Entwicklung und Implementierung von Analysemethoden zur Einzelfehlerdetektion in Verbindung mit Deep Learning und Machine Learning

Verfasser: Alexander Pavlovski
Studiengang: Medieninformatik

Prüfer: Prof. Dr.-Ing. Stefan Wagner
Zweitprüfer: Prof. Dr.-Ing. Ullrich Martin
Betreuer: M. Sc. Sebastián Bahamón-Blanco

Stuttgart, den 29. Juni 2020
Erklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tage eingereichte Bachelorarbeit selbstständig verfasst habe, dass ich keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe, dass die eingereichte Bachelorarbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist, dass ich die Bachelorarbeit weder vollständig noch in Teilen bereits veröffentlicht habe und dass das elektronische Exemplar mit den anderen Exemplaren übereinstimmt.

Stuttgart, den 29. Juni 2020

................................................

Unterschrift Alexander Pavlovski

Hinweis

Danksagung


Außerdem möchte ich ebenso meine Prüfer erwähnen, Prof. Dr.-Ing. Stefan Wagner und Prof. Dr.-Ing. Ullrich Martin, die mir die Möglichkeit gaben dieses Thema auszuarbeiten.

Abschließend geht der Dank an meine Freunde und Familie, die mich ebenso immer unterstützt haben.

Stuttgart, den 29. Juni 2020

................................................
Unterschrift Alexander Pavlovski
Inhaltsverzeichnis

Erklärung ................................................................................................................... I
Hinweis ....................................................................................................................... I
Danksagung .................................................................................................................. II
Inhaltsverzeichnis ......................................................................................................... III
Abbildungsverzeichnis ................................................................................................. V
Tabellenverzeichnis ....................................................................................................... VII
Formelverzeichnis ......................................................................................................... VIII
Abkürzungsverzeichnis ................................................................................................. 1

1 Einführung ................................................................................................................ 2
  1.1 Ziele .................................................................................................................... 2
  1.2 Aktueller Stand der Gleiszustandserfassungsmethoden ...................................... 3

2 Fahrzeug-Fahrwegmodell .......................................................................................... 4
  2.1 Messsystem ......................................................................................................... 5

3 PUL-Anfahr ................................................................................................................. 6

4 Künstliche Intelligenz ............................................................................................... 8
  4.1 Machine Learning ............................................................................................... 9
  4.1.1 Decision Tree Modell ..................................................................................... 11
  4.1.2 Random Forest Modell ................................................................................... 11
  4.2 Deep Learning .................................................................................................... 13
  4.2.1 Feed Forward Modell ..................................................................................... 13
  4.3 Java vs. Python .................................................................................................... 16
  4.4 Anwendung in PUL-Anfahr (Java) ..................................................................... 18

5 Implementierung der Analysemethoden ................................................................ 19
  5.1 Vorbereitung ...................................................................................................... 19
  5.1.1 Trainingsdaten ............................................................................................... 19
  5.1.2 Klassifizierung .............................................................................................. 20
  5.1.3 Import der Messdaten .................................................................................... 21
  5.1.4 Datenaufbereitungsmethoden ...................................................................... 22
  5.2 Einzelfehlerdetektion im Beschleunigungssignal: Deep Learning (supervised) ...... 26
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.1</td>
<td>Feed Forward Modell</td>
<td>26</td>
</tr>
<tr>
<td>5.2.2</td>
<td>„Neural Net Builder“</td>
<td>30</td>
</tr>
<tr>
<td>5.3</td>
<td>Einzelfehlerdetektion im Beschleunigungssignal: Machine Learning</td>
<td>33</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Decision-Tree Modell</td>
<td>33</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Random Forest Modell</td>
<td>36</td>
</tr>
<tr>
<td>5.4</td>
<td>Softwarearchitektur der Modelle</td>
<td>39</td>
</tr>
<tr>
<td>5.5</td>
<td>Serialisierung</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>Ergebnisse</td>
<td>41</td>
</tr>
<tr>
<td>6.1</td>
<td>Vergleich aller Modelle</td>
<td>41</td>
</tr>
<tr>
<td>6.2</td>
<td>Empfehlungen</td>
<td>44</td>
</tr>
<tr>
<td>6.3</td>
<td>Java vs. Python Fazit</td>
<td>46</td>
</tr>
<tr>
<td>6.4</td>
<td>Ausblick</td>
<td>46</td>
</tr>
<tr>
<td>7</td>
<td>Literaturverzeichnis</td>
<td>47</td>
</tr>
<tr>
<td>8</td>
<td>Anhang</td>
<td>49</td>
</tr>
</tbody>
</table>
Abbildungsverzeichnis

Abbildung 1: Fahrzeug- und Fahrwegmodell (IEV, 2020) .................................................. 4
Abbildung 2: PUL-Anfahr Bearbeitungsfenster [8] ................................................................. 6
Abbildung 5: Regression und Klassifikation [10] ................................................................. 10
Abbildung 7: Vereinfachte Darstellung eines Decision Trees (Pavlovski, 2020) ............ 11
Abbildung 8: Unterschied Decision Tree und Random Forest [8] ......................................... 12
Abbildung 9: Feed Forward Modell (Pavlovski, 2019) ........................................................ 14
Abbildung 10: Funktionsweise eines Neurons (Pavlovski, 2020) ........................................... 14
Abbildung 13: Prozentuale Verteilung der populärsten Programmiersprachen (Pavlovski 2020, [9]) .................................................................................................................. 16
Abbildung 14: Fehlerklassifizierungstabelle (Pavlovski, 2020) ........................................... 20
Abbildung 15: Visualisierung von einer Runde Messdaten (Pavlovski, 2020) ..................... 20
Abbildung 16: Screenshot Hauptfenster (Pavlovski, 2020) ...................................................... 21
Abbildung 17: Ablaufdiagramm Datenaufbereitung (Pavlovski, 2020) ............................... 22
Abbildung 18: Erklärung der Batchgröße und Schrittweite (Pavlovski, 2020) ..................... 23
Abbildung 19: Vergleich der Fehlermusterlänge (Pavlovski, 2020) ....................................... 25
Abbildung 20: Trainingsfenster Feed Forward Modell (Pavlovski, 2020) .......................... 26
Abbildung 21: Trainingsablauf Feed Forward Modell (Pavlovski, 2020) ............................. 27
Abbildung 22: Beispielausgabe eines Trainingsprozesses mit Feed Forward  (Pavlovski, 2020) ................................................................................................................ 28
Abbildung 23: Analyse Feed Forward Modell ...................................................................... 29
Abbildung 24: Analyse Verlauf Feed Forward (Pavlovski, 2020) ......................................... 29
Abbildung 25: Neural Net Builder (Pavlovski, 2020) ............................................................ 30
Abbildung 26: Ablauf einer Feed Forward Konfiguration mit Hilfe des Neural Net Builders (Pavlovski, 2020) .......................................................... 32
Abbildung 27: Machine Learning Trainingsfenster (Pavlovski,2020) ....................... 33
Abbildung 28: Ablaufdiagramm Decision Tree Training (Pavlovski, 2020) .................. 34
Abbildung 29: Analysefenster Decision Tree (Pavlovski, 2020) ............................... 35
Abbildung 30: Analyseablauf Decision Tree (Pavlovski, 2020) ............................... 35
Abbildung 31: Random Forest Tab (Pavlovski, 2020) ............................................ 36
Abbildung 32: Random Forest Trainingsablauf (Pavlovski, 2020) ......................... 37
Abbildung 33: Random Forest Tab Analyse (Pavlovski, 2020) .............................. 38
Abbildung 34: Analyseablauf (Pavlovski, 2020) .................................................. 38
Abbildung 35: Model-View-Controller Konzept (Pavlovski,2020) ....................... 39
Abbildung 36: Ordnerstruktur Speicherort KI-Modelle (Pavlovski, 2020) ................. 40
Abbildung 37: Random Forest Ergebnis (Pavlovski, 2020) .................................. 41
Abbildung 38: Decision Tree Beispiel Ergebnis (Pavlovski, 2020) ...................... 42
Abbildung 39: Feed Forward Beispiel Ergebnis (Pavlovski, 2020) ...................... 42
Abbildung 40: Training mit Median Speicherung (Pavlovski, 2020) .................... 45
Abbildung 41: Analyseverlauf mit Hilfe des Medians (Pavlovski,2020) ............... 45
Tabellenverzeichnis

