eignen sich für Anwendungen in denen dynamische Daten verarbeitet werden sollen [18], [20]. Des Weiteren besteht zudem eine gewisse Varianz innerhalb der RNN-Architektur:

1. One-to-one:

Bei einer *One-to-one*-Architektur handelt es sich um ein klassisches KNN, wie es zuvor beschrieben wurde.

2. Many-to-one:

Bei *Many-to-one*-RNNs werden mehrere Eingangsdaten vom Netzwerk aufgenommen und in einen einzigen Ausgangswert verarbeitet. Ein gängiges Beispiel ist die Stimmungsanalyse von Sätzen, bei der bspw. ermittelt wird, ob ein Satz oder Textabschnitt eine positive oder negative Sentimentalität hat [18].

3. One-to-many:

Die *One-to-many*-Architektur zeichnet sich durch einen einzelnen Eingang und mehrere, zusammenhängende Ausgänge aus. Ein klassisches Beispiel ist die Generierung von Bildunterschriften, bei welcher ein Bild vom einem RNN eingelesen wird, woraufhin das RNN eine Beschreibung des Bildes durch mehrere Worte ausgibt [18].

4. Many-to-many:

Genau wie bei Many-to-one-RNNs werden auch bei *Many-to-many*-RNNs mehrere Eingangsdaten vom Netz eingelesen, jedoch werden in diesem Fall auch mehrere Ausgangsdaten vom Netz berechnet und ausgegeben. Deshalb eignen sich Many-to-many-RNNs besonders gut für die Generierung von ganzen Sequenzen wie bspw. Sätzen. Der bekannteste Anwendungsbereich von dieser Art von RNNs ist die Textübersetzung, bei der ein ganzer Satz einer Sprache zunächst eingelesen wird (asynchron) und durch die anschließende Verarbeitung in eine andere Sprache übersetzt wird [18]. Ein Many-to-many-RNN kann Sequenzen auch synchron verarbeiten, d.h. aus dem Input wird direkt ein Output berechnet und ausgegeben, bevor der nächste Input eingelesen wird. Diese Art von RNNs werden bspw. bei Videoklassifikationen eingesetzt, bei denen jedes Bild eines Videos ein passendes Label zugewiesen bekommt [11].

Die einzelnen RNN-Architekturen sind in Abbildung 2.1 dargestellt und anhand der gleichnamigen Bezeichnung zu erkennen. Die linke Variante des Many-to-many-RNNs visualisiert die asynchrone Verarbeitung einer ganzen Sequenz, wohingegen die rechte Variante die synchrone Verarbeitung darstellt. Ein generelles Problem der

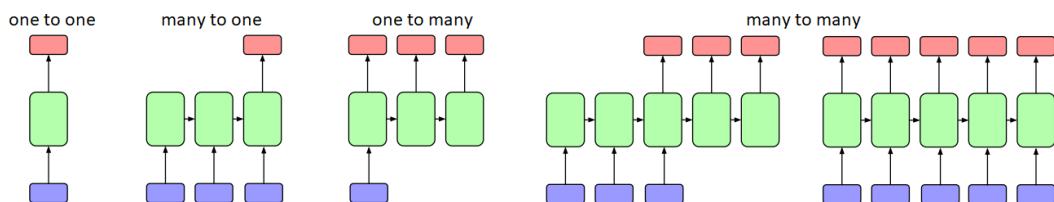


Abbildung 2.1.: Schematische Darstellung von RNNs -
Modifiziert nach [21]

klassischen RNN-Architektur ist, dass diese Schwierigkeiten haben die Zusammenhänge von langen Sequenzen aufzunehmen. Dies liegt vorwiegend an *explodierenden* oder *verschwindenden* Gradienten (englisch: exploding/vanishing gradients). Zur Erinnerung: Bei klassischen RNNs wird der Hidden-Output an die darauffol-

genden Schichten eins-zu-eins weitergegeben. Um dieses Problem zu verringern werden zusätzliche Variablen verwendet, welche bestimmen, wie sehr die vorherigen Hidden-Outputs beibehalten werden. Eine dieser verbesserten Architekturen, welche in dieser Arbeit verwendet wurde, das LSTM, soll im Folgenden genauer erläutert werden.

2.1.2. Long Short Term Memory

Das LSTM ist wie zuvor beschrieben eine verbesserte RNN-Architektur, welche die Eigenschaft besitzt das Problem des *Kurzzeitgedächtnisses* (englisch: short term memory), also das Problem von explodierenden oder verschwindenden Gradienten, zu adressieren und maßgeblich zu vermindern [18]. Dies geschieht durch eine komplexere Berechnung des Hidden-States sowie durch die zusätzliche Berechnung und Verwendung einer zweiten, rekurrenten Variable - dem Cell-State. In Abbildung 2.2 ist der Unterschied zwischen klassischen RNNs und LSTMs schematisch dargestellt. Bei einem LSTM werden zu jedem Zeitpunkt einer Sequenz drei Arten von

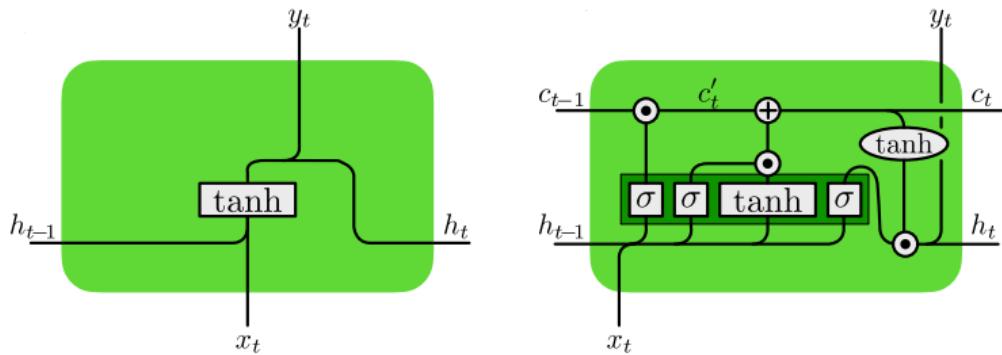


Abbildung 2.2.: Schematische Darstellung von RNNs sowie LSTMs -
Unverändert übernommen von [22]

Informationen verarbeitet und wieder ausgegeben: die Eingangsdaten x_t (englisch: Input), das Arbeitsgedächtnis h_{t-1} (englisch: Hidden-State) und das Langzeitgedächtnis c_{t-1} (englisch: Cell-State).

Im Inneren einer einzelnen LSTM-Zelle finden daher folgende Berechnungen statt:

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \\ 1 \end{pmatrix} \quad (2.4)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad (2.5)$$

$$h_t = o_t \odot \tanh(c_t) \quad (2.6)$$

In Formel 2.4 beschreibt W eine Matrix mit den Gewichten des LSTMs, h_{t-1} den eingehenden Hidden-State und x_t die Eingangsdaten. Der Parameter i_t beschreibt das *Input-Gate*, welches festlegt wie sehr der Cell-State verändert wird; f_t beschreibt das *Forget-Gate*, welches festlegt wie sehr der eingehende Cell-State für die Berechnung des neuen Cell-States verwendet wird; und zuletzt o_t , das *Output-Gate*, welches festlegt, wie sehr der Cell-State den neuen Hidden-State beeinflusst [18]. c_t sowie h_t beschreiben die genauen Berechnungen von Cell- und Hidden-State.

2.1.3. Autoencoder

Autoencoder (AE) sind eine Form unüberwachten Lernens, bei welcher Strukturen innerhalb eines Datensatzes gelernt werden sollen. Ein Autoencoder-Netzwerk besteht grundsätzlich aus zwei distinktiven Teilen: dem Encoder und dem Decoder [18]. Das Ziel klassischer AE ist die Approximation einer Funktion, welche in der Lage ist, den Eingang des Netzwerks bestmöglich nachzubilden, ohne dabei schlichtweg die Daten zu kopieren [23]. Dies geschieht zumeist durch eine sogenannte *Codierung* innerhalb der Encoder-Schicht des AEs. Diese neue Darstellung des Datensatzes wird in der Fachsprache *Hidden Code* oder *Hidden Representation* genannt [23]. Nachdem die Daten in den Hidden Code umgewandelt sind, ist es das Ziel des Decoders die Daten der Hidden Representation bestmöglich in die Ursprungsform wiederherzustellen.

In klassischen Autoencoder-Anwendungen ist die Dimensionalität des Codes wesentlich kleiner als die des Inputs, sodass nicht zwingend der rekonstruierte, bekannte Input von Interesse ist, sondern vielmehr die Informationen des Codes [23]. Eine

erweiterte Anwendungsmöglichkeit bilden sogenannte *denoising AE*, bei denen der Input lediglich unvollständige oder auch leicht abgewandelte Informationen enthält, sodass der Decoder lernen muss, die ursprünglichen also unveränderten Eingabedaten nachzubilden [18]. Solche Abwandlungen des Inputs werden in der Fachsprache als *noise* (deutsch: rauschen) bezeichnet und können die Performance des AEs verbessern [23]. AEs müssen jedoch nicht zwangsläufig darauf trainiert werden den Input bestmöglich nachzubilden. Im Falle von sogenannten *Sequenz zu Sequenz* (englisch: sequence-to-sequence)-Autoencodern lernt das System eine eingehende Sequenz zu verarbeiten, um anschließend eine andere Sequenz zu erzeugen. Bei einer solchen Sequenz kann es sich beispielsweise um Texte einer Sprache handeln, welche durch das Autoencoder-System in eine andere Sprache übersetzt werden [19]. Bei dieser Art von Autoencoder-Systemen steht der Ausgang klar im Vordergrund, weshalb ein solcher AE eine Form des überwachten Lernens darstellt, da im Gegensatz zu klassischen AE der gewünschte Output zu Trainingszeiten bekannt ist und der Code in den Hintergrund rückt. Die Funktionsweise von sequence-to-sequence-AEs bildet die Grundlage der in dieser Arbeit verwendeten Technologie zur Generierung von Animationsdaten und wird in Kapitel 3 am Beispiel der Animationssynthese weiter erläutert.

2.2. Blendshape-Modelle

Ein Blendshape-Modell ist ein lineares Modell, welches aus mehreren verschiedenen Gesichtsausdrücken, den sogenannten *Zielen*, besteht [24]. Dabei kann eine einzelne Blendshape, also ein einzelnes Ziel, einen Muskel, eine AU nach dem FACS-System, einen ganzen Ausdruck oder lediglich einen bestimmten Teil des Gesichtes beschreiben. Blendshape-Modelle sind eine beliebte Methode um Gesichter zu modellieren und folglich zu animieren. Dies liegt vor allem an ihrer Einfachheit, den vielseitigen Einsatzmöglichkeiten und der Möglichkeit sie mit anderen, teilweise komplexeren Technologien kombinieren zu können [24]. Bei einem Blendshape-Modell wird eine Gesichtspose aus der Linearkombination von mehreren, vordefinierten Blendshape-Zielen generiert. Grundlegend ist die Erstellung eines einzelnen Gesichtsausdrucks mithilfe eines Blendshape-Modells keine komplexe Aufgabe, kann aber je nach Anzahl der Blendshapes unterschiedlich viel Zeit in Anspruch nehmen. Bei der animierten Figur *Gollum* aus dem Film *Der Herr der*

Ringe betrug die Anzahl an Blendshapes beispielsweise 946 [24]. Mathematisch betrachtet lässt sich ein Ausdruck durch folgende Formel beschreiben:

$$f = b_0 + \sum_{k=0}^n w_k(b_k - b_0) \quad (2.7)$$

f steht für den Zielausdruck, welcher sich aus n (n = Anzahl der Abwandlungen) Abwandlungen b_k des neutralen Gesichtsausdrucks b_0 berechnen lässt. In dieser Formel (2.7) beschreibt w_k die Aktivierung oder auch die Gewichtung eines einzelnen Ziels (meistens zwischen $[0, 1]$), wobei ein Ziel durch die Differenz $b_k - b_0$ definiert ist. In Abbildung 2.3 ist die Veränderung einer einzelnen Blendshape visuell dargestellt. Hierbei handelt es sich um die Blendshape *Mund öffnen*, welche bei (a) nicht aktiviert ist, d.h. $w = 0$, und für (d) maximal aktiviert ist, also $w = 1$. Mithilfe eines Blendshape-Modells können Gesichtsanimationen auf unterschied-

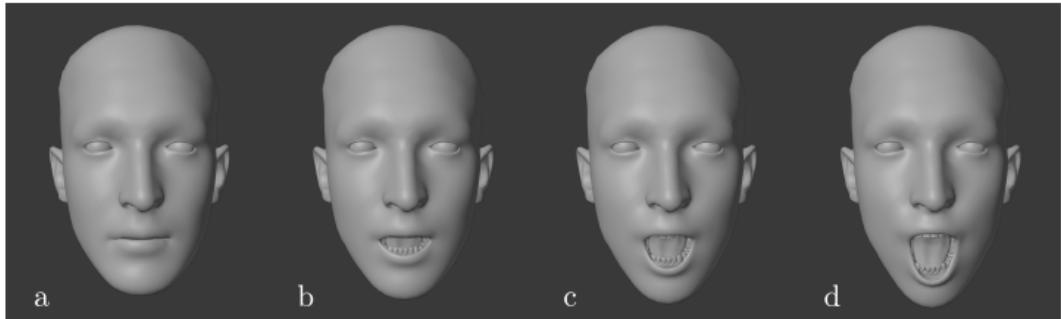


Abbildung 2.3.: Visuelle Darstellung der Aktivierung der Blendshape Mund öffnen - Unverändert übernommen von [15]

lichen Arten erzeugt werden, wobei für das Verständnis dieser Arbeit lediglich auf *Keyframe Animationen* und *Performance-Driven Animationen* eingegangen werden soll.

2.2.1. Keyframe Animationen

Keyframe Animationen beschreiben den klassischen Einsatz von Blendshape-Modellen für das Erstellen von Animationen. Ein Keyframe beschreibt eine Paarung aus einem Zeitpunkt t und eines Ausdrucks f wie in Formel 2.7 definiert. Die konkrete Animation entsteht letztens durch die Interpolation zwischen zwei oder mehreren Keyframes [24]. Die Dauer um eine Animation mithilfe von Keyframes zu erstellen hängt somit primär von der Anzahl der Blendshape-Ziele ab, da

für jeden Keyframe zunächst alle Gewichtungen der Blendshape-Ziele bestimmt werden müssen. Zumeist erfolgt eine solche Bestimmung mithilfe einfacher *Slider* (deutsch: Schieberegler). Der zweite Faktor, welcher die Dauer der Animationserstellung beeinflusst, ist die Animationslänge selbst, sowie die dazugehörige Bildwiederholungsrate.

2.2.2. Performance-Driven Animationen

Bei Performance-Driven Animationen, einer Art von Animationserstellung, wird ein Schauspieler verwendet, um dessen Emotionen und Gesichtsausdrücke auf ein Blendshape-Modell zu übertragen [24]. Die Möglichkeit Schauspieler in die Animationserstellung miteinzubeziehen ist dabei der primäre Grund, weshalb Blendshape-Modelle so beliebt sind [24]. Durch die Kombination von professionellen Schauspielern und einem vordefinierten Blendshape-Modell, kann bei Performance-Driven Animationen der zeitintensive Schritt der Bestimmung jeder einzelnen Blendshape-Gewichtung (vgl. 2.2.1) durch die Performance des Schauspielers ersetzt werden. Die Handlungen des Schauspielers werden dabei mithilfe von Motion Capture-Systemen aufgenommen und verarbeitet. Die Verarbeitung und Berechnung der Blendshape-Gewichtungen können anschließend auf unterschiedliche Art und Weise stattfinden. Diese Arbeit basiert auf dem Ansatz, dass dreidimensional (3D)-Motion Capture-Aufnahmen zur Verfügung stehen und es das Ziel ist, alle Gewichtungen eines Blendshape-Modells Frame für Frame zu berechnen. Das Problem der Berechnung der einzelnen Gewichtungen kann mithilfe von quadratischer Programmierung wie in [25] beschrieben gelöst werden.

Kapitel 3

Stand der Technik

Das Animieren von virtuellen Charakteren oder Avataren kann ähnlich wie die Generierung von Blendshape-Animationen mithilfe von Keyframe-Animationen stattfinden. Hierfür werden die einzelnen Avatarkomponenten, d.h. die Komponenten aus denen der Avatar gebildet wird, wie Torso, Kopf, etc., sowie seine getragenen Gegenstände von Hand in die passenden Positionen gebracht. Die Entwickler greifen hierfür zumeist auf eine Modellierungssoftware wie Blender¹ oder Maya² zu, welche es ihnen ermöglichen ähnlich wie bei Blendshape-Modellen zwischen den Keyframes zu interpolieren, um so letztendlich eine Animation zu erstellen. Ebenso wie bei der Animation von Blendshape-Modellen ist auch hier einer der wesentlichen Zeitfaktoren die Komplexität des verwendeten Modells oder des Avatars. Um den Zeitfaktor für die Animationserstellung zu minimieren, können Entwickler gleichermaßen auf Motion Capture-Aufnahmen zurückgreifen, um die eigenhändige Erstellung der Keyframes zu umgehen. Dabei werden ähnlich wie bei Blendshapes die Bewegungen eines Schauspielers auf einen Avatar übertragen. Dennoch bleibt ein wesentliches Problem bestehen: Der Aufnahmeprozess der Bewegungen durch den Schauspieler ist für sich ein zeit- und arbeitsintensiver Prozess, welcher gleichzeitig an teure Aufnahmesysteme gebunden ist, und daher nicht jedem Entwickler zur Verfügung steht.

Eine mögliche Lösung dieses Problems beschreibt das Forschungsgebiet *datengetriebene Bewegungssynthese*. Die datengetriebene Bewegungssynthese ermöglicht es den Entwicklern Animationen auf Grundlage von sogenannten *high-level Parametern* durchzuführen [26]. Diese Parameter beschreiben keine klassischen Keyfra-

¹<https://www.blender.org/>

²<https://www.autodesk.de/products/maya>

mes, sondern können je nach System unterschiedliche Informationen beschreiben. So kann es sein, dass in einem System eine Strecke oder Distanz, welche ein Avatar zurücklegen soll, vorgegeben wird [26] und in einem anderen System lediglich der Beginn einer Bewegung [11] spezifiziert wird. Daraufhin generiert ein passendes System bspw. die vollständige Bewegung eines Avatars. Die meisten solcher Systeme basieren auf einer Kombination von mehreren KNN-Strukturen und befassten sich zumeist nur mit den Körpern der Avatare. Methoden aus dem Gebiet von KNN-Technologien finden im Bereich von Ausdrucksbearbeitung und Gesichtsanimationen nur bedingt statt.

Aufgrund der höheren Dimensionalität von Gesichtern werden vermehrt KNN-Strukturen benötigt, welche mit dieser Art von Komplexität umgehen können [27]. Insbesondere *Variational Autoencoder (VAE)* sowie *Generative Adversarial Networks (GAN)* gelten als potente Strukturen, um relevante Resultate zu erzielen [26]. Beide dieser Technologien bauen dabei auf dem Prinzip von klassischen Autoencoder-Strukturen auf. Im Fall von VAEs wird der Hidden Code während des Trainings durch einen sogenannten *Regularisierungsparameter* ergänzt, sodass sich dessen neue Eigenschaften besser zur gezielten Generierung von Daten eignen. Konkret bedeutet die Generierung von Daten mittels VAEs, dass Daten durch eine Veränderung der Regularisierungsparameter sowie das Abtasten des Hidden Codes stattfinden [28].

GANs bestehen hingegen aus zwei wesentlichen Komponenten, die gegeneinander agieren: dem *Diskriminator* und dem *Generator*. Der Generator wird dabei für gewöhnlich durch eine Autoencoder-Struktur realisiert, welche für die Generierung der Daten zuständig ist, wohingegen der Diskriminat or ein Klassifikator ist, welcher angibt, ob es sich bei den Daten um echte oder generierte Daten handelt. Der Trick hierbei ist, dass während des Trainings dem Diskriminat or sowohl generierte, als auch bekannte Daten zur Klassifikation gegeben werden. Die Besonderheit von GANs ist, dass die beiden Komponenten gleichzeitig trainiert werden, d.h. der Generator versucht immer bessere Daten zu generieren, um den Diskriminat or zu überlisten. [29]

In [30] wird ein GAN für *Bild-zu-Bild Übersetzungen* (englisch: image-to-image-translation) für mehrere Domänen gleichzeitig implementiert. Bei der Bild-zu-Bild Übersetzung können bestimmte Aspekte eines Bildes auf ein zweites Bild übertragen werden, wobei alle Beispiele mit dem gleichen Aspekt zu einer Domäne zusammengefasst werden. Konkret bedeutet dies, dass die Gesichtsausdrücke einer

Person auf eine zweite Person übertragen werden, sofern die Domäne im Datensatz enthalten ist. Bei den in [30] gelernten Domänen handelt es sich um oberflächliche Emotionen wie glücklich oder traurig, sowie Haarfarbe, Alter, Hautfarbe und Geschlecht [30]. Darüber hinaus ist das GAN lediglich in der Lage statische Bilder zu generieren, weshalb es sich nicht zur Erstellung von Ausdrucksübergängen eignet. Es könnte jedoch dazu eingesetzt werden einen Ausdrucksübergang einer Person auf eine andere zu übertragen, wäre jedoch auf die gelernten, oberflächlichen Ausdrücke limitiert.

Um diese Limitierungen zu umgehen, wird in [31] ein GAN trainiert welches auf AUs, wie sie durch das FACS beschrieben sind, konditioniert ist. Um dies zu erreichen, wird ein Vektor welcher die einzelnen Intensitäten der AU angibt verwendet. Daraus folgt, dass Ausdrücke durch die kombinierte Aktivierung mehrere AUs generiert werden können. Darüber hinaus sind die Autoren in der Lage durch die lineare Interpolation zwischen *inaktiven AUs* und *maximal aktivierten AUs* Ausdrucksübergänge zu synthetisieren. Diese Form der Bewegungssynthese setzt jedoch Linearität bei Ausdrucksübergängen voraus. Sowohl [30] als auch [31] verwenden dabei ausschließlich einzelne Bilddaten, sodass sich diese Ansätze vorwiegend zur Manipulation von Bildern oder Videos eignen.

In Kontrast dazu wird in [32] mithilfe eines klassischen AE eine *Gesichtsverteilung* (englisch: face manifold) aus Animationsdaten gelernt. Mithilfe eines darauf abgestimmten Benutzerinterfaces können dadurch vereinfacht plausible Gesichtsausdrücke sowohl innerhalb, als auch leicht außerhalb der gelernten Verteilung synthetisiert werden [32]. Zudem können durch die Verwendung eines FACS-bezogenen Blendshape-Modells die Ausdrücke auch objektiv beschrieben werden. Das hierbei entwickelte System eignet sich dadurch für die vereinfachte Erstellung einzelner Gesichtsausdrücke, die aufgrund der gelernten Verteilung die Plausibilität des synthetisierten Ausdrucks verbessert. Die Synthetisierung von Ausdrucksübergängen wird dabei nicht thematisiert.

Neben den zuvor beschriebenen Methoden aus dem Bereich der Ausdrucksbearbeitung und Gesichtsanimationen werden im Rahmen dieser Arbeit auch Methoden aus dem Bereich der Ganzkörperbewegungssynthese betrachtet. Der Vorteil der im Folgenden beschriebenen Methoden besteht zum einen daraus, dass das Trainieren jener Methoden weniger Probleme mit sich bringt, wie bspw. dem *Mode Collapse*, bei dem der Generator durch die Fixierung seiner Gewichte nur ähnliche Samples

generiert [33], sowie dass sich die Methoden um jene komplexeren Methoden in Nachhinein erweitern lassen können.

Das Problem der Bewegungssynthese beschreibt in erster Linie den Vorgang der Generierung von sequenziellen Daten. Aufgrund der zuvor genannten Herangehensweise liegt der Fokus dieser Arbeit daher auf sogenannten *Recurrenten Autoencoder-Strukturen*, wie sie in [11], [13], [34] und [35] verwendet werden. Bei dieser Art von KNNs umfassen die Encoder- und Decoder-Schichten eines AEs eine oder mehrere RNN-Schichten. Daraus folgt, dass die RNN-Schichten des Systems die verarbeiteten Ausgänge der Encoder-Schicht und nicht die Informationen der ursprünglichen Sequenzdaten verarbeiten. Die Encoder-Schicht des AEs übernimmt hierbei die Funktion einer Featurevorverarbeitung. Die Decoder-Schicht des Gesamt-systems dient primär zur Datenstabilisierung und transformiert die Daten zudem in ein passendes Ausgangsformat. Die RNN-Schichten, zumeist in Form mehrerer übereinanderliegender LSTM-Module, befinden sich, wie zuvor erwähnt, im Mittelpunkt des Systems und erlernen die temporalen Zusammenhänge der unterschiedlichen Sequenzen. Die Verwendung eines AEs bietet darüber hinaus den Vorteil, dass die sequenziellen Daten nicht zunächst beschriftet werden müssen, da durch das Vorhandensein der ursprünglichen Sequenz sowohl Input als auch Output bekannt sind. Dies ist deshalb von Bedeutung, da das Annotieren von Video- und Motion Capture-Daten einen hohen Zeitaufwand mit sich bringt und deshalb nach Möglichkeit vermieden werden soll [26]. Daraus folgt jedoch auch, dass ein solches Recurrent-Autoencoder-System nur solche Sequenzen erzeugen kann, die den Sequenzen des Trainings grundsätzlich ähneln. Konkret bedeutet das, dass ein System welches mit Daten von „gehenden Menschen“ trainiert worden ist, bspw. nicht in der Lage sein wird, „springende Bewegungen“ zu generieren, da diese Bewegung dem Netz gänzlich fremd ist.

In Abbildung 3.1 ist die Struktur des in [11] verwendeten Recurrent-Autoencoder-Systems schematisch dargestellt. x_t beschreibt den Eingang des Systems zum Zeitpunkt t . Dieser Eingang kann grundsätzlich aus unterschiedlichen Daten bestehen. In [11] ist x_t ein Sequenz-Vektor bestehend aus *3D-Körperdaten* (englisch: body joint angles) abgebildet, welcher Informationen für jedes Gelenk (Position im dreidimensionalen Raum sowie dessen Rotationswinkel) für jeden Frame der Sequenz, enthält. Der Ausgang y_t beschreibt einen vorhergesagten oder einen generierten Vektor-Frame mit gleicher Form wie der Vektor der Eingangssequenz, nur dass es sich um einen einzelnen Frame und keine Sequenz handelt. Eine Animation kann

somit durch das wiederholte Einsetzen der Vorhersage am Ende der ursprünglichen Sequenz generiert werden. Der ursprüngliche erste Frame der Sequenz entfällt nach dem First In First Out-Prinzip. Wie in Abbildung 3.1 ebenfalls zu erkennen ist, werden die Gedächtnisvariablen des LSTMs (Hidden-State und Cell-State) innerhalb einer Sequenz beibehalten. Zu erkennen ist dies an der Verbindung zwischen $Recurrent_{t-1}$ und $Recurrent_t$.

