HAW HAMBURG

MASTERTHESIS
Simon Heinz Wolff

Kamerabasiertes Verfahren mit neuronalem Netz zur Prognose eines Roulettespiels

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Simon Heinz Wolff

Kamerabasiertes Verfahren mit neuronalem Netz zur Prognose eines Roulettespiels

Masterthesis eingereicht im Rahmen der Masterprüfung
im Studiengang Automatisierung
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Hans Peter Kölzer
Zweitgutachter : Prof. Dr. Klaus Jünemann

Abgegeben am 25. Mai 2021

Simon Heinz Wolff

Thema der Masterthesis

Kamerabasiertes Verfahren mit neuronalem Netz zur Prognose eines Roulettespiels

Stichworte

Neuronale Netze, CNN, Bildverarbeitung, Prädiktion chaotischer Systeme, Roulette

Kurzzusammenfassung

Die Approximation mittels Modellbildung ist insbesondere bei Systemen mit chaotischen Tendenzen problematisch, da Abweichungen durch Vereinfachungen oder Vernachlässigungen kleinster Einflüsse bereits erhebliche Auswirkungen haben können. Die vorliegende Masterthesis untersucht die Prädizierbarkeit chaotischer Systeme auf Basis eines neuronalen Netzes und die Fähigkeit eines solchen, physikalische Zusammenhänge anhand visueller Daten eines Roulettespiels zu erlernen. Weiterführend wird untersucht, bis zu welchem Maße solche Lernvorgänge durch speziell entwickelte Kostenfunktionen unterstützt werden können und inwieweit diese konventionellen Ansätzen überlegen sind.

Simon Heinz Wolff

Title of the paper

Camera-based method with a neural network for predicting a game of roulette

Keywords

neural networks, CNN, image processing, prediction of chaotic systems, roulette

Abstract

Modeling systems, especially those with chaotic tendencies, is often challenging, since deviations due to simplifications may have a huge impact on resulting conditions. The present thesis evaluates the predictability of chaotic systems by neural networks and the ability of such networks to learn physical dependencies based on visual data of a game of roulette. For the optimization of neural networks, the learning process is furthermore supported by custom designed loss functions and their performance is compared to conventional approaches.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Motivation und Zielsetzung

Unsere Welt ist beherrscht von chaotischen Systemen in jedem erdenklichen Bereich wie der Biologie, Ökologie, Chemie, Medizin oder Physik. Seien es natürliche Wechselwirkungen, wie die des Wetters oder des Aufkommens von Erdbeben, oder aber menschliche Zusammenhänge, wie Verkehrsaufkommen oder Marktschwankungen; ihre Vorhersage kann einen immensen Wert darstellen. Eine Gemeinsamkeit dieser Systeme ist oft der hochgradig indirekte Bezug zwischen den verfügbaren Daten und dem Ergebnis. Seit der Definition des Schmetterlingseffekts von Edward N. Lorenz im Jahr 1962 [1], wurde mittels verschiedenster Ansätze versucht, diese, insbesondere durch Modellbildung und Algorithmen zur Datenverarbeitung, möglichst präzise zu rekonstruieren und zu prädizieren. Das Problem der Chaotik besteht im Wesentlichen darin, dass, obwohl solche Systeme grundsätzlich deterministisch sind, ihre Trajektorie im Phasenraum extrem empfindlich gegenüber ihren Anfangsbedingungen ist. Ein hinreichend genau prädizierbarer Zeitraum wird somit stark von der Qualität der Messungen und der Approximation des Systems begrenzt. In dieser Arbeit wird am Beispiel der Prädiktion der Ergebnisse eines Roulettespiels untersucht, ob und inwieweit neuronale Netze Ansätzen mit Modellbildung überlegen sein können.

Bei der Nachbildung eines physikalischen Systems durch ein mathematisches Modell, werden oft Einflüsse und Wechselwirkungen vereinfacht bzw. vernachlässigt. Beispielsweise werden für eine physikalische Größe, wie der negativen Beschleunigung durch Reibung, anhand von Testreihen systemspezifische Parameter ermittelt, sodass mit relativ simplen Formeln die tatsächlichen Gegebenheiten näherungsweise abgebildet werden können. Diese Abweichungen von der Realität sind bei der Prädiktion chaotischer Systeme jedoch fatal. Neuronale Netze benötigen zum Erlernen eines solchen Systems aufgrund der notwendigen Genauigkeit zwar deutlich mehr Testergebnisse, sie sind dafür aber in der Lage, hochkomplexe statistische Zusammenhänge auf eine Weise zu erlernen, die jede erfassbare Korrelation zwischen Eingangsdaten und Ergebnis berücksichtigen kann. Beispielsweise kann der Einfluss der Reibung von dem aktellen Ort der Kugel, der Kugelform, dem Drall, der Temperatur, der Luftfeuchtigkeit, dem Strömungsverhalten, der Elastizität der Materialien, verschiedenen Kräften auf die Berührungspunkte und vielen weiteren Faktoren abhängig sein. Bei der Entwicklung neuronaler Netze werden in dieser Arbeit jedoch auch gewisse Eingeständnisse gemacht. Bezüglich der Eingangsdaten werden diese auf visuell erfassbare Daten begrenzt, sodass ein Bezug zu Forschungen der Prädiktion von Roulette mittels Modellbildung möglich ist. Einflüsse, wie die Temperatur, werden somit auch unter Berücksichtigung der Chaotik als vernachlässigbar klein

angenommen. Grundsätzlich können Abweichungen oder fehlende relevante Informationen toleriert werden, solange die Summe der resultierenden Abweichungen des prädizierten Ergebnisses die letztendliche Klassifizierung nicht verändert. Da eine Prädiktion solcher Systeme nie fehlerfrei sein wird, ist die Bestimmung der Relevanz solcher Einflüsse ein wichtiger Bestandteil dieser Arbeit. Hierzu wird systematisch untersucht, inwieweit welche Faktoren unterdrückt bzw. optimiert werden können, um den Anteil richtiger Prädiktionen zu maximieren.

Entgegen der Modellbildung entfernt sich die Entwicklungsarbeit neuronaler Netze von der Notwendigkeit eines tiefgehenden mathematischen Verständnisses des abzubildenden Systems. Der Fokus liegt auf der Optimierung von Hyperparametern[1] deren systemspezifischer Bezug vorab oft nur allgemein definiert werden kann. Dennoch kann es vorteilhaft sein, dem Netz, durch gegebene Freiheiten in dessen Dimensionierung, dazu zu verhelfen, bestimmte Bezüge besser zu erlernen und zu generalisieren. Diesbezüglich wird eine proprietäre Kostenfunktion entwickelt und die Hypothese untersucht, dass eine systemspezifische Fehlerskalierung zu einer Verbesserung des Lernverhaltens führen kann.

[1] Parameter zur Steuerung des Lernprozesses neuronaler Netze

2 Grundlagen

2.1 Künstliche neuronale Netze

Der Grundstein für künstliche neuronale Netze wurde mit der Entwicklung des Perzeptrons bereits in den 1950er Jahren von Frank Rosenblatt gelegt. Mangelnde Rechenressourcen und der theoretische Beweis einer eingeschränkten Leistungsfähigkeit von Perzeptronen durch Marvin Minsky und Seymour Papert haben die Entwicklung neuronaler Netze im 20. Jahrhundert stark ausgebremst (vgl. [2]). In den letzten 20 Jahren hat sich die Forschung, mit einer Vielzahl an neuartigen Netzstrukturen und Performance-optimierenden Algorithmen, rasant weiterentwickelt und ist mittlerweile ein wichtiger Bestandteil in diversen Anwendungsgebieten. In dieser Arbeit werden ausschließlich mehrschichtige vorwärtsgerichtete Netze mit einer dichten Struktur *FFNN* (feedforward neural network) und deren faltende Erweiterung *CNN* (convolutional neural network) untersucht. Die Eingänge eines Neurons einer Schicht sind somit ausschließlich mit den Ausgängen jedes Neurons der vorherigen Schicht verbunden. Ein grundlegendes Verständnis über den Aufbau und die Funktionsweise von künstlichen Neuronen, Bias-Neuronen und neuronalen Netzen wird vorrausgesetzt. In den nachfolgenden Abschnitten wird ein grober Überblick über die in dieser Arbeit verwendeten Techniken gegeben.

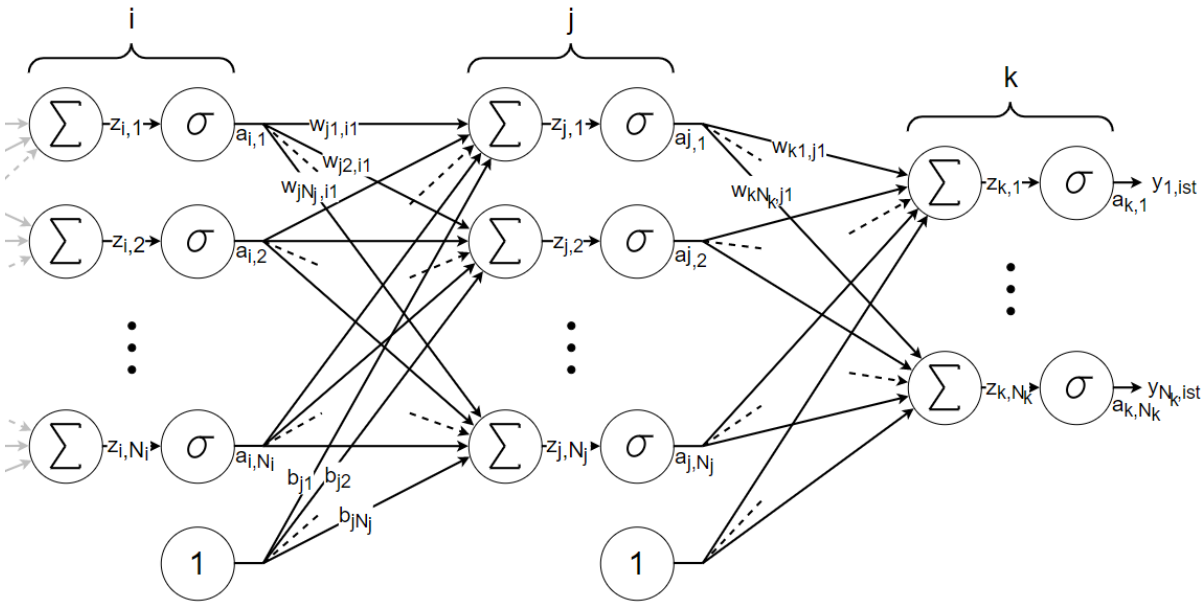


Abbildung 2.1: Allgemeiner Aufbau eines mehrschichtigen, vorwärts gerichteten neuronalen Netzes dichter Struktur mit der in dieser Arbeit verwendeten Notation

2.1.1 Faltende neuronale Netze

FFNN haben den Nachteil, dass die Anzahl der zu trainierenden Parameter für große Netzeingänge, wie es bei Bildern mit hoher Auflösung der Fall ist, beträchtlich ansteigt. Da die meisten relevanten Informationen oft aus lokalen Relationen der Eingangsdaten wie beispielsweise bestimmter Konturen in Bildern bestehen, hat sich gezeigt, dass die Informationsaufbereitung und Komprimierung durch vorgeschaltete Faltungs- und Poolingschichten die, mit normalen FFNN erzielten, Leistungen erheblich verbessert (vgl. [3]).

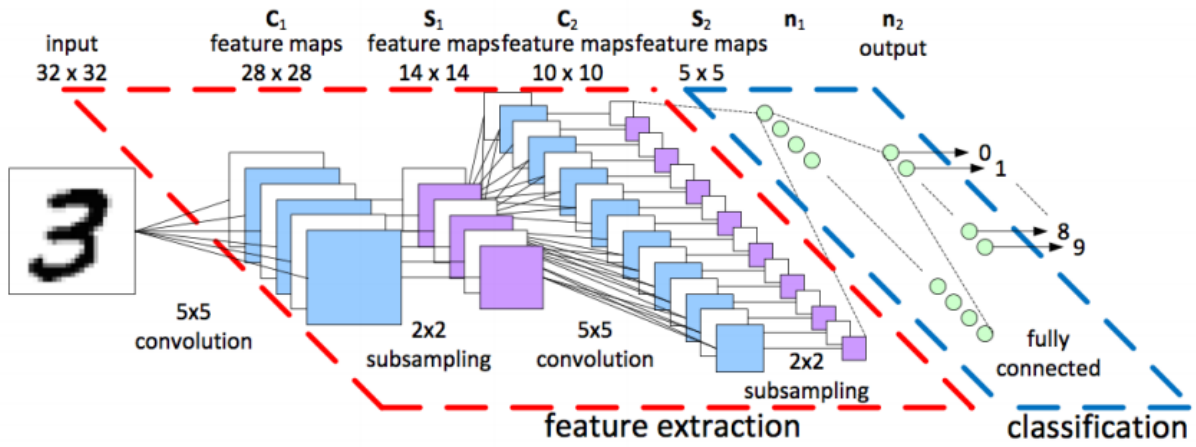


Abbildung 2.2: Beispielhafter Aufbau eines faltenden neuronalen Netzes (Quelle: [4])

Der zugrundeliegende Gedanke ist, durch Faltungen mit sogenannten Faltungsmasken (auch Faltungskern oder Filterkern genannt) bestimmte Eigenschaften aus den Eingangsbildern zu extrahieren, welche für den Bezug zu den Ausgangsdaten relevant sind. Auch diese lassen sich nach dem Prinzip des Backpropagation-Algorithmus aus Kapitel 2.1.5 erlernen, da sich eine Faltungsschicht pro Faltungskern wie eine Schicht von Neuronen mit identischen Gewichten an Verbindungen zu lokalen Neuronen der vorherigen Schicht verhält. Die Dimensionen der verwendeten Faltungsmasken sind in der Regel deutlich kleiner als die der Eingangsdaten, sodass ein Erlernen dieser Masken erheblich weniger Rechenzeit erfordert. Diskrete Faltungen können jedoch relativ rechenaufwendig werden, da für ein Bild B mit einer quadratischen Filtermaske F mit ungeraden Dimensionen, abgesehen von den Randbereichen, für jeden Pixel entsprechend Formel 2.1 das Ergebnis bestimmt werden muss. Teilweise ist es sinnvoll, für eine Faltung gleich mehrere Eingangsdaten zu berücksichtigen, da so auch beispielsweise zeitlich bedingte Eigenschaften verdeutlicht werden können. Der Faltungskern würde somit um eine Dimension erweitert werden, was die Berechnungszeit etwa um die Größe dieser Dimension vervielfacht. Um dies zu kompensieren, können auch kleine Filterkerne aufgespreizt werden, indem zwischen den Elementen eine bestimmte Anzahl an Nullzeilen bzw. -spalten eingefügt wird. Folglich werden aber auch weniger Informationen für die Faltung berücksichtigt, was bei der Modellierung chaotischer Systeme kritisch ist. Sollte eine gleichbleibende Bildgröße erwünscht sein, ist eine Filterung in den Randbereichen durch künstliches Erweitern der Bildmatrix (sogenanntem Padding) möglich (vgl. [2]).

$$B_{i,j} * F = \sum_{m=1}^{\dim(F)} \sum_{n=1}^{\dim(F)} B_{i+m--\frac{\dim(F)-1}{2},j+n--\frac{\dim(F)-1}{2}} \cdot F_{m,n} \quad 2.1$$

Die Anzahl der zu erlernenden Parameter ist also geringer, die relative Berechnungszeit eines Netz-Ergebnisses ist aber, durch die Vielzahl an Filtern pro Faltungsschicht, möglicherweise deutlich größer. Diese lässt sich reduzieren, indem die Filtermaske nur über jedes n-te Element gelegt wird — wobei n über die sogenannten Stride-Parameter einer Schicht für jede Dimension definiert wird. Darüber hinaus lässt sich dies durch die Pooling-Schichten (in Abbildung 2.2 subsampling genannt) kompensieren, welche in der Regel jeder Faltungsschicht angehängt werden. Hier wird versucht, die Größe der durch die Faltungen erzeugten Matrizen ohne relevanten Informationsverlust zu reduzieren, indem eine Gruppe von Feldern anhand einer Funktion zusammengefasst wird. Häufig verwendete Funktionen sind hier die *max()*-Funktion, sowie eine einfache Mittelwertbildung. Es entsteht somit eine Vielzahl kleinerer Matrizen, welche relevante lokale Informationen auf sehr abstrakte Weise abbilden. Durch die Abstrahierung und Skalierung wird zudem eine gewisse Verschiebungs- und Verzerrungsinvarianz erreicht. Neben weiteren Faktoren — wie einer verbesserten Störreduktion durch Faltung — ist diese Art von neuronalen Netzen hierdurch deutlich robuster gegen veränderliche Daten (vgl. [3]).

2.1.2 Aktivierungsfunktion

Gemäß Abbildung 2.1 wird in jedem Neuron, entsprechend Formel 2.6, das Skalarprodukt aus dem Ausgangsvektor der vorherigen Schicht und dem Gewichtevektor gebildet. Ohne eine Aktivierungsfunktion σ wäre also die Übertragungsfunktion eines Neurons linear und eine Verkettung von Neuronen somit ebenfalls. Aktivierungsfunktionen ermöglichen, dass Netze auch nichtlineare Zusammenhänge abbilden können, solange sie selbst nicht linear sind (vgl. [5]). Der Ausgang in jedem Neuron und bei CNN auch in jeder Faltungsschicht wird daher in der Regel durch Aktivierungsfunktionen abgebildet.

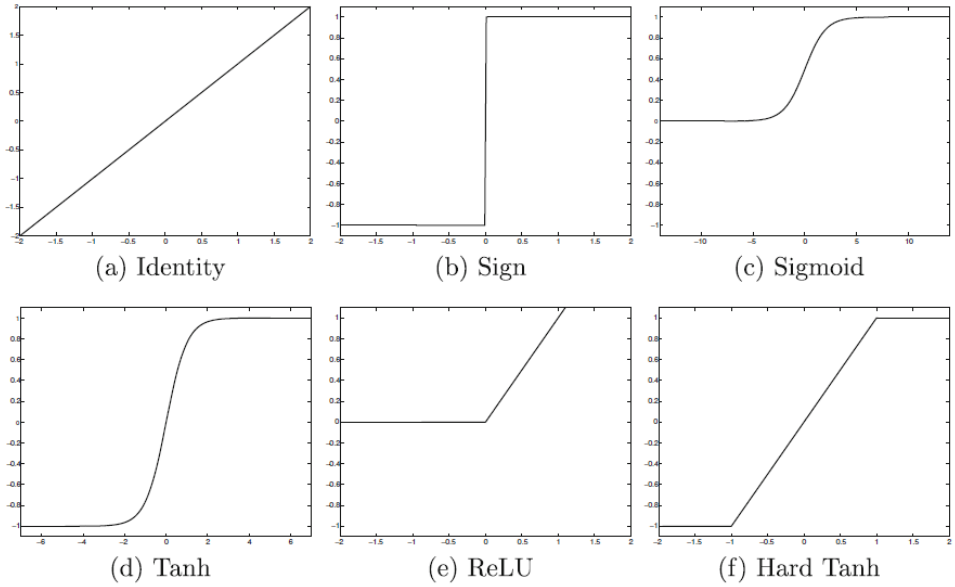


Abbildung 2.3: Gegenüberstellung gängiger Aktivierungsfunktionen (Quelle: [6])

Grundsätzlich eignet sich für diese Arbeit jede Art von nichtlinearer, stetiger, monotoner Funktion. Manche Aktivierungsfunktionen kommen für bestimmte Aufgaben jedoch eher in Frage als andere. Allgemein vorteilhaft sind Funktionen, deren Ableitungen leicht zu berechnen sind, da ihre Berechnungszeit einen erheblichen Einfluss auf die Trainingsgeschwindigkeit hat (siehe Kapitel 2.1.5). Die *Sigmoid*-Funktion hatte sich früher in vielen Anwendungsbereichen als äußerst nützlich erwiesen, da sie den, teils sehr großen, Eingabebereich in einem Bereich zwischen Null und Eins abbildet. Zudem kann ihre Ausgabe in vielen Fällen als eine Art Wahrscheinlichkeit interpretiert werden, was besonders in der Ausgabeschicht gewisse Vorteile bietet. Die *Tanh*-Funktion bildet den Eingang zwischen -1 und 1 ab, sodass auch negative Korrelationen berücksichtigt werden können. Bei vielen Aufgaben ist die *Tanh*-Funktion der *Sigmoid*-Funktion somit überlegen (vgl. [5]). Bei dem heutigen Trend zu immer tieferen Netzen können diese Funktionen jedoch das Training deutlich verlangsamen, was als *Vanishing-Gradient*-Problem bezeichnet wird. Die Ableitung dieser Funktionen strebt für betragsmäßig hohe Eingabewerte gegen Null. Bei Anwendung des *Backpropagation*-Algorithmus gemäß Formel 2.8 ist diese Ableitung Teil des Produktes zur Bestimmung der Gewichte im nächsten Iterationsschritt. Durch die Verstärkung dieses Effektes bei der rückwärtigen Ableitung der Einflüsse von Gewichteänderungen auf den Fehler, verhindert dies, besonders in den früheren Ebenen, eine effiziente Anpassung. Durch die Form der *ReLU*-Funktion tritt dieser Effekt bei großen Eingabewerten nicht auf, weshalb diese insbesondere bei tieferen Netzen oft eine bessere Performance zeigt und daher stark an Popularität gewonnen hat (vgl. [7]). Für negative Eingaben ergibt diese jedoch Null und verhindert somit volkommen eine Anpassung der Gewichte. Die logische Konsequenz bietet die *Elu*-Funktion, welche sich für negative Eingabewerte der -1 annähert, allerdings somit ähnlich zur *Sigmoid*- und *Tanh*-Funktion bei sehr negativen Werten eine Ableitung nahe Null erzeugt.

Ein weiteres Problem vieler Aktivierungsfunktionen liegt in ihrer Verwendung in der Ausgabeschicht. Insbesondere für Klassifizierungsaufgaben tendieren Netze dazu, bei ähnlichen Klassen auch ähnliche Ausgaben zu erzeugen. One-Hot-kodierte Ausgabevektoren erzeugen daher bei der Netzausgabe auch bei grundsätzlich guten Klassifizierungen für Elemente ähnlicher Klassen relativ hohe Werte, obwohl an diesen Stellen durch die Kodierung eigentlich eine Null ausgegeben werden sollte. Durch diese Abweichung ergibt sich ein, Fehler welcher relativ starke Änderungen an den Gewichten verursacht, obwohl diese für die Performance deutlich weniger problematisch sind als andere. Das Problem lässt sich jedoch teilweise kompensieren, indem am Netzausgang die *Softmax*-Funktion verwendet wird:

$$\sigma(z_k) = \frac{e^{z_k}}{\sum_{j=1}^{N_k} e^{z_j}} \quad \text{für } k = 1 \dots N_k \tag{2.2}$$

Auf diese Weise werden Ausgänge ungleich dem Maximalen deutlich stärker unterdrückt und haben somit einen viel kleineren Einfluss auf den Fehler. Zudem lässt sich die Ausgabe durch die Normalisierung auch als eine Wahrscheinlichkeitsverteilung betrachten. Welche Aktivierungsfunktion für welche Schicht tatsächlich die Beste ist, lässt sich jedoch nur vermuten.

2.1.3 Kostenfunktion

Die Kostenfunktion verzerrt die Differenz zwischen der Netzausgabe und dem erzielten Ergebnis auf eine Weise, dass ihre partielle Ableitung nach den einzelnen Gewichten durch den Backpropagation-Algorithmus das Erlernen einer optimalen Gewichte-Kombination unterstützt. Mit ihr kann definiert werden, auf welche Abweichungen der Fokus beim Training gesetzt wird. Beispielsweise ist es üblich, kleineren Abweichungen einen deutlich geringeren Fehler zuzuweisen, sodass beim Lernen die Korrektur grober Abweichungen stark priorisiert wird, bevor das Eliminieren kleiner Fehler möglicherweise ein Overfitting begünstigt. Bei der *MSE*-Funktion wird dies durch simples Quadrieren der Abweichung realisiert [8]:

$$E_{MSE} = \frac{1}{2} \sum_{k=1}^{N_k} \left(y_{k,ist} - y_{k,soll}\right)^2 \tag{2.3}$$

Die Division durch Zwei dient hier lediglich der Vereinfachung der Ableitung und der daraus resultierenden kürzeren Berechnungszeit. Aufgrund ihres immensen Einflusses auf die Trainingsgeschwindigkeit wird hierauf allgemein sehr viel Wert gelegt. Ähnlich zur Wahl der Aktivierungsfunktion lässt sich die Eignung einer Kostenfunktion für eine bestimmte Aufgabe nur relativ allgemein beschreiben und muss durch Tests bestätigt werden, da für den Backpropagation-Algorithmus grundsätzlich jede differenzierbare Funktion verwendet werden kann. Für Klassifikationsaufgaben ist es üblich, in

Kombination mit der Softmax-Aktivierungsfunktion am Netzausgang die Kreuzentropie als Kostenfunktion zu verwenden (vgl. [6]):

$$E_{Kreuzentropie} = -\sum_{k=1}^{N_k} y_{k,soll} \cdot \log\left(y_{k,ist}\right) \tag{2.4}$$

Dies hat den Vorteil, dass sich ein Großteil der Ableitung der Kreuzentropie durch die Ableitung der Softmax-Funktion kürzen lässt. Es existieren diverse weitere Kostenfunktionen, welche auch für diese Arbeit in Frage kommen. Jedoch lässt sich nur vermuten, welche von diesen tatsächlich am besten performt. Die Wahl der Kostenfunktion kann einen signifikanten Einfluss auf die Netzperformance und Trainierbarkeit haben [9]. Diesbezüglich liegt der Fokus dieser Arbeit auf einem proprietären Ansatz, welcher in Kapitel 4.5 genauer beschrieben wird. Gängige Kostenfunktionen dienen hier vorrangig zum Vergleich der Performance.

2.1.4 Lernverfahren

Die Art und Weise, auf welche ein neuronales Netz lernen kann, lässt sich im Wesentlichen in drei Gruppen unterteilen:

- unüberwachtes Lernen
- bestärkendes Lernen
- überwachtes Lernen

Eine Anwendung der jeweiligen Lernmethode hängt von dem Umfang der verfügbaren Daten ab. Beim unüberwachten Lernen stehen für die Berechnung der Abweichung der Netzausgabe keine Sollwerte zur Verfügung. Es muss selbst erlernt werden, auf welche Weise auf die Eingangsdaten reagiert werden soll. Für Klassifizierungsaufgaben ist dies beispielsweise durch Erkennung von Clustern möglich. Beim bestärkenden Lernen kann oft keine genaue Aussage darüber getroffen werden, wie groß der Ausgabefehler eines Netzes ist. Mitunter muss hier nur anhand der Aussagen *richtig* oder *falsch* gelernt werden, was relativ ineffizient sein kann. Für diese Arbeit ist es möglich, einen Datensatz aufzubauen, in welchem zu allen Eingangsdaten auch die zugehörigen Sollwerte existieren. Solch ein überwachtes Lernen ermöglicht es, anhand des Backpropagation-Algorithmus auf verhältnismäßig effiziente Weise einen optimalen Satz an Gewichten zu erlernen (vgl. [10]).

2.1.5 Backpropagation

Der Backpropagation-Algorithmus ist ein 1986 erstmals veröffentlichtes Gradientenverfahren zur iterativen Minimierung des durch die Kostenfunktion abgebildeten Fehlers [11]. In diesem Kapitel wird ein grundlegendes Verständnis über dessen Anwendung bei mehrschichtigen FFNN gegeben. Der Algorithmus besteht im Wesentlichen aus zwei Arbeitspaketen; dem Vorwärts- und dem Rückwärtsschritt. Im Vorwärtsschritt werden für einen Eingangsdatenpunkt der Reihe nach, anhand der Übertragungsfunktionen, für jede Schicht dessen Ausgänge berechnet. Formel 2.5 und 2.6 zeigen diese beispielhaft für das k-te Neuron der Ausgabeschicht aus Abbildung 2.1.

$$a_k = \sigma(z_k) \tag{2.5}$$

$$z_k = \sum_{n_j=1}^{N_j} w_{k,jn_j} \cdot a_{n_j} + b_{kj} \tag{2.6}$$

Gemäß Kapitel 2.1.3 lässt sich zu dieser Ausgabe, abhängig vom Sollwert, ein Maß bestimmen, welches den Fehler des aktuellen Netzes repräsentiert. In Kombination mit der Kostenfunktion bildet das Netz also für jeden Eingangsvektor einen Punkt auf einer Funktion ab, dessen Anzahl an Dimensionen der Menge an einstellbaren Parametern des Netzes entspricht. Das globale Minimum dieser Funktion, unter Berücksichtigung des gesamten Datensatzes, repräsentiert die Kombination an Gewichten, welche die Korrelation dieser Daten zu den Sollwerten optimal abbildet. Um diese zu ermitteln ist ein iteratives Anpassen jedes Parameters, abhängig von dessen Einfluss auf den Fehler, notwendig. Gemäß Formel 2.7 wird hierzu die Funktion partiell nach jedem Gewicht abgeleitet und durch die Lernrate skaliert.

$$\Delta w_{j,i} = \alpha \cdot \delta_j \cdot a_i = -\alpha \cdot \frac{\partial E_{ges}}{\partial w_{j,i}} \tag{2.7}$$

Mit

$\Delta w_{j,i}$ Wert, um welchen das Gewicht zwischen Neuron i und j im nächsten Iterationsschritt auf Basis des aktuellen Eingangsvektors angepasst werden soll

α Lernrate, positiver, frei wählbarer Skalierungsfaktor

δ_j Fehlermaß, um dessen Wert das Neuron j im nächsten Iterationsschritt im Verhältnis zu allen anderen Neuronen die Summe der gewichteten Eingangswerte ändern soll

a_i Ausgabewert des zum Gewicht zugehörigen Neuronenausgangs bzw. des Netzeingangs

$\frac{\partial E_{ges}}{\partial w_{j,i}}$ Partielle Ableitung des Gesamtfehlers nach dem jeweiligen Gewicht bzw. die Projektion dessen Gradienten auf die Richtung dieses Gewichtes. Der Gradient zeigt in die Richtung des stärksten Anstiegs der Kostenfunktion und wird somit durch das Minuszeichen negiert.

Für die Bestimmung der partiellen Ableitung der Kostenfunktion nach den einzelnen Gewichten $\frac{\partial E_{ges}}{\partial w_{k,j}}$ wird, gemäß der Notation aus Abbildung 2.1, zunächst die Ausgabeschicht betrachtet. Entsprechend der Kettenregel lässt sich die partielle Ableitung nach einem Gewicht durch die partiellen Ableitungen der einzelnen Übertragungsglieder beschreiben. Für die Gewichte der Ausgabeschicht entspricht dies der partiellen Ableitung der Übertragungsfunktion eines Ausgangsneurons gemäß Formeln 2.5 und 2.6 und der der Kostenfunktion nach dem jeweiligen Neuronenausgang:

$$\frac{\partial E_{ges}}{\partial w_{k,j}} = \frac{\partial z_k}{\partial w_{k,j}} \cdot \frac{\partial a_k}{\partial z_k} \cdot \frac{\partial E_{ges}}{\partial a_k} = a_j \cdot \sigma'(z_k) \cdot \frac{\partial E_{ges}}{\partial a_k} \qquad 2.8$$

Mit

a_j Partielle Ableitung der Summe der gewichteten Neuroneneingänge nach dem Gewicht $w_{k,j}$, also die Ausgabe des Neurons j

$\sigma'(z_k)$ Ableitung der Aktivierungsfunktion an der Stelle des Wertes $z_k$im aktuellen Iterationsschritt

$\frac{\partial E_{ges}}{\partial a_k}$ Partielle Ableitung des Gesamtfehlers nach dem Ausgang des Neurons

Für die Berechnung des Gewichts b des Bias-Neurons wird die partielle Ableitung $\frac{\partial z_k}{\partial w_{k,j}}$ durch $\frac{\partial z_k}{\partial b_{k,j}} = 1$ ersetzt. Analog dazu lassen sich auch die Gradienten der vorherigen Schichten bestimmen, indem z_k partiell nach dem Ausgabewert der vorherigen Neuronen a_j abgeleitet wird:

$$\frac{\partial E_{ges}}{\partial a_j} = \frac{\partial z_k}{\partial a_j} \cdot \frac{\partial a_k}{\partial z_k} \cdot \frac{\partial E_{ges}}{\partial a_k} = \sum_{k}^{N_k} w_{k,j} \cdot \sigma'(z_k) \cdot \frac{\partial E_{ges}}{\partial a_k} \qquad 2.9$$

Hier ist allerdings zu beachten, dass der Ausgang eines Neurons den Fehler über alle Neuronen nachfolgender Schichten beeinflusst. Wie bei der Ausgabeschicht ist so ein Maß bestimmt, anhand dessen nun nach dem oben beschriebenen Verfahren die aktualisierten Gewichte jeder Schicht rückwärts bis zum Netzeingang ermittelt werden können. Diese Werte berücksichtigen jedoch nur einen Datenpunkt, was in sehr recheneffizienten, aber auch in ungenauen Aktualisierungen der Gewichte resultiert, da dieser nicht ausreicht, um den gesamten Datensatz sinnvoll zu repräsentieren und somit einen Gradienten erzeugt, der für andere Datenpunkte nicht in die Richtung des nächsten Minimums zeigt. Es ist daher üblich, Δw über den gesamten Datensatz oder zumindest über mehreren Daten, sogenannten Batches, zu mitteln, deren Umfang einen hinreichend genauen Gradienten abbildet (vgl. [5]).

2.1.6 Optimierer

Das Training neuronaler Netze mittels Backpropagation ist ein hochdimensionales, nichtlineares Optimierungsproblem. Durch die iterative Herangehensweise ist es unmöglich zu wissen, ob nach dem Training ein Sattelpunkt, ein lokales oder das globale Minimum gefunden wurde (vgl. [12]). Die Wahrscheinlichkeit und die Geschwindigkeit, das globale oder ein sehr gutes lokales Minimum zu finden, lassen sich durch sogenannte Optimierer erhöhen. Diese erweitern den durch die Kostenfunktion bestimmten Fehler mit verschiedenen Faktoren, um bestimmtes Lernverhalten beim Training zu bevorzugen. Beispielsweise lässt sich das Training als eine auf der Kostenfunktion herrabrollende Kugel betrachten. Änderungen des Gewichtes, der Reibung oder des aufbaubaren Schwunges können entscheidend dafür sein, wie stark ein Minimum ausgeprägt und wie es geformt sein muss, damit die Kugel zum Stillstand kommt (vgl. [13]).

2.1.7 Regularisierung

Mit steigender Größe der Netzstruktur wird durch die Kostenfunktion ein zunehmend komplexerer Fehlerraum aufgespannt, welcher deutlich stärker ausgeprägte lokale Minima aufweisen kann. Diese Netze sind in der Lage, Korrelationen zwischen den Eingangsdaten und den Ergebnissen auf eine viel komplexere Weise zu approximieren. Jedoch wird mit fortlaufendem Training mit hoher Wahrscheinlichkeit der Fehler minimiert, ohne die tatsächlichen physikalischen Zusammenhänge zu erlernen, indem Datensatz-spezifische Korrelationen "auswendig gelernt" werden. Dieser Effekt der Überanpassung (*Overfitting* genannt) tritt besonders bei Datensätzen wie dem in dieser Arbeit verwendeten auf, welche das Verhalten des Gesamtsystems nur zu einem relativ kleinen Anteil beschreiben. In den letzten Jahren konnten beim Training mit großen Netzstrukturen mehrere Durchbrüche erzielt werden, welche diesem Problem begegnen. Allgemein werden sie unter dem Begriff Regularisierung zusammengefasst.

Early Stopping

Ein sehr simpler Ansatz ist der des *Early Stoppings*. Während des Trainings wird kontinuierlich geprüft, ob der Fehler auch bei der Prognose eines, dem Netz unbekannten, Validierungs-Datensatzes abnimmt. Sollte dieser nach mehreren Epochen weiter zunehmen, kann angenommen werden, dass die Zusammenhänge, die aktuell auf Grundlage der Trainingsdaten erlernt werden, nicht allgemein gültig sind. In diesem Fall wird das Training frühzeitig abgebrochen, bevor sich die Performance auf den Validierungsdaten weiter verschlechtert.

Datenanreicherung

Eine weitere bewährte Methode ist die der Datenanreicherung. Oft lässt sich der Trainingsdatensatz durch leichte Modifikationen wie durch Drehen, Skalieren oder Verschieben der Eingangsdaten erheblich vergrößern, wodurch das Netz eine gewisse Invarianz gegenüber diesen Faktoren erlernen kann.

Dropout

Sehr effizient und weit verbreitet ist das Dropout-Verfahren. Hier wird während des Trainings bei jedem Batch in festgelegten Schichten ein bestimmter Anteil zufällig ausgewählter Neuronen deaktiviert. Dies verhindert, dass sich starke Abhängigkeiten zu einzelnen Neuronen entwickeln. Es muss also auf verschiedenen Pfaden durch das Netz Dasselbe erlernt werden, was die Robustheit gegenüber unbekannten Daten immens fördern kann. Diese erzwungene Redundanz verringert die Fähigkeit eines Netzes, die Zusammenhänge auf komplexere Weise zu approximieren. Es hat sich aber bewährt, zur Optimierung der Generalisierung mit Dropout die Netze entsprechend zu vergrößern, da die verbesserte Generalisierungsfähigkeit die stärkere Tendenz zum Overfitting übersteigt und somit teilweise deutlich bessere Validierungsergebnisse erreicht werden können (vgl. [6]).

Für dichte Netzstrukturen, wie denen in dieser Arbeit, bei welchen jedes Neuron einer Schicht mit jedem Neuron der folgenden Schicht verknüpft ist, müssen die Ausgabewerte der aktuell aktiven Neuronen um den Faktor $\frac{1}{1-n}$ skaliert werden, damit sich die Summe am Eingang der Neuronen der folgenden Schicht nicht zu stark ändert. Der Wert n beschreibt hier den Anteil deaktivierter Neuronen in der betrachteten Schicht.

Bezüglich der Anwendung des Dropout-Verfahrens in Faltungsschichten ist es nicht sinnvoll einzelne Elemente der Faltungskerne zu deaktivieren, da sich dort keine Redundanzen ausbilden können, sondern vielmehr zufällige Kerne generiert werden. Eine Alternative wäre, einen Anteil an Kernen zu deaktivieren, sodass sich eine Redundanz zwischen den Kernen ausbilden muss.

Batch-Normalisierung

Eine weitere Möglichkeit das Trainingsverhalten zu optimieren ist die der sogenannten Batch-Normalisierung. Diese ist technisch gesehen keine Regularisierungsmethode, sie hat aber als Nebeneffekt einen regularisierenden Einfluss. Insbesondere bei tieferen Netzen kann es passieren, dass die statistische Verteilung der Daten über mehrere Schichten hinweg beim Training mit jedem Batch stark variieren kann. Dies hat zur Folge, dass die Gewichte während des Trainings insbesondere in tieferen Schichten sehr spezifisch zu jedem Batch angepasst werden, wodurch das Training deutlich verlangsamt werden kann, da sie sich somit in jedem Iterationsschritt kaum auf einen gleichbleibenden Wert zubewegen. Allgemein wird dieser Effekt als "internal covariate shift" bezeichnet. Ähnlich zur Normalisierung der Netzeingänge, soll nach den Erkenntnissen von [14] eine Kompensation dieses Effektes durch eine Normalisierung der Werte in jeder Schicht zu einer deutlich schnelleren Konvergenz des Fehlers führen. Hierzu wird jedes Neuron durch eine Übertragungsfunktion erweitert, welche den Eingangswert nach Formel 2.10 normalisiert und anschließend entsprechend Formel 2.11 skaliert.

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} \qquad 2.10$$

Mit

x_i Eingangswert des Normalisierungsblocks durch den Datenpunkt i im Batch
μ_B Mittelwert von x über das Batch B
σ_B^2 Varianz von x beim Batch B
ε kleiner Offset, um eine mögliche Division durch Null bei $\sigma_B^2 = 0$ zu verhindern

$$y_i = \gamma \cdot \hat{x}_i + \beta \qquad 2.11$$

Mit

γ erlernbarer Skalierungsfaktor
β erlernbarer Offset

Da der Mittelwert und die Varianz aus Formel 2.10 Batch-spezifisch sind, wird durch die Normalisierung ein Rauschen in das Netz induziert. Demzufolge tendieren nachfolgende Neuronen weniger dazu, starke Abhängigkeiten zu einzelnen Neuronen auszubilden, was einen regularisierenden Effekt hat. Es wird öfter debattiert, an welcher Stelle in einem Neuron eine Normalisierung am sinnvollsten ist. Da dies jedoch nicht den Fokus dieser Arbeit darstellt, wird das Verfahren entsprechend der Ansichten aus [14] und [15] implementiert. Der Normalisierungsblock wird somit in jedem Neuron zwischen der Summenformel und der Aktivierungsfunktion eingesetzt. Demzufolge hat der Einfluss des Bias durch die Subtraktion des Mittelwertes keinen Effekt und kann deaktiviert werden. Stattdessen wird diese Aufgabe von dem Parameter β aus Formel 2.11 übernommen. Des Weiteren wird der verwendete Wertebereich der Aktivierungsfunktion durch die Normalisierung stark eingegrenzt, was den Einfluss der Nichtlinearität gemäß Kapitel 2.1.2 abschwächen, bzw. beinahe eliminieren kann. Formel 2.11 hat daher den Hintergrund, die normalisierten Werte so zu verschieben und zu skalieren, dass die nichtlinearen Eigenschaften dennoch optimal genutzt werden können. Durch partielle Ableitungen entsprechend Kapitel 2.1.5 nach den Parametern β und γ lassen sich diese während des Trainings auf dieselbe Weise erlernen wie die Gewichte. Grundsätzlich ließe sich also die Normalisierung mit $\beta = \mu_B$ und $\gamma = \sqrt{\sigma_B^2 + \varepsilon}$ komplett verlernen.

L_p-Regularisierung

Darüber hinaus kann auch eine Erweiterung der Kostenfunktion durch einen Term, welcher Gewichte entsprechend ihrer Größe bestraft, ein generalisierenderes Lernverhalten unterstützen, da auch so eine gewisse Redundanz gefördert wird. Dieses Verfahren wird als L_p-Regularisierung[2] bezeichnet. Gemäß der Erkenntnisse aus [16] werden jedoch relevante Einflüsse auf die Gewichte durch eine Batch-Normalisierung kompensiert. Aufgrund der Komplexität der hier untersuchten Korrelation zwischen Eingangsdaten und Ergebnis wurde eine Notwendigkeit tieferer Netze vermutet. In dieser Arbeit werden diese daher mit den oben beschriebenen Verfahren optimiert und eine L_p-Regularisierung nicht genauer betrachtet. Ähnliches konnte auch bei Anwendung der

[2] p definiert hier die Zahl des Exponenten zu jedem Gewicht im Anhang der Kostenfunktion

Batch-Normalisierung in Kombination mit Dropout gemessen werden (vgl. [14]). Durch den generalisierenden Effekt der Batch-Normalisierung soll sich in manchen Fällen auch die Notwendigkeit des Dropout-Verfahrens erübrigen. Dies scheint jedoch mit dem Anwendungsbereich zu variieren, weshalb in dieser Arbeit beide Verfahren untersucht werden.

2.2 Roulette

Roulette ist ein weitverbreitetes Glücksspiel, dessen Erfindung oft dem französischen Mathematiker Blaise Pascal zugeschrieben wird [17]. Nachfolgend werden einige Begriffe eingeführt, welche im Laufe dieser Arbeit oft verwendet werden.

