
Masterarbeit

Ableitung von Bewegungsmodellen für Anwendungen in der
Schüttgutsortierung mittels Machine Learning

Tobias Hornberger

31. Dezember 2018

Referent: Prof. Dr.-Ing. Uwe D. Hanebeck

Korreferent: Prof. Dr.-Ing. Thomas Längle

Betreuer: Dipl.-Inform. Florian Pfaff
Georg Maier, M. Sc.
Dr.-Ing. Benjamin Noack

Zusammenfassung

This work was a cooperation with the Fraunhofer IOSB and centered around their TableSort bulk material sorting system. The aim was to provide an alternative to the labour-intensive task of fine tuning motion models for particle tracking by hand. This was achieved by using neural networks, which were implemented using the Tensorflow framework. [TODO - NOT COMPLETE]

Eidesstattliche Erklärung

Hiermit erkläre ich, die vorliegende Masterarbeit selbstständig angefertigt zu haben.
Die verwendeten Quellen sind im Text gekennzeichnet und im Literaturverzeichnis aufgeführt.

Karlsruhe, 31. Dezember 2018

Tobias Hornberger

Inhaltsverzeichnis

Abbildungsverzeichnis	III
Notation	v
1 Einleitung	1
1.1 Motivation	1
1.2 Aufbau der Arbeit	3
2 Grundlagen	7
2.1 Neuronale Netze	7
2.1.1 Perzepron	8
2.1.2 Feedforward Netze	9
2.1.3 Aktivierungsfunktionen	10
2.1.4 Performance-Maß	11
2.1.5 Regularisierung	13
2.2 TableSort System	14
2.3 Stand der Technik	18
3 Datenverarbeitung	21
3.1 Eigene Aufnahmen	21
3.2 Datenpipeline	22
3.3 Simulierte Daten	23
3.4 Datenformatierung	25
3.5 Datenaugmentierung	27
3.6 Datenumfang	28
4 Umsetzung und Implementierung	37
4.1 Software	37
4.2 Code-Struktur	37
4.3 Hyperparameter	38
4.4 Hyperparameter-Tuning	43

Inhaltsverzeichnis

5 Evaluation	47
5.1 Vergleichsmodelle	47
5.2 Next Step	51
5.3 Separator	52
5.4 Zusammenfassung und Diskussion	57
6 Fazit und Ausblick	61
Literatur	63
A Anhang	65

Abbildungsverzeichnis

1.1	Darstellung einer Fehlseparierung durch die Annahme, dass es keine Bewegung orthogonal zur Transportrichtung gibt. Übersetzt aus [Pfa18].	2
1.2	Visualisierung einer gelösten Probleminstanz eines NextStep-Netzes	4
1.3	Visualisierung einer gelösten Probleminstanz eines Separator-Netzes	5
2.1	Schematischer Aufbau Perzeptron	8
2.2	Vergleich zwischen einem linear und einem nichtlinear separierbaren Problem. Angepasst von [WAZ11]	9
2.3	Schaubild der Ausgabe einer ReLU	11
2.4	Beispiel für die Dropout-Regularisierung [SHK+14]	14
2.5	Visualisierung von Modellen mit unterschiedlichen Kapazitäten	15
2.6	TableSort Schüttgutsortiersystem [Fra17]	16
2.7	Schematische Darstellung des optischen Bandsortierers TableSort nach [PKP+17]	17
3.1	Verschiedene gesammelte Schüttgüter	22
3.2	Verschiedene Schüttgüter auf dem Förderband bzw. auf der Rutsche	29
3.3	Bayer-Matrix auf einem Sensor [WC06]	30
3.4	Visualisierung der DEM-Simulation, übersetzt aus [Pfa18]	30
3.5	Beispiel Feature-Label-Paar NextStep	31
3.6	Geometrische Bestimmung der Labels, nach [Pfa18]	32
3.7	Beispiel Feature-Label-Paar Separator	33
3.8	Visualisierung der Datenaugmentierung durch Spiegelung	34
3.9	Balkendiagramm Verteilung Bilder	35
3.10	Balkendiagramm Verteilung Tracks	35
4.1	Skizzierter Trainingsablauf eines neuronalen Netzes	39
4.2	Skizzierte Ablauf der Evaluation eines neuronalen Netzes	39
4.3	Architektur eines Neuronales Netzes für die NextStep-Prädiktion	44
4.4	Architektur eines Neuronales Netzes für die Separator-Prädiktion	45

Abbildungsverzeichnis

5.1	Visualisierung der Ergebnisse für die realen und simulierten Kugeln mit NextStep-Netzen	53
5.2	Fehlerhistogramm für die NextStep-Ergebnisse der selbst aufgenommenen Kugeln auf dem Förderband	54
5.3	Visualisierung der Ergebnisse für die Plättchen aus der DEM-Simulation	56
5.4	Visualisierung der Ergebnisse für den Weizenkörnern-Datensatz von den selbst gesammelten Daten. Median näher an 0 und möglichst kleine Streuung sind besser.	58
A.1	Visualisierung der Ergebnisse für die Pfefferkörner aus dem selbst aufgenommenen Datensatz.	66
A.2	Visualisierung der Ergebnisse für die Kugeln aus dem selbst aufgenommenen Datensatz.	67

Notation

Konventionen

- x Skalar
 \boldsymbol{x} Zufallsvariable
 \hat{x} Erwartungswert der Zufallsvariable \boldsymbol{x} .
 \underline{x} Spaltenvektor
 $\underline{\boldsymbol{x}}$ Zufallsvektor
 $\hat{\underline{x}}$ Erwartungswert des Zufallsvektors $\underline{\boldsymbol{x}}$.
 \mathbf{A} Matrix
 $(\cdot)_k$ Quantität zum Zeitpunkt k .
 \mathbb{R} Menge der reellen Zahlen.
 \sim Verteilungsoperator.
Z.B. bedeutet $\boldsymbol{x} \sim \mathcal{U}$, dass \boldsymbol{x} gemäß der Verteilung \mathcal{U} verteilt ist.
■ Ende eines Beispiels.
□ Ende eines Beweises.

Abkürzungen

PNG	Portable Network Graphics
CSV	Comma Separated Values
JSON	JavaScript Object Notation
HSV-Farbraum	Hue Saturation Value - Farbraum
RMSE	Root Mean Square Error
MSE	Mean Square Error
DEM	Diskrete-Elemente-Methode

Notation

NN	Neuronales Netz
CV	Constant Velocity
CA	Constant Acceleration
CVBC	Bias-Corrected Constant Velocity
AA	Average Acceleration
IA	Identical Acceleration

KAPITEL 1

Einleitung

Schüttgüter und ihr Transport sind aus unserer modernen, globalisierten Welt nicht mehr wegzudenken. Als Schüttgüter werden solche Materialien bezeichnet, die lose in großen Mengen transportiert werden. Dazu gehören Lebensmittel wie Getreide, Kaffeebohnen oder Zucker, aber auch Bergbauerzeugnisse, wie zum Beispiel Erze oder Kohle. Auch Rohstoffe in Pellet-Form und Granulate zählen zu dieser Kategorie [Sch09]. Laut dem Seefrachtbericht der Vereinten Nationen wurden im Jahr 2017 insgesamt 3196 Millionen Tonnen von den größten Schüttgütern Eisenerze, Kohle und Getreide verschifft [UC18, Unterabschnitt 1.A.2]. Der Anteil, der für die Verarbeitung von Schüttgutmaterialien verwendeten Energie am globalen Energieverbrauch wird als 10 % angegeben [Dur12, Abschnitt 1.2]. In Zeiten der globalen Erwärmung ist der verantwortungsvolle Umgang mit Energie ein wichtiges Thema und eine effizientere Sortierung von Schüttgütern kann dazu beitragen. Für das Sortieren solcher Schüttgüter gibt es verschiedene Methoden, wie zum Beispiel durch den Einsatz von Sieben, Magneten oder Flotationszellen. Diese Sortiermethoden basieren auf unterschiedlichen physikalischen Eigenschaften der Schüttgutpartikel. Im Rahmen dieser Arbeit wird die optische Schüttgutsortierung behandelt, die stattdessen auf den optischen Eigenschaften der einzelnen Schüttgutpartikel aufbaut. In den letzten Jahren haben neuronale Netze für einige beeindruckende Fortschritte in verschiedenen Forschungsfeldern gesorgt, wie zum Beispiel in den Bereichen der Bildverarbeitung und der Spracherkennung. Um die Qualität der Schüttgutsortierung zu steigern, wird im Rahmen dieser Arbeit der Einsatz von neuronalen Netzen für die Bewegungsprädiktion der einzelnen Partikel erprobt.

1.1 Motivation

Der Großteil der heute in der Industrie eingesetzten optischen Schüttgutsortierer verwenden Zeilenkameras. Dabei muss jedoch die Annahme getroffen werden, dass

die Schüttgutpartikel keinerlei Geschwindigkeit orthogonal zur Transportrichtung haben, weil diese nicht erfasst werden kann. In Abbildung 1.1 ist dargestellt wie es zu einer Fehlseparierung kommen kann, wenn diese Annahme verletzt wird. Durch den Einsatz von Flächenkameras ist es möglich, die Position eines Partikels auf dem Förderband zu mehreren Zeitpunkten zu bestimmen. Basierend auf diesen Informationen sollen die Trajektorien der einzelnen Partikel vorhergesagt werden. Diese sollen dazu verwendet werden, um die Sortierqualität zu steigern. Im Rahmen des *TrackSort* Projekts, einer Kooperation zwischen dem Lehrstuhls für Intelligente Sensor-Aktor-Systeme (ISAS) des Karlsruher Instituts für Technologie und dem Fraunhofer-Institut für Optronik, Systemtechnik und Bildauswertung (IOSB), wurde die Verbesserung der Schüttgutsortierung durch den Einsatz von Trackingverfahren betrachtet.

passt das hier rein, oder soll ich das eher wo anders erwähnen?

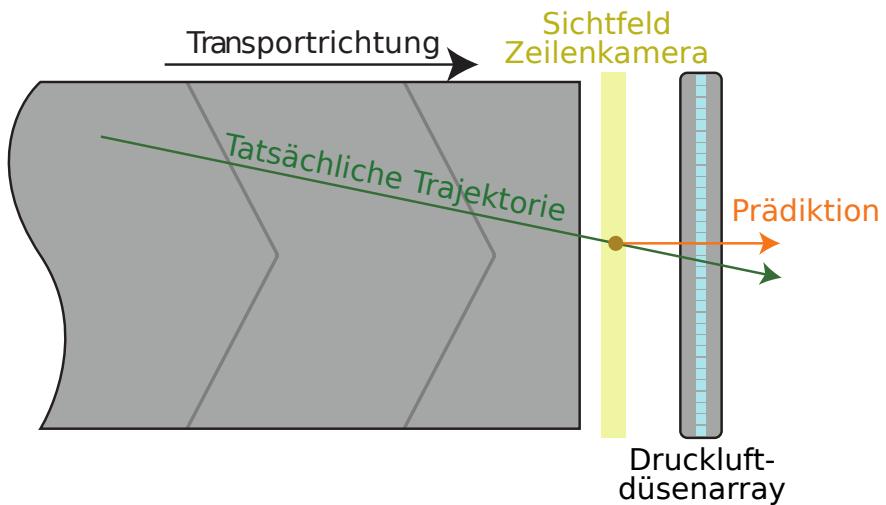


Abbildung 1.1: Darstellung einer Fehlseparierung durch die Annahme, dass es keine Bewegung orthogonal zur Transportrichtung gibt. Übersetzt aus [Pfa18].

Um die zukünftigen Trajektorien der Partikel aus den vergangenen Positionen vorherzusagen, existieren verschiedene Bewegungsmodelle. Diese liefern für unterschiedliche Situationen, mit unterschiedlichen Bandgeschwindigkeiten, Schüttguttypen oder Prädiktionsdistanzen, unterschiedlich gute Ergebnisse. Mittels neuronaler Netze können komplexe, nichtlineare Muster aus bestehenden Daten gelernt werden, ohne dass explizit auf diese hingewiesen werden müssen. Es ist also denkbar, dass sie in der Lage wären die Bewegungen des Schüttguts zu lernen. Die für den Einsatz von neuronalen Netzen essenziellen Daten in ausreichender Menge zu sammeln ist definitiv möglich. In dieser Arbeit soll nun erforscht werden, inwiefern der Einsatz von neuronalen Netzen zu einer Verbesserung gegenüber den bestehenden Ergebnissen führen kann.

Dabei wird exemplarisch mit Daten von dem *TableSort* Schüttgutsortierer gearbeitet.

Dafür sollen im Rahmen dieser Arbeit zwei verschiedene Prädiktionsprobleme durch neuronale Netze gelöst werden. Einerseits soll vorhergesagt werden, an welcher Position sich ein Teilchen im nächsten Zeitschritt befinden wird. Solche Netze werden von hier an als NextStep-Netze bezeichnet. Eine Visualisierung dieser Aufgabe ist in Abbildung 1.2 zu sehen. Dies hilft dabei, das Zuordnungsproblem bei Multi-Target-Tracking zu lösen. Andererseits soll vorhergesagt werden, an welcher Position und wann ein Teilchen das Druckluftdüsenarray passieren wird. Diese Problemstellung wird von sogenannten Separator-Netzen gelöst. Eine Visualisierung dieser Aufgabe ist in Abbildung 1.3 zu sehen. Die Qualität dieser Prädiktion ist ausschlaggebend für den Erfolg der Separation.

1.2 Aufbau der Arbeit

Nachdem im ersten Kapitel eine kurze Einführung und Motivation für das Thema gegeben wurde, wird Kapitel 2 den theoretischen Grundlagen gewidmet, die für das Verständnis der restlichen Arbeit notwendig sind. Dabei wird zu Beginn auf die Funktionsweise von neuronalen Netzen eingegangen, bevor die Funktionsweise und der Aufbau des *TableSort* Schüttgutsortierers beschrieben werden. Am Ende des Grundlagen-Kapitels wird auf den aktuellen Stand Technik eingegangen und erklärt wie das, was im Rahmen dieser Arbeit mittels neuronaler Netze getan werden soll, aktuell funktioniert. Kapitel 3 beschreibt die Daten, die dafür verwendet werden, woher sie stammen, wie sie verarbeitet wurden und in welcher Form sie dann in die neuronalen Netze eingegeben werden. Auch wird in diesem Kapitel auf die bereits existierenden Datensätze eingegangen, für die mittels der *Diskrete-Elemente-Methode* Schüttgut simuliert wurden, und das verwendete Datenaugmentierungsverfahren beschrieben. Danach wird auf den Umfang der verwendeten Daten eingegangen. In Kapitel 4 wird die Implementierung der Arbeit thematisiert. Dabei wird zu nächst auf die verwendete Software eingegangen, bevor die Struktur des Codes beschrieben wird. Es werden die für das Training und die Evaluation relevanten Hyperparameter beschrieben und die Optimierung derselben erläutert. Kapitel 5 beschäftigt sich mit der Evaluation der Ergebnisse. Nachdem kurz die Modelle erläutert werden, mit denen die Ergebnisse der neuronalen Netze verglichen werden, wird jeweils ein Blick auf den NextStep-Fall und den Separator-Fall geworfen und dann die Ergebnisse diskutiert. Den Abschluss der Arbeit stellt Kapitel 6 da, in dem die Ergebnisse der Arbeit noch einmal zusammengefasst werden und ein Ausblick darauf gegeben wird, was in der Zukunft denkbare Fortführungen wären.

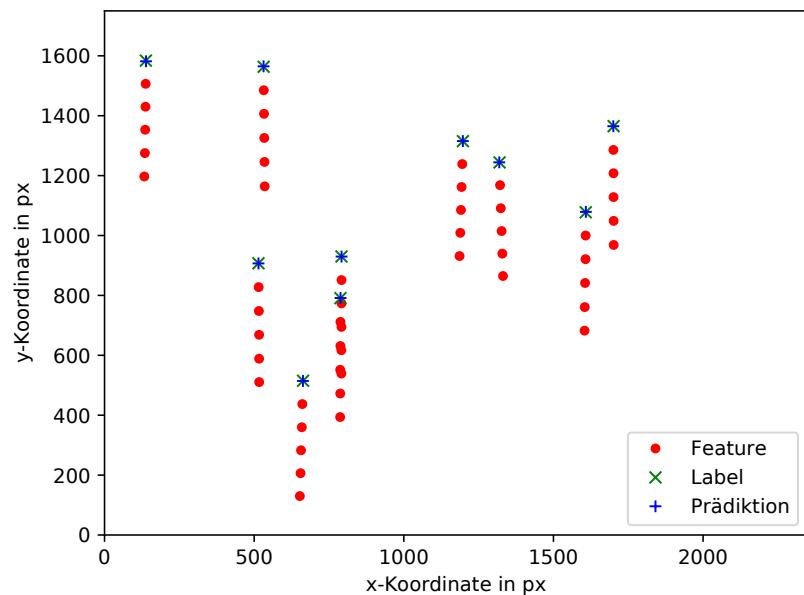


Abbildung 1.2: Visualisierung einer gelösten Probleminstanz eines NextStep-Netzes. Die Transportrichtung ist entlang der y -Achse nach oben. Die Features sind die zeitlich aufeinanderfolgenden, beobachteten Positionen eines Partikels. Das Label ist die Position des entsprechenden Partikels im nächsten Zeitschritt.

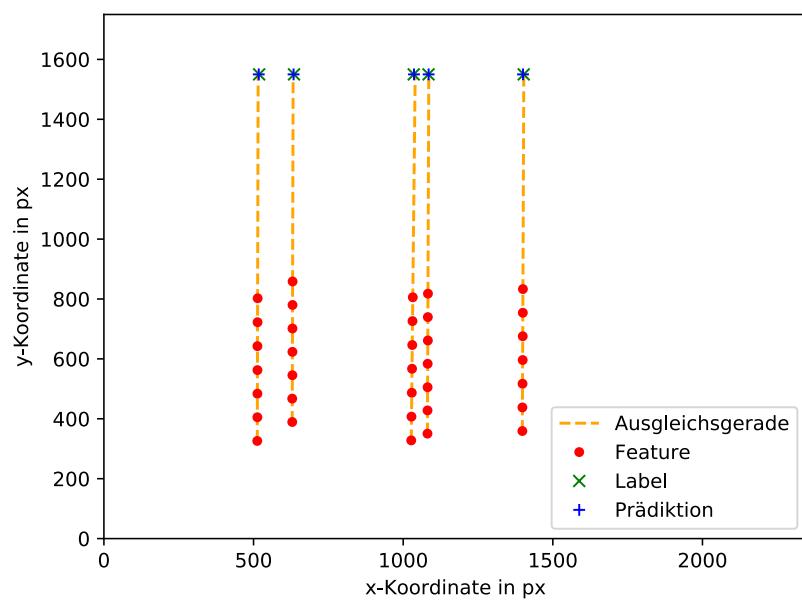


Abbildung 1.3: Visualisierung einer gelösten Probleminstanz eines Separator-Netzes. Wie in Abbildung 1.2 ist die Transportrichtung entlang der y -Achse nach oben. Die Features sind erneut die zeitlich aufeinanderfolgenden, beobachteten Positionen eines Partikels. Das eine Element des Labels ist die Position entlang der x -Achse an der das entsprechende Partikel das Druckluftdüsenarray passiert. Das zweite Element des Labels – der Zeitpunkt, an dem es das Druckluftdüsenarray passiert – ist nicht abgebildet.

KAPITEL 2

Grundlagen

In diesem Kapitel soll eine kurze Einführung in die für das Verständnis der restlichen Arbeit benötigten Themengebiete gegeben werden. Primär sollen zunächst allgemein neuronale Netze und einige ihrer speziellere Aspekte betrachtet werden, bevor ein kurzer Blick auf das bei den Experimenten verwendete Schüttgutsortiersystem *Table-Sort* geworfen wird. Am Ende des Kapitels wird darauf eingegangen, wie das Problem der Prädiktion bei optischen Schüttgutsorierern vor der Arbeit gelöst wird. Die Ausführungen zu den neuronalen Netzen sind, wo nicht anders angemerkt, basierend auf [GBC16], [Mur12] und [Nie15].

2.1 Neuronale Netze

Als Neuronale Netze bezeichnet man in der Informatik Systeme aus künstlichen Neuronen, die heute eine wichtige Rolle im Feld des maschinellen Lernens einnehmen. Manchmal werden sie korrekter als *künstliche neuronale Netze* bezeichnet um sie von *natürlichen neuronalen Netzen* wie dem menschlichen Gehirn zu unterscheiden, nach deren biologischem Vorbild sie modelliert sind. Neuronale Netze sind ein mächtiges Werkzeug. Aufgrund ihrer Fähigkeit komplexe, nichtlineare Muster und Zusammenhänge in Daten zu lernen, ohne dass sie explizit darauf programmiert werden müssen diese zu finden, eröffnen sie in vielen Forschungsrichtungen neue Möglichkeiten.

Die Grundsteine des Feldes wurde bereits 1943 von Warren McCulloch und Walter Pitts gelegt [MP43], als sie ein Neuronenmodell vorschlugen, mit dem sich logische arithmetische Funktionen berechnen lassen. Nach einigen Rückschlägen gab es Phasen von relativ geringer Aufmerksamkeit der wissenschaftlichen Gemeinschaft. Erst als um das Jahr 2010 einige herausragende Ergebnisse, unter anderem im Feld der Sprach- und Bilderkennung, mittels neuronaler Netze erzielt wurden, wurde das Interesse an dem Feld neu entfacht.

2.1.1 Perzeptron

Die kleinste Einheit eines neuronalen Netzes ist das Perzeptron, wie es 1958 von Frank Rosenblatt in [Ros58] beschrieben wurde. Es ist eine Art künstliches Neuron, dass eine Reihe an Eingaben entgegen nimmt und einen einzelnen Wert y ausgibt.