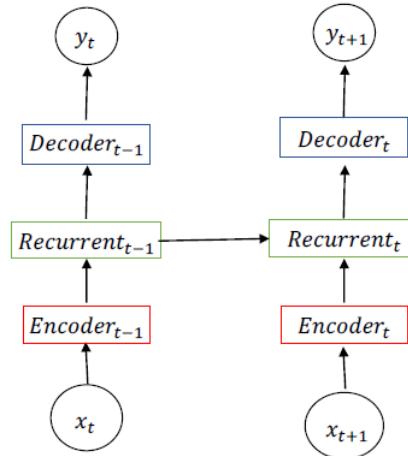


Abbildung 3.1.: Schematische Darstellung des in [11] verwendeten KNN-Systems -
Unverändert übernommen von [11]

In [13] wird das in [11] entwickelte System abgewandelt, um so komplexe, also auch nicht-lineare Übergänge zwischen zwei Frames einer Animation generieren zu können. Der Eingang des *Transition Networks* besteht hierfür aus drei wesentlichen Bestandteilen: Einer dieser Bestandteile ist ebenfalls eine Sequenz, welche eine vordefinierte Anzahl an vergangenen Frames der Animation enthält. Die anderen Bestandteile des Eingangs sind zum einen das Ziel der Sequenz oder der letzte Frame, welcher generiert werden soll, sowie zuletzt die Differenz aus aktuellem Frame (beziehungsweise 3D-Körperdaten des aktuellen Frames) und dem Ziel. Dadurch erlangt der Entwickler ein bestimmtes Maß an Kontrolle über das Generierte und verhindert zugleich sogenannte Standardposen, wie sie u.a. in [11] und [34] beschrieben werden. Diese Standardposen können entstehen sobald das System über einen längeren Zeitraum ausschließlich seiner eigenen, vorausgesagten Daten für darauffolgende Frames verwendet. Dies ist gleichzeitig die wesentliche Problematik von Recurrenten-Autoencoder-Systemen. Eine weitere Limitierung entsteht durch die Generierung selbst, denn dadurch, dass die Sequenz als Ganzes berechnet wird, können Entwickler keine stilistischen Veränderungen innerhalb der Sequenz durchführen, ohne die darauffolgenden Daten maßgeblich zu beeinflussen [26].

3. Stand der Technik

In der Literatur werden daher noch zahlreiche weitere Ansätze zur datengetriebenen Bewegungssynthese von Körpern beschrieben. Beispielsweise nutzen [26] und [36] sogenannte *neuronale Faltungsnetzwerke* Convolutional Neural Network (CNN) anstelle von RNNs, um die zuvor genannten Limitierungen zu umgehen. Auf diese und weitere Systeme soll an dieser Stelle jedoch nicht weiter eingegangen werden, da sie lediglich die große Varianz innerhalb des Forschungsgebiets aufzeigen sollen.

Zusammenfassend bleibt festzuhalten, dass es nicht nur eine Möglichkeit gibt, Animationen mittels KNN-Strukturen zu generieren, sondern zunächst die Frage beantwortet werden muss, welche Anforderungen für die Animationen gelten, und darüber hinaus definiert werden muss, welche Daten dem Entwickler zur Entwicklung eines passenden Systems zur Verfügung stehen.

Kapitel 4

Anforderungen

In diesem Kapitel werden die Anforderungen an das zu entwickelnde System sowie an die zu generierenden Animationen beschrieben. Für ein besseres Verständnis soll zunächst jedoch der Trainingsdatensatz genauer erläutert werden. Hierbei wird sowohl die Erstellung der Trainingsdaten kurz aufgeführt bevor anschließend auf das Format und die Besonderheiten der Trainingsdaten eingegangen wird.

4.1. Vorstellung der Trainingsdaten

Bei den in dieser Thesis verwendeten AUs handelt es sich um vier AUs der oberen Gesichtshälfte und sechs AUs der unteren Gesichtshälfte, welche alle durch die maximale Aktivierung des Avatars in Abbildung 4.1 und Abbildung 4.2 visualisiert werden.



Abbildung 4.1.: Visualisierung der oberen AUs mittels eines Avatars

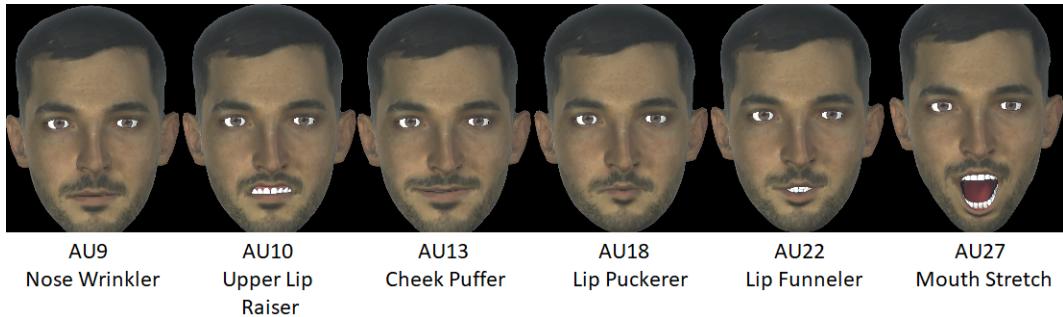


Abbildung 4.2.: Visualisierung der unteren AUs mittels eines Avatars

Für ein besseres Verständnis der verfügbaren Animationsdaten ist die Animation *neutral* zu *überrascht* stufenweise als Avataranimation in Abbildung 4.3 dargestellt, wobei der Avatar links den Startausdruck *neutral* und rechts den Zielausdruck *überrascht* darstellt. Die einzelnen Teilbilder der Animation sind nach jeweils 25 Frames aufgenommen worden, wobei die ganze Animation 125 Frames lang ist, was einer Länge von genau einer Sekunde entspricht¹.



Abbildung 4.3.: Vorstellung der Animation neutral zu überrascht Anhand des Avatars
Die Animation wird hierbei vorwiegend durch die immer stärker werdenden Intensitäten der AUs AU1, AU2, AU6 sowie AU27 definiert.

4.2. Erstellung der Trainingsdaten

Bei den verwendeten Trainingsdaten handelt es sich um Ausdrucksübergänge wie beispielsweise von *glücklich* zu *traurig*. Die Trainingsdaten werden dabei mit dem in [15] entwickelten Prototypen generiert. Der Prototyp in [15] berechnet mittels der in [25] beschriebenen Methode einzelne FACS-codierter AU-Intensitäten. Bei der in [25] beschriebenen Methode handelt es sich um quadratische Programmierung, wie sie grundsätzlich auch für die Berechnung von Blendshape-Intensitäten

¹Die Qualität der Animationen sowie Genauigkeit der AUs nach FACS werden in dieser Arbeit nicht bewertet.

eingesetzt wird. Daraus folgt, dass zunächst jeweils das *neutrale Gesicht* des Akteurs als auch die maximalen Aktivierungen für jede verwendete AU aufgenommen werden müssen, da diese einen wesentlichen Bestandteil zur Lösung von quadratischer Programmierung darstellen. Die in [25] beschriebene Formel zur Lösung von AU-Intensitäten gleicht dabei im Wesentlichen der in [24] beschriebenen Formel zur Berechnung von Blendshape-Intensitäten. Die Ähnlichkeit beider Verfahren beruht dabei auf keinem Zufall und kann anhand der Formeln zur Berechnung von Gesichtsausdrücken beziehungsweise der Berechnung von Blendshape-Modellen verdeutlicht werden. Formel 4.1 beschreibt die Berechnungen eines solchen einzelnen Gesichtsausdrucks E [25]. Dieser Gesichtsausdruck lässt sich aus dem neuralen Gesicht E_0 sowie dem Produkt aus allen Aktivierungsintensität x_i einer AU e_i berechnen, wobei m die Anzahl der Teilausdrücke angibt.

$$E = E_0 + \sum_{i=0}^m x_i e_i \quad (4.1)$$

Bei direktem Vergleich mit der in Abschnitt 2.2 beschriebenen Formel 2.7 können direkte Gemeinsamkeiten festgestellt werden. Konkret folgt daraus, dass die berechneten Gesichtsausdrücke E eins-zu-eins in einen passenden Avatar eingesetzt werden können. Für einen solchen Avatar muss jedoch gelten, dass sich dessen Zielausdruck f aus identischen AUs zusammensetzt. Der in [15] erstellte Avatar erfüllt diese Voraussetzung.

Abgeschlossen wird die Erstellung eines einzelnen Trainingbeispiels durch die Aufnahme der Ausdrucksübergänge sowie die anschließende Berechnung der AU-Intensitäten für den jeweiligen Ausdrucksübergang durch den Prototyp. Diese Intensitäten können dabei in einer .csv-Datei gespeichert werden, wobei jedoch einige Frames verloren gehen. Die verlorenen Frames werden im Rahmen dieser Arbeit mittels Interpolation geschlossen. In Tabelle 4.1 ist das Format anhand eines fiktiven Beispiels schematisch dargestellt. Dabei gibt es keine zeitlichen Einschränkun-

Tabelle 4.1.: Beispiel für das Datenformat eines Ausdrucksübergangs

Frame	AU1L	AU1R	AU2L	...	AU45R
0	0,45	0,43	1,20	...	0,14
1	0,45	0,44	1,20	...	0,15
...
249	1,20	1,19	0,43	...	0,80

gen für die Dauer eines Ausdrucksübergangs, sodass es folglich keine festgelegte Frame-Anzahl gibt.

Abbildung 4.4 visualisiert die vollständige Erstellung eines Trainingsbeispiels und ist grundlegend in die Aufgabenbereiche *Motion Captur Aufnahmen* und *Datenvorverarbeitung* aufgeteilt. Die Motion Capture Aufnahmen werden dabei mithilfe eines Vicon² Motion Capture Systems aufgenommen und durch den in [15] entwickelten Prototypen verarbeitet. Die Aufnahmen werden dabei mit 125 Frames pro Sekunde aufgenommen. Die anschließende Datenvorverarbeitung findet separat vom Aufnahmeprozess statt. Eine einzelne, vorverarbeitete csv-Datei für einen Ausdrucksübergang besteht aus zwei grundsätzlichen Informationen: dem Frame sowie der Intensität einer AU für jede verwendete AU.

4.3. Beschreibung des Datensatzes

Der in dieser Arbeit verwendete Datensatz besteht aus insgesamt 85 verschiedenen Ausdrucksübergängen, welche alle von ein und demselben Akteur durchgeführt wurden. Ein Ausdrucksübergang wird dabei durch den Übergang eines Ausdrucks zu einem anderen Ausdruck definiert. Die in dieser Arbeit verwendeten Ausdrucksübergänge nach der Datenvorverarbeitung sind in Tabelle 4.2 dargestellt. Ursprünglich wurden auch die Übergänge von und zu *wütend* sowie von und zu *ängstlich* aufgenommen, damit jeder der in [8] beschriebenen Grundausdrücke enthalten ist, jedoch mussten jene Aufnahmen während der Datenvorverarbeitung aussortiert werden (vgl. Kapitel 7). Bei den verwendeten Ausdrücken handelt es sich daher um angewidert, stirnrunzelnd, glücklich, neutral, traurig und überrascht. Wie anhand Tabelle 4.2 deutlich zu erkennen ist, sind die vorhandenen Ausdrucksübergänge

Tabelle 4.2.: Ausdrucksübergänge im Datensatz

Ziel \ Start	angewidert	stirnrunzelnd	glücklich	neutral	traurig	überrascht	total
angewidert			5			5	10
stirnrunzelnd		5	5				10
glücklich	5	5		5	5	5	25
neutral		5	5			5	15
traurig			5			5	10
überrascht	5		5	5			15
total	10	10	25	15	5	20	85

²<https://www.vicon.com/software/shogun/>

4. Anforderungen

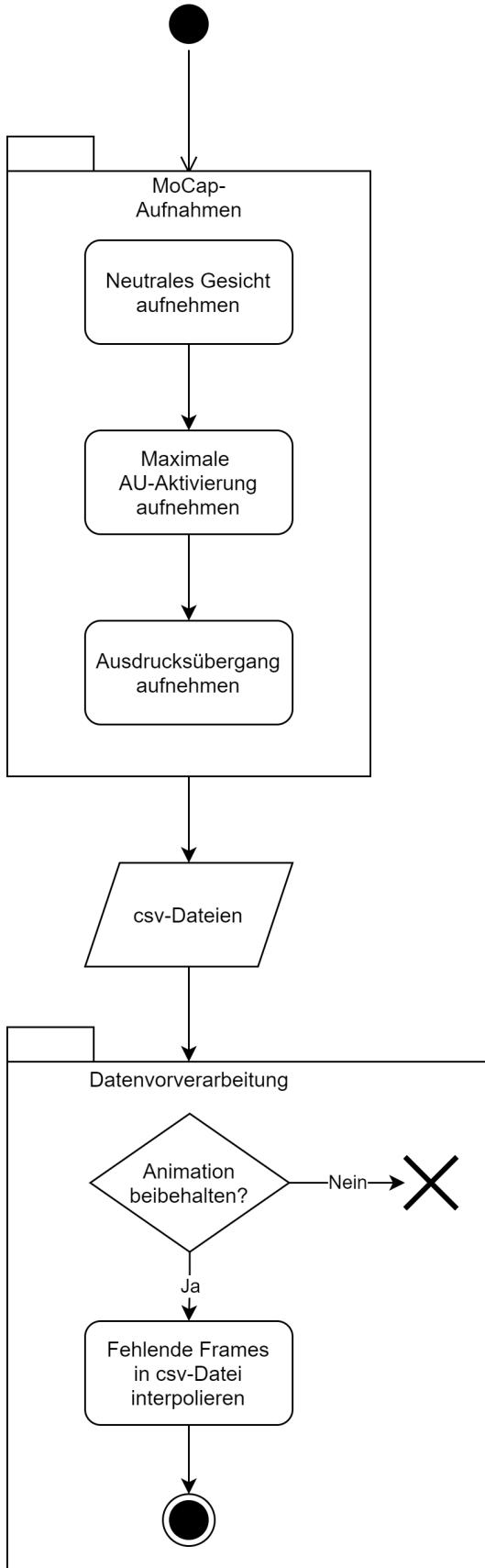


Abbildung 4.4.: Ablaufdiagramm für die Gewinnung von Ausdrucksdaten

fünf Mal im Datensatz vorhanden. Diese weiteren Aufnahmen unterscheiden sich dabei im Detail von den Selbigen ihrer Art, sodass eine gewisse Varianz innerhalb eines Ausdrucksübergangs vorhanden ist. Die Dauer der Übergänge reicht dabei von 41 bis hin zu 402 Frames, wobei die durchschnittliche Anzahl an Frames bei ca. 250 und somit bei zwei Sekunden liegt.

In Abbildung 4.5 sind die verwendeten Ausdrücke anhand des Avatars in jeweils einer der fünf Variationen dargestellt.

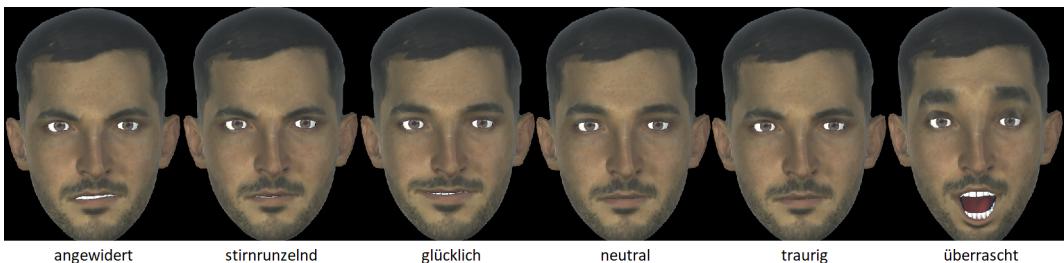


Abbildung 4.5.: Visualisierung der in dieser Arbeit verwendeten Ausdrücke mithilfe eines Avatars

4.4. Das System

An das Modell und die damit generierten Daten ergeben sich folgende funktionale sowie qualitative Anforderungen.

1. Generierung von neuen Animationen:

Das Modell muss in der Lage sein, Ausdrucksübergangsdaten erstellen zu können, welche der Struktur der Trainingsdaten gleichen (Frame, AU1, AU2, ...). Bei den erstellten Daten darf es sich um keine exakten Kopien von bereits vorhandenen Daten handeln. Die Animationen inklusive der notwendigen Daten dieser Animationen, welche diese Voraussetzungen erfüllen, werden fortlaufend als *neue Animationen* oder *neue Animationsdaten* bezeichnet. Wie in Kapitel 3 aufgeführt können hierfür mehrere Ansätze ausgewählt werden. Es eignen sich sowohl die generative Modelle VAEs sowie GANs aus dem Bereich der Gesichtsanimationen, sowie LSTM-Autoencoder-Architekturen, wie sie im Bereich der Bewegungsdatensynthese von Körpern zum Einsatz kommen, um neue Animationen für diese generieren zu können. Aufgrund der Tatsache, dass sich Architekturen wie LSTM-Autoencoder um generative Elemente wie GAN oder VAE erweitern lassen, wird in dieser Arbeit

ein LSTM-Autoencoder-System entwickelt. Wie in Kapitel 3 beschrieben, beschränken sich neue Animationen jedoch auf Sequenzen die den vorhandenen Trainingsdaten ähneln.

2. Keine Sequenzen für die Generierung von Sequenzen notwendig:

Das Modell muss in der Lage sein die neuen Animationen ohne das Vorhandensein von Sequenzen generieren zu können, d.h. es werden keine Ausdrucksübergangsdaten für das Generieren neuer Animationen benötigt. Diese funktionale Anforderung schließt den Einsatz klassischer sequence-to-sequence-Autoencoder, wie sie in Kapitel 2.1.3 beschrieben werden, aus. Diese Anforderung soll zudem gewährleisten, dass neue Animationsdaten ohne die Notwendigkeit eines vorherigen Aufnahmeprozesses generiert werden können.

3. Kontrolle über neue Animationen:

Der Nutzer des Systems soll in der Lage sein, entscheiden zu können welche Ausdrucksübergänge generiert werden. In [11] können Sequenzen auf Grundlage der Startsequenz generiert werden, d.h. sollte die Startsequenz *gehen* sein, generiert das System auch *gehen*. Für den Fall von Ausdrucksübergängen können aus einer Startsequenz jedoch unterschiedliche Übergänge stattfinden, sodass eine Möglichkeit zur Kontrolle implementiert werden muss.

4. Simple Bedienbarkeit des Systems:

Die Generierung neuer Animationen soll ohne eine zeitaufwändige, technische Einarbeitung möglich sein, bspw. durch das Ausführen einer einzelnen Funktion. Eine grafische Oberfläche wird hierzu jedoch nicht zwingend benötigt.

Kapitel 5

Umsetzung

In diesem Kapitel wird zunächst die Datenvorverarbeitung ausführlich beschrieben, bevor die Entwicklung des KNN-Systems detailliert aufgeführt wird. Hierbei wird das Gesamtsystem zunächst in seiner Gänze visuell dargestellt. Anschließend werden sämtliche Komponenten des Gesamtsystems schrittweise erläutert, wobei gleichzeitig die Besonderheiten gegenüber dem Stand der Technik (vgl: Kapitel 3) hervorgehoben werden. Im Nachhinein wird der Trainingsprozess inklusive aller Hyperparameter beschrieben, ehe die Generierung von neuen Animationsdaten mit Hilfe des KNN-Systems dargelegt wird, bevor abschließend die Durchführung einer numerischen Analyse des KNN-Systems anhand unterschiedlicher Datensätze definiert wird. Letzteres beinhaltet die Erklärungen für die Auswahl der unterschiedlichen Datensätze sowie eine Aufführung aller trainierten Modellvariationen.

5.1. Datenvorverarbeitung

Die Datenvorverarbeitung ist ein fundamentaler Bestandteil beim Entwickeln von KNN-Systemen, da das KNN auf Basis jener Dateien die gewünschten Informationen oder Strukturen lernen soll. Um somit zu gewährleisten, dass das KNN plausibel Animationsdaten generieren kann, muss sichergestellt werden, dass die Trainingsdaten ebenfalls plausible sind. Dies wird durch den ersten Schritt der Datenvorverarbeitung (vgl. Abschnitt 4.2 Abbildung 4.4) *Animationen beibehalten* realisiert. Dabei verläuft dieser Schritt im Rahmen dieser Arbeit von Hand, d.h. jede der aufgenommenen Ausdrucksübergänge wird durch die Animation des Avatars aus [15] visuell auf ungewöhnliche Artefakte überprüft. Bei diesen ungewöhnlichen

Artefakten handelt es sich vorwiegend um falsche Blendshapeintensitäten, welche an ungewollten Bewegungen des Avatars zu erkennen sind. Bei dieser Überprüfung werden die Bewegungen und Veränderungen des Avatars direkt mit den Originalaufnahmen verglichen. Gleichzeitig kann durch diesen Vergleich zudem die Essenz der Animationen, also die konkrete Bedeutung der Ausdrücke, überprüft werden. Für den Fall, dass Mängel vorhanden sind, werden die Animationsdaten verworfen. Animationen, bei denen visuelle Auffälligkeiten festgestellt werden konnten, werden fortan als *auffällige Animationen* bezeichnet.

Der nächste Schritt der Datenvorverarbeitung ist das Ausfüllen fehlender Frames innerhalb der csv-Dateien mittels Interpolation (vgl. Abbildung 4.4). Die Notwendigkeit dieses Schrittes ergibt sich aus dem in [15] entwickelten Datenübertragungsprozesses von Vicon zum Python-Programm, bei dem durch die Geschwindigkeit der Datenübertragung nicht jeder Frame verarbeitet werden kann. Um die Anzahl der entstehenden Lücken möglichst gering zu halten, werden daher zunächst die fehlerfreien Animationen mehrfach durch den Prototypen verarbeitet, um zu gewährleisten, dass keine zwei aufeinanderfolgenden Frames ausgelassen werden. Experimentell wird hierbei herausgefunden, dass drei separate Aufnahmen diese Bedingung erfüllen. Diese drei Aufnahmen werden zunächst mittels eines separaten Python-Skriptes *Erstellung_vorverarbeiteter_Datensatz* (vgl. Abbildung 5.1 (a)) vorverarbeitet. Durch dieses Skript werden die einzelnen Daten zunächst nach den Frames sortiert, bevor anschließend Duplikate entfernt werden. Letzteres ist notwendig, da durch das Nichtvorhandensein eines Musters für das Auslassen von Frames einzelne Informationen, also der Frame sowie die AU-Intensitäten, öfter in der Datei vorhanden sein können. Anschließend werden die fehlenden Daten mittels eines weiteren Python-Skripts *Interpolation_fehlender_Daten* (vgl. Abbildung 5.1 (b)) linear interpoliert. Mittels linearer Interpolation können fehlende Datenpunkte $f(x)$ zum Zeitpunkt bzw. an der Stelle x zwischen zwei vorhandenen Datenpunkten f_0 sowie f_1 mittels der allgemeinen Geradengleichung (vgl. Formel 5.1) berechnet werden:

$$f(x) = f_0 + \frac{f_1 - f_0}{x_1 - x_0}(x - x_0) \quad (5.1)$$

Um den Datensatz künstlich zu erweitern werden zudem höhergradige Interpolationsverfahren zur Erstellung weiterer Animationsdatenvarianten verwendet. Zum einen handelt es sich hierbei um *quadratische Interpolation* sowie *kubische Inter-*

polation, welche sich beide durch die Formel 5.2 von Lagrange für $n = 2$ bzw. $n = 3$ beschreiben lassen.

$$p(x) = \sum_{i=0}^n f_i \prod_{k \neq i}^n \frac{x - x_k}{x_i - x_k}, i = 0, \dots, n \quad (5.2)$$

Das so für die Berechnung entstehende Polynom $p(x)$ benötigt jedoch mindestens $n + 1$ Stützpunkte (x_i, f_i) , n gibt den Grad des Polynoms an und das Datenpaar $(x, p(x))$ beschreibt die Stelle sowie den entsprechenden Datenpunkt an dieser Stelle.

Durch die soeben genannte Erweiterung besteht der Trainingsdatensatz für das entwickelte KNN-System letztendlich aus 225 verschiedenen Daten. Aufgrund der Tatsache, dass die Python-Skripte separat vom eigentlichen Modell funktionieren, kann dieser Schritt der Datenvorverarbeitung bei Behebung des Übertragungsfehlers zukünftig ausgelassen werden. Ebenso kann durch die Aufteilung beider Vorverarbeitungsskripte die künstliche Erweiterung des Datensatzes für weitere Experimente ausgelassen werden.

Eine weitere Vorverarbeitung der Daten, wie beispielsweise die Anpassung der Bilderanzahl auf einen festen Wert, sind für das im Folgenden beschriebene KNN-System nicht notwendig, da das System Datensätze mit variablen Datenlängen verarbeiten kann.

In Abbildung 5.1 sind beide dieser Vorverarbeitungsschritte visualisiert. Abbildung 5.1 (a) beschreibt dabei die Verarbeitung der Mehrfachaufnahmen zur Minimierung von fehlenden Daten, wohingegen Abbildung 5.1 (b) die Berechnung von fehlenden Daten aufführt.

Der vollständige Code der Datenvorverarbeitung kann Anhang A entnommen werden, wobei die einzelnen Funktionen anhand der gleichnamigen Skripte *Erstellung_vorverarbeiteter_Datensatz* und *Interpolation_fehlender_Daten* identifiziert werden können.

