Bachelorarbeit

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

Vorgelegt an der Universität Stuttgart

stud E-2020/07

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

Der Inhalt der vorliegenden Abschlussarbeit stellt ausnahmslos die persönliche Ansicht von Alexander Pavlovski dar und muss nicht der des Instituts entsprechen. Eine Empfehlung des Instituts im Sinne des Inhalts und der in dieser Arbeit enthaltenen Schlussfolgerungen kann aus diesem Grund nicht zwingend abgeleitet werden.

Danksagung

Ich möchte einen großen Dank an meinen Betreuer M. Sc. Sebastian Bahamón-Blanco aussprechen, der mich stets mit großer Hilfe und Geduld durch die gesamte Thesis begleitet hat.

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

Abbildungsverzeichnis

Tabellenverzeichnis

Formelverzeichnis

Abkürzungsverzeichnis

KNN Künstliches neuronales Net

KI Künstliche Intelligenz

IEV Institut für Eisenbahn und Verkehrswesen

1 Einführung

In Deutschland beförderte die Deutsche Bahn AG 148 Millionen Fahrgäste und erreichte eine Gesamtbetriebsleistung von 1,09 Milliarden Trassenkilometern. Die Fahrgastanzahl stieg hier um 4% im Vergleich zum Vorjahr [1]. Dieser Trend lässt vermuten, dass in den kommenden Jahren der Zugverkehr weiter zunehmen wird, so auch aber auch die Belastung der Schienenwege. Aus diesem Grund ist es maßgeblich die Instandhaltung dieser weiter zu verbessern, um einen reibungslosen Schienenverkehr zu gewährleisten. Dazu existiert die Möglichkeit die Vertikalbewegung bzw. Vertikalbeschleunigung der Achse eines Zuges auf dem Gleis zu messen und darauf Fehler zu erkennen. Damit der Prozess der Fehlererkennung aus dem Signal vereinfacht wird, beauftragte das Institut für Eisenbahn- und Verkehrswesen im Sommersemester 2019 an der Universität Stuttgart, über das Studienprojekt der Fakultät Informatik, acht Studenten damit, eine Software zu entwickeln, die diesem Problem abhilft. Das Ergebnis ist die Software PUL-Anfahr, welche aus zwei Prozessabläufen besteht: Aufbereitung und Analyse der Messdaten. In dem Analyseteil geschieht die tatsächliche Fehlererkennung. Jede Gleisinstabilität folgt einem bestimmten Muster im Signal. Jedes dieser Fehlermuster gilt es zu erkennen. Insbesondere hatten Machine Learning Algorithmen, ein Teilbereich der Künstlichen Intelligenz, in den letzten Jahren starke Fortschritte im Themengebiet Mustererkennung zu verzeichnen. Somit bietet es sich an solche Algorithmen in PUL-Anfahr zu integrieren, die in dieser Bachelorarbeit implementiert und erörtert werden.

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

Zur Erkennung von punktuellen Instabilitäten durch Signalverarbeitungsmethoden wurde eine umfassende Forschungsarbeit verfasst, die als Ergebnis die Methoden Frequenzanalyse, Kreuzkorrelation, Wavelet Analyse, Analyse nach Amplitudengröße und Analyse basierend auf Amplitudengröße und Wellenlänge nannte, die erfolgreich die Fehler an der korrekten Position an dem Fahrzeug-Fahrweg Modell erkannte. [2]
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.
MatLab [3] bietet dafür seit 2018 die Möglichkeit KI-Modelle, ob aus der Kategorie Machine Learning und Deep Learning (s. Kapitel 4), zu erstellen und zu trainieren. Darauf basierend entstand ein „Recurrent Modell“ [3] , also ein Modell aus der Deep Learning Familie sowie das Modell „Bagged Decision Trees“ [4], aus der Machine Learning Familie. Beide Modelle erreichten gute Ergebnisse in dem die acht vorhandenen Fehlerklassen in drei Kategorien aufgeteilt wurden und damit eine Genauigkeit von 84% bzw. 88% beim *Bagged Decision Tree* Modell und 97% beim *Recurrent Modell* erreichten. [3] [4]
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.
Aus diesem Grund entstand in Verbindung mit den Studienprojekten am Institut für Informatik und dem IEV zwei Softwares PUL-Anfahr (s. Kapitel 3) und PULTrack. Diese stellt eine Datenbank zur Bahnkörpermodellierung bereit. Die Intention ist beide Anwendungen zusammenzuführen, um einen Gesamtablauf von Erzeugung eines Bahnkörpers, Kostenberechnung.