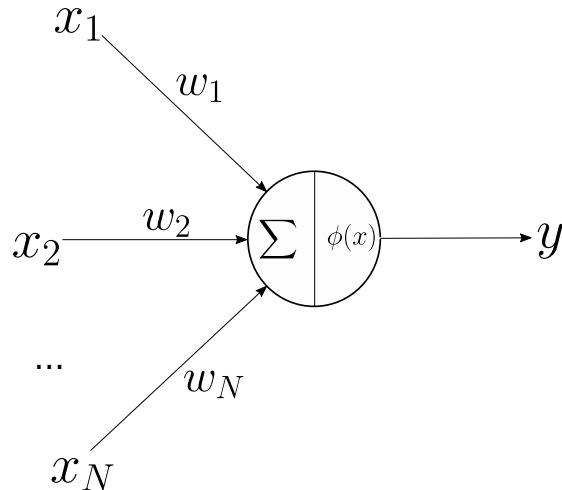


Abbildung 2.1: Aufbau eines Perzeptionsmodells: Die Eingaben x_i werden gewichtet mit w_i aufsummiert und nach Anwendung der Aktivierungsfunktion ϕ als y ausgegeben.

Wie in Abbildung 2.1 zu sehen haben die einzelnen Eingaben x_i jeweils eine Gewichtung w_i . Es existiert ein Schwellwert oder *bias*, der normalerweise durch eine zusätzliche Eingabe x_{m+1} mit dem Wert +1 und dem dazugehörigen Gewicht w_{m+1} modelliert wird. Den Ausgabewert y erhält man dadurch, dass man die gewichteten Eingaben aufsummieren und in die Aktivierungsfunktion ϕ des Perzeptionsmodells gibt. Mathematisch ist die Ausgabe eines Perzeptionsmodells also wie folgt definiert:

$$y = \phi\left(\sum_{i=0}^m w_i x_i\right)$$

Ein Überblick über verschiedene Aktivierungsfunktionen, die für solch ein Perzeptron benutzt werden, ist unter 2.1.3 zu finden. Beim Lernen werden die Gewichte w_i so angepasst, dass die gewünschte Ausgabe für die jeweilige Eingabe erreicht wird. In der Anwendung sind die Gewichte dann fest. Ein einzelnes Perzeptron mit zwei Eingängen kann zur Darstellung der logischen Operatoren AND, OR und NOT genutzt werden

Ein Perzeptron ist jedoch nur ein linearer Klassifikator und kann dementsprechend nur linear separierbare Probleme, wie das Problem in Abbildung 2.2a, lösen. Linear nicht separierbare Probleme wie zum Beispiel den XOR Operator oder das in Abbildung 2.2b gezeigte Problem, können nicht korrekt abgebildet werden. Dies zeigten

Velleicht ein paar Sätze zum Perceptron Learning Algorithm?

Marvin Minsky und Seymour Papert 1969 in einflussreichen Buch *Perceptrons: an introduction to computational geometry*. Um solche nicht linear-separierbare Probleme zu lösen müssen mehrere Perzeptrons hintereinandergeschaltet werden.

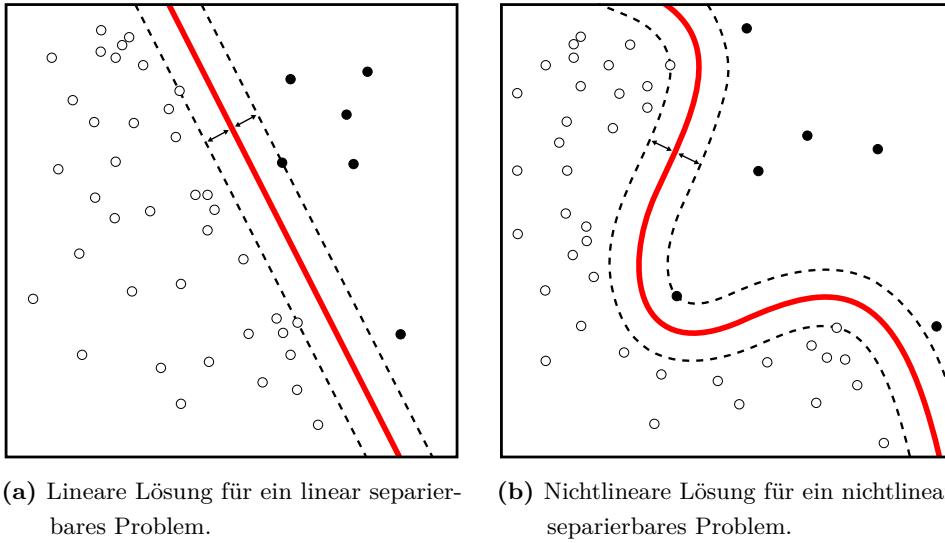


Abbildung 2.2: Vergleich zwischen einem linear und einem nichtlinear separierbaren Problem.
Angepasst von [WAZ11]

2.1.2 Feedforward Netze

Um die Limitation auf lineare Klassifikation eines einzelnen Perzeptrons zu überwinden kann man mehrere davon hintereinander schalten. Die einfachste Art dies zu tun wird als Feedforward Netz bezeichnet. Mit diesem Begriff beschreibt man ein neuronales Netz, zwischen dessen Knoten keine Kreise beziehungsweise Schleifen existieren. Die Informationen wandern in der Verarbeitungsrichtung von den Eingabeneuronen zu den Ausgabeneuronen. Für gewöhnlich sind die einzelnen Knoten in Schichten, sogenannten Layers, organisiert. Auf ein Eingabe- oder auch Input Layer folgen eine Menge an sogenannten Hidden Layers. Den Schlussstein bildet das Ausgabe- oder auch Output Layer. Die Neuronen eines einzelnen Hidden Layers sind meist uniform und verwenden identische Aktivierungsfunktionen, wie sie in Abschnitt 2.1.3 beschrieben sind. Je nachdem welche Aufgabe das neuronale Netz erfüllen soll können unterschiedliche Aktivierungsfunktionen für das Output Layer Sinn ergeben. Häufig genutzt ist die *Softmax* Funktion für Klassifikation, da sie eine Wahrscheinlichkeitsverteilung als Ausgabe hat. Für Regression benutzt man lineare Aktivierungsfunktionen für das Output Layer. Durch die Hintereinanderschaltung von mehreren Layers von Neuronen können auch Probleme, die nicht linear-separierbar sind, gelöst werden. Tatsächlich sind solche Neuronalen Netze bewiesenermaßen universale Funktionsapproximatoren - sie können mit endlich vielen Neuronen in den

Hidden Layers beliebige kontinuierliche Funktionen auf kompakten Subsets von \mathbb{R}^n approximieren. Das heißt zwar, dass es theoretisch für jede solche Funktion ein Netz gibt, das sie beschreiben kann, in der Praxis muss man dieses Netz aber finden und dann auch trainieren können.

mit Quelle belegen? Universal approximation theorem

Es gibt viele verschiedene Unterkategorien von Feedforward Netzen, die in verschiedensten Bereichen Verwendung finden. Convolutional Neural Networks zeichnen sich durch eine spezielle Struktur aus sogenannten Convolutional Layers aus, die als Feature Extractor fungieren. Diese Art von Netz findet unter anderem in der Bildverarbeitung Anwendung. Ein Netz, in dem Kreise beziehungsweise Schleifen existieren, bezeichnet man als *Rekurrentes Neuronales Netz*. Diese Netze werden vor allem dann eingesetzt, wenn die zeitliche Abfolge der Eingaben relevant ist, da sie im Gegensatz zu Feedforward Netzen einen internen Zustand halten.

2.1.3 Aktivierungsfunktionen

Es gibt verschiedene Aktivierungsfunktionen, die für den Einsatz in neuronalen Netzen in Frage kommen. Sie sind notwendig, da ohne eine Nicht-Linearität das Netz in eine lineare Regression kollabiert, die dann wiederum nicht in der Lage ist nicht-lineare Funktionen darzustellen.

Sowohl die Aktivierungsfunktion als auch ihre Ableitung sollte schnell zu berechnen sein, da dies im Rahmen des Trainings mit dem Backpropagation Algorithmus häufig geschieht und sonst beträchtlicher Rechenaufwand entsteht.

In der Vergangenheit wurden einige verschiedene Aktivierungsfunktionen verwendet. Jede dieser Funktionen stellt eine Nicht-Linearität dar und nimmt eine einzelne Zahl, wendet eine bestimmte, festgelegte mathematische Operation auf diese an und gibt das Ergebnis zurück. Historisch ist häufig die Sigmoid-Funktion $\sigma(x)$ verwendet worden, da sie das Verhalten eines natürlichen Neurons gut nachbildet.

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^{x+1}}$$

In der Praxis jedoch haben sich einige Nachteile der Sigmoid-Funktion gezeigt. Einige dieser Probleme konnten mit der Verwendung des Tangens hyperbolicus (\tanh) behoben werden, aber das Problem der Saturierung, dass ihre Ableitung bei großen Beträgen beinahe 0 wird, nicht. In letzter Zeit haben sich sogenannte *Rectified Linear Units*, oder kurz ReLUs, durchgesetzt.

$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 0 & , \text{ falls } x < 0 \\ 1 & , \text{ falls } x > 0 \end{cases}$$

Abbildung 2.3 zeigt das Schaubild einer solchen ReLU. Die Aktivierung von ReLUs ist ein einfacher Schwellwert, der weit weniger rechenintensiv ist, als die aufwendigen Exponentialfunktionen von Sigmoid und tanh. In der Praxis hat sich zudem gezeigt, dass ReLUs deutlich schneller konvergieren als Sigmoid- oder tanh-Neuronen, da sie kein Problem mit Saturierung haben. Krizhevsky et al. haben in ihrem Paper [KSH12] einen Geschwindigkeitsgewinn um Faktor 6 feststellen können. Ein Problem, das mit ReLUs jedoch existiert ist, dass einzelne Neuronen während dem Training „absterben“ können, falls sie irgendwann in dem Bereich landen, wo der Gradient 0 ist. Diese Neuronen sind dann für jeden beliebigen Input inaktiv und können niemals wieder etwas zur Ausgabe des Netzes beitragen. Durch die Wahl einer geeigneten Lernrate oder den Einsatz sogenannter Leaky ReLUs lässt sich dies jedoch vermeiden. Leaky ReLUs haben im Gegensatz zu normalen ReLUs eine kleine positive Steigung im negativen Bereich.

$$f(x) = \begin{cases} x & , \text{ falls } x > 0 \\ 0.01x & , \text{ falls } x \leq 0 \end{cases}$$

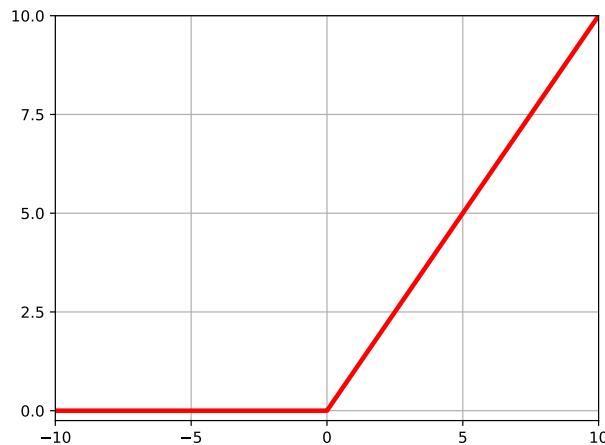


Abbildung 2.3: Schaubild der Ausgabe einer ReLU.

2.1.4 Performance-Maß

Neuronale Netze werden in der Regel danach bewertet, wie gut sie mit unbekannte, das heißt nicht im Training vorgekommenen, Daten umgehen können. Diese Eigenschaft von Trainingsdaten auf unabhängige Testdaten zu schließen wird Generalisierung genannt.

Als Overfitting bezeichnet man es, wenn ein Modell sich zu sehr an ein gegebenes Datenset anpasst und dafür in Kauf nimmt zusätzliche oder zukünftige Daten schlechter zu repräsentieren. Das System wird also schlechter darin zu generalisieren. Im Feld des überwachten Lernens - beziehungsweise der neuronalen Netze - ist Overfitting daran zu erkennen, dass die Qualität die Ausgaben des Netzes auf dem Trainingsdatenset sich weiter verbessert, während sie auf dem Testdatenset schlechter wird. Dies kann zum Beispiel der Fall sein, wenn das Modell Rauschen in den Testdaten als Teil der zugrundeliegenden Struktur interpretiert.

Dem Overfitting gegenüber steht das Underfitting. Als Underfitting bezeichnet man wenn das Netz nicht in der Lage ist eine ausreichend gute Performance auf den Trainingsdaten zu erreichen. Das kann passieren wenn das Modell nicht ausreichend komplex ist, um die zugrundeliegende Struktur der Daten abzubilden.

Sowohl Overfitting als auch Underfitting hängen mit der Kapazität eines Netzes zusammen. Ist die Kapazität zu gering, kann es sein, dass das Netz daran scheitert die Trainingsdaten zu lernen. Ist die Kapazität zu groß, so kann es passieren, dass das Netz gewissermaßen die Trainingsdaten einfach auswendig lernt.

Dies ist beispielhaft in Abbildung 2.5 zu sehen. Aus der zugrundeliegenden Cosinus Funktion werden Stichproben mit einem Rauschterm entnommen. Stellvertretend für das Lernen mit neuronalen Netzen wird hier lineare Regression benutzt, um die Parameter eines Polynoms zu lernen. Die Kapazität des Modells wird hier durch den Grad des Polynoms festgelegt.

In Abbildung 2.5a wird ein Polynom mit dem Grad 1 gelernt. Die Kapazität ist zu niedrig und dementsprechend schafft das Modell es nicht die Stichproben akkurat zu repräsentieren. In Abbildung 2.5b ist der Grad des Polynoms 4. Das resultierende Modell ist eine gute Approximation der ursprünglichen Funktion. In Abbildung 2.5c ist der Grad des Polynoms mit 17 deutlich zu hoch. Man kann am resultierenden Modell erkennen wie es dem Rauschen der Samples folgt und wie es an den Rändern des betrachteten Bereichs weit ausschlägt.

Die beste Methode, um Overfitting zu vermeiden, ist es mehr Trainingsdaten zu bekommen. In der Praxis kann dies oft umständlich, kostspielig oder sogar unmöglich sein, weshalb auf das Generieren von synthetischen Trainingsdaten, auch Datenaugmentierung genannt, zurückgegriffen wird. Auch der Einsatz von Regularisierungsverfahren kann helfen Overfitting zu verringern. Als letzter Schritt kann die Kapazität der Architektur gesenkt werden, zum Beispiel indem die Anzahl der zur Verfügung stehenden Layer reduziert wird.

2.1.5 Regularisierung

Als Regularisierung bezeichnet man Techniken, die zur Vermeidung von Overfitting verwendet wird. Im Gegensatz zu anderen Techniken des maschinellen Lernens, die den Fehler auf den Trainingsdaten reduzieren und damit potenziell indirekt auch den auf den Testdaten, soll hier primär der Fehler auf den Testdaten reduziert werden, selbst wenn das zu schlechteren Ergebnissen auf den Trainingsdaten führt. Das Ziel ist also bessere Generalisierung.

Eine Möglichkeit ist, dass zur Loss Funktion ein Regularisierungsterm R hinzugefügt wird, der die Kosten basierend auf der Komplexität des Systems erhöht.

$$\min_f \sum_{i=1}^m V(f(\underline{x}_i), \underline{y}_i) + \lambda R(f)$$

Dabei ist V die Loss Funktion, beispielsweise *Mean-Square-Error* oder *Mean-Absolute-Error*. n ist die Anzahl der Feature-Label-Paare, x_i und y_i sind die einzelnen Eingabefeatures und das dazugehörige Label. Die Funktion f ist in unserem Fall das neuronale Netz, das die Features entgegennimmt. λ ist ein Parameter, der die Gewichtung des Regularisierungsterm festlegt. Wählt man diesen Parameter zu klein, so kann es sein, dass das Modell trotz Regularisierung noch immer overfittet. Wählt man ihn zu groß, so kann es sein, dass das Modell das Problem nicht mehr korrekt abbildet und es zu Underfitting kommt. Der Regularisierungsterm R wird so gewählt, dass er die Komplexität der Funktion f widerspiegelt. Ein gutes Maß für die Komplexität eines neuronalen Netzes sind die Gewichte zwischen den Neuronen. Beispiele für R wären zum Beispiel die L_1 - oder die L_2 -Regularisierung. Der entscheidende Unterschied zwischen den beiden ist der unterschiedliche Strafterm, zu sehen in Gleichung (2.1) für L_1 und Gleichung (2.2) für L_2 . Die Fehlerfunktionen sind jeweils MSE mit dazugehörigen Strafterm.

$$J(X, Y) = \frac{1}{m} \sum_{i=1}^m (\underline{y}^{(i)} - \hat{\underline{y}}^{(i)})^2 + \sum_{j,k} (|\mathbf{W}_{j,k}|) \quad (2.1)$$

$$J(X, Y) = \frac{1}{m} \sum_{i=1}^m (\underline{y}^{(i)} - \hat{\underline{y}}^{(i)})^2 + \sum_{j,k} (\mathbf{W}_{j,k}^2) \quad (2.2)$$

Ein Regressionsmodell, das L_1 -Regularisierung verwendet wird auch als Lasso Regression bezeichnet, während ein Modell mit L_2 -Regularisierung als Ridge Regression beschrieben werden kann. Vergleicht man die beiden Ansätze, so schrumpft die L_1 -Norm weniger wichtige Gewichte auf 0, was zu dünn besetzten Gewichtsvektoren führt. Dies kann eine wünschenswerte Eigenschaft sein. Im Gegensatz dazu hat die

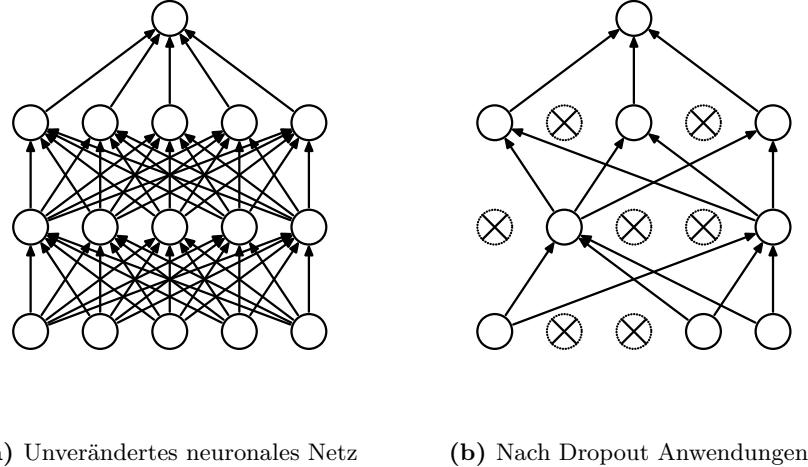


Abbildung 2.4: Beispiel für die Dropout-Regularisierung [SHK+14].

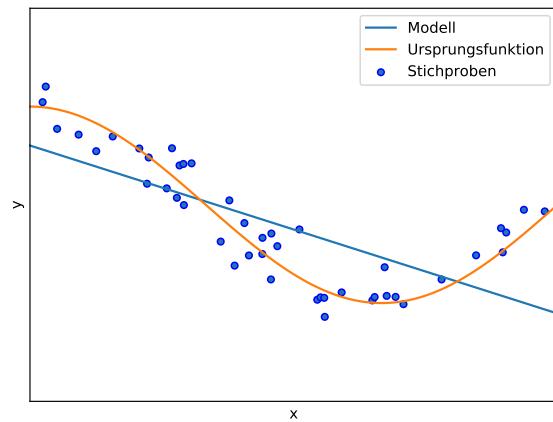
L_2 -Regularisierung, den Vorteil, dass sie effizienter berechnen kann. Der Strafterm von L_2 hat eine geschlossene Form und kann in Form einer Matrix angewendet werden, während die Funktion von L_1 auf Grund des Betrags eine nicht-differenzierbar ist.

Eine weitere Regularisierungstechnik ist Dropout [SHK+14]. Dabei werden während dem Training eines neuronalen Netzes mit einer festgelegten Wahrscheinlichkeit zufällig Neuronen und die dazugehörigen Verbindungen abgeschaltet, wie in Abbildung 2.4 dargestellt. Dies soll insofern Overfitting vermeiden, dass es übermäßige Koadaption von mehreren Neuronen erschwert. Dropout als Technik wird insbesondere bei tiefen neuronalen Netzen mit einer hohen Anzahl von Hidden Layers eingesetzt. Als Early Stopping wird eine Technik bezeichnet, bei der die Regularisierung durch das frühzeitige Beenden des Trainings erreicht wird [GBC16, Kapitel 7.8]. Dabei wird versucht abzuschätzen, an welchem Punkt des Trainings eine Verbesserung der Ergebnisse auf dem Trainingsset zu Lasten der Ergebnisse auf dem Testset geschieht.

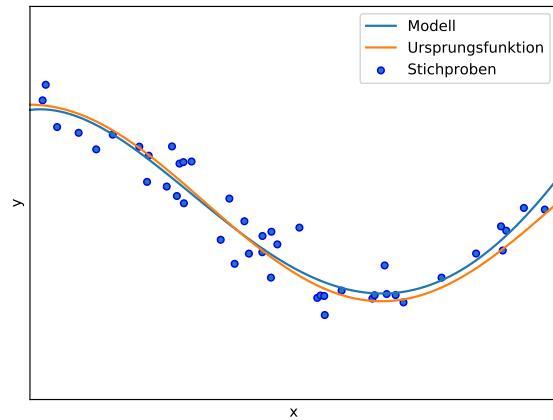
Ist die Grafik
essenziell fürs
Verständnis?