5. Umsetzung

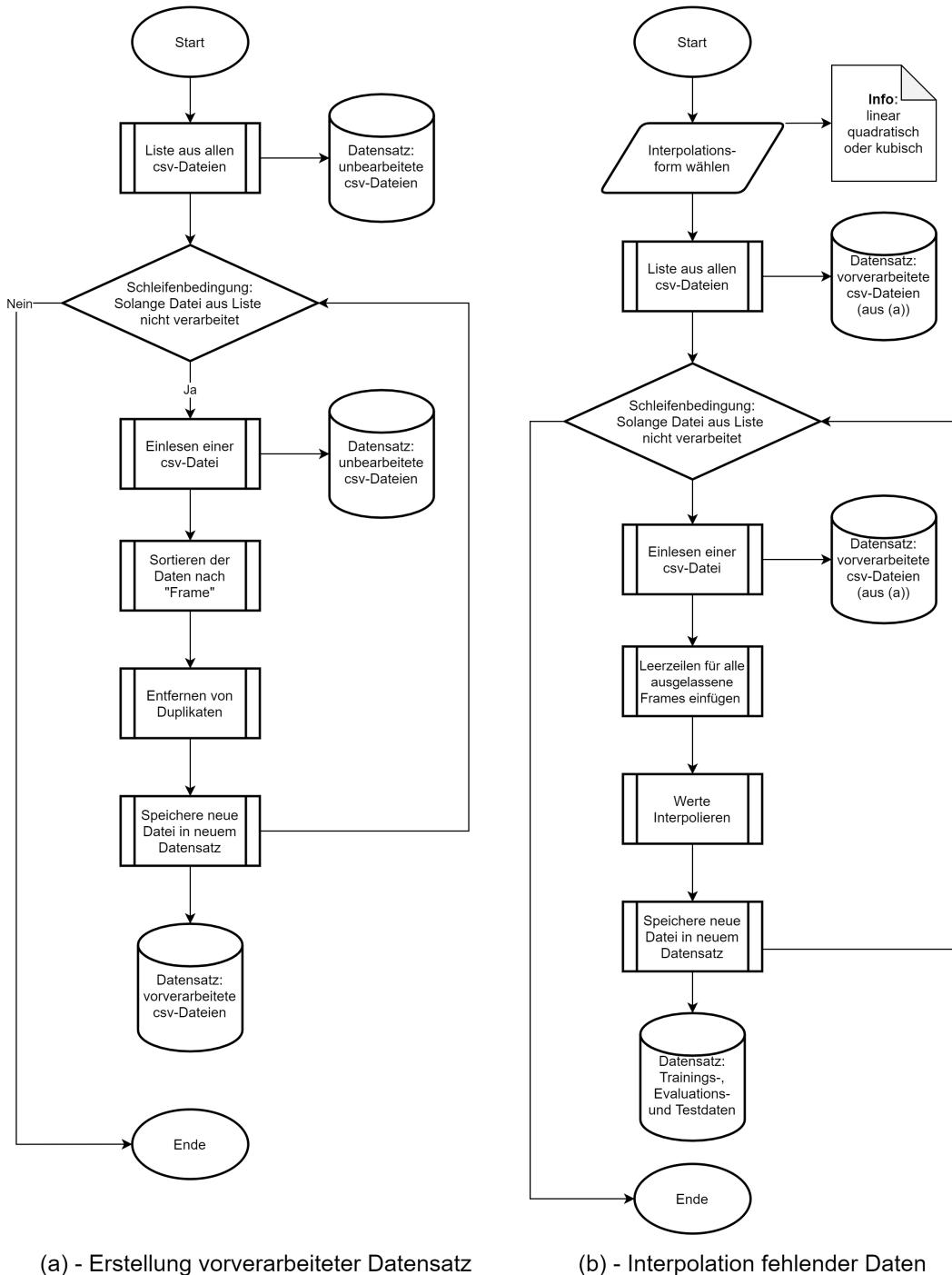


Abbildung 5.1.: Ablaufdiagramm für die Erstellung vorverarbeiteter Daten sowie Interpolation fehlender Daten; links **(a)** Erstellung vorverarbeiteter Daten, rechts **(b)** Interpolation fehlender Daten

5.2. Entwicklung eines KNN-Systems

Um das Ziel der Generierung von neuen Animationen zu erreichen, wird auf Basis der in Abschnitt 2.1 sowie Abschnitt 2.2.2 erläuterten Grundlagen ein KNN-Modell entworfen und mithilfe von Pytorch¹ implementiert. Pytorch ist ein von Facebook² entwickeltes, pythonbasiertes *open source machine learning Framework*, welches sich aufgrund der Bereitstellung von zahlreichen KNN-Strukturen (bspw. LSTMs), Fehlerberechnungsfunktionen sowie verschiedenen Trainingsoptimierern für die Programmierung eines solches Systems eignet. Die Anforderungen aus Kapitel 4 sollen dabei bestmöglich erfüllt werden. Um die Anforderungen *Generierung von neuen Animationen* sowie *Keine Sequenz für die Generierung von Sequenzen notwendig* zu erfüllen, wird ein LSTM-Autoencoder, wie in [11] beschrieben, als Grundlage für das zu entwickelnde System verwendet und auf die Aufgabenstellung sowie eigene Datenstruktur angepasst. Dabei wird maßgeblich die Inkludierung eines Target-Vektors, wie in [13] beschrieben, zur Anforderung Kontrolle über die neuen Animationen adaptiert.

Das entwickelte LSTM-Autoencoder-Modell mit dem Namen *Expression-Generator* ist in Abbildung 5.2 schematisch dargestellt und besteht aus drei wesentlichen Komponenten: Einem *LSTM-Encoder*, einem *Recurrent-Generator* sowie einem *Linearen-Decoder*. Das generelle Ziel des Modells ist die Vorhersage eines Frames x_P , welcher auf den letzten Frame des *Vergangenheitssequenz*-Vektors x folgt. Dabei wird diese Vorhersage ausschlaggebend von einem konstanten *Ziel-Frame* x_Z beeinflusst. Zu jedem Zeitpunkt $t < k - 20$, wobei k die Dauer der zu generierenden Sequenz angibt, generiert das Modell somit eine Vorhersage, welche im Anschluss rekursiv für die Vorhersage eines weiteren Frames verwendet wird. Daraus folgt, dass das System nach exakt 20 Frames lediglich seine eigenen Vorhersagen sowie die statische Variable x_Z für die Vorhersage weiterer Frames verwendet. Das Ziel des Modells ist somit die Approximation einer Funktion f , welche den nächsten Frame eines vordefinierten Ausdrucksübergangs vorhersagt. Die genauen Funktionen der einzelnen Komponenten sowie ein detaillierterer Ablauf werden im Folgenden in den gleichnamigen Abschnitten weiter erläutert. Darüber hinaus wird ebenfalls auf den Ziel-Frame sowie den Vergangenheitssequenz-Vektor eingegangen. Besonderheiten gegenüber den Ansätzen aus [11] sowie [13] werden dabei innerhalb der Abschnitte hervorgehoben.

¹<https://pytorch.org/>

²<https://www.facebook.com/>

5. Umsetzung

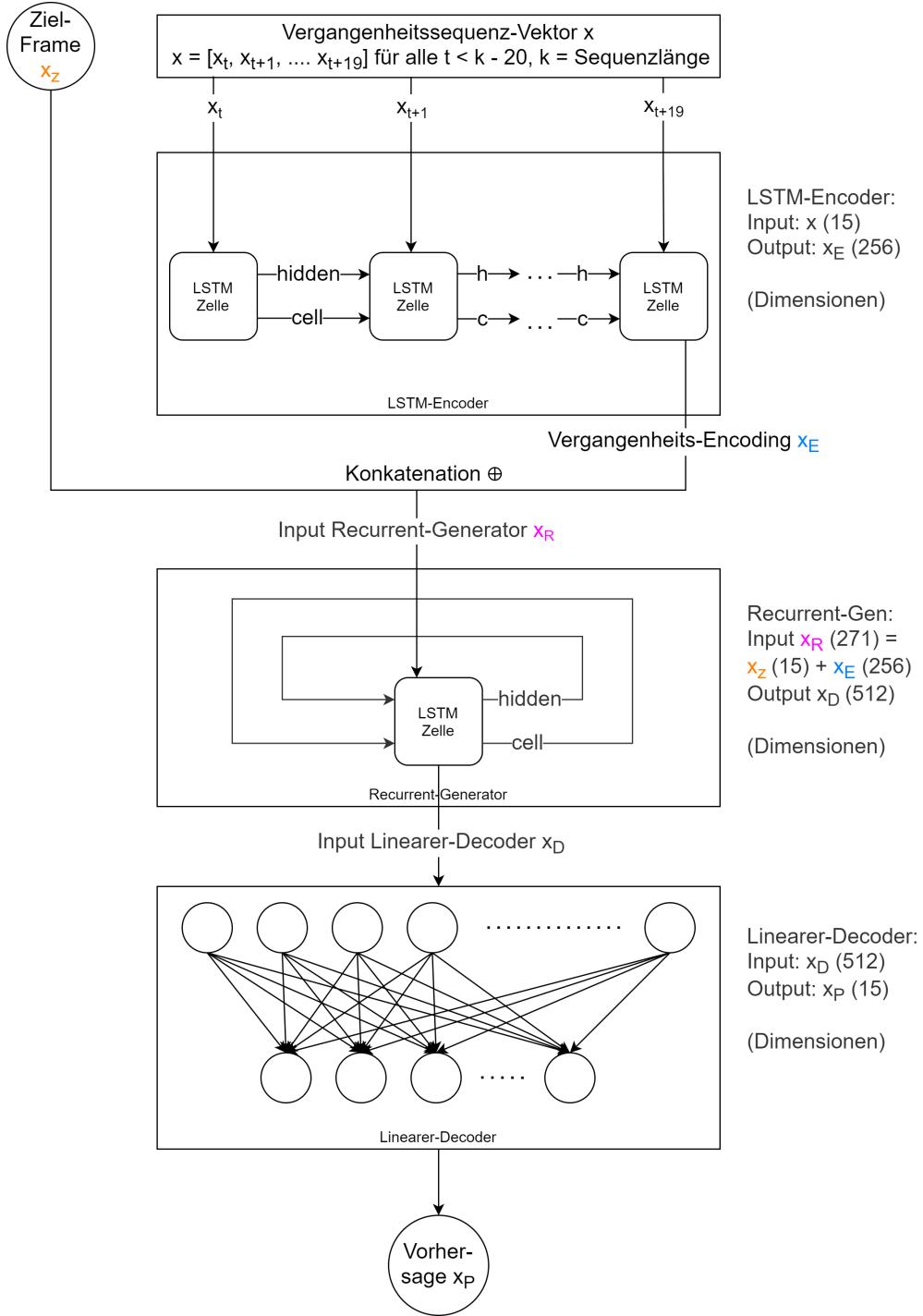


Abbildung 5.2.: Vollständige Darstellung des entwickelten KNN-Systems *Expression-Generator*

An dieser Stelle sei zudem angemerkt, dass während der Entwicklung des Modells mehrere unterschiedliche Architekturen implementiert, trainiert und visuell evaluiert wurden. Eine quantitative Evaluation wurde aufgrund offensichtlicher Fehler innerhalb der Animationen jedoch nicht durchgeführt, da diese Resultate keineswegs

Tabelle 5.1.: In dieser Arbeit verwendete Python-Pakete

Paket	Version
Python	3.7.9
Pytorch	1.7.0
Cudatoolkit	10.1.243
pandas	1.1.3
numpy	1.19.2

dem gewünschten Standard entsprachen. Solche Animationen wechselten beispielsweise in nur einem Frame von einem Ausdruck zum anderen, sodass kein Übergang vorhanden war. In anderen Fällen wurden zudem die gewünschten Start- und Ziel-frame Parameter der Animationen schlichtweg ignoriert und das Netz generierte etwas völlig anderes. Die Ergebnisse dieser Modelle trugen dennoch maßgeblich zur grundlegenden Struktur des letztendlich verwendeten Modells bei, insbesondere die Verwendung eines Vergangenheitssequenz-Vektors in Kombination mit der LSTM-Encoding-Schicht sind hiervon betroffen. Auf die konkreten Auswirkungen der einzelnen Module soll dabei im Kapitel 7 näher eingegangen werden. Eine genauere Beschreibung jener Modelle ist für das Verständnis dieser Arbeit jedoch nicht notwendig. Die für diese Arbeit verwendeten Python-Pakete sind in Tabelle 5.1 zu finden, welche zur fehlerfreien Verwendung des Codes für den Expression-Generators sowie die Datenvorverarbeitung installiert sein müssen.

Der Code für das vollständige KNN-System ist in Anhang A hinterlegt und kann am Namen *Expression_Generator* erkannt werden.

5.2.1. Vergangenheitssequenz-Vektor x

Der Vergangenheitssequenz-Vektor x beschreibt den wesentlichen Bestandteil der Informationen zur Generierung der Vorhersage. Inhalt von x sind zu jedem Zeitpunkt t exakt 20 aufeinanderfolgende Kombinationen sämtlicher verwendeter AU-Intensitäten einer Sequenz. Die Informationen werden dabei in einem Tensor gespeichert. Nach jeder Iteration der Generierung und demnach nach jeder Vorhersage durch das Modell, wird die Vorhersage x_P an die letzte Stelle von x angehängt, wobei gleichzeitig der erste Wert von x entfernt wird. Dieser Vorgang ist in Abbildung 5.3 am Beispiel der ersten Iteration visualisiert. In [11] als auch [13] werden ausschließlich einzelne Vektoren der Vergangenheitssequenz als Input an die Encoding-Schicht übergeben. Darüber hinaus beschreibt die Vorhersage in dieser

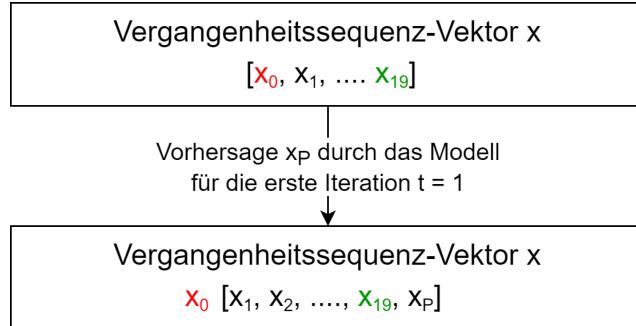


Abbildung 5.3.: Schematische Darstellung des Vergangenheitssequenz-Vektors x am Beispiel der ersten Iteration

Arbeit die absoluten AU-Intensitäten des vorhergesagten Frames, wohingegen in der Literatur lediglich die Distanz bzw. Veränderung vorhergesagt wird [11], [13].

5.2.2. Ziel-Frame x_Z

Die Inkludierung des Ziel-Frames x_Z stellt eine weitere signifikante Abweichung gegenüber der Architektur aus [11] dar. Die Adaptation eines Ziel-Frames basiert auf der in [13] vorgestellten Architektur des Übergangsnetzwerks (vgl. Kapitel 3). x_Z stellt somit einen zusätzlichen Bestandteil der Informationen zur Generierung der Vorhersagen durch das Modell dar. In Abbildung 5.2 ist zudem deutlich zu erkennen, dass x_Z hierfür mit dem Vergangenheits-Encoding x_E konkateniert wird, um so den Input für den Recurrent-Generator x_R zu bilden. Dabei gilt, dass x_Z innerhalb einer Sequenz konstant ist. Für das Format von x_Z gilt, dass dieses dem Format eines einzelnen Tupels von x gleichen muss, d.h. die Reihenfolge der AU-Intensitäten muss für beide Eingänge übereinstimmen. Im direkten Vergleich zu [13] wird zudem deutlich, dass x_Z nicht zunächst durch eine Encoding-Layer vorverarbeitet wird.

5.2.3. LSTM-Encoder

Die Encoding-Schicht des Modells wird durch einen LSTM-Encoder realisiert und nicht wie in [11] und [13] durch einen Multi-Layer-Perceptron (MLP). Diese Abänderung soll zum einen den in [13] beschrieben *Offset-Vektor* (Distanz aus Ziel-Frame und aktueller Frame) ersetzen und zum anderen temporale Zusammenhänge aus den vorherigen Frames extrahieren. Der LSTM-Encoder ist durch ein many-

to-ones-LSTM realisiert (vgl. 2.1.1 many-to-one), bei welchem der letzte Hidden-Output sämtliche Informationen der Sequenz codiert. Zusätzlich folgt aus der many-to-one-Architektur, dass der Input x per-Frame vom LSTM-Encoder eingelesen und verarbeitet wird, wodurch gilt, dass Hidden- und Cell-State kontinuierlich innerhalb der ganzen Sequenz modifiziert werden. Daraus ergibt sich zudem, dass Hidden- und Cell-States nur zu Beginn einer neuen Sequenz manuell verändert werden.

Die Encoding-Schicht des Systems codiert x in das Vergangenheits-Encoding x_E . Die Anzahl an Dimensionen, also die Anzahl an Hidden-Neuronen im LSTM-Encoder, beträgt 256, d.h. aus den ursprünglichen 15 Dimensionen eines Frames (Anzahl an AU-Intensitäten pro Frame) werden nach dem Encoding die Informationen der letzten 20 Frames in 256 Dimensionen gespeichert. Die Berechnung von x_E ist in Formel 5.3 aufgeführt, wobei angemerkt werden muss, dass sich die Berechnungen von c_{t+19} aus Formel 2.5 ergibt. Folglich gilt zudem die Notwendigkeit der Berechnung von c_{t+18} , was rekursiv wiederum bis hin zur Berechnung von $c_{t-1=0}$ sowie $h_{t-1=0}$ führt, wobei wie zuvor erwähnt $h_0 = 0$ sowie $c_0 = 0$ gilt, \odot steht dabei für das elementweise Produkt:

$$x_E = e(x) = h_{t+19}^E = o_t \odot \tanh(c_{t+19}) \quad (5.3)$$

h_E bezeichnet die Berechnungen der Hidden-States innerhalb des Encoder-LSTMs. Innerhalb der Encoding-Schicht findet somit eine erste temporale Extrahierung von Informationen der Sequenz statt. Die weitaus größere Dimension von x_E resultiert daraus, dass x_Z lediglich die Richtung der Animation vorgeben bzw. beeinflussten soll, wohingegen x_E neben der Extrahierung von temporalen Zusammenhängen der Vergangenheit auch zur Stabilität des Gesamtsystems beitragen soll.

5.2.4. Recurrent-Generator

Der Recurrent-Generator ist der Hauptbestandteil des Modells zur Generierung von neuen Animationen und besteht ebenso wie die Encoding-Layer aus einem LSTM-Modul. Dabei unterscheidet sich jedoch die genaue Architektur des LSTMs. Beim Recurrent-Generator handelt es sich im Kern um ein many-to-many-LSTM, bei welchem pro Zeitschritt t der Input x_R (Konkatenation von x_E und x_Z) eingelesen und verarbeitet wird, wobei Hidden- und Cell-State über die ganze Sequenz weiterverarbeitet werden. Der Recurrent-Generator ist demnach für die Modellie-

rung von temporalen Dynamiken der Sequenz zuständig. Im direkten Vergleich mit [11] besteht der Recurrent-Generator des entwickelten Expression-Generators aus nur einer LSTM-Schicht wie in [13].

Aufgrund der einzelnen Berechnung des Outputs x_D pro Zeiteinheit t , welcher den Input des Linearen-Decoders darstellt, kann x_D , analog zur Berechnung von x_E mithilfe von Formel 5.4 berechnet werden. Dabei gilt ebenfalls, dass die Werte der vorherigen Hidden- und Cell-States bekannt sein müssen:

$$x_D = r(x_R) = r(x_Z \oplus x_E) = h_t^R = o_t \odot \tanh(c_t) \quad (5.4)$$

Der \oplus -Operator steht für die Konkatenation zweier Tensoren, wohingegen h_R die Berechnung des Hidden-States innerhalb des Recurrent-Generators darstellt.

Analog zur Encoding-Layer werden auch hier die ersten Hidden- und Cell-States einer Sequenz *nullinitialisiert*. Durch die Konkatenation von x_E und x_Z beträgt die Dimension des Eingangs somit 271, welche durch die Berechnungen innerhalb des Recurrent-Generators auf 512 erweitert werden. Die Anzahl der Hidden-Units entspricht der in [13] verwendeten Anzahl. Darüber hinaus wird auch der Recurrent-Generator vom Expression-Generator durch einen Ziel-Frame beeinflusst, jedoch mit dem Unterschied, dass in [13] ein modifiziertes LSTM-Modul, welches einen zusätzlichen Parameter für die Konditionierung einfügt, verwendet wird. Daraus folgt außerdem, dass eine Zusammenführung von Encoding und Ziel-Frame, wie sie in dieser Arbeit durchgeführt wird, gänzlich ausbleibt. Diese Anpassung der grundlegenden Systematik resultiert aus einem Fehler innerhalb des verwendeten Pytorch-Frameworks, da bei Verwendung einer benutzerdefinierten LSTM-Zelle das Modell nicht fehlerfrei unter der Verwendung einer Grafikkarte trainiert werden kann, wodurch sich die Trainingszeit drastisch erhöht.

5.2.5. Linearer-Decoder

Jeder Output x_D des Recurrent-Generators wird im letzten Schritt der Verarbeitung durch den Linearen-Decoder des Systems verarbeitet. Der Lineare-Decoder ist mittels einer *fully-connected* linearen Schicht von 512 Neuronen auf 15 Neuronen realisiert. Fully-connected bedeutet dabei, dass sämtliche Neuronen einer Schicht mit allen Neuronen der darauffolgenden Schicht verbunden sind. Diese Verbindun-

gen sind in Abbildung 5.2 im Bestandteil *Linearer Decoder* anhand der einzelnen Pfeile, welche diese Verbindungen visualisieren, deutlich erkennbar. Der Lineare-Decoder ist dabei für die Transformierung des Outputs des Recurrent-Generators in das ursprüngliche Formal, bestehend aus 15 Dimensionen, zuständig.

Der Output x_P des Linearen-Decoders lässt sich dabei anhand Formel 5.5 unter Verwendung von x_D berechnen:

$$x_P = d(x_D) = Wx_D + b \quad (5.5)$$

Wobei W für die Gewichtungen und b für die Verzerrungen der einzelnen Neuronen stehen. Das Ergebnis x_P beschreibt zum Zeitpunkt t die AU-Intensitäten jeder einzelnen AU für den darauffolgenden Frame $t + 1$, ohne dass weitere Berechnungen durchgeführt werden müssen. Wie in Abschnitt 5.2.1 bereits erwähnt wird die Vorhersage x_P somit für die weitere Berechnung der Sequenz verwendet. Die vollständige Berechnung x_P ergibt sich somit durch die Kombination der Formeln 5.3, 5.4 und 5.5:

$$x_P = d(r(x_Z \oplus e(x))) \quad (5.6)$$

Die Funktionen d , r und e stehen hierbei für die Berechnungen des Linearen-Decoders, des Recurrent-Generators sowie des Encoding-LSTMs.

Die durch den Expression-Generator berechnete Vorhersage unterscheidet sich wie zuvor erwähnt in der Darstellung. In [11] und [13] werden lediglich die Differenzen beider Frames für die Berechnung der Vorhersage verwendet, weshalb in beiden Arbeiten eine zusätzliche Berechnung der absoluten Werte benötigt wird. Gleichermassen unterscheiden sich die Decoder-Module auch im Aufbau signifikant voneinander. In den beiden verglichenen Systemen werden drei fully-connected-Schichten zu einem MLP zusammengefasst, bei welchen die Neuronen durch die Aktivierungsfunktion *Leaky Rectified Linear Unit* aktiviert werden.

5.3. Training des KNN-Systems

Nachdem das vollständige System des Expression-Generators mithilfe des Pytorch-Frameworks implementiert wurde, ist der nächste Schritt die Entwicklung eines

passenden Trainingsalgorithmus, damit das Netz die gewünschte Funktionalität erfüllen kann. Zur Entwicklung eines solchen Algorithmus müssen zwei wesentliche Faktoren berücksichtigt werden: (1) Die Auswahl einer geeigneten *Fehlerfunktion* sowie (2) die Auswahl eines *Optimierers* bzw. eines *Optimieralgorithmuses*. Abschlossen wird diese Programmierung einer vollständigen Trainingsfunktion durch die Anpassung der Gradienten durch den Optimierer. Im Folgenden wird auf die einzelnen Schritte genauer eingegangen.

5.3.1. Fehlerfunktion

Die Fehlerfunktion eines KNN-Systems ist ein elementarer Bestandteil um zu gewährleisten, dass das System die gewünschte Funktionalität umsetzt und dementsprechend lernt. Die Definition der Fehlerfunktion stellt dabei einen direkten Zusammenhang zwischen Eingang und Ausgang des Systems her. Dieser Zusammenhang bildet somit die Grundlage des Lernens. Aufgrund der Tatsache, dass der Expression-Generator versucht eine korrekte Vorhersage zu treffen, eignet sich die *mittlere quadratische Abweichung* (englisch: Mean Squared Error (MSE)) zwischen dem tatsächlichen Frame, im Folgenden durch x_{GT} für Grundwahrheit (englisch: Groundtruth (GT)) beschrieben, und der Vorhersage durch das Netzwerk x_P . Konkret handelt es sich damit um ein *Regressionsproblem*. In vergleichbaren Arbeiten wie [11], [13] und [26] wird daher ebenfalls der MSE für die Berechnung des Vorhersagefehlers verwendet. Dabei gilt während des Trainings, dass dieser Fehler für die ganze generierte Sequenz $X_{GT} = \{x_{GT_1}, \dots, x_{GT_d}\}$ und $X_P = \{x_{P_1}, \dots, x_{P_d}\}$ und nicht per-Frame berechnet wird. Der vollständige Fehler lässt sich daher folgendermaßen definieren:

$$MSE(X_{GT}, X_P) = \frac{1}{d} \sum_{i=1}^d (x_{GT_i} - x_{P_i})^2 \quad (5.7)$$

Wobei d die Dauer der Sequenz beschreibt und i den Zeitindex darstellt. Abschließend bleibt anzumerken, dass ein einzelner Fehler die Gesamtheit der Fehler aller AU-Intensitäten enthält und nicht für jede AU-Intensität einzeln berechnet wird.

5.3.2. Optimierer

Der in Formel 5.7 beschriebene Fehler kann nun verwendet werden, um mithilfe des Optimierers das KNN-System zu trainieren, indem berechnet wird welche Parameter des Netzwerks den Fehler in wie fern beeinflussen. Die Minimierung des Fehlers durch die Anpassung der Parameter wird dabei als Optimierung bezeichnet. Hierfür werden zunächst wie in Kapitel 2.1 die Gradienten jedes einzelnen Parameters berechnet. Das Ziel der Optimierung ist das Erreichen des bestmöglichen Minimums des Graphen, welcher die Funktion des KNNs beschreibt [19], indem die Parameter abhängig von den Gradienten durch den Optimierer um einen vordefinierten Wert verändert werden. Bei diesem Wert handelt es sich um die sogenannte *Lernrate*, welche einen essenziellen Bestandteil der meisten Optimierer darstellt und zudem als *Hyperparameter* bezeichnet wird.

Für das Trainieren des Expression-Generators wurde der Optimierer *Adam*, was für *adaptive moment estimation* steht, von Kingma und Ba [37] verwendet, wobei die Standardwerte für β_1 und β_2 beibehalten wurden. Die Lernrate wurde mit 0,01 initialisiert und jede 100 Epochen um den Faktor 0,1 verringert. Trainiert wurde das System für 400 Epochen unter der Verwendung eines *Validierungssets*. Nach jeweils 50 Epochen wurden die Gewichte der Epoche mit dem geringsten Fehlerwert bezogen auf das Validierungsset der letzten 50 Epochen ausgewählt, woraufhin das Modell mit diesen Gewichten weiter trainiert wurde. Hierfür wurde der Datensatz in *Trainingssatz*, *Validierungssatz* sowie *Testsatz* in einem Verhältnis von 80 % zu 10 % und 10 % aufgeteilt. Der Testsatz wird zum Schluss des Trainings dazu verwendet, um die Performance des Systems mit gänzlich unbekannten Sequenzen zu überprüfen.

Das Pytorch-Framework stellt hierfür sämtliche Funktionalität zur Verfügung, sodass schlussendlich die eigentliche Schwierigkeit in der Erstellung der Trainingsfunktion liegt. Die gesamte Trainingsfunktion ist in Abbildung 5.4 vereinfacht visualisiert.

Eine Epoche lässt sich dabei als ein Durchlauf aller Trainingsdaten beschreiben. Über diese Trainingsdaten wird folglich für jede Sequenz eine sequenzielle, rekursive Vorhersage durch das Netz getätigt und ebenfalls pro Sequenz mit der Grundwahrheit X_{GT} verglichen. Diesbezüglich wird für jede Sequenz ein Tensor (Container) X_P erstellt, in welchen per-Frame die Vorhersagen x_P gespeichert werden.