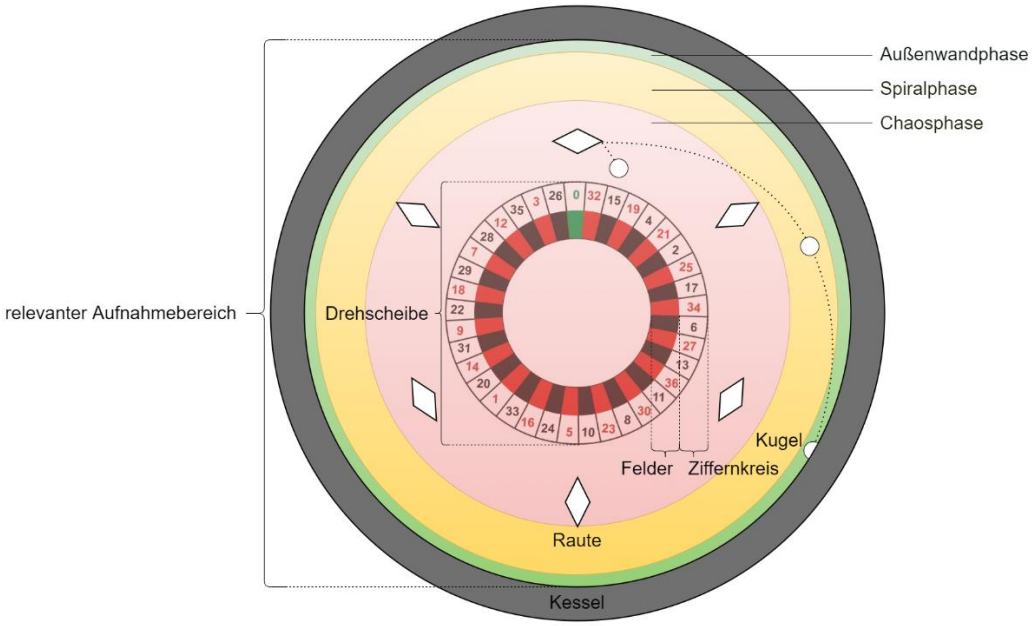


Abbildung 2.4: Überblick Roulettespiel und eingeführte Begriffe

Das Spiel existiert in mehreren Varianten, bei welchen verschiedene Drehscheiben die Gewinnchancen und den Hausvorteil definieren. Der grundlegende Spielablauf ist jedoch, abgesehen von bestimmten Regelungen, wie der des erlaubten Maximaleinsatzes, immer gleich:

Je nach den für die Roulette-Variante üblichen Regeln kann der Spieler, durch das Setzen von Chips (auch Jetons genannt) auf bestimmte Felder oder Gruppen von Feldern, raten, auf welche Zahl die Kugel fallen wird. Wurden genügend Wetten abgeschlossen, setzt der sogenannte Croupier die Drehscheibe in Bewegung und wirft die Kugel in entgegengesetzter Richtung in den Kessel. Kurze Zeit später wird mit der in Deutschland üblichen Aufforderung "Nichts geht mehr" weiteres Platzieren von Wetten verboten. Durch Reibungseffekte verlangsamt sich die Kugel und verlässt nach kurzer Zeit die Außenwand. Gemäß Abbildung 2.4 wird der Zeitraum bis zu diesem Punkt in den

nachfolgenden Kapiteln als Außenwandphase bezeichnet. Die Zeit nach dem Verlassen der Außenwand bis zur Erstberührung mit den Rauten wird hier als Spiralphase definiert. Sobald die Rauten erreicht wurden, beginnt die Chaosphase. Ist die Kugel auf einem der Felder liegen geblieben, sagt der Croupier dessen Ziffer und die zugehörigen gewinnenden allgemeinen Feldgruppen, wie die entsprechende Farbe, an. Anschließend werden die Verluste eingezogen und zuletzt die Gewinne ausgezahlt.

Da das Roulettespiel hier nur als Zufallsgenerator genutzt wird, sind die genauen Spielregeln und Methoden nicht relevant. Entsprechend zu anderen Forschungen zur Prädizierbarkeit von Roulette (siehe [18], [19]) wird auch hier die französische Variante des Spiels betrachtet. Die Drehscheibe besteht somit aus 37 Feldern mit einer Nummerierung von 0 bis 36. Die Wahrscheinlichkeit, dass ein unwissender Spieler das richtige Feld errät, beträgt daher $2,\overline{702}\,\%$. Damit auf verwandte Untersuchungen und deren Testergebnisse Bezug genommen werden kann, wird neben allgemeinen Metriken, wie der Prädiktionswahrscheinlichkeit, auch der Erwartungswert des Gewinns pro Spiel, beim Setzen auf das Feld mit der höchsten Wahrscheinlichkeit, betrachtet. Dieser errechnet sich gemäß Formel 2.12 aus der Wahrscheinlichkeit P(x), das richtige Feld zu treffen, dem erzielbaren Gewinn g und dem bei Falschprädiktion erzeugten Verlust v.

$$E(X) = P(x) \cdot g + \big(1 - P(x)\big) \cdot v \tag{2.12}$$

Wettet man auf eine einzelne Zahl, erhält man im Erfolgsfall üblicherweise, neben dem Einsatz des richtigen Feldes, das 35-fache des Einsatzes zurück (vgl. [20]). Inoffizielle Regelungen, wie Trinkgelder, werden hier nicht berücksichtigt. Um den Hausvorteil zu überwinden und mit einer Software Profit zu machen, muss diese also eine Richtigprädiktionswahrscheinlichkeit P(x) von

$$\begin{aligned} & P(x) \cdot 35 + \big(1 - P(x)\big) \cdot (-1) > 0 \\ \Leftrightarrow \quad & P(x) \cdot 35 > 1 - P(x) \\ \Leftrightarrow \quad & P(x) > \frac{1}{36} = 2,\overline{7}\,\% \end{aligned} \tag{2.13}$$

erzeugen. Neben dem Setzen auf eine einzelne Zahl, ist es beim Roulette auch möglich, auf Gruppen von Feldern zu setzen. Durch die Anordnung der Felder sind diese aber über die gesamte Drehscheibe verteilt. Sobald also ansatzweise eine Aussage getroffen werden kann, in welchem Bereich die Kugel landen wird, werden diese Arten von Wetten uninteressant. Bei Anwendung der in dieser Arbeit entwickelten Software wäre es, aufgrund des Streuverhaltens der Kugeltrajektorie, nur sinnvoll, auf bestimmte Felder relativ zum Wahrscheinlichsten zu wetten.

3 Aktueller Forschungsstand

Seit der digitalen Revolution, etwa um die Jahrtausendwende, gewannen komplexere neuronale Netze durch die erhöhte Rechenleistung zunehmend an Bedeutung. In den letzten Jahren wurden diverse Untersuchungen zur Modellierung physikalischer Systeme durch Netze verschiedenster Form veröffentlicht. Eine Untersuchung der Prädizierbarkeit eines Roulettespiels mittels neuronalen Netzen ist jedoch nicht bekannt. jedoch existieren mehrere Publikationen im Bereich der Prädiktion von Roulette durch physikalische Modellbildung. Neben ausführlichen Untersuchungen in [21] und [22] konnten unter Laborbedingungen Profit-Erwartungswerte von 0,18 [18] und 0,41 [19] erreicht werden, was gegenüber dem Erwartungswert eines unwissenden Spielers von −0,027 beeindruckende Ergebnisse sind. Es wird deutlich, dass eine Prädiktion auch mit veralteter Technik zumindest insoweit möglich ist, dass das Spiel für den Spieler profitabel wird. So konnten die Gewinnchancen im Laufe der Jahre, mit Hilfe simpler Computer, bereits öfter ausreichend gesteigert werden, um teils Gewinne in Millionen-Höhe zu erzielen.

Obwohl neuronale Netze scheinbar noch nicht die Welt des Roulelette erobert haben, gibt es viele Anwendungsbereiche, welche diesem Projekt ähneln. Eine Prädiktion eines physikalischen Systems anhand von Kameradaten wurde beispielsweise von [23] veröffentlicht. Es wurde untersucht, inwieweit faltende neuronale Netze in der Lage sind, zu erkennen, ob ein Turm aus Blöcken einstürzen wird. Die erreichte Performance glich auch bei unbekannten Daten der eines Menschen, wodurch verdeutlicht wurde, dass die neuronalen Netze den Bezug zwischen Eingangsbildern und Ergebnis auf eine Weise verstanden haben, die den tatsächlichen physikalischen Zusammenhängen ähnelt. In [24] wurden Autoencoder verwendet, um aus einem einzelnen Frame die Bewegungen von Pixeln in der nächsten Sekunde eines Videos zu prädizieren. Beispielsweise besitzt ein Bild einer Person auf einer Schaukel eine hohe Wahrscheinlichkeit, dass sich sich diese in den nächsten Frames entsprechend der Schaukelbewegung durch das Bild bewegt und sich dessen Kontur entsprechend der Perspektive verformt. Es konnte gezeigt werden, dass Netze durchaus in der Lage sind, auch Situationszusammenhänge zu verstehen, was aber ein hohes Maß an Wissen erfordert. Dies wäre für eine Anwendung im Casino interessant, um die Bewegungen des Croupiers auch in nicht erfassbaren Bereichen für die Prädiktion berücksichtigen zu können. Für diese Arbeit sind solche Untersuchungen jedoch weniger relevant, da durch die erfassten Konturen des Roulettespiels keine Rückschlüsse auf den zukünftigen Verlauf getroffen werden können und somit in jedem Fall mehrere Bilder notwendig sind, um das Ergebnis erfolgreich zu prädizieren. Passend dazu wird in [25] versucht, anhand von mehreren Bildern zukünftige zu erzeugen. Die Prädiktion wird hier

mit Autoencodern realisiert, welche als rekurrentes neuronales Netz (RNN) verschaltet wurden. Es wurde allerdings deutlich, dass es trotz der multiplen Eingangsbilder sehr schwierig ist, die Veränderungen zwischen zwei Frames auf eine Weise zu erlernen, dass auch bei größerem Prädiktionshorizont sinnvolle Bilder erzeugt werden. Dennoch konnten bei einem Video von hüpfenden Bällen in einer Ebene relativ gute Ergebnisse erzielt werden.

In [26] werden sogenannte intelligente Agenten mit faltenden neuronalen Netzen, in Kombination mit LSTM-Zellen, ausgestattet, um so am Beispiel eines Billiard-Spiels, durch prädizierte Trajektorien, zielgerichtete Entscheidungen treffen zu können. Der Netzeingang besteht hier aus einer Reihe von Bildern in Kombination mit der Kraft die auf die weiße Kugel wirken soll. Es wurden hier sehr gute Ergebnisse in neuen Umgebungen, beispielsweise mit Billiard-Tischen unterschiedlicher Form, erreicht. Dies legt die Vermutung nahe, dass faltende neuronale Netze auch beim Roulettespiel durch Verknüpfung bestimmter Konturen, wie der Größe der Rauten oder dem Kreisumfang der Außenwand, mit dem Ergebnis eine gewisse Robustheit bezüglich der Varianz bestimmter Parameter entwickeln können. So können sie, trotz der Chaotik, nach einem Training mit entsprechendem Fokus auf diese Zusammenhänge, bei anderen Roulettespielen möglicherweise relativ gut performen.

Die Prädiktion von Balltrajektorien mittels neuroner Netze wurde dank weitreichender Anwendungsgebiete, insbesondere in der Sportanalyse, eingehend untersucht. Viele dieser Forschungen betrachten relativ simple Verläufe, bei welchen vor allem die Qualität der Datensätze und die Menge an Eingangsdaten die Performance zu definieren scheinen [27] [28] [29]. Prädiktionen komplexerer Trajektorien, bei welchen Objekte mindestens einmal mit anderen kollidieren, resultierten teilweise in relativ starken Abweichungen. Es wird deutlich, dass die Anforderungen an die Qualität der Datensätze mit zunehmendem chaotischen Verhalten der Trajektorien drastisch steigen. In [30] wurde versucht, anhand von Tischtennis- und Squash-Videos die Position des Balls in den nächsten Frames zu prädizieren. Es wurde teils nur eine Perspektive aufgenommen, sodass dem Netz die Information über die Tiefe des dreidimensionalen Raums der Trajektorie vorenthalten wurde. Entsprechend wurden auch keine sinnvollen Prädiktionen erzeugt. Die fehlende Tiefeninformation ist in dieser Arbeit jedoch unproblematisch, da sich der Bezug zwischen der zweidimensionalen Projektion der dreidimensionalen Oberfläche des Roulette-Tisches nicht ändert und die Kugel während der Aufnahme der Eingangsdaten über den Zeitraum der Außenwand- und Spiralphase durchgehend den Tisch berührt. Die Tiefeninformation der Kugel ist daher rein ortsabhängig und lässt sich somit durch das Netz berücksichtigen. In [30] wurde zudem getestet, ob eine 3D-Faltung am Netzeingang über mehrere Frames hinweg, durch Generierung von Informationen über direkte Bezüge zwischen den Frames, dem Netz einen Vorteil bietet. Es wurden auf diese Weise jedoch deutlich schlechtere Ergebnisse erzeugt, da die Größe der Faltungskerne vermutlich nicht die Bildbereiche der Bälle zweier Frames eingeschlossen hat.
Ähnlich zu [24] wird auch in [31] und [32] versucht, Körpersprache zu interpretieren. In diesem Fall werden jedoch mehrere Frames genutzt, um anhand der Bewegungen vor der Ballberührung mit relativ hoher Genauigkeit die Trakjektorie zu prädizieren. Der Bezug

zwischen Eingangsdaten und Ergebnis ist, ähnlich zu dieser Arbeit, relativ komplex, chaotische Tendenzen sind hier jedoch eher gering. Basierend auf der Idee, dass die meisten Interaktionen zwischen Personen oder Objekten nicht absolute, sondern eher verschwimmende Entscheidungsgrenzen beherbergen, wurde in [33] zur Prädiktion verschiedener Szenarien mit mehreren Entscheidungsträgern eine neuartige Architektur vorgestellt, welche, in Kombination mit LSTM-Zellen, teilweise eine deutlich bessere Performance aufwies. Es wurden hier auch Szenarien betrachtet, bei welchen zukünftige Trajektorien von in einem Raum kollidierenden Kugeln prädiziert werden sollen. Trotz der sichtbar besseren Performance gegenüber anderen Ansätzen, zeigen die Abweichungen durch das chaotische Verhalten, dass auch dieser für eine Roulette-Prädiktion bei Weitem nicht ideal ist.

Untersuchungen, inwieweit Prädiktionen chaotischer Zeitreihen möglich sind, wurden in den letzten Jahren hauptsächlich mit verschiedenen Variationen rekurrenter neuronaler Netze durchgeführt, da diese darauf ausgelegt werden können, zeitlich codierte Informationen aus den Eingangsdaten zu extrahieren [34] [35] [36]. Die meisten dieser Untersuchungen konnten sehr gute Ergebnisse erzielen. Es wurden jedoch hauptsächlich ideale, nicht verrauschte Daten verwendet. Untersuchungen mit unterschiedlich verrauschten Daten in [37] haben gezeigt, dass mit der richtigen Netzstruktur von RBF-Netzen jedoch "zufriedenstellende" Ergebnisse erzeugt werden können und somit eine erfolgreiche Prädiktion hoch chaotischer Systeme auch mit verrauschten Daten grundsätzlich möglich ist.

Es existieren keine bekannten Forschungen, welche dieser Arbeit ausreichend ähneln oder deren Ansätze sich in einem relevanten Gebiet von anderen Publikationen auf eine Weise abheben, die bei Anwendung in dieser Arbeit höhere Erfolge versprechen. Es lassen sich dennoch manche Erkenntnisse, wie der Wert der Qualität der Eingangsdaten oder der Einfluss der Faltungsdimensionen bei CNNs, übertragen.

4 Konzeption

Um eine detaillierte Aussage darüber treffen zu können, inwieweit es möglich ist, anhand von neuronalen Netzen mit Kameradaten chaotische Systeme wie ein Roulettespiel zu prädizieren, müssen weitreichende Untersuchungen mit jeglicher Form von auf neuronalen Netzen basierender Bildverarbeitungssoftware durchgeführt werden. Solche Untersuchungen sind jedoch durch die zur Verfügung stehenden Ressourcen begrenzt. Geht man davon aus, dass jedes physikalische System, unter Berücksichtigung aller Faktoren, deterministisch ist, sollte sich dieses auch durch ein hinreichend komplexes neuronales Netz abbilden lassen. Die Performance einer realen Bildverarbeitungssoftware für diese Aufgabe hängt jedoch davon ab

1. wie groß der Anteil visuell erfassbarer Informationen am Gesamtanteil relevanter Faktoren ist,
2. wie stark deren Qualität, insbesondere in Bezug auf Rauschungenauigkeiten, durch die Aufnahme verschlechtert wird,
3. inwieweit eine Kombination einer Vielzahl an Hyperparametern gefunden und ein Datensatz hinreichender Größe erzeugt werden kann, um neuronale Netze an eine Gewichte-Kombination heranzuführen, welche die Korrelationen zwischen der Aufnahme des Spiels und dem Ergebnis ideal abbildet

Die beiden ersten Punkte werden hauptsächlich durch die verfügbare Hardware definiert und in Kapitel 5 untersucht. Bezüglich der neuronalen Netze muss eine Vielzahl von Entscheidungen getroffen werden; welche Struktur, Kostenfunktion, Optimierung, Regularisierung, Datenaufbereitung und viele weitere Hyperparameter verwendet, beziehungsweise getestet werden. In den nachfolgenden Abschnitten werden, unter Berücksichtigung der Erkenntnisse aus Kapitel 3, bestimmte Einschränkungen getroffen und definiert, auf welche Untersuchungen der Fokus gesetzt wird. Die Wahl der übrigen Hyperparameter und die Eignung verschiedener Verfahren zur Optimierung des Trainings, wird durch Testreihen in Kapitel 6 genauer analysiert.

4.1 Framework

Frameworks sind vorgefertigte Programmbibliotheken, die mit dem Ziel entwickelt wurden, das Arbeiten mit neuronalen Netzen zu erleichtern. In diesen sind bereits vorgefertigte Schichten, Optimierer, Kostenfunktionen, Trainingsalgorithmen und viele weitere Funktionalitäten auf recheneffiziente Weise implementiert. Für diese Arbeit wird das Framework Keras in Kombination mit Tensorflow verwendet. Gegenüber anderen wird hier besonders der Fokus auf die Minimierung von Entwicklungszeiten ohne den Verlust von Nutzerfreiheiten und Flexibilität gelegt. Als meistverwendetes Framework in Industrie und Forschung bietet es zudem eine sehr große und aktive Online-Community.

4.2 Netztyp und Struktur

Im Laufe der Jahre wurden diverse Arten neuronaler Netze entwickelt, um für bestimmte Anwendungsbereiche, durch den Aufbau und die Verschaltung der Neuronen, beim Erlernen von Korrelationen, aufgrund bestimmter Eigenschaften der Daten, Vorteile auszunutzen. Die meistverwendete Netzstruktur bei Forschungen ähnlich zu diesem Thema sind rekurrente neuronale Netze. Besonders bei der Prädiktion von Balltrajektorien und chaotischen Zeitreihen wurden hiermit meist relativ gute Ergebnisse erzielt. Für diese Anwendung müsste für eine Prädiktion des Ergebnisses mittels RNN jedoch neben der Kugeltrajektorie auch die Trajektorie der Drehscheibe prädiziert werden. Dies bietet zwar für die Analyse der Netzperformance eine sehr detallierte Grundlage, erfordert aber auch eine relativ komplexe Auswertung der Netzausgänge, da anhand der Trajektorien bestimmt werden muss, wann sich die Kugelposition relativ zur Postition der Drehscheibe nicht mehr ändert und somit auf einem Feld liegen geblieben ist. Zudem müsste für die Bestimmung des Fehlers ein Bildverarbeitungsprogramm geschrieben werden, welches die gesamte Trajektorie ermittelt. Bei Eintritt in die Chaosphase erfordert dies zudem eine zuverlässige Auswertung von überlappenden Objekten mit einem geringem Kontrast zueinander, sodass rechenaufwendigere Methoden wie Pattern-Matching oder Hough-Transformationen für Kreise verwendet werden müssen, was sich während der Außenwand- und Spiralphase durch einfache Binarisierung und Schwerpunktberechnung realisieren lässt. Genauere Untersuchungen von Forschungsergebnissen haben gezeigt, dass auch ohne chaotisches Verhalten bereits sichtbare Abweichungen bei Prädiktionen von Balltrajektorien erzeugt wurden. Obwohl diese stark von der Qualität der Eingangsdaten abhängig sind, erscheint dieser Ansatz, aufgrund der immensen Fehlerfortpflanzung, durch die chaotischen Eigenschaften des Roulettespiels nur für solche Bereiche vielversprechend, bei welchen eine Prädiktion von sequentiellen Ausgaben zwingend erforderlich ist. Da hier ein statisches Ergebnis prädiziert werden soll, kann die Trajektorie durch das Netz abstrahiert werden. Es werden hier daher nur klassische feedforward-Netze untersucht. Für die Prädiktion anhand von Bilddaten ohne vorgeschaltetes Bildverarbeitungsprogramm gemäß Abschnitt 4.3 wurden zusätzlich entsprechende faltende neuronale Netze verwendet, sodass die Faltungs- und Pooling-Schichten die Aufgabe der Informationsaufbereitung aus den

Bildern übernehmen. Ähnlich zu den Erkenntnissen aus [30] nutzen diese ausschließlich zweidimensionale Faltungen, da die Distanz der Kugel zwischen zwei, als Netzeingang verwendeten, Aufnahmen größer ist als die Abmessungen einer sinnvollen Faltungsmatrix. Somit kann durch dreidimensionale Faltungen, unter Einbezug mehrerer Aufnahmen als dritte Dimension, für die Kugel als wichtigstes Objekt keine bessere Informationsaufbereitung realisiert werden.

4.3 Eingangsdaten

Abgesehen von äußeren Lichteinflüssen, darunter auch vernachlässigbar kleinen Einflüssen, wie durch den Infrarotfilter nicht komplett unterdrückte Wärmestrahlung, sind die Kugel und die Drehscheibe die einzigen visuell erfassbaren veränderlichen Objekte. Solange sich die Testbedingungen, insbesondere die Eigenschaften des Tisches, nicht ändern, beinhalten Ausschnitte aus deren Trajektorien somit alle visuell erfassbaren Informationen, welche für die Prädiktion eines Roulettespiels relevant sind. Aufgrund der starken Abhängigkeit der maximal möglichen Netzperformance von der Qualität der Eingangsdaten, werden hier verschiedene Ansätze betrachtet. Entsprechend den Untersuchungen zur Prädiktion von Roulette mittels Modellbildung aus [19] und [18], lassen sich auch hier Vektordaten beider Objekte über mehrere Frames hinweg verwenden. Durch ein vorgeschaltetes Bildverarbeitungsprogramm zur Ermittlung der Daten tritt unweigerlich, wenn auch in geringer Form, zusätzliches Rauschen auf. Die Vektoren sind, mit vier Skalaren pro Aufnahme, verhältnismäßig klein. Es können somit deutlich mehr Aufnahmen für die Prädiktion berücksichtigt werden, sodass das Rauschen kompensiert werden kann. Zudem ermöglicht die Berücksichtigung der Verläufe über längere Zeiträume, weitere Effekte, wie zum Beispiel trägheitsbedingtes Rutschen der Kugel zu Beginn des Wurfs, einzubeziehen, welche drastische Auswirkungen auf die Prädiktion haben können (siehe Kapitel 5.2). Die Tragweite dieser Einflüsse wird in Kombination mit der Rauschkompensation durch Netzeingänge mit Vektordaten unterschiedlich vieler Aufnahmen untersucht.

Eine netzseitige Auswertung der Bilddaten hat den Vorteil, dass theoretisch eine deutlich robustere und effizientere Lösung gefunden werden kann, welche möglicherweise auch ohne weitere Vorverarbeitungsschritte auskommt. Grundsätzlich wäre es möglich, einem FFNN die Bilddaten direkt zu übergeben. Der Vorteil lokaler Operationen eines faltenden neuronalen Netzes ist für diese Anwendung eher durch die Hervorhebung von Konturen und nicht von Texturen gegeben, welche sich auch von FFNN relativ gut erkennen lassen sollten. Zudem erzeugen vor allem die Pooling-Schichten eines faltenden neuronalen Netzes ein deutlich stärkeres örtliches Rauschen, was für diese Anwendung kritisch Auswirkungen haben kann. Ein FFNN wiederum wäre theoretisch in der Lage, bereits am Netzeingang aus den Bildern und ihrer relativ geringen Informationsdichte relevante Parameter zu selektieren, ohne durch globale Neugestaltungen irrelevanter Bildausschnitte in den Faltungs- und Pooling- Schichten Speicher und Rechenleistung zu verschwenden. Faltende neuronale Netze haben jedoch für diese Untersuchungen den entscheidenden Vorteil, dass die für die Faltungskerne verwendete Anzahl an Gewichten

im Gegensatz zu FFNN nur einen Bruchteil der Größe der Eingangsdaten darstellt. So können auch Bilder mit relativ hohen Auflösungen, wie es hier der Fall ist, verarbeitet werden, ohne das Training mit Matrizen enormer Größe zu behindern. Da hier der Verlauf beweglicher Objekte erfasst werden muss, sind jedoch, anders als bei üblichen Klassifizierungsaufgaben, mehrere Bilder als Netzeingang notwendig. Diese am Eingang separat zu verwenden, hätte den Vorteil, dass alle Information für das Netz erhalten blieben. Ähnlich zu den Erkenntnissen aus [24] wären so auch Erfassungen zeitlich bedingter Faktoren wie Kratzer und Staubablagerungen oder Roulette-Tische verschiedener Größe und Form möglich. Diese sind hier jedoch vermutlich so klein, dass der Nachteil deutlich größerer Eingangsdaten überwiegt. Die Informationsdichte in Bildern ist ohnehin relativ gering. Die Kugel beinhaltet einen Großteil relevanter Informationen, nimmt aber nur einen kleinen Teil des Bildes ein, welcher sich zudem mit jedem Frame ändert und sich zu vorherigen Frames kaum oder gar nicht überschneidet. Bei der Drehscheibe überschneiden sich zwar die Bildausschnitte, es sind aber durch die hohe Anzahl an Konturen auch relativ viele redundante Informationen gegeben. Anders als zu den Ansätzen für diese Arbeit untersuchter Forschungen aus Kapitel 3, werden als Bildeingang daher Differenzbilder verwendet. Neben der Komprimierung der Daten bei Erhaltung aller wichtigen Konturen, hat dies den Vorteil, unbewegliche Objekte der Aufnahmen zu unterdrücken. Während des Trainings wird daher gleich zu Beginn auf die relevanten Teile fokussiert, da der Rest des Bildes, abhängig von zeitlich inkonsistenten Lichteinflüssen, Werte nahe Null annimmt. Besonders durch die Rundung der Kugel und dem damit entstehenden Grauwertverlauf, ist es für eine exakte örtliche Bestimmung wichtig, die Helligkeitsauflösung beizubehalten. Ideal betrachtet müsste sich, abgesehen von visuell nicht erfassbaren Einwirkungen wie dem Drall der Kugel, aus den aktuellen Positionen und Geschwindigkeitsvektoren von Kugel und Drehscheibe der zukünftige Verlauf hinreichend genau prädizieren lassen. Als Netzeingang müssten somit, zumindest theoretisch, nur zwei Aufnahmen berücksichtigt werden, da in diesen alle hierfür notwendigen Informationen enthalten sind. Inwieweit dies tatsächlich der Fall ist, wird in den nachfolgenden Kapiteln genauer untersucht.

Auch die Verwendung eines Differenzbildes ist jedoch an Bedingungen geknüpft. Während Vektordaten ihrer Reihenfolge entsprechend als Netzeingang verwendet werden, geht der zeitliche Bezug bei Differenzbildern verloren. Als Folge müssen Mehrdeutigkeiten bewusst vermieden werden. Da sowohl die Kugel als auch die Drehscheibe ihre jeweilige Richtung relativ zur Kreisbahn nicht ändern, reicht es, hier ein Kriterium zu formulieren, in welcher Relation die Positionen der Kugeln und Drehscheiben jeweils zueinander stehen müssen. Damit eine Eindeutigkeit besteht, darf die Differenz der zurück gelegten Strecke beweglicher Objekte, während des Zeitraumes zwischen beiden verwendeten Aufnahmen bei jeweils maximal und minimal möglicher Geschwindigkeit, höchstens eine halbe Umdrehung/Umrundung betragen. Wenn sich die jeweils gleichen Objekte beider Frames unter dieser Bedingung in einer Halbebene befinden, ist von der anderen Halbebene aus betrachtet immer das rechte bzw. das linke Objekt das zeitlich frühere, was somit vom Netz erlernt werden kann. Allgemein ist es sinnvoll, zwei Aufnamen zu verwenden, welche zeitlich möglichst weit auseinander liegen, um Messungenauigkeiten stärker zu minimieren. Abhängig von den

Geschwindigkeitsbereichen, in welchen sich die Kugel und die Drehscheibe bewegen, wird der maximal erlaubte Abstand jedoch durch dieses Kriterium begrenzt.

4.4 Netzausgang

Der Ausgang der Netze kann grundsätzlich beliebig gewählt werden, solange daraus das richtige Feld extrahiert werden kann. Da nur eine Zahl prädiziert werden soll, ist die naheliegendste Lösung, die Ausgabeschicht auf ein Neuron zu begrenzen. Auf diese Weise ist die prädizierte Differenz zwischen Ist- und Sollwert im Verhältnis zur tatsächlichen, physikalischen Abweichung jedoch nur sehr schwer mathematisch approximierbar, was die Entwicklung einer vernünftigen Kostenfunktion erheblich beeinträchtigt. Eine Neusortierung der Ausgabe, entsprechend der Anordnung auf dem Ziffernkreis, würde zwar einen linearen Bezug zwischen den meisten Abweichungen ermöglichen, aber nicht die kreisförmige Anordnung berücksichtigen, sodass die Entfernung vom 37. zum ersten Feld in einem viel kleineren Verhältnis zur Abweichung stünde als die der Übrigen. Der Netzausgang wurde daher für alle behandelten Netze auf 37 Neuronen, mit einer *Softmax*-Aktivierungsfunktion, festgelegt. Das Ergebnis ist also das one-hot-kodierte Label des Ziffernkreises und erzwingt somit, abgesehen von der Softmax-Funktion, auch keine ungewollten Abhängigkeiten zwischen Feldern. Die Ausgabe lässt sich hierdurch als eine Wahrscheinlichkeitsverteilung betrachten. Zudem können deutlich mehr Informationen für die Bestimmung des Fehlers berücksichtigt werden.

4.5 Kostenfunktion

Für die meisten Anwendungsgebiete genügt der Einsatz konventioneller Kostenfunktionen. Besonders mit dem hier gewählten Netzausgang ist es verlockend die Kreuzentropie zu nutzen, da sich die, für den Backpropagation-Algorithmus notwendige, Ableitung in Kombination mit der Softmax-Funktion zu einer sehr simplen und somit recheneffizienten Formel kürzen lässt. Es kann jedoch von Vorteil sein, bei der Berechnung des Fehlers, abgesehen von einer separaten Betrachtung der Abweichung der Ausgänge, weitere Faktoren zu berücksichtigen. Neben den vorgestellten Kostenfunktionen aus Kapitel 2.1.3 wird daher auch ein proprietärer Ansatz untersucht. Dieser beruht auf der Annahme, dass das Lernverhalten durch eine Fehlerskalierung, abhängig zu vorab definierten Ähnlichkeiten unter den Ausgabeklassen, profitieren kann. Dies wird bei konventionellen Anwendungen nur dadurch berücksichtigt, dass durch Ähnlichkeiten verschwimmende Netzausgaben von der Softmax-Funktion unterdrückt und somit durch die Kostenfunktion weniger bestraft werden. Für eine Prädiktion eines Roulettespiels ist der Satz an Gewichten ideal, welcher die tatsächlichen physikalischen Zusammenhänge zwischen den Eingangsdaten und dem Ergebnis optimal abbildet. Ein Großteil der zu erlernenden Zusammenhänge wird durch die Trajektorie der Kugel beschrieben. Abweichungen in dieser Trajektorie resultieren in einer asymmetrischen Streuung um das "richtige" Feld. Ein Netz, welches also ein Feld daneben eine hohe

Wahrscheinlichkeit ausgibt, hat diese Trajektorie daher vermutlich besser abstrahiert als ein Netz, welches ein weit entferntes Feld prädiziert. Eine Kostenfunktion, welche diese Ähnlichkeiten zwischen verschiedenen Ausgaben entsprechend der "realen" Ähnlichkeit berücksichtigt, also in diesem Fall den Fehler zusätzlich an das Streumuster anpasst, könnte daher einen Gradienten erzeugen, welcher wahrscheinlicher und schneller zu dem globalen Minimum führt.

Der exakte Verlauf der Streuverteilung kann nur durch Simulationen bestimmt werden, welche selbst wiederum auf einem mathematischen Modell dieses Tisches basieren und somit entsprechend den Erkenntnissen aus Kapitel 3 niemals ausreichend genau sein werden. Es wird vereinfachend angekommen, dass der Verlauf mit zunehmender Distanz zwischen dem jeweiligen Feld und dem richtigen streng monoton abnimmt. Da dieser nur geschätzt werden kann, werden verschiedene Parameter eingeführt, um die Form der Kostenfunktion zu steuern. Die Fehlerskalierung $e_{d,\alpha,\delta}$ abhängig zur Distanz, muss den Verlauf auf eine relativ simple und somit recheneffiziente Weise nachbilden:

$$e_{d,\alpha,\delta}(k) = \left(f_\delta\left(\min\left(|k - f_{lf}(y_T)|, 37 - |k - f_{lf}(y_T)|\right)\right)\right)^\alpha \qquad 4.1$$

Mit

k	Feldnummer
y_T	Ziffer des richtigen Feldes
α, δ	frei wählbare Parameter

Zur Berechnung der Distanz zwischen zwei Feldern wird hier die minimale Differenz der Feldnummern entsprechend Abbildung 4.1 unter Berücksichtigung der kreisförmigen Anordnung gebildet. Die Funktion f_{lf} aus Formel 4.1 dient zur Umrechnung in eine Feldnummer über eine Lookup-Tabelle.

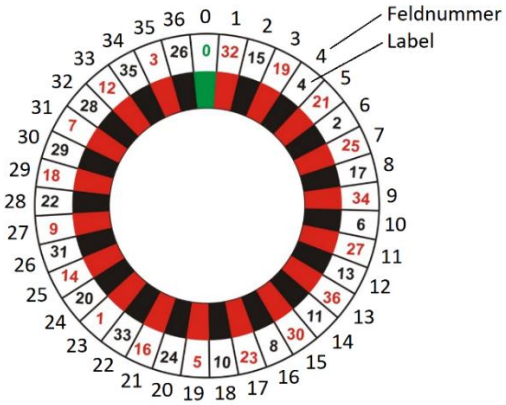


Abbildung 4.1: Umrechnung zwischen Labeln des Ziffernkreises und Feldnummern zur Distanzbestimmung

Anschließend wird die Distanz über die Funktion f_δ skaliert:

$$f_\delta(x) = \begin{cases} 0 & f\ddot{u}r\ x = 0 \\ \left(\frac{x-1}{17} \cdot (\delta - 1)\right) + 1 & f\ddot{u}r\ x \geq 1 \end{cases} \quad 4.2$$

Diese eigentlich unnötig komplexe Skalierung hat den Hintergrund, dass über den Parameter δ auf diese Weise das Verhältnis zwischen der maximalen und der minimalen Entfernung abweichender Felder im Bereich $1 < \delta < \infty$ angegeben werden kann. Wie die Funktion f_{lf}, lässt sich diese Skalierung bereits vor dem Training als fest definierte Lookup-Tabelle implementieren. Die Komplexität ist daher im Bezug auf die Berechnungszeit irrelevant. Das Potenzieren mit α hat, ähnlich zur mittleren quadratischen Abweichung, den Effekt, Distanzen auf nichtlineare Weise zu skalieren. Näher gelegene Felder können somit beispielsweise über mehrere Felder hinweg einen deutlich geringeren Fehler erzeugen, als es weiter entfernte tun. Wählt man hier einen hohen Wert, wird also ein sehr schmales, stark begrenztes Streumuster angenommen und das Netz fokussiert sich hauptsächlich auf das Verkleinern von Fehlern bei Feldern mit großen Distanzen zum richtigen. Die Approximation der Ähnlichkeiten der Ausgabeklassen über die Parameter α und δ ist mit hoher Wahrscheinlichkeit auch bei optimaler Parameterkombination längst nicht ideal. Die Kostenfunktion lässt sich aber nicht in ihrer Gänze in der notwendigen Auflösung vorab berechnen und durch Lookup-Tabellen realisieren und ist somit im Bezug auf Ihre Komplexität an die Berechnungszeit gebunden.

Die Grundstruktur der Kostenfunktion basiert auf einer Kombination des mittleren quadratischen Fehlers und der Kreuzentropie.

$$E_{\alpha,\beta,\gamma,\delta,} = \sum_{k=0}^{36} e_{d,\alpha,\delta}(k) \cdot y_{P,f_{lf}^{-1}(k)}{}^{\beta} + y_{T,f_{lf}^{-1}(k)} \cdot \frac{(1 - y_{P,f_{lf}^{-1}(k)})^{\beta}}{\gamma} \quad 4.3$$

Mit

$y_{P,f_{lf}^{-1}(k)}$	Netzausgabe beim Label mit Feldnummer k (P steht für Predicted)
$y_{T,f_{lf}^{-1}(k)}$	Erzielte Ausgabe beim Label mit Feldnummer k (T steht für True)
α, β, γ, δ	frei wählbare Parameter

Der linke Summand besteht aus dem Produkt des abhängig zur Distanz skalierten Fehlers und der Differenz der zugehörigen Netzausgabe zum entsprechenden Element des Ergebnisvektors. Durch die Distanzskalierung ergibt dieser Summand für das richtige Feld Null und stellt somit den Anteil der Summe der Fehler der falschen Felder dar. Ergänzend dazu wird im rechten Summand der Fehler durch die Abweichung der Netzausgabe des jeweils richtigen Feldes zur Eins bestimmt. Ähnlich zur Berechnung der Kreuzentropie (Formel 2.4) wird dieser Fehler durch die One-Hot-Kodierung des vorgegebenen Ausgabevektors von den anderen separiert. Eine getrennte Berechnung beider Fehler ermöglicht obendrein eine Skalierung des Verhältnisses durch den

Parameter γ. So kann definiert werden, auf welchen Bereich beim Lernen der Fokus gesetzt werden soll. Das Potenzieren der Abweichungen mit dem Parameter β hat, äquivalent zu α, denselben Hintergrund wie bei der Berechnung des mittleren quadratischen Fehlers.
Abbildung 4.2 zeigt die Ausgabe der Kostenfunktion für den Wert eines Neurons, abhängig zur Abweichung vom Zielwert und dessen Distanz zum richtigen Feld. Um den Einfluss der Parametrierung zu verdeutlichen, wurde hier jeweils ein Parameter des mittleren Plots verändert.

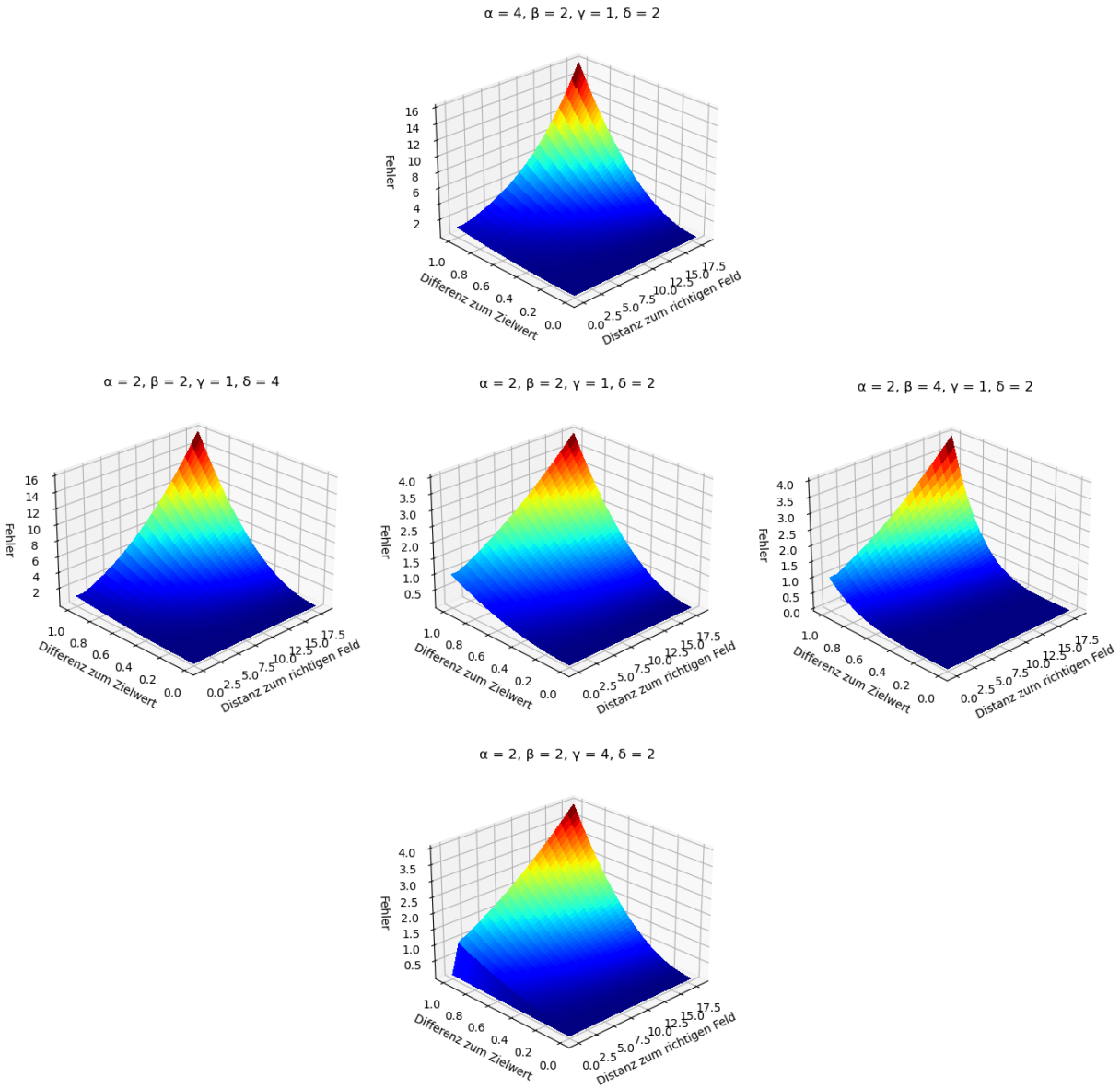


Abbildung 4.2: Visualisierung der Kostenfunktion bei unterschiedlicher Parametrierung

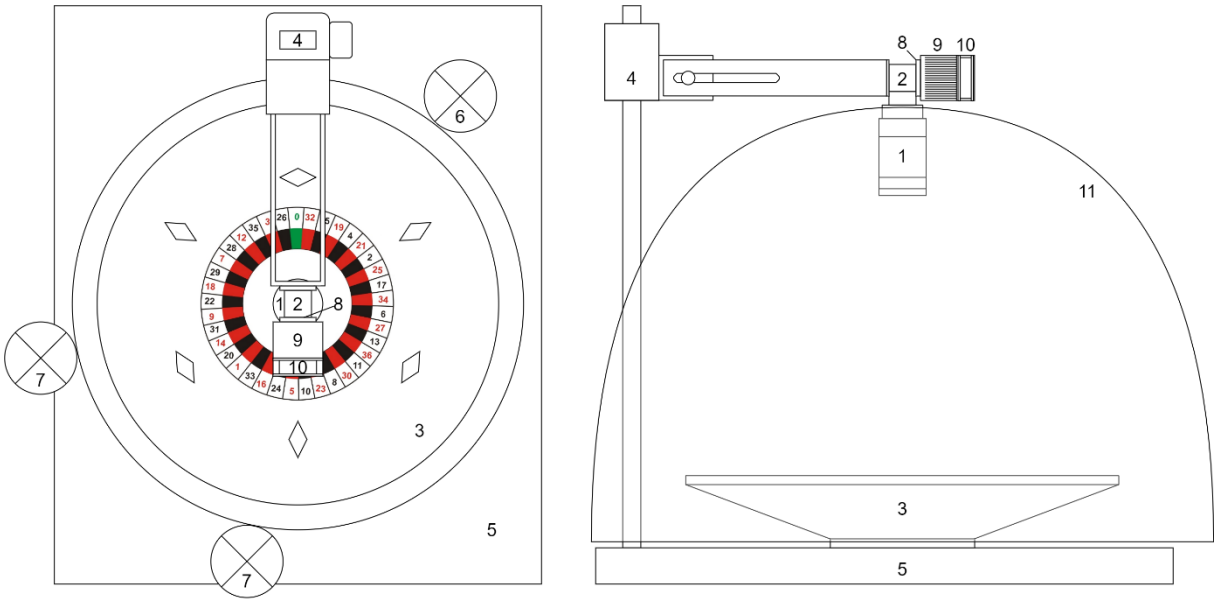
Die Performance der Kostenfunktion gegenüber konventionellen Ansätzen hängt stark von der Parametrierung ab. Diese wird in Kapitel 6.3 durch ausführliche Tests genauer untersucht.