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.

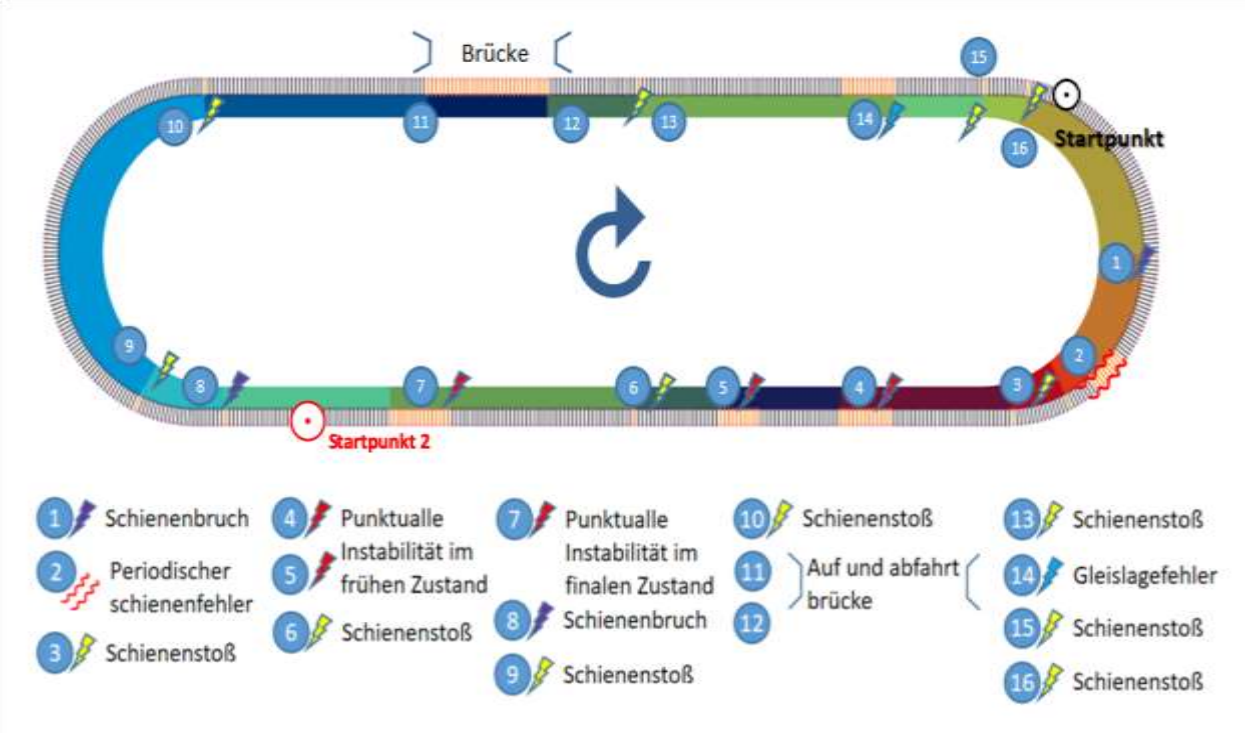


Abbildung 1: Fahrzeug -und Fahrwegmodell (IEV, 2020)

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.

Schienenstoß	Punktuelle Instabiltät	Schienenriss/bruch
		
[4]	[5]	[6]
		
[7]	[4]	[7]
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.