5. Umsetzung

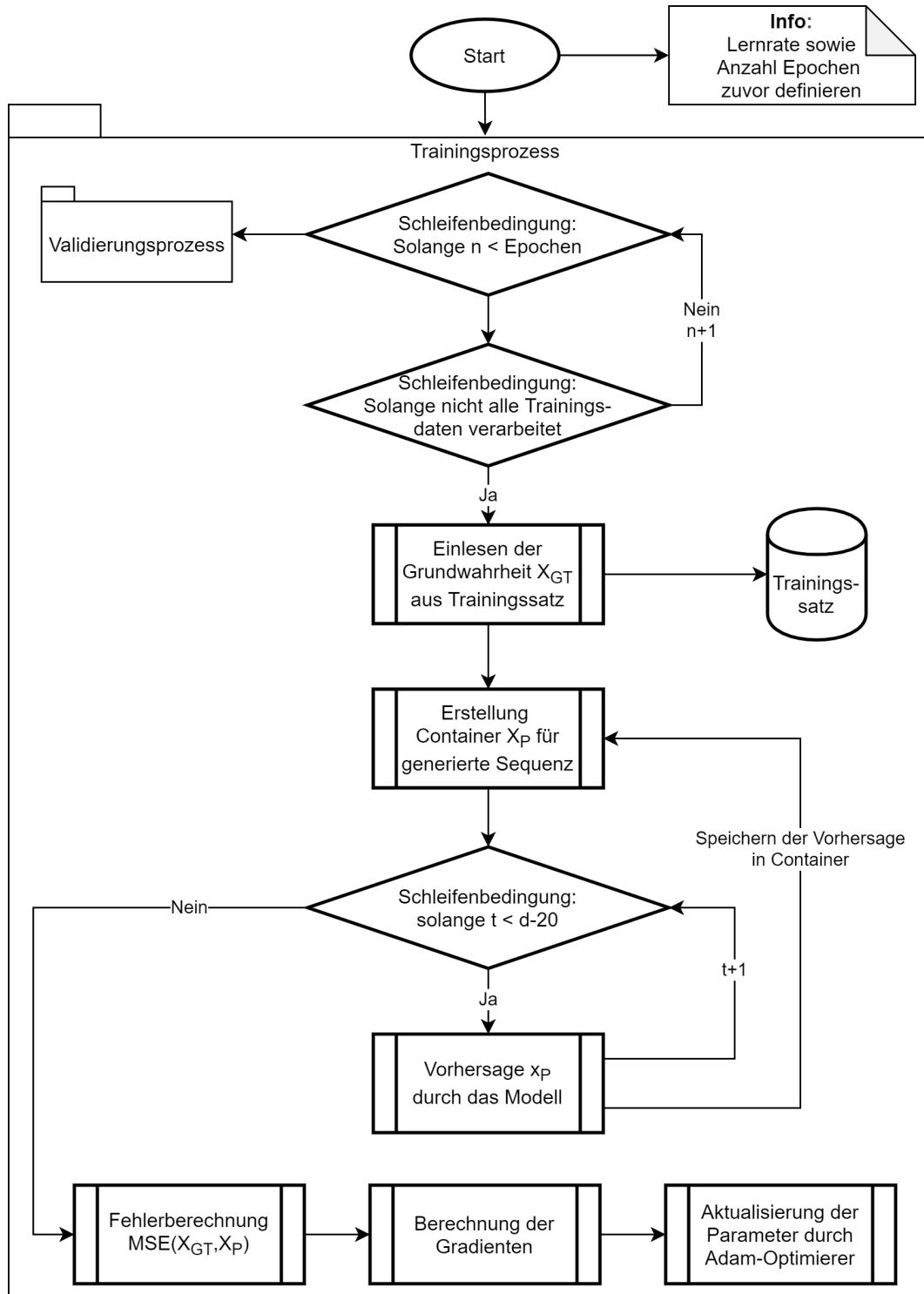


Abbildung 5.4.: Vereinfachtes Ablaufdiagramm für die Trainingsfunktion des Expression-Generators

Der Vorgang der Trainingsratenanpassung sowie der Vergleich mit dem Validierungsset (Validierungsprozess) sind der Übersichtlichkeit wegen nicht dargestellt.

Der vollständige Trainingsalgorithmus kann dem Anhang A innerhalb des *Expression_Generator*-Skripts unter der Funktion *train_model()* entnommen werden. Die Trainingsfunktion endet nach dem Schritt des Validierungsprozesses, woraufhin bspw. die manuelle Anpassung der Lernrate vorgenommen werden kann.

5.4. Generierung von neuen Animationen

Um mit Hilfe des Expression-Generators letztendlich neue Animationsdaten generieren zu können, wurde eine Funktion geschrieben, welche es dem Nutzer ermöglicht mithilfe eines Start- sowie eines Endframes neue Daten zu generieren. Der relevanteste Unterschied zum Trainingsablauf liegt dabei im Vergangenheitssequenz-Vektor x , welcher aufgrund der zweiten Anforderung keine Sequenz enthalten soll. Daraus folgt, dass das Format von x zur Laufzeit leicht abgeändert werden muss. Um eine simple Funktionalität zu gewährleisten wird der Startframe daher für die erste Iteration der Datengenerierung konstant gehalten, d.h. es wird zunächst ein Sequenzvektor der Länge 20 aus diesem Startframe erstellt. Daraus folgt jedoch, dass die Animation erst mit dem 20. Frame beginnt und nicht wie beim Training mit dem ersten. Der Endframe, welcher für die Funktion benötigt wird, gleicht dabei dem in Abschnitt 5.2.2 beschriebenen Ziel-Frame x_Z und definiert die zu erreichenden AU-Intensitäten der neuen Animation.

Damit die Daten den in [15] entwickelten Avatar animieren können, werden die generierten Daten zuletzt in einer csv-Datei gespeichert, welche das in Tabelle 4.1 aufgeführte Format besitzt. Das Format der neuen Animationen ist dabei stets an das Format der Trainingsdaten gebunden und damit gleichzeitig an die selbigen AUs.

5.5. Durchführung der Evaluation

Zur Klärung der Fragestellung wie die Daten überprüft werden können, wurde eine numerische Analyse der generierten Daten durchgeführt. Die numerische Analyse umfasst dabei den Vergleich des vorgestellten Systems unter Verwendung verschiedener Datensätze. Eine Auflistung der Zusammensetzung der verwendeten Daten-

5. Umsetzung

Tabelle 5.2.: Datensatzgrundlage für die Evaluation

Ziel Start	angewidert	stirnrunzelnd	glücklich	neutral	traurig	überrascht	total
angewidert						5	5
stirnrunzelnd			5				5
glücklich							
neutral		5				5	10
traurig					5		5
überrascht	5		5				10
total	5	5		10		15	35

sätze kann Tabelle 5.3 entnommen werden, wobei die Daten aus Tabelle 5.2 als Grundlage für sämtliche Datensätze dienen³.

Ausgangspunkt der Analyse stellen dabei die Ausdrucksübergänge der Animationen von und nach *glücklich* dar, da diese für jeden der anderen Gesichtsausdrücke vorhanden sind. Hierbei wurden drei verschiedene Ansätze verfolgt:

1. Symmetrischer Ansatz:

Beim symmetrischen Ansatz wurden stets *von-* und *nach-Übergänge* aller fünf Varianten der Ausdrucksübergänge verwendet. Hierfür wurden insgesamt fünf verschiedene Datensätze verwendet. Der Datensatz für das erste Modell wurde gänzlich ohne die Ausdrucksübergänge von und nach *glücklich* trainiert. Für das zweite Modell wurden die Übergänge der Kombinationen von und nach *glücklich* und *traurig* verwendet, für das dritte System wurden dann zusätzlich die Übergänge von und nach *glücklich* und *angewidert* verwendet, usw.

2. Asymmetrischer Ansatz:

Für den asymmetrischen Ansatz wurden sämtliche *von-Übergänge* für das Training des Systems verwendet. Die *nach-Übergänge* zu *glücklich* sind dem Modell folglich alle fremd.

3. Variationsansatz:

Bei dem Variationsansatz wurden alle verschiedenen vor- und nach-Übergänge von *glücklich* verwendet (*glücklich* und *traurig*, *glücklich* und *überrascht*, etc.), jedoch nicht mit allen Variationen. Das erste Variationsmodell (*g*) wurde mit nur einer Variante der Ausdrucksübergänge trainiert, wohingegen das zweite Variationsmodell (*h*) mit drei Varianten trainiert wurde.

³Die in Abschnitt 5.1 beschriebene künstliche Erweiterung des Datensatzes um den Faktor drei ist in Tabelle 5.2 nicht dargestellt.

5. Umsetzung

Tabelle 5.3.: Zusammensetzung der Datensätze für die numerische Evaluation

Modell	Enthaltene Übergänge	Zusätzliche Daten zu Tabelle 5.2	Gesamt
<i>a</i>	Keine von- und nach glücklich	Keine	35
<i>b</i>	<i>a</i> + Kombinationen aus traurig und glücklich	5 von- sowie 5 nach-Übergänge	45
<i>c</i>	<i>b</i> + Kombinationen aus angewidert und glücklich	5 von- sowie 5 nach-Übergänge	55
<i>d</i>	<i>c</i> + Kombinationen aus überrascht und glücklich	5 von- sowie 5 nach-Übergänge	65
<i>e</i>	<i>d</i> + Kombinationen aus neutral und glücklich	5 von- sowie 5 nach-Übergänge	75
<i>f</i>	<i>a</i> + Sämtliche von-Übergänge von glücklich	25 von-Übergänge	60
<i>g</i>	<i>a</i> + 1 Variation aller von- und nach-Übergänge von glücklich	5 von- sowie 5 nach-Übergänge	45
<i>h</i>	<i>a</i> + 3 Variationen aller von- und nach-Übergänge von glücklich	15 von- sowie 15 nach-Übergänge	65
<i>i</i>	Gesamter Datensatz	25 von- sowie 25 nach-Übergänge	85

Insgesamt wurden daher neun verschiedene Modelle zu jeweils 400 Epochen trainiert, wobei das neunte Modell das Standardmodell darstellt, welches mit sämtlichen Daten trainiert worden ist (*i* - gesamter Datensatz).

Für jedes symmetrische Modell *a* bis *e* wurden für die Evaluation exemplarisch zwei von sowie zwei nach-Ausdrucksübergänge in zwei Variationen betrachtet und mit Daten des vollständigen Modells verglichen, für das asymmetrischen Modell *f* wurden zwei Beispiele in zwei Variationen für einen *zu glücklich*-Übergang herangezogen (zwei Stück), welche ebenfalls mit Daten des vollständigen Modells verglichen wurden. Zuletzt wurden jeweils zwei Variationen von Ausdrucksübergänge mithilfe der Variationsmodelle erzeugt (zwei mal zwei Stück) und miteinander sowie mit dem vollständigen Modell verglichen. Für eine quantitative Evaluation des Modells *i* sind weiterhin die Vergleiche mit den Grundwahrheiten der jeweiligen Animation von Interesse. Für die Evaluation gilt zudem, dass Start- und Endframes der generierten Animation für alle Modelle *a* bis *h* übereinstimmen.

Für eine vollständige Evaluation des Modells werden daher zunächst Daten des vollständigen Modells mit der Grundwahrheit verglichen. Insbesondere werden dabei die Übergänge der ersten Iterationen der Datensynthese hervorgehoben und der Grundwahrheit gegenübergestellt.

Insgesamt wird die Evaluation somit aus zwei distinktiven Teilen bestehen, zum einen dem Vergleich zwischen dem Modell mit vollständigen Datensatz mit den Grundwahrheiten sowie zum anderen dem Vergleich der einzelnen Modelle *a* – *i*.

Kapitel 6

Ergebnisse

In diesem Kapitel werden zum einen die Ergebnisse bezüglich des Expression-Generators vorgestellt und zum anderen die Ergebnisse der numerischen Analyse, welche den wesentlichen Bestandteil der quantitativen Evaluation darstellen.

6.1. Training

Das KNN-System *Expression-Generator* wurde wie bereits beschrieben für insgesamt 400 Epochen trainiert, wobei pro Epoche zwei Fehler für alle Sequenzen des Datensatzes berechnet wurden: Zum einen handelt es sich dabei um den *Trainingsfehler*, zum anderen um den *Validierungsfehler*, wobei sich beide Fehler durch die mittlere quadratische Abweichung ergeben. Folglich ergibt sich der Trainingsfehler aus allen Daten mit denen der Expression-Generator trainiert wird, wohingegen sich der Validierungsfehler aus unbekannten Daten ergibt. Darüber hinaus wird zusätzlich ein sogenannter Testfehler nach 100 Epochen, 200 Epochen sowie nach 355 Epochen berechnet. Der Testfehler wird hierbei ebenfalls aus ausschließlich unbekannten Daten berechnet. Die Fehler über den ganzen Trainingsprozess von Modell i (vgl. Kapitel 5.5) sind in Abbildung 6.1 aufgeführt, wobei der Trainingsfehler in *blau* und der Validierungsfehler in *orange* dargestellt sind. Der zuvor angesprochene Testfehler ist *rot* hinterlegt. Das Fehlen der Epochen 356 bis 400 resultiert aus der Konvergenz des Modells bei Epoche 355, d.h. der Fehler veränderte sich nicht mehr. Der MSE-Fehler beschreibt dabei den durchschnittlichen Fehler aller Sequenzen eines Datensatzes (Anzahl Sequenzen: Trainingssatz - 205, Validierungssatz - 25 und Testsatz - 25), wobei sich der Fehler pro Sequenz aus der Summe aus den Differen-

6. Ergebnisse

zen zwischen tatsächlichen AU-Intensitäten und generierter AU-Intensitäten aller AUs berechnen lässt.

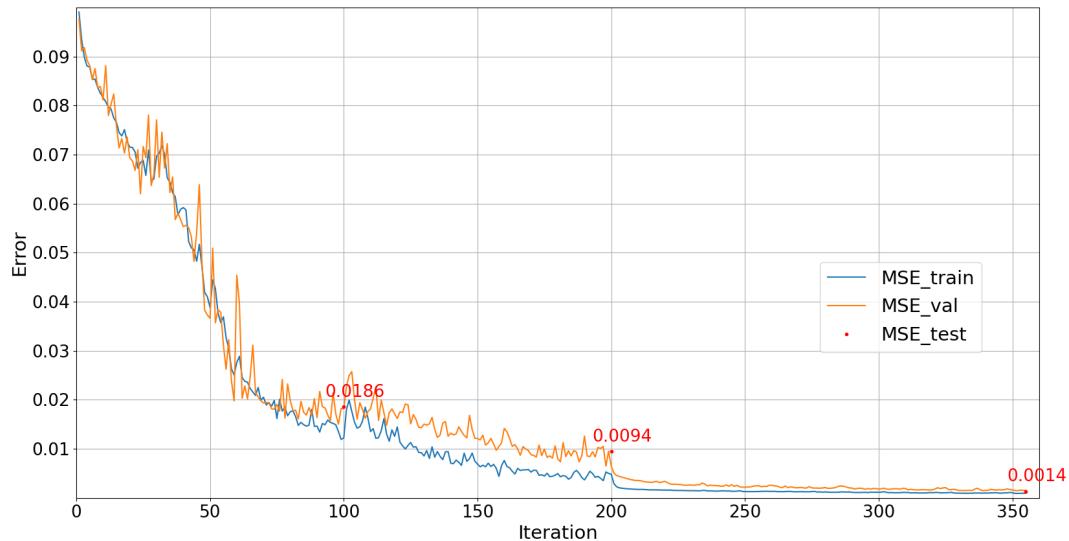


Abbildung 6.1.: Trainingsfehler über 355 Epochen des Expression-Generators

Wie an den Kurven deutlich zu erkennen ist, startet das Modell bei zufälligen Modellparametern bei einem MSE von 0,1 für sowohl Trainings- als auch Validierungsfehler. In den ersten 100 Epochen findet die größte Veränderung statt, wobei Trainings- und Validierungsfehler sich nahezu identisch entwickeln. Nach 100 Epochen wird erstmals der Testerror berechnet, welcher mit durchschnittlich 0,0186 ungefähr gleichauf mit dem Validierungsfehler liegt und somit bereits bei ca. minus 81,4 % des Ursprungsfehlers. In den nächsten 100 Epochen findet dagegen nur eine weitaus geringere Verbesserung statt, welche für den Testerror von 0,0094 bei ca. minus 9,2 % des Ausgangswertes liegt. Im letzten Abschnitt von Epoche 201 bis Epoche 355 findet zu Beginn eine, im Vergleich zum Rest des Trainingsabschnitts, relativ große Veränderung statt, bevor der Trainingsfehler anschließend beinahe stagniert. Der Validierungsfehler nähert sich im Verlauf dieses Abschnitts immer mehr dem Trainingsfehler an. Für das beste Modell nach 355 Epochen beträgt der Testfehler gerundet 0,0014, was prozentual gesehen eine Veränderung von minus 98,6 % zum Ausgangswert darstellt. Allerdings muss an dieser Stelle beachtet werden, dass der Fehler durch das Quadrieren bei der Fehlerberechnung deutlich kleiner ausfällt, als der absolute Unterschied.

6.2. Expression-Generator

In den nachfolgenden Abschnitten werden Ergebnisse des Modells i , welches mit dem vollständigen Datensatz trainiert worden ist, vorgestellt. Dabei wird zunächst der absolute Fehler anhand von zwei Varianten von Daten, welche im Datensatz enthaltenen sind, vorgestellt bevor anschließend datensatzfremde Animationen einbezogen und visualisiert werden. Bei diesen generierten Daten wurde ausschließlich das beste Modell im Bezug auf die Validierungsdaten des gesamten Trainingsprozesses verwendet. Bei den ausgewählten Beispielen handelt es sich zum einen um Animationen, bei denen visuell, d.h. bei der Animation des Avatars, keine ungewöhnlichen bzw. unpassenden Artefakte gesehen werden können, und zum anderen um Animationen, welche solche Artefakte aufweisen (auffällige Animationen).

6.2.1. Absoluter Fehler

Der absolute Fehler wird exemplarisch für jeweils zwei Animationen pro Ausdrucksübergang aufgeführt, wobei es sich bei der zweiten Animationsvariante, um eine datensatzfremde Variante der Animation handelt. Für ein besseres Verständnis der Daten sollen die zwei Ausdrucksübergänge zunächst jedoch am Beispiel der visuell unauffälligen Animation genauer erläutert und anschließend visualisiert werden. Der erste Ausdrucksübergang (visuell unauffällig) wurde aus dem Ausdrucks-paar *angewidert* zu *glücklich* generiert, welcher so identisch im Datensatz vorhanden ist. Der zweite Ausdrucksübergang *angewidert* zu *glücklich* resultiert aus einem semantisch vorhandenen Ausdruckspaar, jedoch wurden hierbei die AU-Intensitäten verschiedener anderer Ausdrucksübergänge verwendet. Konkret bedeutet dies, dass der Startausdruck *angewidert* der generierten Animation aus einem Endausdruck einer unterschiedlichen Animation (überrascht zu angewidert) entnommen wurde. Für den Endausdruck gilt analog, dass der Startausdruck einer unterschiedlichen Animation (glücklich zu neutral) verwendet wurde. Bei dieser Animation handelt es sich ebenfalls um eine visuell unauffällige Animation.

Die Abbildungen 6.2a und 6.2b visualisieren die ganze Sequenz des vorhandenen Ausdrucksübergangs pro Frame für jede verwendete AU. Auf der linken Seite 6.2a sind die Daten der Grundwahrheit visualisiert, auf der rechten 6.2b die Daten des Expression-Generators. Eine wesentliche Auffälligkeit innerhalb der Daten ist direkt zu Beginn, also ab dem 20. Frame zu erkennen.

6. Ergebnisse

Tabelle 6.1.: Erster generierter Frame für bekannte Daten angewidert zu glücklich

Frame	AU-Intensität														
	AU1L	AU1R	AU2L	AU2R	AU4L	AU4R	AU6L	AU6R	AU9	AU10	AU13L	AU13R	AU18	AU22	AU27
19	0.00	0.00	0.00	0.04	1.20	1.20	0.03	0.00	0.00	0.24	0.38	0.00	0.00	0.00	0.00
20	0.06	0.07	0.04	0.12	1.24	1.24	0.07	0.09	0.10	0.18	0.24	-0.05	-0.08	-0.01	0.00
Δ	0.06	0.07	0.04	0.08	0.04	0.04	0.04	0.09	0.10	-0.06	-0.14	-0.05	-0.08	-0.01	0.00
Genauigkeit %	95.30	94.35	96.94	93.08	96.71	96.40	96.43	92.84	92.07	94.92	88.27	95.96	93.71	99.32	99.74
Abweichung %	4.70	5.65	3.06	6.92	3.29	3.60	3.57	7.16	7.93	5.08	11.73	4.04	6.29	0.68	0.26

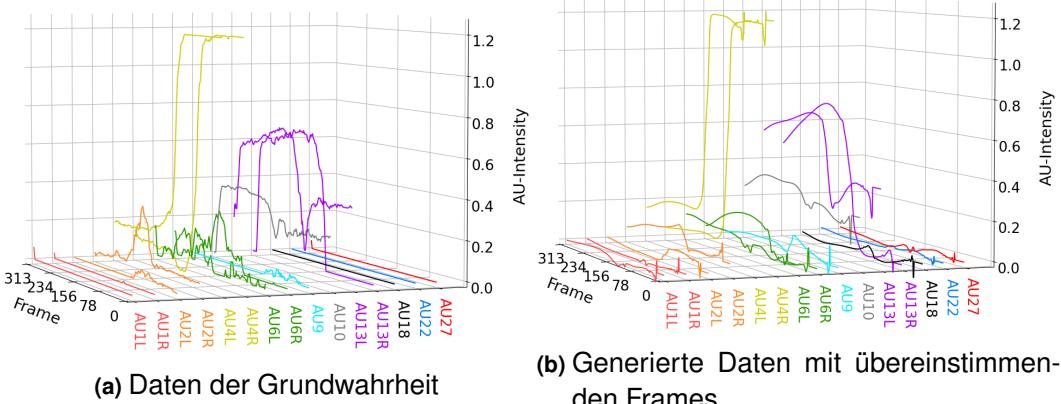


Abbildung 6.2.: Modell i und GT - Vergleich für die Animation angewidert zu glücklich

Die AU-Intensitäten des Expression-Generators weisen eine ersichtliche Veränderung am 20. Frame der Animation auf, welche in den darauffolgenden Frames wieder rückgängig gemacht werden. Diese Veränderungen, fortan als *Sprünge* bezeichnet, sind gesondert in Tabelle 6.1 festgehalten.

Anhand Tabelle 6.1 sind diese Sprünge im Bereich zwischen 0,14 bis 0,00 einzuschränken, was einer prozentualen Veränderung in Abhängigkeit der Maximalintensität 1,20 zwischen 11,73 % und <1 % entspricht, wobei der Mittelwert bei 4,93 % liegt. Ein wesentlicher Unterschied ist anhand der Endpunkte der AUs AU10 sowie AU13L/R zu erkennen, für welche die Intensitäten des Expression-Generators beim Endwert nicht die Werte der Grundwahrheit entsprechen. Eine generelle Auffälligkeit ist die Glattheit der Kurven des Expression-Generators im Vergleich zu den Daten der Grundwahrheit, bei denen die Intensitäten zumeist kleinere Schwankungen aufweisen. Für einen quantitativen Vergleich der Performance des Expression-Generators wurde, wie zuvor beschrieben, auch eine Variation der glücklich-zu-angewidert-Animation generiert. Die Visualisierung dieser Variation ist in Abbildung 6.3 zu sehen, wobei die Daten der Grundwahrheit erneut links zu erkennen sind. Bereits bei erster Betrachtung sind die Unterschiede deutlich zu erkennen, wobei sich einige dieser Unterschiede durch die verschiedene Ausführung der Gesichtsausdrücke durch den Akteur erklären lassen.

6. Ergebnisse

Tabelle 6.2.: Erster generierter Frame für Variationsdaten angewidert zu glücklich

Frame	AU-Intensität														
	AU1L	AU1R	AU2L	AU2R	AU4L	AU4R	AU6L	AU6R	AU9	AU10	AU13L	AU13R	AU18	AU22	AU27
19	0.00	0.00	0.13	0.07	1.20	1.17	0.21	0.05	0.04	0.45	0.40	0.00	0.00	0.00	0.00
20	-0.20	-0.16	-0.20	-0.07	1.04	1.13	0.07	0.04	0.13	0.16	0.21	0.02	0.11	0.07	-0.08
Δ	-0.20	-0.16	-0.33	-0.14	-0.16	-0.04	-0.15	-0.01	0.09	-0.30	-0.19	0.02	0.11	0.07	-0.08
Genauigkeit %	83.00	86.93	72.65	88.58	86.28	96.83	87.74	99.18	92.69	75.36	84.02	98.50	91.12	93.78	92.99
Abweichung %	17.00	13.07	27.35	11.42	13.72	3.17	12.26	0.82	7.31	24.64	15.98	1.50	8.88	6.22	7.01

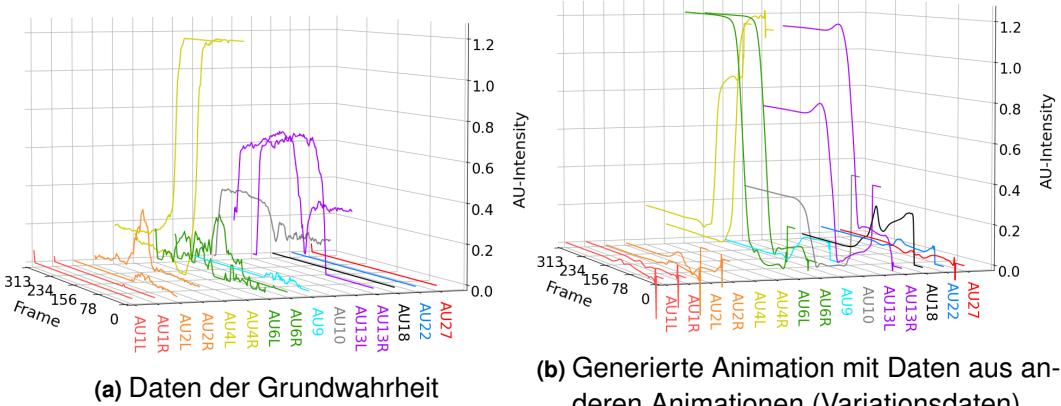


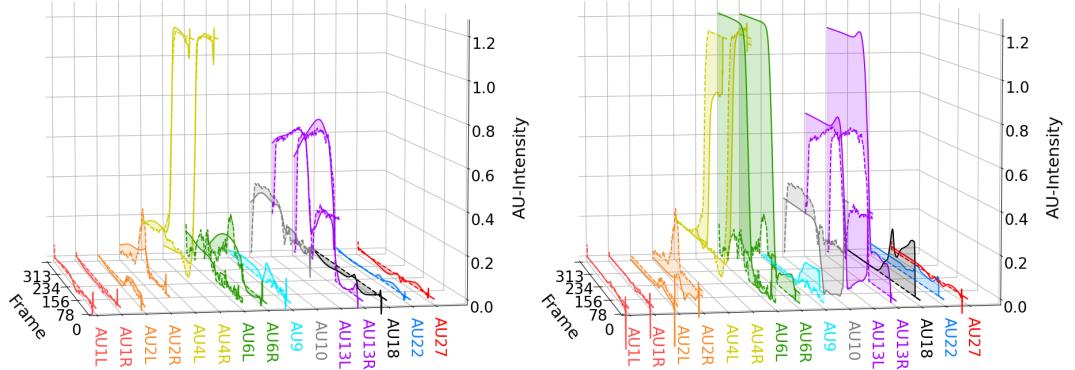
Abbildung 6.3.: Modell i und GT - Vergleich für die Animation angewidert zu glücklich mit Variationsdaten

Dies gilt insbesondere für die AU6 (grün) ("Cheek-Raiser"), welche für die Bewegungen des oberen Backenbereichs zuständig ist, sowie für die AU13R (lila) ("Cheek-Puffer"), welche für den unteren Backenbereich zuständig ist [7].