Tabelle 1: Fehlerarten am Modell und am echten Gleis [7] ......................................................... 5
Tabelle 2: Unterschiede Java und Python (Pavlovski, 2020) .......................................................... 17
Tabelle 3: Verschiedene Typisierung Java und Python (Pavlovski, 2020) ....................................... 17
Tabelle 4: Einlesen einer Textdatei mit Java und Python (Pavlovski, 2020) ................................. 18
Tabelle 5: Eigenschaften der Trainingsdaten (Pavlovski, 2020) .................................................... 19
Tabelle 6: Charakteristika Signalabschnitt (Pavlovski, 2020) ....................................................... 25
Tabelle 7: Parameter und deren Funktion [13] ............................................................................. 31
Tabelle 8: Mediane aller Fehlerklassen inklusive Gesamtdurchschnitt (Pavlovski, 2020) ......... 44
Formelverzeichnis

Formel 1: Berechnung im Neuron ................................................................. 15
Abkürzungsverzeichnis

| KNN | Künstliches neuronales Net |
| KI  | Künstliche Intelligenz     |
| IEV | Institut für Eisenbahn und Verkehrswesen |
1 Einführung


1.1 Ziele

- Integrierung der beiden Analysemethoden Einzelfehlerdetektion mittels Deep Learning und Einzelfehlerdetektion mittels Machine Learning in die Software PUL-Anfahr:
  - Entwickeln von Methoden zur Datenaufbereitung für den Trainingsprozess der KI-Algorithmen
  - Der Trainingsprozess eines KI Algorithmus soll vollständig in PUL-Anfahr geschehen
  - Implementierung eines internen Speichers für neuronale Netze, „decision trees“ und „random forests“
  - Visualisierung der Analyseergebnisse der beiden KI Algorithmen
- Vergleich der beiden Programmiersprachen Java und Python in Bezug auf diese Problemstellung
1.2 Aktueller Stand der Gleiszustandserfassungsmethoden

Diese Methoden erkennen zwar die Fehler, sind aber an feste Werte gebunden, die die Erkennung limitieren. [2]
Aufgrund dieser Inflexibilität wurde das Themenfeld künstliche Intelligenz mit der Hoffnung aufgegriffen nicht mehr an feste Parameter gebunden zu sein und mit Hilfe von KI-Algorithmen die Fehler automatisch, nach Fehlerkategorie, erkennen zu lassen.
Der Prozess der Fehlererkennung, sei es mit gängigen Analysemethoden wie Kreuzkorrelation oder Analysemethoden in Verbindung stehend mit Künstlicher Intelligenz, umfasst viele Zwischenschritte, um letztendlich einen Fehler im Signal zu erkennen.
2 Fahrzeug-Fahrwegmodell

Für das Testen der entwickelten Algorithmen wurde ein Fahrzeug-Fahrwegmodell aufgebaut, womit stochastische Einflüsse im Beschleunigungssignal sowie Beschleunigungswerte für typische Schienen- und Gleislagefehler erzeugt werden können.

Das Modell (Abbildung 1) ist im Maßstab 1:87 errichtet und besteht aus einer 4,04m langen Teststrecke, die sich aus vier Streckenabschnitten (zwei Kreisbögen und zwei Geraden) zusammensetzt.

Konstruiert ist die Strecke auf Basis einer Feder-Schrauben Konstruktion wie eine Hochbahn, mit der es die Möglichkeit gibt, das Gleis millimetergenau in der Vertikalen zu verschieben. Somit lassen sich verschiedene Gleislagefehler mit unterschiedlichen Fehleramplituden und Wellenlängen erzeugen. [2]

Zur Veranschaulichung zeigt Tabelle 1 den Vergleich zwischen den simulierten Fehlerarten am Modell and an einem echten Gleis.
Das Modell (Abbildung 1) hat an den Positionen 3, 6, 9, 10, 13, 15 und 16 einen Schienenstoß (engl. rail joint).
Hier sind an den Positionen 4 und 5 punktuelle Instabilitäten im frühen Zustand und an Position 7 im finalem Zustand.
Zusätzlich ist an Positionen 11 und 12 jeweils die Auf- und Abfahrt einer Brücke simuliert.
An den Positionen 1 und 8 befinden sich Schienenbrüche.

<table>
<thead>
<tr>
<th>Schienenstoß</th>
<th>Punktuelle Instabilität</th>
<th>Schienenriss/bruch</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
<td><img src="image3" alt="Image" /></td>
</tr>
<tr>
<td><img src="image4" alt="Image" /></td>
<td><img src="image5" alt="Image" /></td>
<td><img src="image6" alt="Image" /></td>
</tr>
<tr>
<td><img src="image7" alt="Image" /></td>
<td><img src="image8" alt="Image" /></td>
<td><img src="image9" alt="Image" /></td>
</tr>
</tbody>
</table>

**Tabelle 1: Fehlerarten am Modell und am echten Gleis** [7]

### 2.1 Messsystem

Das Messsystem bzw. der Messzug besteht aus einer Lokomotive und zwei Flachwagen. Auf dem ersten Flachwagen befindet sich die Batterie und auf dem zweiten der Messsensor. Der Messsensor misst die Vertikalbeschleunigung und leitet diese direkt an MatLab weiter, wo die Visualisierung und die Speicherung stattfinden. [3]
Um eine Varianz der Daten zu erzeugen, wurden zwei feste Startpunkte festgelegt (siehe Abbildung 1) und das Testmodell mit zwei unterschiedlichen Geschwindigkeiten fortbewegt: 0.14 m/s und 0.25 m/s.