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 Messensor 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 0.25m/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

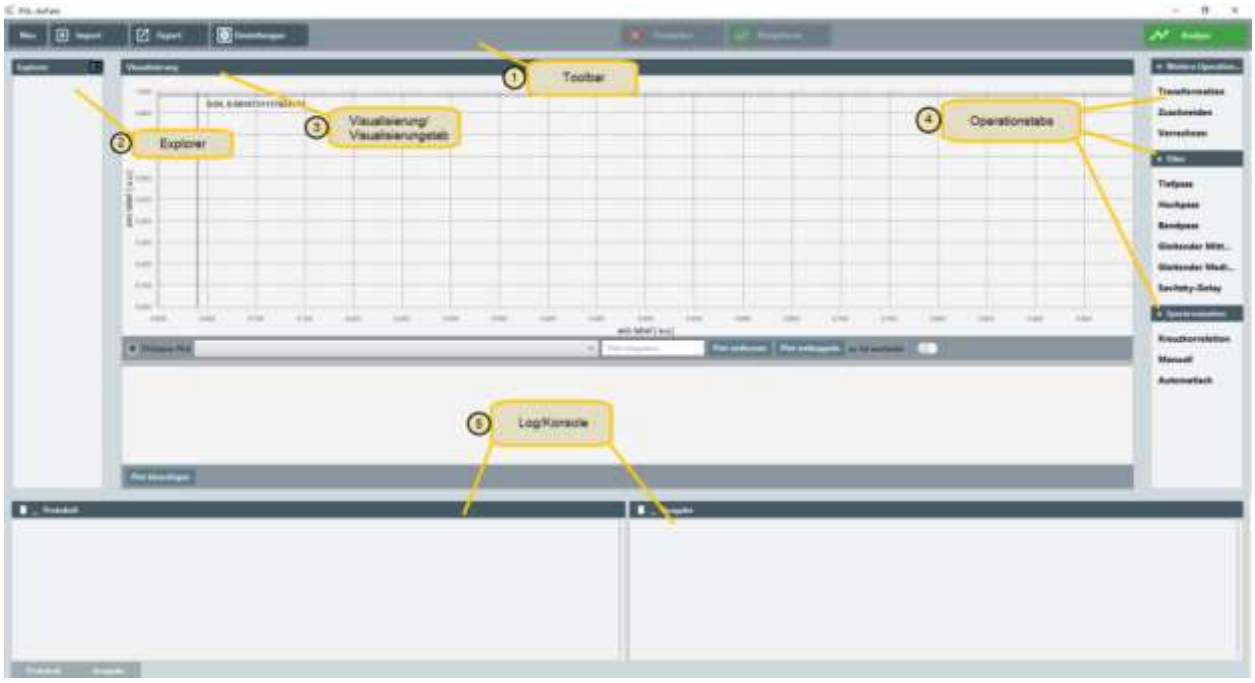


Abbildung 2: PUL-Anfahr Bearbeitungsfenster [8]

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 Kreuzkor-relation

  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

Künstliche Intelligenz (engl. artificial intelligence) ist ein Teilbereich der Informatik, bei dem es um maschinelles Lösen von Problemstellungen geht.
Hierbei soll die Maschine bzw. der Algorithmus im Allgemeinen, möglichst korrekt, externe Daten interpretieren, von diesen lernen und die Lernerfolge verwenden, um ein bestimmtes Ziel zu erreichen. Somit ist der Anwendungsbereich ziemlich breit gefächert und findet vor allem Anklang im Themenfeld Bilderkennung. Darunter fällt Erkennung von Schrift oder Erkennung von bestimmten Objekten im Bild. Ein gutes Beispiel ist hierfür die Gesichtserkennung durch Überwachungskameras. Innerhalb von Sekunden kann ein Gesicht erkannt und mit einer Datenbank abgeglichen werden. [9] Dieses Fallbespiel weist jedoch eine gewisse Komplexität auf, da für einen korrekten Abgleich der Gesichter sehr viele Parameter nötig sind und es dementsprechend schwieriger ist ein Muster zu finden bzw. zu erkennen.

Anders sieht es bei der Mustererkennung in einem einfacheren zweidimensionalen Signal aus. Hier sind die zu klassifizierten Daten deutlich weniger komplex und es wird resultierend weniger Rechenleistung oder Speicherkapazität gebraucht. Das ist der Grund warum dieses Themenfeld auch Zuspruch im Eisenbahnsektor bekommt.