Im Vergleich zu der generierten Animation aus den Daten der Grundwahrheit fällt der Sprung beim 20. Frame zudem deutlich größer aus. Sämtliche Differenzen sind hierbei in Tabelle 6.2 aufgeführt. Die prozentuale Abweichung der Variation liegt zwischen 26,26 % und 0,79 % und weist einen Mittelwert von 10,9 % auf, welcher demnach mehr als doppelt so hoch wie bei der Animation, welche aus dem Ausdrucksübergang innerhalb des Datensatzes erstellt wurde, liegt. Ein direkter Vergleich beider Animationen ist in Abbildung 6.4 visualisiert. Die ausgefüllte Fläche zwischen den Graphen hebt hierbei die Differenz zwischen der Grundwahrheit und den generierten Daten hervor, wobei die Daten aus Abbildung 6.2 und Abbildung 6.3 zusammengeführt sind. Neben der zuvor angesprochenen Differenz der AU6 sowie AU13 wird in dieser Darstellung ebenfalls eine Differenz in der AU4 (gelb) ("brow lowerer"), welche für die Senkung der Augenbrauen zuständig ist [7], ersichtlich. Diese Differenz resultiert hierbei aus einer hauptsächlich temporalen Verschiebung der Aktivierung, also der Veränderung der AU.

Für einen direkten Vergleich beider Animationen eignet sich neben dem Vergleich der Daten ebenfalls der absolute Fehler zwischen Grundwahrheit und der generier-

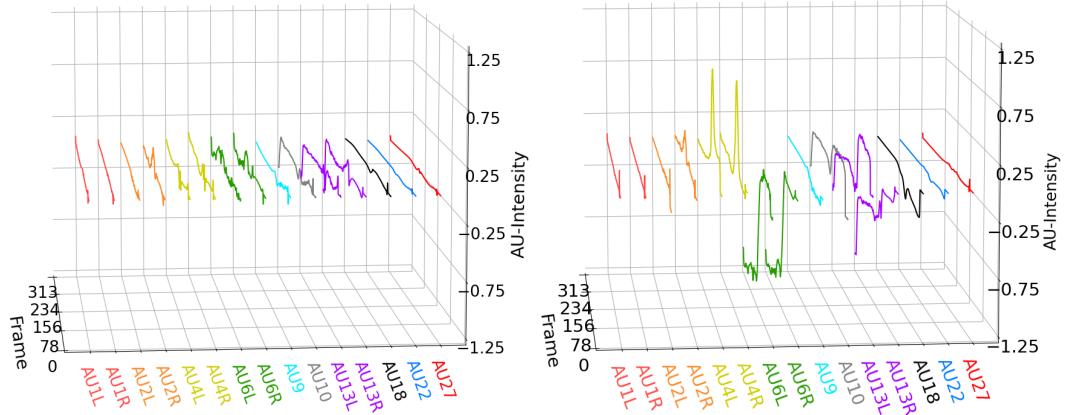
6. Ergebnisse



(a) Vergleich GT mit generierter Animation aus gleichen Frames (b) Vergleich GT mit generierter Animation aus Variationsdaten

Abbildung 6.4.: Modell i und GT - Visualisierung der Abweichungen der generierten Animationen für angewidert zu glücklich

ten Animation. Dieser absolute Fehler lässt sich dabei aus der Differenz zwischen Grundwahrheit und der generierten Animation pro Frame berechnen und visualisieren. Diese Differenz ist in Abbildung 6.5 für beide Varianten des Ausdrucksübergangs glücklich zu angewidert visualisiert. In Abbildung 6.5a sind die zuvor angesprochenen Sprünge ebenfalls kaum erkennbar, wohingegen in Abbildung 6.5b die Sprünge deutlich erkennbar sind. Weiterhin werden große Abweichungen wie bswp. bei AU6 hervorgehoben.



(a) Differenz GT mit generierter Animation aus gleichen Frames (b) Differenz GT mit generierter Animation aus Variationsdaten

Abbildung 6.5.: Modell i und GT - Visualisierung der Differenz zwischen GT und Modell i der beiden vorgestellten generierten Animationen für angewidert zu glücklich

Nachdem bisher die Daten der visuell unauffälligen Animation *angewidert zu glücklich* visualisiert wurden, werden im Folgenden die Daten einer visuell auffälligen

6. Ergebnisse

Animation vorgestellt. Bei dieser Animation handelt es sich um den Ausdrucksübergang von *stirnrunzelnd* zu *neutral*. Bei der visuell auffälligen Animation handelt es sich jedoch lediglich um das Variationsmodell, weshalb die Animation mit den übereinstimmenden Daten nicht aufgeführt wird. Bei der visuellen Auffälligkeit der Variation handelt es sich um deutlich wahrnehmbare Sprünge von einem Startausdruck zum Endausdruck, eine stetige Veränderung ist nicht sichtbar. Der Verlauf der Daten dieser Animation, zum einen der Grundwahrheit, zum anderen der Variation, sind in Abbildung 6.6 dargestellt. Bei der ersten Betrachtung der Animationsdaten sind bereits deutliche Unterschiede unverkennbar. Zum einen handelt es sich dabei um auffällige Sprünge direkt zu Beginn der Animation für AU6, AU8, AU10, AU13 sowie AU18 und AU22, zum anderen ist jedoch auch eine merkbare Aktivierung der AU6 nicht vorgesehen, vgl. AU6 in 6.6a.

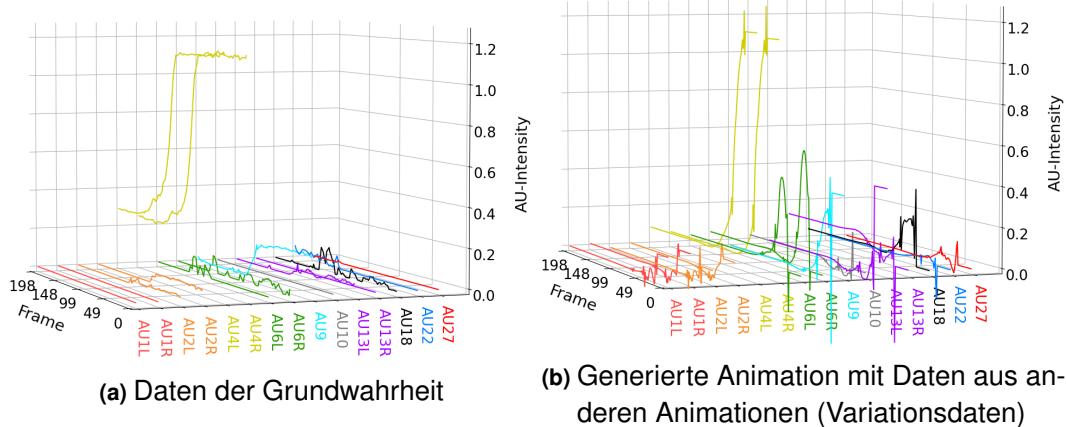


Abbildung 6.6.: Modell i und GT - Vergleich für die Animation stirnzunzelnd zu neutral mit Variationsdaten

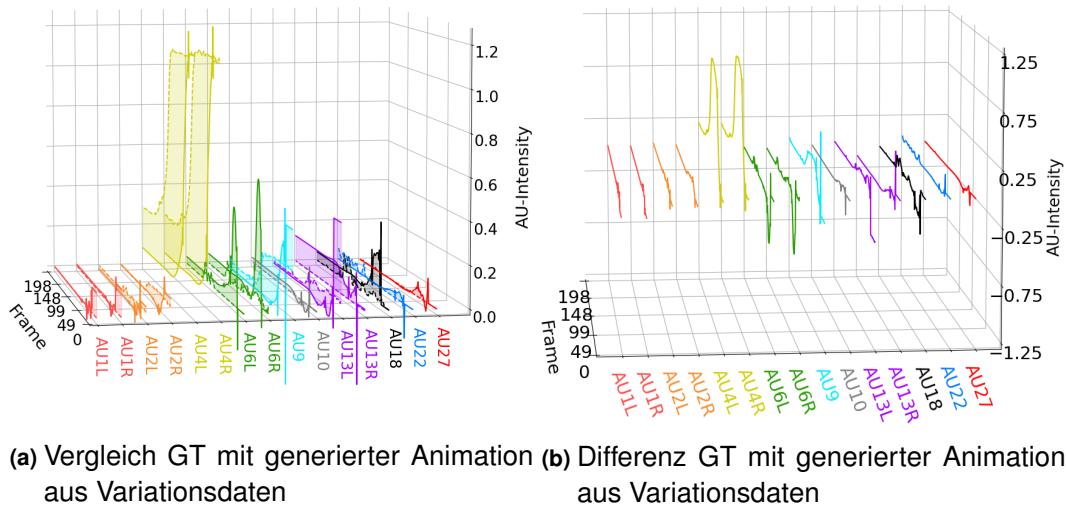
Die einzelnen Sprünge für den ersten Frame, also beim 20. Frame der Animation, sind für diesen Ausdrucksübergang Tabelle 6.3 zu entnehmen. Die prozentualen Abweichungen von Frame 19 zu Frame 20 liegen zwischen 1,13 % und 61,18 % und weisen im Durchschnitt einen Wert von 17,49 % auf. Somit liegt der Mittelwert deutlich höher, als bei den bisher betrachteten Animationen, wobei die Sprünge für einige AUs (z.B. AU1 und AU2) in einem vergleichbaren Bereich liegen.

Tabelle 6.3.: Erster generierter Frame für Variationsdaten stirnrunzelnd zu neutral

Frame	AU-Intensität														
	AU1L	AU1R	AU2L	AU2R	AU4L	AU4R	AU6L	AU6R	AU9	AU10	AU13L	AU13R	AU18	AU22	AU27
19	0.12	0.13	0.05	0.08	1.16	1.13	0.00	0.00	0.38	0.00	0.41	0.01	0.00	0.00	0.00
20	0.02	0.10	-0.04	0.09	1.05	1.05	-0.19	-0.09	-0.36	0.18	-0.10	-0.37	0.38	-0.16	0.12
Δ	-0.10	-0.03	-0.08	0.01	-0.11	-0.08	-0.19	-0.09	-0.73	0.18	-0.51	-0.37	0.38	-0.16	0.12
Genauigkeit %	91.77	97.43	93.22	98.87	90.86	93.33	84.15	92.34	38.82	85.07	57.67	68.77	68.26	86.77	90.31
Abweichung %	8.23	2.57	6.78	1.13	9.14	6.67	15.85	7.66	61.18	14.93	42.33	31.23	31.74	13.23	9.69

6. Ergebnisse

Für eine direkte Gegenüberstellung findet ebenfalls ein Vergleich zwischen Grundwahrheit und generierter Animation statt. Die Visualisierung für den Vergleich sowie die Visualisierung der Differenzen sind in Abbildung 6.7 dargestellt. Mithilfe der Abbildung 6.7a werden die temporalen Unterschiede von AU4 verdeutlicht, wobei diese nicht nur früher, sondern auch verstärkter auftreten, was an der erhöhten Steigung zu erkennen ist. Die abschließende Differenz zwischen Grundwahrheit und generierter Animation lässt sich durch die unterschiedliche Ausführung durch den Akteur erklären.



(a) Vergleich GT mit generierter Animation aus Variationsdaten

(b) Differenz GT mit generierter Animation aus Variationsdaten

Abbildung 6.7.: Modell i und GT - Visualisierung der Unterschiede für die Animation stirnrunzelnd zu neutral mit Variationsdaten

Die zu Beginn angesprochene Wahrnehmung von lediglich sprunghaften Veränderungen der AUs werden in Abbildung 6.7b verdeutlicht. Besonders der Verlauf der Differenzen bei AU9 spiegeln dieses Verhalten wider. Die Differenz zwischen generierter Animation und Grundwahrheit wechselt innerhalb weniger Frames von unter null zu über null und wieder zurück, bevor anschließend eine steile, jedoch stetige Steigung erkennbar ist. Aufgrund der Tatsache, dass sämtliche Veränderungen innerhalb weniger Frames stattfinden, sind diese für das Auge kaum wahrzunehmen. Zur Erinnerung: die Animationen werden mit 120 Frames pro Sekunde aufgenommen.

6.2.2. Datensatzfremde Animationen

Zuletzt sollen für das vollständige Modell (*i*) auch Ausdrucksübergänge von Ausdrücken betrachtet werden, die in keiner Weise im Datensatz vorhanden sind. Die Daten, also Start- und Endframes dieser Ausdrucksübergänge werden nach dem selben Prinzip wie die Daten der Variationsanimation gewonnen, d.h. Start- sowie Endframe stammen von unterschiedlichen Animationen, wobei wie für die Variation gilt, dass Startframe aus dem Ende einer Animation und Endframe aus dem Start einer anderen Animation entnommen wurden. Für die datensatzfremden Animationen wurde exemplarisch ein Beispiel für eine visuell unauffällige Animation - *überrascht zu angewidert*, herangezogen sowie ein Beispiel für eine visuell Auffällige - *überrascht zu traurig*.

Die Daten der visuell unauffälligen Animation *überrascht zu angewidert* sind in Abbildung 6.8 dargestellt, wobei es sich bei beiden Teilabbildungen 6.8a und 6.8b um die selben Daten, lediglich in einem anderen Betrachtungswinkel handelt, damit die Daten besser beschrieben werden können. In Abbildung 6.8a sind die relativ gleichmäßigen Veränderungen der AUs deutlich wahrnehmbar. Weiterhin ist erkennbar, dass die Sprünge vom 19. auf den 20. Frame nicht auf einmal stattfinden, sondern über mehrere Frames. Sämtliche Sprünge sind hierfür in Tabelle 6.4 angegeben und bestätigen diese Beobachtung, welche bspw. anhand der AU1L deutlich abzulesen ist. In der visuellen Darstellung kann wahrgenommen werden, dass die Intensität der AU zunächst minimal ansteigt, bevor diese anschließend um ca. 0,2 absinkt, was eher einem Sprung entspricht. Es bleibt jedoch zu erwähnen, dass solch geringe Abweichungen bei der Avataranimation kaum wahrzunehmen sind.

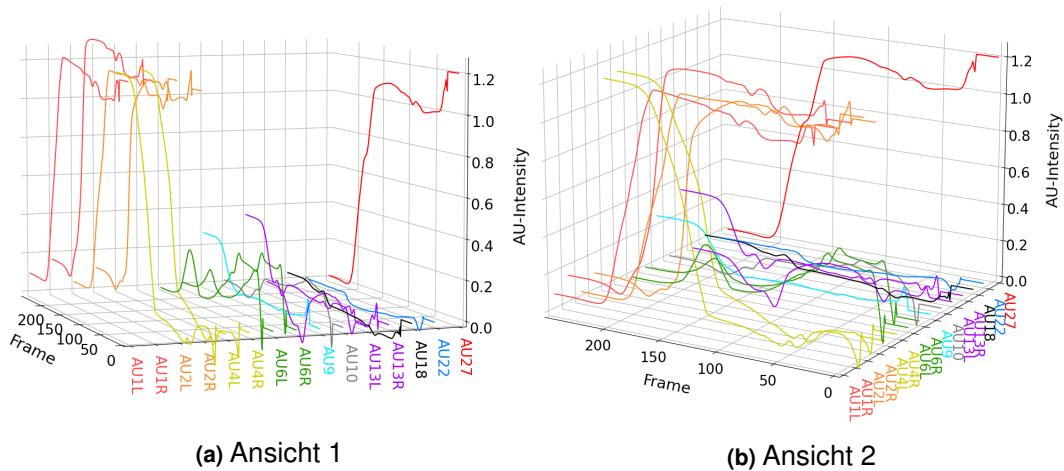


Abbildung 6.8.: Modell i - Datensatzfremde Animation überrascht zu stirnrunzelnd

6. Ergebnisse

Tabelle 6.4.: Erster generierter Frame für datensatzfremde Animation überrascht zu angewidert

Frame	AU-Intensität														
	AU1L	AU1R	AU2L	AU2R	AU4L	AU4R	AU6L	AU6R	AU9	AU10	AU13L	AU13R	AU18	AU22	AU27
19	1.20	1.20	1.20	1.15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.20
20	1.19	1.17	1.22	1.12	0.19	0.17	0.11	0.08	-0.02	0.05	-0.03	-0.07	0.00	0.01	1.15
Δ	-0.01	-0.03	0.02	-0.03	0.19	0.17	0.11	0.08	-0.02	0.05	-0.03	-0.07	0.00	0.01	-0.05
Genauigkeit %	98.81	97.79	98.37	97.91	83.99	85.75	90.60	93.69	98.69	95.84	97.34	93.93	99.68	99.57	95.61
Abweichung %	1.19	2.21	1.63	2.09	16.01	14.25	9.40	6.31	1.31	4.16	2.66	6.07	0.32	0.43	4.39

Gleichzeitig kann in Abbildung 6.8b beobachtet werden, dass nicht relevante AUs wie beispielsweise AU18 und AU22 (beide für die Lippen zuständig) kaum Veränderungen aufweisen und größtenteils inaktiv bleiben.

Die in Tabelle 6.4 aufgelisteten Sprünge reichen von einem Minimalwert von 0,32 %, was einer AU-Intensitätsabweichung von weniger als 0,00 entspricht, bis hin zum Maximalwert von 16,01 %, was wiederum einer Intensitätsabweichung von ca. 0,19 entspricht. Im Durchschnitt liegt die Abweichung der visuell unauffälligen, datensatzfremden Animation bei 4,83 %, was unter dem Mittelwert der in Abschnitt 6.2.1 beschriebenen Abweichung der Animation mit übereinstimmenden Daten liegt (vgl. Tabelle 6.1).

Im Vergleich dazu sind die Daten der visuell auffälligen Animation in Abbildung 6.9 dargestellt. Bei der visuellen Betrachtung dieser Animation können zwei distinktive Auffälligkeiten beobachtet werden. Zum einen ist zu Beginn der Animation ein deutlicher Sprung erkennbar, zum anderen kann beim konkreten Übergang von überrascht zu traurig eine ungewollte Reaktivierung mehrerer AUs wahrgenommen werden, bevor der Übergang vollendet wird.

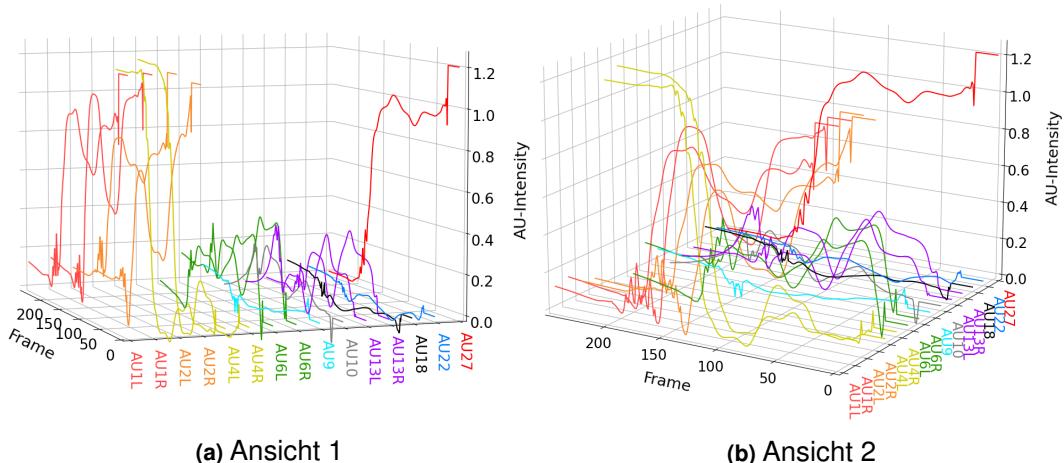


Abbildung 6.9.: Modell i - Datensatzfremde Animation überrascht zu traurig

6. Ergebnisse

Tabelle 6.5.: Erster generierter Frame für datensatzfremde Animation überrascht zu traurig

Frame	AU-Intensität														
	AU1L	AU1R	AU2L	AU2R	AU4L	AU4R	AU6L	AU6R	AU9	AU10	AU13L	AU13R	AU18	AU22	AU27
19	1.20	1.20	1.20	1.15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.20
20	0.87	0.93	0.85	0.92	0.07	0.09	0.10	0.13	0.04	-0.07	0.01	0.01	-0.04	0.03	0.91
Δ	-0.33	-0.27	-0.35	-0.23	0.07	0.09	0.10	0.13	0.04	-0.07	0.01	0.01	-0.04	0.03	-0.29
Genauigkeit %	72.46	77.87	70.93	80.66	94.07	92.67	91.70	88.88	96.64	93.88	99.11	99.40	96.95	97.12	75.94
Abweichung %	27.54	22.13	29.07	19.34	5.93	7.33	8.30	11.12	3.36	6.12	0.89	0.60	3.05	2.88	24.06

Beide soeben beschriebenen Auffälligkeiten lassen sich in Abbildung 6.9 wiederfinden. Der unverkennbare Sprung der Animation ist in der Teilabbildung 6.9b deutlich anhand der Verläufe der AU1L/R, AU2L/R sowie AU27 absehbar, wohingegen die Reaktivierung deutlich durch das Ansteigen der Intensitäten, insbesondere für AU1L/R sowie AU2L/R, in der Teilabbildung 6.9a sichtbar ist.

Die genauen Werte für die Sprünge sind in Tabelle 6.5 zu finden. Die Sprünge reichen hierbei von einer Intensitätsänderung von 0,01 bis 0,35, was prozentual den Abweichungen von 0,89 % und 29,07 % entspricht. Im Durchschnitt liegt die Abweichung bei 11,45 %, jedoch muss festgehalten werden, dass nur wenige Werte um den tatsächlichen Mittelwert liegen.

6.3. Quantitative Evaluation

6.3.1. Definition der Kriterien

Nachdem in 6.2 die Ergebnisse des *i-ten* Modells vorgestellt wurden, ist es möglich Kriterien zu definieren, nach denen visuell auffällige Animationen anhand ihrer Daten erkannt werden können. Im Rahmen dieser Arbeit gilt daher:

Eine Animation gilt fortan als visuell auffällig, sofern eine oder mehrere der folgenden Bedingungen eintrifft:

1. Absolute Differenz Grundwahrheit und generierter Animation > 1.00
2. Existenz von Sprüngen:
 - ein Sprung mit $> 40\%$ Abweichung
 - drei Sprünge mit $> 20\%$ Abweichung
3. Deutliche Reaktivierung von AUs ($> 25\%$ Reaktivierung)
4. AU-Intensitäten für Start- und/oder Zielframe unterscheiden sich zu jeglicher Variation der Ausdrücke ($> 25\%$ Abweichung)

Anmerkung:

Die soeben genannten Kriterien wurden nach subjektiver Beobachtung herausgearbeitet und stellen lediglich eine vereinfachte Grundlage zur quantitativen Bewertung von generierten Animationen dar, weshalb sämtliche Animationen zusätzlich visuelle beurteilt wurden, um die Ergebnisse und Richtwerte zu validieren. Weiterhin gilt, dass für die numerische Analyse vorwiegend die Kriterien eins und zwei überprüft werden.

6.3.2. Numerische Analyse

Bei den für die quantitative Evaluation generierten Animationen handelt es sich um folgende Ausdrucksübergänge:

- von glücklich zu neutral
 - von glücklich zu überrascht
 - von neutral zu glücklich
 - von überrascht zu glücklich
- sowie
- von überrascht zu angewidert

Letztere Animation ist in sämtlichen Datensätzen enthalten und dient der Validierung, dass das Modell in der generell in der Lage ist, visuell unauffällige Animationen zu erzeugen. Jede Animation wurde zudem in zwei Variationen, also jeweils zwei verschiedene Start- und Endframes, erstellt. Eine Übersicht über welche Daten für das Training der Modelle verwendet wurden, können Tabelle 5.3 entnommen werden.

Symmetrische Modelle a bis e

Die absoluten Fehler für die symmetrischen Modelle können Tabelle 6.6 entnommen werden. Zusätzlich zu den absoluten Fehlern in Tabelle 6.6 sind in Anhang C die Sprünge aller Modelle aufgeführt. Durch eine gleichzeitige Betrachtung des absoluten Fehlers sowie der Sprünge ist es möglich, Animationen weitestgehend als visuell auffällig bzw. unauffällig zu klassifizieren.

6. Ergebnisse

Tabelle 6.6.: Absoluter Fehler für die symmetrischen Modelle inklusive des vollständigen Modells

Animation			Fehler					
von	zu	Variation	a	b	c	d	e	i
<i>glücklich</i>	<i>neutral</i>	1	3,34	1,08	2,58	1,67	0,20	0,12
		5	2,42	1,18	3,78	2,20	0,21	0,16
<i>glücklich</i>	<i>überrascht</i>	1	3,47	3,69	3,13	0,56	0,23	0,25
		5	2,56	2,47	2,29	0,31	0,24	0,27
<i>neutral</i>	<i>glücklich</i>	1	8,68	2,99	3,27	1,99	0,46	0,11
		5	6,79	1,71	1,20	1,31	0,11	0,07
<i>überrascht</i>	<i>glücklich</i>	1	2,37	1,07	2,92	0,68	0,43	0,58
		5	2,62	3,85	4,28	0,22	0,14	0,07
<i>überrascht</i>	<i>angewidert</i>	1	0,13	0,28	0,17	0,33	0,15	0,08
		5	0,19	1,07	0,21	0,37	0,20	0,07

Anhand der in 6.3.1 definierten Kriterien *Absolute Differenz Grundwahrheit und generierter Animation* > 1.00 und *Existenz von Sprüngen* kann die grundlegende Funktionalität lediglich bei den Modellen *a* und *e* validiert werden. Modell *c* erzielt, bezogen auf die Sprünge sowie den absoluten Fehler, eine merkbar bessere Performance als die Modelle *b* und *d*, wird dennoch aufgrund der Existenz von drei Sprüngen über 20 % als auffällig klassifiziert. Es ist zu beachten, dass Ausdrucksübergänge für *glücklich* zu *überrascht* und *überrascht* zu *glücklich* in den Trainingsdaten für die Modelle *d* und *e* enthalten sind. Die Ausdrucksübergänge *neutral* zu *glücklich* sowie die umgekehrte Variante *glücklich* zu *neutral* sind lediglich im Trainingsdatensatz für Modell *e* enthalten. Bei der Untersuchung der Animationen von Modell *a* durch den Avatar, wird deutlich, dass sämtliche Animationen Elemente von *überrascht* aufweisen.

Für das Modell *b* fallen in Spalte *b* die Fehlerwerte für *glücklich* zu *neutral* sowie die erste Variation für *überrascht* zu *glücklich* durch einen auffallend geringeren Wert als die restlichen Fehlerwerte dieses Modells auf. Deshalb werden im Folgenden genau diese beiden Animationen weiter beleuchtet, um tiefer zu untersuchen, wie das Modell die unbekannten Animationsdaten generiert. Die Existenz der Sprünge und die damit verbundene Klassifizierung als auffällig wird hierbei zunächst bewusst vernachlässigt.

In Abbildung 6.10 sind die zuvor hervorgehobenen Animationen für das Modell *b* visuell dargestellt, wobei die gestrichelte Kurven die Animationen der Grundwahrheit darstellt. Rechts sind jeweils die Daten von Modell *i* für einen direkten Vergleich beider Modelle visualisiert.