2.2 TableSort System

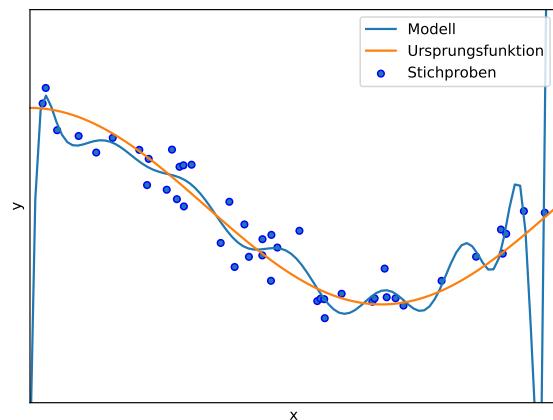
Der *TableSort* Schüttgutsortierer ist eine modulare Versuchsplattform, die konzipiert wurde um neue Schüttgutsortierkonzepte in einem kleineren Rahmen experimentell erproben zu können. Industrielle Schüttgutsortieranlagen sind sehr groß und um etwas an ihrer Konfiguration zu ändern ist sowohl zeitaufwendig als auch arbeitsintensiv, weshalb es sinnvoll ist, eine Plattform zu haben, die flexibel und transportabel ist und leicht umgebaut werden kann. Das System ist im Rahmen des *TrackSort* Projekts in



(a) Underfitting



(b) Passendes Modell



(c) Overfitting

Abbildung 2.5: Visualisierung von Modellen mit unterschiedlichen Kapazitäten.

Kooperation zwischen dem Fraunhofer IOSB, Abteilung Sichtprüfsysteme, und dem Institut für Intelligente Sensor Aktor Systeme des Karlsruher Institut für Technologie entstanden[Dol15]. Es kann in zwei verschiedenen Konfigurationen bezüglich des Schüttguttransports benutzt werden. Einmal werden die Schüttgutpartikel mittels eines Förderbands beschleunigt - diese Konfiguration ist in Abbildung 2.6 zu sehen. Dazu kommt die Möglichkeit dieses Förderband mit einer Rutsche zu ersetzen, sodass die Schüttgutpartikel stattdessen durch die Schwerkraft beschleunigt werden. Im Rahmen dieser Arbeit wurden die Flächenkamera Konfiguration mit einem Ringlicht zur Beleuchtung verwendet. Schematisch ist der Ablauf der Schüttgutsortierung in Abbildung 2.7 zu sehen. Die Funktionsweise des Systems wird im Folgenden erklärt. Das zu sortierende Schüttgut wird durch einen Schwingförderer nach und nach über eine Rutsche auf das Förderband aufgebracht. Dort wird es in Richtung des Druckluftdüsenarrays beschleunigt. Auf das vom Ringlicht beleuchtete Ende des Förderbands ist die Flächenkamera gerichtet. Kurz nachdem die Schüttgutpartikel die Flugphase beginnen passieren sie das Druckluftdüsenarray, wo sie entweder abgelenkt werden, falls die Verarbeitung der Bilddaten von der Kamera sie als auszusortieren klassifiziert.

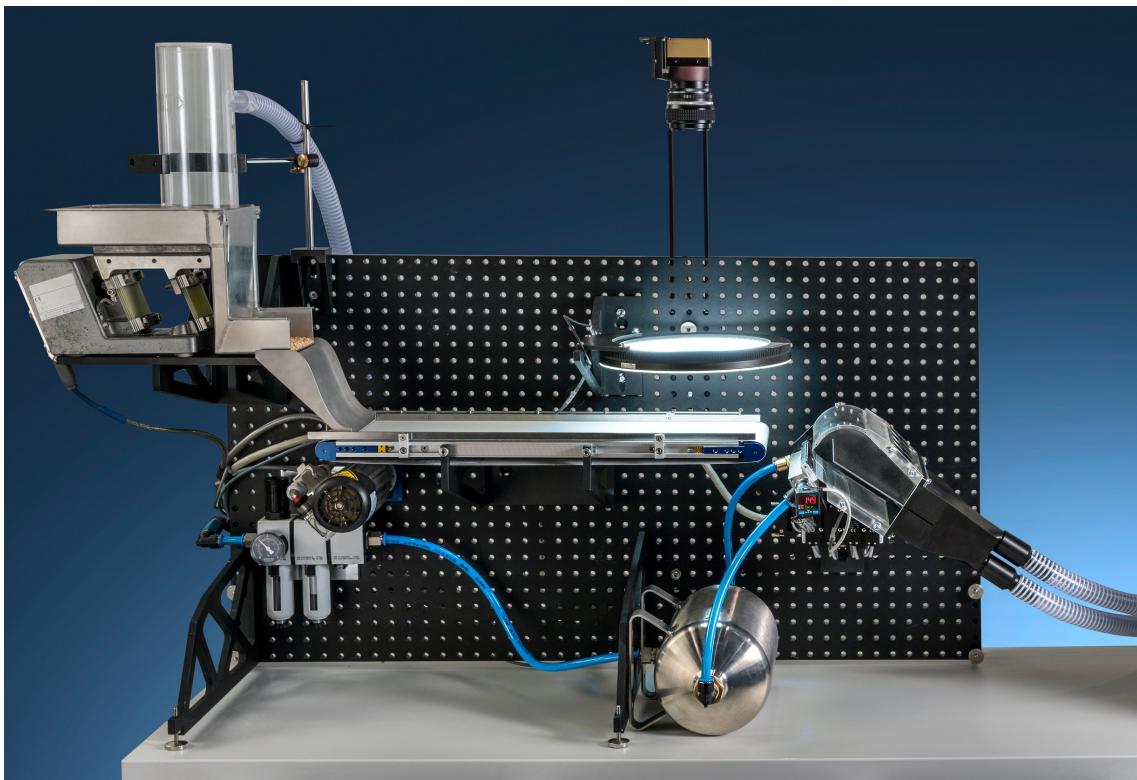


Abbildung 2.6: TableSort Schüttgutsortiersystem [Fra17]

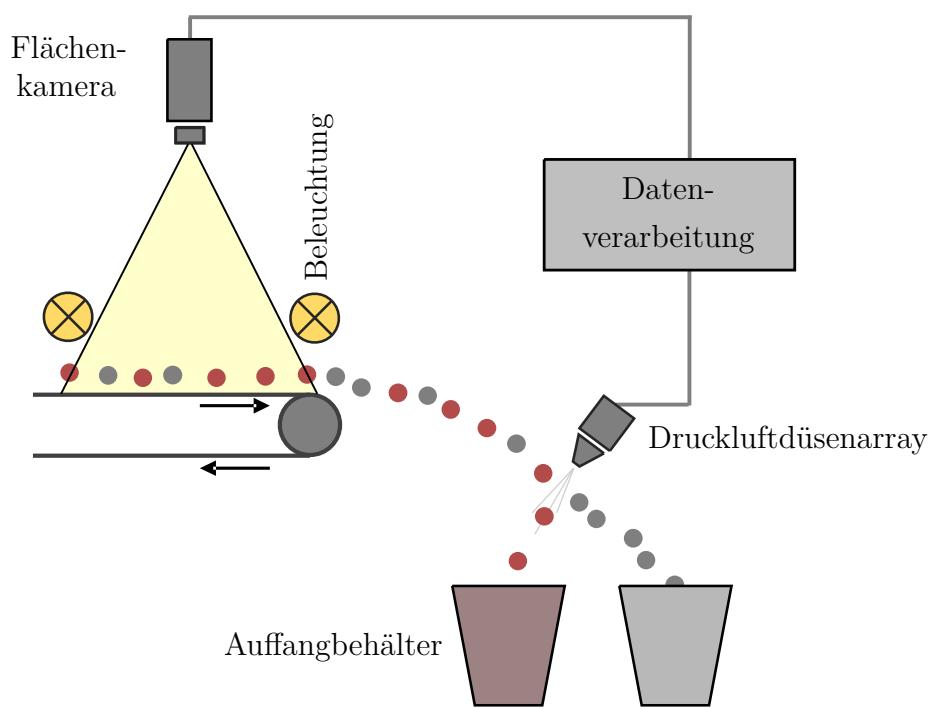


Abbildung 2.7: Schematische Darstellung des optischen Bandsortierers TableSort nach [PKP+17].

2.3 Stand der Technik

Im Rahmen dieser Arbeit geht es um die Prädiktion der Bewegung von Schüttgutpartikeln. Der Einsatz von verschiedenen Bewegungsmodellen ist erst für Schüttgutsortierer mit Flächenkamera sinnvoll. Eine Zeilenkamera liefert nur einen einzelnen Datenpunkt bezüglich Zeit und Position eines Partikels. In [Pfa18] wird unter anderem ein Bewegungsmodell beschrieben, das das Verhalten eines Schüttgutsortierers mit Zeilenkamera emuliert. Wie in schon in Abbildung 1.1 dargestellt wurde, muss angenommen werden, dass es zu keinerlei Bewegung orthogonal zur Transportrichtung kommt. Für die Prädiktion des Zeitpunkts wird die durchschnittliche Zeit bestimmt, die ein Partikel von der Position der Zeilenkamera zur Position des Druckluftdüsenarrays benötigt und diese als konstanter Offset für jede Partikeldetektion angenommen.

Um den Separationsprozess durch den Einsatz von prädiktiven Tracking Methoden und Bewegungsmodellen zu verbessern ist eine Assoziation der beobachteten Partikelpositionen zu tatsächlichen Tracks notwendig. Im Rahmen dieser Arbeit wird dieses Problem nicht betrachtet. Es wird direkt mit den assoziierten Trackdaten gearbeitet, obwohl dieser Assoziationsprozess noch Gegenstand aktueller Forschung ist.

Die grundlegenden Bewegungsmodelle, die in [Pfa18] beschrieben werden sind einerseits das Constant Velocity Modell und andererseits das Constant Acceleration Modell. Das Constant Velocity Modell prädiziert die Bewegung eines Teilchens unter der Annahme, dass es sich mit einer konstanten Geschwindigkeit bewegt. Basierend auf den letzten zwei bekannten Positionen des Partikels wird dessen Geschwindigkeit entlang der beiden Achsen bestimmt und davon die zukünftige Bewegung abgeleitet. Diese Annahme ist jedoch nicht immer korrekt. Es kann sein, dass bei einem Bandsortierer das Förderband nicht lang genug ist, um das Schüttgut komplett zu beruhigen. Dann haben die Teilchen eine Beschleunigung, die nicht 0 ist. Das Constant Acceleration Modell dahingegen prädiziert die Bewegung des Teilchens unter der Annahme, dass es sich mit einer konstanten Beschleunigung bewegt. Diese Beschleunigung wird anhand der letzten drei bekannten Positionen bestimmt. Anhang dieser Beschleunigung und der aktuellen Geschwindigkeit werden die zukünftigen Positionen abgeleitet.

In [Pfa18] werden weitere, szenariospezifische Bewegungsmodelle beschrieben. Bei dem sogenannten Bias-Corrected Constant Velocity Modell wird das Constant Velocity Modell als Grundlage genommen und ein Korrekturterm eingeführt. Basierend auf den zuvor beobachteten Schüttgutpartikeln wird ein durchschnittlicher temporaler Bias bestimmt, der von den zukünftigen Prädiktionen abgezogen wird. Die Annahme ist hier, dass der Bias von zukünftige Partikel ähnlich zu dem der zuvor beobachteten Partikeln sein wird. Beim Identical Acceleration Modell wird ebenfalls

hier schon erwähnen, dass die Assoziation auf den selbst gesammelten Daten vielleicht flawed ist?

ein Korrekturterm benutzt, der im Gegensatz zum Bias-Corrected Constant Velocity Modell jedoch nicht absolut, sondern abhängig von der letzten bekannten Position des Partikels ist. Auf jedem der zuvor beobachteten Partikeln wird der Wert einer zusätzlichen Beschleunigung bestimmt, die zu einer optimalen Prädiktion führen würde und dann der Durchschnitt dieser Beschleunigungen gebildet. Dieser wird dann als Korrekturterm auf ein Constant Velocity Modell addiert, sodass sich eine Formel ähnlich zu der des Constant Acceleration Modelles ergibt. Um das Verhalten von den Partikeln, deren Geschwindigkeit sich der des Förderbands nähert ohne sie zu überschreiten, besser abzubilden als durch ein Constant Acceleration Modell, wird das Constant Acceleration with Limited Velocity Modell beschrieben. Dabei wird die Geschwindigkeit des Förderbands bestimmt und Partikel, die diese erreichen, von dann aus mit konstanter Geschwindigkeit weiter prädiziert. Es wurden zudem zwei Modelle, Constant Acceleration Disallowing Sign Change Modell und Ratio-Based Deceleration Modell, vorgestellt, die spezifisch konzipiert wurden um die Bewegungsprädiktion orthogonal zur Transportrichtung zu verbessern.

KAPITEL 3

Datenverarbeitung

Bei maschinellen Lernverfahren ist die Qualität und die Menge der zugrundeliegenden Daten sehr wichtig, da diese die Güte des Ergebnisses stark beeinflussen können. Im Rahmen dieser Masterarbeit wurden zweierlei Arten von Daten benutzt: Einerseits wurden am *TableSort* Schüttgutsortierer des Fraunhofer IOSBs Aufnahmen gemacht, die dann über mehrere Arbeitsschritte in das richtige Datenformat übersetzt wurden. Zudem existieren Datensätze, die durch eine Simulation mittels der *Diskrete-Elemente-Methode* (DEM) entstanden sind. In diesem Kapitel soll nun beschrieben werden, woher diese Daten kommen, wie sie verarbeitet werden und in welcher Form sie dann letztendlich in die neuronalen Netze gegeben werden. Abschließend wird die Datenaugmentierung, die vorgenommen wurde, vorgestellt und erklärt.

3.1 Eigene Aufnahmen

In diesem Abschnitt soll beschrieben werden, welche Form, die am Fraunhofer IOSB selbst aufgenommenen Daten haben und welche Konfigurationen bezüglich Schüttgut und Sortierer sie darstellen.

Zur Aufnahme der Daten wurde eine Bonito CL-400 200 FPS Kamera benutzt. Diese wurde, wie in Abbildung 2.6 dargestellt, oberhalb des Förderbandes beziehungsweise der Rutsche angebracht. Die Bilder, die von der Kamera aufgenommen werden, haben eine Auflösung von 2320x1726 Pixeln [All14]. Anhand mehrerer Kalibrierungsbilder wurde bestimmt, dass 1 Pixel im Bild ungefähr 0.056 mm entspricht. Im weiteren Verlauf der Arbeit wird mit Pixeln (px) als der Einheit des Orts für die selbst aufgenommenen Bilder gearbeitet.

Aufgenommen wurden Daten von vier verschiedenen Schüttgütern: Kugeln, grüne Pfefferkörner, Zylinder und Weizenkörner. Diese Schüttgüter sind in Abbildung 3.1 in Schüsseln und in Abbildung 3.2 während den Aufnahmen zu sehen.

Die Kugeln bestehen aus Holz und haben einen Durchmesser von 5 mm. Die Zylinder bestehen ebenfalls aus Holz. Sie haben eine Länge von 1 cm und einen Durchmesser von 3 mm. Die Kugeln und der Pfeffer, sowie die Zylinder und die Weizenkörner, bilden jeweils ein Paar aus einem geometrischen Körper und einem echten Objekt, das grob dessen Form ähnelt.

Alle Schüttgüter wurden auf der Förderbandkonfiguration des *TableSort* Systems aufgenommen. Zusätzlich dazu wurden Kugeln und Pfefferkörner auf der Rutschenkonfiguration des *TableSort* Systems aufgenommen.

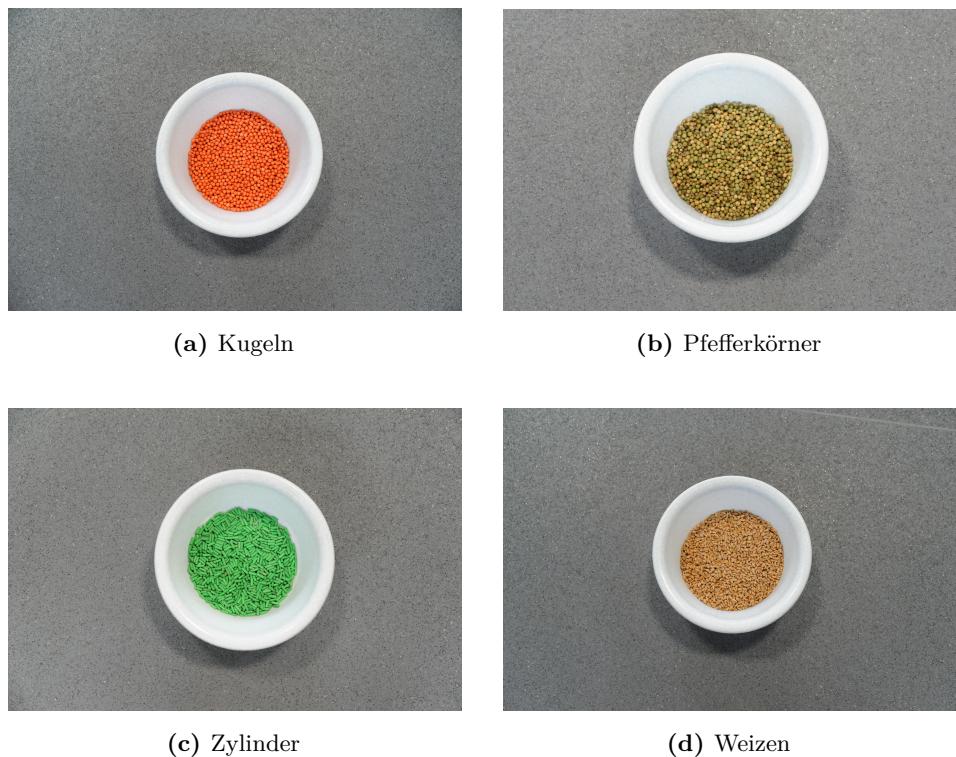


Abbildung 3.1: Verschiedene gesammelte Schüttgüter.

3.2 Datenpipeline

Die Features, die für das Trainieren der Netze benutzt werden, sind die Koordinaten der Mittelpunkte der Partikel. Diese aus den aufgenommenen Bildern zu bestimmen, ist die Aufgabe dieser Pipeline.

Am Anfang davon stehen die Bilder, die die Bonito-Kamera in Form einer Bayer-Matrix aufnimmt, wie sie in Abbildung 3.3 zu sehen ist. Die Bilder werden in Batches von je 3500 Bildern gesammelt und in Bitmap-Dateien geschrieben. Auf Grund der Menge an Bildern und dem damit verbundenen Speicherbedarf wurden die Bilder zunächst in das PNG-Dateiformat übertragen. Danach müssen zunächst die gebayerten

Dateien mittels eines Vorgangs, der als *demosaicing* bezeichnet wird, als Farbbilder rekonstruiert werden. Das geschieht mit einem Skript des Fraunhofer IOSB, das die Open Source Computer Vision Library OpenCV¹ benutzt. Dieses Skript wurde angepasst und damit und dann damit die einzelnen Bilder in RGB-Farbbilder konvertiert. Auf diesen Bildern kann dann eine Segmentierung vorgenommen werden. Dafür wurde ebenfalls ein existierendes Skript des Fraunhofer IOSBs angepasst, in dem erneut die Computer Vision Library OpenCV benutzt wird. Für jede Sorte von Schüttgut wurde ein eigenes Parameterprofil von Hand optimiert. Ein solches Parameterprofil besteht aus einem oberen und unteren Grenzwert in jedem Kanal des HSV-Raums und einer minimalen Fläche, die ein Teilchen umfassen muss. Entsprechend der im Profil festgelegten Parameter werden für die einzelnen Bilder Masken angelegt, die angeben ob die HSV-Werte der einzelnen Pixel innerhalb oder außerhalb der Grenzwerte liegen. Mit diesen Masken werden dann alle möglichen Konturen von Schüttgutpartikeln extrahiert. Die Konturen werden dann bezüglich der Plausibilität ihrer Größe und Form gefiltert. Im letzten Schritt wird nun der gewichtete Mittelpunkt der verbleibenden Konturen bestimmt und abgespeichert.

Das Ergebnis von diesem Segmentierungsskripts ist eine CSV-Datei für jeden Batch. Ein Beispiel für einen Ausschnitt aus solch einer Datei ist in Tabelle 3.1 zu sehen. Eine Zeile repräsentiert jeweils ein Bild aus dem Batch, also einen Zeitschritt. Zu Beginn jeder Zeile steht zunächst die Frame-Nummer, gefolgt von der Anzahl der detektierten Partikel und den Koordinaten der Mittelpunkte der detektierten Partikel. Aus den Mittelpunkten in dieser CSV-Datei werden nun mittels des in MATLAB implementierten *Multi-Target-Tracking*-Algorithmus die in dem Datensatz vorhandenen Tracks abgeleitet, die dann wiederum in einer neuen CSV-Datei gespeichert werden. Die einzelnen Tracks werden als Spaltenpaare dargestellt, mit jeweils einer Spalte für die X- und Y-Koordinaten und einer Zeile für jeden Zeitschritt. Ein Ausschnitt aus einer solchen Datei ist in Tabelle 3.2 zu sehen. Dies ist der Zustand in dem die Daten dann im Programmcode geladen werden.