3 PUL-Anfahr

Im Rahmen des Studienprojekts 2019 der Fakultät für Informatik, beauftragte das Eisbahninstitut IEV, acht Studenten mit der Entwicklung einer Software zur Bearbeitung und Analyse von Gleismessdaten. Die Bearbeitung dieser Problemstellung sollte in zwei Arbeitsprozesse mit den gegebenen Anforderungen aufgeteilt werden:

1. Bearbeitung und Visualisierung (Abbildung 2):
   - Import von textbasierten Messdaten in den Formaten: CSV, XML, ASCII-Dateien
   - Grafische Visualisierung dieser Messdaten (als Graph)
   - Bearbeitung der Messdaten durch diverse und gängige Signalfiltermethoden wie Tiefpass oder Hochpassfilter
   - Export der bearbeiteten Messdaten

Abbildung 2: PUL-Anfahr Bearbeitungsfenster [8]
2. Analyse und Visualisierung (Abbildung 3)

- Die bearbeiteten Messdaten sollen mit folgenden Analysemethoden analysiert und schließlich visualisiert werden:
  - Bestimmung der Gleislagequalität mittels Leistungsdichtespektrum und Einzelfehlerdetektion im Beschleunigungssignal mittels
  - Einzelfehlerdetektion im Beschleunigungssignal mittels Wavelet-Analyse
  - Bestimmung der Gleisqualität nach Ril 821
  - Einzelfehlerdetektion im Beschleunigungssignal mittels Kreuzkorrelation

Die beiden Analysemethoden, die in Verbindung mit KI stehen, wurden als optionale Anforderung für das Studienprojekt eingestuft und nicht implementiert:
  - Einzelfehlerdetektion im Beschleunigungssignal mittels Deep Learning
  - Einzelfehlerdetektion im Beschleunigungssignal mittels Machine Learning

- Export der Analyseergebnisse

Abbildung 3: PUL-Anfahr Analysefenster [8]
Aufgrund der stetigen Forschung im Gebiet der Mustererkennung in Verbindung mit KI am IEV und der guten erzielten Resultate (s. Kapitel 1.2) ist es mehr als notwendig diese noch fehlenden Module zu entwickeln und die Ergebnisse zu analysieren. Somit bietet das vollständig entwickelte PUL-Anfahr dann nicht nur die Möglichkeit mit den gängigen Analysemethoden zu arbeiten, sondern auch mit DeepLearning- und Machine Learning Modellen, die in PUL-Anfahr komplett trainiert, persistent gespeichert und letztendlich zur Fehlererkennung genutzt werden können.

4 Künstliche Intelligenz


Das Themenfeld Künstliche Intelligenz wird hierbei in weitere Subkategorien, die Abbildung 4 zeigt und in Kapitel 2.1 sowie 2.2 weiter erklärt werden.
4.1 Machine Learning

Machine Learning ist ein großer Teilbereich der künstlichen Intelligenz und fokussiert sich auf Entwicklung von Programmen bzw. Algorithmen, die enorm viele Daten erhalten und aus diesen lernen. Das Hauptziel ist somit, dem Programm beizubringen, möglichst selbstständig, bei gegebenen Problemstellungen, korrekt zu entscheiden und das Problem letztendlich zu klassifizieren und zu lösen.

Hierbei unterscheidet man grob in drei Kategorien:

- *Supervised learning*

  In dieser Kategorie arbeitet man mit sogenannter „labeled data“. Diese Daten sind im Voraus beschriftet und geben dem Algorithmus eine logische Richtung vor. Ziel ist es anschließend ähnliche Datensätze korrekt auswerten zu können.
• *Unsupervised learning*
Hier verwendet man keine beschrifteten Daten („unlabeled data“). Der Algorithmus versucht selbstständig bestimmte Strukturen oder Cluster zu finden.

• *Semi-supervised learning*
Hierunter versteht man eine Kombination aus *supervised learning* und *unsupervised learning*. Hier wird der Algorithmus mit teils beschrifteten Datensätzen trainiert.

Im Zuge dieser Bachelorarbeit werden ausschließlich *supervised* Modelle verwendet, die in den kommenden Kapiteln weiter erklärt werden.
4.1.1 Decision Tree Modell


Abbildung 7: Vereinfachte Darstellung eines Decision Trees (Pavlovski, 2020)

Um nun ein Klassifizierungsproblem damit lösen zu können wird am Wurzelknoten gestartet. Von dort aus geht es abwärts und bei jedem Knoten wird ein Attribut, welches eine Zahl oder Kategorie sein kann, abgefragt. Diesen Ablauf setzt man so lange fort bis ein Blatt, also ein Knoten ohne Nachfolger, erreicht wird. Die Blätter stellen die verschiedenen Klassen dar.

4.1.2 Random Forest Modell

Diese werden zusammengeführt, um eine noch genauere Aussage oder Ergebnis treffen zu können.

Abbildung 8: Unterschied Decision Tree und Random Forest [8]

Abbildung 8 stellt nochmals den genauen Vergleich beider Modelle dar. In diesem Modell existieren zwei Hauptparameter, die ausschlaggebend für die Genauigkeit des Modells sind:

- Anzahl der Bäume im Wald
  Der Name impliziert die genaue Anzahl an Bäumen, die sich in dem Random Forest befinden
- Bagging Größe (Bootstrap aggregating)
  Das Ziel von Bagging ist die Ergebnisse bzw. Vorhersagen eines Baumes im Random Forest Modell zu mitteln, um ein Gesamtmodell mit einer möglichst geringen Varianz zu erhalten [12]
4.2 **Deep Learning**


Solch ein künstliches neuronales Netz ist vom Aufbau vergleichbar mit dem neuronalen Netz im menschlichen Gehirn, bis auf, dass die Neuronen in einem künstlich neuronalen Netz nur schichtweise miteinander verbunden sind. Der Begriff „deep“ impliziert dabei die verschiedenen Schichten eines KNN. Diese Schichten ermöglichen es dem KNN Muster zu erlernen und damit verbundene Probleme zu lösen. [13]

Allgemein besitzt jedes neuronale Netz die Schichten: *input layer, hidden layers und output layer* mit folgenden Funktionen [13]:

1. **Input Layer**: Diese Schicht erhält die initialen Daten
2. **Hidden Layers**: Diese Schichten befinden sich genau zwischen dem input und output Layer. Hier finden die eigentlichen Berechnungen statt.
3. **Output Layer**: Diese Schicht produziert die Ergebnisse.

Abbildung 5 zeigt die verschiedenen Schichten anhand des „Feed Forward“ Modells, was im nächsten Kapitel weiter erklärt wird.

Auch hier existieren verschiedene Modelle mit einem unterschiedlichen Aufbau, von denen das „Feed Forward“ Modell im Zuge dieser Forschungsarbeit weiter erörtert wird.

4.2.1 **Feed Forward Modell**

Abbildung 9: Feed Forward Modell (Pavlovski, 2019)


Abbildung 10: Funktionsweise eines Neurons (Pavlovski, 2020)
Hierbei repräsentieren die Variablen folgendes:

- $x_1, \ldots, x_n$: Input für das nächste Neuron
- $w_1, \ldots, w_n$: Gewicht der Kante
- $b$: bias
- $n$: Anzahl der Inputs aus der vorherigen Schicht
- $i$: Zählvariable
- $f(\ldots)$: Aktivierungsfunktion


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

**Formel 1: Berechnung im Neuron**

Der Gewichtswert $w_1, \ldots, w_n$ der Kanten zwischen zwei Neuronen, ist der jeweils einzige Wert, der durch das Lernen modifiziert und optimiert wird. Nach allen Additionen können Werte in einem unendlichen Zahlenintervall ausgegeben werden, weswegen der Output Wert durch eine Aktivierungsfunktion optimiert werden muss. Die beiden am häufigsten, verwendeten Funktionen sind *Sigmoid* und *tanh*.