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

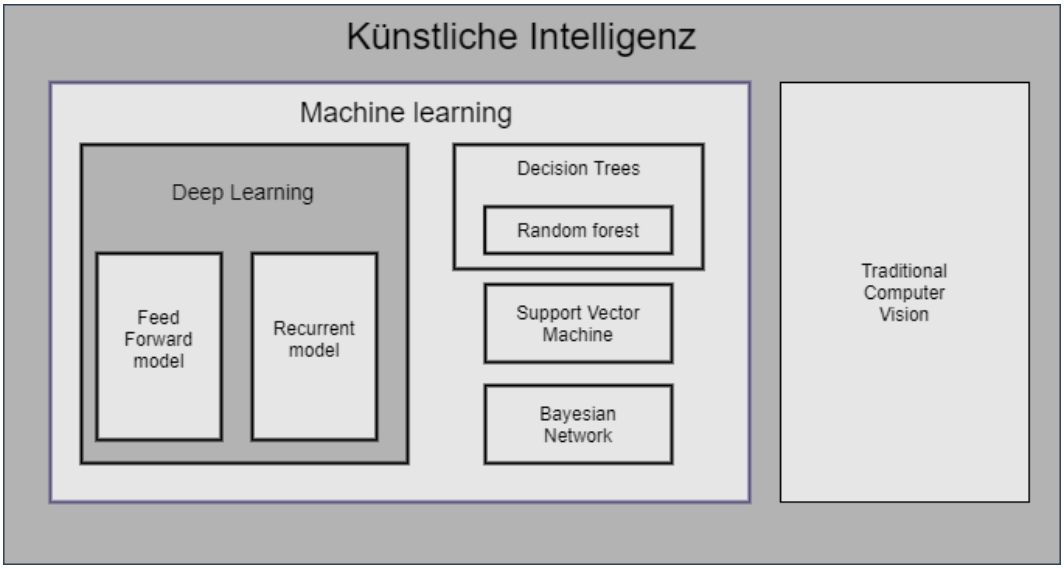


Abbildung 4: Künstliche Intelligenz Komponenten [3]

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.

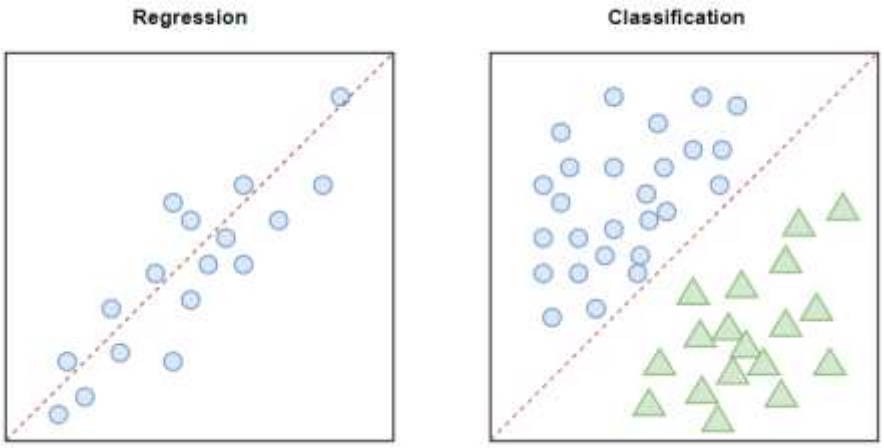


Abbildung 5: Regression und Klassifikation [10]

- *Unsupervised learning*

Hier verwendet man keine beschrifteten Daten („unlabeled data“). Der Algorithmus versucht selbstständig bestimmte Strukturen oder Cluster zu finden.