5 Hardware

Aufgrund der hohen Anforderungen durch die Chaotik der Kugeltrajektorie wurde bei der Auswahl und Dimensionierung der Hardware viel Wert darauf gelegt, für die Aufnahme optimale Bedingungen zu schaffen. Hinsichtlich der starken Fehlerfortpflanzung ist es wichtig, relevante Objekte möglichst originalgetreu abbilden zu können. Zudem wird nur eine Perspektive untersucht, sodass die Netze nicht erlernen müssen, die Bilder und Vektoren zu entzerren. Dies ermöglicht außerdem eine relativ simple Vervielfältigung der Datensätze (siehe Kapitel 6.1.4). Entsprechend der Erkenntnisse aus Kapitel 3, hängt das Potential eines Netzes stark von der Qualität der Eingangsdaten ab. Nachfolgend wird untersucht, inwiefern die verfügbare Hardware eine Korrelation zwischen visuell extrahierbaren Daten und dem Ergebnis beeinträchtigt und inwieweit eine Anwendung der konzeptionierten Ansätze sinvoll ist.

5.1 Versuchsaufbau

Abgesehen von anfänglichen Tests zur Bestätigung der Berechnungen in nachfolgenden Unterkapiteln und zur Untersuchung der Aufnahmequalität, wurde für sämtliche Aufnahmen der gleiche Aufbau verwendet.



1 Objektiv
2 Kamera
3 Roulette-Tisch
4 Stativ
5 Stativfuß
6 Baustrahler
7 Halogen-Lampen
8 Peltier-Element
9 Kühlkörper
10 Lüfter
11 Schirm

Abbildung 5.1: Versuchsaufbau, Draufsicht (links), Seitenansicht (rechts)

Die Kamera befindet sich exakt über dem Zentrum des Roulette-Tisches. Ihre Höhe wird durch den Öffnungswinkel, die Abmessungen des Objektivs und die Maße des relevanten Aufnahmebereichs bestimmt. Sie wurde so eingestellt, dass die Bildauflösung des Bereichs, auf dem sich die Kugel bewegen kann, maximal ist. Um exakt diese Perspektive beizubehalten, wurde der Roulette-Tisch mit dem Stativfuß verklebt. Die Leuchten wurden, unter Berücksichtigung ihrer Leuchtstärke, so ausgerichtet, dass der relevante Bereich möglichst gleichmäßig indirekt über den Schirm bestrahlt wird. Der Kühlkörper und das Peltier-Element wurden mit Wärmeleitpaste verbunden und gegen die Kamera gepresst. Der Schirm bedeckt den gesamten Aufnahmebereich und wird unten durch den Stativfuß abgeschlossen. Der gesamte Aufbau steht auf einem Tisch mit gedämpften Füßen, sodass eine eventuelle Schräglage konstant bleibt.

5.2 Roulette-Tisch

Für das Projekt wurde ein Roulette-Tisch der Firma *Renzo Romagnoli* verwendet. Grundsätzlich eignet sich für eine solche Untersuchung jede Art von Zufallsgenerator, solange das Ergebnis anhand der visuell extrahierbaren Daten determinierbar ist. Die genauen Eigenschaften des Roulettespiels, wie dessen Maße oder Reibkoeffizienten, sind für diese Arbeit irrelevant, da sie automatisch auf abstrakte Weise durch das Netz ermittelt werden. Es ist jedoch unabdingbar, dass sich diese Eigenschaften während der Tests nicht ändern, da bereits kleinste Veränderungen der physikalischen Wechselwirkungen zwischen Drehscheibe, Kessel und Kugel enorme Auswirkungen auf die Kugeltrajektorie haben können und sich somit die Qualität der Trainingsdaten erheblich verschlechtern. Die Materialien, aus denen die Komponenten gefertigt sind, sind hauptsächlich Metalle und ein Kunststoff mit dem Markenzeichen Bakelit. Beide sind besonders widerstandsfähig gegen mechanische Einwirkungen und Hitze. Sie sollten daher trotz der schwankenden Temperaturen durch die Beleuchtung kein Problem darstellen. Leider wurden durch die Kugel bei vorherigen Spielen bereits deutlich erkennbare Kratzer im Lack hinterlassen, welche sich im Laufe dieser Arbeit vermutlich in ihrer Anzahl verdoppelt haben. Diese sind in jedem Fall ein Indiz für die Verschlechterung der maximal möglichen Performance der Netze, da diese zeitlich bedingte Veränderungen nicht exakt berücksichtigen können. Zudem haben nähere Untersuchungen Ungenauigkeiten in der Fertigung, wie schief aufgeklebte Ziffern, Unebenheiten im Kesselboden, eine eiförmige Drehscheibe und etwas verschobene Rauten offengelegt. Diese sind jedoch alle ortsabhängig und lassen sich somit problemlos vom Netz berücksichtigen. Nachfolgend wird untersucht, inwieweit die in Kapitel 4.2 definierten Netzeingänge ausreichende Informationen für eine Prädiktion eines Spiels mit diesem Roulette-Tisch zur Verfügung stellen. Hierfür wurden 20 Kugelwürfe bis zum Erreichen der Rauten aufgenommen und die Koordinaten der Kugel und der Drehscheibe aus jedem Frame gemäß den Spezifikationen aus Kapitel 4.2 extrahiert. Die hieraus erzeugten Plots wurden nach bestem Ermessen übereinandergelegt, sodass nicht prädizierbare Abweichungen verdeutlicht werden. Es ist hier wichtig zu erwähnen, dass diese Plots nicht exakt sind. Neben sehr kleinen aber existenten Ungenauigkeiten in der Bildverarbeitung, beispielsweise durch veränderliche Konturen bei der Binarisierung durch ortsabhängige Beleuchtungsintensitäten, könnten hier auch Abbildungsfehler durch das Objektiv deutlich werden (siehe Kapitel 5.5.3).

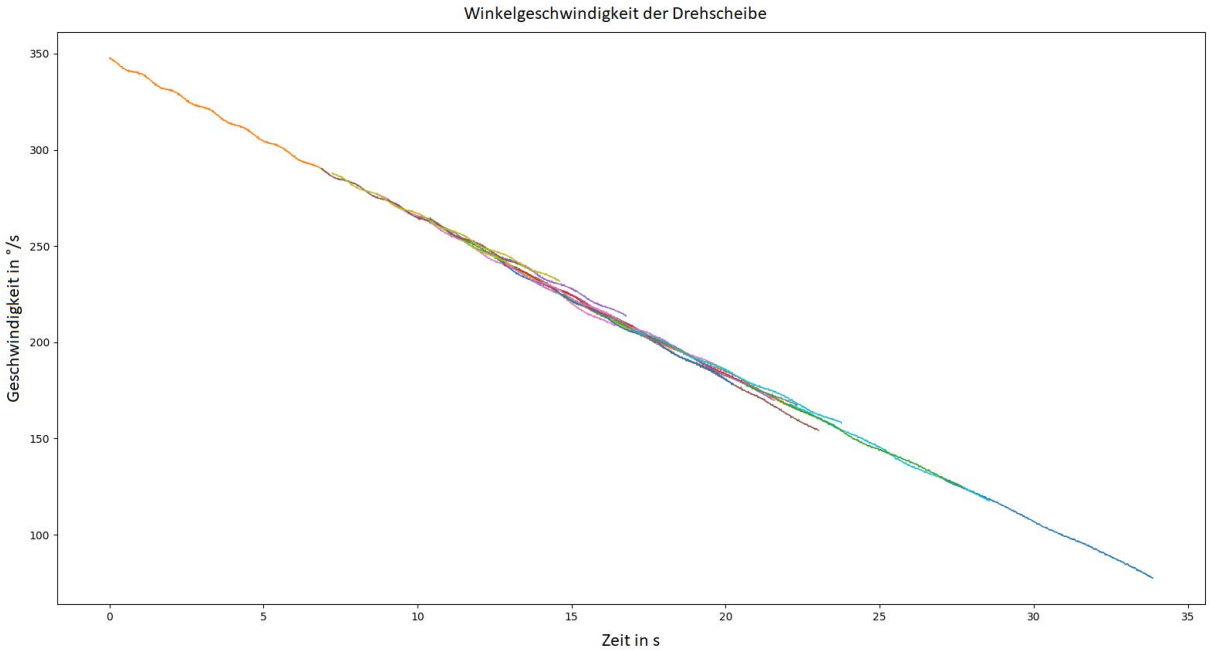


Abbildung 5.2: Beispielhafte Verläufe der Winkelgeschwindigkeiten der Drehscheibe möglichst deckungsgleich übereinandergelegt

Bis auf periodische Schwankungen sind die Geschwindigkeitsverläufe der Drehscheibe relativ linear. Die Amplitude dieser Schwingungen in Abhängigkeit zur Geschwindigkeit lässt vermuten, dass es sich hierbei um die Reibung an dem Lager handelt. Aus der Periodendauer und der aktuellen Winkelgeschwindigkeit lässt sich ableiten, dass die Reibung von der aktuellen Ausrichtung der Drehscheibe abhängig ist. Diese wird bei allen in Kapitel 4.2 spezifizierten Netzeingängen mitgeliefert und lässt sich somit durch das Netz berücksichtigen. Für Netzeingänge in Bild-Form war der ursprüngliche Gedanke, nur zwei Frames zu berücksichtigen. Unabhängig von dem zeitlichen Abstand der Frames, bilden diese nur einen Punkt auf dem durch den Abstand definierten Geschwindigkeitsverlauf ab. Damit der zukünftige Verlauf dennoch prädizierbar ist, muss es möglich sein, die Verläufe in den Abbildungen dieses Abschnitts, abgesehen von positionsabhängigen Einflüssen, deckungsgleich aufeinanderlegen zu können. In Abbildung 5.2 sind Abweichungen von bis zu $4\frac{°/s}{7s} = 0{,}57\,°/s^2$ zu sehen. Abhängig von der Anfangsgeschwindigkeit der Kugel dauert es bis zu zwölf Sekunden bis zum Erreichen des finalen Feldes. Während dieser Zeit können also nicht prädizierbare Abweichungen des Drehscheibenwinkels von bis zu:

$$\int_{0\,s}^{12\,s} 0{,}57\,°/s^2 \cdot t\,dt = \frac{0{,}57}{2}\,°/s^2 \cdot (12\,s)^2 = 41\,° \qquad 5.1$$

entstehen. Ein Feld belegt $\frac{360\,°}{37} = 9{,}73\,°$ der Drehscheibe. Eine Abweichung von bis zu 41° entspricht also 4,2 Feldern allein für die Prädiktion der Ausrichtung der Drehscheibe. Dies ist zwar das worst-case Szenario dieser Messung, welche möglicherweise auch durch Messungenauigkeiten verfälscht wird, es verdeutlicht aber dennoch, dass sogar das Verhalten der Drehscheibe nicht hinreichend durch zwei Frames erfasst werden kann. Für einen Netzeingang in Bildform müssen daher entgegen der ursprünglichen Konzeption

mindestens drei Frames berücksichtigt werden, damit auch die positionsunabhängige negative Beschleuningung der Drehscheibe vom Netz erfasst wird. Ähnliche Abweichungen lassen sich auch bei den Verläufen der Kugel beobachten.

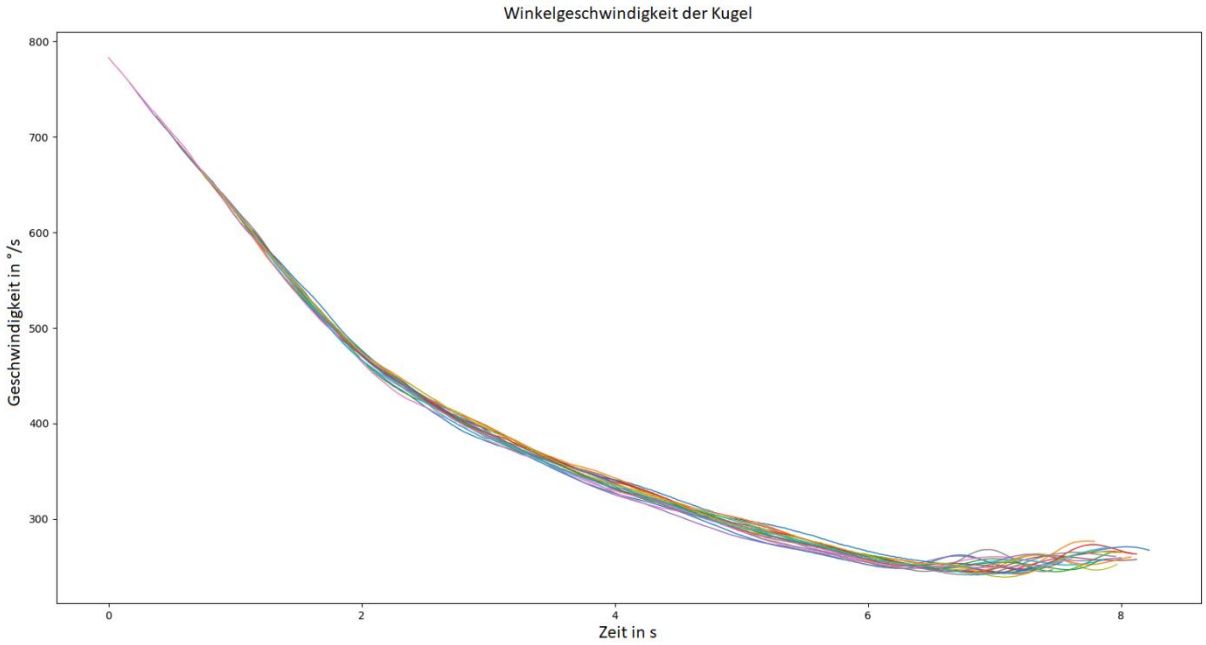


Abbildung 5.3: Beispielhafte Verläufe der Winkelgeschwindigkeiten der Kugel möglichst deckungsgleich übereinandergelegt

Nach dem Verlassen der Außenwand, etwa ab Sekunde 6,5, sind hier typische Schwankungen für eine Bewegung in einem Zentralpotential zu sehen, durch welche die Winkelgeschwindigkeit teilweise sogar zunimmt. Die Kugel rollt hier nicht auf einer idealen Spiralbahn, sondern eher in Form einer Elipse. Verschiedene Geschwindigkeiten beim Verlassen der Außenwand lassen sich auch auf positionsabhängige Einflüsse zurückführen. Es sind auch geschwindigkeitsabhängige Einflüsse, etwa ab 450 °/s, zu sehen, durch welche ein relativ linearer Verlauf in eine Krümmung übergeht. Auch diese behindern nicht die Korrelation zwischen Netzeingang und Ergebnis. Es werden dennoch größere Geschwindigkeitsabweichungen deutlich, welche bereits vor der Chaosphase in noch viel größeren Abweichungen der Positionen resultieren. Inwieweit diese ortsabhängig oder durch optisch nicht erfassbare Einflüsse verursacht werden, lässt sich nicht genau sagen. Es wird aber deutlich, dass bereits sehr kleine Unterschiede, wie die Unebenheiten einer scheinbar ebenen Oberfläche oder die Inkonsistenzen des mittleren Reibwiderstandes der Drehscheibe bereits vor Beginn der Chaosphase sehr große Abweichungen erzeugen können. Eine Prädiktion des Kugelwurfes noch während der Außenwandphase scheint daher besonders mit Netzeingängen, die nur wenige Zustände definieren, auch mit einem Netz welches alle Korrelationen zwischen den optisch erfassbaren Informationen und dem Ergebnis perfekt berücksicht, eher unwahrscheinlich. Neben der Tatsache des begrenzten Speichers werden Netzeingänge in Bildform daher nur für die Spiralphase untersucht.

5.3 Beleuchtung

Die Wahl der Beleuchtungsart hängt von der Oberflächenstruktur und dem erzielten Effekt ab. In diesem Fall sollen matte und glänzende Objekte unterschiedlicher Struktur gleichmäßg hervorgehoben werden. Der Fokus liegt darauf, relevante Kontraste bei der Kugel und dem Ziffernkreis zu verstärken. Es ist wichtig, dass keine Lichtquellen über den Kesselboden in das Objektiv reflektiert werden. Die dadurch entstehenden Lichtflecken würden die Kontur der ebenfalls weißen Kugel eliminieren. Es wurde daher eine Dombeleuchtung realisiert. Diese erzeugt sehr diffuses und homogenes Licht, welches strukturierte Objekte wie die Kugel und auch glänzende Oberflächen wie den Ziffernkreis gleichmäßig hervorhebt. Auf diese Weise werden auch Fehler während der Bestimmung der Koordinaten minimiert, was in weniger verrauschten Vektordaten resultiert. In Abbildung 6.2 aus Kapitel 6.1.2 lässt sich gut erkennen, wie diese Beleuchtung die Aufnahme, insbesondere in der Außenwand- und Spiralphase, begünstigt. Zudem minimiert der Schirm auch äußere Einflüsse wie Sonnenlicht oder Luftströmungen.

Stärkeres Rauschen kann auch bei Netzeingängen in Bildform das Lernen behindern, da zum einen für die Prädiktion relevante Korrelationen geschwächt werden und zum anderen ungewollte Korrelationen zwischen dem Rauschen und dem Ergebnis entstehen (vgl. [5]). Der Rauschanteil eines Bildes wird nach EMVA 1288 [38] durch die Summe aus dem Halbleiterrauschen der Kamera, dem Quantisierungsrauschen bei der Digitalisierung und dem Lichtrauschen bestimmt. Um diesen zu minimieren, muss das Verhältnis zwischen Eingangssignal und Rauschen maximiert werden. Erhöht man die Beleuchtungsstärke, dominiert nach EMVA 1288 das Lichtrauschen gegenüber dem Rauschen der Kamera und das Verhältnis nähert sich von unten der Funktion

$$SNR_{optimal} = \sqrt{\eta \cdot \mu_P} \qquad 5.2$$

Mit

η Quanteneffizienz der Kamera (Verhältnis erzeugter Ladungsträgerpaare pro eintreffenden Photonen)

μ_P Erwartungswert eintreffender Photonen auf den Sensor

Einen minimalen Rauschanteil erhält man somit bei der maximal möglichen Beleuchtungsstärke, bevor der Kamerasensor in die Sättigung geht. Es wurden daher alle verfügbaren Lichtquellen verwendet, welche sich für diesen Testaufbau eigneten.

Tabelle 5.1: verfügbare relevante Lichtquellen

Art	Lichtstrom
Halogen-Lampe	370 Lumen
Halogen-Lampe	630 Lumen
LED-Baustrahler	5000 Lumen

Auch wenn Leuchtdioden das Farbspektrum deutlich schlechter abbilden als es beispielsweise Halogenlampen tun, setzen sie sich zunehmend in der Bildverarbeitung durch, da sie überlastbar sind, eine deutlich längere Lebensdauer haben, nicht flackern und eine farbgetreue Abbildung meist unwichtig ist. Leider ist der Baustrahler nicht für die Bildverarbeitung ausgelegt. Zum Dimmen wird hier ein PWM-Signal eingesetzt, was auch auf der höchsten Einstellung sehr starke stroboskopische Effekte erzeugt.
In Kombination mit dem Flackern der Halogenlampen wurden so, bei einer Auflösung von 8 Bit, trotz einer vorwiegend schwarzen Oberfläche im Mittel Schwingungen von etwa 30 Helligkeitswerten gemessen. Diese lassen sich von neuronalen Netzen grundsätzlich kompensieren. Dennoch erzeugen Daten mit unterschiedlich stark ausgeprägten Werten gleicher Relevanz auch unterschiedlich starke Änderungen der Gewichte (vgl. [5]), sodass hellere Bilder im Allgemeinen beim Lernen stärker berücksichtigt werden, weil das hier erzeugte Fehlermaß δ für Neuronen der Eingabeschicht nach Formel 2.7 mit größeren Eingangswerten multipliziert wird. Zudem erschweren stroboskopische Effekte die Entwicklung der Bildverarbeitungssoftware zur Generierung der Vektor- und Diagnosedaten, weshalb sie sowohl für die Bildverarbeitung als auch für die Netzeingänge in Bildform, vorab softwareseitig auf etwa 1 minimiert wurden.

5.4 Kamera

Die Qualität der Trainingsdaten und damit das Leistungspotential der Netze wird sehr von der Charakteristik der Kamera geprägt. Die relevanten Daten der verfügbaren Kameras sind in Tabelle 5.2 gegenübergestellt.

Tabelle 5.2: Details verfügbarer Kameras

Kamera	BASLER acA1920-40uc	BASLER acA2040-90uc
max. Sensorauflösung	1936 x 1216	2040 x 2046
Sensorgröße	11,3 mm x 7,1 mm	11,3 x 11,3
Pixelmaße	5,86 µm x 5,86 µm	5,5 µm x 5,5 µm
SNR	45 dB	40,8 dB
Dynamikbereich	74	58,7
Quanteneffizienz	70 %	62,1 %
Farbe/Monochrom	Farbe	Farbe
Sensortyp	CMOS	CMOS
Schnittstelle	USB 3.0	USB 3.0
Verschlusstechnik	global shutter	global shutter
Bildwiederholrate	41 fps	90 fps

Die Pixelgröße, das Signal-Rausch-Verhältnis und die Quanteneffizienz des Sensors weisen darauf hin, dass die Kamera *acA1920-40uc* ein deutlich besseres Rauschverhalten aufweist. Für dieses Anwendungsgebiet liegt der Fokus bei der Auswahl jedoch vorwiegend auf der Auflösung des Sensors. Diese bestimmt das örtliche Quantisierungsrauschen beweglicher Objekte und hat somit, besonders bei

Netzeingängen, welche nur wenige Frames berücksichtigen, durch die Fehlerfortpflanzung einen viel größeren Einfluss als das stärkere Bildrauschen. Darüber hinaus lassen sich mit der Kamera *acA2040-90uc* mehr als die doppelte Anzahl an Bildern pro Sekunde generieren. Dies ermöglicht eine deutlich genauere zeitliche Auflösung der Trajektorien und somit auch mehr als die doppelte Menge an gewonnenen Trainingsdaten pro Wurf. Nachfolgend wird daher ausschließlich die Kamera acA2040-90uc genauer untersucht.

5.4.1 Farbtreue

Der Farbfilter der Kamera entspricht dem Bayer-Format. Ein Demosaicing würde den Speicherbedarf verdreifachen, ohne zusätzlich relevante Informationen zu generieren. Die Frames werden daher als Rohdaten ausgelesen und nur für spezielle Anwendungen wie bei der Erzeugung der Vektordaten oder der Datenvervielfältigung in separate Farbkanäle konvertiert. Eine originalgetreue Farbwahrnehmung ist für diesen Versuch irrelevant. Es ist jedoch vorteilhaft, einen Weißabgleich durchzuführen. Ansonsten erscheint der Blaukanal durch das leicht gelbliche Licht dunkler, sodass Schwarz-Weiß-Übergänge im Bayer-Format ausfransen und somit die Konturen verwischen (siehe Anhang A).

5.4.2 Kontinuität der Aufnahme

Das Netz stellt zwischen den Positionsdaten einzelner Frames den Bezug zur Geschwindigkeit beweglicher Objekte her. Inkonsistenzen in den Zeitabständen der einzelnen Frames würden den Zusammenhang zwischen der Differenz der Positionsdaten und der Geschwindigkeit der Objekte stören. Wie in Kapitel 5.2 untersucht, ist die Fehlerfortpflanzung durch Abweichungen der Geschwindigkeiten für die Prädiktion fatal. Die Frames werden daher asynchron aufgenommen und im Speicher der Kamera gepuffert. Mit den verfügbaren Messinstrumenten konnten auch nach längeren Tests keine Inkonsistenzen und auch keine verlorenen Frames festgestellt werden, welche nicht von den Fehlerprüfroutinen der entwickelten Software abgefangen wurden.

5.4.3 Belichtungszeit, Verstärkung Gammakorrektur und Kontrastspreizung

Sowohl das sogenannte Dunkelrauschen, hervorgerufen durch thermisch induzierte Elektronen in der Kameraelektronik, als auch das erfasste Lichtrauschen, aufgrund der Ähnlichkeit des Photonenstroms zu einer Poisson-Verteilung, werden durch die Belichtungszeit beeinflusst. Grundsätzlich nimmt das Rauschen bei längerer Belichtung zu. Im Verhältnis zum Erwartungswert nimmt es jedoch ab. Bei der Wahl der

Belichtungszeit muss daher ein Kompromiss zwischen dem Rauschanteil des Bildes und den Bewegungsunschärfen der Objekte gefunden werden.

Ungenauigkeiten durch Unschärfeeffekte sind besonders bei Netzeingängen in Bildform aus Kapitel 4.3 wichtig, da hier nur wenige Informationen zur Verfügung stehen, durch welche sich Abweichungen vom Netz berücksichtigen lassen. Bewegungsunschärfen haben einerseits zur Folge, dass Konturen verwischen, was besonders in Kombination mit Rauschen zu stärker abweichenden Schwerpunkten führen kann, andererseits verschiebt sich der Schwerpunkt in Abhängigkeit zur Geschwindigkeitsänderung während der Belichtung. Letzteres ist im Verhältnis aber sehr gering und kann grundsätzlich durch das Netz berücksichtigt werden. In Relation zum Rauschen ist der Einfluss auf die Netze daher vermutlich relativ klein.

Die maximale Geschwindigkeit des schnellsten Objektes (der Kugel) von etwa 65 cm/s tritt bei Netzeingängen in Bildform zum Zeitpunkt des Verlassens der Außenwandphase auf. Bei einer Auflösung von 2040 Pixeln über eine Länge von 295 mm benötigt die Kugel mindestens 222 µs, um die Strecke, die durch einen Pixel abgebildet wird, zurückzulegen. Belichtungszeiten in diesem Bereich erzeugen trotz der Beleuchtung stark verrauschte Bilder. Tests haben ergeben, dass die Aufnahmen erst bei einer Belichtungszeit von etwa 1000 µs ausreichend wenig verrauscht waren. Die Konturen verwischen hier bis zu 2,25 Pixel beziehungsweise 0,33 mm. Abweichungen, welche aus diesen Unschärfen entstehen, sind daher tolerierbar.

Eine kameraseitige Verstärkung und Gamma-Korrektur wurden hier vermieden, da sie für die Netze keine Vorteile liefern und Informationsverluste verursachen können. Die Aufnahmen werden jedoch, zusätzlich zur Flackerkorrektur aus Kapitel 5.3, einer Kontrastspreizung unterzogen. Diese dient lediglich zur Vereinfachung der Entwicklung der Bildverarbeitungssoftware und der Visualisierung von Testergebnissen.

5.4.4 Pixelfehler

Für die Untersuchung des Prädiktionspotentials der Netze muss auch überprüft werden, inwieweit der Sensor und die Ausleseelektronik fehlerfrei funktionieren. Tests haben ergeben, dass keine toten Pixel exisistieren. Allerdings wurden, auch bei der Aufnahme einer weißen Oberfläche, teilweise starke Abweichungen vom Mittelwert des Umfeldes gemessen. Informationsverluste und Rauschen sind bis zu einem gewissen Grad unvermeidlich. Es wurde jedoch deutlich, dass thermisch bedingtes Rauschen die Bildqualität in manchen Bereichen immens verschlechtert hat. Um den Rauschanteil der Kamera so gering wie möglich zu halten, wurde daher eine Kühlung durch ein Peltier-Element realisiert. Sichtbare verrauschte Streifen auf den Aufnahmen konnten so komplett eliminiert werden. Messergebnisse zu diesen Untersuchungen befinden sich im Anhang A.

5.5 Objektiv

Die hohe Auflösung der Kamera kann nur vollständig ausgenutzt werden, wenn das Objektiv die Lichtstrahlen optimal in der Ebene des Sensors bündelt. Nachfolgend sind die relevanten Daten aller verfügbaren Objektive gegenübergestellt.

Tabelle 5.3: Details verfügbarer Objektive

Objektiv	PENTAX H1214-M	PENTAX H612A	Kowa LM12SC
Festbrennweite	12 mm	6 mm	12 mm
Blendenzahlen	1,4; 2; 4; 8; 16	1,2; 2; 2,8; 4; 5,6; 8; 11; 16	1,8; 2; 2,8; 4; 5,6; 16
max. Bildkreis	1/2 "	1/2 "	1 "
min. Objektdistanz	0,25 m	0,2 m	0,1 m

Leider bildet nur das Objektiv von Kowa die Fläche auf einem Bildkreis ab, der groß genug ist um keine Vignettierungen zu erzeugen. Inwiefern sich dieses für die Anwendung und die ausgewählte Kamera eignet, wird in den folgenden Unterkapiteln genauer erläutert.

5.5.1 Tiefenschärfe

Weicht die Distanz eines Objektpunktes von der eingestellten Gegenstandsweite g ab, verwischt dieser auf dem Sensor zu einer Unschärfescheibe mit dem Durchmesser ε. Für kleine Pixelgrößen (wenige μm) und geringe Gegenstandsweiten (kleiner als 1 m), wie es hier der Fall ist, lässt sich der zulässige Entfernungsunterschied Δg zwischen Linse und Gegenstand näherungsweise wie folgt berechnen (vgl. [39]):

$$\Delta g_{fern,nah} \approx k \cdot \frac{1 - \frac{f}{g}}{\left(\frac{f}{g}\right)^2} \cdot \varepsilon_{max} \qquad 5.3$$

Tabelle 5.4 zeigt $\Delta g_{fern,nah}$ für die BASLER Kamera *acA2040-90uc* in Kombination mit dem Kowa Objektiv *LM12SC* für alle einstellbaren Blendenzahlen. Der Parameter ε_{max} wird hier gleich der Seitenlänge eines Pixels auf dem Sensor gesetzt, sodass zwei Bildpunkte mit dem Abstand eines Pixels in jedem Fall unterschieden werden können.

Tabelle 5.4: Tiefenschärfen bei unterschiedlichen Blendenzahlen und optimaler Gegenstandsweite mit dem Objektiv *Kowa LM12SC*

Blendenzahl k	1,8	2	2,8	4	5,6	16
$\Delta g_{fern,nah}$ [mm]	7,0	7,8	10,9	15,6	21,8	62,3

In einem optimalen Bild sollten alle relevanten Bereiche scharf abgebildet werden. Die maximale Differenz von 35 mm zwischen den relevanten Gegenstandsweiten wird durch den Außenrand der Kugel während der Außenwandphase und durch den Ziffernkreis der

Drehscheibe bestimmt. $\Delta g_{fern,nah}$ muss also $\frac{35\,mm}{2} = 17{,}5\,mm$ überschreiten, wenn die Mitte beider Entfernungen fokussiert wird. In Tabelle 5.4 sind alle Werte grün hinterlegt, welche dieses Kriterium erfüllen. Grundsätzlich sollte die Blendenzahl so klein wie möglich gewählt werden, da sich dieser Parameter antiproportional zum Durchmesser der Eintrittspupille verhält. Je geringer der Durchmesser ist, desto weniger Photonen treffen auf den Sensor, was zu einem kleineren SNR, also zu einem größeren Rauschanteil führt.

5.5.2 Beugungsbegrenzte Auflösung

Auch bei einem idealen Objektiv, welches keine Abbildungsfehler erzeugt, verschlechtern dennoch Beugungseffekte durch die Welleneigenschaften des Lichts die Bildqualität. Jeder Lichtpunkt wird auf dem Sensor als Interferenzmuster abgebildet, dessen Hauptmaxima die Größe eines Pixels überschreiten kann. Damit dieser Effekt durch seine Unschärfen nicht das maximale Auflösungsvermögen der Kamera beeinträchtigt, müssen zwei Objektpunkte mit dem Abstand eines Pixels im Bild noch unterscheidbar sein. Entsprechend dem Rayleigh-Kriterium berechnet sich der Radius des Hauptmaximums zum ersten Minimum, auch Airy-Scheibchen genannt, in Abhängigkeit der Wellenlänge λ und der Blendenzahl k wie folgt:

$$r = 1{,}22 \cdot \lambda \cdot k \qquad 5.4$$

Die größten Airy-Scheibchen treten also bei der maximal erfassbaren Wellenlänge auf. Laut Datenblatt der Kamera wird diese durch den Infrarotsperrfilter etwa bei $\lambda_{max} = 900$ nm begrenzt. Tabelle 5.5 zeigt die Radien der größtmöglichen Airy-Scheibchen in µm für die BASLER Kamera *acA2040-90uc* in Kombination mit dem Kowa Objektiv *LM12SC*.

Tabelle 5.5: Radien der Airy-Scheibchen bei einer Wellenlänge von 900 nm

Blendenzahl k	1,8	2	2,8	4	5,6	16
Radius r [µm]	2,0	2,2	3,0	4,4	6,1	17,6

Damit zwei Airy-Scheibchen voneinander unterscheidbar sind, müssen ihre Zentren nach dem Rayleigh-Kriterium mindestens um ihren Radius voneinander entfernt sein. Blendenzahlen bei welchen die Radien der Airy-Scheibchen die Abmessungen eines Pixels des jeweiligen Sensors überschreiten sind daher nicht sinvoll und wurden rot hinterlegt.

Vergleicht man Tabelle 5.4 und Tabelle 5.5, ergibt sich für die Einstellung der Blendenzahl keine Schnittmenge. Es ist aber tolerierbar, wenn der Ziffernkreis nicht komplett scharf abgebildet wird, da durch die Konturen der Ziffern sehr viele Informationen zur Positionsbestimmung zur Verfügung stehen. Eine Blendenzahl von k = 4 stellt daher einen guten Kompromiss dar.

5.5.3 Abbildungsfehler

Zur Untersuchung auf Abbildungsfehler wurden verschiedene Testbilder ausgedruckt und aufgenommen. Die Aufnahmen wurden auf folgende Abbildungsfehler untersucht:

- sphärische Aberrationen
- Koma-Effekte
- Astigmatismen
- Bildfeldwölbungen
- Verzeichnungen
- chromatische Aberrationen

Bis auf eine tonnenförmige Verzeichnung konnten keine Abbildungsfehler visuell erkannt werden. Ähnlich zu Aufnahmen aus verschiedenen Perspektiven, lässt sich der Verzeichnungseffekt durch das Netz problemlos auf abstrakte Weise entzerren. Grundsätzlich entsteht ein leichter Informationsverlust, da Positionsdaten bei gleicher Auflösung in den Randbereichen gestaucht werden, sodass diese nach einer Entzerrung durch die Quantisierung stärker verrauscht sind. Dieser Effekt ist hier aber so gering, dass er für die Beurteilung der Netzperformance vernachlässigt werden kann.

5.6 Ressourcen zur Datenverarbeitung

Für die Entwicklung der Software, diverser Tests und für das Training der Netze müssen Datensätze im Terrabyte-Bereich erstellt, getestet und verarbeitet werden. Neben dem Entwicklungscomputer stand hierfür ein Server mit entsprechenden Ressourcen zur Verfügung. Die Spezifikationen beider Computer sind in Tabelle 5.6 aufgelistet.

Tabelle 5.6: Spezifikationen verfügbarer Hardwareressourcen

	Server	Entwicklungscomputer
System	64-Bit Linux/Ubuntu	64-Bit Windows 10 Pro
CPU	4x GeForce RTX 2080 Ti	1x GeForce RTX 2070
GPU	14x Intel i9-7940X getaktet auf 3.10 GHz	1x AMD Ryzen 7-3700X getaktet auf 3,60 GHz
RAM	64 GB	16 GB
Festplatte	469 GB	638 GB
Raid System	7,3 TB	nicht vorhanden

6 Entwicklung, Training und Test

Nach Abschluss der Voruntersuchungen zur Hardware erfolgt die Erzeugung der Trainingsdaten und die Entwicklung neuronaler Netze. Entsprechend den Ergebnissen aus dem vorherigen Kapitel und der Konzeption, wird hier systematisch untersucht, welche Kombination aus den vorgestellten Verfahren und Hyperparametern das Erlernen der kamerabasierten Prädiktion eines chaotischen Systems, wie einem Roulettespiel, optimal unterstützt. In den nachfolgenden Abschnitten wird ein Überblick über die Gedankengänge und Entscheidungen innerhalb der einzelnen Arbeitspakete gegeben.

6.1 Erzeugung der Trainingsdaten

Insgesamt wurden 761 Videos als Trainingsdaten und 50 Videos als Validierungsdaten aufgenommen. Diese wurden bewusst separiert, da eine Aufteilung eines Videos auf beide Datensätze nach den Erkenntnissen aus Kapitel 7.2 fälschlich gute Ergebnisse erzeugen würde. Um eine Aussage darüber treffen zu können, ob und in welcher Form sich bestimmte Verfahren wie die der Datenaugmentierung eignen, wie bestimmte Parameter, darunter die Größe des Eingangsvektors, gewählt werden sollten und inwieweit diese Datenmenge ausreicht, um das Verhalten des Roulettespiels abzubilden, müssen diese Videos auf unterschiedliche Weise aufbereitet und verarbeitet werden. Um den Rechenaufwand möglichst gering zu halten, wurde hier ein modularer Ansatz gewählt. Die Erzeugung der Trainingsdaten wurde somit gemäß Abbildung 6.1 als Softwarepaket realisiert. Diese greifen untereinander auf gemeinsame Datensätze zu, welche verschiedene Stadien der Informationsaufbereitung und -verarbeitung repräsentieren. Die Struktur orientiert sich daran, den Speicherbedarf und die Berechnungszeiten möglichst gering zu halten. Programme wie die zur Vervielfältigung oder dem Zuschnitt der Daten, welche zu erheblich größeren Datenmengen führen, werden somit erst gegen Ende der Arbeitskette eingesetzt. Je nach Anwendungsgebiet werden zudem weitere Programme genutzt, welche beispielsweise eine feste Anzahl zufällig ausgewählter Daten kopieren, Kennzeichnungen rückgängig machen, Datensätze zusammenführen, oder zusätzliche Veränderungen, wie das Skalieren der Auflösung, einbauen.

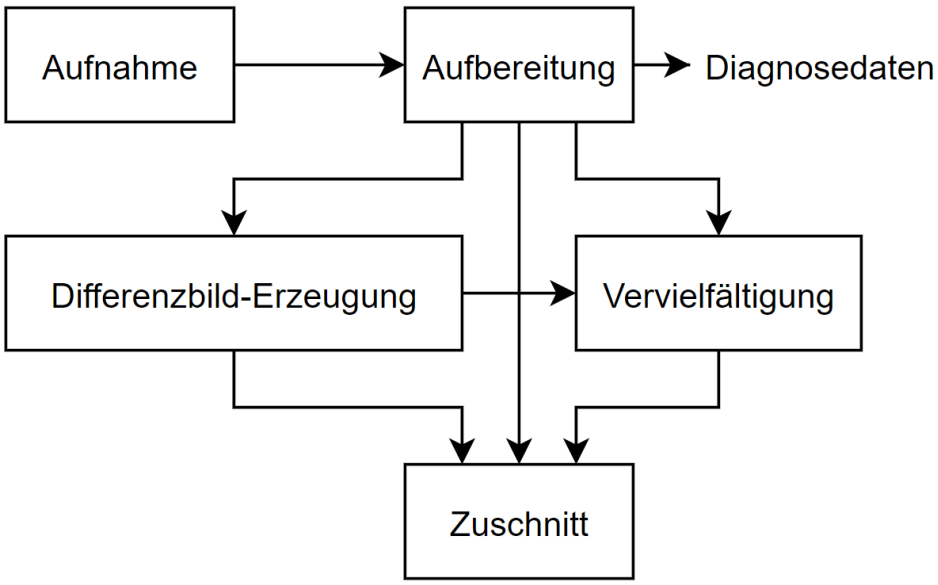


Abbildung 6.1: Überblick über die Module zur Erzeugung der Datensätze

Jedes Programm greift auf ein separates Quell- und Zielverzeichnis zu. Videodaten werden im AVI-Format in unkomprimierter Form gespeichert, Vektordaten als Textdatei entsprechend dem CSV-Format. Alternativ wäre auch eine Implementierung mit Bibliotheken ähnlich zu SQL möglich. Die meisten der damit verbundenen Vorteile finden hier jedoch keine Anwendung, da der gesamte Datensatz grundsätzlich nur von einem Programm zur Zeit verwendet wird, sodass der damit verbundene Entwicklungsaufwand und die geringere Datennähe die Vorteile überwiegen. Die Benennung der Daten erfolgt konsequent nach dem gleichen Schema:

Klasse_Nummer.txt bzw. *Klasse_Nummer.avi*

Wurde eine Datei von einem Programm verarbeitet, wird dessen Bezeichnung um den Präfix *bearbeitet_* erweitert. Datensätze können somit problemlos dynamisch erweitert und Programme mehrfach ausgeführt werden, ohne Konflikte zu erzeugen. Da auf zwei separaten Systemen gearbeitet wurde, welche zudem über längere Zeiträume unbeaufsichtigt große Datenmengen verarbeitet haben, wurde in jeder Software eine Logging-Routine implementiert. Durch diese werden Statusinformationen, Warnungen und Fehler mit zugehörigem Zeitstempel und Softwarekennzeichnung in einer gemeinsamen Logdatei festgehalten.

Gemäß Abbildung 6.1 ist eine Separierung der Aufnahme und Aufbereitung der Videos im Bezug auf die damit verbundenen Freiheiten zur Datenaugmentierung nicht nötig. Bei einer Auflösung von 2040 x 2040 Pixel und einer Framerate von 90 fps werden jedoch pro Sekunde etwa 374 MB an Daten generiert. Eine Verarbeitung solcher Datenmengen erfordert eine Wartezeit, welche sich so während der Aufnahmephase vermeiden ließ. Abhängig von dem jeweiligen Test und verwendeten Dateiformat, wurden die Daten in

der dargestellten Reihenfolge weiter verarbeitet. In den nachfolgenden Abschnitten wird ein Überblick über die wichtigsten Programme zur Erzeugung der Trainingsdaten und deren Ablauf gegeben. Eine genaue Auflistung der gesamten in dieser Arbeit entwickelten Software befindet sich im Anhang E.