![Sigmoid Funktion](https://example.com/sigmoid.png)

**Abbildung 11: Sigmoid Funktion [3]**
Abbildung 11 zeigt die *Sigmoid* Funktion, die einen Wert zwischen 0 und 1 ausgibt. Somit wäre der Anwendungsfall eine Wahrscheinlichkeitsberechnung. [3]

Ist das Ziel auch negative Werte mit einzubeziehen, ist die *tanh* Funktion zu wählen, die Werte zwischen -1 und 1 ausgibt. (Abbildung 12)

![Abbildung 12: tanh Funktion [3]](image)

4.3 **Java vs. Python**

Zum gegenwärtigen Zeitpunkt wurden etliche Programmiersprachen entwickelt, die jeweils besondere Merkmale aufweisen und für unterschiedliche Ziele konzipiert wurden. Java und Python stechen in Bezug auf Popularität jedoch hervor, wie in Abbildung 13 zu sehen ist. [9].

![Abbildung 13: Prozentuale Verteilung der populärsten Programmiersprachen (Pavlovski 2020, [9])]
Obwohl diese beiden Programmiersprachen die populärsten scheinen, bestehen einige grundlegende Unterschiede in deren Aufbau und der Funktionsweise, wie in Tabelle 2 zu erkennen.

<table>
<thead>
<tr>
<th>Übersetzer</th>
<th>Compiler</th>
<th>Interpreter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typisierung</td>
<td>Statisch</td>
<td>Dynamisch</td>
</tr>
<tr>
<td>Programmierparadigma</td>
<td>Objektorientiert</td>
<td>Objektorientiert, aspektorientiert, prozedural</td>
</tr>
<tr>
<td>Plattformunabhängig?</td>
<td>Ja</td>
<td></td>
</tr>
</tbody>
</table>

**Tabelle 2: Unterschiede Java und Python (Pavlovski, 2020)**

Einer der größten Unterschiede, ist die unterschiedliche Typisierung. Java verwendet eine statische und Python eine dynamische Typisierung. Das heißt, dass der Python Compiler die Typ-Prüfung einer Variable, zur Laufzeit vornimmt. Im Gegensatz führt Java dies zum Zeitpunkt der Kompilierung aus.

Resultierend fordert Java bei jeder Deklaration einen Typen, Python nicht (Tabelle 3).

<table>
<thead>
<tr>
<th>Java</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. int a = 4;</td>
<td>1. &gt;&gt;&gt; a = 5</td>
</tr>
<tr>
<td>2. String car = &quot;Nissan&quot;;</td>
<td>2. &gt;&gt;&gt; car = 'Nissan'</td>
</tr>
</tbody>
</table>

**Tabelle 3: Verschiedene Typisierung Java und Python (Pavlovski, 2020)**

Das erleichtert die Lesbarkeit des Codes und hilft dadurch Programmieranfängern Python schneller und einfacher zu lernen. Allerdings bringt das Einbußen in der Performance mit sich, da der Python Interpreter Zeile für Zeile durchgehen muss um den korrekten Variablentyp zu bestimmen [10].

Tabelle 4: Einlesen einer Textdatei mit Java und Python (Pavlovski, 2020)

<table>
<thead>
<tr>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. File file = new File(&quot;test.txt&quot;);</td>
</tr>
<tr>
<td>2. FileReader reader = new FileReader(file);</td>
</tr>
<tr>
<td>3. BufferedReader in = new BufferedReader(reader);</td>
</tr>
<tr>
<td>4. String readline;</td>
</tr>
<tr>
<td>5. while ((readLine = in.readLine()) != null) {</td>
</tr>
<tr>
<td>6. System.out.println(readLine);</td>
</tr>
<tr>
<td>7. }</td>
</tr>
<tr>
<td>8. in.close();</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. myFile = open(&quot;test.txt&quot;)</td>
</tr>
<tr>
<td>2. print myFile.read();</td>
</tr>
</tbody>
</table>

4.4 Anwendung in PUL-Anfahr (Java)

5 Implementierung der Analysemethoden

5.1 Vorbereitung

5.1.1 Trainingsdaten

Insgesamt liegen 16 Messdaten in vier Gruppen aufgeteilt vor, abhängig vom Level der lokalen Instabilität, von der es vier verschiedene Level gibt. Die beiden möglichen Fahrtrichtungen „im Uhrzeigersinn“ und „gegen den Uhrzeigersinn“ sind mit den analogen englischen Begriffen „clockwise“ (CW) und „anticlockwise“ (ACW) gekennzeichnet. (Siehe Tabelle 5)

<table>
<thead>
<tr>
<th>Dateiname</th>
<th>Anzahl Runden</th>
<th>Fahrtrichtung</th>
<th>Geschwindigkeit</th>
<th>Startpunkt</th>
<th>Level (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>trainingData0_1</td>
<td>5</td>
<td>CW</td>
<td>Minimum</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>trainingData0_2</td>
<td>7</td>
<td>CW</td>
<td>Maximum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trainingData0_3</td>
<td>6</td>
<td>ACW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trainingData0_4</td>
<td>6</td>
<td>CW</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>trainingData1_1</td>
<td>7</td>
<td>CW</td>
<td>Minimum</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>trainingData1_2</td>
<td>9</td>
<td>CW</td>
<td>Maximum</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>trainingData1_3</td>
<td>7</td>
<td>ACW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trainingData1_4</td>
<td>8</td>
<td>CW</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>trainingData2_1</td>
<td>10</td>
<td>CW</td>
<td>Minimum</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>trainingData2_2</td>
<td>12</td>
<td>CW</td>
<td>Maximum</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>trainingData2_3</td>
<td>11</td>
<td>ACW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trainingData2_4</td>
<td>9</td>
<td>CW</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>trainingData3_1</td>
<td>12</td>
<td>CW</td>
<td>Minimum</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>trainingData3_2</td>
<td>14</td>
<td>CW</td>
<td>Maximum</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>trainingData3_3</td>
<td>8</td>
<td>ACW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trainingData3_4</td>
<td>13</td>
<td>CW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>144</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tabelle 5: Eigenschaften der Trainingsdaten (Pavlovski,2020)

Bei 144 Runden ergibt das 2016 Fehlermuster, die zum Training zur Verfügung stehen.
5.1.2 Klassifizierung

In jeder Messdatei werden die Fehlermuster mit Hilfe des Brushing Tools in MatLab markiert und letztendlich klassifiziert. Abhängig von der Fehlerart, wird die entsprechende Fehlerklassennummer an den jeweiligen Index geschrieben.

<table>
<thead>
<tr>
<th>Klasse</th>
<th>Fehlerart</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Bewegung</td>
</tr>
<tr>
<td>1</td>
<td>Schienenstoß</td>
</tr>
<tr>
<td>2</td>
<td>Schienenbruch</td>
</tr>
<tr>
<td>3</td>
<td>Periodischer Schienenfehler</td>
</tr>
<tr>
<td>4</td>
<td>Punktuelle Instabilität im frühen Zustand (verändert)</td>
</tr>
<tr>
<td>5</td>
<td>Punktuelle Instabilität im finalen Zustand</td>
</tr>
<tr>
<td>6</td>
<td>Auf und Abfahrt Brücke</td>
</tr>
<tr>
<td>7</td>
<td>Gleisagefehler</td>
</tr>
<tr>
<td>8</td>
<td>Punktuelle Instabilität im frühen Zustand (unverändert)</td>
</tr>
</tbody>
</table>

Abbildung 14: Fehlerklassifizierungstabelle (Pavlovski, 2020)


Abbildung 15: Visualisierung von einer Runde Messdaten (Pavlovski, 2020)
Abbildung 15 zeigt die aufgezeichneten Vertikalbeschleunigungen von einer Runde auf der Teststrecke in blau und die Klassifizierung in orange. Die Fehlerklasse liest man, wie auch die Vertikalbeschleunigung, an der y-Achse ab.