3.3 Simulierte Daten

Neben den selbst aufgenommenen Daten wurde im Rahmen dieser Arbeit auch mit einigen existierenden Datensätzen gearbeitet. In [PKEW+16] und [PPM+17] wird dargelegt wie diese Datensätze basierend auf einer hochgenauen numerischen Simulation des *TableSort* Systems mittels der Diskrete-Elemente-Methode erstellt wurden. In Abbildung 3.4 ist die virtuelle Nachbildung des Schüttgutsortierers zu sehen, auf dem die Simulation durchgeführt wurde. Die Positionsdaten in den

¹ <https://opencv.org/>

Tabelle 3.1: Ausschnitt aus dem Ergebnis des Segmentierungsskripts

Frame	#MP	MP_1_x	MP_1_y	MP_2_x	MP_2_y	MP_3_x	MP_3_y
636	1	1222.9975	92.7641	NaN	NaN	NaN	NaN
637	1	1223.4063	182.9758	NaN	NaN	NaN	NaN
638	1	1223.6052	273.2425	NaN	NaN	NaN	NaN
639	1	1223.7067	364.0339	NaN	NaN	NaN	NaN
640	1	1224.0704	453.9057	NaN	NaN	NaN	NaN
641	2	1224.2051	544.5191	1692.4549	43.8822	NaN	NaN
642	2	1224.5793	634.7288	1696.6901	135.9595	NaN	NaN
643	2	1224.9082	726.0094	1700.451	229.1195	NaN	NaN
644	2	1225.2296	815.9663	1704.1472	321.2075	NaN	NaN
645	2	1225.4286	906.7078	1708.0593	414.2785	NaN	NaN
646	2	1225.7588	996.0286	1711.5309	506.0545	NaN	NaN
647	3	1226.0411	1086.5729	1714.8831	599.5417	961.8821	62.7111
648	3	1226.2337	1175.9271	1718.1401	691.6325	958.5526	154.3124
649	3	1226.2073	1265.7495	1721.6618	784.5927	955.3107	246.5241
650	3	1226.2543	1354.9362	1724.9158	876.7192	952.4919	338.1123
651	3	1226.2634	1444.5903	1728.3341	970.2909	949.2896	430.9692
652	3	1226.0845	1533.0901	1732.1745	1062.4624	946.3455	522.8667
653	3	1225.7319	1621.8461	1735.8759	1155.2937	943.3384	615.4545
654	2	1739.6714	1247.1867	940.2511	707.7306	NaN	NaN
655	2	1743.4279	1339.4146	937.2216	800.4557	NaN	NaN
656	2	1747.1525	1430.2501	934.5311	891.7249	NaN	NaN
657	2	1750.9771	1521.8102	931.6626	984.2284	NaN	NaN
658	2	1754.1491	1612.5565	928.7587	1076.4749	NaN	NaN
659	1	925.8463	1168.794	NaN	NaN	NaN	NaN
660	1	922.8752	1260.7461	NaN	NaN	NaN	NaN
661	1	920.2056	1352.8549	NaN	NaN	NaN	NaN
662	1	917.4051	1444.3431	NaN	NaN	NaN	NaN
663	1	914.6493	1535.5131	NaN	NaN	NaN	NaN
664	1	911.8565	1626.5341	NaN	NaN	NaN	NaN

Tabelle 3.2: Ausschnitt aus dem Ergebnis des *Multi-Target-Tracking*-Algorithmus

TrackID_4_X	TrackID_4_Y	TrackID_5_X	TrackID_5_Y	TrackID_6_X	TrackID_6_Y
1036.4613	82.3719	1899.9239	83.2049	1654.4423	50.6811
1033.0189	174.9809	1896.8142	171.3283	1655.3193	143.9749
1029.6167	266.4979	1893.5937	259.8098	1656.0221	237.1573
1026.3908	358.4831	1890.3912	348.1731	1656.8966	329.8636
1023.0203	449.6429	1887.1035	436.4588	1657.6308	423.1592
1019.5391	542.2334	1883.7761	525.1073	NaN	NaN
NaN	NaN	1880.2716	613.0896	NaN	NaN
NaN	NaN	1876.6054	701.9719	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN

Datensätzen waren ursprünglich mit einer Frequenz von 1000 Hz aufgelistet. Um jedoch Vergleichbarkeit mit den realen Daten zu erhalten wurde diese Frequenz auf 200 Hz reduziert. Die Positionsangaben der Simulation werden in Metern angeben. Das Förderband erstreckt sich entlang der y-Achse von 0.0 m bis 0.18 m und entlang der x-Achse von 0.388 m bis 0.788 m

Dennoch gibt es einige Unterschiede zwischen den selbst aufgenommenen und den simulierten Daten. Von diesen Unterschieden ist der schwerwiegendste, dass die Positionen der simulierten Partikel eine absolut verlässliche Ground Truth sind, die direkt aus Simulation entnommen wurde. So verlässlich sind die Partikelpositionen bei den selbst aufgenommenen Daten nicht. Während den einzelnen in Sektion 3.2 beschrieben Schritten kann es zu Fehlern kommen, die sich durch die gesamte Pipeline fortpflanzen. Die Segmentierung kann an einigen Stellen nicht präzise sein, indem z. B. ein Stück Schattens als Teil des Partikels interpretiert wird und dadurch den Mittelpunkt verschiebt. Bei einer Kollision von zwei Partikeln kann es dazu kommen, dass der *Multi-Target-Tracking*-Algorithmus die beiden Tracks vertauscht. Des Weiteren bewegt sich das Förderband in der Simulation mit 1.5 m s^{-1} , während Messungen am realen *TableSort* System darauf hindeuten, dass das Förderband dort nur eine Geschwindigkeit von circa 1.1 m s^{-1} hat. Ein weiterer Unterschied ist der Aufnahmebereich. Im Gegensatz zu den selbst aufgenommenen Daten, die nur Informationen zu den Bewegungen der Partikel im Bereich, auf den die Hochgeschwindigkeitskamera gerichtet ist umfassen, befinden sich in den DEM-Datensätzen die Positionen der Teilchen über die gesamte Länge des Förderbandes. Das bedeutet auch, dass in den DEM-Datensätzen die Phase der Partikel bevor und während sie vom Förderband beruhigt werden enthalten ist. Außerdem hat es zur Folge, dass die individuellen Tracks deutlich länger sind, sprich mehr Messungen enthalten. Bei der NextStep-Prädiktion sorgt das dafür, dass es eine deutlich größere Anzahl an Feature-Label-Paaren gibt. Die Bewegungsrichtung der Partikel ist ebenfalls nicht identisch. Während die selbst aufgenommenen Datensätze eine Bewegung entlang der y-Achse haben, bewegen sich die simulierten Partikel entlang der x-Achse.

simulierte haben viel mehr Partikel gleichzeitig auf dem Band - mehr kollisionen...?

3.4 Datenformatierung

Das Format, das die Daten annehmen, ist für die NextStep- und den Separator-Netze leicht unterschiedlich. Die Feature-Label-Paare der unterschiedlichen Anwendungen unterscheiden sich nur in den Labels. Die Eingangsdaten sind – abhängig von einem

Hyperparameter – in beiden Fällen die Position des Partikels zu den letzten n Zeitschritten. Deshalb muss beim Format der Features kein Unterschied zwischen der Art der Aufgabe gemacht werden. Dieser Hyperparameter wird *FeatureSize* genannt. Die Features sind also ein $2n$ -Tupel, bestehend aus n X-Koordinaten und n Y-Koordinaten von den Mittelpunkten der n beobachteten Teilchen. Die Reihenfolge der Features ist für das neuronale Netz irrelevant, solange sie konsistent zwischen Training und Evaluation bleibt. Was die Reihenfolge der Features angeht, so ist sie für das neuronale Netz irrelevant, solange sie konsistent zwischen Training und Evaluation bleibt. In der umgesetzten Implementierung ist es so, dass zuerst die X-Koordinaten und dann die Y-Koordinaten in chronologischer Reihenfolge aufgereiht sind.

Die Labels, die das NextStep-Netz benutzt, sind 2-Tupel. Diese bestehen aus den X- und Y-Koordinaten des Partikels im nächsten Zeitschritt. Wie in Abbildung 3.5 dargestellt, befinden diese sich in der nächsten Zeile nach den Features.

Die Labels des Separator-Netzes sind ebenfalls 2-Tupel. Im Gegensatz zu denen des NextStep-Netzes können diese nicht direkt aus der CSV-Datei ausgelesen werden, da sie nicht gemessen werden. Stattdessen müssen sie berechnet werden. Dies ist in Abbildung 3.6 dargestellt. Für jedes Partikel müssen die letzte Position vor dem Überqueren des Druckluftdüsenarrays f_n und die erste Position nach dem Überqueren f_{n+1} bestimmt werden. In Abbildung 3.7 sind dies die blau markierten Einträge. Dabei wird die angenommen, dass die Geschwindigkeit des Partikels zwischen f_n und f_{n+1} konstant ist. Dies ist eine Approximation, die ebenfalls in [Pfa18] verwendet wird. Das erste Element eines jeden Labels ist die Koordinate entlang der Achse orthogonal zur Bewegungsrichtung des Förderbandes, wo das Partikel den Druckluftdüsenarray

Bewegungsrichtung passiert. Diese erhalten wir, indem wir den Schnittpunkt s zwischen der Strecke f_n nach f_{n+1} und der Gerade des Druckluftdüsenarray bestimmen. Das zweite Element ist die Zeit, die das Partikel noch brauchen wird, bis es das Druckluftdüsenarray passiert. Sie wird in der Einheit Frames angegeben. Die ganzzahlige Komponente hiervon ist durch das Zählen der Messungen im Track zu bestimmen. Die Nachkommastelle wird bestimmt als das Verhältnis von der Distanz zwischen f_n und s , und der Distanz zwischen f_n und f_{n+1} .

Damit die Fehler der beiden Labelelemente sich ähnlich auf den Gesamtgradienten auswirken, selbst wenn ihre Werte in sehr unterschiedliche Wertebereichen liegen oder einfach unterschiedliche Einheiten haben, werden die Labels komponentenweise standardisiert. Dieses Problem würde zum Beispiel auftreten, wenn sich die Werte des einen Elements im Bereich zwischen 0 und 1700 bewegen, während die des Anderen zwischen 1 und 25 liegen, so wie es bei den Separator-Netzen der Fall sein kann. Bei dem NextStep-Fall wäre die Standardisierung nicht zwingend

notwendig, sie wurde allerdings, um die Einheitlichkeit zu wahren, für beide Fälle implementiert. Für die Standardisierung wird von der jeweiligen Komponente der Mittelwert und die Standardabweichung berechnet. Im Anschluss wird vom jeweiligen Eintrag X der zugehörige Mittelwert μ subtrahiert, und durch die korrespondierende Standardabweichung σ dividiert.

$$X_{\text{neu}} = \frac{X - \mu}{\sigma}$$

Durch die Standardisierung wird jedes Label so skaliert, dass die beiden Elemente der Labels jeweils einen Erwartungswert von 0.0 und eine Standardabweichung von 1.0 haben. Die Ausgaben des Netzes müssen umgekehrt zurück in das ursprüngliche Format umgerechnet werden.

Im Code gibt es die Möglichkeit, unterschiedliche Label-Elemente unterschiedlich zu gewichten, indem Weighted Mean Squared Error als Fehlerfunktion verwendet werden kann. Von dieser Möglichkeit wird in den später vorgestellten Ergebnissen nicht Gebrauch gemacht, beziehungsweise den beiden Elementen wird jeweils das Gewicht 1.0 zugewiesen. Diese Option könnte nützlich sein, falls die Ansteuerung der Druckluftdüsen eine maximale Auflösung hat, jenseits derer eine bessere Zeitprädiktion keine Verbesserung der Sortierqualität mehr erzielt.

3.5 Datenaugmentierung

Als Datenaugmentierung oder auch Data Augmentation bezeichnet man Verfahren, die dem eigenen Datenset Datenpunkte hinzufügen ohne zusätzliche Daten aufzunehmen [GBC16, Kapitel 7.4]. Man generiert aus den bestehenden Daten zusätzliche, synthetische Daten, die dann im Trainingsset eingesetzt werden können. Ausreichend viele Trainingsbeispiele zu haben ist notwendig, um mit neuronalen Netzen eine gute Performance zu erzielen. Die synthetischen Beispiele müssen jedoch konsistent mit den Originaldaten sein, da sie sonst die Qualität der Ausgabe des Netzes negativ beeinträchtigen können.

Für Netze, die in der Computer Vision eingesetzt werden gibt es einige weit verbreitete Techniken, zum Beispiel Rotation, Translation, Spiegeln und das Ausschneiden von Teildingen.

Für den gegebenen Fall mit den Mittelpunkten von Schüttgut-Partikeln als Features resultiert von diesen Techniken nur das Spiegeln in sinnvollen Daten. Eine beispielhafte Darstellung der vorgenommenen Data Augmentation ist in Abbildung 3.8 zu finden. Gespiegelt wird an der Mittellinie des Bands, beziehungsweise der Rutsche

entlang der Bewegungsrichtung. Tracks, die eine gewisse Distanz zur Spiegelachse überschreiten, werden ausgenommen, da zumindest bei den selbst aufgenommenen Daten die Kamera nicht perfekt zentriert ist. Deshalb könnte nicht ausgeschlossen werden, dass unrealistische Feature-Label-Paare dem Trainingsset hinzugefügt werden würden.

Die Anwendung dieser Technik auf die vorhandenen Daten führt beinahe zu einer Verdoppelung der benutzbaren Feature-Label-Paare im Trainingsset, wovon man sich einen positiven Effekt auf die Qualität der Ergebnisse verspricht.

3.6 Datenumfang

Im Rahmen dieser Arbeit wurden insgesamt 247 951 Bilder aufgenommen. Davon waren 177 951 Bilder auf dem Förderband und 70 000 Bilder auf der Rutsche. Die Verteilung dieser Bilder ist in Abbildung 3.9 dargestellt.

Der *Multi-Target-Tracking*-Algorithmus wurde benutzt, um die Anzahl der Kugeln zu bestimmen. Auf dem Förderband wurden 7712 Tracks von Kugeln, 7170 Tracks Pfefferkörnern, 19 200 Tracks von Zylindern, und 8702 Tracks von Weizenkörnern zugeordnet. Zudem wurden 5132 Tracks Kugeln und 3609 Tracks von Pfefferkörnern auf der Rutsche zugeordnet.

Im DEM-Datensatz sind die Tracks von 3713 Kugeln und 4357 Plättchen enthalten. Die Verteilung der Tracks ist in Abbildung 3.10 zu sehen.

Die Feature-Label-Paare, die aus den Tracks extrahiert werden, müssen in ein Trainingsset und ein Testset aufgeteilt werden. In manchen Situationen wird noch ein drittes Set, das so genannte Validationset, benötigt. Dies ist zum Beispiel beim Hyperparameter Tuning der Fall.

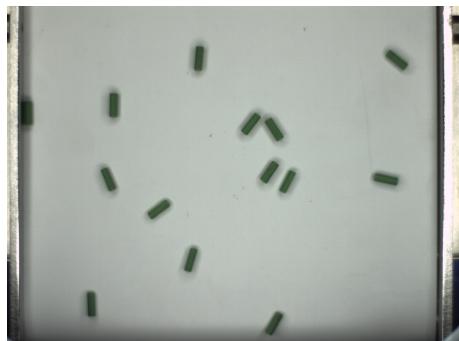
Hierbei ist wichtig, dass das Testset eine gute Repräsentation des gesamten Datensets ist und ausreichend groß ist, um statistisch aussagekräftige Ergebnisse zu produzieren. Im Rahmen dieser Arbeit wurde meist eine Aufteilung im Verhältnis 90:10 zwischen Trainings- und Testset vorgenommen. Es ist jedoch möglich, beliebige andere Splits in der Hyperparameter-Datei festzulegen.



(a) Kugeln auf dem Förderband



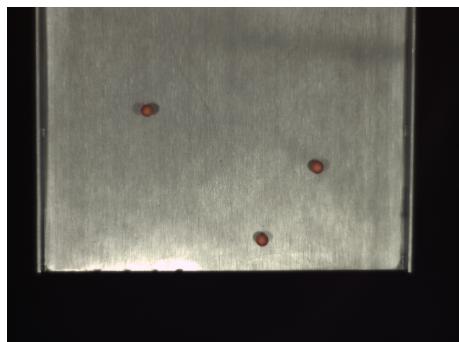
(b) Pfefferkörner auf dem Förderband



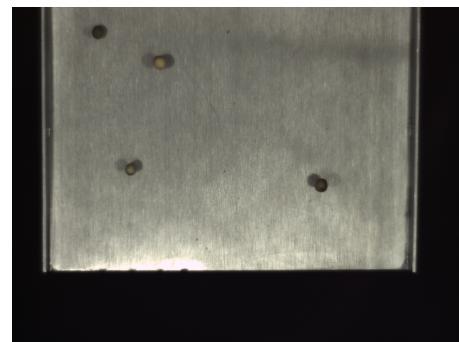
(c) Zylinder auf dem Förderband



(d) Weizen auf dem Förderband



(e) Kugeln auf der Rutsche



(f) Pfefferkörner auf der Rutsche

Abbildung 3.2: Verschiedene Schüttgüter auf dem Förderband bzw. auf der Rutsche.

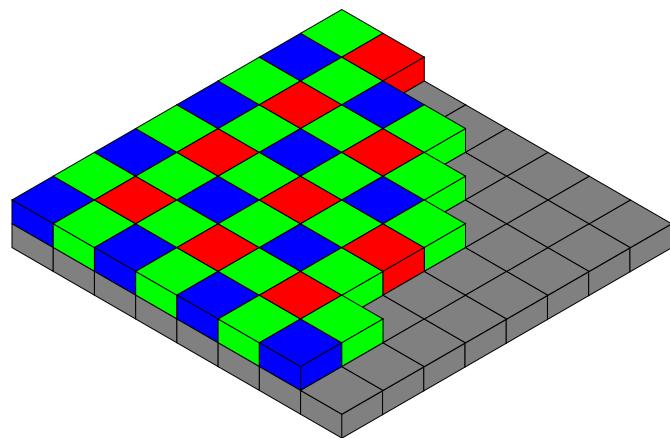


Abbildung 3.3: Bayer-Matrix auf einem Sensor [WC06].

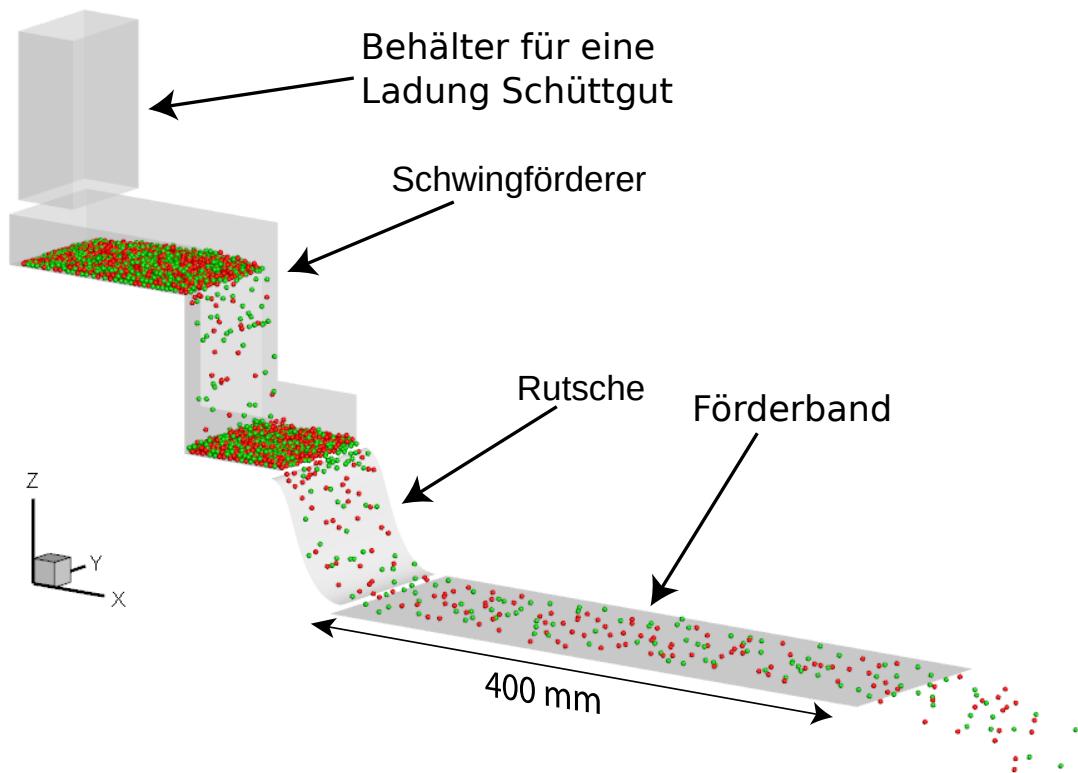


Abbildung 3.4: Visualisierung der DEM-Simulation, übersetzt aus [Pfa18].

TrackID 4 X	TrackID 4 Y
1036.4613	82.3719
1033.0189	174.9809
1029.6167	266.4979
1026.3908	358.483
1023.0203	449.6429
1019.5391	542.2334
1016.1206	633.1533
1012.8021	725.3765
1009.4512	816.8757
1006.0784	909.1367
1002.6553	1000.4947
999.3077	1092.9584
996.0751	1183.6545
992.8973	1274.7595
989.7239	1365.358
986.3243	1456.6952
982.8463	1546.6372
979.6845	1637.0949
NaN	NaN

Abbildung 3.5: An diesem Ausschnitt eines Tracks wird beispielhaft mit FeatureSize 5 gezeigt, wie die einzelnen Feature-Label-Paare im NextStep-Fall entstehen. Der rote Rahmen markiert die 10 Features. Der blaue Rahmen markiert das dazugehörigen Label.