6.1.1 Videoaufnahme

Die Aufnahme des Roulettespiels erfolgt über das Programm *record_video.py*. Um zu gewährleisten, dass durch andere Software, wie dem herstellerseitigen *Pylon Viewer*, keine Parameter verändert werden, wird die Kamera bei jedem Programmstart erneut parametriert. Des Weiteren wird überprüft, ob die vorgegebenen Parameter auch untereinander kompatibel sind. Die Aufnahme der Frames erfolgt asynchron über einen internen Taktgeber. Die Frames werden kameraintern gepuffert, sodass sich Inkonsistenzen während der Übertragung oder andere Einflüsse, wie systemseitige Interrupts oder wartende Prozesse nicht auf die Framerate auswirken können. Sollte ein irreversibler Paketverlust auftreten, wird die Aufnahme automatisch abgebrochen und der Nutzer durch ein Warnfenster in Kenntnis gesetzt. Abhängig von dem eingestellten Modus, läuft die Aufnahme entweder für eine feste Anzahl an Frames oder bis diese durch den Druck der Escape-Taste beendet wird. Alternativ ist auch ein Verwerfen des Videos durch Drücken der Rücktaste möglich. Wurde die Aufnahme fehlerfrei abgeschlossen, kann diese anschließend durch ein simples Menü zugeschnitten werden. Nach Eingabe des zugehörigen Ergebnisfeldes wird das Video unter dem nächsten freien Dateinamen unkomprimiert im AVI-Format gespeichert.

6.1.2 Aufbereitung der Videodaten

Entsprechend den Spezifikationen aus Kapitel 5.3 und 5.4.3 müssen die Videos in gewissem Maße nachbearbeitet werden. Da dies in demselben Arbeitsschritt geschehen kann, wie das Erzeugen der Vektordaten, wurde beides im Programm *video_processing.py* zusammengefasst. Nach der Korrektur stroboskopischer Effekte und einer Kontrastspreizung werden hier, mittels Binarsierung, Schwellwertbildung und Demosaicing in jedem Frame die Koordinaten der Drehscheibe und der Kugel ermittelt. Eine Folge dieser Art der Bildverarbeitung ist der leicht verschobene Schwerpunkt der Kugel während der Außenwandphase, da diese hier teilweise durch den Rand des Roulette-Tisches bedeckt wird. Lösungen für dieses Problem wären beispielsweise Pattern-Matching, die Addition eines Offsets zum Radius der Kreisbahn oder eine Hough-Transformation für Kreise. Die Verschiebung hat jedoch keinen Einfluss auf die Netzperformance, da die Entfernung der Kugel zur Tischmitte bei einer Prädiktion während der Außenwandphase konstant ist, bzw. im direkten Bezug zu den ermittelten Koordinaten steht. Eine Korrektur ist somit überflüssig.

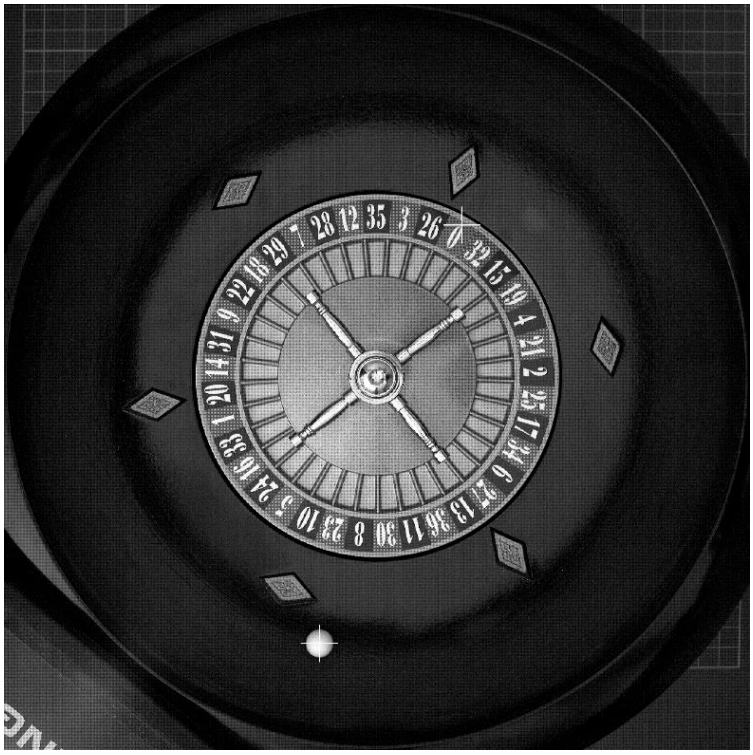


Abbildung 6.2: Debug-Ausgabe des Programms *video_processing.py* mit Markierung der ermittelten Vektordaten (Ein erkennbares Raster ist ein Resulat von Alias-Effekten durch das Bayer-Muster und nicht in der tatsächlichen Aufnahme enthalten)

Parallel zur Vektorbestimmung wird der Zeitpunkt markiert, an dem die Kugel von der Außenwand- in die Spiralphase übergeht. Ist das Ende der Spiralphase erreicht, wird die Aufbereitung vorzeitig abgebrochen, da Daten der Chaosphase nicht als Netzeingang verwendet werden. Um die Qualität der Vektordaten zu verifizieren, werden diese anschließend auf Ausreißer überprüft. Überschreiten sie die erlaubte Geschwindigkeitsänderung zwischen zwei Frame-Übergängen wird das Video verworfen. Laut der geloggten Informationen ist dies jedoch nur in anfänglichen Tests aufgetreten. Der Rauschanteil der Vektordaten ist somit dank der Flacker-Korrektur sehr gering. Wurden keine Fehler entdeckt, werden alle Koordinaten zur späteren Diagnose entsprechend einer CSV-Datei in einem separaten Verzeichnis gespeichert. In diesen sind, wie bei allen Vektordaten, die einzelnen Frames zeilenweise nach dem Format [$x_{Drehscheibe}$ $y_{Drehscheibe}$ x_{Kugel} y_{Kugel}] hinterlegt. Der Koordinatenursprung befindet sich in der oberen, linken Ecke eines Frames. Den Spezifikationen aus Kapitel 4.3 und 6.1.3 entsprechend, wird zudem sichergestellt, dass die Drehscheibe während der Spiralphase die definierte maximale Winkelgeschwindigkeit nicht überschreitet. Ist dies der Fall, werden die betroffenen Frames abgeschnitten. Im Erfolgsfall werden das zugeschnittene aufbereitete Video, die zugeschnittenen Koordinaten während der Außenwandphase und die der Spiralphase jeweils in separate Verzeichnisse unter dem oben angegebenen Format gespeichert.

6.1.3 Erzeugung der Differenzbilder

Gemäß den Spezifikationen aus Kapitel 4.3 werden Videodaten nach der Aufbereitung durch das Programm *create_image_differences.py* komprimiert. Dies hat den Hintergrund, die Größe des Netzeinganges und den erforderlichen Festplattenspeicher möglichst zu minimieren. Der aufbereitete Trainingsdatensatz der Videodaten belegt bereits etwa 578 GB an Speicher. Nach einer Vervielfältigung (siehe Abschnitt 6.1.4) würde dieser ohne Komprimierung etwa 21,4 TB umfassen, was die Kapazität des Servers um ein Vielfaches übersteigt. Damit eine Aussage getroffen werden kann, inwieweit welche Art einer Komprimierung für diese Anwendung sinnvoll ist, wurden anhand der Vektordaten verschiedene Tests durchgeführt, durch welche zumindest begrenzt Rückschlüsse auf die mögliche Performance mit Bilddaten gezogen werden können.

Ein Ansatz, diesem Problem entgegenzuwirken, war die Implementierung eines Programmes, welche bestimmte Bildteile extrahiert und diese neu anordnet. So konnte ein Video auf etwa 51 % der Originalgröße verkleinert werden, ohne einen relevanten Informationsverlust zu generieren. Hierdurch geht jedoch der örtliche Bezug verloren, welcher für die Faltungskerne essentiell ist. Eine Dekompression, wie auch bei konventionellen verlustfreien Kompressionsmethoden, kam während des Trainings nicht infrage. Obwohl diese asynchron auf der CPU durchgeführt werden würde, würde sie vermutlich, als vergleichsweise aufwendiger Vorverarbeitungsschritt, den Flaschenhals der Arbeitskette während eines Trainings darstellen.

Unter der Bedingung, die Speicherverwaltung innerhalb des 8-Bit-Formats als Netzeingang der Kompression mehrerer Frames in einem einem Bild vorzubehalten, bietet sich zur Reduktion der Datensatzgröße eine Verringerung der Auflösung an. Viele der Voruntersuchungen für diese Arbeit hatten den Hintergrund, die örtliche Genauigkeit aufgrund der Chaotik möglichst zu maximieren. Die Trainingsergebnisse auf Vektordaten haben jedoch gezeigt, dass die Trajektorien der Kugel nur auf eine sehr allgemeine Weise abstrahiert werden konnten, da eine Überanpassung der Trainingsdaten — aufgrund der geringen Datenvielfalt — die Validierungsergebnisse ansonsten bereits sehr früh verschlechtert hat (siehe Kapitel 7.2). Auf Grundlage dessen ließ sich vermuten, dass der Detailgrad der Trajektorien überhaupt nicht ausgeschöpft wurde. Es wurde daher untersucht, wie stark eine Verringerung der Vektorauflösung mit der Netzperformance zusammenhängt.
Um das tatsächliche Lernverhalten der Netze in dieser Arbeit möglichst detailliert vergleichen zu können, wurden verschiedene Metriken implementiert, welche die Ausgabevektoren des Netzes in der Reihenfolge der Ziffern auf der Drehscheibe anordnen und so verschieben, dass der Ausgang, der dem Label des aktuellen Eingabevektors entspricht, in der Mitte der Ausgabevektoren verschiedener Metriken liegt. Auf diese Weise lässt sich beobachten, wie sich das Ausgabeverhalten des Netzes, abhängig von der Relation zu den jeweils richtigen Feldern, bei verschiedenen Hyperparametern oder Datensätzen unterscheidet. Abbildung 6.3 zeigt einen Vergleich der Entwicklung des Prädiktionsverhaltens desselben Netzes bei Training und Validierung mit den Originaldaten und bei einer starken Verringerung derer Auflösung.

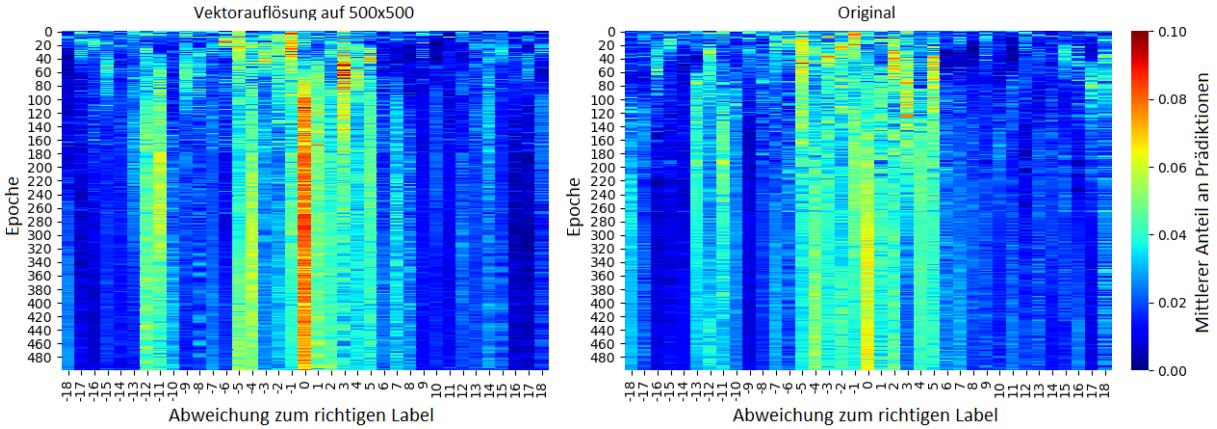


Abbildung 6.3: Gegenüberstellung Einfluss der Vektorauflösung auf das Lernverhalten der Validierungsdaten relativ zu den richtigen Ergebnissen während der Spiralphase

Auf der Abszisse ist die Distanz in Feldern des mit dem jeweiligen Netzausgang korrespondierenden Feldes zu dem jeweils richtigen aufgetragen. Auf der Ordinate ist der Trainingsfortschritt in Epochen dargestellt. Die Farbskala beschreibt den Anteil an Prädiktionen des jeweiligen Feldes in der Epoche. Höhere Werte nahe der Mitte bedeuten also, dass die meisten prädizierten Felder in diesem Trainingsverlauf auch nahe dem jeweils richtigen Feld lagen. Ein perfektes Netz würde in dieser Darstellung einen tiefroten Streifen bei der Null und ein dunkles Blau bei allen anderen Entfernungen aufweisen. Diese Darstellungsform wird in den nächsten Abschnitten sehr oft verwendet. Um Platz zu sparen, wird, sofern sich diese nicht ändern, auf die Achsenbeschriftungen verzichtet. Eine genauere Analyse des Lernverhaltens der Netze befindet sich in Kapitel 7.

Obwohl die Tests, aufgrund der notwendigen zufälligen Initialisierung der Gewichte, gewissen Schwankungen unterliegen, stand das Generalisierungsverhalten der Validierungsdaten, bei einer deutlich geringeren Vektorauflösung, den Originaldaten in keiner Weise nach. So konnten, wie in diesem Beispiel, auch oft Trainingsverläufe gefunden werden, bei welchen die Performance der Validierungsdaten bei einer geringen Vektorauflösung die der Originaldaten deutlich übersteigt. Zu betonen ist, dass die Vektordaten die Schwerpunkte der Objekte repräsentieren, welche, abhängig von der Objektgröße und Form, teilweise eine deutlich höhere Auflösung ermöglichen, als die der Kamera. Eine Vektorauflösung von 500x500 ist demnach nicht mit derselben Auflösung einer Aufnahme gleichzusetzen, sodass angenommen werden kann, dass eine Skalierung der Auflösung der Videodaten auf 500x500 Pixel, bei gleicher Menge an verwendeten Frames die Netzperformance nicht behindern wird. Für diesen Test wurde allerdings ein, für das Training mit Vektordaten hier üblicher, Vektor mit 400 Elementen verwendet, was einer Berücksichtigung von 100 Aufnahmen am Netzeingang entspricht. Das stärkere örtliche Rauschen durch die Skalierung wurde also womöglich problemlos durch die hohe Redundanz an Informationen gefiltert. Für eine Beurteilung der Anforderungen an die Kompression mehrerer Frames in einem Eingangsbild muss daher untersucht werden, wie stark der Einfluss der Menge an Eingangsdaten auf die Performance ist. Hierfür wurden die Vektordaten auf verschiedene Längen zugeschnitten (siehe Abschnitt 6.1.5) und dasselbe Netz trainiert.

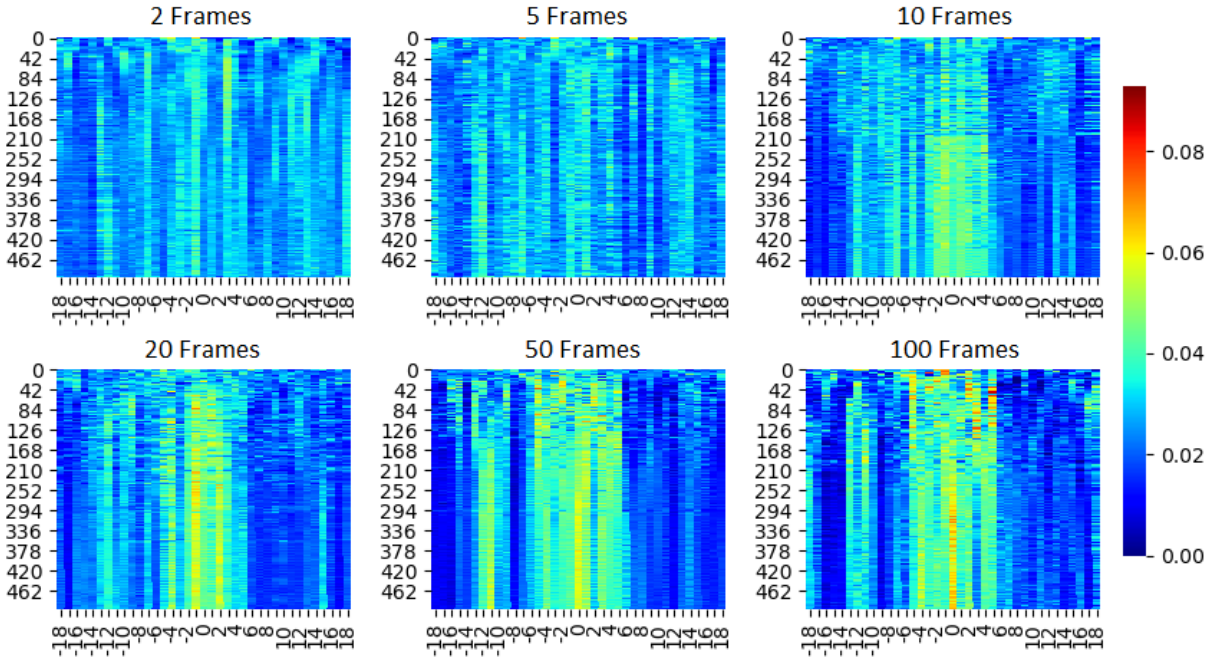


Abbildung 6.4: Gegenüberstellung Einfluss der Menge an verwendeten Frames am Netzeingang auf die Prädiktionswahrscheinlichkeit relativ zum richtigen Feld von Validierungsdaten während der Spiralphase über mehrere Epochen

Bei einem Zuschnitt der Daten in kleinere Vektoren werden mehr Datenpunkte erzeugt, was den Vergleich für diese Untersuchungen erschwert, da der Zuwachs an Daten nicht dem der Videodaten entspricht. Es wurde daher ein Programm geschrieben, welches aus dem Datensatz für dieses Training eine feste Anzahl an Daten zufällig auswählt, sodass die hier zu sehenden Unterschiede hauptsächlich auf die Länge des Eingangsvektors zurückzuführen sind. Die Untersuchung einer optimalen Vektorlänge für das Training mit Vektordaten befindet sich im Abschnitt 6.1.5. Es lässt sich gut erkennen, dass bei einer Verwendung von Vektoren, welche Informationen von weniger als 20 Frames berücksichtigen, die Menge an Informationen nicht ausreicht, um die Bezüge zum Ergebnis vernünftig zu approximieren. Neben einer möglicherweise mangelhaften Rauschunterdrückung, ist dies auch ein Indiz dafür, dass weitere, nur über längere Zeiträume erfassbare Effekte — wie trägheitsbedingtes Rutschen, dem Drall der Kugel oder situationsbedingte ortsunabhängige Reibungseffekte — einen deutlich stärkeren Einfluss haben als anfangs erhofft. Die Anforderungen an die Kompression mehrerer Frames in einem Bild sind daher relativ hoch. Es muss jedoch betont werden, dass die für diese Tests zugeschnittenen Vektordaten auf direkt aufeinanderfolgenden Frames basieren. Für die Kompression der Bilddaten kann der Abstand zwischen verwendeteten Aufnahmen jedoch deutlich erhöht werden, solange eine Eindeutigkeit des Bildes zu dem Ergebnis gewährleistet ist. Da die örtliche Auflösung hier scheinbar ohnehin nicht genutzt wird und die Videos auch zur Bestimmung der Vektordaten binarisiert werden, sollte eine Binarisierung der Differenzbilder an einer Schwelle, welche relevante Konturen möglichst originalgetreu hervorhebt, keine allzu großen Nachteile verursachen. Damit die Faltungskerne den Vorteil der Farbwahrnehmung dennoch zumindest in begrenzter Form

nutzen können, wird das Bayer-Format beibehalten, indem jeder Farbkanal separat binarisiert wird. Um keinen weiteren Informationsverlust zu erzeugen, werden acht binäre Differenzbilder durch Bitverschiebung ineinander gefügt. Insgesamt werden also 16 Frames in einem Bild vereint. Ist die Distanz zwischen den Frames passend gewählt, finden bei den Bildausschnitten der Kugeln keine Überlagerungen statt. Aufgrund des Trajektorienverlaufs ist deren Anzahl hinreichend groß, dass keine Mehrdeutigkeiten entstehen können. Jede der binarisierten Kugelflächen kann somit auf den maximalen Helligkeitswert gesetzt werden. Bei der Drehscheibe sind die Überlagerungen jedoch so groß, dass der gesamte Wertebereich ausgenutzt werden sollte um möglichst viele Informationen der Frames zu behalten.
Die Entscheidung zur Binarisierung der Eingangsdaten widerspricht teilweise der Argumentation für eine Korrektur stroboskopischer Effekte aus Kapitel 5.3. Dennoch bietet diese auch hier große Vorteile, da beispielsweise keine dynamische Schwellwertvergabe implementiert werden muss. Eine Kompression der Bildbereiche der Drehscheibe durch Bitverschiebung insofern nachteilhaft, dass Korrelationen von Aufnahmen an Stellen niederwertiger Bits beim Training deutlich schwächer berücksichtigt werden. Eine gute Alternative wäre, die Helligkeitswerte, abhängig von der Auftrittswahrscheinlichkeit der Überlagerungen aus der Summe der Aufnahmen, so zu mappen, dass die am häufigsten auftretenden Konturen den größtmöglichen Kontrast erhalten. Eine Ausgabe der Co-occurrence-Matrizen mit den Offsets einer Vierer-Nachbarschaft hat gezeigt, dass die meisten Überlagerungsübergänge bei zwei bis drei Bildern gleichzeitig auftreten. Anhand dieser Matrizen ließe sich ein Algorithmus entwickeln, welcher die Informationen der einzelnen Frames optimal ineinanderfügt. Da sich die Drehscheibe jedoch relativ linear und wenig chaotisch verhält, verspricht dieser Ansatz mehr Aufwand als Nutzen. Entsprechend den Erkenntnisse aus Kapitel 5.2, sollten, selbst bei Vernachlässigung der niederwertigen Bits, mit einer Bitverschiebung bereits ausreichend Informationen zur Verfügung stehen.

Bei einer Berücksichtigung von 16 Aufnahmen pro Netzeingang wird die erlaubte Distanz zwischen den verwendeten Frames nicht durch die Anforderung der Eindeutigkeit, sondern durch die Länge der Videos definiert. Hierfür wurde die Häufigkeitsverteilung der Videos in Relation zur deren Länge in Frames berechnet. Es hat sich gezeigt, dass, abgesehen von anfänglichen Aufnahmen, bei welchen der automatische Zuschnitt noch nicht korrekt funktioniert hat, alle Videos eine Mindestlänge von 112 Frames aufweisen. Um hierraus zumindest ein Bild zu erzeugen, muss die Distanz zwischen den verwendeten Aufnahmen kleiner sein als $\frac{112-1}{15} = 7{,}4$ Frames und wurde daher auf sieben Frames festgelegt. Abbildung 6.5 zeigt ein beispielhaftes Bild am Netzeingang nach Anwendung der hier erläuterten Verfahren.

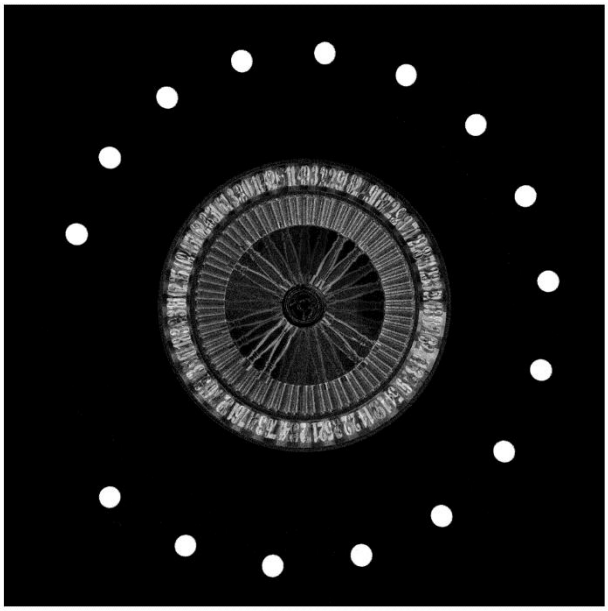


Abbildung 6.5: Beispiel eines Frames des Trainingsdatensatzes der Videodaten

Verglichen mit dem Informationsgehalt eines Datensatzes aus auf 100 Frames zugeschnittenen Vektordaten, werden hier vor der Vervielfältigung pro Video sechs Datenpunkte weniger erzeugt. Zudem werden nur 16 Aufnahmen berücksichtigt, was nach den Testergebnissen aus Abbildung 6.4, insbesondere durch die Skalierung der Auflösung, kritisch sein könnte. Da aus dem Test nicht hervorgeht, ob die Menge an Messwerten oder die Zeitspanne, über welche diese aufgenommen wurden, das Potential der Netzperformance begrenzen, stellt dies dennoch einen guten Kompromiss beider Faktoren dar.

6.1.4 Vervielfältigung der Datensätze

Die Größe des Datensatzes ist darüber entscheidend, inwieweit das Netz in der Lage sein kann, die tatsächlichen physikalischen Zusammenhänge abzubilden und somit unbekannte Daten zu generalisieren. Aufgrund des stark chaotischen Verhaltens der Kugeltrajektorie, muss vermutlich eine deutlich größere Anzahl an Trainingsdaten erzeugt werden als es bei typischen Klassifizierungsaufgaben der Fall ist, um eine vergleichbare Performance zu erreichen. Ein üblicher Ansatz dies zu realisieren, ist eine künstliche Vervielfältigung der Daten. Standardverfahren sind beispielsweise das Verschieben, Verzerren, Spiegeln oder Drehen zu klassifizierender Objekte im Abbildungsbereich. In diesem Fall wird nur eine Perspektive betrachtet, wodurch die Möglichkeiten der Datenaugmentierung erheblich eingrenzt werden. Der hier verfolgte Ansatz basiert auf der Annahme, dass sich die Trajektorie der Kugel und der Drehscheibe zu bestimmten Begebenheiten wiederholen. So kann die Drehscheibe in jedem Frame künstlich um Vielfache des Winkels eines Feldes gedreht werden, da bei einem idealen Roulettespiel die Kugel und die Kanten der Drehscheibe an diesen Stellen auf gleiche Weise aufeinander treffen. Ähnliches gilt für die Kugeltrajektorien. Aufgrund der sich wiederholenden Anordnung der Rauten wiederholt sich ideal betrachtet auch die Trajektorie, unter

Berücksichtigung der Drehscheibe, alle 120 °. Es ist somit möglich, das 37 · 3 = 111-fache an Daten zu generieren, was erhebliche Einflüsse auf die Generalisierungsfähigkeit haben kann. Basierend auf den Erkenntnissen aus Kapitel 5.2, wirken sowohl auf die Kugel, als auch auf die Drehscheibe positionsabhängige Einflüsse, wie ungleichmäßige Reibungen, Unebenheiten im Kesselboden oder ungenau eingelassene Rauten. Insbesondere bei der Vervielfältigung der Kugeltrajektorie haben diese Einflüsse, durch die Chaotik und die damit verbundene Fehlerfortpflanzung, immense Auswirkungen, was die Qualität der vervielfältigten Daten wiederum erheblich verschlechtern kann. Für diese Untersuchungen wurden zwei Programme — *duplication_of_vectordata.py* und *duplication_of_videodata.py* — geschrieben. Abhängig von der Parametrierung werden in diesen entweder mittels Vektorrechnung, oder durch das Drehen von Ausschnitten der Bildmatrix, die erwünschten Daten generiert. Zu betonen ist, dass bei der Vervielfältigung der Videodaten, für das Drehen der Bildausschnitte, eine temporäre Konvertierung des Bayer-Formats in RGB notwendig ist. Ansonsten würden die Farbkanäle verwischen und somit einen Informationsverlust erzeugen. Für das Drehen ist es wichtig, dass sich der Bildmittelpunkt exakt auf dem Mittelpunkt des Roulette-Tisches befindet. Dies wurde erreicht, indem beim Verkleben des Tisches mit dem Stativboden, bei gleichzeitiger Aufnahme, ein Kreis mit sklalierbarem Radius über die erfassten Frames gelegt wurde. Die Abweichung der Mittelpunkte beträgt etwa ein Pixel.

Der optimale Kompromiss zwischen Datenmenge und Datenqualität wurde, wie bei allen Tests auf Einflüsse des Datensatzes, mit einem Netz und einer Kostenfunktion ermittelt, deren charakteristisches Lernverhalten eine relativ geringe Abhängigkeit gegenüber der Initialisierung der Gewichte aufweist. Dennoch wurde jede Konfiguration auch hier mindestens dreifach trainiert. Nachfolgende Tabellen zeigen eine beispielhafte Übersicht über das Lernverhalten, in Abhängigkeit von den vorgestellten Methoden zur Datenvervielfältigung.

Tabelle 6.1: Maximalwert an erreichten stabilen, richtigen Prädiktionen auf den Validierungsdaten bei verschiedenen Methoden zur Datenvervielfältigung

		Drehscheibe verändert	
		Nein	Ja
Gesamtverlauf verändert	Nein	5,0 %	6,0 %
	Ja	1,9 %	5,9 %

Tabelle 6.2: Abweichung der Prädiktionen zum richtigen Feld auf den Validierungsdaten zum Zeitpunkt der Werte aus Tabelle 6.1

		Drehscheibe verändert	
		Nein	Ja
Gesamtverlauf verändert	Nein	8,9 Felder	7,5 Felder
	Ja	9 Felder	7,4 Felder

Tabelle 6.3: Mittelwert der Differenz zwischen dem Anteil an richtigen Prädiktionen auf den Trainings- und Validierungsdaten bei gleichem Trainingsfortschritt mit verschiedenen Methoden zur Datenvervielfältigung

		Drehscheibe verändert	
		Nein	Ja
Gesamtverlauf verändert	Nein	28,0 %	11,7 %
	Ja	23,5 %	13,7 %

Es wird deutlich, dass eine Vervielfältigung der Daten durch das Drehen des Gesamtverlaufs entsprechend der eingelassenen Rauten, trotz Verdreifachung des Datenvolumens, aufgrund der hohen Fehlerfortpflanzung keinen Mehrwert liefert. Auffällig ist, dass der Anteil an richtigen Prädiktionen beim Lernen von den üblichen 2,7 % eines unwissenden Spielers auf 1,9 % abgesunken ist. Das Netz hat also von den vervielfältigten Trainingsdaten gelernt, die Validierungsdaten falsch zu prädizieren.

Ein deutlich besserer Effekt konnte durch eine Veränderung der Trajektorie der Drehscheibe erreicht werden. Während die Datenqualität auch hier aufgrund der Vernachlässigung positionsabhängiger Reibungen und sonstiger Einflüsse leidet (vgl. Abbildung 5.2), hat die Fehlerfortpflanzung durch die geringeren chaotischen Tendenzen einen viel kleineren Einfluss ,als bei Änderungen der Kugeltrajektorie. Zudem wird das 37-fache der Daten erzeugt, wodurch der Effekt deutlich verstärkt wird. Ein mögliche Folge der Datenerzeugung ist das einseitige Lernen. Stehen für eine Klasse mehr Informationen zur Verfügung, haben diese auch einen größeren Einfluss auf den Gesamtfehler. Die Gewichte werden somit stärker nach dem Bezug zu dieser Klasse angepasst. Obwohl hier ein Klassifikator erlernt werden soll, beschreibt der Ablauf ein Regressionsproblem. Die Komplexität des Zusammenhangs wird fast ausschließlich durch den — nicht-klassenspezifischen — Verlauf der Kugeltrajektorie definiert, das Ergebnis aber etwa zum gleichen Teil durch die Ausrichtung der Drehscheibe. Die Art und Weise, wie ein ideales Ergebnis ermittelt wird, ist somit für jede Klasse fast identisch und unterscheidet sich vermutlich im Wesentlichen durch einen abstrahierten Offset. Der Effekt des einseitigen Lernens würde also eher gering ausfallen. Durch eine Augmentation der Datenpunkte zu jeder Klasse, wie es hier der Fall ist, und der daraus resultierenden Gleichverteilung an Informationen über alle Klassen, wird dieser Effekt jedoch ohnehin weitestgehend eliminiert.

Eine Kombination aus beiden Ansätzen hat ähnlich vielversprechende Ergebnisse erzeugt, welche aufgrund der Schwankungen der Lernverläufe bei weiteren Tests möglicherweise sogar das Optimum darstellt. Obwohl zwei Drittel der Informationen auf veränderten Kugeltrajektorien basieren, bietet die enorme Menge an Daten eine ausreichende Basis. Der Umgang mit einer dreimal größeren Datenmenge, die vermutlich keine Verbesserung der Netzperformance darstellt, ist jedoch nicht sinnvoll.

6.1.5 Zuschnitt

Der Zuschnitt dient dazu, eine einheitliche Größe für den Netzeingang zu schaffen und die Daten anschließend in einer definierten Verzeichnisstruktur zu speichern, welche für den Import beim Training notwendig ist (siehe Kapitel 6.2.1). Abhängig davon, welches Datenformat aktuell untersucht wird, ist hierfür entweder das Programm *cut_vectordata.py* oder *cut_videodata.py* zuständig. Für die Videodaten wird die Anzahl der verwendeten Aufnahmen bereits während der Erzeugung der Differenzbilder definiert. Hier müssen daher nur die einzelnen Frames separiert und als Bilder im PNG-Format abgespeichert werden.

Aus den Vektordaten werden, bei einer Zuschnittslänge x für jedes Frame, sofern möglich, die Daten der nächsten x Frames in eine Zeile einer Textdatei geschrieben und abgespeichert. Hier muss untersucht werden, welche Zuschnittslänge sich für diese Arbeit als optimal herausstellt. Je länger diese gewählt wird, desto mehr Informationen stehen am Netzeingang zur Verfügung. Die Anzahl an Datenpunkten, welche aus einem Video generiert werden können, nimmt jedoch ab. Entgegen den Tests aus Abschnitt 6.1.3, wurde die Größe des erzeugten Datensatzes für diesen also nicht begrenzt. Abbildung 6.6 zeigt beispielhafte Ausschnitte von Lernverhalten desselben Netzes bei unterschiedlicher Vektorlänge.

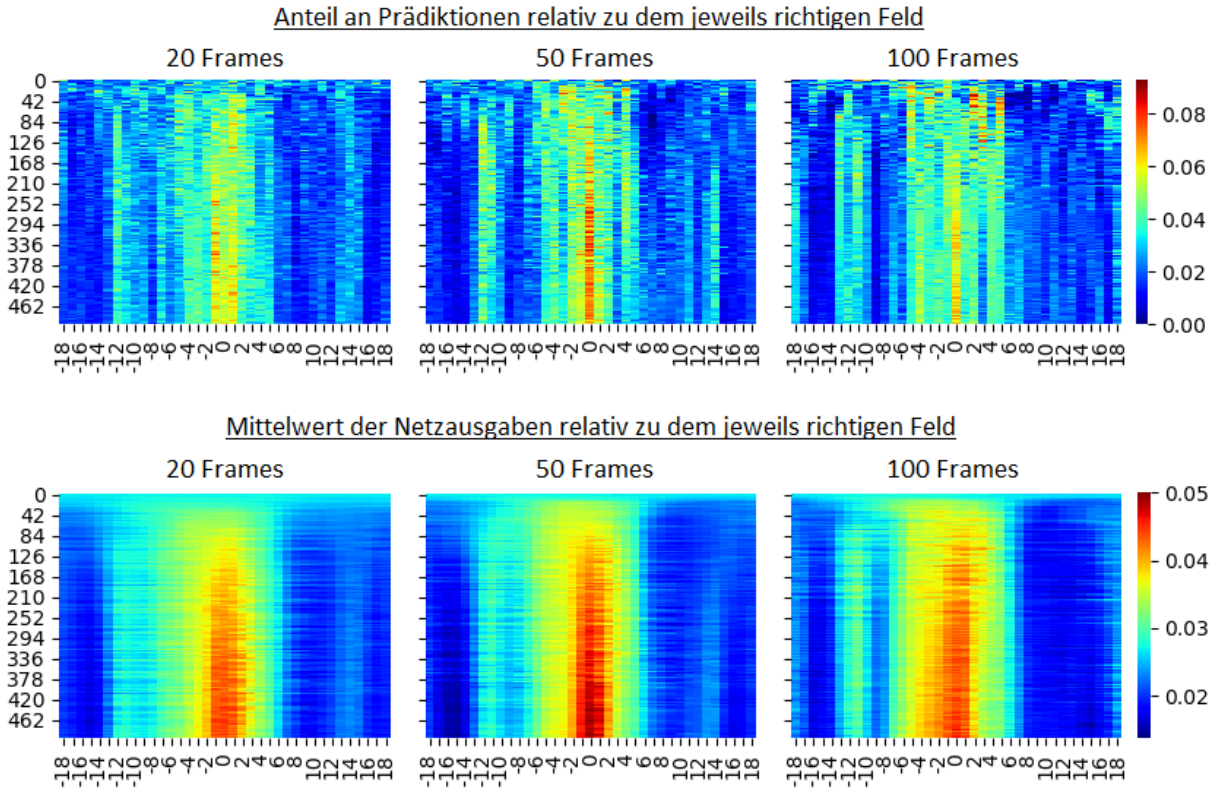


Abbildung 6.6: Gegenüberstellung Einfluss der Menge an verwendeten Frames am Netzeingang auf die Prädiktionswahrscheinlichkeit und das Netz-Ausgabeverhalten relativ zum richtigen Feld bei Validierungsdaten während der Spiralphase über mehrere Epochen

Insbesondere das Prädiktionsverhalten der Netze unterliegt einer starken Abhängigkeit zu den Initialwerten der Gewichte. Um den Einfluss der Vektorlänge zu verdeutlichen, wurden hier daher als Vergleich die zugehörigen Mittelwerte der Netzausgaben während des Trainings jeweils relativ zum richtigen Ergebnis geplottet. Hier gehen also nicht nur die Maximalwerte des Ausgabevektors, sondern alle Werte in die Berechnung ein. Es lässt sich gut erkennen, dass sich das Optimum vermutlich nahe einer Vektorlänge befindet, welche 50 Aufnahmen berücksichtigt. Eine längerer Vektor scheint zwar am Eingang eine höhere Redundanz zu ermöglichen, liefert aber keine weiteren Informationen, welche das Training weiter verbessern würden. Durch die Nutzung aller Frames eines Videos entsteht eine Tendenz, beim Training den Fokus auf Bezüge in längeren Videos zu setzen, da deren Anteil im Datensatz und somit im gemittelten Gradienten größer ist. Bei kleinerer Vektorlänge sinkt dieser Effekt leicht, da das Verhältnis an generierten Informationen zwischen dem längsten und dem kürzesten Video verringert wird. Bei Vektordaten der Außenwandphase könnte sich das Optimum aufgrund der tendenziell längeren Videos und einer höheren Anforderug an eine Redundanz leicht verschieben. Bei einer Zuschnittlänge auf 50 Frames konnten für die Spiralphase etwa 4,1 Millionen und für die Außenwandphase etwa 13,7 Millionen Datenpunkte generiert werden.

6.2 Umsetzung in Keras und Tensorflow

Die Umsetzung in Keras ist relativ simpel gestaltet. Viele Funktionalitäten sind als parametrierbare Funktion oder Klasse realisiert, welche sich in nur einer Zeile Code einem Netzmodell anhängen lässt. Implementierungen wie Dropout, Batch-Normalisierung oder Vorverarbeitungen am Netzeingang lassen sich so in wenigen Minuten implementieren und bei Bedarf in vollem Umfang parametrieren. Der grundlegende Aufbau orientiert sich an dem der Keras-Dokumentation [40]. Dieser bietet nur wenig Mehrwert für das Verständnis über den Umfang und die Gedankengänge zu den Untersuchungen in dieser Arbeit und wird daher nicht genauer erläutert. Darüber hinaus sind verschiedene anwendungsspezifische Implementierungen und die Nutzung weiterer Verfahren notwendig, welche in den nachfolgenden Abschnitten beschrieben werden.

6.2.1 Einlesen der Trainings- und Validierungsdaten

Wie bereits in Kapitel 6.1 erläutert, erreichen die Datensätze teilweise eine Größe, welche den verfügbaren Arbeitsspeicher bei weitem übersteigt. Während des Trainings ist es unvermeidlich, diese in sogenannte Batches bzw. Minibatches zu unterteilen und stückweise von den Festplatten zu laden. In Keras bzw. Tensorflow wird dies über sogenannte Input-Pipelines realisiert. Während das eigentliche Training des aktuellen Batches auf der GPU durchgeführt wird, ist es möglich, anhand der Input-Pipeline das nächste Batch asynchron aus dem Verzeichnis zu laden und über eine Reihe von *Mapping*-Funktionen transformiert in sogenannten Tensorflow-spezifischen *Dataset*-Objekten zu speichern. Ein Großteil dieses Arbeitspakets ist bereits in den Funktionen

text_dataset_from_directory bzw. *image_dataset_from_directory* implementiert. Neben einer automatisierten Zuweisung von Labeln entsprechend der implementierten Verzeichnisstruktur und einer Aufteilung in eine vorgegebene Batchgröße, ist auch eine Vorverarbeitung und Mischung der Daten möglich. Die Vorverarbeitung dient hier vorrangig zur Gewährleistung einer gewissen Einheitlichkeit am Netzeingang. Bei Bilddaten entspricht dies beispielsweise einer Skalierung der Eingangsbilder auf eine definierte Auflösung. Die Funktion *text_dataset_from_directory* dient dem Import von Texten aus Text-Dateien und entsprechendem Mapping auf einen Zahlenbereich. Ein Einlesen von Gleitkommazahlen in Textform ist durch die Parametrierung nicht vorgesehen. Um dennoch die vielen Vorteile dieser Funktion nutzen zu können, wurden Teile der Unterfunktionen umgeschrieben, sodass, anstelle des Mappings von Wörtern, mittels verschiedener Tensorflow-Funktionen Vektordaten in Textform separiert und konvertiert werden. Anstelle des Tensorflow-Pakets wird diese Funktion separat über die Datei *vector_data_input_pipeline.py* eingebunden. Bezüglich dieses Ansatzes existieren mehrere Alternativen. Beispielsweise sind andere Datenformate, wie sogenannte TFRecord-Dateien möglich, in welchen der gesamte Datensatz theoretisch auch in einer Datei untergebracht werden könnte. Solche Formate sind den Textdateien vermutlich in Bezug auf notwendige Ladezeiten und Speicherverbrauch überlegen. Für die Untersuchungen wurde jedoch viel Wert auf Datennähe gelegt. Textdateien lassen sich auch ohne Funktionen oder spezielle Programme öffnen und überprüfen. Fehler können während der Entwicklung sehr leicht und schnell getestet werden. Beispielsweise lässt sich die Implementierung einer Skalierung der Auflösung der Daten aus Kapitel 6.1.3 auf einen Blick verifizieren. Durch die Separierung der Datenpunkte ist es möglich, bereits im Explorer gezielte Veränderungen am Datensatz vorzunehmen oder die Anzahl der Datenpunkte zu prüfen. Das Einlesen mehrerer Millionen Textdateien dauert ohnehin nur wenige Minuten und der Speicherverbrauch liegt, aufgrund der Informationsdichte, bei wenigen Gigabyte. Dies stellt nur einen kleinen Teil des verfügbare Arbeitsspeichers dar, sodass sich hieraus vorverarbeitete Cache-Dateien erzeugen lassen, welche zu Beginn des Trainings geladen werden. Mögliche Nachteile von Textdateien werden somit komplett eliminiert.