5.1.3 Import der Messdaten

Alle Messdaten für das Training und das Testen werden über das schon implementierte Importmodul importiert. Da die Trainingsdaten als MatLab Datei vorliegen und doppelspaltig sind, musste die Klasse `MatLabImporter.java` erweitert werden und importiert nun die Messdaten und die Klassifizierung als separate Signale. Das gibt dem Nutzer die Möglichkeit die rohen Messdaten zu bearbeiten oder zu filtern, um einen besseren Trainingserfolg zu erzielen. Zusätzlich, aufgrund der identischen Spaltennamen der Trainingsdaten, werden an diese in chronologischer Reihenfolge, Zahlen angehängt, damit der Import fehlerfrei verläuft und die Rohdaten sowie die dazugehörigen Klassifizierungsdaten einander zugeordnet werden können.

Abbildung 16: Screenshot Hauptfenster (Pavlovski, 2020)

Um den Trainingsprozess zu starten befindet sich rechts im Operationsframe ein neu hinzugefügter Tab „AI Training“, in dem sich die Buttons „DeepLearning“ und „MachineLearning“ befinden, die jeweils das entsprechende AI-Training Fenster öffnen.
5.1.4 Datenaufbereitungsmethoden

Ein wichtiger Schritt des Lernprozesses ist die Datenaufbereitung bzw. die Datennuestrukturierung damit das jeweilige Modell die Daten möglichst gut und effizient verstehen kann. Um diese Aufgabe kümmert sich die Klasse `AIPreProcessingModule.java`. Diese fungiert somit als weiterer Layer zwischen dem Import der Daten und dem eigentlichen Lernprozess im jeweiligen AI-Moduls (s. Abbildung 17).

![Ablaufdiagramm Datenaufbereitung](image)

*Abbildung 17: Ablaufdiagramm Datenaufbereitung (Pavlovski, 2020)*

Die Aufspaltung der Datenaufbereitung und des Lernprozesses in verschiedene Klassen bringt den Vorteil, dass die Modelle mit unterschiedlichen Darstellungen der Trainingsdaten lernen können, die alle aus derselben Java Klasse kommen, um die beste Datenstruktur für das jeweilige Modell zu finden.
Methode 1: Statistische Darstellung von Signalabschnitten

Die erste Methode fügt die vorliegenden Messdaten zu einer großen Datei zusammen, die anschließend in mehrere, gleich große Batches aufgeteilt wird. Der Aufteilungsprozess ist von der Batchgröße und der Schrittweite ("movestep") abhängig (s. Abbildung 18: Erklärung der Batchgröße und Schrittweiten).

Abbildung 18: Erklärung der Batchgröße und Schrittweite (Pavlovski, 2020)
Die Batchgröße stellt hierbei die Breite des Fensters da, was über das Signal wandert und nach jedem „movestep“ den aktuell betrachteten Signalabschnitt ausschneidet. Wird ein Wert für „movestep“ gewählt, der kleiner ist als die Batchgröße, überlappen sich die Fenster und man erhält somit mehr Trainingsdaten und eine eventuell bessere Genauigkeit des Modells. Werden jetzt nur die Vertikalbeschleunigungen an ein Modell weitergeleitet, bieten diese Daten zu wenig singuläre Eigenschaften damit ein erfolgreicher Trainingsprozess stattfinden kann. Aus diesem Grund bietet es sich an die Eigenschaften eines Fehlermusters auszuwerten, wie beispielsweise Durchschnitt oder Standardabweichung [7]. Nimmt man weitere Eigenschaften hinzu erreicht man eine hinreichende Charakterisierung (s. Tabelle 6: Charakteristika Signalabschnitt).

<table>
<thead>
<tr>
<th>Anzahl</th>
<th>Eigenschaft</th>
<th>Erklärung &amp; Java Methode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Durchschnitt</td>
<td>Der Durchschnitt aller Werte berechnet</td>
</tr>
<tr>
<td>========</td>
<td></td>
<td>Durch AIPreprocessingModule.calcMean()</td>
</tr>
<tr>
<td>1</td>
<td>Quadratisches Mittel</td>
<td>Das quadratische Mittel aller Werte berechnet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>durch AIPreprocessingModule.calcRMS()</td>
</tr>
<tr>
<td>1</td>
<td>Standardabweichung</td>
<td>Die Standardabweichung aller Werte berechnet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>durch AIPreprocessingModule.calcStandardDeviation()</td>
</tr>
<tr>
<td>8</td>
<td>Spektrale Hochpunkte</td>
<td>Auf die gegebenen Werte wird eine Frequenzanalyse durchgeführt und davon die vier größten Punkte genommen</td>
</tr>
<tr>
<td>1</td>
<td>Hauptkomponentenanalyse</td>
<td>Der größte Wert der Hauptkomponentenanalyse wird durch AIPreProcessingModule.calcPCA() berechnet</td>
</tr>
<tr>
<td>1</td>
<td>Schiefe</td>
<td>Beschreibt die Art und Stärke der Asymmetrie aller Werte berechnet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>durch AIPreProcessingModule.calcSkewness()</td>
</tr>
<tr>
<td>1</td>
<td>Varianz</td>
<td>Die Varianz der gegebenen Werte berechnet</td>
</tr>
<tr>
<td></td>
<td></td>
<td>durch AIPreProcessingModule.calcVariance()</td>
</tr>
<tr>
<td>1</td>
<td>Erstes Moment</td>
<td>Das erste Moment der gegebenen Werte durch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AIPreProcessingModule.calcFirstMoment()</td>
</tr>
<tr>
<td>1</td>
<td>Zweiter Moment</td>
<td>Das zweite Moment der gegebenen Werte durch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AIPreProcessingModule.calcSecondMoment()</td>
</tr>
</tbody>
</table>
Tabelle 6: Charakteristika Signalabschnitt (Pavlovski, 2020)

<table>
<thead>
<tr>
<th>1</th>
<th>Dritter Moment</th>
<th>Das dritte Moment der gegebenen Werte durch AIPreProcessingModule.calcThirdMoment()</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Vierter Moment</td>
<td>Das vierte Moment der gegebenen Werte durch AIPreProcessingModule.calcFourthMoment()</td>
</tr>
</tbody>
</table>

Insgesamt ergeben sich dadurch 19 statistische Eigenschaften, durch die ein ganzer Signalabschnitt dargestellt werden kann. Der Ausschneideprozess generiert pro herausgeschnittenem Fehlermuster ein AIDataVector Objekt mit den schon berechneten statistischen Eigenschaften des Abschnitts. Diese werden im letzten Schritt an das jeweilige Modell weitergeleitet. (Abbildung 18)

Methode 2: Statistische Darstellung der einzelnen Fehlermuster

5.1.4.2


Abbildung 19: Vergleich der Fehlermusterlänge (Pavlovski, 2020)
So bietet es sich auch hier an nicht die Vertikalbeschleunigungen an das Modell weiter zu geben, sondern die statistische Darstellung, die im vorherigen Kapitel erklärt wurde, zu verwenden.