6. Ergebnisse

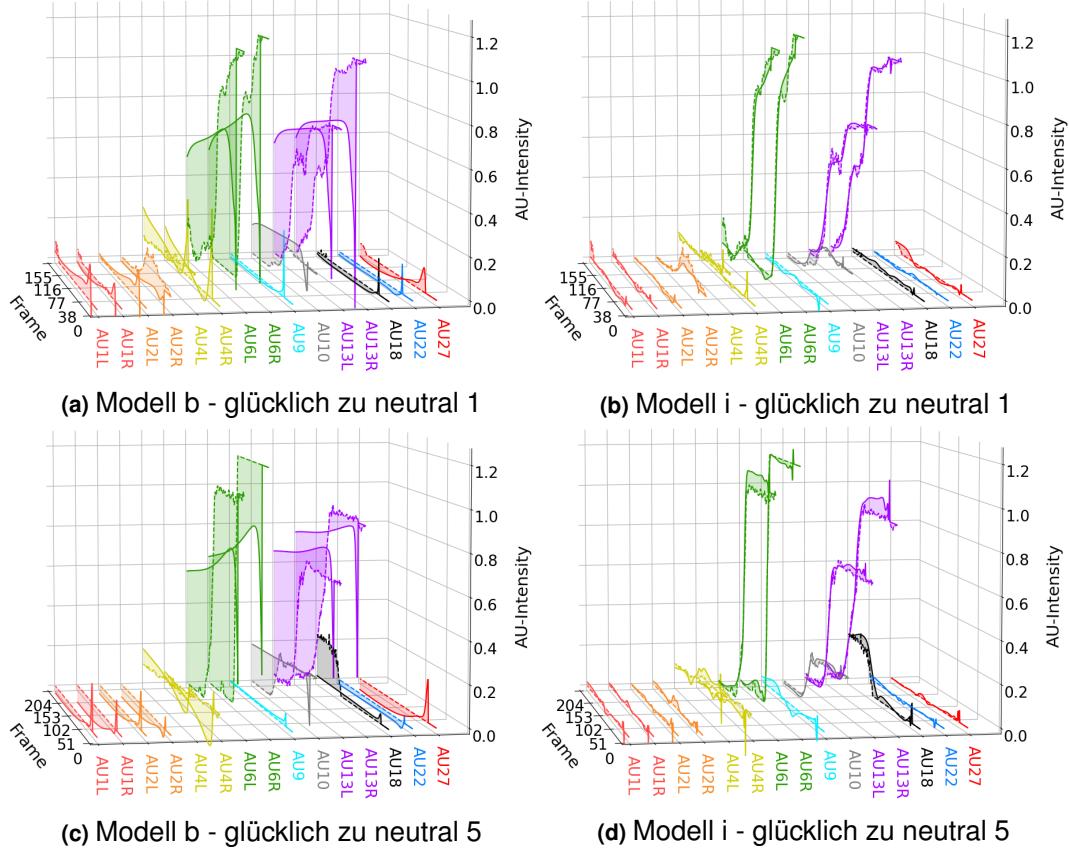


Abbildung 6.10.: Modell b und i - Visueller Vergleich glücklich zu neutral in zwei Variationen

Anhand der Verläufe der Kurven in 6.10a und 6.10c kann der geringe Fehler aufgrund der kaum aktiven AUs (alle bis auf AU6L/R (grün) sowie AU13L/R (lila)) erklärt werden, welche die Fehler der aktiven AUs ausgleichen. Dies liegt vorwiegend an der Art der Fehlerberechnung, bei welcher sich der Gesamtfehler aus dem Durchschnitt aller AUs berechnet.

In Abbildung 6.11 sind die Daten der Animation *überrascht zu glücklich* von Variation 1 der Modelle *b* und *i* dargestellt. Im Vergleich zu den zuvor betrachteten Animationen kann eine visuelle Auffälligkeit ausschließlich anhand der signifikanten Sprünge (vgl. 6.3.1), insbesondere für AU1L/R (rot-links), AU2L/R (orange), AU4L/R (gelb) sowie AU27 (rot-rechts) identifiziert werden. Die verhältnismäßig starke Aktivierung der AU6R (grün) wird visuell nicht wahrgenommen. Daraus folgt, dass abgesehen von den signifikanten Sprüngen diese Animation ohne das Vorhandensein im Datensatz generiert werden kann.

6. Ergebnisse

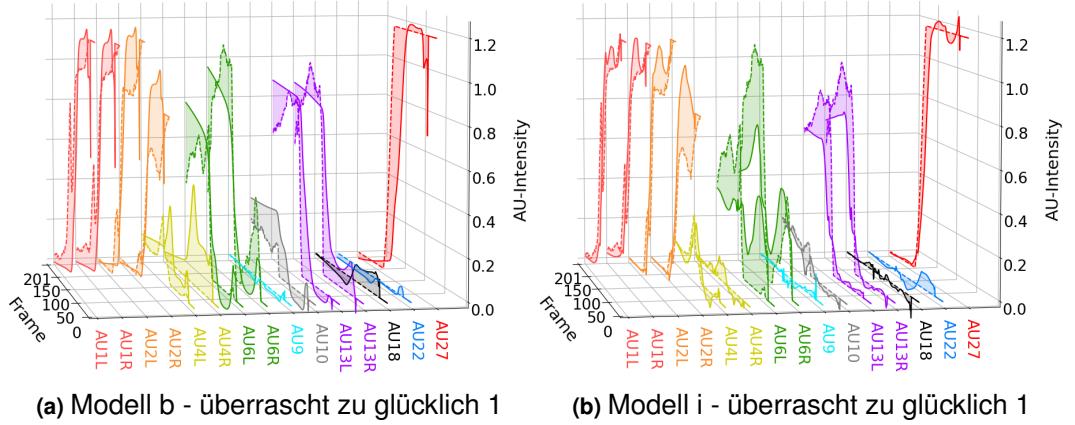


Abbildung 6.11.: Modell b und i - Visueller Vergleich überrascht zu glücklich

Asymmetrisches Modell f

Die absoluten Fehler für das asymmetrischen Modells können Tabelle 6.7 entnommen werden. Ebenfalls wie bei den Modellen zuvor sind in Anhang C auch die Sprünge des asymmetrischen Modells enthalten. Anhand der Validierungsdaten sowohl für die Fehler als auch für die Sprünge, kann eine grundlegende Funktionalität dieses Modells festgehalten werden. Bei Betrachtung der Fehlerwerte ist eine deutliche Diskrepanz zwischen von- und zu- *glücklich*-Ausdrücken erkennbar. Bei näherer Betrachtung der Sprünge in Tabelle C.6 für jene Animationen außerhalb des Trainingsdatensatzes können teilweise Abweichung von über 100 % beobachtet werden, wohingegen die restlichen Abweichungen (Daten in Trainingsdatensatz) in einem ähnlichen Bereich liegen wie bei den Animationen, welche mit den Variationsdaten unter Verwendung von Modell *i* generiert worden sind.

Tabelle 6.7.: Absoluter Fehler für das asymmetrische Modell inklusive des vollständigen Modells

Animation			Fehler	
von	zu	Variation	<i>f</i>	<i>i</i>
<i>glücklich</i>	<i>neutral</i>	1	0,09	0,12
		5	0,22	0,16
<i>glücklich</i>	<i>überrascht</i>	1	0,16	0,25
		5	0,18	0,27
<i>neutral</i>	<i>glücklich</i>	1	5,33	0,11
		5	6,25	0,07
<i>überrascht</i>	<i>glücklich</i>	1	4,44	0,58
		5	7,09	0,07
<i>überrascht</i>	<i>angewidert</i>	1	0,05	0,08
		5	0,06	0,07

Variationsmodelle g und h

Die Trainingsdatensätze der Modelle g und h enthalten nur einen Teil der Animationsvariationen sämtlicher von- sowie zu- *glücklich*-Animationsdaten. Für Modell g handelt es sich dabei nur um eine Variation - Variation eins, der Trainingsdatensatz von Modell h beinhaltet die Variationen eins bis drei.

Nach Überprüfung der Kriterien *Absolute Differenz Grundwahrheit und generierter Animation > 1.00* und *Existenz von Sprünge* kann die grundlegende Funktionalität beider Modelle validiert werden, da sowohl der Fehler als auch die Sprünge im unauffällig definierten Bereich liegen. Bei direktem Vergleich beider Variationsmodelle kann darüber hinaus festgestellt werden, dass die Fehler des Modells g teilweise geringer ausfallen, als die von Modell h . Ein Beispiel hierfür ist die Animation *glücklich zu neutral* in der ersten Variation, für welche das Modell g einen Fehler von 0,10 und das Modell h einen Fehler von 0,15 aufweist. Bei der Betrachtung der Sprünge ist dagegen genau das Gegenteil zu beobachten. Die Sprünge bei Modell g betragen im Durchschnitt 6,22 % mit einem Maximalwert von 23,88 %, wohingegen die Sprünge bei h im Durchschnitt bei 5,25 % mit einem Maximalwert von 13,08 % liegen. Für die restlichen Animationen ist dieser gegensätzliche Zusammenhang jedoch nicht zu erkennen.

Eine weitere Auffälligkeit ist jedoch die deutliche Fehlerdifferenz bei der Animation *neutral zu glücklich* der fünften Variation, für welche der Fehler für Modell h mehr als doppelt so hoch ist wie der von Modell g . Beide Animationen sollen deshalb anhand der visuellen Darstellung in Abbildung 6.12 genauer untersucht werden.

Tabelle 6.8.: Absoluter Fehler für die Variationsmodelle inklusive des vollständigen Modells

Animation			Fehler		
von	zu	Variation	g	h	i
<i>glücklich</i>	<i>neutral</i>	1	0,10	0,15	0,12
		5	1,20	1,17	0,16
<i>glücklich</i>	<i>überrascht</i>	1	0,27	0,26	0,25
		5	0,97	0,89	0,27
<i>neutral</i>	<i>glücklich</i>	1	0,07	0,08	0,11
		5	0,88	1,97	0,07
<i>überrascht</i>	<i>glücklich</i>	1	0,08	0,75	0,58
		5	2,12	1,87	0,07
<i>überrascht</i>	<i>angewidert</i>	1	0,20	0,19	0,08
		5	0,18	0,32	0,07

6. Ergebnisse

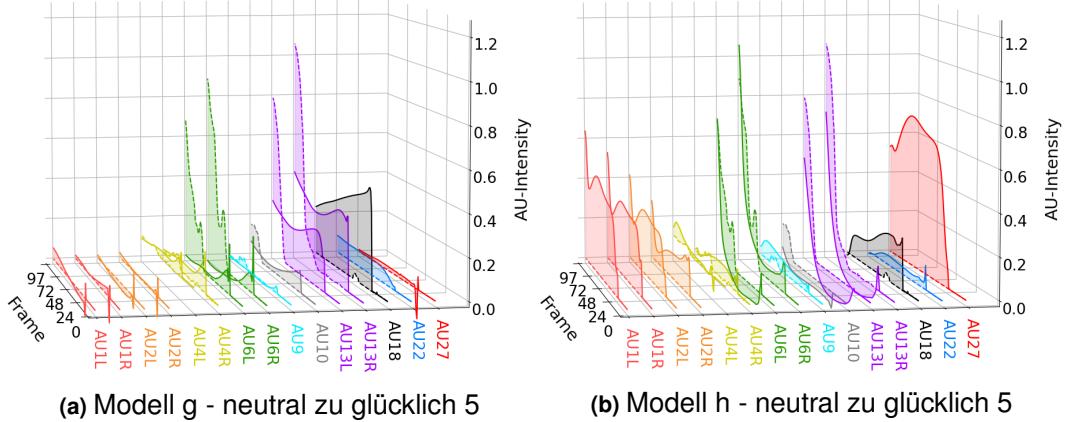


Abbildung 6.12.: Modell g und h - Visueller Vergleich überrascht zu glücklich

Anhand der Graphen von Modell *h* kann der erhöhte Fehlerwert auf die maßgeblichen Differenzen der eigentlich passiven AUs AU1L/R (rot-links), AU2L/R (orange) und AU27 (rot-rechts) zurückgeführt werden. Die Daten des Modells *g* sowie die Daten der Grundwahrheit (gestrichelte Linien) zeigen, dass diese AUs eigentlich kaum aktiv sind. Für die AUs AU6L/R (grün) und AU13L/R (lila), welche den Großteil des Ausdrucksübergangs definieren, sind die Differenzen zur Grundwahrheit für das Modell *h* deutlich geringer und variieren vorwiegend im Zeitpunkt (temporale Verschiebung). Bei der Betrachtung der dazugehörigen Avataranimation fallen die fälschlicherweise aktivierte AUs von Modell *h* jedoch wesentlich deutlicher ins Gewicht, als die nicht vollständig aktivierte AUs bei Modell *g*.

Kapitel 7

Diskussion

In diesem Kapitel werden die Umsetzung sowie Ergebnisse der zuvor angesprochenen Methoden diskutiert. Zunächst wird der Datensatz, welcher die Grundlage der Animationen darstellt, ausführlich besprochen. Anschließend werden die Ergebnisse des finalen Expression-Generators genauer betrachtet, bevor die Limitierungen des Systems aufgeführt werden. Zuletzt findet eine ausführliche Interpretation der Ergebnisse der quantitativen Evaluation statt.

7.1. Datensatz

Bevor die Performance des Expression-Generators ausführlich besprochen werden kann, gilt es die Trainingsdaten bzw. den Datensatz, welcher für das Training verwendet wurde, zu bewerten. Die Daten eines Trainingsdatensatz sind bei sämtlichen generativen KNN-Systemen maßgeblich an dem Ergebnis der Modelle beteiligt. Dies bedeutet generell, dass die generierten Daten solcher Systeme die Fehler bzw. Ungenauigkeiten jener Daten ungewollt übernehmen bzw. lernen. Die in Kapitel 5.1 angesprochene Vorverarbeitung bildet dabei die erste Hürde für das entwickelte KNN-System: *Expression-Generator*. Bei dieser Hürde handelt es sich zum einen um die künstliche Erweiterung des Datensatzes durch die Verwendung unterschiedlicher Interpolationsverfahren und zum anderen um die Zusammensetzung des Datensatzes.

Das Problem bei der Interpolation entsteht durch die unterschiedliche Anzahl an fehlenden Daten der einzelnen Animationen. Für den Fall, dass bei der dreifachen Aufnahme lediglich ein geringer Bruchteil an Frames nicht verarbeitet wurden,

stimmen die Daten der unterschiedlichen Interpolationsverfahren größtenteils überein, was zu folge hat, dass die Animationsdaten im Grunde dreimal im Datensatz vorhanden sind. Dies kann bei einem unausgeglichenen Datensatz dazu führen, dass weniger vorhandene Animationen schlichtweg ignoriert werden, d.h. beim Versuch diese Animationen zu generieren, werden lediglich Daten häufigerer Animationen verwendet, da der gesamte Fehler durch die Übereinstimmung mit den häufig vorhandenen Daten trotz schlechter Ergebnisse bei den seltenen Animationen gering gehalten wird. Dieses Verhalten ist mithilfe der Interpretation der Animationen des Modells *a* aus Kapitel 6.3.2 sowie der dazugehörigen Trainingsdaten belegbar. Bei der Animation des Avatars unter der Verwendung jeglicher von- oder *zufällig*-Animationen kann festgestellt werden, dass sämtliche Animationen Bestandteile von *überrascht*-Animationen aufweisen. Konkret bedeutet dies, dass das KNN-System die Animationen, welche *überrascht* enthalten als maßgeblich beeinflussend ansieht. Zu erklären ist dies schlichtweg durch die Anzahl an Trainingsdaten, welche genau diesen Ausdruck beinhalten. In Tabelle 5.2 kann abgelesen werden, dass 25 von 35 Animationen, bzw. 75 von 105 nach der Verdreifachung der Daten, den Gesichtsausdruck *glücklich* beinhalten, was einer prozentualen Verteilung von 71,43 % entspricht. Daraus folgt, dass das System vorwiegend lernt, dass der Fehler durch den Einbezug der Animation *überrascht* geringer ausfällt, als unter der Verwendung der restlichen Animationen. Bereits bei der Betrachtung der Avataranimationen von Modell *b* kann beobachtet werden, dass die grundlegende Verwendung der *überrascht*-Animation ausbleibt. Der prozentuale Anteil von *überrascht*-Animationen beträgt hierbei zwar immer noch 55,56 %, jedoch scheint diese Abweichung bereits signifikant zu sein. Zwar steigt für dieses Modell der Fehler (vgl. Tabelle 6.6), jedoch lernt das Modell zwischen den Daten zu interpolieren und lernt nicht wie zuvor schlichtweg auswendig. Daraus lässt sich grundlegend ableiten, dass mehr Daten dem Auswendiglernen des Systems entgegenwirken. Dies ist auch der Grund, weshalb zu Beginn die Verdreifachung der Daten mittels Interpolationsverfahren eingeführt wurde. Ältere Versionen des Expression-Generators, welche lediglich mit dem einfachen Datensatz trainiert worden sind, wiesen einen deutlich größeren MSE-Fehler auf, als Modelle mit dem angepassten Datensatz, weshalb für das finale Modell lediglich mit dem dreifachen Datensatz trainiert worden ist.

Ein weiteres Problem ergibt sich aus der limitierten Anzahl der verwendeten AUs. Ursprünglich wurden neben den eingesetzten noch weitere AUs bei der Aufnahme

herangezogen. Die Intensitäten dieser AUs wurden jedoch aufgrund eines Kamerafehlers bei der Aufnahme stark verfälscht, weshalb diese für das Training nicht einbezogen werden konnten. Daraus folgt, dass das System aufgrund des mangelhaften Trainingsmaterials unter der Verwendung jener AUs ebenfalls mangelhafte Animationen generiert hätte. Bei den vorhandenen AUs handelt es sich dennoch größtenteils um deutlich sichtbare AUs wie beispielsweise AU27 - Mund öffnen - oder AU1, AU2 sowie AU4, welche für die Bewegung der Augenbrauen zuständig sind. Darüber hinaus bleibt zudem festzuhalten, dass die Anzahl an verschiedenen Ausdrucksübergängen, trotz der Nichtverwendung der Ausdrücke *wütend* und *ängstlich*, durchaus reich an Variation ist und das System dadurch in der Lage ist, bekannte sowie teilweise neue oder abgeänderte Variationen der Animationen zu generieren. Für zukünftige Untersuchungen sollten jedoch alle Ausdrucksübergänge in gleicher Quantität aufgenommen und für das Training eingesetzt werden. Weiterhin sollten die Trainingsdaten zukünftig von einem professionellen Schauspieler aufgenommen werden, um zu gewährleisten, dass die Ausdrücke richtig ausgeführt werden. Der Akteur, dessen Aufnahmen für diese Arbeit verwendet wurden, weist beispielsweise eine unterschiedlich starke Ausprägung der AU13 auf.

7.2. Expression-Generator

In den nachfolgenden Abschnitten werden vorwiegend die Ergebnisse von Modell *i* betrachtet und erläutert. Insbesondere wird dabei auf das Training des Modells, das finale Design der Modells sowie zuletzt auf die generierten Animationen durch das Modell eingegangen.

7.2.1. Training des Modells

Bei Betrachtung der Fehlerwerte des Trainingsprozesses fällt direkt der geringe Fehlerwert im Dezimalbereich auf. Bei näherer Untersuchung der Fehlerberechnung kann dieser geringe Wert jedoch zum einen auf das Quadrieren in Formel 5.7 zurückgeführt werden und zum anderen auf die Null-Komma-Zahlen, welche bei der Subtraktion zweier AU-Intensitäten entstehen, die durch das anschließende Quadrieren noch kleiner werden. Weiterhin muss beachtet werden, dass sich der Fehler auf den Durchschnitt aller Frames sowie aller einzelnen AUs bezieht.

Ein weiterer Punkt, welcher bezüglich des Trainings angesprochen werden muss, sind die explodierenden bzw. schwindenden Gradienten, welche während des Trainingprozesses teilweise beobachtet werden konnten. Beim Training von KNNs unter Verwendung des Optimierers *Adam*, kann es gelegentlich zu Divisionen durch Werte nahe null kommen, sodass die aus der Berechnung resultierenden Gradienten stark erhöht werden. Dieses Problem lässt sich eigentlich mithilfe der im Pytorch-Framework eingebauten Funktion *grad_clip()* beseitigen, jedoch führte der Einsatz dieser Funktion zu weiteren Problemen beim Training unter der Verwendung einer Grafikkarte, sodass für das Training des Expression-Generators ein *Trail-and-Error-Ansatz*, wie er in Abschnitt 5.3.2 erläutert wird, verwendet wurde. Explizit bedeutet dieses Vorgehen, dass die Variablen des KNN nach jeder Epoche gespeichert werden müssen, damit für den Fall, dass die Gradienten explodieren, das Training abgebrochen und mit den zuletzt gespeicherten Variablen von Neuem begonnen werden kann. Die Umstellung auf diese Methode bringt jedoch einen erheblichen Mehraufwand mit sich, was die Trainingsdauer von ca. acht Stunden ohne Interventionen noch einmal um mehrere Minuten bis Stunden erhöhen kann. Die Häufigkeit des Aufkommen von explodierenden Gradienten konnte zwar durch eine Verkleinerung der Lernrate verringert werden, jedoch bringt diese ebenfalls eine erhöhte Trainingsdauer mit sich, sodass in dieser Arbeit ein Mittelweg gefunden wurde. Dieser besteht darin die Trainingsrate nach jeweils 100 Epochen um den Faktor 0,1 zu verringern. Grundsätzlich ist der *Adam* Optimierer zwar so entworfen, dass eine solche Anpassung der Lernrate nicht notwendig wäre, jedoch konnte der Fehler dadurch effizienter verringert werden, sodass diese Methode für zukünftige Versuche beibehalten werden sollte.

7.2.2. Design des Modells

Das Design des Expression-Generators ist der Aspekt dieser Arbeit, welcher sich während der Erstellung am meisten verändert hat. In älteren Versionen wurde versucht das Modell gleich wie das System aus [11] aufzubauen. Das so entwickelte System wurde dabei mit dem gleichen Datensatz trainiert wie das finale Modell in Abbildung 5.2. Die Ergebnisse, beziehungsweise die generierten Animationen eines solchen trainierten Modells, ignorierten jedoch entweder den Start- oder den Endframe der Animation, d.h. wenn die Animation *überrascht zu neutral* generiert werden sollte, erzeugte das Modell die Animation *überrascht zu glücklich* mit einem

eindeutigen Sprung von *glücklich* zu *neutral* beim letzten Frame der Animation. Eine Ursache dieses Problems bei diesem Modell waren die gleichen Dimensionen von Start- und Endframe nach Berechnungen eines linearen Encodings (vgl. [11]). Diese Codierungen wurden anschließend direkt als Input für den in 5.2.5 beschriebenen Recurrent-Generator verwendet. Im direkten Vergleich dazu entsteht durch die Berechnung eines Vergangenheits-Encodings aus dem Vergangenheitsvektor x , beschrieben in 5.2.1, ein deutliches Ungleichgewicht zwischen dem variablen Start- bzw. dem aktuellen Frame und dem konstanten Endframe. Die Vermutung liegt nahe, dass im Laufe des Trainings der konstante Start-Frame einen zu großen Einfluss auf die generierten Animationen nimmt. Gleichzeitig waren auch die MSE-Fehlerwerte dieses Modells deutlich höher als die des finalen Expression-Generators. Der geringste Testfehler bezogen auf das vorherige, ältere Modells beträgt ca. 0,0262, wohingegen der Expression-Generator einen Testfehler von 0,0014 aufweist (vgl. Kapitel 6.1), was eine fundamentale Verringerung des Fehlers darstellt, welche wie bereits erwähnt auch anhand der Avataranimationen festgestellt werden kann.

Die einzelnen Dimensionen des finalen Expression-Generators wurde während dieser Arbeit jedoch nicht variiert, sodass durch die Abänderung einzelner Teil-Module durchaus Verbesserungen möglich sind. Auch die Verwendung eines mehrschichtigen Decoder-Moduls, wie er in den vergleichbaren Arbeiten [11] und [13] verwendet wird, ist daher denkbar.

7.2.3. Generierte Animationen

Anhand der Tabellen 6.6 und C.9 kann nach den Kriterien 1. - *Absolute Differenz Grundwahrheit und generierter Animation > 1.00* und 2. - *Existenz von Sprüngen* (vgl. 6.3.1) sowohl die grundlegende Funktionalität als auch die Funktionalität für die vier weiteren Animationen, welche für die numerische Analyse in Kapitel 6.3.2 generiert worden sind, bestätigt werden. Bei der Betrachtung der dazugehörigen Avataranimationen konnten hierbei ebenfalls keine merkbaren Auffälligkeiten festgestellt werden. Weiterhin können durch die Ergebnisse aus Kapitel 6.2.1 teilweise auch Variationen der Animationen bestätigt werden. Insbesondere durch die Erkenntnisse letzterer Animationen kann festgehalten werden, dass das System in der Lage ist, aus bekannten Start- und Endframes neue Animationen (wie in Abschnitt 4.4 definiert) zu erzeugen. Bei weiteren Versuchen mit Variationsdaten konnten zu-

Tabelle 7.1.: Bewertung neue Animationen von Modell i

Start \ Ziel	angewidert	stirnrunzelnd	glücklich	neutral	traurig	überrascht
angewidert	+	+	+	o	-	
stirnrunzelnd	-	o	-	-	o	
glücklich	+	+		+	+	+
neutral	-	+	+		-	+
traurig	o	-	o	-		-
überrascht	+	+	+	+	+	

dem weitere Erkenntnisse gewonnen werden, welche im Folgenden kurz vorgestellt und kritisch betrachtet werden.

Die Bewertungen dieser Animationen können Tabelle 7.1 entnommen werden, wobei ein + für eine visuell unauffällige Animation steht, ein o für eine Animation, welche nur geringe Auffälligkeiten aufweist und zuletzt - für visuell auffällige Animationen. Fettgedruckte Symbole stehen hierbei für Animationen, welche abgesehen vom Einsatz von Variationsdaten im Datensatz enthalten sind (vgl. Tabelle 4.2). Aus Tabelle 4.2 folgt außerdem, dass 60 % der möglichen Animationen (Ausdrucksübergänge) im Datensatz enthalten sind. Von diesen 60 % sind lediglich drei Animationen als eindeutig visuell auffällig bewertet worden, was einem Prozentwert von 16,67 % entspricht. Ferner kann beobachtet werden, dass keine Animation als *visuell auffällig* bewertet wurde, welche den von oder zu *glücklich* Gesichtsausdruck enthält. In diesen Fällen ist das KNN in der Lage vorwiegend visuell unauffällige Animationen zu generieren.