6.2.2 Nutzung mehrerer Grafikkarten

Im Allgemeinen existieren zwei Ansätze zur Aufteilung des Trainings auf mehrere Geräte, welche als *data parallelism* und *model parallelism* bezeichnet werden. Letzterer spaltet das zu trainierende Netz in mehrere Teile, welche gemeinsam dasselbe Batch an Daten bearbeiten. Dieser Ansatz eignet sich vor allem für Netzstrukturen mit parallelen Schichten welche erst am Ende des Netzes zusammengeführt werden, deren Untersuchung jedoch nicht Bestandteil dieser Arbeit ist. Es wird daher der Daten-Parallelismus angewendet. Hierbei wird dasselbe Netzmodell auf die gewünschten Geräte repliziert und auf verschiedenen Daten trainiert, dessen Resultate anschließend vereint werden. Im Unterschied zu anderen Varianten dieses Ansatzes, erfolgt der hier implementierte Zugriff und die anschließende Zusammenführung der Daten synchron. Jedes Batch wird, abhängig von der Anzahl verwendeter Grafikkarten, gleichmäßig in

kleinere, sogenannte *local batches* unterteilt und jeder GPU zugeteilt. Auf diese Weise ändert sich das Trainingsverhalten gegenüber dem Training auf einer einzigen GPU/CPU nicht. In Keras steht hierfür die *tf.distribute.MirroredStrategy* API zur Verfügung.

6.2.3 Umsetzung der Kostenfunktion

In Keras ist bereits eine große Auswahl konventioneller Kostenfunktionen implementiert. Bezüglich einer proprietären Lösung sind die Anforderungen laut der Dokumentation sehr gering. Hier wird jede Form von Funktion akzeptiert, welche auf zwei übergebene Tensoren[3] *y_true* und *y_pred* mit einem Tensor gleicher Länge antwortet, in dem für jedes Element der Eingangstensoren der zugehörige Fehler enthalten ist. Die tatsächlichen Umsetzungsmöglichkeiten unterliegen jedoch weitaus mehr Einschränkungen. Aufgrund der Art und Weise, wie die Funktion kompiliert wird und wie Tensorflow diese optimiert, ist es erforderlich, teilweise verschiedene Tensorflow-spezifische Verfahren umzusetzen, um Speicherreservierungen vor dem Training oder Fehler bei Übergabeparametern einer Länge von *None* zu vermeiden. Der in Kapitel 4.5 vorgestellte Ansatz wird unter dem Namen *custom_loss_functions.py* eingebunden. Die hier implementierte Form weicht leicht von der Theorie aus Kapitel 4.5 ab. Im rechten Summanden aus Formel 4.3 wird beim Zugriff auf die Elemente der Vektoren $\bar{y}_T$ und $\bar{y}_P$ auf die Konvertierung der Feldnummern zum entsprechenden Label verzichtet. Der Fehler der Prädiktion des richtigen Feldes wird dadurch nicht bei k = $f_{lf}^{-1}(y_T)$ sondern bei k = y_T berechnet. Entsprechend dem Kommutativgesetz hat dies auf den Gesamtfehler jedoch keinen Einfluss und spart etwas Rechenzeit.

6.2.4 Metriken

Für das Training der Netze lassen sich sogenannte Metriken definieren, welche dem Netzmodell vorab übergeben werden. Mit Metriken sind hier Funktionen gemeint, anhand welcher sich die Performance des Netzes während des Trainings beurteilen lässt. Sie werden sowohl auf den Trainings- als auch auf den Validierungsdaten angewandt. Ihr Unterschied zu einer Kostenfunktion ist lediglich die Tatsache, dass ihre Rückgabewerte keinen Einfluss auf das Netz haben, sondern nur zur Messung der Performance dienen. Die Anforderungen an selbst geschriebene Metriken sind daher dieselben, sodass sich auch jede Kostenfunktion als Metrik verwenden ließe. Selbst geschriebene Metriken werden in dieser Arbeit über die Datei *custom_metrics.py* eingebunden. Nachfolgend wird eine Übersicht über deren Inhalt und Funktion gegeben.

[3] Tensorflow-spezifisches Objekt ähnlich zu einem Array

mean_correct_predictions:
Als prädiziertes Ergebnis wird das Element des Netz-Ausgabevektors gewertet, welches den höchsten Wert besitzt. Der Rückgabewert ist die, über das aktuelle Batch gemittelte, Summe der Vorkommen, an welchen dieses Element dem richtigen Ergebnis entspricht. Dies stellt die Wahrscheinlichkeit dar, mi der das Netz aktuell exakt das richtige Feld prädiziert.

mean_prediction_correct_label_plus_offset:
Dies ist die Erweiterung zur Metrik *mean_correct_predictions*. Über einen Offset-Parameter lässt sich definieren, welches Feld relativ zu dem jeweils richtigen betrachtet wird. In Relation stehen hier nicht die Label, nach welchen der Netz-Ausgabevektor sortiert ist, sondern die zugehörigen Feldnummern in kreisförmiger Anordnung. Ein Offset von Zwei bedeutet also gemäß Abbildung 4.1 zwei Felder neben dem richtigen Ergebnis im Uhrzeigersinn. Nach der Definition der Keras-Dokumentation [40] sind weitere Übergabeparameter an Metriken bzw. mehrere Rückgabeparameter pro Datenpunkt nicht erlaubt. Hier ist es jedoch von Vorteil, das Prädiktionsverhalten aller Felder relativ zu dem richtigen zu untersuchen, um einen detaillierten Einblick in das Lernverhalten der Netze zu bekommen. Bezüglich dieses Problems konnten drei Lösungsansätze gefunden werden. Es wäre möglich, direkt in der Metrik Daten zu protokollieren. So könnten ganze Vektoren oder Matrizen erstellt und beispielsweise als Histogramm über das *summary*-Modul aus Tensorflow gespeichert werden. Da die Metrik jedoch keine weiteren Informationen erhält, wäre keine netzspezifische Namensgebung möglich, was entweder in aufwändigen oder unübersichtlichen Testreihen resultieren würde. Des Weiteren ließe sich ein Callback formulieren (siehe Abschnitt 6.2.5), in dem die Netzausgänge und die richtigen Label des aktuellen Batches, beispielsweise durch eine selbst programmierte Trainingsschleife, verwendet und ausgewertet werden können. Neben einem höheren Entwicklungsaufwand ist dieser Ansatz jedoch auch, wie der vorherige, eher unschön, da Arbeitspakete an Programmstellen ausgeführt würden, an welche sie eigentlich nicht hingehören. Stattdessen wurde hier für jeden Offset eine separate Metrik definiert, welche ihren jeweiligen Offset an diese Funktion übergibt. Abgesehen von einer deutlich größeren Anzahl an Metriken sind hierdurch keine weiteren Nachteile gegeben.

mean_deviation:
Konvertiert man für das gesamte Batch die jeweils richtigen und prädizierten Label in ihre zugehörigen Feldnummern, lässt sich daraus die mittlere minimale Abweichung des Prädiktionsergebnisses in Feldern ermitteln. Netze, welche beispielsweise oft Felder nahe dem richtigen prädizieren, jedoch weniger oft exakt das richtige, sind für die Entwicklung sehr interessant, da sie dennoch grundsätzlich ein relativ originalgetreues Verständnis der Korrelationen entwickelt haben.

max_deviation:
Dies ist in sehr optimistischer Ansatz zur Protokollierung der maximalen Abweichung einer Prädiktion zum richtigen Feld in einem Batch. In keiner der Messreihen wurde jedoch jemals ein Batch registriert, in welchem die ungenaueste Prädiktion nicht exakt auf der anderen Seite des Ziffernkreises lag.

mean_profit_per_game_full_number:
Für einen oberflächlichen Vergleich zu den Profit-Erwartungswerten aus Forschungen mittels Modellbildung, wird hier nach Formel 2.12 der durchschnittliche Erwartungswert eines Batches bei einer Wette auf die prädizierte Zahl berechnet. In der Regel ist jedoch eine Maximierung des Profits durch Wetten auf weitere Felder möglich, welche jedoch, aufgrund des unnötigen Rechenaufwands, erst nach dem Training für bestimmte Netze untersucht wurde.

mean_certainty_max_value:
Hier wird der Mittelwert der jeweils maximalen Netz-Ausgabewerte über ein Batch berechnet. Hierdurch ist ein Maß gegeben, wie sicher das Netz im Mittel mit der jeweiligen Prädiktion ist.

mean_value_correct_label:
Eine reine Betrachtung des Prädiktionsverhaltens kann in einigen Fällen etwas ungenau sein, da in die Berechnung nur das Vorkommen der jeweiligen Maximalwerte eingeht. Diese Metrik berechnet den Mittelwert aller Netz-Ausgabewerte an der Stelle des jeweils richtigen Labels. Prädiziert das Netz beispielsweise oft daneben, erzeugt aber an der richtigen Stelle ähnlich hohe Werte, wird dies hiermit erfasst.

mean_value_correct_label_plus_offset:
Auch bei den gemittelten Ausgabewerten ist es interessant, das gesamte Ausgabeverhalten in Relation zu dem jeweils richtigen Feld zu betrachten. Wie bei der Berechnung der mittleren Prädiktionen, wurde dies auch hier durch eine Reihe von Metriken realisiert, welche auf diese Funktion zugreifen und ihren Offset übergeben.

std_deviation_correct_label:
Durch die Mittelung der gesamten Netzausgabe ist bereits eine erheblich genauere Untersuchung des Lernverhaltens möglich, als es das reine Prädiktionsverhalten erlaubt. Es kann aber keine Aussage über die Verteilung der einzelnen Werte getroffen werden, da grundsätzlich das Batch als Ganzes betrachtet wird. Unter Berücksichtigung der Standardabweichung der Werte der Elemente des Ausgabevektors, welche dem richtigen Label entsprechen, ist ein Maß gegeben, wie stark diese variieren. Insbesondere für die Entwicklung der Kostenfunktion ist dies eine weitere Hilfestellung, Rückschlüsse vom Lernverhalten auf die Parametrierung ziehen zu können.

std_deviation_correct_label_plus_offset:
Auch dies definiert keine Metrik, sondern repräsentiert nur eine Reihe von Metriken welche auf eine Funktion mit diesem Namen zugreifen. Äquivalent zu den anderen dieses Typs ist es hier möglich, zu jeder Netzausgabe relativ zur richtigen Feldnummer die Standardabweichung zu berechnen. So kann beispielsweise untersucht werden, ob das Lernverhalten einer Kostenfunktion dazu tendiert, an gewissen Stellen stärker zu schwanken.

6.2.5 Callbacks

Während des Trainings kann es von Vorteil sein, Subroutinen zu nutzen, um bestimmte Aspekte des Trainings zu erweitern. Teilweise ist hierfür eine benutzerdefinierte Trainingsschleife nötig. In dieser Arbeit konnten jedoch alle notwendigen Funktionalitäten durch sogenannte Callbacks realisiert werden. Alle Callbacks sind Unterklassen der Klasse *keras.callbacks.Callback*, in welcher bestimmte Funktionen überschrieben werden. Zu Beginn des Trainings werden Objekte davon als Liste der *model.fit*-Funktion[4] übergeben. Diese greift zu gegebener Zeit in den Objekten auf entsprechende Funktionen, wie *__init__*, *on_batch_end* oder *on_epoch_end*, zu und führt, nach Übergabe der aktuellen Metriken, des aktuellen Netzmodells, und ggf. weiteren Informationen, wie der aktuellen Epoche, dessen Inhalt aus. Es sind bereits diverse Standard-Callbacks implementiert, von denen folgende bei Bedarf genutzt wurden:

TensorBoard
Das Toolkit *Tensorboard* dient zur Visualisierung von Metriken und Netzinformationen. Durch den Callback werden diese, entsprechend der Parametrierung, in bestimmten Abständen gespeichert und können parallel über einen Internetbrowser ausgegeben werden. Dies ist besonders vorteilhaft für eine Live-Überwachung des Trainings und zur Betrachtung größerer Testreihen.

ModelCheckpoint
Mit diesem Callback werden, abhängig von einer übergebenen Metrik-Bezeichnung, entsprechend der Parametrierung zu bestimmten Zeitpunkten Sicherheitskopien des Modells erstellt. Auf diese Weise wird während des Trainings der aktuell beste Satz an Gewichten festgehalten. Bei Bedarf kann dieser bei der Fortsetzung eines Trainings, zur Evaluierung oder beim Einsatz eines Netzes geladen werden.

[4] Funktion eines Keras-Obektes eines Netzmodells, in welcher das eigentliche Training durchgeführt wird

Teilweise war der Umfang der erforderlichen Funktionalitäten für diese Untersuchungen nicht ausreichend. Es wurden daher zusätzlich folgende benutzerdefinierte Callbacks implementiert, welche über die Datei *custom_callbacks.py* eingebunden werden:

SaveMetrics
Alle Metriken eines Trainings handfest in einer Textdatei zu speichern, war oft sehr vorteilhaft. Diese kann beliebig ausgelesen, weiterverarbeitet und visualisiert werden, wodurch die Darstellungsmöglichkeiten erheblich erweitert und vereinfacht wurden. Zudem ist es möglich, der Datei allgemeine Informationen zu den Netzparametern und Trainingseinstellungen anzuhängen, sodass auch nachträglich verifiziert werden kann, was diese Metriken im Detail repräsentieren. Für diese Zwecke wird hier, üblicherweise am Ende jeder Epoche, zeilenweise, ähnlich dem CSV-Format, jede Metrik in eine Textdatei mit übergebenem Namen geschrieben.

LearningRateScheduler
Bei voranschreitendem Training kann es von Vorteil sein, die zu Beginn definierte Lernrate, welche den, durch den Backpropagation-Algorithmus ermittelten, Gradienten skaliert, zu verringern. Ansonsten kann es vorkommen, dass, die Abbildung des Gradienten auf ein oder eine Reihe von Gewichten, bei schmaleren Minima im Fehlerraum, eine alternierende, teilweise stark divergente Charakteristik aufweist. Dies kann in einer plötzlichen Fehlerzunahme resultieren, wodurch das Training stark behindert wird. Ist der Datensatz hinreichend groß, kann eine zu hohe Lernrate verhindern, dass das optimale Minimum gefunden wird, weil in jedem Iterationsschritt die Gewichte über das Minimum hinweg aktualisiert werden (vgl. [6]). Dennoch sind hohe Lernraten zu Beginn des Trainings präferabel, da das Training ansonsten aufgrund einer zu geringen Schrittweite sehr langsam erfolgt. Neben der Wahl eines geeigneten Optimierers, wurde daher ähnlich der Keras Dokumentation [40] ein Callback implementiert, welchem eine Liste mit Epochen und Lernraten übergeben wird. Ist die nächste Epoche in der Liste erreicht, wird die Lernrate aktualisiert und eine entsprechende Information in die die Logdatei geschrieben.

DynamicLearningRateScheduler
Eine Definition des Verlaufs der Lernrate vor dem Training weißt den Nachteil auf, dass der perfekte Zeitpunkt, wann eine Lernrate aktualisiert werden soll, geschätzt werden muss. Der Callback *LearningRateScheduler* wurde daher durch einen dynamischeren Ansatz ersetzt. Anstatt einer vorgegebenen Liste zu folgen, wird hier geprüft, ob der Fehler beginnt, größer zu werden. Abhängig von übergebenen Parametern zur erlaubten Steigung und einem Faktor, um welchen die Lernrate bei Überschreitung reduziert werden soll, werden die Gewichte des besten Modells der letzten zehn Epochen geladen, die Lernrate aktualisiert und ein Update in die Logdatei geschrieben. Die Lernrate kann somit zu Beginn des Trainings relativ hoch gesetzt werden und wird bei Bedarf automatisch skaliert.

EarlyStopping

Ein Callback zur Realisierung des *Early Stoppings* aus Kapitel 2.1.7 ist bereits in Keras implementiert. Leider hat dieser in Kombination mit anderen Callbacks nicht ausgelöst, weshalb eine eigene Version geschrieben wurde. Diese prüft nach jeder Epoche, ob sich die durch einen Übergabeparameter festgelegte Metrik verbessert hat. Ist dies nach einer definierten Anzahl an Epochen nicht der Fall, wird das Training automatisch beendet und ein Update in die Logdatei geschrieben.

6.3 Entwicklung der Kostenfunktion

Um die Unterschiede im Lernverhalten und die Vorteile des proprietären Ansatzes aus Kapitel 4.5 zu verdeutlichen, wurde in Abbildung 6.7 ein Netz, welches zuvor mit demselben Satz an Gewichten initialisiert wurde, mit verschieden Kostenfunktionen trainiert und anhand der Metriken aus Kapitel 6.2.4 ausgewertet.

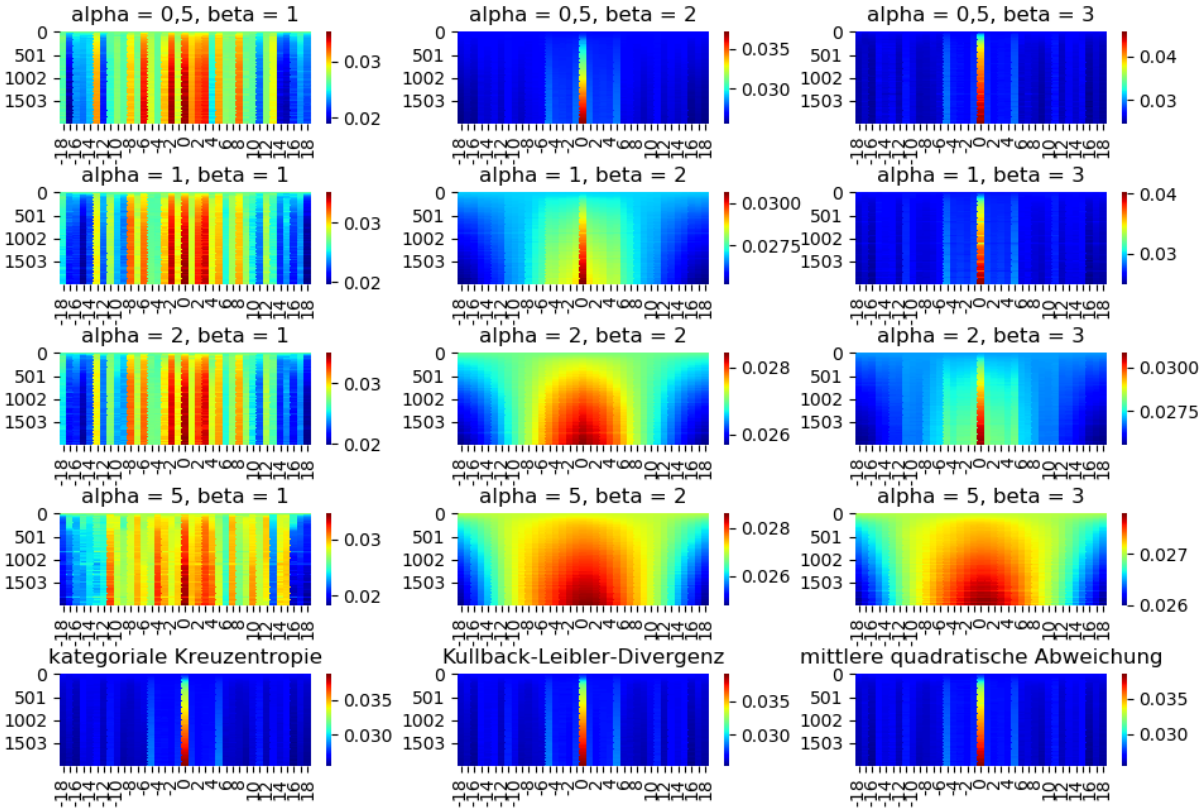


Abbildung 6.7: Übersicht mittlere Netzausgaben relativ zu den richtigen Ergebnissen auf den Trainings-Vektordaten der Spiralphase eines Netzes mit verschiedenen Kostenfunktionen über mehrere Epochen

Die oberen vier Zeilen beschreiben das gemittelte Ausgabeverhalten des Netzes, bei unterschiedlicher Parametrierung der in Kapitel 4.5 vorgestellten Kostenfunktion. Die Parameter γ und δ wurden auf ihren Standartwerten ($\gamma = 1$, $\delta = 18$) belassen und haben somit keinen Einfluss. Die letzte Zeile stellt die Verwendung beispielhafter konventioneller und in Keras bereits implementierter Kostenfunktionen aus Kapitel 2.1.3 dar.

Bei koventionellen Funktionen lässt sich gut erkennen, wie sich der Netzausgang des richtigen Feldes bereits kurz nach Beginn des Trainings deutlich von den anderen Ausgängen abhebt. Mit zunehmender Distanz der Felder tendiert das Netz zu einer etwas geringeren Ausgabe, obwohl die Kostenfunktionen hier weiter entfernte Felder genauso bewerten wie nahe gelegene. Das Netz lernt also auch hier eine Kombination von Gewichten, welche die realen pyhsikalischen Zusammenhänge, sofern sich das beurteilen lässt, zumindest teilweise nachbildet. Bei der proprietären Kostenfunktion wird diese Tendenz durch die Berücksichtigung der Distanz mit den Parametern α und δ deutlich stärker skaliert, was in weitaus breiteren Wahrscheinlichkeitsverteilungen der Netzausgabe resultieren kann. Ein solches Verhalten ist teilweise präferabel, da wie bereits in Kapitel 4.5 erwähnt, bei Feldern, welche nahe beieinander liegen, grundsätzlich auch eine ähnliche Wahrscheinlichkeit existiert, dass die Kugel auf diesen liegen bleibt. Die natürliche Tendenz der Netze, diese Ähnlichkeiten zu erkennen, wird bereits durch die Softmax-Funktion verstärkt, da so geringere Wahrscheinlichkeiten noch deutlich weiter verkleinert werden und somit auch bei konventionellen Ansätzen nicht allzu starke Fehler verursachen. Es hat sich jedoch gezeigt, dass eine genaue Skalierung dieses Effekts deutlich bessere Ergebnisse in den für diese Arbeit relevanten Bereichen

- Wahrscheinlichkeit richtiger Prädiktionen
- mittlere Abweichung der Prädiktion zum richtigen Feld
- Generalisierungsfähigkeit (gemessen an der Divergenz der Prädiktionswahrscheinlichkeit zwischen Trainings und Validierungsdaten)

erzielen konnte. Wird α im Verhältnis zu β zu groß gewählt (Bereich links und unten in der Abbildung), wird der Fehler so stark durch die Distanz bestimmt, dass auch größere Differenzen zwischen den *Ist*- und *Soll*-Werten, skaliert durch β, bei Netzausgängen nahe dem richtigen Feld einen zu geringen Einfluss auf den Fehler haben. Das Lernen fokussiert sich also auf die Verringerung des Fehlers bei großen Distanzen und wertet Fehler mit niedrigen Distanzen nahezu gleich. Dies resultiert bei $\beta = 1$ sogar darin, dass sich sehr starke Nebenmaxima ausbilden, da die Ableitung der Differenz nach den Netzausgängen im Verhältnis zur Distanz fast keinen Einfluss hat. Das andere Extrem findet sich im oberen rechten Teil der Abbildung. Hier hat die Distanz im Verhältnis zur Differenz einen so geringen Einfluss, dass die Plots denen der konventionellen Kostenfunktionen ähneln, welche die Distanz komplett nicht berücksichtigen.

An dieser Stelle ist es wichtig zu erwähnen, dass diese Verläufe zwar mit dem Prädiktionsverhalten der Netze korrelieren, mit ihnen aber nicht identisch sind. Als prädiziertes Feld gilt das mit der höchsten Wahrscheinlichkeit. Netze, die ausgeprägte Maxima bei dem richtigen Feld aufweisen, könnten dennoch bei jeder Prädiktion für ein beliebiges anderes Feld eine höhere Wahrscheinlichkeit ausgeben, welche sich hier aber im Mittel über die Felder verteilt. Abbildung 6.8 zeigt die mittlere Wahrscheinlichkeitsverteilung der Prädiktionen zu den in Abbildung 6.7 dargestellten Trainingsverläufen.

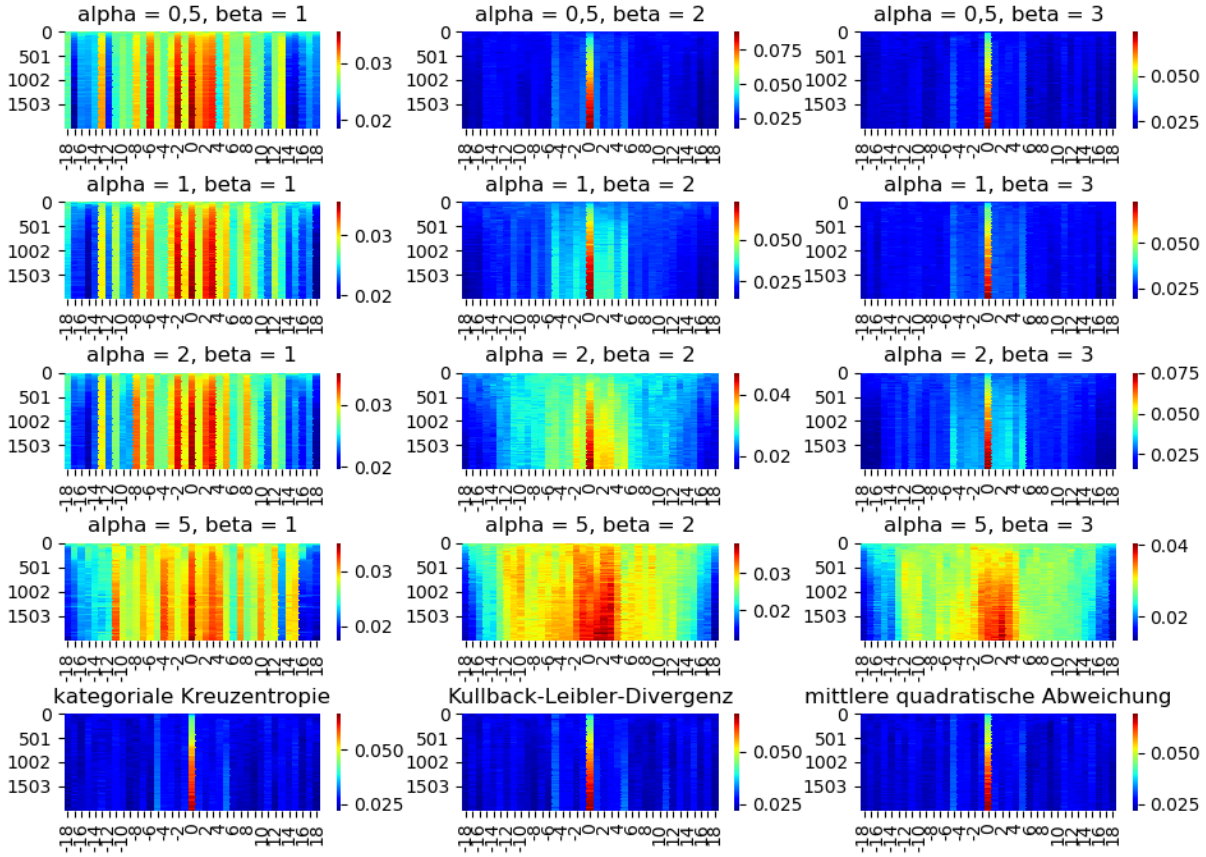


Abbildung 6.8: Übersicht Prädiktionswahrscheinlichkeit relativ zu den richtigen Ergebnissen der Trainings-Vektordaten der Spiralphase eines Netzes mit verschiedenen Kostenfunktionen über mehrere Epochen

Obwohl in Abbildung 6.7 die Verläufe der letzten Zeile und die Parameterkombinationen [$\alpha = 1$; $\beta = 3$] und besonders [$\alpha = 0{,}5$; $\beta = 3$] die höchsten Maxima aufweisen, ermöglicht die Kombination [$\alpha = 0{,}5$; $\beta = 2$] auf den Trainingsdaten die meisten richtigen Prädiktionen. Dennoch wird diese bei der Messung der mittleren Abweichung der Prädiktionen zum richtigen Feld von den Kostenfunktionen mit [$\alpha = 2$; $\beta = 3$] und noch deutlicher [$\alpha = 1$; $\beta = 2$] übertroffen, da diese im Verhältnis die Distanz zum richtigen Feld stärker mit einbeziehen.

Betrachtet man die Generalisierungsfähigkeit der Netze als normalisierte Differenz des Mittelwertes richtiger Prädiktionen von Trainings- und Validierungsdaten, weisen die Netze mit den Parametern [$\alpha = 2$; $\beta = 2$], [$\alpha = 5$; $\beta = 2$] und, [$\alpha = 5$; $\beta = 3$] die besten Ergebnisse auf. Diese Verläufe stechen besonders durch ihre runde Form hervor. Im Gegensatz zu den anderen hebt sich hier der mittlere Netzausgang des richtigen Feldes nicht von den Netzausgängen der benachbarten Felder ab (siehe Abbildung 6.7). Dieser Effekt ließe sich bei den anderen Parameterkombinationen durch eine Veränderung des Parameters γ realisieren, durch welchen ein Fokus auf exakte Prädiktionen belohnt oder bestraft werden kann. Ähnliche Verläufe sind vermutlich auch bei menschlichem Erlernen simplerer Zusammenhänge zu beobachten, da Menschen dank ihrer Generalisierungsfähigkeit in der Lage sind, physikalische Zusammenhänge zu verstehen und somit den Prädiktionsbereich einzugrenzen, aber nicht dazu tendieren, zufällige Zusammenhänge zu erkennen, welche richtige Prädiktionen im Verhältnis zu

Prädiktionen nahe den richtigen sprunghaft anheben. Äquivalent zu anderen Regularisierungsmethoden wird hier der Fokus einzelner Neuronen möglicherweise dadurch unterdrückt, dass der Gradient, durch die erhöhte Abhängigkeit der Distanz der Felder, in Kombination mit der stark variierenden Wahrscheinlichkeitsverteilung der Ausgänge, die Anpassung weniger relevanter Neuronen stärker mit einbezieht.

Für die Ermittlung der optimalen Parameterkombinationen muss die Performance auf den Validierungsdaten genauer betrachtet werden. Es hat sich gezeigt, dass Kostenfunktionen, welche auf den Trainingsdaten sogar schlechter performen als konventionelle Verfahren, durch eine deutlich bessere Generalisierungsfähigkeit auf den Validierungsdaten die besten Ergebnisse liefern. Die nachfolgenden Untersuchungen werden bei gleicher Netzstruktur ohne Regularisierungsverfahren durchgeführt. Abhängig von der Parametrierung dieser Verfahren würden diese ansonsten wiederum Einflüsse auf die optimale Parametrierung der Kostenfunktion haben, deren Effekt in Kapitel 6.4 genauer untersucht wird.

Die Bestimmung der optimalen Parameterkombination erfolgt iterativ. Das Ermitteln des globalen Optimums dieses vierdimensionalen Parameterraums erfordert sehr viel Rechenleistung. Aufgrund der begrenzten Ressourcen wurden die Tests mit etwa 20 % der Vektordaten über jeweils 2000 Epochen durchgeführt. Auf diese Weise konnte in kurzer Zeit eine Vielzahl an Kombinationen anhand von Metriken untersucht werden, welche die tatsächlichen Eigenschaften der Kostenfunktionen vermutlich ausreichend repräsentieren. Nachfolgend sind ausgewählte Kombinationen anhand oben genannter Kriterien gegenübergestellt. Ein Überblick über die gesamten untersuchten Parameterkombinationen befindet sich im Anhang B. Die Annotation entspricht hier dem folgenden Schema:

($\alpha|\beta|\gamma|\delta$):[Differenz der prozentualen Wahrscheinlichkeit richtiger Prädiktionen zwischen Trainings- und Validierungsdaten am Ende des Trainings]

Die Färbung gibt einen groben Überblick über diese Differenz, also ein ungefähres Maß, wie gut das jeweilige Netz die Daten generalisiert hat.

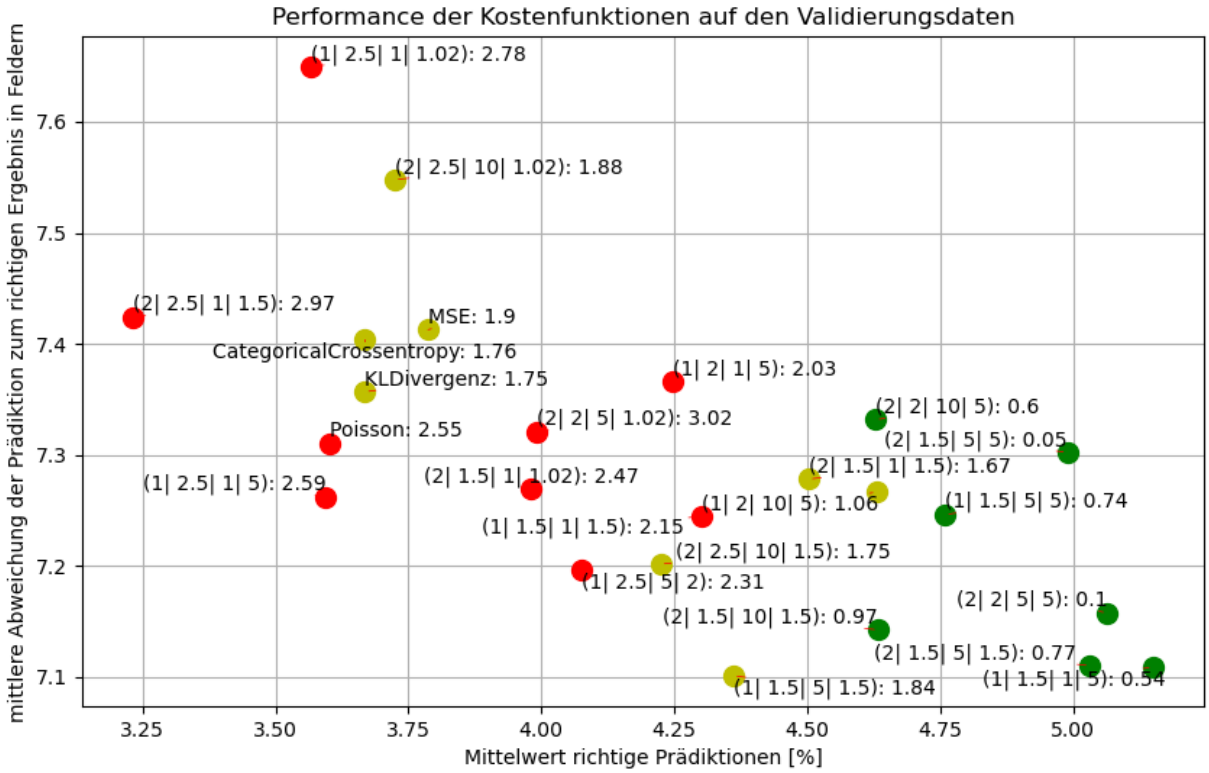


Abbildung 6.9: Ausschnitt eines Perfomance-Vergleichs der proprietären Kostenfunktion mit verschiedener Parametrierung und konventionellen Ansätzen auf den Validierungs-Vektordaten der Spiralphase

Es lässt sich gut erkennen, dass auch auf den Validierungsdaten diverse Parameterkombinationen gefunden werden konnten, welche konventionellen Ansätzen deutlich überlegen sind. Zudem ist eine Korrelation zwischen den Metriken untereinander und zu der Generalisierungsfähigkeit zu beobachten. Der Wertebereich der mittleren Abweichung der Prädiktionen ist jedoch so gering, dass allein durch die Schwankungen bei verschieden Initialwerten der Gewichte nur sehr grobe Rückschlüsse auf die Parametrierung gezogen werden können. An der Clusterbildung der Farben lässt sich erkennen, dass, anstelle der Performance auf den Trainingsdaten, die Generalisierungsfähigkeit der Netze die Validierungsergebnisse dominiert. Für die Ermittlung des Optimums wurden in jedem Iterationsschritt in Bezug auf die oben genannten Kriterien die besten Kostenfunktionen ausgewählt und deren Parameter jeweils in separaten Tests leicht verschoben. Abhängig von den Initialgewichten, den Hyperparametern und dem Datensatz, werden vermutlich verschiedene Parameterkombination favorisiert. Um das tatsächliche Optimum für jeden Ansatz zu finden, müssten diese Untersuchungen daher wiederholt werden, was sehr zeitaufwendig ist. Stattdessen wurden die hier gewonnenen Erkenntnisse als Grundlage dafür verwendet, anhand der Metriken — bei nachfolgenden Untersuchungen zur Netzoptimierung — leichte Anpassungen in der Parametrierung vorzunehmen. Die Vermutung aus Kapitel 4.5, Trainingsvorteile durch Einbezug von Ähnlichkeiten der Ausgabeklassen gewinnen zu können, konnte bei beliebigen Hyperparametern bestätigt werden.

6.4 Training und Hyperparametertuning

Zur Durchführung der geplanten Testreihen in gegebener Zeit, musste der Fokus auf eine möglichst zeitoptimale, aber gleichzeitig hinreichend genaue Herangehensweise gesetzt werden. Allgemeinere Tests, wie eine Untersuchung auf hinreichende Datensatzgröße, der Einfluss der Vervielfältigung der Daten oder deren Auflösung, sowie diverse Tests zur Wahl der Hyperparameter wie der Batchgröße, der Lernrate oder der Optimierer bzw. deren Parametrierung wurden ausschließlich mit Vektordaten aus der Spiralphase untersucht. Vektordaten haben eine verhältnismäßig hohe Informationsdichte und ermöglichen somit ein Training mit relativ kleinen Netzeingängen, was die Anzahl an notwendigen Gewichten deutlich verringert. Zudem kann der gesamte Datensatz in den Arbeitsspeicher geladen werden. Durch die stark verkürzten Zugriffszeiten wurde das Training somit um ein Vielfaches beschleunigt. Eine Priorisierung der Daten der Spiralphase hat den Vorteil, dass die zu erlernenden Korrelationen zwischen den Ein- und Ausgangsdaten simpler sind. Getestete Einflüsse lassen sich durch die hieraus resultierenden größeren Performance-Unterschiede leichter erkennen. Tests zur Beschleunigung des Trainings, wie die Wahl der Batchgröße, der Lernrate oder des Optmierers, wurden, sofern möglich, vorgezogen. Um die Schwankungen der Messwerte beurteilen zu können, wurde jeder Test in der Regel mindestens dreimal mit verschieden Initialwerten der Gewichte durchgeführt. Ähnlich zur Erzeugung der Datensätze, wurden die Trainingsprogramme (*FFNN_vectordata.py* und *conv2d_videodata.py)* möglichst adaptiv aufgebaut. Für längere Testreihen wurde der Hauptteil des Programms als Funktion definiert, an welche die aktuell zu testenden Parameteränderungen übergeben wurden. Die Namensgebung der gespeicherten Metriken und Sicherheitskopien erfolgt automatisch nach demselben Schema und basiert auf der Netzversion, den wichtigsten Parametern, einem optionalen Identifizierungscode und einem Zeitstempel. Programme zur Auswertung und Visualisierung können somit bei Bedarf bereits aus dem Namen bestimmte Informationen extrahieren. Die Visualisierung von Metriken erfolgt im Wesentlichen durch das Programm *visualize_metric_data.py*. Insbesondere in den anfänglichen Untersuchungen wurde zudem das Toolkit Tensorboard verwendet. Neben der Möglichkeit, die Verläufe der Metriken während des Trainings automatisiert zu visualisieren, ist mit Tensorboard, ohne großen Implementierungsaufwand eine Ausgabe von Histogrammen über die Entwicklung der Verteilungen der Gewichte möglich. Hierdurch kann überprüft werden, ob sich der Trainingsverlauf plausibel verhält oder Gewichte dazu tendieren, extreme Werte anzunehmen. Des Weiteren wurden verschiedene Programme zur Evaluierung und Visualisierung der trainierten Gewichte, Metriken und Datensätze gschrieben, anhand welcher deren Richtigkeit und Qualität validiert werden konnte. In den nachfolgenden Unterkapiteln werden einzelne Tests zur allgemeinen Optimierung des Trainingsverhaltens genauer beschrieben.

6.4.1 Wahl der Hyperparameter

Bei der Optimierung neuronaler Netze existiert eine Vielzahl an Hyperparametern welche sich wiederum untereinander beeinflussen können. Deren genaue Untersuchung stellt jedoch in den meisten Fällen nicht den Fokus dieser Arbeit dar. Für einen Großteil dieser Parameter wurden daher die Einflüsse untereinander vernachlässigt und stattdessen ein Optimum, basierend auf den Optima vorheriger Untersuchungen oder einschlägiger Literatur, bestimmt.

Bei der Wahl der Batchgröße muss zwischen den verwendeten Datensätzen unterschieden werden, da die hierdurch definierte Genauigkeit des Gradienten und die Allokation des Arbeits- bzw. des Grafikkartenspeichers, mit der Größe und Diversität der Daten variiert. Allgemein ist es vorteilhaft, die Batches so groß zu wählen, wie es der Speicher erlaubt, da so ein Gradient erzeugt wird, welcher so genau wie möglich auf das Minimum des durch die Kostenfunktion aufgespannten Fehlerraums zeigt. Solange der erzeugte Gradient hinreichend genau ist, kann das Training dennoch durch kleinere Batches deutlich beschleunigt werden, da bei einer Verkleinerung der Batchgröße die Anzahl der Aktualisierungen der Gewichte im Verhältnis zu ihrer Berechnungszeit erhöht wird. Die Diversität der möglichen Kugeltrajektorien lässt vermuten, dass der hier erzeugte Datensatz wahrscheinlich nicht einmal ansatzweise ausreichend Informationen zur Verfügung stellt, um alle Eventualitäten hinreichend abzubilden. Selbst bei Verwendung des gesamten Datensatzes wird daher vermutlich kein hinreichend genauer Gradient erzeugt. Es hat sich dennoch gezeigt, dass bei Vektordaten ab einer Batchgröße von etwa 250.000 Datenpunkten, die maximal mögliche Performance nicht durch die Genauigkeit des Gradienten definiert wurde. Dies lässt sich vor allem dadurch erklären, dass Daten eines Videos zum Großteil dieselbe Trajektorie definieren und somit keine weitere Diversität, bzw. zu dem Gesamtgradienten keine weitere Richtung, beitragen. Dies wird durch die Blockbildung in Abbildung 7.4 aus Kapitel 7.2 untermauert. Beim Training der Differenzbilder ist die Informationsdichte so gering, dass besonders der, für die Tensoren der Gewichte und aktuellen Werte des Netzes erforderliche, Speicher die verfügbaren Hardwareressourcen sehr schnell übersteigt. Abhängig davon, wie die Netzstruktur und die Anzahl und Größe der Filterkerne gewählt wird, sind hier nur Batchgrößen mehrerer hundert Datenpunkte möglich.

Die Wahl des Optimierers und dessen Parametrierung kann ein sehr wichtiger Faktor sein, ob und wie schnell ein Minimum im Fehlerraum gefunden wird (vgl. [13]). In Keras sind bereits eine Reihe konventioneller, allgemein bekannter, Optimierer implementiert, von denen folgende in ihrer Standardparametrierung untersucht wurden:

- SGD
- RMSprop
- Adam
- Adadelta
- Adagrad
- Adamax
- Nadam

Tendenziell wurden mit dem Optimierer Adam die besten und schnellsten Ergebnisse erzielt. Mit diesem wurde daher untersucht, ob sich das Trainingsverhalten durch entsprechende Parametrierung weiter verbessern ließ. Hierbei konnte sich jedoch nicht auf Vermutungen oder Erfahrungswerte berufen werden, weshalb eine automatisierte Parameteroptimierung verwendet wurde. Diese ist in Keras bereits unter dem Namen *Keras-Tuner* implementiert. Hierdurch ist es möglich, mittels Ummantelung der Modellbildung des Netzes mit einer parametrierbaren Funktion, anstelle der üblichen Trainingsschleife, einen der verfügbaren Algorithmen zu nutzen, um ein definiertes Optimum zu finden. Die Freiheitsgrade lassen sich durch Ersetzen bestimmter Parameter mit Objektzugriffen definieren. Auf diese Weise ließen sich für den Adam-Optimierer mehrere Parameterkombinationen finden, welche gute Ergebnisse erzielten, jedoch keine, welche die Standardparametrierung übertraf. Auf die Nutzung des Tuners für andere Parameteroptimierungen, wie die der Kostenfunktion, wurde verzichtet, da die Wahrscheinlichkeit mit hinreichender Effizienz eine so zeitkritische Aufgabe zu lösen wiederum stark von der Wahl und Parametrierung des Tuning-Algorithmus abhängig ist und daher wenig erfolgversprechend war.