Der Ausschneideprozess generiert pro herausgeschnittenem Fehlermuster ein \textit{AIFailureVector} Objekt mit den vertikalen Beschleunigungswerten. Diese werden in einem \textit{AIFailureVectorList} Objekt gesammelt. Danach werden die \textit{AIFailureVector} Objekte in \textit{AIDataVector} Objekte umgewandelt, die die zwölf statistischen Eigenschaften beinhalten.

Der letzte Schritt ist erneut die Weitergabe an das jeweilige Modell (Abbildung 18).

5.2 Einzelfehlerdetektion im Beschleunigungssignal: Deep Learning (supervised)

5.2.1 Feed Forward Modell

Das Feed Forward Modell kann über den Operationstab \textit{Deep Learning} geöffnet werden und es erscheint das, in Abbildung 20, zu sehende Fenster. Links bietet sich die Möglichkeit die Signale mit den Vertikalbeschleunigungen samt Klassifizierungssignalen auszuwählen und rechts die in Kapitel 5.1.4 vorgestellten Datenaufbereitungsmethoden durch Checkboxen.

Hinzu kommt der Button \textit{Neural Net Builder}, der eine Benutzeroberfläche für die Konfigurierung eines neuronalen Netzes öffnet. (s. Kapitel 5.2.2)

\[<\text{Abbildung 20: Trainingsfenster Feed Forward Modell (Pavlovski, 2020)}>\]
Abbildung 21: Trainingsablauf Feed Forward Modell (Pavlovski, 2020)
Abbildung 21 zeigt einen möglichen Ablauf des Trainingsprozesses. Der Nutzer kann aus schon konfigurierten Modellen wählen oder über den Neural Net Builder eine neue Konfiguration festlegen (Kapitel „Neural Net Builder“ 5.2.2). Anschließend kann aus den in Kapitel 5.1.4 erklärten Datenaufbereitungsmethoden gewählt und mit dem Betätigen des Apply Buttons letztendlich der Trainingsprozess gestartet werden.

Bei erfolgreichem Trainingsprozess erscheint im Output Fenster von PUL-Anfahr neben der Confusion-Matrix folgende Auswertungskriterien (s. Abbildung 22):

1. **Accuracy**  
   Genauigkeit des Modells, d.h. das Verhältnis von korrekten klassifizierten Vorhersagen zu allen Vorhersagen

2. **Precision**  
   Verhältnis von den korrekten Vorhersagen zu den insgesamt vom Modell als korrekt gekennzeichnete Vorhersagen

3. **Recall (deutsch: Trefferquote)**  
   Verhältnis von korrekten klassifizierten Vorhersagen zu allen Vorhersagen in der jeweiligen Klasse

4. **F1 Score**  
   Verhältnis von Recall und Precision

**Abbildung 22: Beispielausgabe eines Trainingsprozesses mit Feed Forward**  
(Pavlovski, 2020)
Außerdem kann auch eine Warnung ausgegeben werden, die beschreibt, ob Klassen überhaupt vorhergesagt wurden oder nicht.

Der Analyseprozess, der über den -Button erreichbar ist, gleicht den anderen Modellen, wo man links aus dem zu analysierenden Signal und rechts das gewünschte Modell, batchsize und movestep auswählt, wie in Abbildung 23 zu erkennen.

Abbildung 23: Analyse Feed Forward Modell

Abbildung 24 zeigt den Analyseverlauf.

Abbildung 24: Analyse Verlauf Feed Forward (Pavlovski, 2020)
5.2.2 „Neural Net Builder“


Der Neural Net Builder besteht aus fünf verschiedenen Hauptkomponenten (Abbildung 25):

1. **Architecture - Komponente**
   Hier kann bisher nur aus einem Deep Learning Modell gewählt werden: *Feed Forward*

2. **Network Layer – Komponente**
   Neben der Möglichkeit die Anzahl der Zwischenschichten (*hidden layer*) einzustellen, können hier globale Parameter für alle Schichten gesetzt werden. Oft sind diese pro Schicht dieselben, weswegen es sich anbietet eine schnelle Einstellung bereit zu stellen. Damit die Werte für die jeweilige Schicht übernommen werden, muss der Nutzer auf den **Save**-Button klicken in der Layer – Komponente klicken.

![Abbildung 25: Neural Net Builder (Pavlovski, 2020)]
3 Layer – Komponente

4 Template – Komponente
Diese Komponente füllt alle Felder mit einer Beispielkonfiguration, die anschließend bearbeitet und gespeichert werden kann.

5 Layer builder – Komponente

<table>
<thead>
<tr>
<th><strong>NN Schicht Parameter</strong></th>
<th><strong>Funktion</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Layer type</strong></td>
<td>Typ der Schicht</td>
</tr>
<tr>
<td><strong>Updater</strong></td>
<td>Optimierungsfunktion: Verändert interne Parameter, wie Gewichte, um die Qualität der Voraussagen zu verbessern</td>
</tr>
<tr>
<td><strong>Seed</strong></td>
<td>Zufällig gewählte Zahl, die die Reproduzierbarkeit gewährleistet</td>
</tr>
<tr>
<td><strong>Learning rate</strong></td>
<td>Die Lernrate des Modells</td>
</tr>
<tr>
<td><strong>Weight initialization</strong></td>
<td>Gewichtsinitialisierungsfunktion: Initialisiert die Gewichtswerte</td>
</tr>
<tr>
<td><strong>Activation function</strong></td>
<td>Aktivierungsfunktion</td>
</tr>
<tr>
<td><strong>Regularization</strong></td>
<td>Gewichtsregulierung was overfitting vorbeugt</td>
</tr>
<tr>
<td><strong>Regularization rate</strong></td>
<td>Der Wert der Gewichtsregulierung</td>
</tr>
<tr>
<td><strong>Loss Function (nur bei Output Layer)</strong></td>
<td>Die Verlustfunktion</td>
</tr>
</tbody>
</table>

Tabelle 7: Parameter und deren Funktion [13]
Der gesamte Prozess zur korrekten Konfiguration ist Abbildung 26 zu entnehmen.

Abbildung 26: Ablauf einer Feed Forward Konfiguration mit Hilfe des Neural Net Builders (Pavlovski, 2020)
5.3 Einzelfehlerdetektion im Beschleunigungssignal: Machine Learning

5.3.1 Decision-Tree Modell

Wie in Kapitel 5.1.3 erwähnt, gelangt man über den „MachineLearning“ Button im AITraining Tab in das Decision-Tree Trainingsfenster (s. Abbildung 27).

Links sind alle importierten Messdaten zu erkennen inklusive Klassifizierung. Auf der rechten Seite darf sich der Nutzer entscheiden welche Methode zur Datenaufbereitung (Kapitel 4.1.4) er verwenden möchte. Hier existiert keine Möglichkeit Methode 1 auszuwählen.

Bricht der Trainingsprozess ab wird der Fehler im Output-Fenster ausgegeben und es kann versucht werden einen neuen Prozess zu starten. Ist das Training erfolgreich wird das Modell persistent gespeichert (s. Kapitel 5.4) und das Fenster geschlossen.