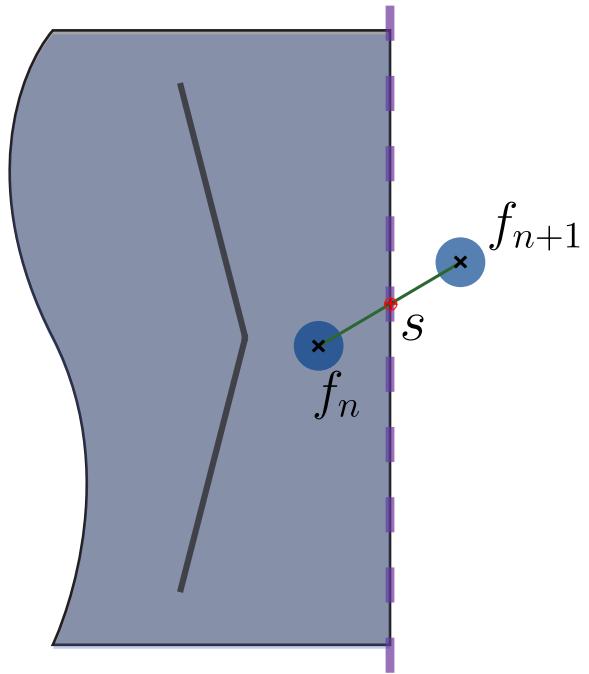


Abbildung 3.6: Geometrische Bestimmung der Labels, nach [Pfa18]. f_n ist die Position des Partikels im Zeitschritt n , dem letzten bevor es das Druckluftdüsenarray passiert. f_{n+1} ist die Position des Partikels im Zeitschritt $n + 1$, dem ersten nach der Passage. s ist der Punkt an dem das Partikeln das Druckluftdüsenarray passiert.

TrackID 4 X	TrackID 4 Y
1036.4613	82.3719
1033.0189	174.9809
1029.6167	266.4979
1026.3908	358.483
1023.0203	449.6429
1019.5391	542.2334
1016.1206	633.1533
1012.8021	725.3765
1009.4512	816.8757
1006.0784	909.1367
1002.6553	1000.4947
999.3077	1092.9584
996.0751	1183.6545
992.8973	1274.7595
989.7239	1365.358
986.3243	1456.6952
982.8463	1546.6372
979.6845	1637.0949
NaN	NaN

Abbildung 3.7: An diesem Ausschnitt eines Tracks wird beispielhaft mit FeatureSize 5 gezeigt, wie die einzelnen Feature-Label-Paare im Separator-Fall entstehen. Analog zu Abbildung 3.5 ist ein Featurevektor markiert. Der Unterschied besteht in der Bestimmung der Labels. Der Schnittpunkt mit dem Druckluftdüsenarray befindet sich zwischen den beiden blau markierten Messungen und kann dementsprechend nicht direkt abgelesen werden.

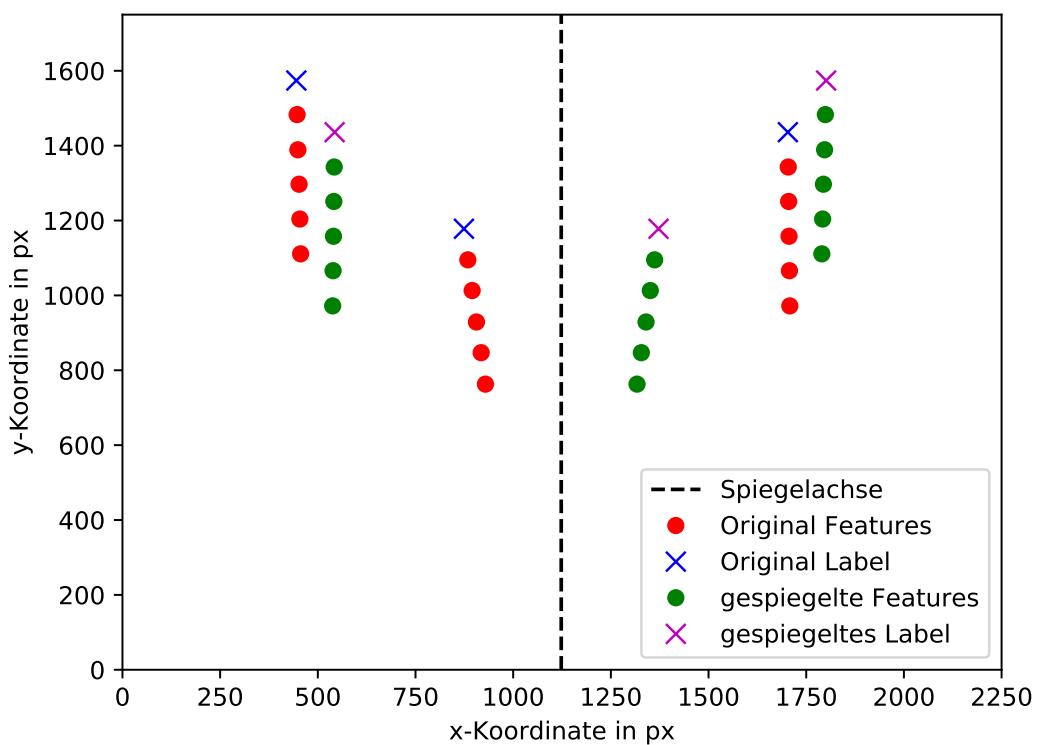


Abbildung 3.8: Visualisierung der Datenaugmentierung durch Spiegelung.

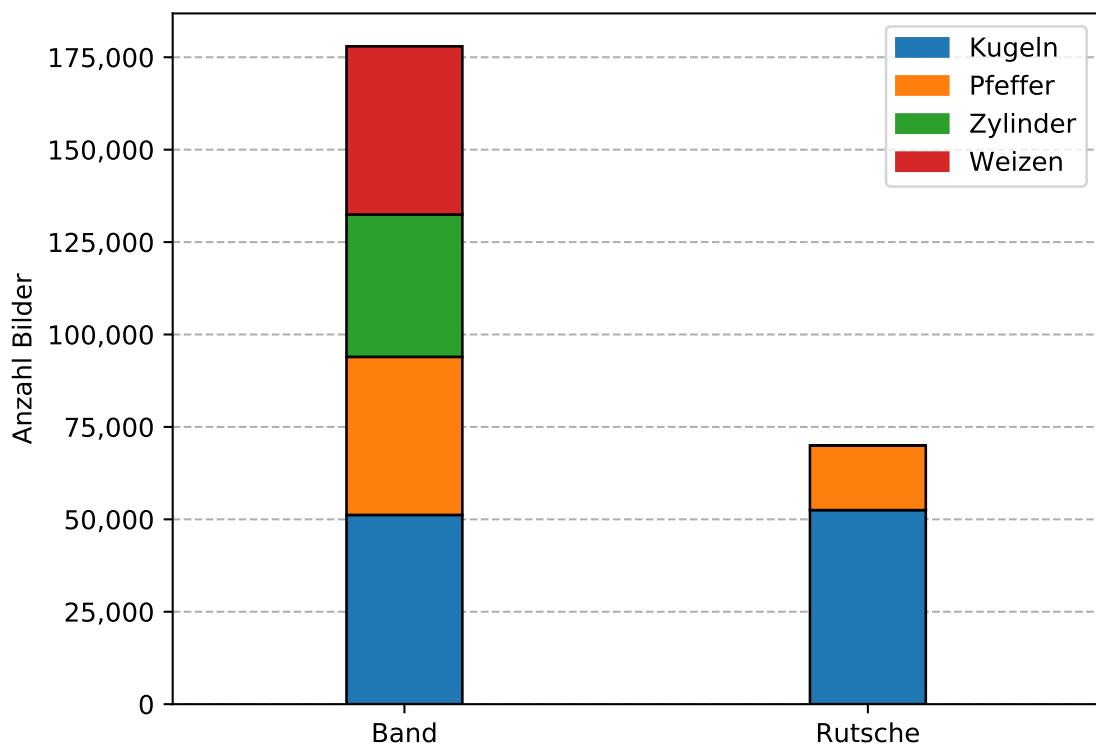


Abbildung 3.9: Balkendiagramm Verteilung Bilder.

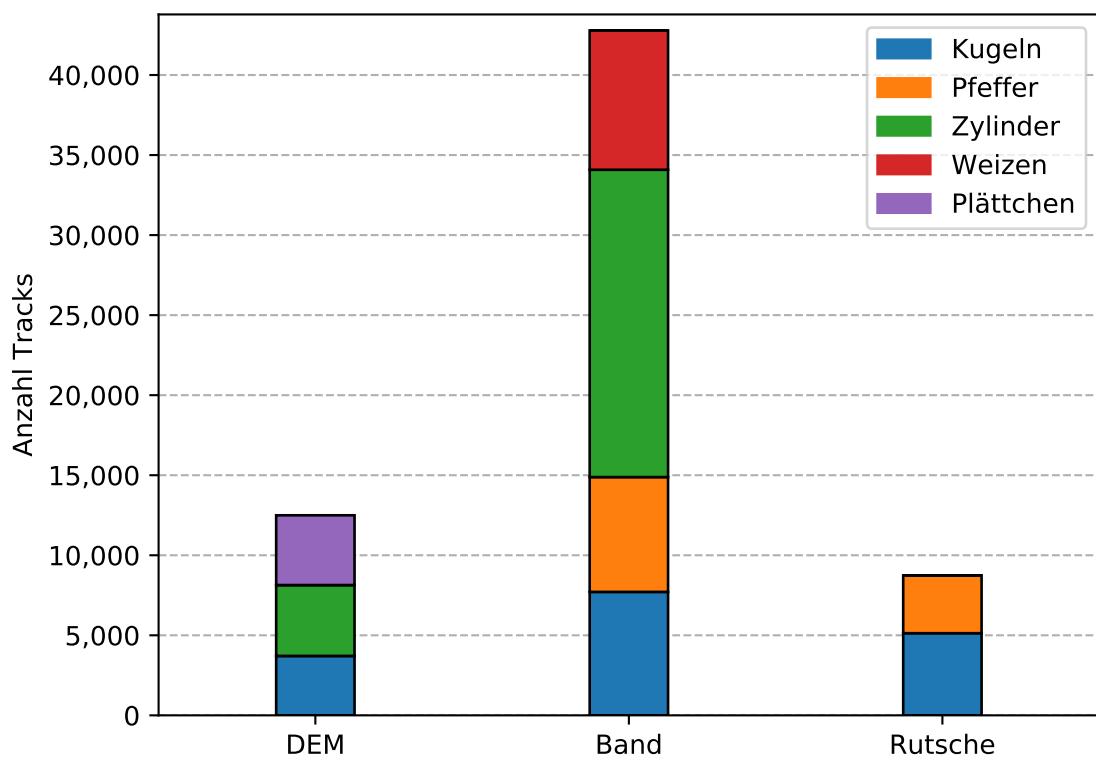


Abbildung 3.10: Balkendiagramm Verteilung Tracks.

KAPITEL 4

Umsetzung und Implementierung

In diesem Kapitel wird beschrieben, wie die neuronalen Netze, die für die Schüttgutprädiktion verwendet werden sollen, umgesetzt wurden. Dafür wird auf die verwendete Software eingegangen und die Struktur der Implementierung anhand eines Flussdiagramms betrachtet. Am Ende des Kapitels wird ein genauerer Blick auf die verwendeten Hyperparameter und deren Optimierung geworfen.

4.1 Software

Für die Umsetzung der neuronalen Netze wurde im Rahmen dieser Arbeit das Open Source Framework TensorFlow¹ verwendet. Es wurde die tensorflow-gpu Variante mittels pip in einer Python 3.6.5 virtuellen Umgebung installiert. Zu Beginn der Arbeit wurde die zum damaligen Zeitpunkt aktuelle TensorFlow Version 1.8 installiert, die zu einem späteren Zeitpunkt durch die Version 1.11 ersetzt wurde. Für den Umgang mit den Daten wurde die Python-Bibliothek *pandas* in der Version 0.23.1 genutzt. Weitere wichtige Software, die verwendet wurde, ist die Python-Bibliothek *matplotlib* in der Version 3.0.0, mit dem die Visualisierung der Ergebnisse und die meisten Grafiken dieser Arbeit generiert wurden. Des Weiteren wurden die Python-Bindings von *OpenCV* in der Version 3.4.1.15 für die Verarbeitung der aufgenommen Bilder verwendet. Der existierende *Multi-Target-Tracking*-Algorithmus ist in MATLAB implementiert worden. Deshalb sind Teile des Projekts, die direkt darauf aufbauen, ebenfalls in MATLAB geschrieben.

4.2 Code-Struktur

Die grundlegende Struktur des Codes, der für das Training verantwortlich ist, ist in Abbildung 4.1 dargestellt. Aus der Hyperparameter-Datei werden die Hyperpara-

¹ <https://www.tensorflow.org/>

meter geladen und basierend darauf der TensorFlow Custom Estimator initialisiert. Außerdem werden die Daten aus den entsprechenden CSV-Dateien geladen und daraus jeweils ein *pandas DataFrame* mit den Feature-Label-Paaren des Trainingsssets und des Testsets generiert. In der Trainingssschleife wird dann ein Modell mit dem Trainingsset trainiert. Dies resultiert in dem trainierten Modell, das nun für die Prädiktion genutzt werden kann.

Der Ablauf der Evaluation ist in Abbildung 4.2 dargestellt. Um das aus dem Training resultierende Modell auszuwerten, werden die Feature-Label-Paare des Testsets benutzt. Dem Modell werden die Features als Eingabe gegeben, um die Prädiktionen des Netzes zu erhalten. Am Ende steht als Ergebnis ein einzelnes *pandas DataFrame* in dem alle Features, das zurücktransformierte Label, die Prädiktionen des Netzes und die Prädiktionen der verschiedenen Vergleichsmodelle jeweils für alle Feature-Label-Paare des Testsets enthalten sind. Hinzu kommen noch die Fehler all dieser Prädiktionen, also die Differenz zwischen der Ground Truth, die im Label steht, und den Prädiktionen selbst. Diese Fehler werden dann analysiert. Es werden Durchschnitt, Median und Quartile gebildet, welche dann mittels Boxplots und Histogrammen visualisiert werden. Diese Diagramme sind in Kapitel 5 zu finden.

Tabelle mit inhalt von Dataframe?

4.3 Hyperparameter

Im Kontext des maschinellen Lernens bezeichnet man jene Variablen als Hyperparameter, die bereits vor dem Beginn des Trainings festgelegt sein müssen. Beispiele für solche Hyperparameter sind unter anderem die Netzarchitektur, wie die Anzahl der Layers und ihre jeweilige Größe oder die *FeatureSize*. Dieser Parameter legt fest, welche Form der Input des Netzes annimmt. Auch Parameter, die festlegen wie das Netz lernt – also Lernrate oder die Anzahl der Epochen beim Training – zählen dazu.

Bei der Implementierung dieser Arbeit wurde die Definition der Hyperparameter aus dem Programmcode selbst ausgelagert. Es können sogenannte Hyperparameter-Dateien im JSON-Dateiformat definiert werden, die dann vom Programm geladen werden. Das vereinfacht das Training und das Evaluieren von Netzen mit unterschiedlichen Hyperparameter-Konfigurationen. Einzelne Parameter aus den Dateien können per Kommandozeilenparameter überschrieben werden. Ein Beispiel für eine solche Hyperparameter-Datei ist in Listing 4.1 zu sehen. Die Bedeutung der einzelnen Parameter wird im Folgenden erklärt. Sollte es für einen Parameter eine begrenzte Menge an zulässigen Optionen geben, so können die entsprechenden Strings aus dem Begleitdokument entnommen werden.

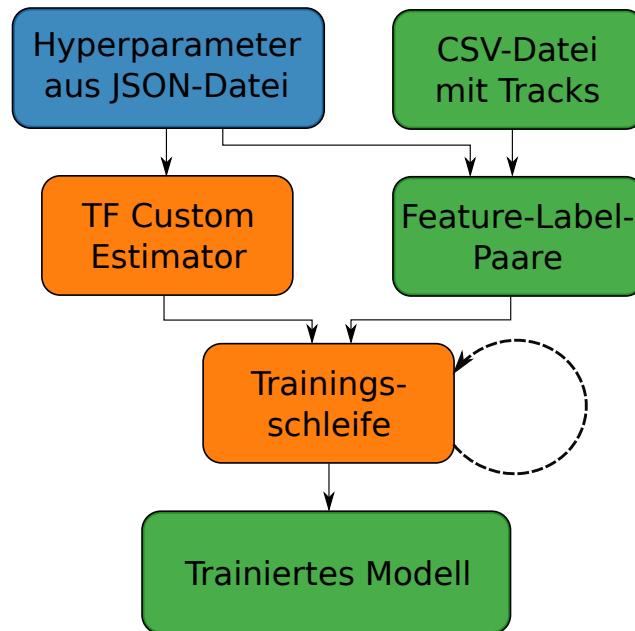


Abbildung 4.1: Skizziert Trainingsablauf eines neuronalen Netzes.

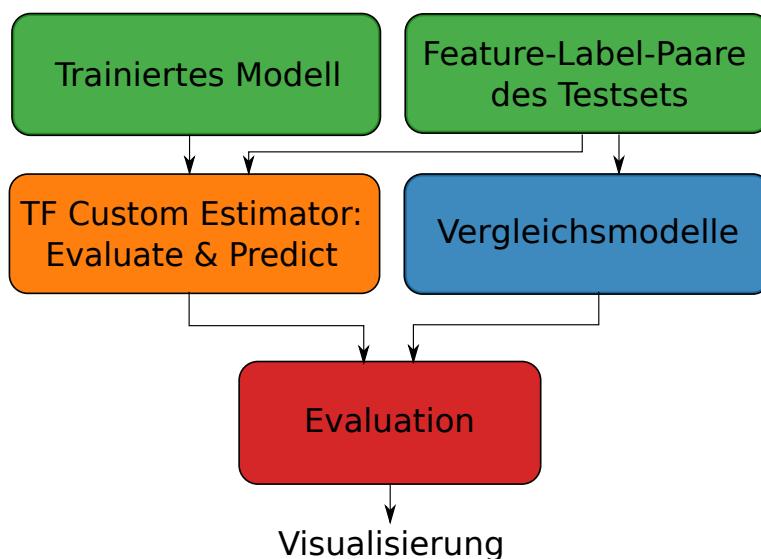


Abbildung 4.2: Skizzierte Ablauf der Evaluation eines neuronalen Netzes.

```
1
2      {
3          "arch": {
4              "dropoutRate": 0.0,
5              "hiddenLayers": [16, 16, 16],
6              "featureSize": 5,
7              "activation": "leaky_relu",
8              "regularization": "no"
9          },
10         "problem": {
11             "dataPath": "/home/hornberger/MasterarbeitTobias/data/
12                         selbstgesammelteDaten2/Kugeln-Band-Juli/",
13             "modelBasePath": "/home/hornberger/MasterarbeitTobias/
14                         models/real/NextStep/NextStep-SpheresReal",
15             "imagePath": "/home/hornberger/MasterarbeitTobias/images/
16                         RealSpheres-NextStep-45kDecaySteps",
17             "separator": 0,
18             "separatorPosition": 1550,
19             "thresholdPoint": 1200,
20             "predictionCutOff": 1500
21
22         },
23         "train": {
24             "batchSize": 500,
25             "epochs": 3000,
26             "stepsPerEpoch": 5000,
27             "learningRate": 0.005,
28             "decaySteps": 45000,
29             "optimizer": "Adam"
30         },
31         "data": {
32             "testSize": 0.1,
33             "augmentMidpoint": 1123,
34             "augmentRange": 1000,
35             "direction": "x",
36             "unitLoc": "px",
37             "unitTime": "1/100 Frames",
38             "limits": [0, 2350, 0, 1750]
39         }
40     }
```

Listing 4.1: Beispiel einer Hyperparameter-Datei in JSON.

arch

dropoutRate	Wahrscheinlichkeit für das zufällige Ausschalten von einzelnen Neuronen beim Training, entsprechend der Beschreibung in Unterabschnitt 2.1.5.
hiddenLayers	Ein eindimensionales Array an Zahlen, das die Architektur der Hidden Layers repräsentiert. Jede Zahl repräsentiert ein Fully-Connected Layer mit der entsprechenden Anzahl Neuronen.
featureSize	Anzahl der letzten Positionen des Partikels, die das Netz als Eingabe bekommt. Wie in Kapitel 3 beschrieben.
activation	Legt die Aktivierungsfunktionen für die Neuronen der Hidden Layers fest. Valide Optionen sind ReLU, Leaky ReLU und lineare Aktivierungsfunktionen.

problem

dataPath	Ort im Dateisystem, wo die CSV-Dateien liegen, in denen die Tracks gespeichert sind.
modelBasePath	Ort im Dateisystem, an dem das Modell des Netzes und alle damit verbundenen Dateien gespeichert wird, beziehungsweise woher es geladen wird.
imagePath	Ort im Dateisystem, wo Schaubilder, Visualisierungen oder Ähnliches gespeichert werden sollen.
separator	Hyperparameter, der festlegt, für welche Art von Aufgabe das Netz trainiert werden soll. 0 bedeutet es wird ein NextStep-Netz trainiert. 1 bedeutet es wird ein Separator-Netz trainiert.
separatorPosition	Koordinate des Düsenbalken und Ziel der Prädiktion entlang der Bewegungsrichtung des Förderbands.
thresholdPoint	Koordinate des Punkts, nach dem von der nächsten Beobachtung prädiziert wird.
predictionCutOff	Koordinate des Punkts, ab dem die Tracks beim Laden abgeschnitten werden.

train

batchSize	Die Anzahl der Feature-Label-Paare, die in einer Iteration betrachtet werden.
epochs	Die Anzahl der Epochen nach denen das Trainings beendet wird.
stepsPerEpoch	Die Anzahl der Iterationen, die in einer Epoche durchgeführt werden.
learningRate	Anfängliche maximale Lernrate, mit der die Gewichte geändert werden
decaySteps	Parameter, der festlegt, wie schnell die maximale Lernrate reduziert wird. Wie dieser Parameter verwendet wird, wird im weiteren Kapitel erklärt.
optimizer	Der Optimierer, der fürs Training verwendet wird. Zulässig sind Adams, Adagrad und Gradient Descent

data

testSize	Parameter, der die Aufteilung zwischen Trainings- und Testdaten festlegt. Ein Wert von 0.1 würde einen 90:10 Split Training zu Test bedeuten.
augmentMidpoint	Koordinate der Spiegelachse, an der für die Data Augmentation gespiegelt wird.
augmentRange	Maximale Distanz eines Feature-Label-Paars zur Spiegelachse, bis wohin die Spiegelung angewendet werden soll. Feature-Label-Paare außerhalb des Bandes werden ignoriert.
direction	Bewegungsrichtung der Partikel. "x" bei simulierten und "y" bei selbst gesammelten Daten.
unitLoc	Einheit des Orts in der Visualisierung. "m" beziehungsweise Meter bei simulierten Daten, "px" beziehungsweise Pixel bei selbst gesammelten Daten.
unitTime	Einheit der Zeit in der Visualisierung. "Frames" beziehungsweise "1/100 Frames" für die hier verwendeten Daten.
limits	Bereich, der in der Visualisierung dargestellt wird. [0.388, 0.788, 0, 0.18] bei simulierten Daten und [0, 2350, 0, 1750] bei selbst gesammelten Daten.