Nach dem Abschluss allgemeiner Voruntersuchungen, wurde die Performance durch Tests verschiedener Netzstrukturen, basierend auf Tendenzen ihrer Prädiktionsverhalten, in Kombination mit Regularisierungsmethoden nachfolgender Abschnitte bis an die Grenzen verfügbarer Hardwareressourcen optmiert. Die Aktivierungsfunktionen innerer Schichten wurden bei der Relu-Funktion belassen. Da diese keine Probleme verursacht hat und bereits als Standard beim Training tieferer Netze definiert werden kann (vgl. [6]), wurden auch aus zeitlichen Gründen keine Alternativen untersucht.

6.4.2 Netzoptimierung durch Dropout

Der Einfluss des Dropout-Verfahrens variiert stark mit der verwendeten Kostenfunktion. Erkenntnisse aus Kapitel 6.3 lassen sich daher nur als Richtwert verwenden und müssen in Kombination mit Dropout erneut getestet werden. Für diese Untersuchungen wurde die Anzahl der Neuronen pro Schicht in mehreren Abstufungen so gewählt, dass das Netz beim Training allgemein deutlich zum Overfitting neigt. Mit dem Ziel, den Anteil an richtigen Prädiktionen zu maximieren, wurden verschiedene Netztiefen und Kostenfunktionen mit unterschiedlichem Dropout-Anteil in jeder Schicht getestet. Anfängliche Untersuchungen haben gezeigt, dass Netze mit proprietären Kostenfunktionen aus Kapitel 6.3, welche sich durch eine relativ gute Generalisierungsfähigkeit auszeichnen, im Allgemeinen Performance-Optima bei geringerem Dropout-Anteil aufweisen. Dies untermauert die Hypothese, dass diese Art von Kostenfunktion verfügbare Neuronen tatsächlich gleichmäßiger nutzt und somit auch ohne Dropout eine höhere Redundanz erreicht. Bei verhältnismäßig kleinen Netzstrukturen konnte daher beobachtet werden, wie ein relativ gutes Prädiktionsverhalten durch Dropout aufgelöst wurde. Bewertet man die Performance eines Netzes jedoch nur durch die Wahrscheinlichkeit einer richtigen Prädiktion (in Abbildung 6.10 durch die Null auf der Abszisse dargestellt), bewirkt das Dropout-Verfahren auch in diesem Beispiel, durch das Auflösen des verschobenen Maximums, eine bessere Performance.

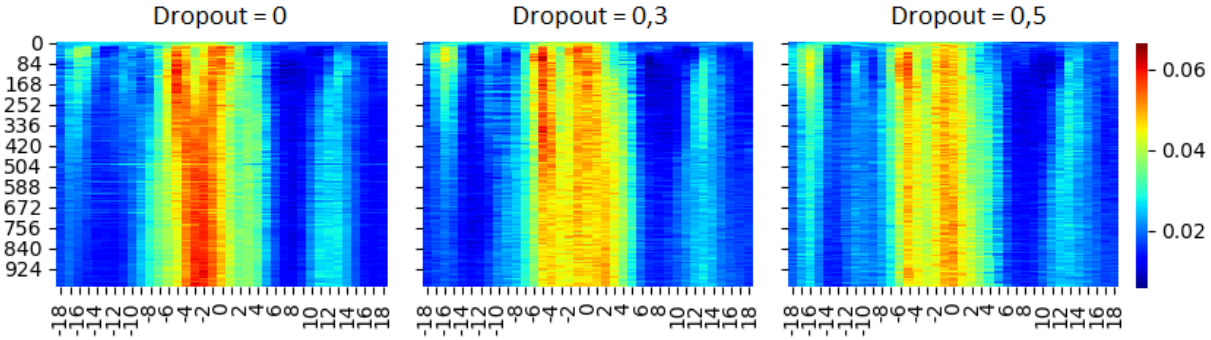


Abbildung 6.10: Einfluss Dropout auf das Prädiktionsverhalten relativ zu den richtigen Ergebnissen von Validierungs-Vektordaten der Spiralphase bei kleiner Netzstruktur und stark generalisierender Kostenfunktionen über mehrere Epochen

Das Auflösen des verschobenen Minimums, insbesondere bei eher kleineren Strukturen, lässt sich dadurch erklären, dass den verhältnismäßig stärker beeinträchtigten Netzen während des Trainings deutlich weniger Neuronen zur Verfügung stehen, um das chaotische Verhalten zu approximieren. Ausgeprägte Maxima auf den Validierungsdaten, hervorgerufen durch zufällige abstrakte Ähnlichkeiten mit den Trainingsdaten, werden somit stärker unterdrückt. Hiervon profitieren besonders Netze mit der Art von Kostenfunktion, bei welcher das Lernen auf das Optimieren der Trainingsdaten fokussiert ist. Das Netz erlernt diese zwar deutlich schneller, begünstigt aber auch stärker zufällige Korrelationen, welche nur für diesen Datensatz typisch sind, was bei den Validierungsdaten ohne Dropout in Verläufen mit stärkeren Abstufungen und deutlicheren Nebenmaxima resultieren kann (siehe Abbildung 6.11).

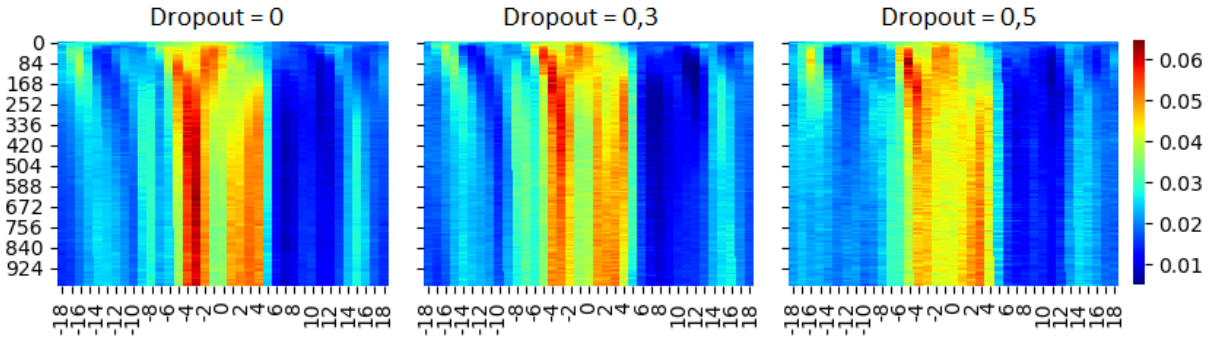


Abbildung 6.11: Einfluss Dropout auf das Prädiktionsverhalten relativ zu den richtigen Ergebnissen von Validierungs-Vektordaten der Spiralphase bei kleinerer Netzstruktur und der Kreuzentropie als Kostenfunktion über mehrere Epochen

Untersuchungen mit größeren Netzen haben gezeigt, dass Kostenfunktionen, wie die Kreuzentropie, durch ihre deutlich bessere Performance auf den Trainingsdaten, in Kombination mit Dropout ohnehin gut generalisierenden Kostenfunktionen überlegen sind. Abbildung 6.12 zeigt eine Gegenüberstellung des Prädiktionsverhaltens eines Netzes mit einer größeren Struktur — unter Anwendung der Kreuzentropie — und einer gut generalisierenden proprietären Kostenfunktion mit dem jeweils ermittelten optimalen Dropout-Anteil in den entsprechenden Schichten. Während bei dem proprietären Ansatz durch den stärkeren Fokus auf die Minimierung der Fehler entfernter Felder exakte Prädiktionen stark vernachlässigt werden, kann bei der Kreuzentropie, aufgrund des deutlich höheren Anteils an richtigen Prädiktionen auf den Trainingsdaten, in Kombination mit Dropout auch auf den Validierungsdaten in Bezug auf exakte Prädiktionen ein gutes Ergebnis erzielt werden.

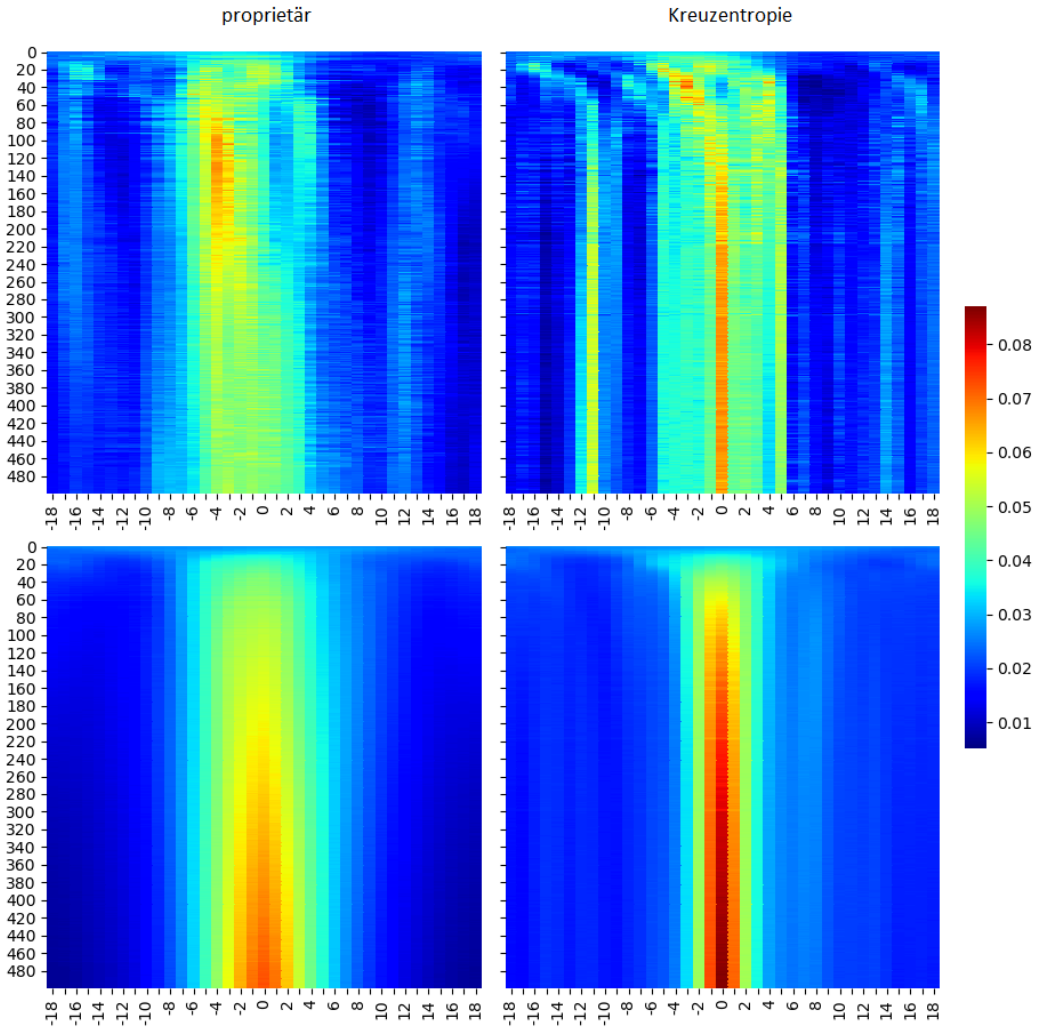


Abbildung 6.12: Gegenüberstellung Prädiktionsverhalten relativ zu den richtigen Ergebnissen mit der Kreuzentropie und einem gut generalisierendem Ansatz mit jeweils optimalem Dropout-Anteil auf den Trainingsdaten (unten) und den Validierungsdaten (oben) von Vektoren der Spiralphase über mehrere Epochen mit einem relativ breiten Netz

Dies legt die Vermutung nahe, dass proprietäre Kostenfunktionen, welche auf den Trainingsdaten deutlich besser performen als konventionelle wie die Kreuzentropie, durch gezielte Regularisierung auf den Validierungsdaten eine noch bessere Performance erreichen können. Es hat sich jedoch herausgestellt, dass bei einem solchen Lernverhalten auch mit Dropout keine deutliche Verbesserung in der Generalisierungsfähigkeit der Netze erreicht werden konnte, welche die der Kreuzentropie bei gleicher Netzgröße übertrifft. Genauere Untersuchungen der Plots haben jedoch gezeigt, dass während des Trainings im Verhältnis zur Kreuzentropie kurzzeitig deutlich bessere Ergebnisse erzeugt wurden. Diese Performance-Spitzen lassen aber vermuten, dass hier zufällige Ähnlichkeiten beider Datensätze erkannt wurden, welche aber nicht repräsentativ für unbekannte Daten sind (siehe Anhang C). Dennoch konnte ein Kompromiss in der Parametrierung des proprietären Ansatzes gefunden werden, welcher in Bezug auf den Anteil an exakten Prädiktionen auf den Trainings- und Validierungsdaten die Performance der Kreuzentropie bei gleicher Netzstruktur mit und ohne Dropout deutlich überragt:

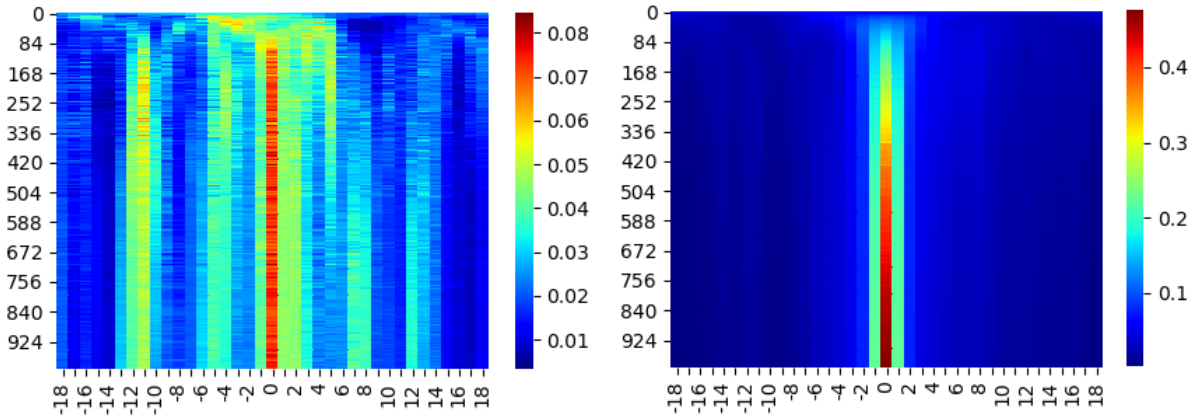


Abbildung 6.13: Prädiktionsverhalten relativ zu den richtigen Ergebnissen mit optimalem proprietärem Ansatz bei einem Dropout-Anteil von 0,5 auf den Validierungsdaten (links) und den Trainingsdaten (rechts) von Vektoren der Spiralphase über mehrere Epochen

Erfahrungen haben gezeigt, dass bei einer Deaktivierung von Neuronen in früheren Schichten oft keine besseren Ergebnisse erzielt wurden. Inwieweit dieser Effekt tatsächlich durch den Einfluss von Dropout verursacht wurde, lässt sich aufgrund mangelnder Messwerte schwer sagen. Eine mögliche Erklärung wäre, dass die Informationsaufbereitung der Eingangsdaten in den ersten Schichten meist auf eine sehr ähnliche Weise geschieht, sodass eine Redundanz durch Dropout weniger Einfluss hat.

6.4.3 Netzoptimierung durch Batch-Normalisierung

Um einen positiven Einfluss durch Batch-Normalisierung zu verifizieren, wurde diese an Netzen unterschiedlicher Tiefe getestet. Da das Trainingsverhalten auch hier mit der Wahl der Kostenfunktion variieren kann, ist, entsprechend den Untersuchungen zum Dropout-Verfahren, ein weiterer Vergleich mit verschiedenen Ansätzen notwendig.

Unabhängig von der Parameterwahl, wurden auf den Trainingsdaten in fast allen Tests, sowohl bei der Lerngeschwindigkeit, als auch in Bezug auf die maximal erreichte Performance deutlich bessere Ergebnisse gemessen. Eine Verbesserung der Validierungsergebnisse scheint jedoch mit der Netztiefe bzw. Netzgröße zu korrelieren. Während der Effekt des Overfittings bei flachen Netzen sogar verstärkt wurde, konnte die Performance auf unbekannten Daten bei tieferen Netzen fast um 50 % gesteigert werden. Abhängig von der implementierten Kostenfunktion, wurden auch hier deutliche Unterschiede beim Lernverhalten gemessen. Ähnlich zu den Tests zur Untersuchung des Dropout-Verfahrens, haben proprietäre Kostenfunktionen am besten performt, welche ohne Regularisierungsmethoden einen Kompromiss aus guter Trainings-Performance und gutem Generalisierungsverhalten darstellen. Eine beispielhafte Übersicht über die Testergebnisse zu dieser Untersuchung ist im Anhang D zu finden.

7 Analyse und Auswertung der Ergebnisse

Während der Untersuchung und Optimiererung einzelner Netze wurde eine Vielzahl an Testreihen durchgeführt, deren schriftliche Auswertung in ihrer Gänze nicht zweckmäßig ist. In diesem Kapitel werden die jeweils besten Ergebnisse genauer analysiert und ausgewertet. Ausschnitte weiterer interessanter Messungen befinden sich im Anhang. Die größte Herausforderung bestand darin, ein Lernverhalten zu ermöglichen, welches die Daten möglichst gut generalisiert, ohne Eingeständnisse in der Prädiktionsgenauigkeit zu machen. Der Fokus nachfolgender Abschnitte liegt auf der Analyse des Lernverhaltens bei Implementierung der in Kapitel 4.3 definierten Netzeingänge, der Untersuchung von Faktoren, welche die erreichbare Performance begrenzen und dem Vergleich der Performance zu verwandten Forschungen mittels Modellbildung.

7.1 Analyse der Datensätze

Während der Aufbereitung der Videos wurden alle Vektordaten in ihrer Rohform separat abgespeichert und analysiert. Insbesondere in Bezug auf die Diversität der Daten ist es interessant, beurteilen zu können, inwieweit diese mit der Größe des Datensatzes korreliert. Hierfür wurden alle Koordinaten während der Spiralphase in einer Matrix mit geringer Auflösung zusammengefasst. Die Häufigkeit ähnlicher Kugelverläufe die gleichen Elemente der Matrix zu treffen, nimmt bei der richtigen Auflösung stark zu und kann somit hervorgehoben werden.

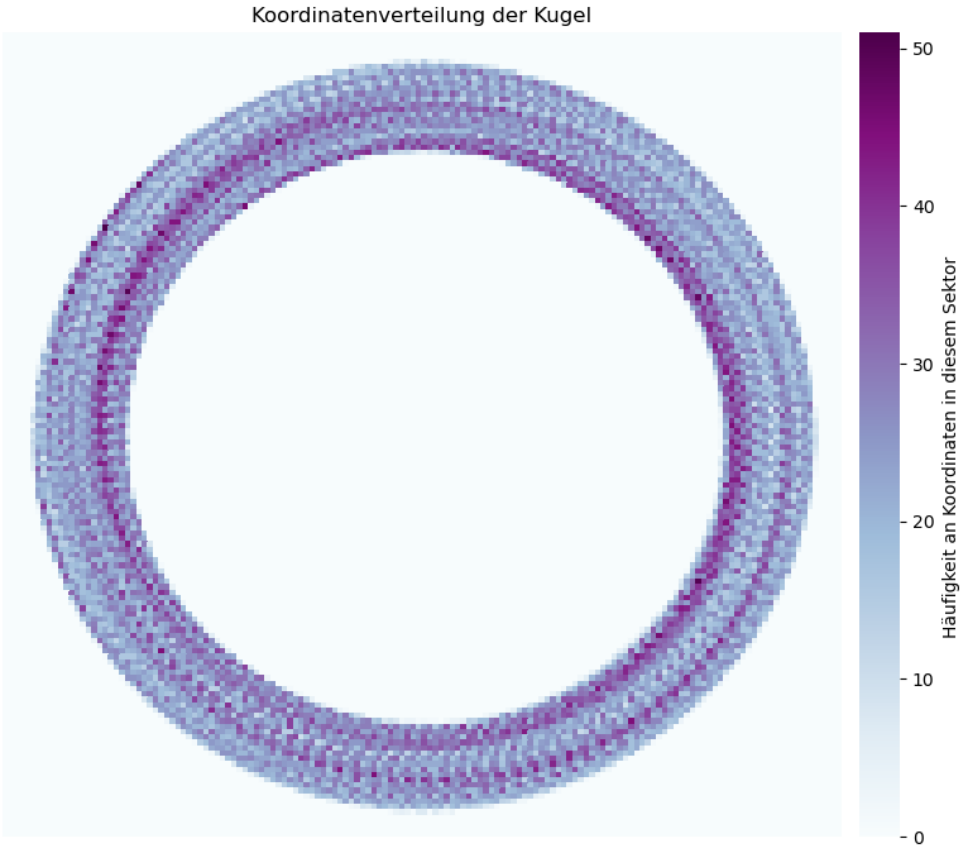


Abbildung 7.1: Verteilung der Kugelkoordinaten in den Aufnahmen des gesamten Datensatzes während der Spiralphase

In einem optimalen Datensatz würde jedes der farbigen Felder die gleiche Häufigkeit darstellen. Es wird jedoch deutlich, dass sich die Kugel öfter auf einer ähnlichen Bahn Richtung Mitte bewegt. Dies wirft die Frage auf, ob die Ursache dafür auf Eigenschaften des Tisches — wie Unebenheiten seiner Oberfläche oder positionsabhängige Reibungen — zurückzuführen ist, welche sich durch das Netz erlernen lassen und für diesen Tisch allgemein gültig sind, oder hier ein einseitiges Wurfverhalten des Croupiers datensatzspezifische Schwerpunkte von Korrelationen fördert und somit die Generalisierungsfähigkeit beeinträchtigt. Um dies zu beantworten werden in Abbildung 7.2 die Vektoren jedes Videos berücksichtigt, welche den Beginn und das Ende der jeweiligen Spiralphase definieren.

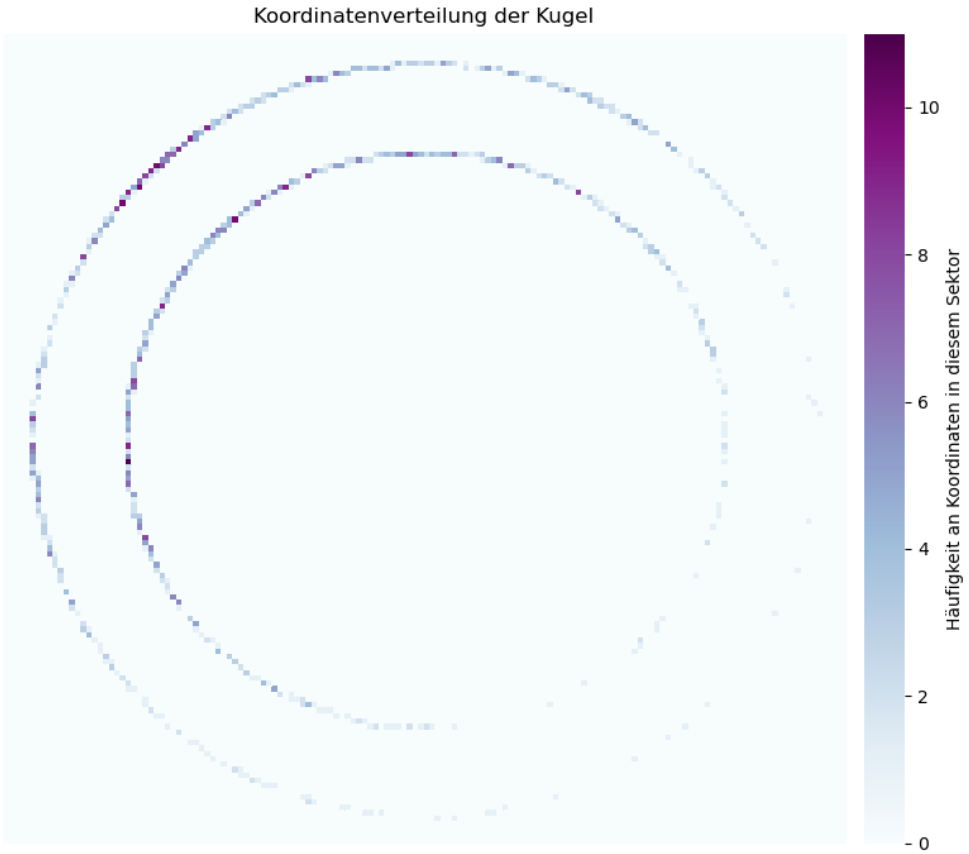


Abbildung 7.2: Verteilung der Kugelkoordinaten zu Beginn und Ende der Spiralphase in den Aufnahmen des gesamten Datensatzes

Betrachtet man den äußeren Ring, lässt sich gut erkennen, dass der Verlauf der Orte, an welchen die Kugel die Außenwand verlässt, einer Normalverteilung ähnelt, welche, durch die Tatsache, dass die Kugel immer an der gleichen Stelle eingeworfen wurde, mit hoher Wahrscheinlichkeit auf die Geschwindigkeitsverteilung beim Einwurf zurückzuführen ist. Die Verteilung der Orte an, welchen die Kugel den hier definierten Bereich der Spiralphase wieder verlässt, unterscheidet sich leicht von der äußeren und scheint mehr geglättet. In diesen Unterschieden werden inkonsistente Einflüsse, wie Unebenheiten des Bodens, deutlich.

Durch die Chaotik resultieren ähnliche Verläufe in der Spiralphase dennoch in sehr verschiedenen Trajektorien und haben somit auch stark verschiedene Bezüge zu dem Ergebnis. Dennoch existiert eine Tendenz zu einem bestimmten relativen Wertebereich der Eingangsdaten, dessen entsprechende Ähnlichkeiten in den Abbildungen der zugehörigen Gradienten auf die vorderen Schichten des Netzes vermutlich den Fokus des Trainings definieren. Derselbe Verlauf ist jedoch auch bei den Validierungsdaten aufgetreten, weshalb der Performanceunterschied der Messungen bei der Validierung nicht damit begründet werden kann. Als Folge ist jedoch zu erwarten, dass bei Würfen anderer Personen tendenziell schlechter prädiziert wird.

Gemäß der Vermutung aus Kapitel 6.1.4, sind die Kugelverläufe, ohne Einbezug der Relation zur Drehscheibe, zwar unabhängig zu den einzelnen Klassen, werden aber ohne Vervielfältigung dennoch klassenspezifisch erlernt. Mit einer Vervielfältigung der Daten durch ein Drehen der Drehscheibe und der daraus resultierenden Gleichverteilung, wird dieses Problem weitestgehend vermieden. Dennoch ist eine Vervielfältigung der Daten nicht ideal und und bringt somit einen gewissen Qualitätsverlust mit sich. Die Verteilung der Datenqualität und der damit einhergehenden unverfälschten Korrelationen, bleibt somit erhalten was sich theoretisch auf die Generalisierungsfähigkeit einzelner Klassen auswirken kann. Um deren möglichen Einfluss zu überprüfen wurden in Abbildung 7.3 die Häufigkeiten der Videos der einzelnen Klassen im Trainingsdatensatz dargestellt.

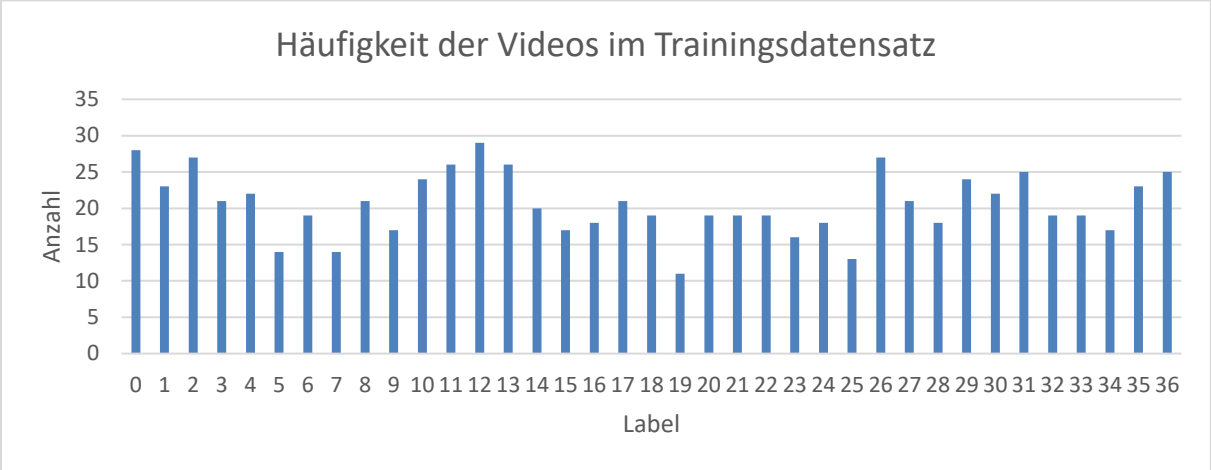


Abbildung 7.3: Häufigkeit der Videos zu jedem Roulette-Ergebnis im Trainingsdatensatz

Obwohl Videos mancher Klassen mehr als doppelt so oft vertreten sind wie andere, wird die statistisch zu erwartende Gleichverteilung dennoch deutlich. Der Einfluss der Qualitätsunterschiede ist somit vermutlich so gering, dass sich eine klassenspezifische Netzperformance hierdurch nicht begründen ließe.

7.2 Einfluss der Datensatzgröße auf die Performance

Damit eine Aussage darüber getroffen werden kann, bis zu welchem Grad es möglich ist, mit diesem Versuchsaufbau durch neuronale Netze das Verhalten eines Roulettespiels abzubilden, ist es wichtig, beurteilen zu können, ob die Größe des verwendeten Datensatzes hinreichend ist, das System in einer Genauigkeit zu beschreiben, die es dem Netz ermöglicht die Lücken vernünftig zu generalisieren. Die Komplexität und Diversität der Trajektorien lässt vermuten, dass die 761 für den Trainingsdatensatz aufgenommenen, Videos die notwendige Anzahl weit unterschreiten. Ist dies der Fall, müsste eine Kürzung des Datensatzes theoretisch eine sichtbare Verschlechterung der Performance nach sich ziehen.

Hierfür wurde ein Programm geschrieben, welches , unter der Bedingung, von jeder Klasse mindestens ein Video zu berücksichtigen, einen vorgegebenen Prozentsatz zufällig ausgewählter Daten kopiert. In der Aufbereitungskette gemäß Abbildung 6.1 wurde dieser Arbeitsschritt bereits vor der Verfielfältigung der Daten implementiert, da ansonsten zwar die Anzahl, aber nicht die Diversität der Daten nach der prozentualen Vorgabe gekürzt werden würde. Entsprechend anderen Tests zur Datenmanipulation, wurden auch hier ein Netz und eine Kostenfunktion verwendet, welche wenig anfällig gegenüber den Initialwerten der Gewichte sind. Von Regularisierungsmethoden wurde hier kein Gebrauch gemacht, da diese oft zu stärkeren Abweichungen bei wiederholtem Training gleicher Netze geführt haben.

Tabelle 7.1: Übersicht Einfluss der Datensatzgröße auf die Netzperformance

Anteil verwendeter Datensatz	20 %	40 %	60 %	80 %	100 %
Maximalanteil an erreichten, stabilen, richtigen Prädiktionen	4,5 %	5,5 %	6,3 %	6,0 %	6,0 %
mittlere Abweichung der Prädiktionen in Feldern	7,5	8	7,8	7,5	7,5
Differenz Anteil richtiger Prädiktionen Trainingsdaten - Validierungsdaten bei gleichem Trainingsfortschritt	18,5 %	20,0 %	16,0 %	15,5 %	11,7 %

Obwohl die Performance auf den Trainings- und Validierungsdaten bei geringerem Datenvolumen deutlich stärker divergiert, ermöglicht ein Anteil oberhalb von 50 % auf den Validierungsdaten keine Verbesserung der Wahrscheinlichkeit richtiger Prädiktionen. Eine naheliegende Ursache wäre, dass eine zu kleine Netzstruktur verhindert, das System, trotz ausreichender Trainingsdaten, auf komplexere Weise abzubilden und somit die maximale Performance begrenzt. Tests haben ergeben, dass auch mit dieser Netzstrukur mindestens 8,5 % der Validierungsdaten richtig prädiziert werden können. Da das Netz auch hier stark zum Overfitting neigt, liegt die Ursache vermutlich woanders. Eine weitere Möglichkeit wäre, dass, aufgrund der Qualität der Daten, ein Limit erreicht ist, welches allenfalls mit Regularisierungsmethoden überschritten werden kann. Die Untersuchungen in Kapitel 6.1.3 zum Einfluss der Vektorauflösung haben jedoch bestätigt, dass deren Potential längst nicht ausgeschöpft wurde. Da die Wahrscheinlichkeit einer richtigen Prädiktion allgemein sehr gering ist, hat das Netz eventuell simplere Bezüge erlernt, welche in den Trainingsdaten mit einer gewissen Redundanz vertreten sind und womöglich bei Datenmengen über 40 % nicht gekürzt wurden. Das würde jedoch bedeuten, dass die Netze auf den Validierungsdaten bei gleichen Datenpunkten auch ähnliche Ausgaben erzeugen. Wie in Abbildung 7.4 zu sehen ist, trifft dies ab 40 % der Trainingsdaten auch vermehrt zu.

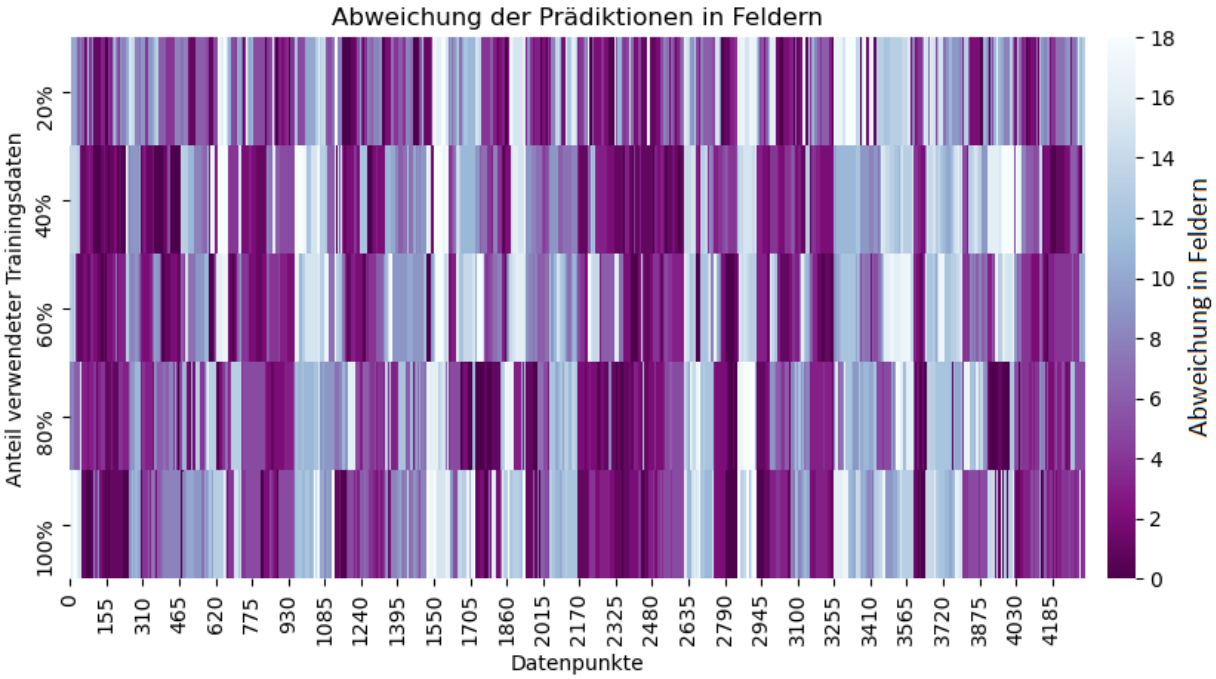


Abbildung 7.4: Übersicht Abweichung der Prädiktionen jedes Datenpunktes der Validierungs-Vektordaten der Spiralphase zum richtigen Ergebnis in Relation zum Anteil verwendeter Trainingsdaten mit jeweils optimaler Netzperformance bei gleichen Hyperparametern

Auffällig ist, dass oftmals ganze Blöcke von Daten sehr ähnliche Ausgaben erzeugen. Dies lässt sich darauf zurückführen, dass der Vektordatensatz eines Videos beim Zuschnitt durch die fortlaufende Nummerierung, bei dieser Evaluierung in gleicher Reihenfolge dargestellt wird. Hat das Netz den Bezug eines Videos erlernt, erzeugt es durch die Ähnlichkeiten der Eingangsdaten für alle Daten eines Videos vergleichbare Ergebnisse. Scheinbar existieren auch diverse Videos in den Validierungsdaten deren Bezüge von keinem der Netze verstanden wurden. Vermutlich sind in dem Trainingsdatensatz keine, oder nur sehr wenige, Beispiele enthalten, welche diesen Bezügen hinreichend ähneln. Betrachtet man die Performance bei 20 % der Trainingsdaten im Verhältnis zu den übrigen, tendieren die Blöcke dazu, kleiner zu werden und somit bei Daten von gleichen Videos andere Ergebnisse zu erzeugen.

Die Abhängigkeit der Performance zur Datensatzgröße ist vermutlich stark nichtlinear und wird in großem Ausmaß durch die Diversität der Daten beeinflusst. In Anbetracht der unvermeindlichen Schwankungen durch das Training ist die Wahrscheinlichkeit hoch, dass bei einer Berücksichtigung von deutlich mehr Daten als es hier der Fall ist, die Netzperformance auch weiter steigt.

7.3 Prädiktion mittels Vektordaten der Außenwandphase

Während der Aufnahme der Datensätze wurde Wert darauf gelegt, durch hohe Anfangsgeschwindigkeiten der Kugel, in der Außenwandphase möglichst viele Daten zu generieren, um die indirekteren Bezüge zu dem Ergebnis, im Vergleich zu Daten der Spiralphase, zumindest teilweise zu kompensieren. Knapp 77 % der erzeugten Trainingsdaten in vektorieller Form sind daher der Außenwandphase zuzuordnen. Während der Entwicklung zur Prädiktion dieser Daten wurde viel experimentiert, welche Netzstruktur, Kostenfunktion und Regularisierungsmethode für diese Aufgabe am besten geeignet ist. Ausgangswerte waren stets Messergebnisse und Erkenntnisse aus den vorangegangen Testreihen zur Spiralphase. Es hat sich gezeigt, dass, ab einer Tiefe von mehr als drei Schichten, ungeachtet der Komplexität, keine besseren Messergebnisse erzeugt wurden. Obwohl breitere Netze positivere Tendenzen aufwiesen, wurde die Rate an richtigen Prädiktionen, trotz Batch-Normalisierung, meist hauptsächlich durch die Initialwerte der Gewichte bestimmt. Abbildung 7.5 zeigt das Prädiktionsverhalten des besten Netzes, welches für diese Aufgabe entwickelt wurde. Der Fokus der Kostenfunktion wurde hier relativ stark auf die Differenz der Werte und nicht auf die Distanz der Felder gelegt. Dennoch wurde der Fehler bei einer Entfernung von 18 Feldern um das vierfache höher dimensioniert, als bei dem jeweils richtigen.

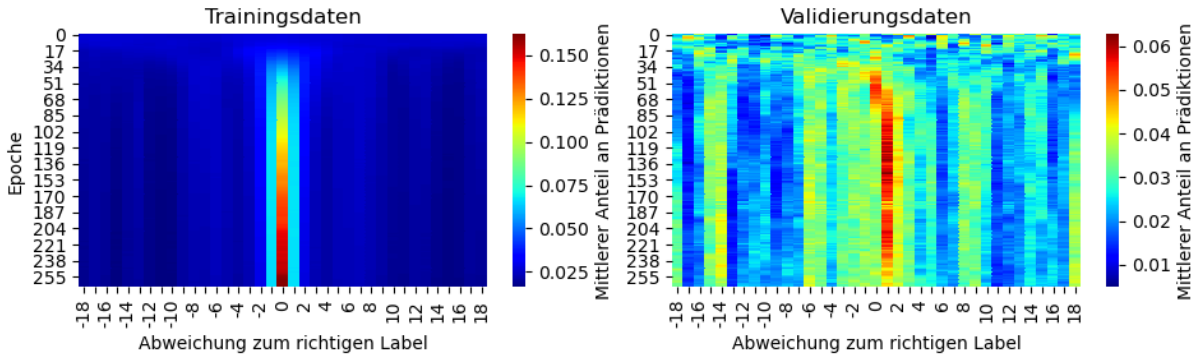


Abbildung 7.5: Bestes Prädiktionsverhalten relativ zu den richtigen Ergebnissen auf den Vektordaten der Außenwandphase über mehrere Epochen

Der Verlauf des Maximums auf den Validierungsdaten ist hier sehr typisch. Eine Überanpassung der Trainingsdaten ist in jedem der Tests bereits sehr früh eingetreten, was auch ein Indiz für eine mangelnde Datenvielfalt ist. Bei den Untersuchungen zur Außenwandphase hat dies ausschließlich darin resultiert, dass das Hauptmaximum der prädizierten Felder relativ zu dem richtigen für einen längeren Traingingszeitraum ein Feld daneben erzeugt wurde. Auch bei Tests zur Spiralphase konnte in diesen Plots sehr oft beobachtet werden, wie sich Haupt- und Nebenmaxima innerhalb der ersten 100 Epochen über die Abszisse verschoben haben, als würde sich im Netz ein Offset ausbilden. Dieser ist jedoch bei Tests zur Spiralphase sehr unterschiedlich ausgefallen und lässt sich daher nicht auf Unterschiede der Eigenschaften des Versuchsaufbaus bei der Aufnahme des Validierungsdatensatzes, wie erhöhter Reibung durch Staubbildung, zurückführen, welche etwa einen Monat später erfolgt ist. Vielmehr scheint hier die

tatsächliche Relation zwischen den Daten der Kugel und der Drehscheibe nicht hinreichend approximiert. Die starke Abhängigkeit zu den Initialwerten der Gewichte lässt sich scheinbar darauf zurückführen, inwieweit diese die Ausbildung des Hauptmaximums begünstigen, bevor ein datensatzspezifischer Offset erlernt wurde. In dem dargestellten Fall hat dies zur Folge, dass das Netz etwa bei Epoche 50 einen relativ guten Anteil richtiger Prädiktionen erzeugt, bevor der Offset das Maximum verschiebt. Teilweise konnte dieser durch gezielte Regularisierungsmethoden zumindest in geringer Form kompensiert werden. Ähnlich zu den Erkenntnissen aus [14] wurde hier jedoch keine Kombination aus der Batch-Normalisierung und einem Dropout-Anteil gefunden, welche die Messergebnisse verbessert hat. Inwieweit dies auf den Umfang der Untersuchungen zurückzuführen ist, lässt sich schwer sagen.