Dieses Ergebnis lässt sich durch das Vorhandensein jeglicher Ausdrucksübergänge von und zu *glücklich* im Datensatz erklären. In diesen Fällen hat das Netz gelernt von und zu diesen Ausdrücken, unter der Beachtung der spezifischen Dynamiken innerhalb des Ausdrucksübergangs, zu interpolieren. Für diese Bewertung wurden dabei bewusst Variationsdaten verwendet, da der Einsatz dieser vielmehr die reale Umgebung eines solchen Systems beschreibt. Durch die soeben besprochenen Ergebnisse lassen sich daher die folgenden Einsatzmöglichkeiten beschreiben, welche aufgrund der Eigenschaften des entwickelten Systems gleichzeitig zwei distinktive Vorteile gegenüber den klassischen Methoden mit sich bringt:

1. Vereinfachte Generierung von Ausdrucksübergängen:

Bei der klassischen Erzeugung von Ausdrucksübergängen werden wie in Kapitel 2.2 entweder viel Zeit für die manuelle Erstellung von Keyframes für

Keyframeanimationen benötigt, oder es müssen Schauspieler und teure Motion Capture-Systeme für die Übertragung von Keyframes verwendet werden. Zwar werden bei klassischen Keyframe-Animationen auch Frames zwischen zwei oder mehreren Keyframes interpoliert, jedoch handelt es sich bei einer solche Interpolation nur um das Ausfüllen weniger Frames bzw. der Animation von neutralem Gesichtsausdruck zu einer konkreten Emotion. Dies liegt vorwiegend daran, dass bei einem Ausdrucksübergang von einem Ausdruck direkt zum nächsten Ausdruck die AU-Intensitäten bzw. die Blendshape-Intensitäten keine eindeutige Linearität aufweisen. Diese Nichtlinearität kann beispielsweise mithilfe der Grundwahrheitsdaten in den Abbildungen 6.5 beobachtet werden. Daraus folgt, dass, sofern keine eindeutigen Zusammenhänge von Ausdruck zu Ausdruck ermittelt werden können, jede AU bzw. Blendshape einzeln bearbeitet werden muss. Im Vergleich dazu kann der trainierte Expression-Generator zwischen zwei beliebigen Frames verschiedener Ausdrücke interpolieren. Diese Methode ist daher nicht nur schneller, sondern im Endeffekt auch kostengünstiger bezogen sowohl auf die Arbeitszeit als auch auf die Rechenleistung. Ein solches KNN-System könnte beispielsweise online in Videospielen eingesetzt werden, um dynamische Animationen zu generieren, welche durch das Verhalten des Spieler beeinflusst werden, ohne das diese zuvor manuell programmiert bzw. generiert werden müssen. Weiterhin besteht durch die Natur der Daten ein direkter FACS-Bezug, weshalb sich auf diese Art und Weise generierten Daten auch für Forschungszwecke eignen. Dies resultiert aus der Tatsache, dass sich die in dieser Arbeit generierten Daten ausschließlich auf FACS-definierte AUs beziehen, sodass lediglich die Intensitäten übersetzt werden müssten (vgl. Abschnitt 1.1).

2. Erweiterbarkeit des Modells:

Aktuell gibt es bei der Generierung von neuen Animationen nur limitierte Möglichkeiten der Anpassung, d.h. bis auf Start- und Endframe kann der Nutzer keinen weiteren Einfluss auf das Generierte nehmen. Durch diese Limitierung kann zwar die Einfachheit der Systems gewährleistet werden, jedoch nur auf Kosten der Funktionalität. Eine mögliche Erweiterung der aktuellen Funktionalität könnte durch den Einsatz von komplexeren Methoden wie VAEs oder GANs nachträglich hinzugefügt werden. Ein VAE könnte beispielsweise in der Lage sein abhängig vom Regularisierungsparameter ver-

schiedene Variationen (bspw. Geschwindigkeit) einer Animation zu generieren, wohingegen GANs aufgrund ihrer vielseitigen Eigenschaften zahlreiche Erweiterungsmöglichkeiten mit sich brächten.

7.3. Limitierungen des Systems

Eine Limitierung des Systems, die bereits mehrfach angesprochen wurde, ist die deutliche Verbindung aus vorhandenen Daten und generierten Animationen. Wie bei den meisten KNN-Systemen werden für die Entwicklung eines neuen KNN-Systems eine Vielzahl von Daten benötigt. Weiterhin gilt für generative Modelle für Animationen der bereits angesprochene Zusammenhang zwischen der Qualität der Trainingsdaten und der Qualität der neuen Animationen (vgl. 7.1). Der Expression-Generator bildet hierbei keine Ausnahme. Die Übernahme von fehlerhaften Eigenschaften kann bestmöglich am Beispiel der unterschiedlich starken Intensitäten der AU2 (orange) in Abbildung 6.2a verdeutlicht werden. Die vom Akteur unterschiedlichen starken Veränderungen der linken und rechten Seiten werden so auch vom Expression-Generator gelernt und wiedergegeben (vgl. 6.2b). Daraus folgt, dass mit einem gelernten System zwar qualitative Daten generiert werden können, jedoch werden hierfür zunächst genau solche Daten benötigt. Dennoch bietet der Expression-Generator einen Mehrwert, da dieser es ermöglicht diese Daten in verschiedenen, leicht abgeänderten Varianten zu vervielfachen, um so letztendlich eine stetig wachsende Datenbank an Ausdrucksübergängen zu popularisieren. Neben der zuvor angesprochenen Datenqualität der Trainingsdaten müsste hierfür jedoch auch das Problem der Sprünge behoben werden, da selbst das finale Modell einen Durchschnittswert von 5,52 % Abweichung zwischen Frame 19 und Frame 20 aufweist (C.9). Zum Vergleich: Die Sprünge bei den Daten der Grundwahrheit betragen durchschnittlich gerade einmal 0,13 % (C.10). Zur Behebung dieser Problematik gibt es prinzipiell zwei Herangehensweisen, welche ähnlich u.a. auch in [13] zu finden sind: (1) *Nachbearbeitung der Animationen* oder (2) *Hidden-State Initialisierung*.

(1) *Nachbearbeitung der Animationen*:

Mithilfe einer Nachbereitung der generierten Animationen könnten die Sprünge auf eine einfache, jedoch effektive Art und Weise entfernt werden, ohne dass ausdrucksdefinierende Charakteristiken verloren gehen. Die vorgeschla-

gene Methode basiert auf klassischer Interpolation wie sie in 2.2.1 erläutert wurde. Während der Nachbearbeitung könnten die ersten 25 Frames, d.h. bei der aktuellen Architektur die ersten fünf generierten Frames, der Animation mittels klassischer Interpolation ersetzt werden, indem zwischen dem ersten und 26. Frame interpoliert wird. Dies wäre prinzipiell möglich, da sich die Sprünge zumeist auf nur einzelne Frames direkt zu Beginn der generierten Animation auswirken. Eine ähnliche Methode wird in [13] für die Übergänge zwischen den generierten Frames und dem tatsächlichen Zielframe genutzt, um auch hier die Differenzen zwischen beiden zu überbrücken.

(2) Hidden-State Initialisierung:

Eine weitere Möglichkeit die Sprünge zu minimieren, könnte eine alternative Hidden-State Initialisierung darstellen. Aktuell werden die Hidden-States des *LSTM-Encoders* sowie des *Recurrent-Generators* null-initialisiert, wodurch die gelernten Parameter dieser Module für alle Daten zunächst gleich sind und ausgeglichen werden müssen [13]. In [13] wird für die Initialisierung der Hidden-States ein MLP trainiert, dessen Outputs die unterschiedlichen Hidden-States konstruiert und somit den Rekonstruktionsfehler der ersten Frames reduziert.

Neben den soeben genannten Möglichkeiten wäre es zudem möglich den Fehlern zu Beginn als auch gegen Ende der Animation, folglich bei den Übergängen von Startframe zur generierten Animation sowie von generierter Animation zu Endframe, zusätzliche Relevanz hinzuzufügen. Beispielsweise könnte durch eine Multiplikation mit einem zusätzlichen Hyperparameter dieser Fehler verstärkt werden, sodass das System explizit lernt die Differenzen zu verringern.

7.4. Quantitative Evaluation

Durch die in Abschnitt 6.3.2 vorgestellten Ergebnisse der numerischen Analyse kann abgeleitet werden, wie genau das System Expression-Generator aus den verfügbaren Daten lernt, wie sich die Anzahl sowie Verteilung der Trainingsdaten auf die Resultate auswirken und was zukünftig beachtet werden sollte. Die dazugehörigen Avataranimation können Anhang B entnommen werden.

Symmetrische Modelle a bis e

In Abschnitt 6.3.2 werden die Modelle *b*, *c* und *d* alle aufgrund des zweiten Kriteriums (Existenz von Sprüngen) als ungeeignet klassifiziert, dennoch kann mithilfe von Abbildung 6.11 ein grundlegendes Verständnis für die Animation und damit für die gewünschte Funktionalität bewiesen werden. Hierbei ist das Modell *b* bereits in der Lage zwischen einem vorhandenen und einem nur in Verbindung mit einem anderen Ausdrucksübergang (*traurig zu glücklich*) unbekannten Ausdruck zu interpolieren. Bei einer solchen Interpolation handelt es sich jedoch um eine weitaus komplexere Art von Interpolation als durch Formel 5.1 oder 5.2 gegeben und wird dabei durch die grundlegende Funktionalität von AE-Architekturen realisiert. Eine solche Funktionalität konnte auch bei weiteren Modellen für Ausdrucksübergänge von und zu *überrascht* beobachtet werden. Dies lässt sich durch die im Vergleich zu den anderen Ausdrücken erhöhte Anzahl an *überrascht*-Ausdrucksübergängen im Datensatz erklären.

Weiterhin kann den Ergebnissen der Modelle *d* und *e* ebenfalls der Zusammenhang zwischen *Vorhandensein der Animation im Trainingsdatensatz* und *Qualität der generierten Animation* erkannt werden. Dies ist eindeutig an den deutlich geringeren Fehlerwerten der Animation *glücklich zu überrascht* sowie der umgekehrten Variante erkennbar, welche bei Modell *c* durchschnittliche bei 3,15 liegt, wohingegen der Fehlerwert bei Modell *d* bei durchschnittlich 0,44 und bei Modell *e* 0,25 beträgt. Gleicher Verhalten lässt sich demnach auch bei der Animation *glücklich zu neutral* sowie auch dieser umgekehrten Variante zwischen den Modellen *d* und *e* erkennen, bei welchen der Fehlerwert ebenfalls deutlich abnimmt.

Zuletzt kann bei der Betrachtung der Sprünge für diese Modelle ein Zusammenhang zwischen Datenquantität und Intensität der Sprünge erkannt werden. Die durchschnittliche Intensität der Sprünge sinkt für eine erhöhten Anzahl an Datenbeispielen im Trainingsdatensatz. Dieser Verlauf kann anhand der Sprünge für die vorhandenen symmetrischen Modelle (mit Ausnahme von Modell *d*) eindeutig beobachtet werden. Die durchschnittliche Intensität der Sprünge weist dabei eine stark negative Korrelation von -0,92 mit der Anzahl an Daten (vgl. Tabelle 5.3) auf.

Aus den Ergebnissen der symmetrischen Modelle lassen sich somit zwei Schlussfolgerungen ziehen:

1. Das System kann bei ausreichend vorhandenen Daten auch unbekannte Animationen aus bekannten Daten ableiten¹.
2. Die Genauigkeit der generierten Animation ist vom Vorhandensein der Animation im Trainingsdatensatz abhängig.
3. Die Intensität der Sprünge nimmt mit zunehmenden Datenbeispielen ab.

Asymmetrisches Modell f

Das Modell f wurde mit der Idee, ob es ausreicht das System mit nur einer Richtung von Ausdrucksübergängen zu trainieren, entwickelt. Die Ergebnisse in 6.7 beweisen eindeutig, dass das System hierzu nicht in der Lage ist. Die Fehlerwerte der im Datensatz enthaltenen *von-glücklich*-Animationen sind deutlich geringer als die Fehlerwerte der umgekehrten Varianten der *zu-glücklich*-Animationen. Aus den Ergebnissen des asymmetrischen Modells lässt sich daher die folgende Schlussfolgerung ableiten:

4. Es reicht nicht aus das System mit ausschließlich *von-* oder *zu-Übergängen* zu trainieren.

Variationsmodelle g und h

Die Modelle g und h weisen die Besonderheit auf, dass sich diese lediglich in der Anzahl der von- und zu-*glücklich*-Animationen unterscheiden. Mithilfe dieser Modelle soll überprüft werden, wie sich die Anzahl verschiedener Animationsvarianten auf die Performance des Modells auswirkt.

Anhand der Ergebnisse aus Tabelle 6.8 können jedoch auf den ersten Blick keine eindeutigen Ergebnisse festgestellt werden. Teilweise sind die Fehlerwerte für das Modell h niedriger (*glücklich zu überrascht*), jedoch ist auch das Gegenteil zu beobachten (*neutral zu glücklich*). Aus den unterschiedlichen Ergebnissen der ersten und fünften Variationen lässt sich dennoch ein verstärktes Auswendiglernen von Modell g ableiten. D.h. das Modell g kann aufgrund der geringeren Variation innerhalb eines Ausdrucksübergangs bessere Animation für genau diese Variation generieren, eignet sich jedoch schlechter für datensatzfremde Animationen. Da letzteres jedoch

¹Aufgrund des Vorhandenseins von Sprüngen müssen die Daten jedoch zuvor manuell nachbearbeitet werden.

dem realen Anwendungsfall eines solchen Systems deutlich näher kommt, sollten stets mehrere Variationen einer Animation für das Training verwendet werden.

Eine weitere Erkenntnis kann bei genauerer Betrachtung der einzelnen Sprünge in Tabelle C.8 beobachtet werden. Diese Erkenntnis bezieht sich auf den in Abschnitt 7.4 angesprochenen Zusammenhang zwischen *Intensität der Sprünge* und *Anzahl Datenbeispielen*, welcher in dieser Form nicht wahrgenommen werden kann. Die durchschnittliche Abweichung bei Modell g beträgt hierbei 7,86 %, wohingegen dieser Wert für Modell h bei 7,93 % liegt. Stattdessen kann jedoch eine ausgeglichene Verteilung der Sprung-Intensitäten festgestellt werden. Dies liegt vorwiegend an der Verringerung größerer Abweichungen. Diese Veränderung kann dabei ebenfalls auf die zuvor angesprochene Tendenz zum Auswendiglernen von Modell g zurückgeführt werden, da dieses Modell bei bekannten Daten geringere Differenzen aufweist als das Modell h .

Die Ergebnisse der Variationsmodelle resultieren daher in der folgenden Erkenntnis:

5. Eine erhöhte Anzahl an Variationen eines Ausdrucksübergangs wirkt dem Auswendiglernen des Systems entgegen.

Insgesamt ermöglichen die Ergebnisse und Schlussfolgerungen der quantitativen Evaluation ein tieferes Verständnis für die Funktionsweise des in dieser Arbeit entwickelten Systems und bilden eine fundierte Grundlage in den Zusammenhängen zwischen dem Aufbau des Datensatzes und den generierten Animationen. Hierbei ist nicht nur die Anzahl an verschiedenen Ausdrucksübergängen relevant, sondern auch die Variation, d.h. die Anzahl an verschiedenen Ausführungen eines Ausdrucksübergangs, von wichtiger Bedeutung. Bei einer ausreichenden Datenvielfalt ist es zudem grundsätzlich möglich unbekannte Animationen aus vorhandenen Daten zu generieren, jedoch eignet sich das System in seiner aktuellen Form primär zur Vervielfältigung von vorhandenen Daten.

Kapitel 8

Fazit

Das in dieser Arbeit entwickelte KNN-System *Expression-Generator* für die Erstellung von Gesichtsanimationen mit FACS-Bezug ist in der Lage aus lediglich zwei Frames verschiedener Gesichtsausdrücke Daten für eine vollständige Animation von einem Ausdruck zu einem anderen zu generieren. Vor allem bei Übergängen, welche denen der Trainingsdaten ähneln, erzielt das System repräsentative Ergebnisse. Zudem konnte innerhalb einer Versuchsreihe festgestellt werden, dass das System auch in der Lage ist zwischen unbekannten Frames zweier Ausdrücke zu interpolieren. Realisiert wird eine solche Interpolation durch eine angepasste *LSTM-Autoencoder*-Architektur wie sie im Bereich der Ganzkörpersynthese Anwendung findet. Die wesentliche Anpassung wird durch die Einführung eines *Vergangenheitssequenz-Vektors* realisiert, welcher den Inhalt mittels eines *many-to-one-LSTM-Encoders* in ein *Vergangenheitsencoding* codiert. Dieser Vektor, bestehend aus 20 vergangenen Frames der Animation, bildet zusammen mit dem *Ziel-Frame* der Animation - dem Zielausdruck - den Input eines *Recurrent-Generators*, welcher für die Modellierung von temporalen Dynamiken der Sequenz zuständig ist. Beim Recurrent-Generator handelt es sich um ein *many-to-many-LSTM*, dessen Inputs mittels eines *Linearen Decoders* in den nächsten Frame der Animation decodiert werden.

Mithilfe der Daten der quantitativen Evaluation können Zusammenhänge zwischen dem Aufbau des Trainingdatensatzes und den generierten Animationen festgehalten werden. Die Ergebnisse belegen, dass das System zum einen durch eine gleichmäßige Verteilung der Ausdrucksübergänge und zum anderen durch mehrere Varianten eines solchen Ausdrucksübergangs bessere Ergebnisse erzielt und damit unauffälligere Animationen generiert.

Ein wesentliches Problem bei der Generierung von Animationen durch den Expression-Generator ist die Existenz von *Springen* nach dem 19. Frame der Animation, bei welchen die AU-Intensitäten eine ersichtliche Veränderung aufweisen. Die Intensität dieser Sprünge nimmt zwar mit zunehmenden Datenbeispielen ab, konnte im Rahmen dieser Arbeit jedoch nicht vollständig beseitigt werden. Eine Lösung dieses Problems wurde in Form einer *Nachbearbeitung der Animation* oder durch eine *Hidden-State Initialisierung* vorgeschlagen. Bei der Nachbearbeitung könnten die Sprünge mittels klassischer Interpolationsverfahren entfernt werden, wohingegen bei der Hidden-State Initialisierung ein zusätzlicher MLP für die Initialisierung der Hidden-States des LSTMs, mit dem Ziel den Rekonstruktionsfehler weiter zu reduzieren, trainiert werden könnte.

In weiteren Forschungsarbeiten sollten für das Training des Systems daher zuvor entschieden werden, wie den Sprüngen entgegengewirkt werden soll, da für den Einsatz des zuvor angesprochenen MLPs das System zunächst modifiziert werden müsste. Weiterhin sollten die Empfehlungen bezüglich des Datensatzes bei der Aufnahme von Motion Capture-Sequenzen berücksichtigt werden. In dieser Arbeit konnten aufgrund eines Kamerafehlers bei diesen Aufnahmen nur wenige AUs verarbeitet werden. Das während des Verarbeitungsprozesses der Motion Capture-Aufnahmen auftretende Auslassen von Frames konnte zwar mithilfe eines Vorverarbeitungsschrittes ausgeglichen werden, sollte für zukünftige Arbeiten jedoch gänzlich behoben werden. Gleichzeitig sollte die künstliche Vervielfachung des Datensatzes mittels unterschiedlicher Interpolationsverfahren nach Möglichkeit vermieden werden, indem grundsätzlich mehr Ausdrucksübergänge aufgenommen und verarbeitet werden. Ferner wird empfohlen die generierten Daten und somit das KNN-System per Nutzerumfrage weiter zu validieren. Vermutlich würde es genügen Unterschiede zwischen Grundwahrheitsdaten und den generierten Animation zu untersuchen.

Besonders hervorzuheben ist die Möglichkeit mithilfe des Expression-Generators FACS-annotierte Datensätze erweitern zu können, um so beispielsweise die Analyse von Gesichtsausdrücken zur frühzeitigen Erkennung von Depressionen zu unterstützen. Aber auch in der Unterhaltungsbranche könnte ein solches System angewandt werden, um die Gesichter virtueller Avatare zu animieren, ohne das sämtliche Ausdrucksübergänge zuvor von Hand generiert werden müssten. Darüber hinaus ermöglicht der simple Aufbau des Systems die Erweiterung um komplexere Architekturen wie *VAEs* oder *GANs*, welche weitere Funktionalitäten einführen könnten. Ob

8. Fazit

diese Erweiterungen die gewünschten Resultate mit sich brächten, müsste jedoch zunächst analysiert werden.

Abkürzungsverzeichnis

3DMM	3D Morphable Model
3D	dreidimensional
AE	Autoencoder
AU	Action Unit
CNN	Convolutional Neural Network
FACS	Facial Action Coding System
GRU	Gated Recurrent Units
GAN	Generative Adversarial Networks
GT	Groundtruth
KI	Künstliche Intelligenz
KNN	Künstliches Neuronales Netz
LSTM	Long Short Term Memory
MLP	Multi-Layer-Perceptron
MSE	Mean Squared Error
RNN	Recurrent Neural Network
VAE	Variational Autoencoder

Tabellenverzeichnis

4.1. Beispiel für das Datenformat eines Ausdrucksübergangs	23
4.2. Ausdrucksübergänge im Datensatz	24
5.1. In dieser Arbeit verwendete Python-Pakete	34
5.2. Datensatzgrundlage für die Evaluation	43
5.3. Zusammensetzung der Datensätze für die numerische Evaluation . .	44
6.1. Erster generierter Frame für bekannte Daten angewidert zu glücklich	48
6.2. Erster generierter Frame für Variationsdaten angewidert zu glücklich	49
6.3. Erster generierter Frame für Variationsdaten stirnrunzelnd zu neutral	51
6.4. Erster generierter Frame für datensatzfremde Animation überrascht zu angewidert	54
6.5. Erster generierter Frame für datensatzfremde Animation überrascht zu traurig	55
6.6. Absoluter Fehler für die symmetrischen Modelle inklusive des voll- ständigen Modells	57
6.7. Absoluter Fehler für das asymmetrische Modell inklusive des voll- ständigen Modells	59
6.8. Absoluter Fehler für die Variationsmodelle inklusive des vollstän- digen Modells	60
7.1. Bewertung neue Animationen von Modell i	67
C.1. Modell a - Sprünge	xvii
C.2. Modell b - Sprünge	xvii
C.3. Modell c - Sprünge	xviii
C.4. Modell d - Sprünge	xviii
C.5. Modell e - Sprünge	xix
C.6. Modell f - Sprünge	xix
C.7. Modell g - Sprünge	xx
C.8. Modell h - Sprünge	xx
C.9. Modell i (Expression-Generator) - Sprünge	xxi
C.10. Grundwahrheit - Sprünge	xxi

Abbildungsverzeichnis

2.1. Schematische Darstellung von RNNs	9
2.2. Vergleich von RNNs und LSTMs	10
2.3. Aktivierung der Blandshape Mund öffnen	13
3.1. Schematische Darstellung des Encoder-Recurrent-Decoder-Modells	19
4.1. Visualisierung der oberen AUs mittels eines Avatars	21
4.2. Visualisierung der unteren AUs mittels eines Avatars	22
4.3. Vorstellung der Animation neutral zu überrascht anhand des Avatars	22
4.4. Ablaufdiagramm für die Datengewinnung	25
4.5. Visualisierung der in dieser Arbeit verwendeten Ausdrücke mithilfe eines Avatars	26
5.1. Ablaufdiagramm für zwei Vorverarbeitungsschritte	31
5.2. Schematische Darstellung des Expression-Generators	33
5.3. Schematische Darstellung des Vergangenheitssequenz-Vektors x . .	35
5.4. Ablaufdiagramm Trainingsprozess	41
6.1. Trainingsfehler Modell i	46
6.2. Modell i und GT - Vergleich für die Animation angewidert zu glücklich	48
6.3. Modell i und GT - Vergleich für die Animation angewidert zu glücklich mit Variationsdaten	49
6.4. Modell i und GT - Visualisierung der Abweichungen der generierten Animationen für angewidert zu glücklich	50
6.5. Modell i und GT - Visualisierung der Differenz zwischen GT und Modell i der beiden vorgestellten generierten Animationen für angewidert zu glücklich	50
6.6. Modell i und GT - Vergleich für die Animation stirnunzelnd zu neutral mit Variationsdaten	51
6.7. Modell i und GT - Visualisierung der Unterschiede für die Animation stirnunzelnd zu neutral mit Variationsdaten	52
6.8. Modell i - Datensatzfremde Animation überrascht zu stirnunzelnd .	53
6.9. Modell i - Datensatzfremde Animation überrascht zu traurig	54

Abbildungsverzeichnis

6.10. Modell b und i - Visueller Vergleich glücklich zu neutral in zwei Variationen	58
6.11. Modell b und i - Visueller Vergleich überrascht zu glücklich	59
6.12. Modell g und h - Visueller Vergleich überrascht zu glücklich	61

Literatur

- [1] M. Zollhöfer, J. Thies, P. Garrido, D. Bradley, T. Beeler, P. Pérez, M. Stammering, M. Nießner und C. Theobalt, „State of the Art on Monocular 3D Face Reconstruction, Tracking, and Applications“, *Computer Graphics Forum (Eurographics State of the Art Reports 2018)*, Jg. 37, Nr. 2, 2018.
- [2] P. Ekman und D. Keltner, „Universal facial expressions of emotion“, *Segestrale U, P. Molnar P, eds. Nonverbal communication: Where nature meets culture*, S. 27–46, 1997.
- [3] P. Ekman, „Darwin, deception, and facial expression“, *Annals of the New York Academy of Sciences*, Jg. 1000, Nr. 1, S. 205–221, 2003. DOI: <https://doi.org/10.1196/annals.1280.010>.
- [4] S. Du, Y. Tao und A. M. Martinez, „Compound facial expressions of emotion“, *Proceedings of the National Academy of Sciences*, Jg. 111, Nr. 15, E1454–E1462, 2014. DOI: 10.1073/pnas.1322355111. eprint: [https://www.pnas.org/content/111/15/E1454](https://www.pnas.org/content/111/15/E1454.full.pdf). Adresse: <https://www.pnas.org/content/111/15/E1454>.
- [5] B. Egger, W. A. P. Smith, A. Tewari, S. Wuhrer, M. Zollhoefer, T. Beeler, F. Bernard, T. Bolkart, A. Kortylewski, S. Romdhani, C. Theobalt, V. Blanz und T. Vetter, „3d morphable face models - past, present and future“, *ACM Transactions on Graphics*, Jg. 39, Nr. 5, Aug. 2020.
- [6] Z. Liu, G. Song, J. Cai, T.-J. Cham und J. Zhang, „Conditional adversarial synthesis of 3d facial action units“, *Neurocomputing*, Jg. 355, 200–208, 2019. DOI: 10.1016/j.neucom.2019.05.003. Adresse: <http://dx.doi.org/10.1016/j.neucom.2019.05.003>.
- [7] P. Ekman und W. V. Friesen, *Facial Action Coding System (FACS)*. Consulting Psychologists Press, 1978.
- [8] G. Donato, J. Hager, P. Ekman und T. Sejnowski, „Classifying facial actions“, *IEEE transactions on pattern analysis and machine intelligence*, Jg. 21, S. 974, Okt. 1999. DOI: 10.1109/34.799905.