4.4 Hyperparameter-Tuning

Als Hyperparameter-Optimierung oder auch Hyperparameter-Tuning bezeichnet man den Vorgang, das am besten geeignete Set an Hyperparametern für einen Lernalgorithmus zu finden.

In Rahmen dieser Arbeit wurden die Hyperparameter für die verschiedenen Szenarien manuell angepasst und verglichen. Da es zu zeitintensiv wäre, diesen Vorgang für jedes einzelne Szenario, bestehend aus dem verwendeten Schüttgut und der Art der Aufgabe, durchzuführen, wurden insgesamt nur zwei Sets an Hyperparametern gesucht. Eines davon wurde für NextStep-Netze angepasst und das andere für Separator-Netze. Diese wurden beide auf dem DEM-Kugel-Datensatz optimiert und dann auf die anderen Datensätze angewendet. Von dieser Betrachtung ausgenommen wurden Hyperparameter, die das Lernen nicht beeinflussen. Dies sind die Parameter der Kategorien **problem**, in der Dateipfade und Problemdefinitionen zu finden sind, und **data**, in der unter anderem die relative Größe des Testdatensets festgelegt wird.

Für beide Arten von Netzen hat sich gezeigt, dass eine Regularisierung mittels Dropout nicht nur keine Verbesserung mit sich bringt, sondern tatsächlich die Qualität der Ausgabe merklich reduziert. Deshalb wurde für die letztendlich verwendete Trainingskonfiguration die Dropout Rate für alle Szenarien auf 0.0 gesetzt. Auch bezüglich der Aktivierungsfunktionen und der Optimierer unterscheiden sich die beiden Aufgaben nicht. In beiden Fällen haben sich Leaky ReLU und der Adam Optimizer als beste Option hervorgetan.

Sollte ich erklären was genau der Adam Optimizer eigentlich macht?

Eine während des Trainings sinkende Lernrate hat einen positiven Einfluss auf die Güte der Ausgabe der Netze gezeigt. Es wurde eine Technik namens *exponential learning rate decay* eingesetzt, die die Lernrate, die an den Optimierer übergeben wird, in Abhängigkeit der absolvierten Trainingsschritte reduziert. Die verwendete Lernrate entspricht dabei der folgenden Formel:

$$\text{aktuelle Lernrate} = \text{Basislernrate} \cdot \lambda^{\frac{\text{globaler Trainingsschritt}}{\text{Zerfallsschritte}}}$$

Die Zerfallskonstante λ wurde als 0.96 festgelegt, während die Basislernrate und die Zerfallsschritte anpassbare Hyperparameter sind.

Das Ergebnis des Hyperparameter-Tunings bezüglich der Netzarchitektur für die NextStep-Netze ist in Abbildung 4.3 zu sehen. Auf den Testdaten produzierte von den erprobten Konfigurationen eine FeatureSize von 5 in Kombination mit 3 Hidden Layers mit jeweils 16 Neuronen die besten Ergebnisse. Weder L_1 - noch L_2 -Regularisierung haben einen positiven Effekt auf die Trainingsergebnisse gezeigt, weshalb für die

NextStep-Netze darauf verzichtet wurde. Was die Lernparameter angeht, so sieht das Ergebnis des Hyperparameter-Tunings folgendermaßen aus:

- Batch Größe von 500
- Training über insgesamt 3000 Epochen mit jeweils 5000 Iterationen
- Basislernrate von 0.005 und 45 000 Zerfallsschritte

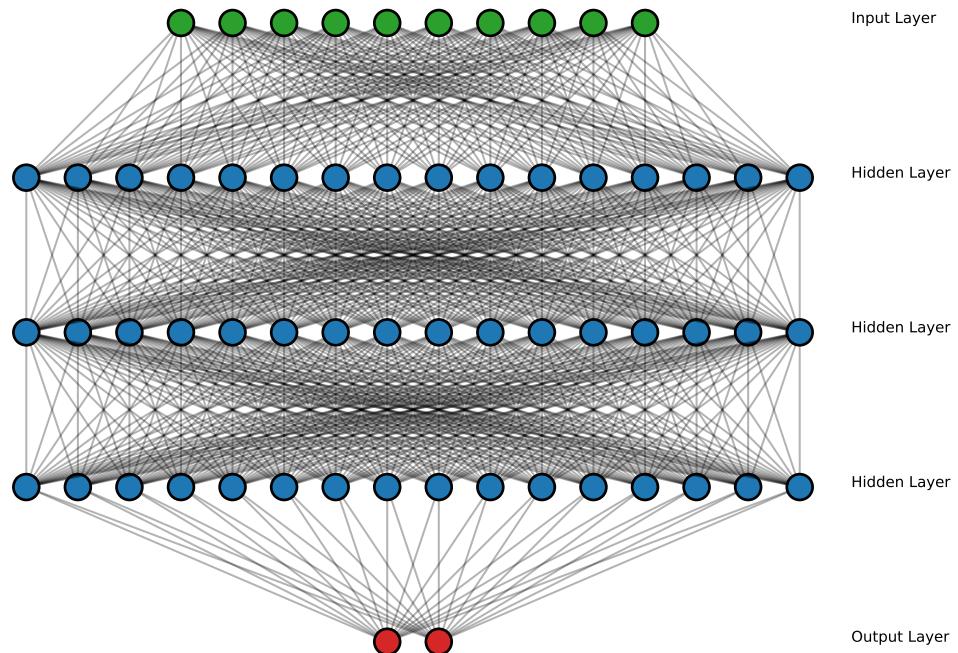


Abbildung 4.3: Architektur eines Neuronales Netzes für die NextStep-Prädiktion.

Die Netzarchitektur, die für die Separator-Netze ermittelt wurde ist in Abbildung 4.4 zu sehen. Hier sorgt eine FeatureSize von 7 für bessere Ergebnisse. Dies könnte zumindest teilweise damit erklärt werden, dass für die Separator-Netze im Gegensatz zu den NextStep-Netzen eine höhere FeatureSize nicht mit einer Reduktion der Anzahl an Trainingsbeispielen einhergeht. Des Weiteren haben 4 Layers mit jeweils 16 Neuronen die besten Ausgaben von den erprobten Konfigurationen erzielt. Im Gegensatz zu den NextStep-Netzen hat die L_1 -Regularisierung bei den Separator-Netzen die Güte der Ergebnisse auf den Testdaten verbessert. Das Hyperparameter-Tuning hat für die restlichen Hyperparameter die folgenden Werte ergeben:

- Batch Größe von 100
- Training über insgesamt 1000 Epochen mit jeweils 5000 Iterationen
- Basislernrate von 0.005 und 60 000 Zerfallsschritte

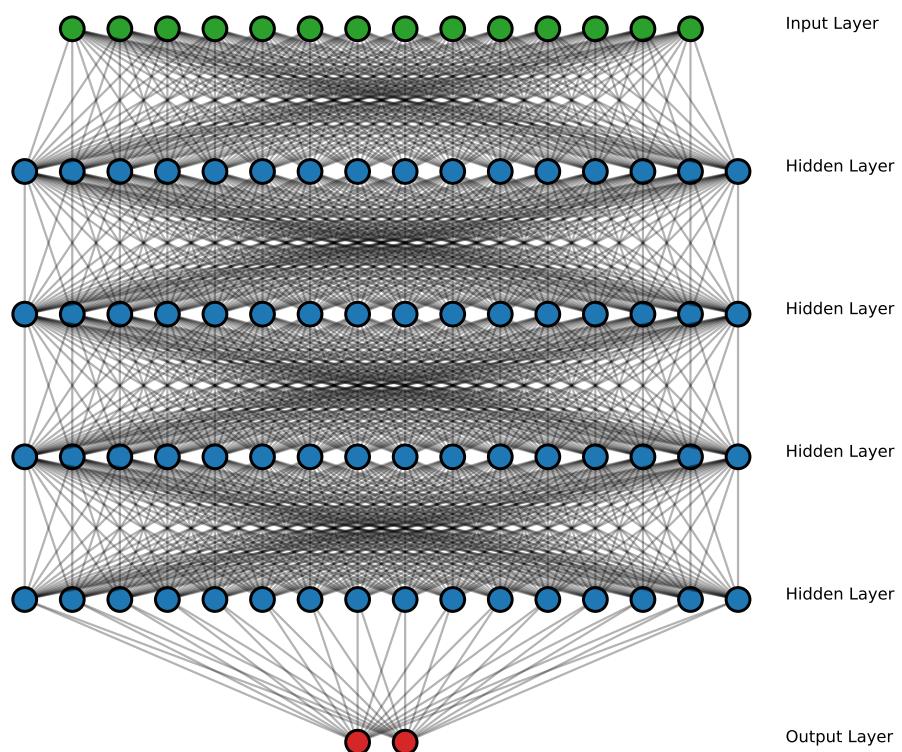


Abbildung 4.4: Architektur eines Neuronales Netzes für die Separator-Prädiktion.

KAPITEL 5

Evaluation

Im vorhergegangenen Kapitel wurde die Implementierung der neuronalen Netze beschrieben. Nun wird in diesem Kapitel betrachtet, wie gut die Ergebnisse sind, die von diesen Netzen geliefert werden. Dazu wird zunächst beschrieben auf was für einem System das Training und die Evaluation durchgeführt wurde. Es werden mehrere, bereits aus dem Stand der Technik bekannte Modelle beschrieben, die dann mit den Ergebnissen der neuronalen Netze verglichen werden. Anschließend werden diese Ergebnisse diskutiert.

Alle in dieser Arbeit vorgestellten Netze wurden auf einem Ubuntu 18.04 Linux System trainiert und evaluiert. Dieses verfügt über einen Intel i7-7700k CPU mit 4.20 GHz, eine NVIDIA GeForce 1080Ti Grafikkarte mit 11 GB GDDR5X und 32 GB RAM.

5.1 Vergleichsmodelle

In diesem Abschnitt sollen einige der existierenden Modelle detaillierter beschrieben werden mit denen die Ergebnisse der Neuronalen Netze nun verglichen werden. Diese Modelle wurden schon in Abschnitt 2.3 erwähnt, dort aber ohne die Berechnung zu erklären. Mit Ausnahme des *Average Acceleration* Modells stammen sie alle aus [Pfa18] und sowohl Definition als auch Notation wurden von dort übernommen. Dabei seien x die Achse entlang der Bewegungsrichtung des Förderbands und y die Achse orthogonal zur Bewegungsrichtung des Förderbands. Zeitdiskreten Messungen entlang der einzelnen Achsen werden als x_t beziehungsweise y_t dargestellt. Die daraus rekonstruierten zeitkontinuierlichen Positionsgleichungen werden als $x(t)$ beziehungsweise $y(t)$ bezeichnet. Sei t^{Last} der Zeitpunkt der Beobachtung des aktuellsten Features und x^{Last} und y^{Last} die dazugehörigen Positionen entlang der beiden Achsen. Sei x^{PredTo} die Position des Druckluftdüsenarrays entlang der x -Achse. Sei t^{Pred} der Zeitpunkt an dem ein Partikel den Druckluftdüsenarray passiert. Sei $y^{\text{Pred}} = y(t^{\text{Pred}})$ die Position entlang der y -Achse an dem der Partikel den Druckluftdüsenarray passiert.

$\Delta t = t^{\text{Pred}} - t^{\text{Last}}$ und y^{Pred} sind die Labels der einzelnen Feature-Label-Paare für das Separator-Netz. $x(t^{\text{Last}} + 1)$ und $y(t^{\text{Last}} + 1)$ sind die Labels der Feature-Label-Paare für das NextStep-Netz. Die verschiedenen Modelle können nun ebenso wie die Ergebnisse der verschiedenen Netze bewertet werden, indem man die Abweichung zwischen dem Ergebnis in dem Modell und der Ground Truth in den Feature-Label-Paaren bestimmt.

Das formatting hier ist total schlecht - überlegen wie ich das gut machen kann.

Für Modelle, die unabhängig von der Beschleunigung der Teilchen sind, ist der Zustandsvektor als

$$\underline{x}(t) = \begin{bmatrix} x(t) \\ \dot{x}(t) \\ y(t) \\ \dot{y}(t) \end{bmatrix}$$

definiert. Für Modelle, die die Beschleunigung der Teilchen mit einbeziehen, ist der Zustandsvektor folgendermaßen definiert:

$$\underline{x}(t) = \begin{bmatrix} x(t) \\ \dot{x}(t) \\ \ddot{x}(t) \\ y(t) \\ \dot{y}(t) \\ \ddot{y}(t) \end{bmatrix}.$$

Irgendwo irgendwas dazu sagen, wie es mutmaßlich wirklich ist? Geschwindigkeit Förderband ist eine obere Grenze - bis dahin beschleunigen, dann konstante Geschwindigkeit. Wie schnell diese Geschwindigkeit erreicht wird hängt von der Art des Schüttguts ab. Je länger das Band desto mehr Zeit hat das Schüttgut sich an die Geschwindigkeit anzupassen.

Constant Velocity Modell Das Constant Velocity (CV) Modell arbeitet unter der Annahme, dass sich Partikel, abgesehen von einem Rauschterm, mit einer konstanten Geschwindigkeit bewegen. Es kann mittels folgender Differenzialgleichung dargestellt werden.

$$\dot{\underline{x}}(t) = \mathbf{A}\underline{x}(t), \quad \mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Daraus folgen die Positionsgleichungen

$$\underline{x}(t) = \underline{x}^{\text{Last}} + (t - t^{\text{Last}})\dot{\underline{x}}^{\text{Last}},$$

$$y(t) = y^{\text{Last}} + (t - t^{\text{Last}})\dot{y}^{\text{Last}}$$

entlang der einzelnen Achsen. Für das NextStep-Szenario ergibt sich die Prädiktionen mittels des CV Modells also aus

$$\underline{x}(t^{\text{Last}} + 1) = \underline{x}^{\text{Last}} + \dot{\underline{x}}^{\text{Last}},$$

$$y(t^{\text{Last}} + 1) = y^{\text{Last}} + \dot{y}^{\text{Last}}. \quad (5.1)$$

Für das Separator-Szenario wird die Gleichung

$$\underline{x}^{\text{PredTo}} = \underline{x}^{\text{Last}} + (t - t^{\text{Last}})\dot{\underline{x}}^{\text{Last}}$$

nach t gelöst um t^{Pred} zu erhalten. Durch das Einsetzen von t^{Pred} in (5.1) ergibt sich y^{Pred} .

Constant Acceleration Modell Im Constant Acceleration (CA) Modell wird davon ausgegangen, dass das das Teilchen mit einer konstanten Beschleunigung schneller wird. Es kann mittels folgender Differenzialgleichung dargestellt werden.

$$\dot{\underline{x}}(t) = \mathbf{A}\underline{x}(t), \quad \mathbf{A} = \begin{bmatrix} \mathbf{A}_x & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_y \end{bmatrix}, \quad \mathbf{A}_x = \mathbf{A}_y = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

Analog zum Constant Velocity Modell können daraus die Positionsgleichungen

$$\underline{x}(t) = \underline{x}^{\text{Last}} + (t - t^{\text{Last}})\dot{\underline{x}}^{\text{Last}} + \frac{1}{2}(t - t^{\text{Last}})^2 \ddot{\underline{x}}^{\text{Last}},$$

$$y(t) = y^{\text{Last}} + (t - t^{\text{Last}})\dot{y}^{\text{Last}} + \frac{1}{2}(t - t^{\text{Last}})^2 \ddot{y}^{\text{Last}}$$

abgeleitet werden. Genau wie beim Constant Velocity Modell werden nun jeweils entweder die Gleichung für $t = \text{Last}+1$ gelöst um die Prädiktion für den NextStep-Fall oder Δt und daraus y^{Pred} für den Separator-Fall gelöst.

Bias-Corrected Constant Velocity Modell In [Pfa18] wurden weitere szenariospezifische Bewegungsmodelle beschrieben, die insbesondere die Qualität der zeitlichen Prädiktion für den Separator-Fall verbessern. Das erste von diesen Modellen, das hier zum Vergleich mit den Ergebnissen der neuronalen Netze betrachtet werden soll, ist eine Verbesserung des Constant Velocity Modells und wird als Bias-Corrected Constant Velocity (CVBC) Modell bezeichnet. Durch das Bestimmen des durchschnittlichen Bias bezüglich der Ankunftszeit der Partikel am Druckluftdüsenarray in den Trainingsdaten und das Abziehen desselben vom der Prädiktion der Testdaten kann die Qualität dieser Prädiktion verbessert werden. Dieser neue Wert kann nun für die Ortsprädiktion benutzt werden und verbessert auch diese. Damit ergibt sich für die vorhergesagte Ankunftszeit

$$t^{\text{Pred, CVBC}} = t^{\text{Pred, CV}} - t^{\text{Bias}},$$

wobei t^{Bias} den durchschnittlichen Bias bezüglich der Ankunftszeit bezeichnet. Analog zum unveränderten Constant Velocity Modell erhält man nun durch das Einsetzen von $t^{\text{Pred, CVBC}}$ in (5.1) erneut $y^{\text{Pred, CVBC}}$. Die Verbesserung der prädizierten Zeit führt ebenfalls zu einer Verbesserung des prädizierten Orts.

Average Acceleration Modell Das Average Acceleration (AA) Modell ist eine Erweiterung des CVBC-Modells. Anstatt die durchschnittliche Abweichung der prädizierten Zeit in den Trainingsdaten auf die prädizierte Zeit zu addieren, wird angenommen, dass diese Abweichung aus einer nicht betrachteten Beschleunigung resultiert. Diese Beschleunigung wird mutmaßlich vom Band verursacht werden und alle Partikel mehr oder weniger ähnlich betreffen. Deshalb wird die Beschleunigung des Partikels in allen Feature-Label-Paaren des Trainingssets bestimmt und von diesen der Median \ddot{x}^{Median} gewählt als eine einheitliche Beschleunigung, die zum CV Modell hinzugefügt wird. Es folgt die Positionsgleichung

$$\mathbf{x}(t) = \mathbf{x}^{\text{Last}} + (t - t^{\text{Last}}) \dot{\mathbf{x}}^{\text{Last}} + \frac{1}{2}(t - t^{\text{Last}})^2 \ddot{\mathbf{x}}^{\text{Median}},$$

die nach t für $\mathbf{x}(t) = \mathbf{x}^{\text{PredTo}}$ gelöst wird.

Identical Acceleration Modell Das Identical Acceleration (IA) Modell ist ebenfalls eine Verbesserung des CVBC-Modells. Bei diesem werden die Korrekturterme, die als Beschleunigung auf die Positionsgleichungen addiert werden, nicht unabhängig von der letzten Position des Partikels bestimmt, sondern beziehen diese mit ein. Wie oben wird hierbei die Annahme getroffen, dass die zusätzliche Beschleunigung, die für die zeitliche Abweichung sorgt, für alle Partikel ungefähr gleich ist. Die Bestimmung dieser Korrekturterme ist jedoch unterschiedlich. Für die Feature-Label-Paare des

Trainingssets hat man Zugang zu der Ground Truth, wann die Partikel das Druckluftdüsenarray passieren.

Dementsprechen wird für jedes Partikel i aus dem Trainingset die Gleichung

$$x^{\text{PredTo}} = x^{\text{Last}, i} + (t^{\text{GT}, i} - t^{\text{Last}, i}) \dot{x}^{\text{Last}, i} + \frac{1}{2} (t^{\text{GT}, i} - t^{\text{Last}, i})^2 \ddot{x}^{\text{Optimal}, i}$$

gelöst, um herauszufinden, mit welcher zusätzlichen Beschleunigung $\ddot{x}^{\text{Optimal}, i}$ es optimal die Zeit, die es noch braucht, vorhersagen würde. Nun sei \ddot{x}^{Avg} der Durchschnitt von allen $\ddot{x}^{\text{Optimal}, i}$. Für die Partikel des Testsets werden die Zeit und der Ort des Passierens des Druckluftdüsenarrays nun basierend auf deren beobachteter Geschwindigkeit \dot{x}^{Last} und der errechneten Beschleunigung \ddot{x}^{Avg} berechnet. Die Zeitprädiktion t^{Pred} wird bestimmt indem wir

$$x(t) = x^{\text{Last}} + (t - t^{\text{Last}}) \dot{x}^{\text{Last}} + \frac{1}{2} (t - t^{\text{Last}})^2 \ddot{x}^{\text{Avg}}$$

nach t lösen.