Das gespeicherte Modell kann nun über das Aufrufen vom MachineLearning Analysefenster über **Analyse ➔ MachineLearning** verwendet werden, wie in Abbildung 29 zu erkennen.
Abbildung 29: Analysefenster Decision Tree (Pavlovski, 2020)

Links kann ein Signal zur Analyse mit dem dazugehörigen Modell aus der Decision Tree Liste ausgewählt werden. Es besteht die Auswahl die batchsize und movestep zu wählen oder die Werte des trainierten Modells zu übernehmen. Mit der anschließenden Betätigung des - Buttons startet die Analyse mit der Visualisierung. Der genau Ablauf kann Abbildung 30 entnommen werden.

Abbildung 30: Analyseablauf Decision Tree (Pavlovski, 2020)
5.3.2 Random Forest Modell

Im Machine Learning Trainingsfenster befindet sich ebenfalls der „Random Forest“ Tab, wie Abbildung 31 zeigt.

Abbildung 31: Random Forest Tab (Pavlovski, 2020)

Der Aufbau ist sehr ähnlich wie beim Decision Tree Model, bis auf die hinzukommenen Inputfelder:

- *number of subtrees:*
  Anzahl der Bäume im Wald (engl. *forest*)
- *bagging size*
  Wird hier ein Wert ungleich 0 genommen, wird *bagging* eingeschaltet und dieser Wert repräsentiert die Größe eines jeden bags bzw. die Größe der Teilmenge, die aus den Gesamtdaten zufällig herausgenommen wird (s. Kapitel 4.1.2)
- *probability limit*
  Dieser Parameter gibt die Wahrscheinlichkeit wieder, ob ein Knoten, indem ein Vergleich von Attributen stattfindet, übersprungen wird.
Nach der Parametereingabe für das Modell an sich, kann erneut aus zwei möglichen Datenaufbereitungsmethoden gewählt werden. Methode 1 kann aus demselben Grund nicht angewandt werden, wie in Kapitel 4.3.2 erklärt. Der genaue Ablauf kann Abbildung 31 entnommen werden.

Abbildung 32: Random Forest Trainingsablauf (Pavlovski, 2020)

Der Trainingsablauf ist derselbe wie beim Decision Tree Modell, bis auf die zusätzlichen hinzugekommenen Parameter. Ist das Modell trainiert, kann wie auch bei den anderen Analysemethoden, über das Analyse Hauptfenster und über den Button MachineLearning der Random Forest Tab aufgerufen werden, wie in Abbildung 33 zu erkennen. Ebenso existiert erneut die Möglichkeit die batchsize und den movestep selbst zu wählen, oder dieselben Werte, mit denen das Modell trainiert wurde, zu verwenden.
Abbildung 33: Random Forest Tab Analyse (Pavlovski, 2020)

Der Aufbau des Random Forest Tabs gleicht dem Aufbau des Decision Tree Tabs. Der Ablauf, wie man ein Modell benutzt, ist ebenfalls derselbe (Abbildung 34). Das garantiert einen routinierten Prozess für den Nutzer, unabhängig davon, welches Modell zur Analyse ausgewählt wird.

Abbildung 34: Analyseablauf (Pavlovski, 2020)
5.4 Softwarearchitektur der Modelle

PUL-Anfahr wurde nach dem MVC (Model-View-Controller) Konzept aufgebaut, mittels dessen versucht wurde eine Software in ihre logischen Bestandteile aufzuteilen. Die Abkürzung MVC impliziert hierbei die folgenden Komponenten:

- **Model**
  Verwaltung diverser interner Datenstrukturen
- **View**
  Zuständig für die Benutzeroberfläche sowie Benutzereingaben
- **Controller**
  Zuständig für die Programmlogik und dient als Verbindungsstück zwischen View und Model

Abbildung 35 zeigt das Modell mit Interaktion eines Benutzers.

Abbildung 35: Model-View-Controller Konzept (Pavlovski, 2020)

Die zusätzlich hinzugekommenen Module wurden vollständig in dieses Konzept integriert, was nach wie vor die leichte Erweiterung oder Verbesserung von PUL-Anfahr garantiert. Darüber hinaus wurde das Grundkonzept, von Trennung der Bearbeitung und Analyse der Daten, vollständig aufrechterhalten.
Das Training der Modelle erfolgt im Bearbeitungsteil von PUL-Anfahr neben den anderen Bearbeitungsmethoden, wie in Kapitel 5.1.3 erwähnt. Nach dem Trainingsprozess kann die tatsächliche Analyse nur über den Analyse Button aufgerufen werden, wo sich auch alle anderen Analysemethoden befinden. Neben der ersichtlichen Trennung auf der Benutzeroberfläche, existiert die Trennung auch in der Paketstruktur und alle Datenaufbereitung, Trainings- und Analyseklassen befinden sich im jeweils zugehörigen Paket. Um die AI-Module noch besser zu erkennen, hat jede AI-Klasse einen Prefix „AI“.

5.5 Serialisierung

Neben der eigentlichen Implementierung der Analysemethoden, ist ein wichtiger Teil die persistente Speichermöglichkeit aller Modelle. Dazu wurde die Klasse `PersistenceManager.java` entsprechend erweitert, um dieser Aufgabe gerecht zu werden. Abbildung 36 zeigt die Ordnerstruktur wohin jeweils die Modelle gespeichert werden. Demnach befinden sich die trainierten Modelle im `resources` Ordner, wo sich auch alle anderen von PUL-Anfahr benutzten Ressourcen befinden.

Abbildung 36: Ordnerstruktur Speicherort KI-Modelle (Pavlovski, 2020)

Zudem wurde pro Modell eine eigene Klasse erstellt, die nicht nur die trainierten Modelle speichert, sondern auch alle Eigenschaften, mit welchen das Modell trainiert wurde.
6 Ergebnisse

6.1 Vergleich aller Modelle


Abbildung 37: Random Forest Ergebnis (Pavlovski, 2020)

Im Zuge dieser Bachelorarbeit wurden folgende Parameter für das Random Forest Modell als die Geeignetsten empfunden:

- Batchsize = 128
- Movestep = 128
- Anzahl der Bäume: 128
- Bagging Größe: 64
- Wahrscheinlichkeitslimit: 0.5

Abbildung 38: Decision Tree Beispiel Ergebnis (Pavlovski, 2020)

Diese Fehlvorhersagen sind mit ○ und □ in Abbildung 38 gekennzeichnet. 
Folgende Parameter wurden als die Geeignetsten empfunden:

- Batchsize = 128
- Movestep = 128


Abbildung 39: Feed Forward Beispiel Ergebnis (Pavlovski, 2020)
Im Zuge dieser Arbeit stellten sich folgende Parameter als passend dar:

- **Layer 1** (input layer):
  - Layertyp: DenseLayer
  - Anzahl Input Neuronen: 19 (Anzahl der statistischen Eigenschaften)
  - Anzahl Output Neuronen: 17
  - Activation: TANH
  - Weight Init: XAVIER
  - Updater: Adam
  - Learning Rate: 0.01

- **Layer 2** (hidden layer):
  - Layertyp: DenseLayer
  - Anzahl Input Neuronen: 17 (Anzahl der statistischen Eigenschaften)
  - Anzahl Output Neuronen: 17
  - Activation: TANH
  - Weight Init: XAVIER
  - Updater: Adam
  - Learning Rate: 0.01

- **Layer 3** (hidden layer):
  - Layertyp: DenseLayer
  - Anzahl Input Neuronen: 17 (Anzahl der statistischen Eigenschaften)
  - Anzahl Output Neuronen: 17
  - Activation: TANH
  - Weight Init: XAVIER
  - Updater: Adam
  - Learning Rate: 0.01