- [9] S. Li und W. Deng, „Deep facial expression recognition: A survey“, *IEEE Transactions on Affective Computing*, 1–1, 2020. DOI: 10.1109/taffc.2020.2981446. Adresse: <http://dx.doi.org/10.1109/TAFFC.2020.2981446>.
- [10] B. Martinez, M. Valstar, B. Jiang und M. Pantic, „Automatic analysis of facial actions: A survey“, *IEEE Transactions on Affective Computing*, Jg. PP, S. 1–1, Juli 2017. DOI: 10.1109/TAFFC.2017.2731763.
- [11] K. Fragkiadaki, S. Levine, P. Felsen und J. Malik, *Recurrent network models for human dynamics*, 2015. arXiv: 1508.00271 [cs.CV].
- [12] I. Habibie, D. Holden, J. Schwarz, J. Yearsley und T. Komura, „A recurrent variational autoencoder for human motion synthesis“, Jan. 2017. DOI: 10.5244/C.31.119.
- [13] F. G. Harvey und C. Pal, „Recurrent transition networks for character locomotion“, in *SIGGRAPH Asia 2018 Technical Briefs*, Ser. SA ’18, Tokyo, Japan: Association for Computing Machinery, 2018. DOI: 10.1145/3283254.3283277. Adresse: <https://doi.org/10.1145/3283254.3283277>.
- [14] D. Holden, T. Komura und J. Saito, „Phase-functioned neural networks for character control“, *ACM Trans. Graph.*, Jg. 36, Nr. 4, Juli 2017. DOI: 10.1145/3072959.3073663. Adresse: <https://doi.org/10.1145/3072959.3073663>.
- [15] T. Wiedemer, *Virtuelle nonverbale echtzeitkommunikation basierend auf einem motion capture system und avatar technologien*, Masterthesis, Hochschule Reutlingen, 2020.
- [16] G. D. Rey und K. F. Wender, *Neuronale Netze : eine Einfuehrung in die Grundlagen, Anwendungen und Datenauswertung / Guenter Daniel Rey, Karl F. Wender*. Hogrefe, 2018.
- [17] I. N. da Silva, D. Hernane Spatti, R. Andrade Flauzino, L. H. B. Liboni und S. F. dos Reis Alves, *Artificial Neural Networks : A Practical Course*, Ser. SpringerLink Buecher. Springer, 2017.
- [18] J. Fan, C. Ma und Y. Zhong, *A selective overview of deep learning*, 2019. eprint: 1904.05526.
- [19] J. Frochte, *Maschinelles Lernen - Grundlagen und Algorithmen in Python*. Carl Hanser Verlag GmbH Co KG, 2020.
- [20] I. Goodfellow, Y. Bengio und A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.

- [21] Z. C. Lipton, „A critical review of recurrent neural networks for sequence learning“, *ArXiv*, Jg. abs/1506.00019, 2015.
- [22] T. Neupert, M. Fischer, E. Greplová, K. Choo und M. Denner, „Introduction to machine learning for the sciences“, *ArXiv*, Jg. abs/2102.04883, 2021.
- [23] M. A. Ponti, L. S. F. Ribeiro, T. S. Nazare, T. Bui und J. Collomosse, „Everything you wanted to know about deep learning for computer vision but were afraid to ask“, in *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T)*, 2017, S. 17–41. DOI: 10.1109/SIBGRAPI-T.2017.12.
- [24] J. P. Lewis, K. Anjyo, T. Rhee, M. Zhang, F. Pighin und Z. Deng, „Practice and theory of blendshape facial models“, in *Eurographics*, 2014.
- [25] Byoungwon Choe und Hyeong-Seok Ko, „Analysis and synthesis of facial expressions with hand-generated muscle actuation basis“, in *Proceedings Computer Animation 2001. Fourteenth Conference on Computer Animation (Cat. No.01TH8596)*, 2001, S. 12–19. DOI: 10.1109/CA.2001.982372.
- [26] D. Holden, J. Saito und T. Komura, „A deep learning framework for character motion synthesis and editing“, *ACM Transactions on Graphics*, Jg. 35, S. 1–11, Juli 2016. DOI: 10.1145/2897824.2925975.
- [27] W. Paier, A. Hilsmann und P. Eisert, „Interactive facial animation with deep neural networks“, *IET Computer Vision*, Jg. 14, Nr. 6, S. 359–369, 2020. DOI: 10.1049/iet-cvi.2019.0790.
- [28] C. Doersch, *Tutorial on variational autoencoders*, 2016. arXiv: 1606.05908 [stat.ML].
- [29] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville und Y. Bengio, „Generative adversarial nets“, in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, Ser. NIPS’14, Montreal, Canada: MIT Press, 2014, 2672–2680.
- [30] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim und J. Choo, „Stargan: Unified generative adversarial networks for multi-domain image-to-image translation“, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.
- [31] A. Pumarola, A. Agudo, A. Martinez, A. Sanfeliu und F. Moreno-Noguer, „Ganimation: One-shot anatomically consistent facial animation“, 2019.

- [32] R. Abdrashitov, F. Chevalier und K. Singh, „Interactive exploration and refinement of facial expression using manifold learning“, Ser. UIST ’20, Virtual Event, USA: Association for Computing Machinery, 2020, 778–790. DOI: 10.1145/3379337.3415877. Adresse: <https://doi.org/10.1145/3379337.3415877>.
- [33] A. Wiegand, *Eine einfuehrung in generative adverserial network(gan)*, 2018. Adresse: https://cogsys.uni-bamberg.de/teaching/ws1718/sem_m2/GAN-AndreasWiegand.pdf.
- [34] Q. Wang, T. Artieres, M. Chen und L. Denoyer, „Adversarial learning for modeling human motion“, *The Visual Computer*, Jg. 36, Sep. 2018. DOI: 10.1007/s00371-018-1594-7.
- [35] Y. Wang, W. Che und B. Xu, „Encoder–decoder recurrent network model for interactive character animation generation“, *The Visual Computer*, Jg. 33, S. 971–980, 2017.
- [36] M. Kaufmann, E. Aksan, J. Song, F. Pece, R. Ziegler und O. Hilliges, „Convolutional autoencoders for human motion infilling“, *2020 International Conference on 3D Vision (3DV)*, S. 918–927, 2020.
- [37] D. P. Kingma und J. Ba, „Adam: A method for stochastic optimization“, *CoRR*, Jg. abs/1412.6980, 2015.

Anhang A

Python Code

Der Quellcode für die in dieser Arbeit verwendete Datenvorverarbeitung sowie der Quellcode des Expression-Generators ist im Ordner *Anhang/ExpressionGenerator* hinterlegt.

Anhang B

Videos

Videos der generierten Ausdrucksübergänge für sämtliche Modelle a bis i können dem Ordner *Anhang/Evaluation* entnommen werden.

Anhang C

Numerische Analyse

Die nachfolgenden Tabellen geben die Sprünge der generierten Daten aller trainierten Modelle der numerischen Analyse an.

Bei den Tabellen zu den Modellen a bis e handelt es sich um die *Symmetrischen Modelle* aus 6.3.2.

Bei der Tabelle zum Modell f handelt es sich um das *asymmetrische Modell* aus 6.3.2

Die Tabellen zu den Modellen g und h beschreiben die Sprünge der *Variationsmodelle* aus 6.3.2. Als letztes sind außerdem die Werte für das Modell i , also für den Expression-Generator aus Kapitel 5.2, sowie zuletzt die Sprünge bei den Daten der Grundwahrheit angehängt.

Tabelle C.1: Modell a - Sprünge

		Animation										Sprung (Abweichung in %)						
von	zu	Var	AU1L	AU1R	AU2L	AU2R	AU4L	AU4R	AU6L	AU6R	AU9	AU10	AU13L	AU13R	AU18	AU22	AU27	
<i>glücklich</i>	<i>neutral</i>	1	30.34	23.56	33.82	12.28	22.73	32.05	125.2	126.7	16.68	10.11	64.9	108.8	16.32	21.87	35.77	
<i>glücklich</i>	<i>überrascht</i>	5	25.55	18.04	27.47	6.54	12.48	17.06	126.6	132.2	17.95	20.15	55.41	98.14	13.68	24.04	32.35	
<i>neutral</i>	<i>überrascht</i>	1	8.71	1.12	8.14	4.82	9.48	13.98	49.02	53.13	0.54	15.98	52.22	78.49	0.68	4.56	0.6	
<i>neutral</i>	<i>glücklich</i>	5	7.71	1.83	7.67	5.87	34.01	38.86	72.25	78.19	2.35	8.36	64.51	95.5	1.54	6.41	2.78	
<i>überrascht</i>	<i>glücklich</i>	1	145.8	152.9	156.2	150.2	14.28	25.92	88.83	77.74	14.28	2.69	12.96	0.87	48.78	12.03	116.3	
<i>überrascht</i>	<i>überrascht</i>	5	60.08	71.18	78.35	83.16	16.38	10.2	84.73	71.79	14.4	5.68	14.96	33.5	6.87	27.61	54.12	
<i>übergangsweise</i>	<i>angewidert</i>	1	14.57	17.97	9.63	19.7	7.84	6.91	8.68	9.97	0.26	3.02	0.14	1.86	1.69	2.21	3.6	
<i>übergangsweise</i>	<i>angewidert</i>	5	14.11	17.73	10.88	21.18	5.8	5.36	10.64	2.77	0.18	1.09	2.5	2.07	2.38	2.46	5.89	

Tabelle C.2: Modell b - Sprünge

		Animation										Sprung (Abweichung in %)						
von	zu	Var	AU1L	AU1R	AU2L	AU2R	AU4L	AU4R	AU6L	AU6R	AU9	AU10	AU13L	AU13R	AU18	AU22	AU27	
<i>glücklich</i>	<i>neutral</i>	1	5.36	5.56	6.15	9.81	25.82	34.41	92.08	94.2	25.55	2.11	59.06	95.49	13.9	11.54	0.16	
<i>glücklich</i>	<i>überrascht</i>	5	17.74	11.72	14.21	4.94	3.02	6.67	72.66	79.91	5.01	20.11	37.51	59.28	3.52	3.03	16.68	
<i>neutral</i>	<i>überrascht</i>	1	17.82	9.88	14.97	4.61	16.07	19.35	31.12	35.42	2.96	15.81	46.63	71.38	0.95	3.07	7.49	
<i>neutral</i>	<i>glücklich</i>	5	18.25	10.25	16.12	5.4	18.54	21.11	51.05	56.87	4.1	11.41	59.81	85.3	1.71	3.98	7.56	
<i>übergangsweise</i>	<i>glücklich</i>	1	14.88	12.02	15.95	9.22	29.49	39.82	17.41	20.16	1.31	3.28	8.44	2.44	48.15	7.07	12.86	
<i>übergangsweise</i>	<i>überrascht</i>	5	20.35	16.78	20.86	15.04	23.88	38.87	13.86	17.53	0.56	2.94	7.7	14.53	8.32	0.54	18.02	
<i>übergangsweise</i>	<i>angewidert</i>	1	44.07	46.74	45.26	5.95	11.89	0.86	6.71	3.52	4.5	4.04	5.69	7.1	2.18	38.17		
<i>übergangsweise</i>	<i>angewidert</i>	5	47.51	46.19	42.23	36.27	2.46	10.92	3.12	7.99	2.74	7.09	3.51	3.39	10.69	2.24	41.09	
<i>übergangsweise</i>	<i>angewidert</i>	1	44.25	49.41	37.62	52.05	36.85	32.76	5.7	7.43	12.49	13.27	3.63	6.11	9.99	6.42	35.05	
<i>übergangsweise</i>	<i>angewidert</i>	5	44.08	49.05	37.84	52.03	34.92	30.09	9.62	2.44	12.89	12.32	3.83	5.54	9.1	6.77	34.98	

Tabelle C.3.: Modell c - Sprünge

		Animation										Sprung (Abweichung in %)									
von	zu	Var	AU1L	AU1R	AU2L	AU2R	AU4L	AU4R	AU6L	AU6R	AU9	AU10	AU13L	AU13R	AU18	AU22	AU27				
<i>glücklich</i>	<i>neutral</i>	1	1.43	1.22	3.91	2.73	15.01	15.81	4.74	0.23	3.61	16.25	11.91	6.82	0.62	1.7	3.97				
		5	7.72	6.62	10.58	5.25	23.4	29.98	7.16	6.98	1.97	4.84	17.33	18.02	0.33	0.64	7.97				
<i>glücklich</i>	<i>überrascht</i>	1	1.91	0.68	0.4	0.31	7.59	10.67	21.52	24.24	2.72	8.76	34.24	54.37	2.72	1.4	6.05				
		5	1.86	1.11	0.85	1.47	4.12	5.91	25.76	30.59	4.62	2.84	39.06	56.79	3.84	0.68	6.39				
<i>neutral</i>	<i>glücklich</i>	1	39.61	30.14	36.72	18.93	10.1	23.5	20.49	15.04	1.73	9.69	3.05	6.62	54.13	9.28	29.43				
		5	21.04	17.77	22.16	20.1	25.84	45.52	6.71	4.9	6.79	1.74	6.72	20.68	19	6.69	11.24				
<i>überrascht</i>	<i>glücklich</i>	1	28.54	32.89	28.52	13.35	15.69	1.67	9.27	2.66	2.09	2.65	4.69	4.51	6.56	6.23	31.89				
		5	35.08	35.44	29.36	27.31	11.44	16.03	2.51	1.29	5.21	6.9	3.66	12.76	10.76	4.09	41.14				
<i>überrascht</i>	<i>angewidert</i>	1	20.47	24.38	15.54	24.33	26.07	17.55	4.92	7.4	2.32	4.79	6.17	9.98	4.92	7.52	15.96				
		5	12.32	17.07	8.36	17.6	21.75	12.44	11.48	3.49	1.98	0.18	1.54	7.29	6.54	7.68	8.97				

Tabelle C.4.: Modell d - Sprünge

		Animation										Sprung (Abweichung in %)									
von	zu	Var	AU1L	AU1R	AU2L	AU2R	AU4L	AU4R	AU6L	AU6R	AU9	AU10	AU13L	AU13R	AU18	AU22	AU27				
<i>glücklich</i>	<i>neutral</i>	1	12.13	11.19	9.15	4.24	1.83	3.35	44.08	47	0.21	4.22	24.33	39.14	2.81	1.15	10.87				
		5	11.87	10.08	8.59	7.25	10.87	19.14	36.52	45.14	1.4	15.04	12.34	20.25	0.86	0.68	11.85				
<i>glücklich</i>	<i>überrascht</i>	1	4.5	2.81	1.06	0.2	17.38	17.64	8.78	7.38	3.97	7.47	25.95	46.41	2.98	4	3.76				
		5	9.48	6.23	5.56	2.17	9.65	8.16	19.68	19.14	4.58	5.56	35.12	53.05	4.05	5.47	4.62				
<i>neutral</i>	<i>glücklich</i>	1	4.54	2.33	3.2	5.71	30.57	35.84	32.93	39.61	5.52	0.59	20.59	3.69	44.81	0.72	6.67				
		5	3.28	0.98	4.68	4.43	26.93	37.38	19.01	27.41	7.57	2.64	18.94	12.74	15.17	9.2	9.17				
<i>überrascht</i>	<i>glücklich</i>	1	27.02	34.61	32.38	20.27	13.1	10.55	17.97	16.86	0.3	0.04	7.93	1.94	5.47	3.95	29.05				
		5	30.78	35.58	30.05	31.47	7.62	5.83	14.65	15.72	0.98	1.86	9.1	7.57	9.03	5.67	32.02				
<i>überrascht</i>	<i>angewidert</i>	1	28.05	35.03	26.22	42.09	35.18	29	12.39	12.17	0.37	11.03	8.07	3.59	1.54	0.35	27.21				
		5	26.45	33.78	25.84	42.11	34	27.04	0.39	4.71	0.86	9.76	9.03	3.77	2.89	1.77	26.82				

Tabelle C.5: Modell e - Sprünge

		Animation												Sprung (Abweichung in %)					
von	zu	Var	AU1L	AU1R	AU2L	AU2R	AU4L	AU4R	AU6L	AU6R	AU9	AU10	AU13L	AU13R	AU18	AU22	AU27		
<i>glücklich</i>	<i>neutral</i>	1	7.54	5.07	5.39	2.55	5.96	4.79	12.37	15.42	4.78	10.16	6.6	9.58	3.75	2.23	5.46		
		5	9.18	7.19	8.99	15.21	13.01	20.51	9.85	0.99	14.99	4.32	7.06	19.13	1.37	9.09	8.96		
<i>glücklich</i>	<i>überrascht</i>	1	0.85	0.82	1.3	4.18	10.78	12.76	9.29	5.26	3.92	5.03	16.7	33.69	2.23	4.08	2.81		
		5	10.92	7.88	8.92	4.52	2.32	4.52	13.75	12.74	0.78	2.45	25.46	36.12	1.38	0.35	3.6		
<i>neutral</i>	<i>glücklich</i>	1	1.82	4.24	4.34	3.36	3.06	4.91	11.18	15.11	6.6	0.37	8.76	1.63	26.57	2.43	10.36		
		5	17.88	14.37	12.54	8.09	4.78	3.39	9.51	12.03	2.8	2.07	2.52	7.69	0.98	2.34	7.37		
<i>überrascht</i>	<i>glücklich</i>	1	9.07	11.64	13.5	10.45	2.36	2.25	16.39	18.2	2.83	0.98	2.36	1.7	3.32	2.9	10.05		
		5	5.98	7.18	3.62	17.82	0.24	1.21	14.51	16.31	6.96	0.02	6.85	9.55	0.1	0.63	6.56		
<i>überrascht</i>	<i>angewidert</i>	1	5.2	1.42	7.44	21.21	10.37	5.91	8.76	3.95	9.82	0.07	5.83	5.79	1.51	1.45	3.99		
		5	7.94	4.84	9.62	19.12	6.81	3.1	6.04	4.92	9.11	4.44	7.41	6.75	1.18	0.25	5.68		

Tabelle C.6: Modell f - Sprünge

		Animation												Sprung (Abweichung in %)					
von	zu	Var	AU1L	AU1R	AU2L	AU2R	AU4L	AU4R	AU6L	AU6R	AU9	AU10	AU13L	AU13R	AU18	AU22	AU27		
<i>glücklich</i>	<i>neutral</i>	1	1.93	0.06	2.37	4.08	7.14	5.43	0.83	3.03	2.17	8.86	0.86	3.8	1.77	1.64	2.6		
		5	1.29	1.79	2.41	0.85	5.95	10.63	18.07	10.64	1.8	2.48	12.93	18.71	2.02	1.15	0.57		
<i>glücklich</i>	<i>überrascht</i>	1	2.7	4.85	4.36	6.66	5.36	4.98	2.75	5.74	0.97	0.57	6.06	19.86	0.82	1.3	6.37		
		5	4.74	1.62	2.87	1.2	2.55	3.17	8.49	5.2	0.75	2.33	17.66	29.81	0.45	0.17	1.96		
<i>neutral</i>	<i>glücklich</i>	1	27.1	19.03	19.49	24.92	85.43	82.6	115.8	95.11	24.43	9.97	41.22	49.92	62.51	0.42	19.78		
		5	19.76	16.01	11.64	3.5	13.96	11.38	107.2	87.39	7.76	7.57	50.01	103.3	3.18	7.26	9.48		
<i>überrascht</i>	<i>glücklich</i>	1	5.16	2.7	0.86	3.11	0.14	3.53	67.41	53.32	4.7	6.36	35.1	69.33	3.5	0.32	4.65		
		5	20.54	24.31	25.11	0.83	6.93	93.87	75.1	5.64	17.89	55.74	106.9	4.57	1.2	27.59			
<i>überrascht</i>	<i>angewidert</i>	1	1.97	3.48	5.1	10.21	5.74	8.48	10.32	0.44	1.23	1.79	2.74	2.54	0.12	0.05			
		5	4.65	2.24	5.28	7.05	9.12	5.35	10.34	0.87	0.17	7.87	0.45	1.45	1.15	1.23	0.24		

C. Numerische Analyse

Tabelle C.7: Modell g - Sprünge

Animation			Sprung (Abweichung in %)														
von	zu	Var	AU1L	AU1R	AU2L	AU2R	AU4L	AU4R	AU6L	AU6R	AU9	AU10	AU13L	AU13R	AU18	AU22	AU27
<i>glücklich</i>	<i>neutral</i>	1	3.7	1.7	6.74	0.18	0.17	1.89	21.84	23.88	0.34	2.72	8.24	11.9	4.48	0.77	4.86
<i>glücklich</i>	<i>überrascht</i>	5	3.6	6.62	0.25	0.07	1.33	4.54	16.36	24.85	1.09	6.52	0.03	3.27	0.39	1.47	0.86
<i>neutral</i>	<i>glücklich</i>	1	16.71	7.52	15.8	4	1.92	1.03	16.61	20.13	0.95	13.02	34.2	43.71	2.01	1.03	12.31
<i>überrascht</i>	<i>glücklich</i>	5	22.63	12.4	21.28	8.64	2.38	0.43	36.54	42.62	0.57	10.02	49.24	59.39	1.35	0.21	13.96
<i>überrascht</i>	<i>angewidert</i>	1	1.15	0.22	4.15	1.64	0.44	5.93	12.77	16.85	2.81	2.62	3.87	5.19	10.21	5.63	0
<i>überrascht</i>	<i>glücklich</i>	5	5.94	4.8	10.2	7.23	3.63	9.29	19.58	22.27	1.97	6.53	12.88	29.29	30.3	1.4	5.07
<i>überrascht</i>	<i>angewidert</i>	1	7.39	5.64	9.5	0.96	8.97	5.04	4.86	6.46	2.16	0.63	2.26	3.85	3.15	2.16	1.33
<i>überrascht</i>	<i>angewidert</i>	5	11.13	10.6	11.16	4.32	3.28	0.29	11.98	3.88	2.15	2	1.62	2.34	4.16	1.86	4.88

Tabelle C.8: Modell h - Sprünge

Animation			Sprung (Abweichung in %)														
von	zu	Var	AU1L	AU1R	AU2L	AU2R	AU4L	AU4R	AU6L	AU6R	AU9	AU10	AU13L	AU13R	AU18	AU22	AU27
<i>glücklich</i>	<i>neutral</i>	1	2.37	1.43	4.43	9.59	2.51	2.58	13.08	10.72	1.58	2.49	7.31	7.76	3.9	1.14	8.05
<i>glücklich</i>	<i>überrascht</i>	5	5.89	5.33	6.15	0.15	2.9	2.38	12.42	16.14	2.16	9.34	6.75	0.77	8.95	0.79	1.66
<i>neutral</i>	<i>glücklich</i>	1	3.22	3.58	0.96	6.41	2.94	0.24	16.46	16.41	2.97	7.25	21.48	39.4	0.19	0.58	4
<i>überrascht</i>	<i>glücklich</i>	5	8.59	1.51	6.54	2.94	1.14	2.44	33.51	36.16	2.79	3.92	37.21	52.14	2.42	2.5	1.28
<i>überrascht</i>	<i>angewidert</i>	1	13.81	9.46	12.21	1.18	12.06	5.73	21.81	23.13	2.01	1.52	9.85	7.23	17.63	2.55	7.55
<i>überrascht</i>	<i>angewidert</i>	5	7.98	9.05	3.29	9.11	5.99	9.35	7.77	11.95	3.15	4.62	0.47	4.01	20.16	9.49	0.62
<i>überrascht</i>	<i>glücklich</i>	1	10.22	13.52	11.96	2.34	0.09	2.01	14.89	18.69	0.67	5.02	1.49	1.06	1.09	11.02	8.3
<i>überrascht</i>	<i>angewidert</i>	1	4.59	8.83	0.07	12.41	18.86	9.56	15.38	18.31	2.43	10.08	5.51	10.25	4.03	3.05	4.38
<i>überrascht</i>	<i>angewidert</i>	5	6.5	9.93	3.66	14.52	16.17	7.11	1.03	10.19	0.9	5.57	7.01	11.76	3.21	4.42	7.23

C. Numerische Analyse

Tabelle C.9.: Modell i (Expression-Generator) - Sprünge

		Animation										Sprung (Abweichung in %)						
von	zu	Var	AU1L	AU1R	AU2L	AU2R	AU4L	AU4R	AU6L	AU6R	AU9	AU10	AU13L	AU13R	AU18	AU22	AU27	
<i>glücklich</i>	<i>neutral</i>	1	2.22	2.45	2.3	0.81	6.57	10.5	4	5.18	5.34	4.04	0.49	4.41	2.51	0.03	3.24	
		5	0.03	1.92	2.81	3	11.87	17.77	9.29	1.69	4.09	1.48	6.08	15.82	0.26	1.54	4.57	
<i>glücklich</i>	<i>überrascht</i>	1	3.68	2.87	5.93	3.37	0.78	0.92	8.14	13.93	0.48	4.46	0.01	8.34	1.77	0.74	8.33	
		5	3.83	2.81	2.57	0.49	9.3	8	12.85	10.32	0.98	1.57	15.01	27.19	1.07	1.91	5.19	
<i>neutral</i>	<i>glücklich</i>	1	7.88	6.07	8.89	5.6	21.9	29.83	4.74	4.06	4.98	4.23	0.38	12.83	10.41	4.75	0.12	
		5	1.81	2.06	1.99	2.37	1.31	7.25	1.15	3.06	0.38	0.14	2.41	5.08	4.5	2.23	5.32	
<i>überrascht</i>	<i>glücklich</i>	1	7.98	16.33	8.6	5.87	0.52	2.66	12.81	13.36	9.78	4.24	0.17	0.58	8.19	3.37	5.61	
		5	11.9	16.19	4.64	9.83	1.45	0.12	13.71	14.21	12.56	4.88	0.17	6.25	6.14	4.13	6.29	
<i>überrascht</i>	<i>angewidert</i>	1	1.74	2.35	1.23	5.78	16.24	14.88	5.51	6.96	1.26	4.33	2.3	5.36	0.2	0.63	4.62	
		5	1.76	1.33	5.11	1.73	15.44	12.33	5.6	1.86	4.09	4.5	3.11	6.67	3.26	3.18	1.89	

Tabelle C.10.: Grundwahrheit - Sprünge

		Animation										Sprung (Abweichung in %)						
von	zu	Var	AU1L	AU1R	AU2L	AU2R	AU4L	AU4R	AU6L	AU6R	AU9	AU10	AU13L	AU13R	AU18	AU22	AU27	
<i>glücklich</i>	<i>neutral</i>	1	0	0	0.03	0.23	0	0.41	0.8	0	0.06	0.39	0.36	0	0	0	0	
		5	0	0	0	0.03	0.32	0.04	0	0	0.38	0.08	0.72	0	0	0	0	
<i>glücklich</i>	<i>überrascht</i>	1	0	0	0	0.02	0	0.08	0.61	0	0.24	0.41	1.12	0	0	0	0	
		5	0	0	0	0.2	0.26	0.4	0.71	0	0.47	0.86	0.73	0	0	0	0	
<i>neutral</i>	<i>glücklich</i>	1	0	0.04	0	0	0.27	0.17	0.25	0.46	0	0	1.04	0.28	0.49	0.97	0	
		5	0	0	0	0	0.21	0	0	0	0	0	0	0	0	0	0	
<i>überrascht</i>	<i>glücklich</i>	1	0	0.17	1.08	0.82	0.38	0	0	1.02	0	0	0	0	0	0	0	
		5	0.04	0.23	0.37	0.4	0.01	0.02	0	0	0	0	0	0	0	0	0	
<i>überrascht</i>	<i>angewidert</i>	1	0	0	0	0	0	0	0.04	0	0	0	0	0	0	0	0	
		5	0	0	0	0	0	0.09	0.09	0	0	0	0	0	0	0	0.16	

Erklärung

Ich versichere, dass ich diese Thesis/Arbeit ohne fremde Hilfe selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie alle wörtlichen oder sinngemäß übernommenen Stellen in der Arbeit gekennzeichnet habe. Die Arbeit wurde noch keiner Kommission zur Prüfung vorgelegt und verletzt in keiner Weise Rechte Dritter.

Reutlingen, Abgabe: 29.03.2021

Dominik Walczak