Anhand der Erkenntnisse aus Kapitel 7.2 wird hier deutlich, dass nicht die Menge, sondern vor allem die Diversität der Daten entscheidend ist, da Eingangsdaten aus verschiedenen Punkten derselben Trajektorie, aufgrund der identischen Bezüge während der Chaophase, sehr ähnlich erlernt wurden. Die Menge an Daten, welche aus einem Video gewonnen werden kann, ist somit deutlich weniger relevant, als die Anzahl der Videos.

Ähnlich zu Schießübungen, ist eine Clusterbildung auch ein gutes Zeichen, selbst wenn sich dieses nicht im Zentrum befindet. Stärker ausgeprägte Maxima, relativ zum richtigen Ergebnis, bedeuten, dass zumindest ein Teil global geltender Korrelationen richtig erlernt wurde. Betrachtet man den Mittelwert der Netzausgänge der Felder relativ zum richtigen, erkennt man, dass das Ausgabeverhalten grundsätzlich eine gute Approximation der Bezüge widerspiegelt. Obwohl das Maximum verschoben ist, lässt sich deutlich erkennen, dass das Netz im Mittel die höchsten Wahrscheinlichkeiten in der Nähe des richtigen Ergebnisses ausgibt und bei entfernteren Feldern meist richtig erkannt hat, dass die Kugel hier nicht auftreffen wird.

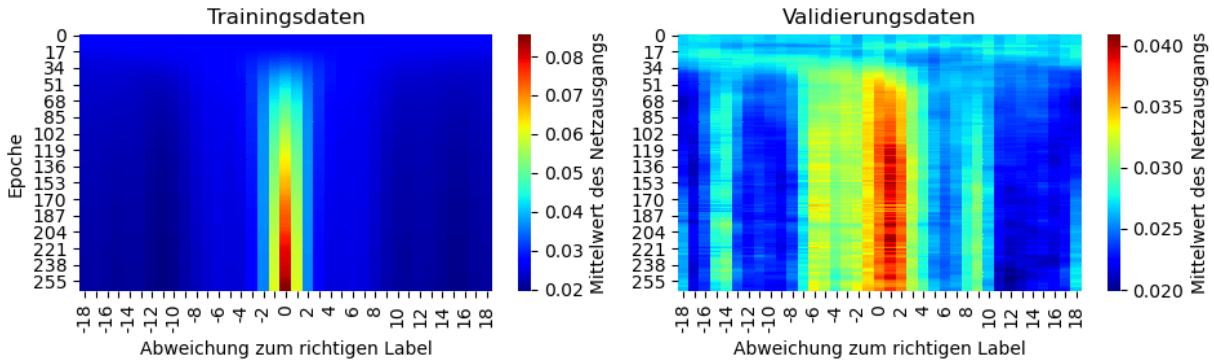


Abbildung 7.6: Mittlere Netzausgabe relativ zu den richtigen Ergebnissen auf den Vektordaten der Außenwandphase mit dem besten Netz über mehrere Epochen

Auffällig ist auch, dass, obwohl oftmals Felder weit entfernt von dem richtigen prädiziert wurden, das Netz aber an dieser Stelle in der Regel niedrige Werte erzeugt. Abbildung 7.7 zeigt die Standardabweichung der mittleren Netzausgabe aus Abbildung 7.6. Bis auf Nebenmaxima, welche auch in der Netzausgabe zu sehen sind, liefert das Netz in Bereichen weit entfernt von dem richtigen Feld relativ konstante Ausgaben. Pädiktionen in diesem Bereich sind also in dieser Darstellungsweise einzelne Fluktuationen durch

mangelnde Approximation, spiegeln aber eher weniger das tatsächliche Verhalten des Netzes wieder. Zu betonen ist jedoch, dass es sich bei der Netzausgabe um Mittelwerte handelt, deren Wertebereich von der Standardabweichung deutlich übertroffen wird, sodass hier lediglich Tendenzen betrachtet werden. Zu berücksichtigen ist aber auch, dass der Netzausgang durch die Softmax-Funktion zu Werten näher an der Null oder an der Eins tendiert. Dem größten Wert der Ausgangsneuronen vor der Aktivierungsfunktion wird ein verhältnismäßig deutlich größerer Ausgangswert zugewiesen, wodurch eine natürliche Tendenz zu einer höheren Standardabweichung entstehen kann.

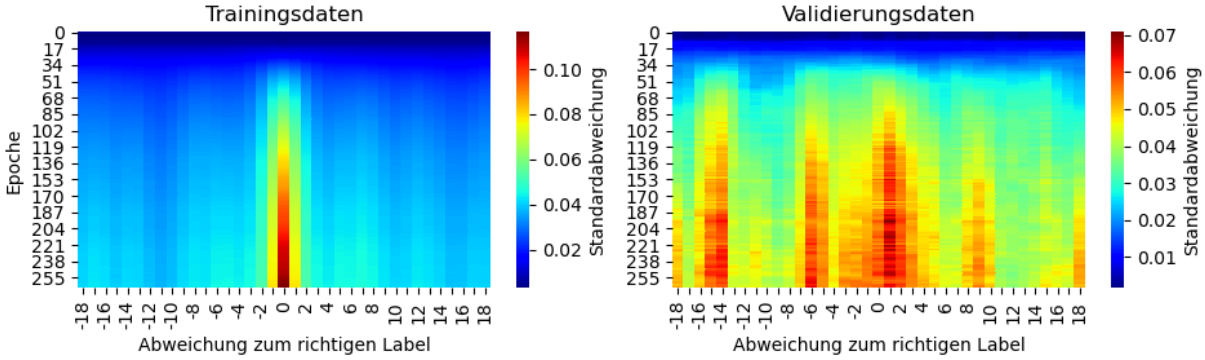


Abbildung 7.7: Standardabweichung der Netzausgabe relativ zu den richtigen Ergebnissen auf den Vektordaten der Außenwandphase mit dem besten Netz über mehrere Epochen

Die allgemeine Netzperformance auf den Validierungsdaten ist, in Anbetracht der Komplexität der Aufgabe, grundsätzlich nicht schlecht, dennoch sind die Unterschiede zu den Trainingsdaten immens. Während beim Training teilweise Wahrscheinlichkeiten einer richtigen Prädiktion von über 80 % erreicht wurden, konnten auf unbekannten Daten maximal etwa 5,5 % der Felder richtig prädiziert werden. Dieser Wert unterschreitet deutlich die Performance der Netze auf Daten der Spiralphase in Kapitel 7.4. Obwohl die Trajektorien der Kugel und der Drehscheibe vom Netz abstrahiert werden, bietet es sich an, im Bezug auf das Prädiktionspotential, Eigenschaften einzelner Phasen separat zu betrachten. Im Vergleich zur Prädiktion auf Basis von Daten der Spiralphase sind hier, durch die indirekteren Bezüge zwischen den Eingangsdaten und dem Eintrittsvektor in die Chaosphase, deutlich höhere Anforderungen an die Prädiktionsgenauigkeit gestellt. Sind diese nicht hinreichend erfüllt, werden Abweichungen durch eine schlechte Approximation, anschließend durch die Chaotik immens verstärkt. Eine weitere Erklärung für den Performanceunterschied sind schlecht gewählte Parameter auf Basis der Spiralphase. Beispielsweise werden hier deutlich größere Kugelgeschwindigkeiten untersucht. Entsprechend den Erkenntnissen aus Kapitel 6.1.5 wurden für die Vektordaten Informationen aus 50 aufeinanderfolgenden Frames verwendet. Möglicherweise liegt das dort ermittelte Optimum, durch die, mit der Geschwindigkeit korrelierenden, Messungenauigkeiten in der Außenwandphase, bei einer größeren Vektorlänge.

Obwohl diese Ergebnisse im Vergleich zu denen der Spiralphase etwas enttäuschend sind, wurde die erreichte Performance der Untersuchungen mit Modellbildung aus [18] und [19] deutlich übertroffen.
Abbildung 7.8 zeigt den Verlauf des Erwartungswertes, abhängig von der Anzahl an Feldern, welche rechts und links neben dem Prädizierten in die Wette mit einbezogen wurden. Dies entspricht dem Ansatz der Gewinnmaximierung aus [18] und [19]. Alternativ ließe sich auf eine bestimmte Anzahl der Netzausgaben mit der höchsten Wahrscheinlichkeit oder auf alle Felder, deren Netzausgabe eine gewissen Wert überschreitet, wetten. Letzteres wird durch die Softmax-Funktion beeinflusst, deren Notwendigkeit bei einer Anwendung proprietärer Kostenfunktionen jedoch nur durch die Eigenschaft begründet wird, dass der Netzausgang als Wahrscheinlichkeitsverteilung betrachtet werden kann. Der erreichbare Profit ließe sich zudem vermutlich dadurch optimieren, dass der Einsatz entsprechend der ausgegebenen Wahrscheinlichkeit skaliert wird und, anstelle von Gruppen von Feldern, entsprechend dem Streuverhalten der Kugel, auf bestimmte Muster gewettet wird.

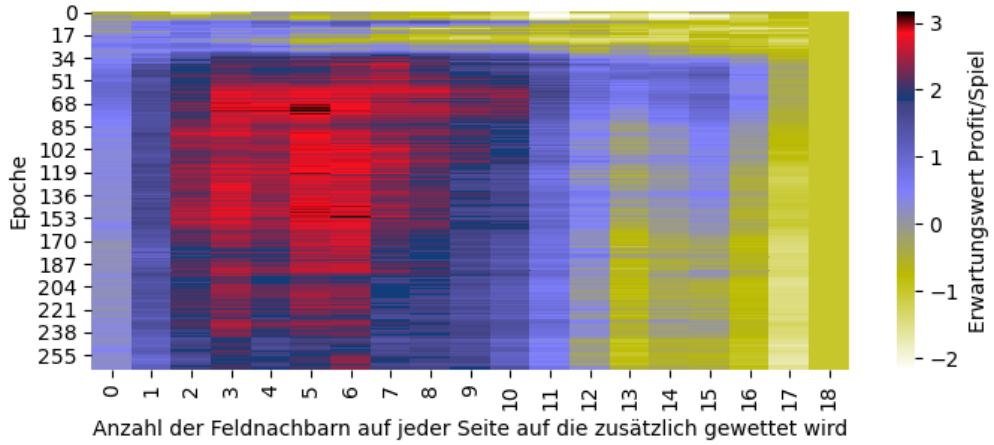


Abbildung 7.8: Verlauf Erwartungswert Profit des besten Netzes auf Validierungs-Vektordaten der Außenwandphase beim Wetten auf die prädizierte Zahl und dessen Feldnachbarn mit gleichem Einsatz über mehrere Epochen

Das Wetten auf ein Feld und dessen 18 Nachbarn auf beiden Seiten, entspricht einer Wette auf jedes Feld. Man würde somit unabhängig vom Ergebnis das 35-fache erhalten, aber durch das Wetten auf alle falschen Felder das 36-fache verlieren. Der Wert der Spalte rechts außen ändert sich daher nicht. Obwohl sich beim Prädiktionsverhalten in Abbildung 7.5 zeitweise ein deutliches Maximum bei dem richtigen Ergebnis ausgebildet hat, ist eine Wette auf Feldgruppen deutlich profitabler. Abhängig davon, wie genau der Validierungsdatensatz unbekannte Daten beschreibt, bzw. bis zu welchem Maße zufällige, nicht global geltende, Korrelationen mit dem Trainingsdatensatz übereinstimmen und weitere Bezüge in keinem der Datensätze enthalten sind, ist es mit diesem Netz möglich, mit jedem Spiel im Mittel etwa das dreifache des Einsatzes eines Feldes an Profit zu erzeugen. Dies übersteigt den besten mittels Modellbildung erzielten Erwartungswert von 0,41 bei Weitem [19]. Zu betonen ist, dass die Untersuchungen mittels Modellbildung profitorientiert durchgeführt wurden, während der Fokus dieser Arbeit vor allem auf der Maximierung des Anteils exakter Prädiktionen lag. Würden diese Netze für die

Anwendung in einem Casino entwickelt werden, wären Netze mit entsprechenden Kostenfunktionen besser geeignet, welche nicht zwingend exakt, aber dafür sehr oft nahe dem richtigen Ergebniss prädizieren. An den gelblichen Farbbereichen lässt sich erkennen, dass die Schwelle, ab welcher mit diesem Netz Profit gemacht werden kann, bereits nach wenigen Epochen überschritten wurde. Mit hinreichenden Trainingsdaten sind neuronale Netze einer manuellen Modellbildung somit — zumindest in der Approximation chaotischer Systeme — vermutlich deutlich überlegen.

7.4 Prädiktion mittels Vektordaten der Spiralphase

Die Untersuchungen von Vektordaten der Spiralphase stellen in dieser Arbeit die beste erreichte Performance zur Prädiktion eines chaotischen Systems dar. Dies ist zum Teil darauf zurückzuführen, dass jede Art von Netzoptimierung und Parametrierung aus Kapitel 6 durch umfangreiche Tests auf diese Daten ausgelegt wurde. Zur Optimierung der Netzstruktur wurden Netze mit einer bis sechs Schichten einer Breite von bis zu 2000 Neuronen pro Schicht untersucht. Auch bei diesen Tests wurden ab einer Tiefe von drei Schichten kaum Verbesserungen im Lernverhalten festgestellt. Scheinbar ist die Diversität der Bezüge in den Daten nicht hinreichend, sodass eine Approximation der tatsächlichen Komplexität nicht notwendig ist. Dennoch konnte die beste Performance bei einem relativ breiten Netz mit sechs Schichten erreicht werden. Bei Netzen dieser Größenordnung war die Überanpassung auf die Trainingsdaten ein noch größeres Problem. Ohne Verwendung einer proprietären Kostenfunktion, deren Parametrierung sich als stärker regularisierend herrausgestellt hat, musste dies sehr gezielt durch herkömmliche Regularisierungsmethoden verhindert werden. Abbildung 7.9 zeigt das jeweils beste entwickelte Ausgabeverhalten, unter Anwendung der Kreuzentropie und einem proprietärem Ansatz. Die Trainingsdaten werden nicht mit dargestellt, da ihr Lernverhalten hier keinen Mehrwert bietet.

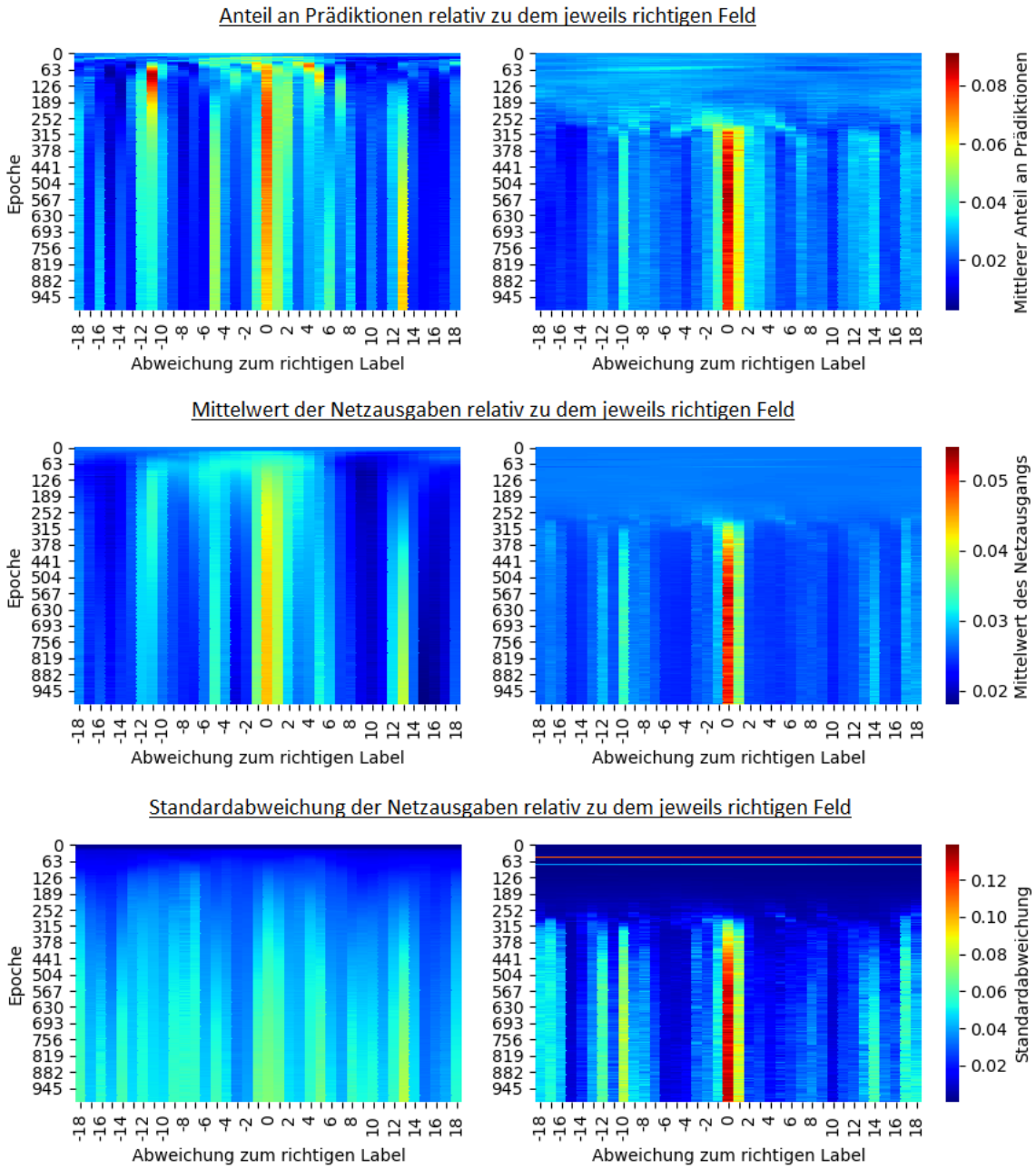


Abbildung 7.9: Gegenüberstellung Netzperformance unter Anwendung der Kreuzentropie (links) und proprietärem Ansatz (rechts) auf Validierungs-Vektordaten der Spiralphase über mehrere Epochen

Obwohl beide Performance-Optima mit derselben Netzstruktur erreicht wurden, unterscheidet sich das Ausgabeverhalten stark. Trotz Anwendung verschiedener Regularisierungsmethoden ist dieses Verhalten auf die Kostenfunktion zurückzuführen. Die Tendenz, entferntere Nebenmaxima nicht zu unterdrücken, ist für das Prädiktionsverhalten der Kreuzentropie hier typisch. Auch hier lässt sich leicht der, in Kapitel 7.3 erwähnte, Verschub der Maxima über die Abszisse erkennen. Insbesondere die Ausprägung von Nebenmaxima, bei einer Entfernung von fünf oder sechs, bzw. von zehn bis vierzehn Feldern von dem richtigen Ergebnis, ist in den Testreihen sehr oft aufgetreten. Die Symmetrie lässt vermuten, dass es sich hierbei nicht um das Resultat einer falschen Approximation durch Overfitting handelt, da datensatzspezifische Bezüge

auf Validierungsdaten entweder zufällige Prädiktionen oder einseitige Maxima, wie in Kapitel 7.3, begünstigen müssten. Symmetrische Maxima ließen sich allgemein an dem Streuverhalten der Kugel bei Abweichungen der Trajektorie begründen. Beispielsweise ist bei der Aufnahme der Datensätze öfter der Fall eingetreten, dass die Kugel begonnen hat, kurz vor dem Stillstand durch die glatte Oberflläche auf dem Ziffernkreis zu rollen, und somit deutlich mehr Strecke zurück gelegt hat, als wenn sie einen Millimeter weiter auf die Kante eines Feldes getroffen wäre. Solche Effekte begünstigen verschwindend kleine Entscheidungsgrenzen bei der Auswertung der Eingangsdaten und erhöhen durch das Streuverhalten die auszugebene Wahrscheinlichkeit relativ zum richtigen Feld, je nachdem, ob die Kugel weiter gerollt ist als prädiziert oder umgekehrt. Die Tatsache, dass die Werte bei -11, Null und 13 in der Summe fast konstant sind, untermauert diese Hypothese.

Beim Prädiktionsverhalten des proprietären Ansatzes ist erst etwa ab Epoche 280 ein Fortschritt zu sehen. Die Informationen der Logdatei und Untersuchungen der Metriken haben gezeigt, dass der Standard-Initialwert der Lernrate von 0,001 hier deutlich zu hoch war. Diese wurde automatisch von dem Callback *DynamicLearningRateScheduler* aus Kapitel 6.2.5 angepasst. In Anbetracht der Dauer, bis ein Lernfortschritt zu erkennen war, und einer Notwendigkeit von vier Aktualisierungen der Lernrate, wurden dessen Übergabeparameter womöglich nicht optimal gewählt.

Die in der Kostenfunktion verwendete Parameterkombination ist ein Kompromiss aus der Gewichtung der Abweichung der Werte und der Distanz der Felder. Wie beim Optimum des Prädiktionsverhaltens zur Außenwandphase, wurde auch hier die Distanz durch den Parameter $\alpha = 1$ linear skaliert und die Abweichung zum erzielten Wert durch den Parameter $\beta = 2$ quadriert. Anstatt höhere Distanzen deutlich stärker zu gewichten, war es hier vorteilhaft, den Fokus allgemein auf die Korrektur von Abweichungen falscher Felder zu legen.

Betrachtet man das Ausgabeverhalten beider Netze, wird ähnlich zu Kapitel 7.3 auch hier deutlich, dass Prädiktionen bei manchen Distanzen nicht der Norm der dortigen Netzausgabe entsprechen. Besonders der proprietäre Ansatz scheint die tatsächlichen Bezüge zu dem Ergebnis im Mittel sehr gut approximiert zu haben. Die hohe Standardabweichung ist jedoch grundsätzlich kein gutes Zeichen. Auch hier ist diese darauf zurückzuführen, dass sich das Netz bei manchen Datenpunkten sehr sicher ist und somit Werte nahe Eins ausgibt, welche stark von der Norm abweichen. Auffällig sind auch die zwei Linien etwa um Epoche 63. An diesen Stellen ist das Netz durch die hohe Lernrate aus einem lokalen Minimum ausgebrochen und hat dadurch deutlich andere Werte erzeugt. Durch den Callback *DynamicLearningRateScheduler* wurden an diesen Stellen vorherige Gewichte geladen, sodass dieser Effekt nur in jeweils einer Epoche zu sehen ist.

7.5 Prädiktion mittels Differenzbildern der Spiralphase

Aufgrund der immensen Trainingsdauer von Differenzbildern, war eine tiefgehende Untersuchung zur Hyperparameter-Optimierung mit den gegebenen Hardware-Ressourcen leider nicht möglich. Viele Einstellungen wurden nach bestem Ermessen aus Testergebnissen mit Vektordaten übernommen. Da das Ermitteln des Dropoutanteils ein iterativer Prozess ist, welcher bei falscher Parametrierung zu deutlich schlechteren Ergebnissen führen kann, wurde dieses Verfahren bei der Untersuchung von Differenzbildern nicht berücksichtigt.

Die Aufgabe der Faltungs- und Poolingschichten bestand darin, wichtige Konturen zu selektieren und so stark zu komprimieren, dass die Gewichte-Tensoren hinreichend klein sind, ohne die, durch die Skalierung ohnehin verrauschten, Positions-Informationen zu stark zu verwischen. Mit dem Gedanken an typische Hochpass-Filter, wie dem Prewitt- oder Sobel-Operator, und dem verfügbaren Grafik-Speicher schien hier eine Verwendung von Filtermasken der Dimension 3x3, ohne Erweiterung durch künstliche Null-Zeilen und Spalten, hinreichend. Ein optimaler Kompromiss zwischen Genauigkeit, Datenaufbereitung und Speicherverbrauch wurde auf zwei Faltungs- und Pooling-Schichten, mit einer Faltung bei jedem Bildpunkt und einer max()-Funktion beim Pooling, geschätzt. Ein Auffüllen der Randbereiche durch Padding entspricht hier einer Generierung unnötiger Information und wurde daher deaktiviert.

Obwohl die Differenzbilder bereits stark skaliert wurden, stieß der Server leider an seine Grenzen. Die Notwendigkeit einer sehr kleinen Batchgröße (siehe Abschnitt 6.4.1) spiegelt sich auch in den Messergebnissen wider. Durch die hier verwendete Batchgröße von gerademal 300 bei 2.632.300 Bildern, werden die Gewichte pro Epoche 8745 mal aktualisiert. Bei sonst gleichen Einstellungen führt dies dazu, dass das Netz bereits nach einer Epoche stark zum Overfitting neigt. Um diesen Effekt zu kompensieren wäre eine Verringerung des Initialwertes der Lernrate entsprechend dem Verhältnis der Batchgröße zu der beim Training mit Vektordaten vermutlich ein guter Richtwert, sodass batchspezifische Einflüsse durch die geringe Schrittweite möglichst minimiert werden. Dies hätte das Training jedoch so stark verlangsamt, dass kein vollständiger Test in gegebener Zeit möglich wäre. Stattdessen wurde der Callback *SaveMetrics* umgeschrieben, sodass der Trainingsfortschritt in diesem bereits alle 50 Batches validiert wird und das Potential von Differenzbildern, unter Berücksichtigung dieser Faktoren, analysiert werden kann. Abbildung 7.10 zeigt das resultierende Prädiktionsverhalten der Trainings- und Validierungsdaten. Entgegen der üblichen Darstellung beschreibt die Ordinate nicht den Lernfortschritt in Epochen, sondern das Vielfache von 50 Batches. Insgesamt wurden also etwa 50.000 Batches trainiert. Während des Trainings einer Epoche wurden somit 55 Metrik-Vektoren gespeichert.

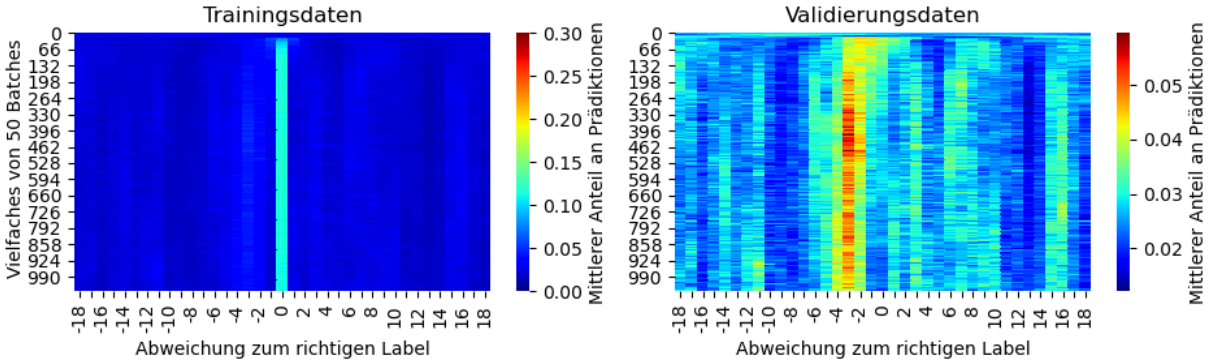


Abbildung 7.10: Bestes Prädiktionsverhalten relativ zu den richtigen Ergebnissen mit Differenzbildern der Spiralphase auf Trainingsdaten des aktuellen Batches (links) und den gesamten Validierungsdaten (rechts) über mehrere Epochen

Beurteilt man die Netzperformance an dem Anteil richtiger Prädiktionen, wurde deren Optimum bereits nach einer halben Epoche erreicht. Auch unter Berücksichtigung der Redundanz der Daten, durch identische Trajektorien während der Chaosphase, ist dies beachtlich und lässt sich neben der Batchgröße möglicherweise auf die Batch-Normalisierung zurückführen. Der starke Verschub des Maximums ist vermutlich durch das batchspezifische Training entstanden, wobei ausgeprägtere batchspezifische Minima eine bessere Appoximation der Relation zwischen Drehscheibe und Kugel verhindert haben könnten. Auffällig ist auch, dass auf den Trainingsdaten bereits sehr früh ein Grenzwert erreicht wurde, welcher mit dem Lernfortschritt der Validierungsdaten aber nur bedingt korreliert. Der Grenzwert lässt sich durch die Größe der Lernrate relativ zur Genauigkeit des Gradienten erklären. Der Callback, welcher für die Aktualisierung der Lernrate verantwortlich ist, benötigt die Fehler von zehn Epochen als Referenzwerte und hat während dieses Trainings somit viel zu spät reagiert. Schmalere Minima wurden nicht erlernt, da der Gradient durch ein wohlmöglich stark alternierendes Verhalten in vielen Dimensionen und zu großen Schrittweiten deutlich zu grobe Aktualisierungen im nächsten Iterationsschritt verursacht hat. Teilweise wurden jedoch etwa 5,5 % unbekannter Daten konsequent drei Felder gegen den Uhrzeigersinn pädiziert, was, ähnlich Kapitel 7.3, durchaus ein gutes Zeichen ist und das Potential dieser Eingangsdaten verdeutlicht. Trotz des starken Rauschanteils von Positions-Informationen, verstärkt durch die Faltungs- und Poolingschichten, wird das Verständnis der Bezüge vermutlich wenig durch die Form des Netzeingangs beeinträchtigt. Zu berücksichtigen ist auch, dass eine allgemein schlechtere Performance zumindest zum Teil durch fehlendes Hyperparametertuning begründet werden kann.

Da eine Vektorbestimmung und die damit verbundene Berechnungszeit und Datenaufbereitung mit Differenzbildern, auch bei Anwendung der Netze, in einem solchem Maß nicht notwendig ist, ist eine Implementierung nach einem Training mit entsprechenden Rechenressourcen einer Prädiktion mit Vektordaten möglicherweise vorzuziehen, bis eine Performance erreicht wird, bei der die örtliche Genauigkeit, auch unter Berücksichtigung einer solchen Menge an Aufnahmen am Netzeingang, eine größere Rolle spielt.

8 Fazit und Ausblick

Es konnte gezeigt werden, dass eine systemspezifische Fehlerskalierung konventionellen Kostenfunktionen deutlich überlegen sein kann. Ungeachtet der übrigen Hyperparameter, wurde die beste Netzperformance ausnahmslos durch eine proprietäre Kostenfunktion definiert. Entsprechend aktueller Forschungen konnte auch hier gezeigt werden, dass neuronale Netze durchaus in der Lage sind, anhand von visuellen Daten physikalische Zusammenhänge zu erlernen. In Anbetracht der hohen und frühen Divergenz zwischen dem Anteil richtiger Prädiktionen auf Trainings- und Validierungsdaten und der Erkenntnisse aus Kapitel 7.2 und 7.3, ist die geringe Vielfalt der Datensätze aktuell jedoch vermutlich der entscheidende Faktor. Die Informationslücken sind so groß, dass die Wahrscheinlichkeit, diese richtig zu generalisieren, deutlich zu gering ist. Die Bezüge zum Ergebnis wurden somit auf eine so simple Weise abstrahiert, dass das Potential proprietärer Kostenfunktionen möglicherweise noch nicht ausgeschöpft wurde. Zudem sind weitere Einflüsse, wie veränderliche Testbedingungen oder Fehler bei der manuellen Bestimmung der Klassenzugehörigkeit während der Aufnahme, mit einzubeziehen. Die Relevanz nicht berücksichtigter Messwerte wie der Temperatur, Luftfeuchtigkeit oder Vibrationen kann nicht ausgeschlossen werden, ist aber höchstwahrscheinlich so gering, dass auch bei einer nahezu perfekten Netzperformance dennoch andere Faktoren den verbleibenden Fehler dominieren würden.

Auch der negative Einfluss der Daten war im Hinblick auf deren Qualität vermutlich allgemein sehr klein. Abgesehen von der Verteilung der Eintrittspunkte in die Spiralphase aus Kapitel 7.1 und den daraus resultierenden Folgen für die Prädiktion bei anderen Croupiers, wurden bezüglich der Genauigkeit der Voruntersuchungen, insbesondere zur Hardware, in Anbetracht ihrer Notwendigkeit bei erreichter Performance, deutlich zu hohe Anforderungen gestellt. Dennoch war die Abhängigkeit der Performance von der Anzahl verwendeter Aufnahmen am Netzeingang um ein Vielfaches größer als erwartet. Da eine starke Verringerung der Auflösung des Wertebereichs der Daten die mögliche Netzperformance jedoch tendenziell sogar positiv beeinflusste, lässt sich die notwendige Redundanz der Eingangsdaten nicht allein auf Rauscheinflüsse durch die Hardware oder den Algorithmus zur Vektorbestimmung zurückführen. Dies lässt vermuten, dass die in Kapitel 5.2 verdeutlichten Abweichungen tatsächlich durch Einflüsse, wie dem Drall der Kugel, verursacht wurden, welche durch einzelne Positions- und Geschwindigkeitsvektoren beweglicher Objekte nicht definiert werden.

Verwandte Forschungen über die Prädiktion von Roulette mittels Modellbildung wurden um ein Vielfaches übertroffen. Obwohl das Potential neuronaler Netze in dieser Arbeit vermutlich bei Weitem nicht ausgeschöpft wurde, ließen sich deren Vorteile gegenüber einer Modellbildung in der Approximation chaotischer Systeme bestätigen. Welche Form des Netzeinganges überlegen ist, hängt vom Anwendungsfall ab. Eine Prädiktion mittels Differenzbildern ist insofern interessant, als dass, bei Anwendung in einem Casino, das Wetten nach Einwerfen der Kugel relativ schnell verboten wird. Eine Bestimmung der Vektordaten könnte somit bereits zu viel Rechenzeit in Anspruch nehmen. Bezüglich der Forschungen zur Prädiktion chaotischer Zeitreihen, wurden hier teilweise deutlich größere Abweichungen erzeugt. Dies hängt jedoch damit zusammen, dass die Genauigkeit der prädizierten Zustände, aufgrund der Chaotik, mit zunehmender Zeitspanne immens abfällt. Solche Ergebnisse lassen sich daher sehr schlecht vergleichen.

Weiterführend zu diesen Untersuchungen wäre, neben einer starken Erweiterung der Trainings-Datensätze, ein tiefgehenderes Hyperparametertuning, besonders mit faltenden neuronalen Netzen sowie bei der Prädiktion von Vektordaten der Außenwandphase, ein guter Ansatz, sodass die tatsächliche maximal mögliche Performance mit solchen Daten in Relation zueinander und zur Prädiktion mittels Vektordaten der Spiralphase besser verglichen werden kann.

Sollte der Fokus auf eine Profitoptimierung gelegt werden, ließe sich auch diese, unter Berücksichtigung von Wetten auf Feldgruppen verschiedener Größe, in der Kostenfunktion unterbringen. Zudem wurden die Ähnlichkeiten einzelner Klassen durch eine relativ simple Funktion approximiert. Vorteilhaft ist, dass der zu optimierende Parameterraum hierfür somit relativ wenig Dimensionen aufweist. Dennoch ließe sich die manuelle Skalierung beispielsweise durch Lookup-Tabellen realisieren, in welchen die Ähnlichkeiten der Klassen vom Entwickler definiert und optimiert werden können. Darüber hinaus wurde der Effekt eines besseren Generalisierungsverhaltens mancher proprietärer Ansätze nur oberflächlich untersucht. Der Hintergrund eines solchen Verhaltens ließe sich beispielsweise anhand der Entwicklung der Gewichte und des Informationsflusses durch das Netz bei verschiedenen Netzeingängen genauer erörtern.

Im Bezug auf die Untersuchung möglicher Netzstrukturen und Netztypen ließe sich der Ansatz verfolgen, die Prädiktion, ähnlich zu den Forschungen zur Approximation chaotischer Zeitreihen, als Regressionsproblem zu lösen. Zudem wurde vermutet, dass verschobene Hauptmaxima im Prädiktionsverhalten der Netze durch eine schlechte Approximation der Relation zwischen den Positionen der Kugel und der Drehscheibe verursacht werden. Dieser Effekt würde vermutlich durch mehr Trainingsdaten kompensiert werden, ließe sich aber möglicherweise besser unterdrücken, indem die Informationenen der Drehscheibe erst in späteren Schichten in das Netz einfließen. Auf diese Weise würde das Netz gezwungen werden, die deutlich komplexeren Bezüge der Kugel zum Ergebnis auch auf eine entsprechend komplexere Weise zu erlernen und die Relation zur Drehscheibe, ähnlich zu dem realen Verlauf, erst später mit einzubeziehen.

Entsprechend der Argumentation aus Kapitel 4.3, könnten FFNN zur Auswertung von Bildern, durch die Erhaltung örtlicher Informationen, in diesem Anwendungsfall mit entsprechenden Rechenressourcen und einem hinreichend großen Trainingsdatensatz deutlich von Vorteil sein. Alternativ ließen sich auch Algorithmen zur Objekterkennung, wie dem YOLO- oder SSD-Algorithmus, implementieren. Das Problem geringer Batchgrößen ließe sich – etwas zeitaufwendig – auch lösen, indem batchspezifische Gradienten durch benutzerdefinierte Trainingsschleifen vor der Aktualisierung der Gewichte auf die Festplatte ausgelagert und anschließend zusammengefasst werden.

Literaturverzeichnis

[1] E. N. Lorenz, „Deterministic Nonperiodic Flow," 1962.

[2] J. C. Uwe Lämmel, Künstliche Intelligenz Wissensverarbeitung neuronaler Netze, 5. überarbeitete Auflage Hrsg., HANSER.

[3] K. L. Andriy Burkov, Machine learning kompakt: alles, was Sie wissen müssen, 1. Auflage Hrsg., Frechen, 2019.

[4] M. H. Maximilian Ochs, *Computational-Intelligence Convolutional Neural Networks (CNN),* 2016.

[5] A. Trask, Grokking deep learning, 1. Auflage Hrsg., Frechen, 2020.

[6] C. C. Aggarwal, Neural Networks and Deep Learning, Springer.

[7] D. Pedamonti, „Comparison of non-linear activation functions for deep neural networks on MNIST classification task," 2018.

[8] C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), New York: Springer, 2006..

[9] C. B. C. B. F. K. C. M. M. S. Rudolf Kruse, Computational Intelligence eine methodische Einführung in künstliche neuronale Netze, evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze, 2. Auflage Hrsg.

[10] J. Frochte, Maschinelles Lernen Grundlagen und Algorithmen in Python, 2. aktualisierte Auflage Hrsg., HANSER.

[11] J. L. M. David E. Rumelhart, Parallel Distributed Processing, 1986.

[12] R. P. C. G. K. C. S. G. Y. B. Yann N. Dauphin, „Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," 2014.

[13] S. Ruder, „An overview of gradient descent optimization algorithms," 2017.

[14] C. S. Sergey Ioffe, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015.

[15] Y. B. Caglar Gülcehre, „Knowledge Matters: Importance of Prior Information for Optimization," 2013.

[16] T. v. Laarhoven, „L2 Regularization versus Batch and Weight Normalization," 2017.

[17] C. Koken, Roulette: Computersimulation & Wahrscheinlichkeitsanalyse von Spiel und Strategien, 1984.

[18] C. K. T. Michael Small, „Predicting the outcome of roulette," 2012.

[19] C. K. T. Michael Small, „Feasible implementation of a prediction algorithm for the game of roulette," 2009.

[20] S. Hamburg, Spielbank Hamburg Jahr + Achterfeld GmbH & Co. KG, [Online]. Available: https://www.spielbank-hamburg.de/spiele/. [Zugriff am 29 03 2021].

[21] P. Basieux, Roulette - Glück und Geschick, Springer Spektrum, 2013.

[22] J.-I. Eichberger, „Roulette Physics," 2003.

[23] S. G. R. F. Adam Lerer, „Learning Physical Intuition of Block Towers by Example," 2016.

[24] C. D. A. G. M. H. Jacob Walker, „An Uncertain Future: Forecasting from Static Images Using Variational Autoencoders".

[25] R. M. K. K. Vincent Michalski, „Modeling Deep Temporal Dependencies with Recurrent "Grammar Cells"".

[26] P. A. S. L. J. M. Katerina Fragkiadaki, „Learning visual predictive models of physics for playing billiards," 2016.

[27] R. R. Rajiv C. Shah, „Applying Deep Learning to Basketball Trajectories," 2016.

[28] M. P. K. Mironov, „Applying neural networks for prediction of flying objects trajectory," 2013.

[29] Y.-C. H. Hsien-I Lin, „Ball Trajectory Tracking and Prediction for a Ping-Pong Robot," 2019.

[30] T. L. F. W. Alexander Warnecke, „Convolutional Neural Networks for Movement Prediction in Videos".

[31] B. Bačić, „Predicting golf ball trajectories from swing plane: An artificial neural networks approach," 2016.

[32] Y. M. H. S. Shuya Suda, „Prediction of Volleyball Trajectory Using Skeletal Motions of Setter Player".

[33] H. Z. D. T. M. Z. Y. L. Nitin Kamra, „Multi-agent Trajectory Prediction with Fuzzy Query Attention," 2020.

[34] J. B. P. Y. H. C. Sung Jin Yoo, „Stable Predictive Control of Chaotic Systems Using Self-Recurrent Wavelet Neural Network," International Journal of Control, Automation, and Systems, 2005.

[35] X.-C. X. Jia-Shu Zhang, „Predicting Chaotic Time Series Using Recurrent Neural Network," 2000.

[36] J. X. S. X. F.-L. Y. Min Han, „Prediction of Chaotic Time Series Based on the Recurrent Predictor Neural Network," 2004.

[37] T. L. S. W. Henry Leung, „Prediction of Noisy Chaotic Time Series Using an Optimal Radial Basis Function Neural Network," 2001.

[38] E. 1. s. Committee, „EMVA Standard 1288, Standard for Characterization of Image, Sensors and Cameras," 2016.

[39] F. P. L. C. F. Jürgen Beyerer, „Automatische Sichtprüfung : Grundlagen, Methoden und Praxis der Bildgewinnung und Bildauswertung," Springer, Berlin Heidelberg, 2012.

[40] „Keras," [Online]. Available: https://keras.io. [Zugriff am 01 02 2021].

[41] M. F. P. H. G. S. Alfred Nischwitz, Bildverarbeitung Band II des Standardwerks Computergrafik und Bildverarbeitung, 4. Auflage Hrsg., Wiesbaden: Springer, 2020.

[42] Schwär und Toth, „White paper - Kameraauswahl," Basler AG, Ahrensburg, 2014.

[43] A. R. J.-M. K. Jose C. Prinzipe, „Prediction of chaotic time series with neural networks and the issue of dynamic modeling," *International Journal of Bifurcation and Chaos,* 1992.

[44] H. H. Herbert Jaeger, „Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication".

[45] B. R. T. M. L. M. L.P. Maguire, „Predicting a chaotic time series using a fuzzy neural network," 1998.

[46] S.-Y. L. Dulakshi S.K. Karunasinghe, „Chaotic time series prediction with a global model: Artificial neural network," Journal of Hydrology 323, 2006.

[47] A. V. E. C. d. S. Kenya Andreesia de Oliveira, „Using artificial neural networks to forecast chaotic time series," Physica A, 2000.

[48] S. Z. Muhammad Ardalani-Farsa, „Chaotic timeseries prediction with residual analysis method using hybrid Elman–NARX neural networks," elsevier, 2010.

[49] R. R. Rajiv C. Shah, „Applying Deep Learning to Basketball Trajectories," 2016.

[50] P. K. A. J. B. Jonathan W. Woolley, „Modeling and prediction of chaotic systems with artificial neural networks," 2009.

[51] A. Jansson, „Predicting trajectories of golf balls using recurrent neural networks," 2017.

[52] F. Rosell, „Tracking a ball during bounce and roll using recurrent neural networks," 2018.

[53] L. v. Graph, Roulette. Regeln, Chancen, Strategien, Hugendubel Heinrich GmbH, 1990.