- **Layer 4** (output layer):
  - Layertyp: Output Layer
  - Anzahl Input Neuronen: 17 (Anzahl der statistischen Eigenschaften)
  - Anzahl Output Neuronen: 9
  - Activation: SIGMOID
  - Weight Init: XAVIER
  - Updater: Adam
  - Learning Rate: 0.01
  - Verlustfunktion: RECONSTRUCTION.CROSSENTROPY
6.2 Empfehlungen


<table>
<thead>
<tr>
<th>Fehlerklasse</th>
<th>Fehlerart</th>
<th>Median der Fehler</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Schienenstoß</td>
<td>168</td>
</tr>
<tr>
<td>2</td>
<td>Schienenbruch</td>
<td>153</td>
</tr>
<tr>
<td>3</td>
<td>Periodischer Schienenfehler</td>
<td>223</td>
</tr>
<tr>
<td>4</td>
<td>Punktuelle Instabilität im frühen Zustand (verändert)</td>
<td>155</td>
</tr>
<tr>
<td>5</td>
<td>Punktuelle Instabilität im finalen Zustand</td>
<td>200</td>
</tr>
<tr>
<td>6</td>
<td>Auf und Abfahrt Brücke</td>
<td>370</td>
</tr>
<tr>
<td>7</td>
<td>Gleislagefehler</td>
<td>129</td>
</tr>
<tr>
<td>8</td>
<td>Punktuelle Instabilität im frühen Zustand (unverändert)</td>
<td>108</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ø 188,25</td>
</tr>
</tbody>
</table>

Tabelle 8: Mediane aller Fehlerklassen inklusive Gesamtdurchschnitt (Pavlovski, 2020)

Die größten Genauigkeiten erreichen die Modelle, wenn die batchsize (Kapitel 5.1.4), beim Aufteilen der Daten, die Durchschnittslänge aller Fehlerklassen annimmt. Daraus kann geschlossen werden, dass es sich lohnen könnte jedes Feilermuster herauszuschneiden, dies über statistische Eigenschaften anders darstellen und damit das gewählte Modell zu trainieren.
Im Datenaufbereitungsprozess werden zusätzlich alle Mediane der jeweiligen Fehlerklasse gespeichert.

Abbildung 40: Training mit Median Speicherung (Pavlovski, 2020)

Im anschließenden Analyseprozess läuft das trainierte Modell insgesamt acht Mal (abhängig von der Anzahl der Fehlerklassen) mit dem zugehörigen Median das zu analysierende Signal ab und sammelt alle Wahrscheinlichkeiten, die das Modell ausgibt. Als letzter Schritt werden die besten Wahrscheinlichkeiten der jeweiligen Fehlerklasse aus jedem Durchlauf extrahiert und daraus eine Klassifikation geschaffen. Diese Herangehensweise sollte ein noch genaueres Modell liefern. Abbildung 41 zeigt den Ablauf.

Abbildung 41: Analyseverlauf mit Hilfe des Medians (Pavlovski, 2020)
6.3 Java vs. Python Fazit


6.4 Ausblick

- Mit dem heutigen Stand von PUL-Anfahrt hat der Benutzer die Möglichkeit ein Feed Forward Modell mit allen von der Bibliothek „deeplearning4j“ bereitgestellten Einstellungsmöglichkeiten über eine eigenständig entwickelte Benutzeroberfläche zu nutzen. Darüber hinaus sind die beiden Module Decision Tree und Random Forest vollständig implementiert.

- Die Bibliothek, „deeplearning4j“, bietet neben dem Feed Forward Modell, das Recurrent Model und Convolutional Model an, die zukünftig in den Neural Net Builder integriert werden könnten.

- Um die Modellgenauigkeit weiter zu verbessern, wäre es zu überdenken mehr Messdaten zu erheben und diese ans Modell weiterzugeben. Insbesondere mehr Fehlermuster von lokalen Instabilitäten sammeln, damit die Modelle mehr Fehlermuster von dieser Fehlerkategorie nutzen können.
7 Literaturverzeichnis


### 8 Anhang

<table>
<thead>
<tr>
<th>Parameter neuronales Netz</th>
<th>Mögliche Parametereinstellungen</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Layer type</strong></td>
<td>DenseLayer, OutputLayer</td>
</tr>
<tr>
<td><strong>Updater</strong></td>
<td>SGD, ADAM, ADAMAX, ADADELTA, NESTEROVS, NADAM, ADAGRAD, RMSPROP, NONE, CUSTOM</td>
</tr>
<tr>
<td><strong>Seed</strong></td>
<td>123, 123456</td>
</tr>
<tr>
<td><strong>Dropout</strong></td>
<td>0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.0</td>
</tr>
<tr>
<td><strong>Learning rate</strong></td>
<td>0.00001, 0.0001, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0, 3.0, 10.0</td>
</tr>
<tr>
<td><strong>Weight initialization</strong></td>
<td>DISTRIBUTION, ZERO, ONES, SIGMOID_UNIFORM, NORMAL, LECUN_NORMAL, UNIFORM, XAVIER, XAVIER_UNIFORM, XAVIER_FAN_IN, XAVIER_LEGACY, RELU, RELU_UNIFORM, IDENTITY, LECUN_UNIFORM, VAR_SCALING_NORMAL_FAN_IN, VAR_SCALING_NORMAL_FAN_OUT, VAR_SCALING_NORMAL_FAN_AVG, VAR_SCALING_UNIFORM_FAN_IN, VAR_SCALING_UNIFORM_FAN_OUT, VAR_SCALING_UNIFORM_FAN_AVG</td>
</tr>
<tr>
<td><strong>Activation function</strong></td>
<td>CUBE, ELU, HARDSIGMOID, HARTANH, IDENTITY, LEAKYRELU, RATIONALTANH, RELU, RELU6, RRELU, SIGMOID, SOFTMAX, SOFTPLUS, SOFTSIGN, TANH, RECTIFIEDTANH, SELU, SWISH, THRESHOLDEDRELU, GELU</td>
</tr>
<tr>
<td><strong>Regularization</strong></td>
<td>L1, L2, NONE</td>
</tr>
<tr>
<td><strong>Regularization rate</strong></td>
<td>0.00001, 0.0001, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0, 3.0, 10.0</td>
</tr>
<tr>
<td><strong>Loss Function</strong></td>
<td>MSE, L1, EXPLL, XENT, MCXENT, RMSE_XENT, SQUARED_LOSS, RECONSTRUCTION_CROSSENTROPY</td>
</tr>
<tr>
<td>NEGATIVELOGLIKELIHOOD, CUSTOM,</td>
<td></td>
</tr>
<tr>
<td>COSINE_PROXIMITY, HINGE,</td>
<td></td>
</tr>
<tr>
<td>SQUARED_HINGE, KL_DIVERGENCE,</td>
<td></td>
</tr>
<tr>
<td>MEAN_ABSOLUTE_ERROR, L2,</td>
<td></td>
</tr>
<tr>
<td>MEAN_ABSOLUTE_PERCENTAGE_ERROR,</td>
<td></td>
</tr>
<tr>
<td>MEAN_SQUARED_LOGARITHMIC_ERROR,</td>
<td></td>
</tr>
<tr>
<td>POISSON, WASSERSTEIN</td>
<td></td>
</tr>
</tbody>
</table>