5.2 Next Step

In dieser Sektion sollen die Ergebnisse der NextStep-Netze in verschiedenen Szenarien betrachtet werden. Der Vergleich zwischen den Ergebnissen der unterschiedlichen Modelle wird mittels sogenannter Boxplots visualisiert. Die fünf relevanten Charakteristiken, die in einem Boxplot dargestellt werden sind der Median, das untere und obere Quartil sowie zwei sogenannte „Antennen“ oder auch „Whisker“, die die Position den letzten Datenpunkt innerhalb des 1.5-fachen Interquartilsabstands beschreiben. Die Position des Medians wird durch eine rote Linie verdeutlicht. Die namensgebende Box ist zwischen dem unteren und dem oberen Quartil aufgespannt. Für die Darstellung in dieser Arbeit wurde darauf verzichtet Ausreißer außerhalb der Antennen abzubilden. Als Evaluationskriterium für die NextStep-Netze wurde die Euklidische Distanz zwischen der Prädiktion des Modell und der Ground Truth gewählt. Der Gesamtfehler $\varepsilon^{\text{Total}}$ ist also durch

$$\begin{aligned} x^{\text{Err}} &= x^{\text{Pred}} - x^{\text{GT}}, \\ y^{\text{Err}} &= y^{\text{Pred}} - y^{\text{GT}}, \\ \varepsilon^{\text{Total}} &= \sqrt{x^{\text{Err}}{}^2 + y^{\text{Err}}{}^2} \end{aligned}$$

definiert. Verglichen wird das NextStep-Netz mit einem Constant Velocity Modell und einen Constant Acceleration Modell.

Repräsentativ für ihre jeweiligen Szenariokategorien sind in Abbildung 5.1a die Boxplots für die selbst aufgenommenen Kugeln auf dem Förderband, und in Abbildung 5.1b die Boxplots für die mittels DEM simulierten Kugeln zu sehen. In beiden Fällen sind die Ergebnisse des neuronalen Netzes besser als die Vergleichsmodelle. Sowohl der Median als auch die Quartile sind für die Ergebnisse des neuronalen Netzes geringer. Der Bias der Ergebnisse ist kleiner und die weniger weit gestreut als die der beiden Vergleichsmodelle. Es ist auffällig, dass bei den simulierten Daten das CA-Modell besser als das CV-Modell ist, während es im anderen Fall umgekehrt ist. Das könnte auf die höhere Bandgeschwindigkeit bei der Simulation zurückzuführen. In Abbildung 5.2 sieht man das Fehlerhistogramm, dass zu Abbildung 5.1a gehört.

Die Boxplots und Histogramme der übrigen Szenarien sind im Anhang zu finden.

hier könnte ich noch mehr schreiben zu verschiedenen anderen Szenarien, wenn es notwendig sein sollte.

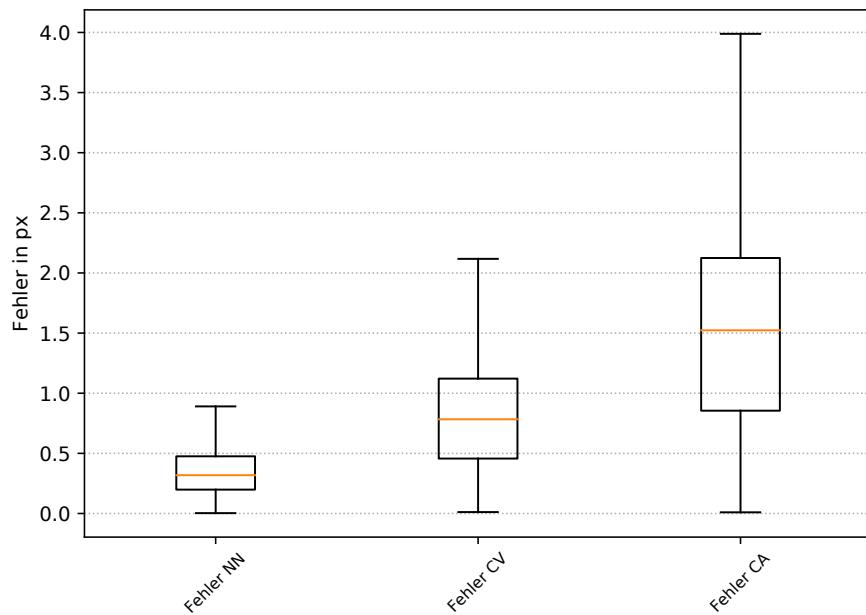
NextStep Boxplot schön machen ohne titel und in mm

5.3 Separator

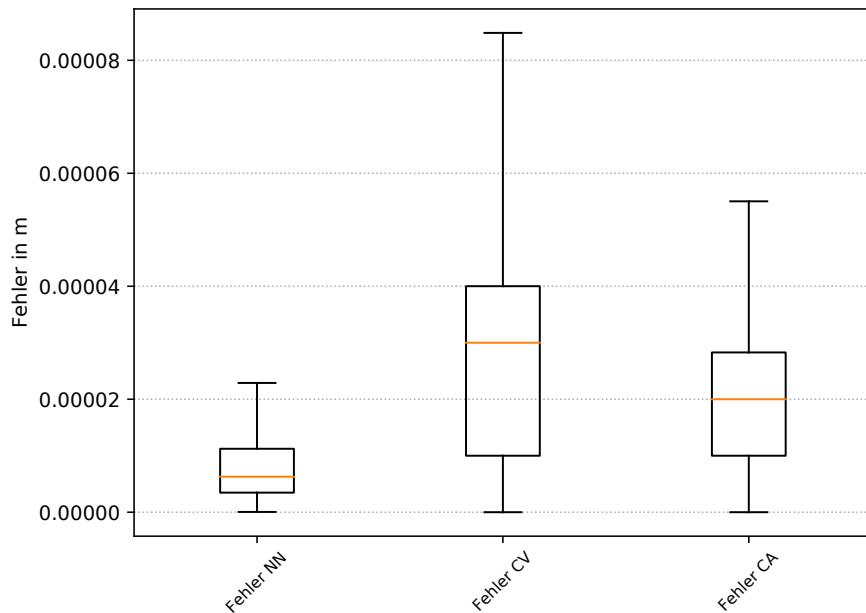
In dieser Sektion sollen nun die Ergebnisse der Separator-Netze in verschiedenen Szenarien betrachtet werden. Hier wird die zeitliche Abweichung vom prädizierten Kontaktzeitpunkt mit dem Druckluftdüsenarray und die örtliche Abweichung des Schnittpunkts der Bahn des Partikels mit dem Druckluftdüsenarray separat betrachtet. Wie bei den NextStep-Netzen werden ein Constant Velocity- und ein Constant Acceleration Modell als Vergleichsmodelle benutzt. Zusätzlich werden noch mehr Vergleichsmodelle hinzugezogen. Das Bias-Corrected Constant Velocity Modell liefert eine sowohl Prädiktion für den Zeit als auch für den Ort. Die Average Acceleration und Identical Acceleration Modelle werden nur für die zeitliche Prädiktion zum Vergleich herangezogen.

Die Evaluation, die hier durchgeführt wurde erfolgt auf folgendem Szenario. Als Trainingsdaten wurden nur das jeweils letzte Feature-Label-Paar eines Tracks benutzt, bevor das entsprechende Partikel die Prädiktionsphase verlässt. Es wäre in der Zukunft möglich auf allen möglichen Feature-Label-Paaren zu trainieren, um zu erreichen, dass der Ausgangspunkt verschoben werden kann ohne das Netz neu trainieren zu müssen. Für die DEM-Datensätze wurde von $x = 0.55$ m nach $x = 0.70$ m prädiziert, also 15 cm nach vorne. Da der gesamte Bildausschnitt der selbst aufgenommenen Daten weniger als 15 cm entlang der Bewegungsrichtung misst, musste diese Distanz

Hier weg und dafür in den Ausblick?



(a) Boxplots für die NextStep-Ergebnisse der selbst aufgenommenen Kugeln auf dem Förderband.



(b) Boxplots für die NextStep-Ergebnisse der Kugeln aus dem DEM-Datensatz.

Abbildung 5.1: Visualisierung der Ergebnisse für die realen und simulierten Kugeln mit NextStep-Netzen. Kleinere Werte sind besser.

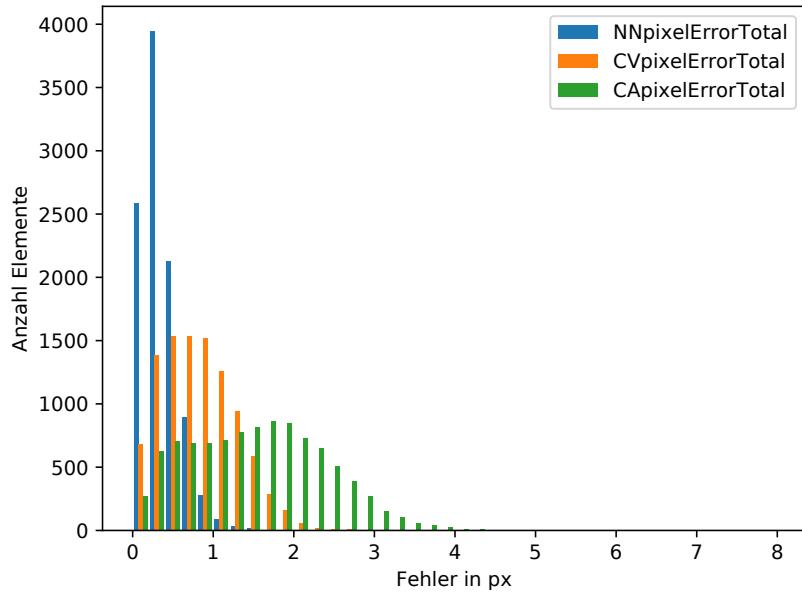


Abbildung 5.2: Histogramm der Fehler für die NextStep-Ergebnisse der selbst aufgenommenen Kugeln auf dem Förderband. Mehr Elemente mit kleinerem Fehler sind besser.

reduziert werden. Die auf den selbst gesammelten Daten trainierten Netze prädizieren von $y = 800 \text{ px}$ nach $y = 1550 \text{ px}$, also eine Distanz von 750 px .

Die Ergebnisse für den Separator-Fall werden zweigeteilt als

$$t^{\text{Err}} = t^{\text{Pred}} - t^{\text{GT}},$$

$$y^{\text{Err}} = y^{\text{Pred}} - y^{\text{GT}}$$

definiert. Im Gegensatz zu dem NextStep-Fall ist es nicht sinnvoll diese Fehler zu einem Gesamtfehler zu kombinieren. Zusätzlich ist das Vorzeichen der Fehler für die einzelnen Partikel hier relevant. Für die zeitlichen Fehler bedeutet ein positiver Wert, dass das Teilchen früher den Druckluftdüsenarray passiert hat als von dem Modell vorhergesagt.

Die Ergebnisse der neuronalen Netze für die Separator-Prädiktion sind insgesamt zufriedenstellend. Sie sind auf allen Datensätzen besser als das CV-Modell, das CV-Modell und das CVBC-Modell. Auf den beiden simulierten Datensätzen sind die Ergebnisse ein wenig schlechter als das IA-Modell, aber deutlich besser als die grundlegenden Modelle und das Bias-Corrected Constant Velocity Modell. Dies ist in Abbildung 5.3 exemplarisch durch die Boxplots der simulierten Plättchen dargestellt. Zunächst soll der zeitliche Fehler betrachtet werden. Obwohl der Median des Prädiktionsfehlers einen kleinen Bias ins Positive hat und das untere Quartil tatsächlich positiv ist, werden die Fehler der neuronalen Netzes komplett von denen

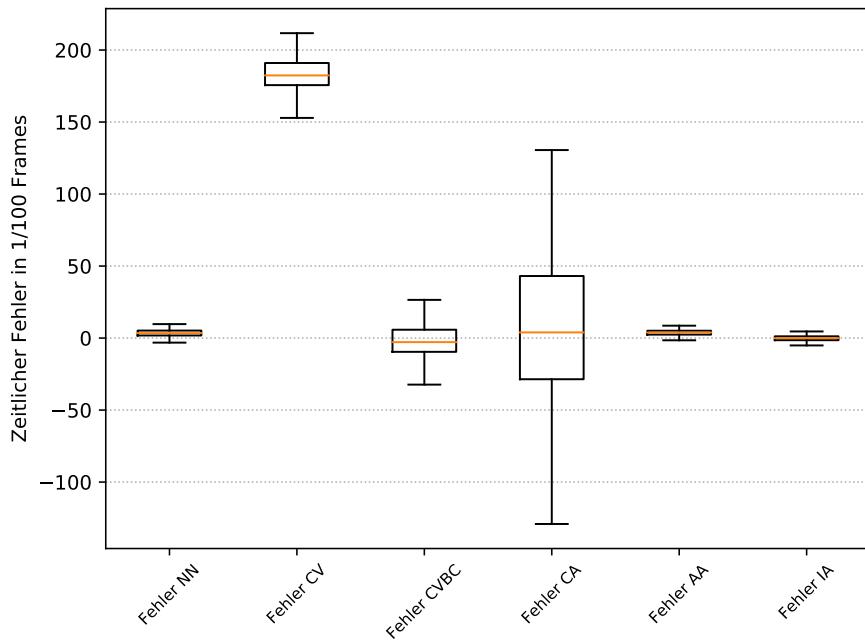
hier umrechnung in cm
dazumachen?
ungefähr 4.5
cm

Reihenfolge?
erst Daten
dann schluss-
folgerungen

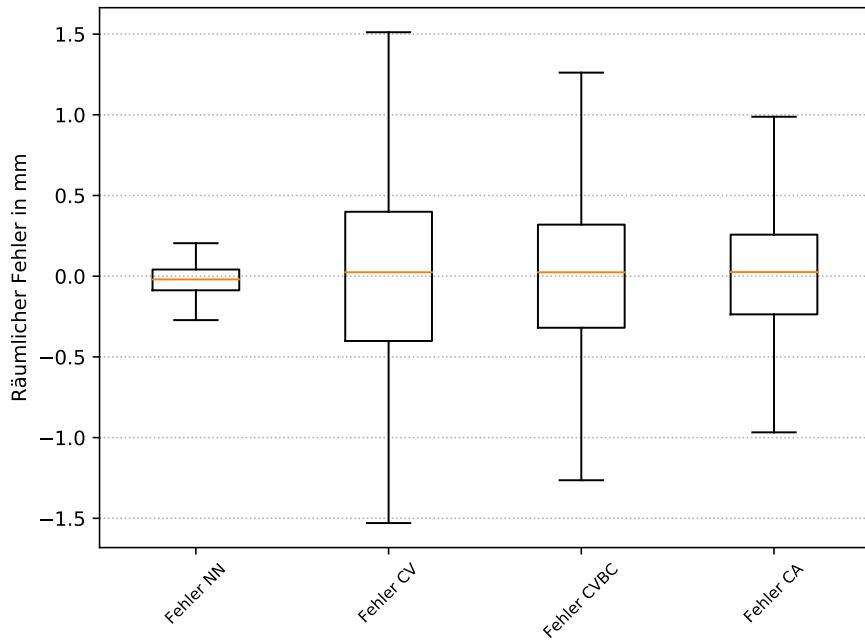
des CVBC-Modells überdeckt. Das CV- und das CA-Modell sind beide noch einmal bedeutend schlechter als das CVBC-Modell. Das CV-Modell hat einen Bias von beinah zwei ganzen Zeitschritten während das CA-Modell zwar beinah biasfrei ist, ungefähr in der gleichen Größenordnung wie das neuronale Netz, hat es mit Abstand die größte Standardabweichung. Diese ist ungefähr dreimal so groß wie die des Ergebnisses des neuronalen Netzes. Das beste Modell in diesem Szenario ist das IA-Modell. Es ist beinah komplett biasfrei und hat die geringste Standardabweichung von allen Modellen. Bei der Betrachtung des örtlichen Fehlers ist das CA-Modell tatsächlich besser als das CV- oder das CVBC-Modell, aber dennoch deutlich schlechter als das neuronale Netz. Alle vier Modelle sind beinahe biasfrei, aber der Interquartilsabstands der Ausgabe des neuronalen Netzes sind deutlich geringer.

Im Vergleich dazu sind die Unterschiede zwischen den verschiedenen Modellen auf den selbst aufgenommenen Daten deutlich kleiner. Das Constant Velocity Modell schneidet oft beinah so gut und bei den Weizenkörnern sogar besser ab, als das Identical Acceleration Modell. Auf den selbst gesammelten Daten schneidet von den bestehenden Modellen das Bias-Corrected Constant Velocity Modell am besten ab. Die neuronalen Netze sind jedoch noch einmal besser. Wenn man die Boxplots der zeitlichen Fehler der verschiedenen Modelle auf dem Weizenkörner-Datensatz der selbst aufgenommenen Daten, dargestellt in Abbildung 5.4a, im Detail betrachtet gibt es einige Auffälligkeiten. Im Gegensatz zu dem in Abbildung 5.3a dargestellten Szenario zeigt der zeitliche Fehler des CV-Modells nur einen geringen Bias, der sogar eher negativ ist. Während es im bei den simulierten Plättchen so ist, dass der Interquartilsabstands des CV-Modells um Faktoren größer ist als der der IA-Modells, ist er hier fast gleich groß. Tatsächlich unterscheiden sich die Ergebnisse der CV-, CVBC-, AA- und IA- fast nur in ihrem Bias. Dem gegenüber stehen die Ergebnisse des neuronalen Netzes, die fast komplett biasfrei sind und einen merklich kleineren Interquartilsabstands hat als jedes der bestehenden Modelle. Ähnlich ist es beim räumlichen Fehler, der in Abbildung 5.4b dargestellt ist. Obwohl die Ergebnisse des neuronalen Netzes einen kleinen Bias haben, wird der Bereich zwischen ihrem oberen und unteren Quartil immer noch komplett überdeckt von der Box des CVBC-Modells, das das Beste der Vergleichsmodelle ist.

Die Resultate der bestehenden Modelle hängen höchstwahrscheinlich mit der niedrigeren Bandgeschwindigkeit und dem kürzeren Prädiktionsabstand zusammen. Auch die könnten die unterschiedlichen Verfahren unterschiedlich anfällig gegen das Messrauschen sein, das auf den simulierten Daten nicht existiert. Für die bestehenden Modelle gilt, dass eine relative Verbesserung der zeitlichen Prädiktion zu einer Verbesserung der örtlichen Prädiktion führt, da die zeitliche Prädiktion benutzt wird um die örtliche zu bestimmen. Das ist besonders gut in Abbildung 5.3 zu erkennen. Die



(a) Boxplots des zeitlichen Fehlers für die Separator-Ergebnisse der Plättchen aus dem DEM-Datensatz.



(b) Boxplots des örtlichen Fehlers für die Separator-Ergebnisse der Plättchen aus dem DEM-Datensatz.

Abbildung 5.3: Visualisierung der Ergebnisse für die Plättchen aus der DEM-Simulation. Median näher an 0 und möglichst kleine Streuung sind besser.

Verbesserung der zeitlichen Prädiktion, zu sehen in Abbildung 5.3a, zwischen dem CV- und dem CVBC-Modell sorgt auch direkt für eine Verbesserung der örtlichen Prädiktion, zu sehen in Abbildung 5.3b. Für die Ausgabe der neuronalen Netze gilt das jedoch nicht. Hier werden beide Ausgaben parallel bestimmt.

Wie schon für die NextStep-Netze sind die Boxplots und Histogramme der restlichen Szenarien im Anhang zu finden.