[54] M. F. P. H. G. S. Alfred Nischwitz, Bildverarbeitung Band II des Standardwerks Computergrafik und Bildverarbeitung, 4. Auflage Hrsg., Bd. 2, Springer Vieweg.

[55] J. K. J. S. Christina Stoica-Klüver, Modellierung komplexer Prozesse durch naturanaloge Verfahren, VIEWEG + TEUBNER.

[56] A. D. Hans-Joachim Bentz, Neuromathematik und Assoziativmaschinen, Springer Vieweg.

[57] A. Geron, Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow: Konzepte, Tools und Techniken für intelligente Systeme, 1. Auflage Hrsg.

[58] S. F. Crone, Neuronale Netze zur Prognose und Disposition im Handel, Wiesbaden: Gabler, 2010.

[59] P. B. J. B. T. Jessica Hamrick, „Internal physics models guide probabilistic judgments about object dynamics," Boston, Massachusetts, USA, 2011.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

_____ _____

Ort, Datum Unterschrift

Anhang A Ergänzende Hardwaretests

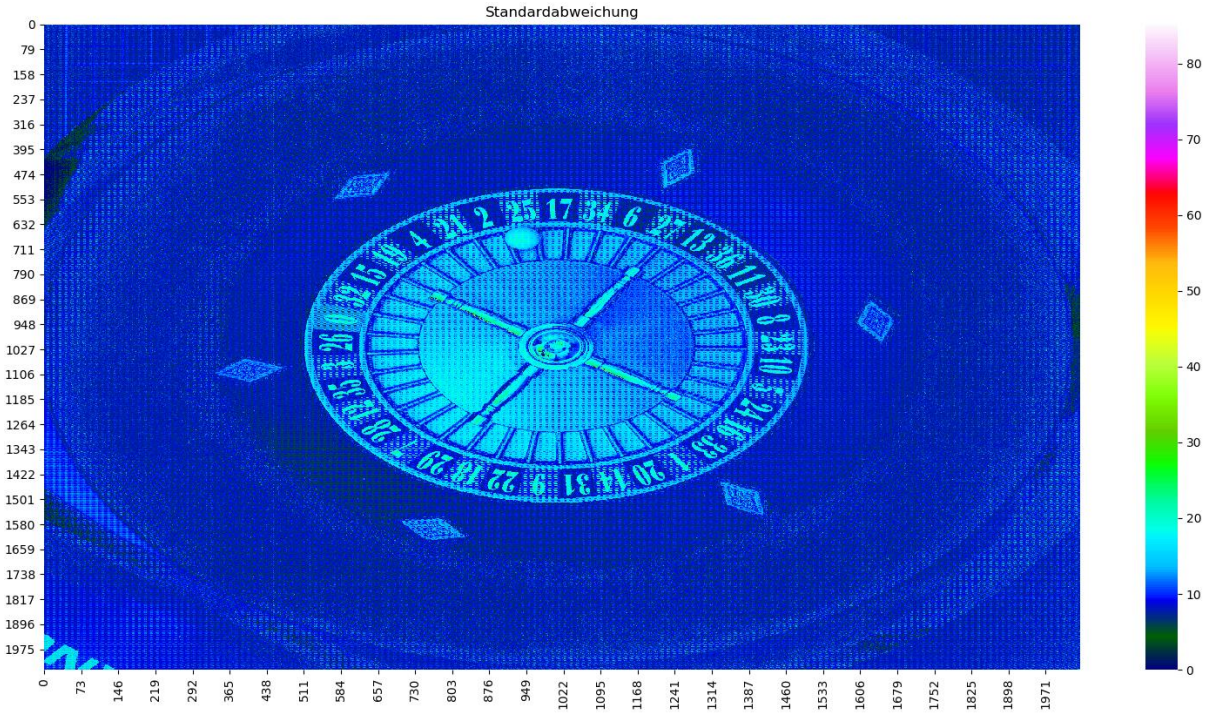


Abbildung A.1: Standardabweichung der Rohdaten eines Videos in Helligkeitswerten bei einer Auflösung von 8-Bit dominiert durch stroboskopische Effekte - das Raster ist ein Resulat von Alias-Effekten durch den Bayerfilter

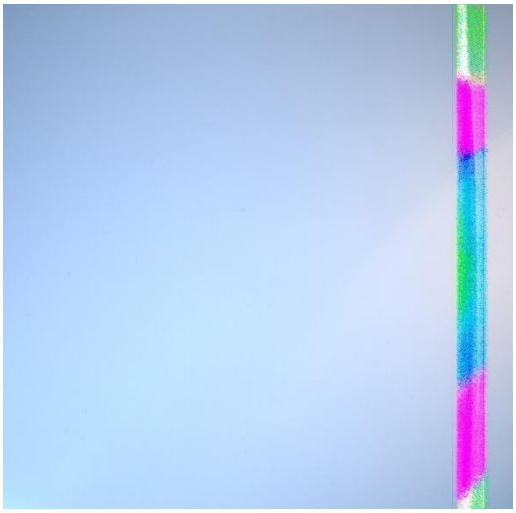


Abbildung A.2: Verdeutlichung des thermisch bedingten Kamerafehlers nach längerer Laufzeit auf einem weißen Blatt Papier

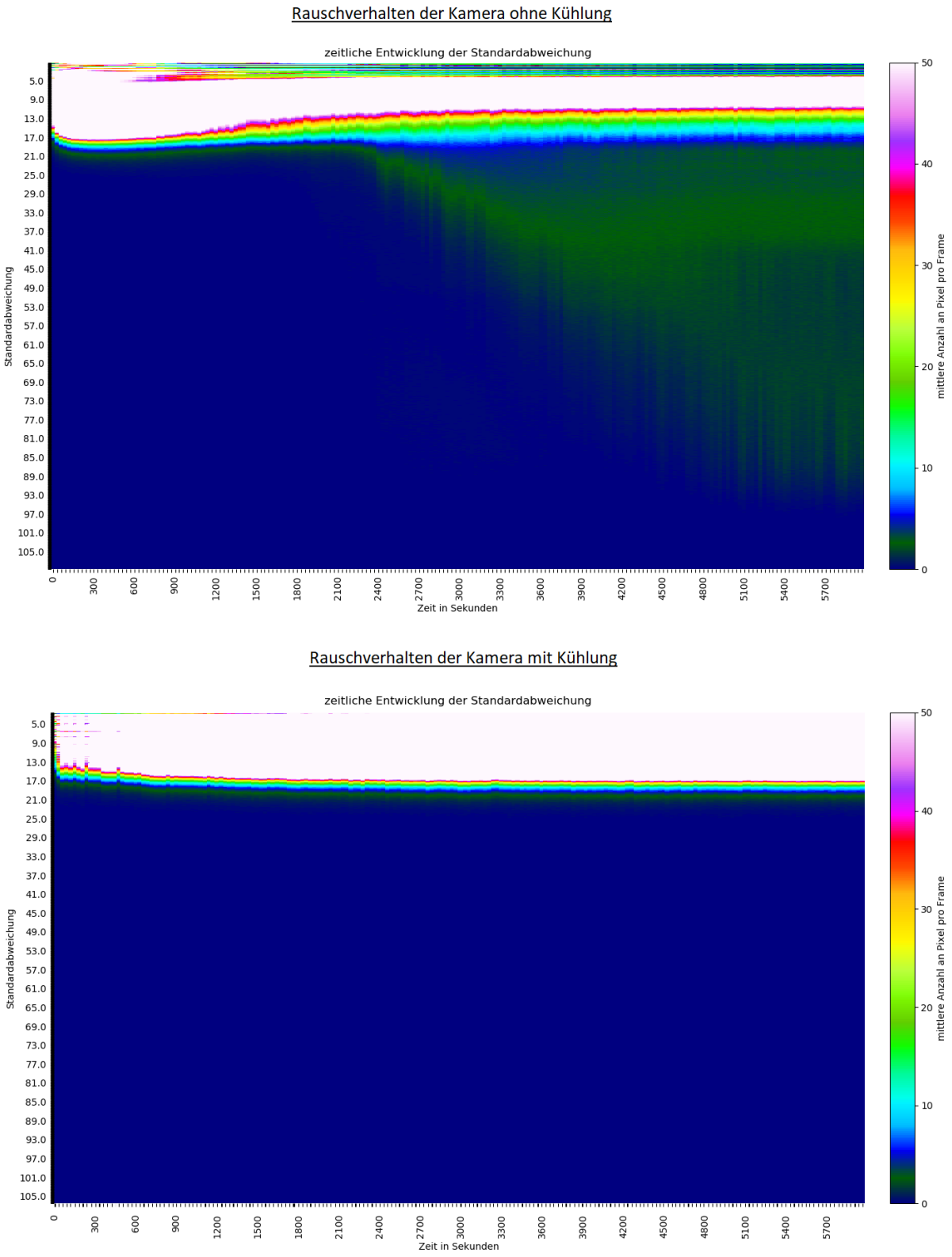


Abbildung A.3: Verdeutlichung Einfluss der Kühlung durch ein Peltier-Element auf die zeitliche Entwicklung des Rauschverhaltens der Pixel der Kamera mit Objektivkappe

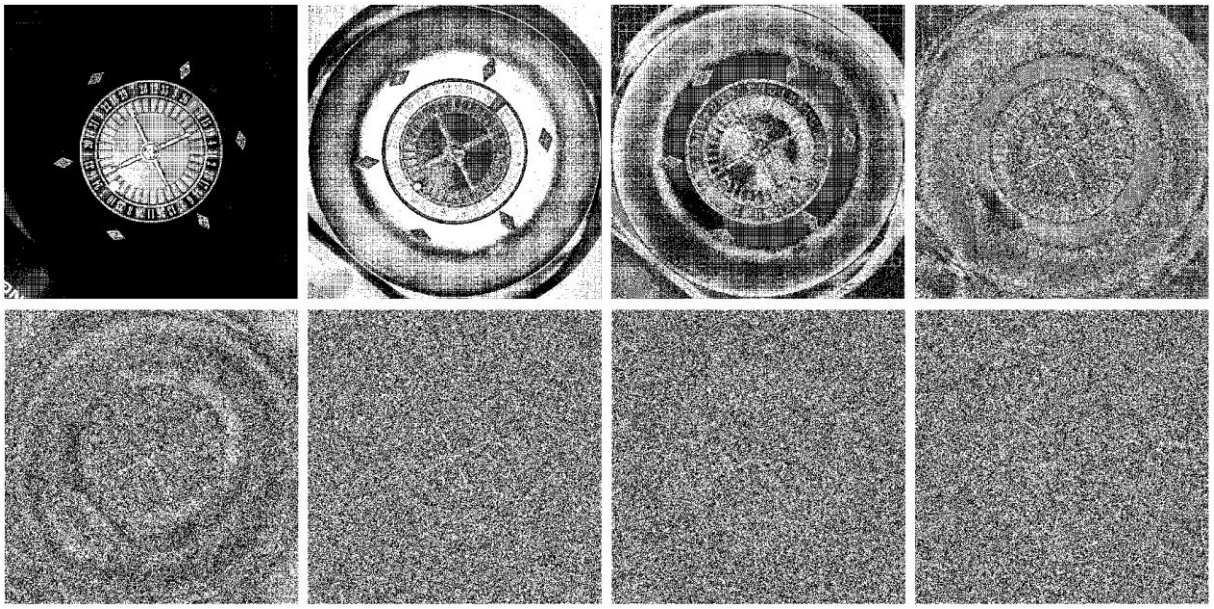


Abbildung A.4: Informationsgehalt einzelner Bits einer Aufnahme — mit dem in Kapitel 5 spezifizierten Versuchsaufbau — zeilenweise vom MSB (oben links) bis zum LSB (unten rechts)

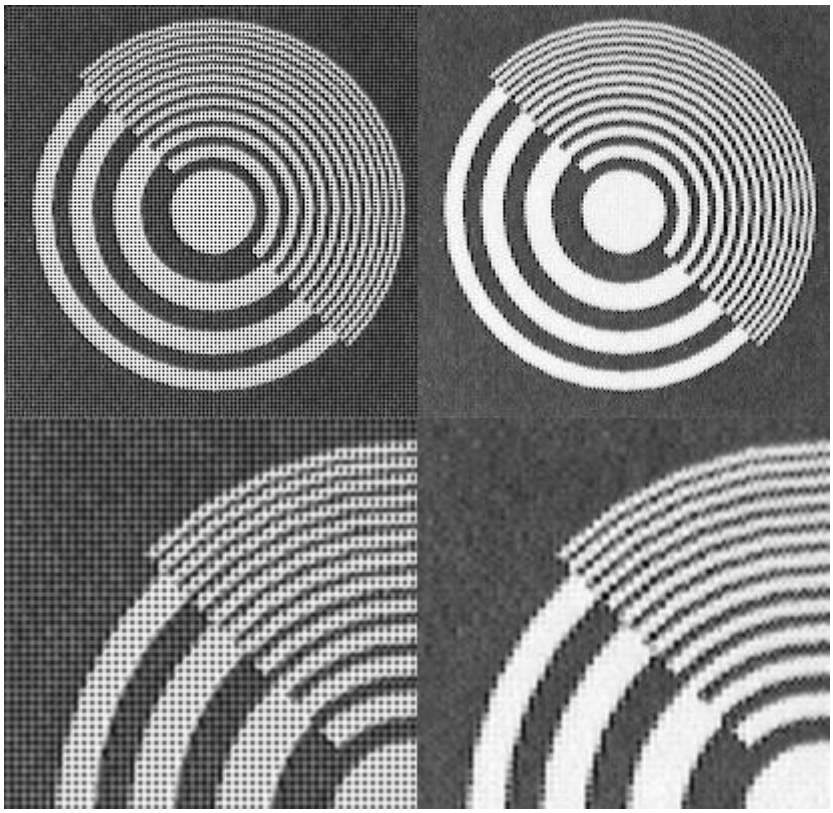


Abbildung A.5: Einfluss des Weißabgleichs nach Umschalten von weißem zu gelblichem Licht wie es in dieser Arbeit verwendet wurde (links ohne Weißabgleich, rechts mit Weißabgleich) auf Rohdaten im Bayer-Format

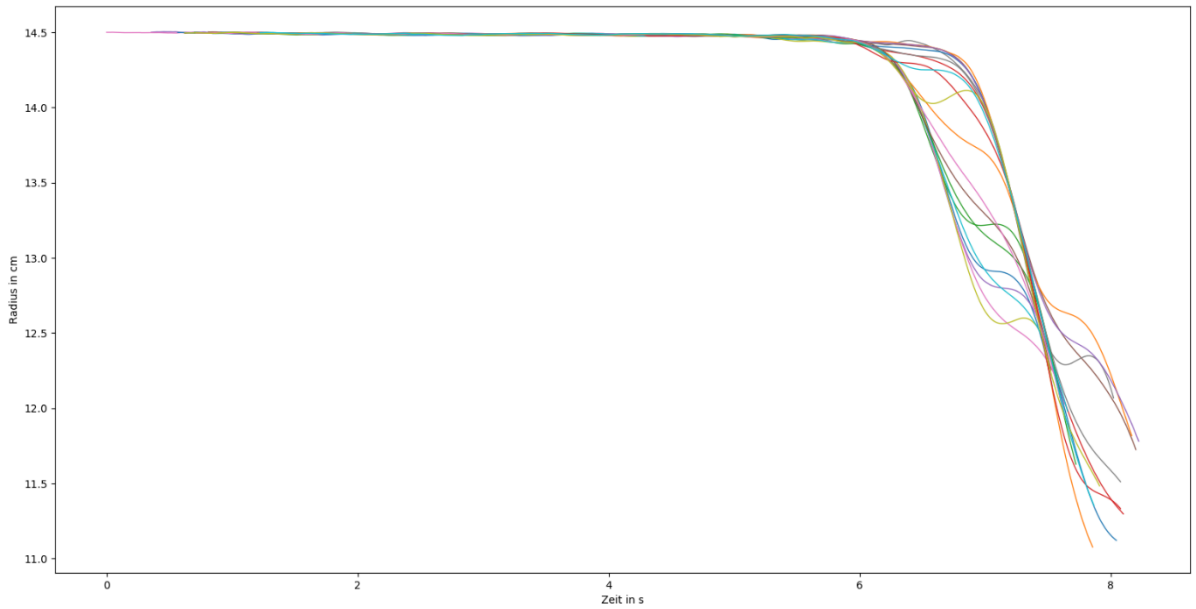


Abbildung A.6: Radien von 20 Kugelverläufen zum Mittelpunkt bis zum Eintritt in die Chaosphase nach bestem Ermessen übereinander gelegt (weiterführende Visualisierung der Messergebnisse aus Kapitel 5.2)

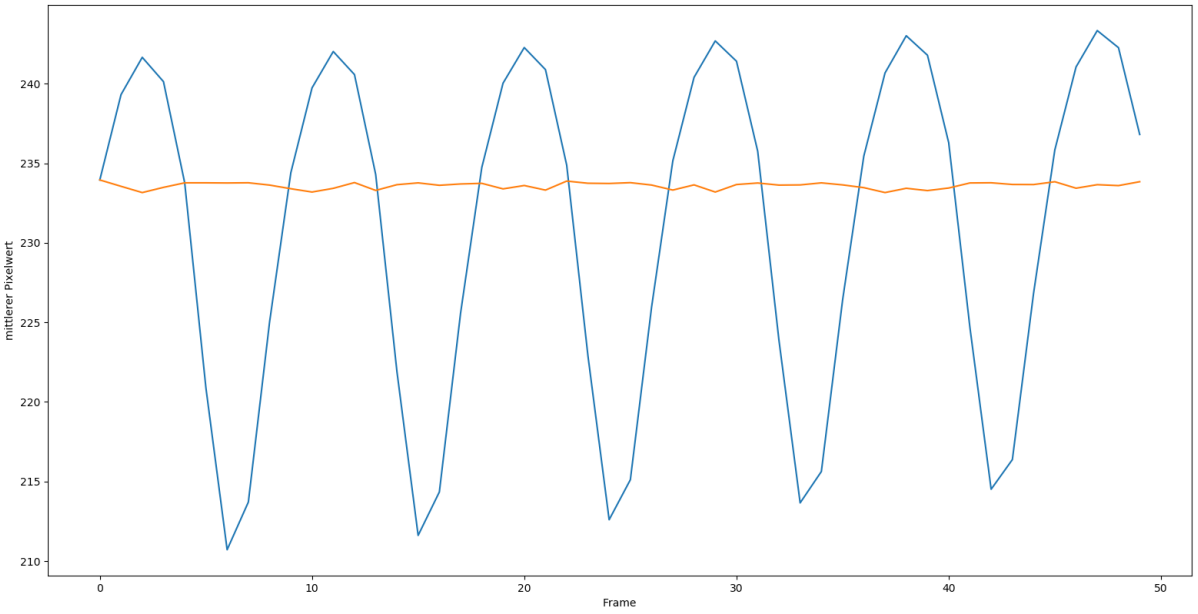


Abbildung A.7: Mittlerer Helligkeitswert eines Videos eines weißen Blatts Papier vor der Korrektur stroboskopischer Effekte (Blau) und danach (Orange) welche in der Software *video_processing.py* Anwendung findet

Anhang B Gegenüberstellung der Kostenfunktionen

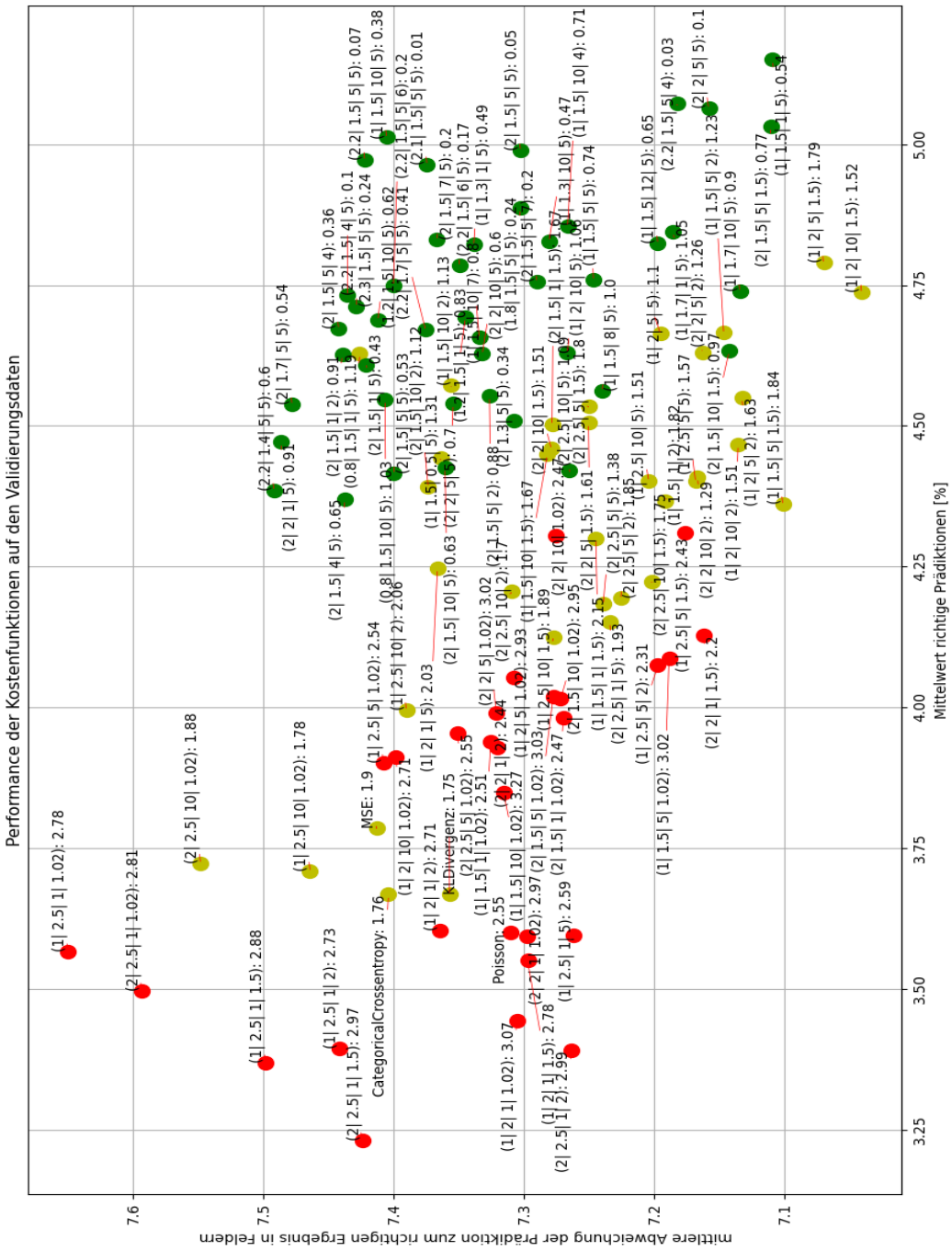


Abbildung B.1: Gegenüberstellung aller Messergebnisse zur Optimierung der Kostenfunktion mit verschiedener Parametrierung und konventionellen Ansätzen auf den Validierungs-Vektordaten der Spiralphase nach dem Schema aus Abbildung 6.9

Anhang C Einfluss des Dropout Verfahrens

Nachfolgend sind beispielhafe Ausschnitte zum Einfluss des Dropout-Verfahrens bei unterschiedlicher Netzstruktur und verschiedenen Kostenfunktionen zu sehen.

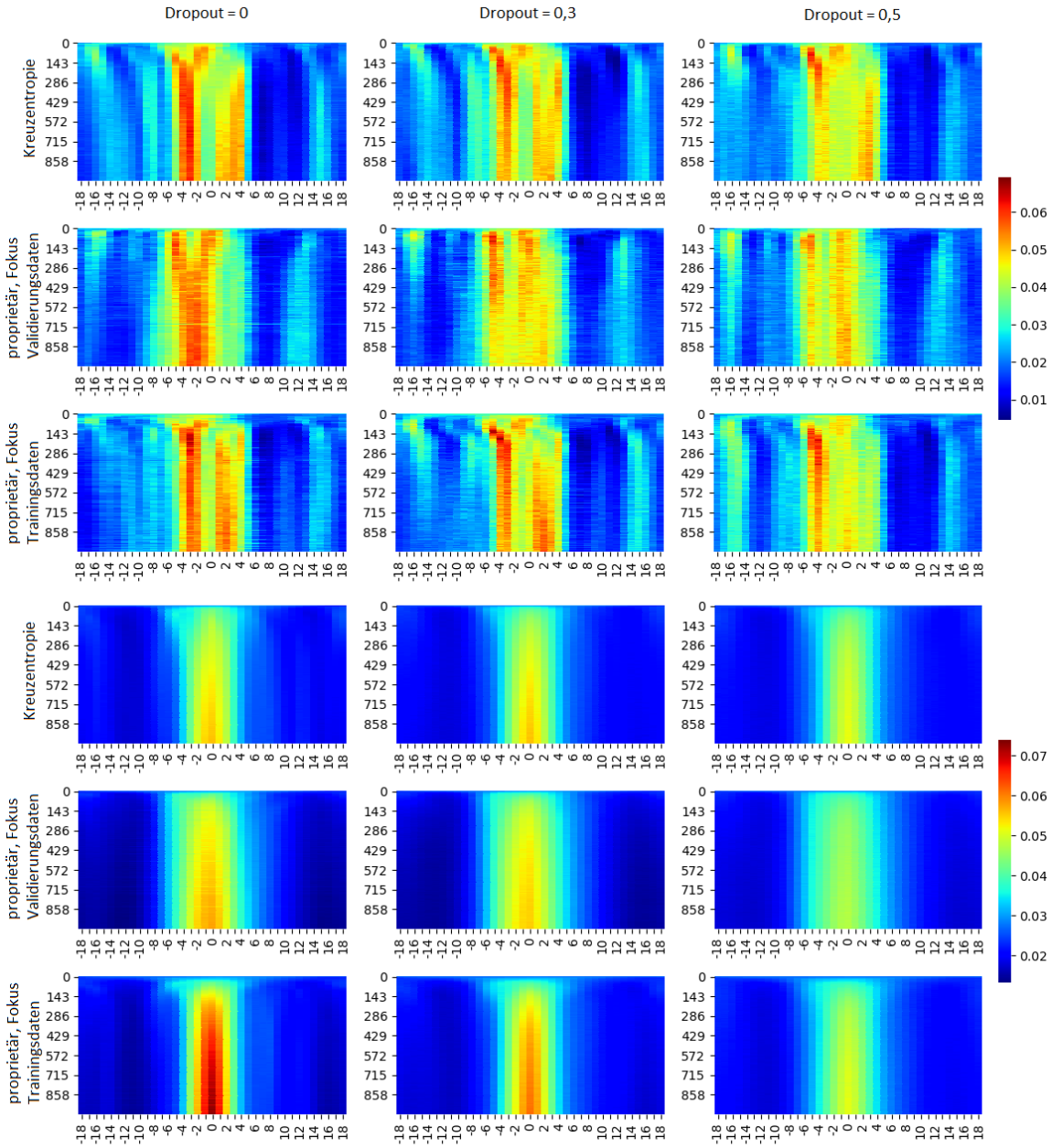


Abbildung C.1: Gegenüberstellung des Prädiktionsverhaltens relativ zum richtigen Ergebnis eines verhältnismäßig schmalen Netzes mit proprietären Ansätzen als Kostenfunktion (optimiert auf hohe Performance bei den Trainings- bzw. Validierungsdaten) und der Kreuzentropie auf den Trainingsdaten (unten) und den Validierungsdaten (oben) von Vektoren der Spiralphase über mehrere Epochen

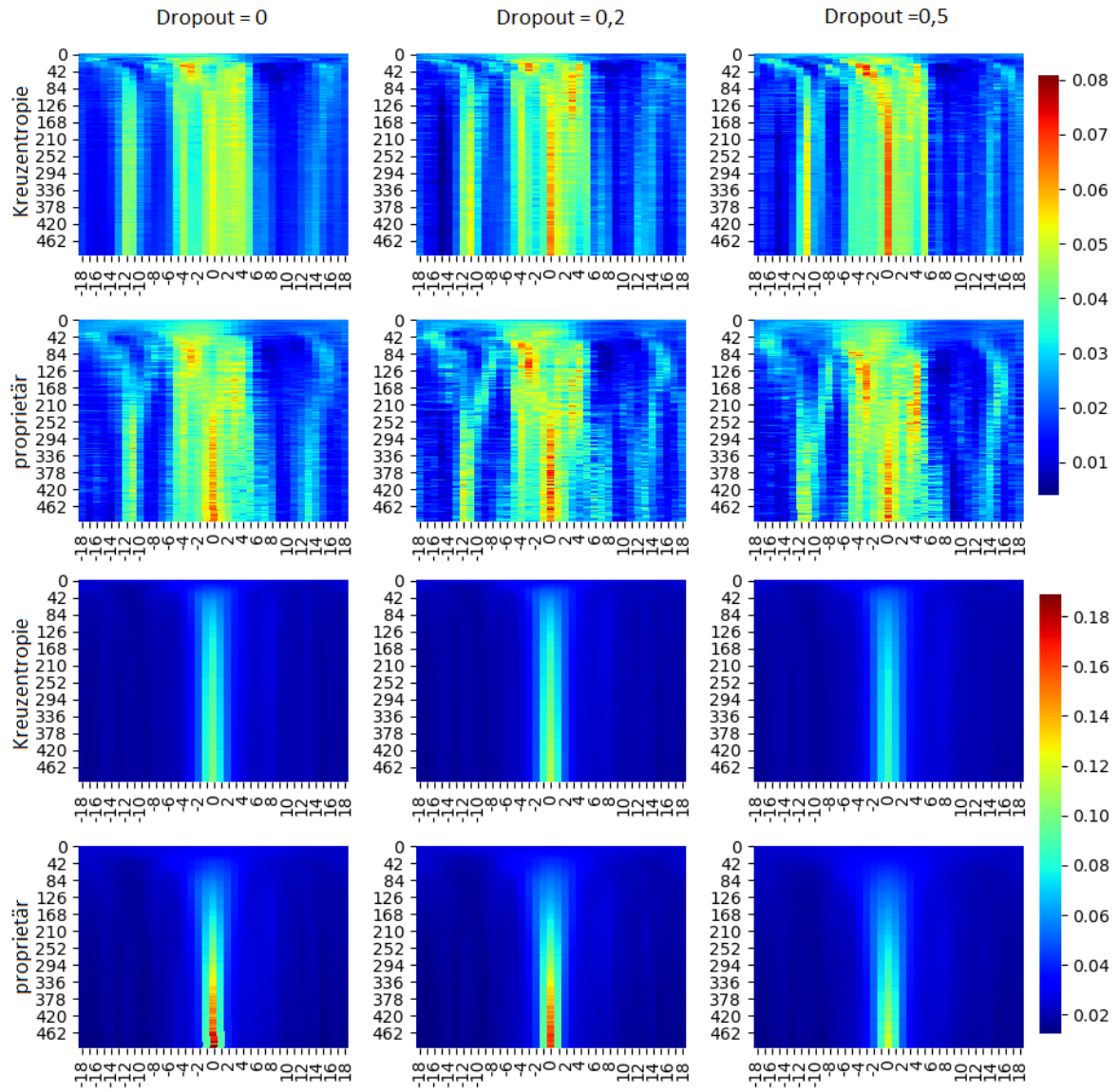


Abbildung C.2: Gegenüberstellung des Prädiktionsverhaltens relativ zum richtigen Ergebnis eines verhältnismäßig breiten Netzes mit proprietärem Ansatz als Kostenfunktion (optimiert auf hohe Performance bei den Trainingsdaten) und der Kreuzentropie auf den Trainingsdaten (unten) und den Validierungsdaten (oben) von Vektoren der Spiralphase über mehrere Epochen

Anhang D Einfluss der Batch-Normalisierung

In den nachfolgenden Abbildungen sind beispielhafte Ausschnitte der Messergebnisse zur Untersuchung der Abhängigkeit des Lernverhaltens von der Batch-Normalisierung dargestellt. Abbildung D.2 beschreibt zudem ein relativ großes Netz mit sechs inneren Schichten in Kombination mit zwei Dropout-Schichten vor der fünften und sechsten Schicht und einem Dropout-Anteil von 0,3 mit welchem sehr gute Ergebnisse erzielt wurden.

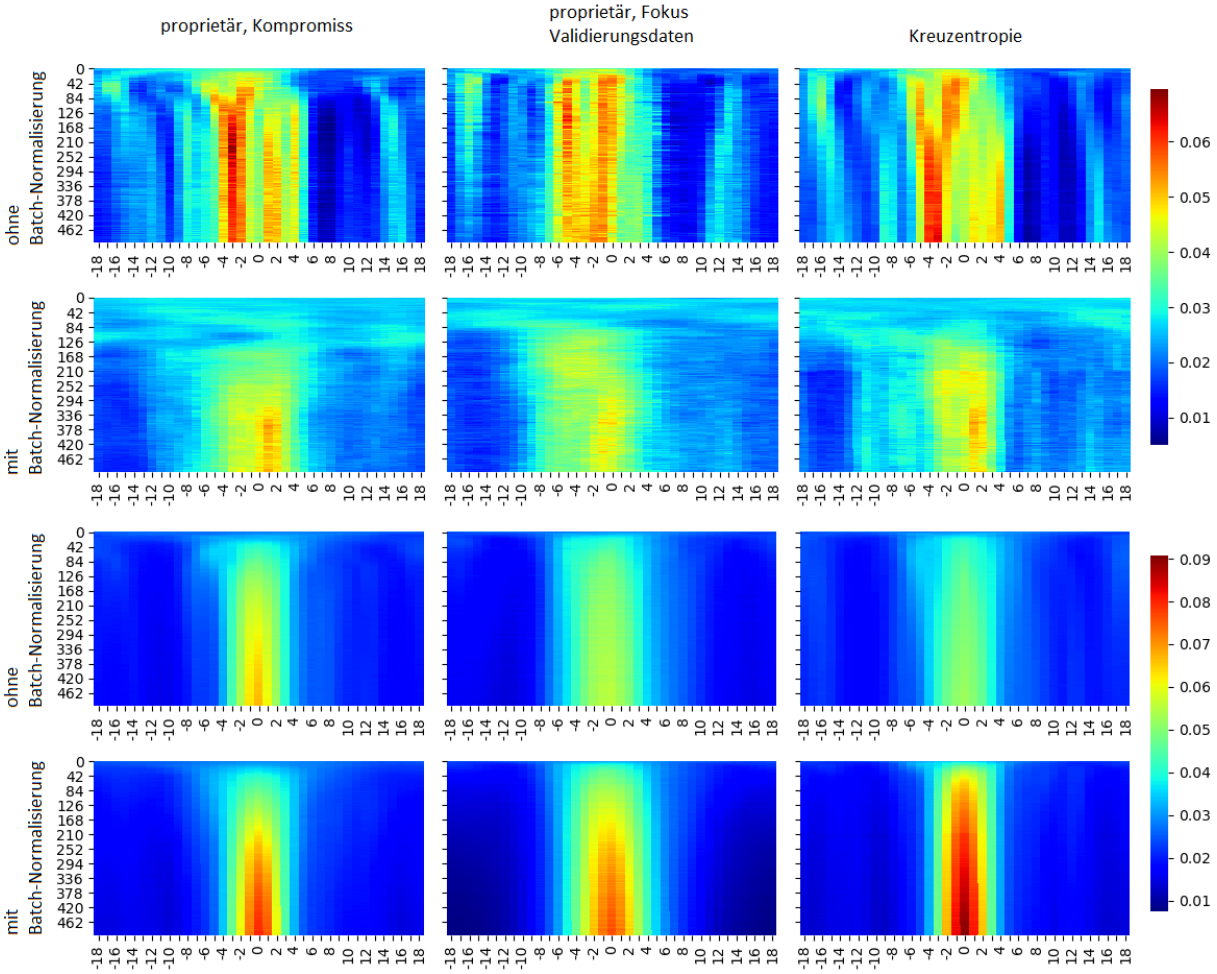


Abbildung D.1: Gegenüberstellung des Prädiktionsverhaltens relativ zum richtigen Ergebnis unter Einfluss der Batch-Normalisierung eines verhältnismäßig kleinen Netzes mit proprietären Ansätzen als Kostenfunktion (optimiert auf hohe Performance bei den Validierungsdaten bzw. Kompromiss zwischen Trainings- und Validierungs-Performance) und der Kreuzentropie auf den Trainingsdaten (untere Hälfte) und den Validierungsdaten (obere Hälfte) von Vektoren der Spiralphase über mehrere Epochen

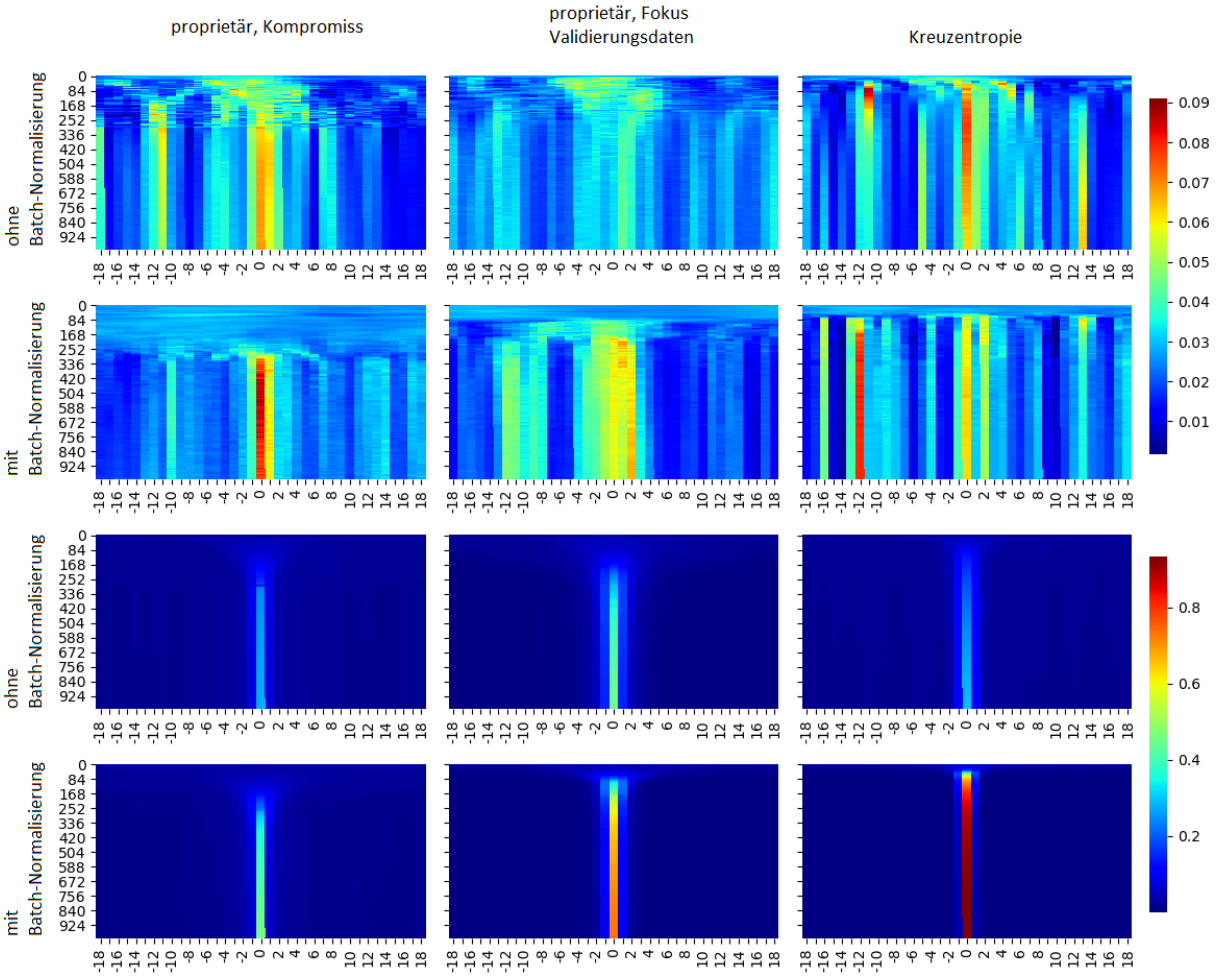


Abbildung D.2: Gegenüberstellung des Prädiktionsverhaltens relativ zum richtigen Ergebnis unter Einfluss der Batch-Normalisierung eines verhältnismäßig großen Netzes mit proprietären Ansätzen als Kostenfunktion Kostenfunktion (optimiert auf hohe Performance bei den Validierungsdaten bzw. Kompromiss zwischen Trainings- und Validierungs-Performance) und der Kreuzentropie auf den Trainingsdaten (untere Hälfte) und den Validierungsdaten (obere Hälfte) von Vektoren der Spiralphase über mehrere Epochen

Anhang E Übersicht entwickelte Software

Nachfolgend ist eine Liste an Kurzbeschreibungen beigefügter, für diese Arbeit entwickelter, Software. Es existieren zudem Programme zur Untersuchung kleinerer Tests zur Rauschentwicklung der Kamera, zur Kontinuität der Frames während der Aufnahme, zu möglichem Datenverlust bei Speichermethoden, zu Untersuchungen der Bildqualität, verschiedene Visualisierungen und Aufbereitungen von Messergebnissen und diverse Weitere. Deren Relevanz ist für diese Arbeit jedoch relativ gering weshalb sie hier nur erwähnt wurden.

conv2d_videodata.py
Softwarebeispiel zum Training faltender neuronaler Netze. In diesem sind alle verwendeten Verfahren enthalten, welche in dieser Arbeit untersucht wurden.

create_image_difference.py
Erzeugung der Differenzbilder entsprechend der Spezifikationen aus Kapitel 6.1.3.

custom_callbacks.py
Beinhaltet alle in Kapitel 6.2.5 definierte Callbacks.

custom_functions.py
Für diese Arbeit entwickelte Funktionen.

custom_loss_functions.py
Finale Implementierung der Kostenfunktion aus Kapitel 4.5.

custom_metrics.py
Beinhaltet alle in Kapitel 6.2.4 definierten Metriken.

cut_vectordata.py
Zuschnitt der Vektordaten entsprechend der Spezifikationen aus Kapitel 6.1.5.

cut_videodata.py
Zuschnitt der Videodaten entsprechend der Spezifikationen aus Kapitel 6.1.5.

duplication_of_vectordata.py
Vervielfältigung der Vektordaten entsprechend der Spezifikationen aus Kapitel 6.1.4.

duplication_of_videodata.py
Vervielfältigung der Videodaten entsprechend der Spezifikationen aus Kapitel 6.1.4.

evaluate_model.py
Dient zur Evaluierung von gespeicherten Gewichten anhand definierter Daten. Die Darstellung erfolgt in Confusion-Matrizen.

FFNN_vectordata.py
Softwarebeispiel zum Training neuronaler Netze mit Vektordaten. In diesem sind alle verwendeten Verfahren enthalten, welche in dieser Arbeit untersucht wurden.

record_video.py
Software zur Aufnahme der Videos gemäß der Spezifikationen aus Kapitel 6.1.1.

vector_data_input_pipeline.py
Überarbeiteter Code der Funktion *preprocessing.text_dataset_from_directory* aus Keras gemäß der Spezifkiationen aus Kapitel 6.2.1.

video_processing.py
Aufbereitung von Videodaten gemäß der Spezifikationen aus Kapitel 6.1.2.

visualize_metrics.py
Visualisierung von Metrik-Dateien welche durch den Callback *SaveMetrics* aus Kapitel 6.2.5 erzeugt wurden.

metric_example.txt
Metrik-Datei, welche von *visualize_metrics.py* ausgelesen werden kann. Diese beinhaltet den Lernverlauf des besten Netzes zur Prädiktion von Vektordaten der Außenwandphase aus Kapitel 7.3.