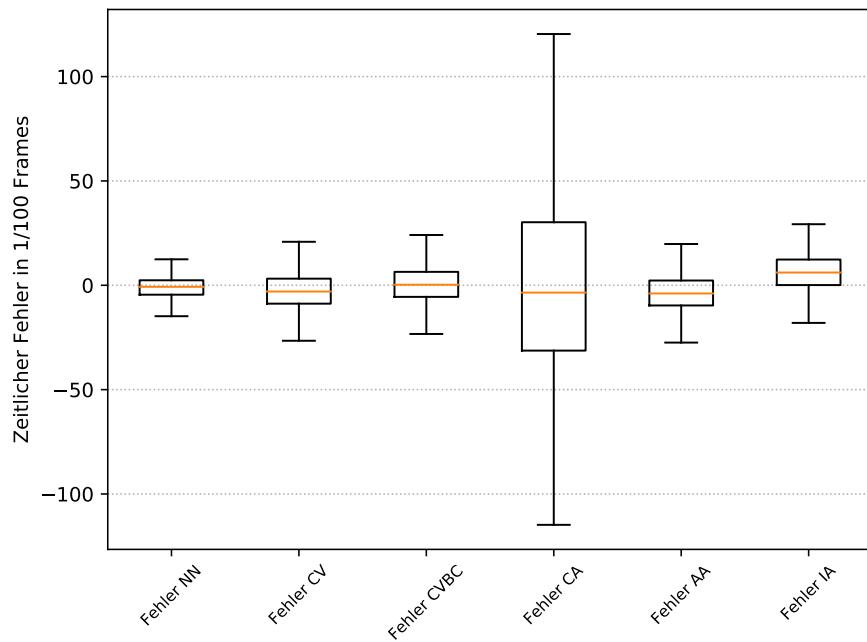
5.4 Zusammenfassung und Diskussion

Einleitung und Zusammenfassung

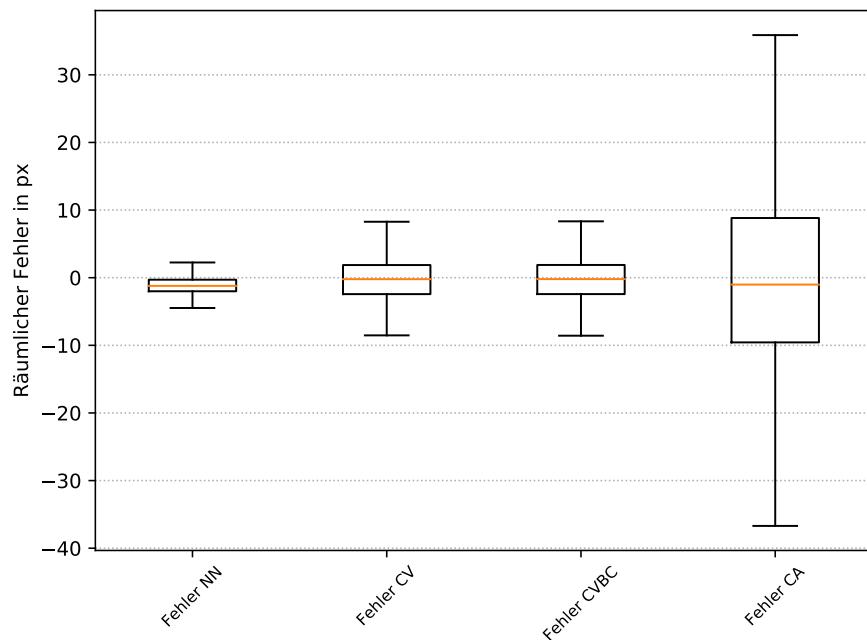
Die Tatsache, dass die Separator-Netze auf den DEM-Datensätzen, im Gegensatz zu allen anderen Szenarios, hinter dem aktuellen Stand der Technik zurück bleiben könnte darauf zurückzuführen sein, dass dort verhältnismäßig weniger Trainingsdaten zur Verfügung standen. Wie bereits in Kapitel 3 erwähnt, haben DEM-Datensätzen im Vergleich zu den selbst aufgenommenen Daten weniger, aber dafür deutlich längere Tracks. Für die NextStep-Netze sorgt die größere Länge dazu, dass sogar mehr Feature-Label-Paare aus den DEM-Daten extrahiert werden können als aus den selbst aufgenommenen. Für die Separator Netze ist die Länge der Tracks jedoch unerheblich. Aus einem Track entsteht nur ein einzelnes Feature-Label-Paar. Trotz der Datenaugmentierung scheint es so, als ob nicht genug Trainingsdaten vorhanden seien.

In der Zukunft wäre es wünschenswert ein präziseres Hyperparameter-Tuning für mehr Szenarios durchzuführen. Wie in Abschnitt 4.3 beschrieben, wurden nur zwei Hyperparameter-Konfigurationen optimiert, was jeweils auf dem DEM-Kugel-Datensatz durchgeführt wurde. Diese Konfigurationen, eine für NextStep-Netze und eine für Separator-Netze, wurden dann für alle verschiedenen Szenarien benutzt. Auch wenn es ein beträchtlicher Zeitaufwand wäre, würden individuelles Hyperparameter-Tuning für die einzelnen Schüttgutsorten wahrscheinlich einen positiven Effekt auf die Güte der Ergebnisse haben. Insbesondere ist es so, dass das Ergebnis für die Separator-Netze nicht gut auf die Rutschenkonfiguration des *TableSort* Systems anwendbar war. Die Partikel bewegen sich in dieser Konfiguration deutlich schneller als auf dem Förderband, weshalb weniger Beobachtungen pro Track im Sichtbereich der Kamera gemacht werden. Eine FeatureSize von 7, wie von der Hyperparametern-Konfiguration für Separator-Netze vorgesehen, ist bei einer durchschnittlichen Tracklänge von 10 nicht sinnvoll.

Es konnte beobachtet werden, dass bei mehrmaligem Trainieren eines Netzes mit identischen Hyperparametern und Daten, unterschiedliche Ergebnisse mit einem



(a) Boxplots des zeitlichen Fehlers für die Separator-Ergebnisse des Weizenkörnern-Datensatz von den selbst gesammelten Daten.



(b) Boxplots des örtlichen Fehlers für die Separator-Ergebnisse des Weizenkörnern-Datensatz von den selbst gesammelten Daten.

Abbildung 5.4: Visualisierung der Ergebnisse für den Weizenkörnern-Datensatz von den selbst gesammelten Daten. Median näher an 0 und möglichst kleine Streuung sind besser.

ähnlichen MSE insgesamt aber einer Verschiebung zwischen dem Mittelwert und der Standardabweichung an dessen Ende standen. Für Separator-Netze kam es sogar zu Verschiebungen zwischen der örtlichen und der zeitlichen Prädiktion. Die Fehlerfunktion, nach der die neuronalen Netze optimiert worden ist, ist die MSE-Funktion, wonach zwei Ergebnisse mit identischem MSE wertig sind, egal wie dieser zustande kommt. Dies ist auf verschiedene stochastische Vorgänge während dem Training zurück zu führen, wie die Initialisierung der Gewichte zwischen den Neuronen und die Reihenfolge beziehungsweise das Batching der Trainingsbeispiele. Der Einsatz einer anderen Fehlerfunktion, zum Beispiel einer, die tatsächlich den Anteil der korrekt separierten Teilchen bewertet könnte dieses Problem beheben.

den Ganzen Absatz weglassen? So wirklich was trägt er nicht bei zum Ergebnis und er steht ohne daten da auch sehr allein da.

KAPITEL 6

Fazit und Ausblick

Im Rahmen dieser Arbeit wurde betrachtet, ob neuronale Netze ein Werkzeug sind, das für die Bewegungsprädiktion von Schüttgutpartikeln eingesetzt werden kann. Dazu wurden zunächst beinah 250 000 Bilder von verschiedenen Schüttgütern auf dem *TableSort* Schüttgutsortierer aufgenommen und eine Pipeline entwickelt mit der die relevanten Features aus solchen Bildern extrahiert werden können. Zusätzlich wurde eine einfache Form der Datenaugmentierung implementiert, die das Volumen von verfügbaren Trainingsdaten aus einem Datenset beinahe verdoppelt. Es wurde mit Hilfe des TensorFlow Frameworks eine Implementierung erarbeitet, mit der verschiedene neuronale Netze trainiert werden können. Mit dieser Implementierung wurden mehrere Netze auf verschiedenen Datensets – den selbst aufgenommenen und bereits existierenden, mittels DEM simulierten – trainiert und deren Ergebnisse evaluiert. Allgemein kann gesagt werden, dass die Ergebnisse zufriedenstellend sind. Es konnte gezeigt werden, dass sowohl NextStep-, als auch Separator-Netze gut in der Lage sind, die an sie gestellten Probleme zu lösen. In jedem Szenario konnten Ergebnisse erzielt werden, die besser als die der zwei grundlegenden Bewegungsmodelle, und vergleichbar mit dem aktuellen Stand der Technik waren. Insbesondere auf den Datensätzen, die auf realen Aufnahmen basieren, wurden die Ergebnisse des aktuellen Stands der Technik übertroffen.

In dieser Arbeit wurde nur ein Ansatz für den Einsatz von neuronalen Netzen für die Bewegungsprädiktion von Schüttgutpartikeln erprobt. Und obwohl positive Ergebnisse erzielt wurden, gibt es noch viele weitere vielversprechende Ansätze, die es wert sind, betrachtet zu werden. Naheliegende Ideen wären zum Beispiel alternative Modellierungen, wie das relative Positionen statt absoluten als Eingabe verwendet werden, oder zusätzliche Features, die zu den existierenden Objektmittelpunkten hinzugekommen. Beispielsweise ein Indikator für die Orientierung würde speziell bei den Zylindern Sinn ergeben. Auch ist denkbar, dass andere Netzwerkarchitekturen erprobt werden könnten. Rekurrente neuronale Netze sind besonders gut dafür geeignet Daten, bei denen die zeitliche Abfolge wichtig ist sequenziell zu verarbeiten.

Kapitel 6. Fazit und Ausblick

Maschinelles Lernen allgemein und neuronale Netze im Besonderen sind ein Feld, auf dem momentan sehr schnell eine Menge Fortschritte gemacht werden, die in naher Zukunft vielleicht ganz neue Möglichkeiten eröffnen werden. Ein Aspekt, in dem Convolutional Neural Networks schon heute beeindruckende Ergebnisse liefern, ist das Extrahieren von Informationen direkt aus Bildern. Es gibt bereits heute CNN basierte Online Multi-Object-Tracking Verfahren die sehr gute Ergebnisse erzielen.

Deshalb liegt es nahe, dass es möglich sein sollte auf den Bildern direkt Ende-zu-Ende zu trainieren. Das würde unter anderem das Problem von Messunsicherheiten bei der Bestimmung der Objektmittelpunkte irrelevant machen. Sowohl im aktuellen

Stand der Technik als auch bei den im Rahmen dieser Arbeit trainierten Netzen werden alle Schüttgutpartikel individuell betrachtet. Dies führt dazu, dass Kollisionen zwischen Partikeln, die diese von ihrer Bahn ablenken höchstens detektiert, aber nicht sinnvoll in die Modelle miteinbezogen werden können. Ein Ansatz, der dies tut, könnte insbesondere für Schüttgüter mit einer starken Querbewegung zu einer Verbesserung der Sortierqualität führen.

hier eine Quelle?

hier anmerken,
dass das wahr-
scheinlich ein
größeres Pro-
jekt wäre und
mehr als nur
ne MA?

Literatur

- [Sch09] D. Schulze, *Pulver und Schüttgüter: Fließeigenschaften und Handhabung*, 2., bearb. Aufl, Ser. Verfahrenstechnik. Berlin: Springer, 2009, 520 S., OCLC: 304395912, ISBN: 978-3-540-88449-1 978-3-540-88448-4.
- [UC18] United Nations Conference on Trade and Development und CNUCED, „Review of Maritime Transport 2018“, 2018, OCLC: 7898712265, ISSN: 0566-7682.
- [Dur12] J. Duran, *Sands, Powders, and Grains: An Introduction to the Physics of Granular Materials*. Springer Science & Business Media, 2012.
- [Pfa18] F. Pfaff, „Multitarget Tracking Using Orientation Estimation for Optical Belt Sorting“, Verteidigt am 13.11.2018, Dissertation, Karlsruher Institut für Technologie, Karlsruhe, 2018.
- [GBC16] I. Goodfellow, Y. Bengio und A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [Mur12] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012, ISBN: 0-262-01802-0 978-0-262-01802-9.
- [Nie15] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination press USA, 2015, Bd. 25.
- [MP43] W. S. McCulloch und W. ter Pitts, „A Logical Calculus of the Ideas Immanent in Nervous Activity“, *The bulletin of mathematical biophysics*, Jg. 5, Nr. 4, S. 115–133, 1943.
- [Ros58] F. Rosenblatt, „The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.“, *Psychological review*, Jg. 65, Nr. 6, S. 386, 1958.
- [WAZ11] Wikimedia Commons, Alisneaky und Zirguezi, „Kernel Machines Are Used to Compute Non-Linearly Separable Functions into a Higher Dimension Linearly Separable Function.“, 17. Apr. 2011. Adresse: https://commons.wikimedia.org/wiki/File:Kernel_Machine.svg (besucht am 12.12.2018).

- [KSH12] A. Krizhevsky, I. Sutskever und G. E. Hinton, „ImageNet Classification with Deep Convolutional Neural Networks“, in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou und K. Q. Weinberger, Hrsg., Curran Associates, Inc., 2012, S. 1097–1105. Adresse: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.
- [SHK+14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever und R. Salakhutdinov, „Dropout: A Simple Way to Prevent Neural Networks from Overfitting“, *Journal of Machine Learning Research*, Jg. 15, S. 1929–1958, 2014. Adresse: <http://jmlr.org/papers/v15/srivastava14a.html>.
- [Dol15] F. Doll, „Konzeption, Planung, Konstruktion Und Integration Eines Miniaturisierten, Modularen Schüttgutsortierers“, KIT, Aug. 2015.
- [Fra17] Fraunhofer IOSB, *TableSort Schüttgutsortierer*, 19. Mai 2017.
- [PKP+17] F. Pfaff, G. Kurz, C. Pieper, G. Maier, B. Noack, H. Kruggel-Emden, R. Gruna, U. D. Hanebeck, S. Wirtz, V. Scherer, T. Langle und J. Beyerer, „Improving Multitarget Tracking Using Orientation Estimates for Sorting Bulk Materials“, IEEE, Nov. 2017, S. 553–558, ISBN: 978-1-5090-6064-1. doi: [10.1109/MFI.2017.8170379](https://doi.org/10.1109/MFI.2017.8170379).
- [All14] Allied Vision Technologies GmbH, *User Manual for Bonito CL-400 200 Fps High Speed Camera*. Nov. 2014, Rev. I.
- [WC06] Wikimedia Commons und Cburnett, „A Bayer Pattern on a Sensor in Isometric Perspective/Projection“, 26. Dez. 2006. Adresse: https://commons.wikimedia.org/wiki/File:Bayer_pattern_on_sensor.svg (besucht am 27.11.2018).
- [PKEW+16] C Pieper, H Kruggel-Emden, S Wirtz, V Scherer, F Pfaff, B Noack, U. Hanebeck, G Maier, R Gruna, T Länge und others, „Numerical Investigation of Optical Sorting Using the Discrete Element Method“, in *International Conference on Discrete Element Methods*, Springer, 2016, S. 1105–1113.
- [PPM+17] C Pieper, F Pfaff, G Maier, H Kruggel-Emden, S Wirtz, B Noack, R Gruna, V Scherer, U. Hanebeck, T Länge und others, „Numerical Modelling of an Optical Belt Sorter Using a DEM–CFD Approach Coupled with Particle Tracking and Comparison with Experiments“, 2017.

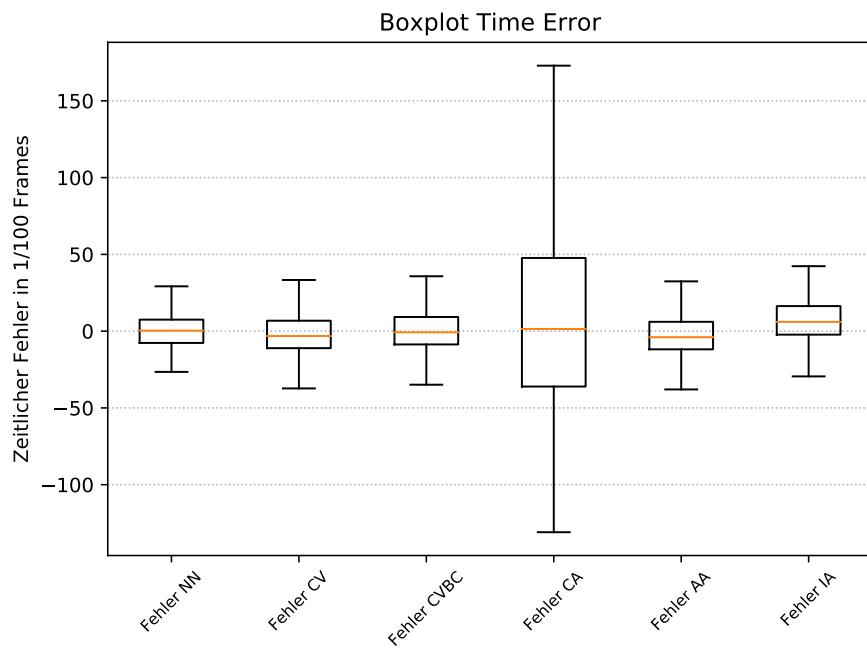
ANHANG A

Anhang

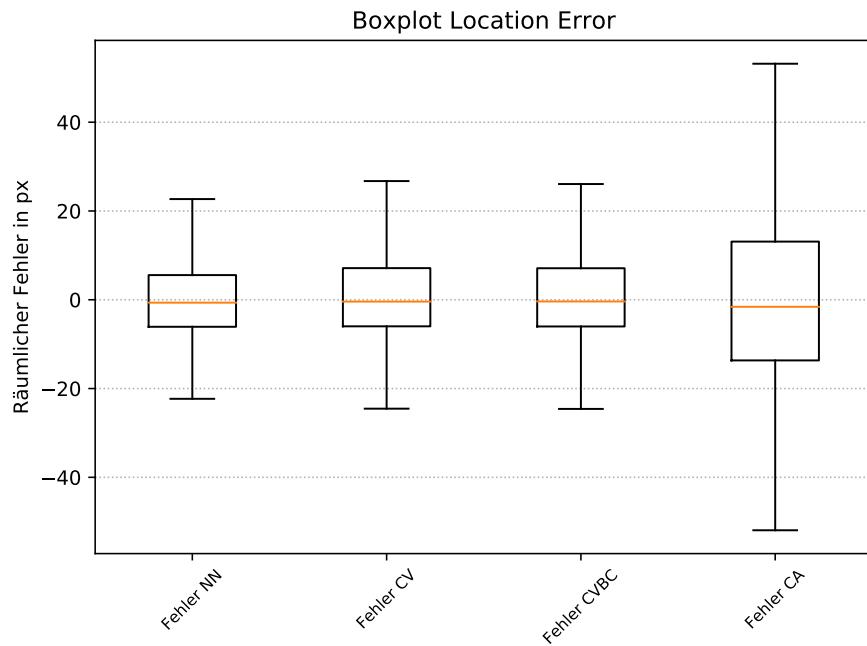
Hier ist der Anhang. Hier kommen Dinge Rein, wie Evaluationsergebnisse, die den Hauptteil zu voll machen würden, Tabellen mit daten, die nur begrenzt was mit der Arbeit zu tun haben,

...

more to come

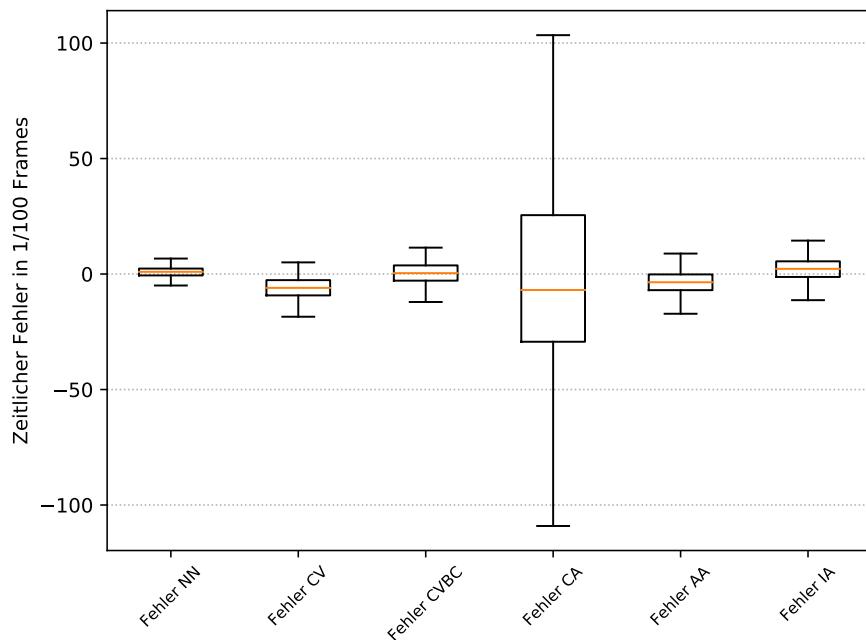


(a) Boxplots des zeitlichen Fehlers für die Separator-Ergebnisse der Pfefferkörner aus dem selbst aufgenommenen Datensatz.

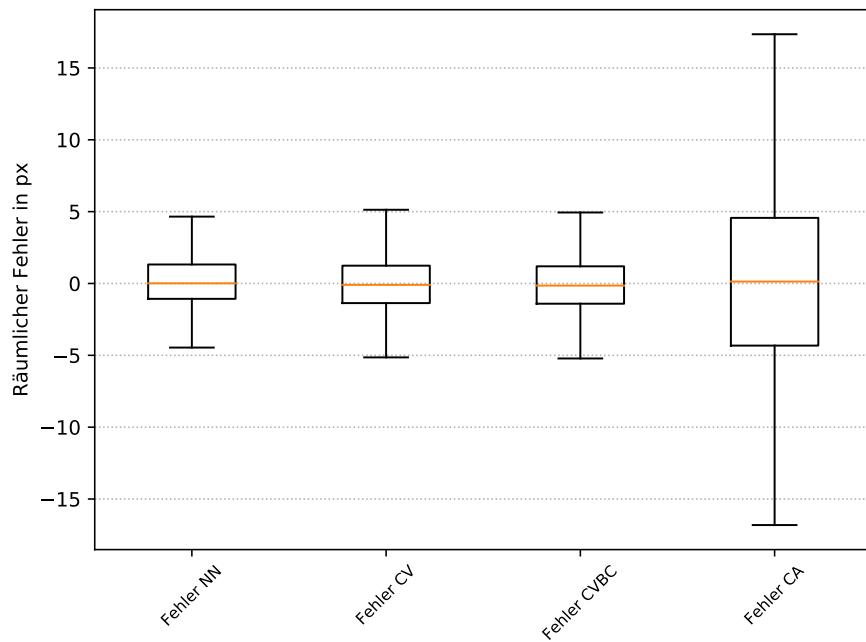


(b) Boxplots des örtlichen Fehlers für die Separator-Ergebnisse der Pfefferkörner aus dem selbst aufgenommenen Datensatz.

Abbildung A.1: Visualisierung der Ergebnisse für die Pfefferkörner aus dem selbst aufgenommenen Datensatz.



(a) Boxplots des zeitlichen Fehlers für die Separator-Ergebnisse der Kugeln aus dem selbst aufgenommenen Datensatz.



(b) Boxplots des örtlichen Fehlers für die Separator-Ergebnisse der Kugeln aus dem selbst aufgenommenen Datensatz.

Abbildung A.2: Visualisierung der Ergebnisse für die Kugeln aus dem selbst aufgenommenen Datensatz.