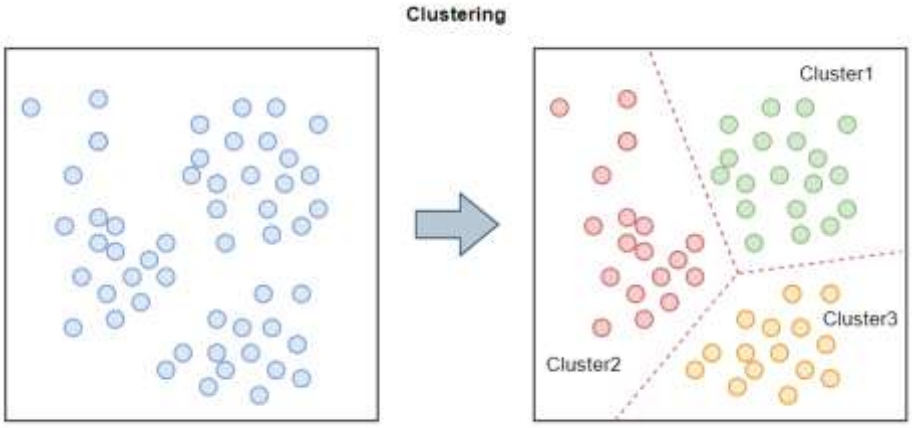


Abbildung 6: Clustering Methode [11]

- *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

Das „Decision Tree“ Modell gehört zu den *Machine Learning* Methoden und löst Regressions als auch Klassifizierungsprobleme. Der Name impliziert den Aufbau des Algorithmus auf Basis einer Baumstruktur. Abbildung 7 stellt dies vereinfacht dar.

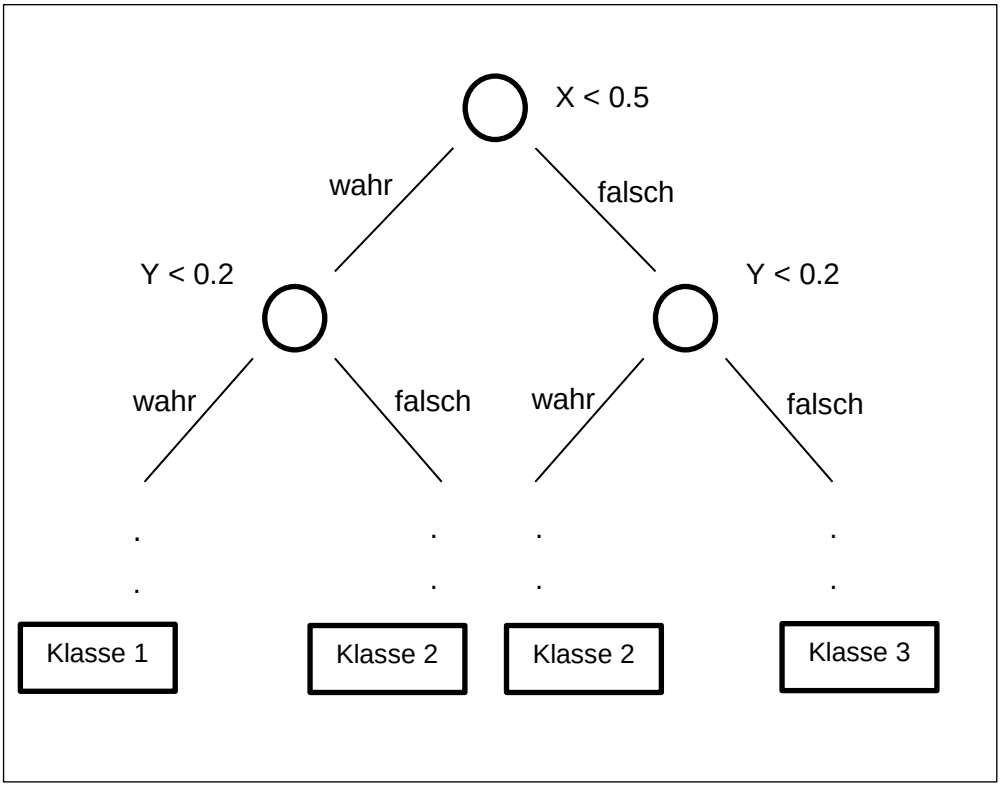


Abbildung 7: Vereinfachte Darstellung eines Decision Trees (Pavlovski, 2020)

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

Dieses Modell gehört ebenso zu den *Machine Learning* Methoden und löst Regressions als auch Klassifizierungsprobleme. Der Name impliziert einen „zufälligen Wald“. Tatsächlich besteht dieser „Wald“ aus mehreren *Decision Trees*, die im vorherigen Kapitel erörtert wurden.

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

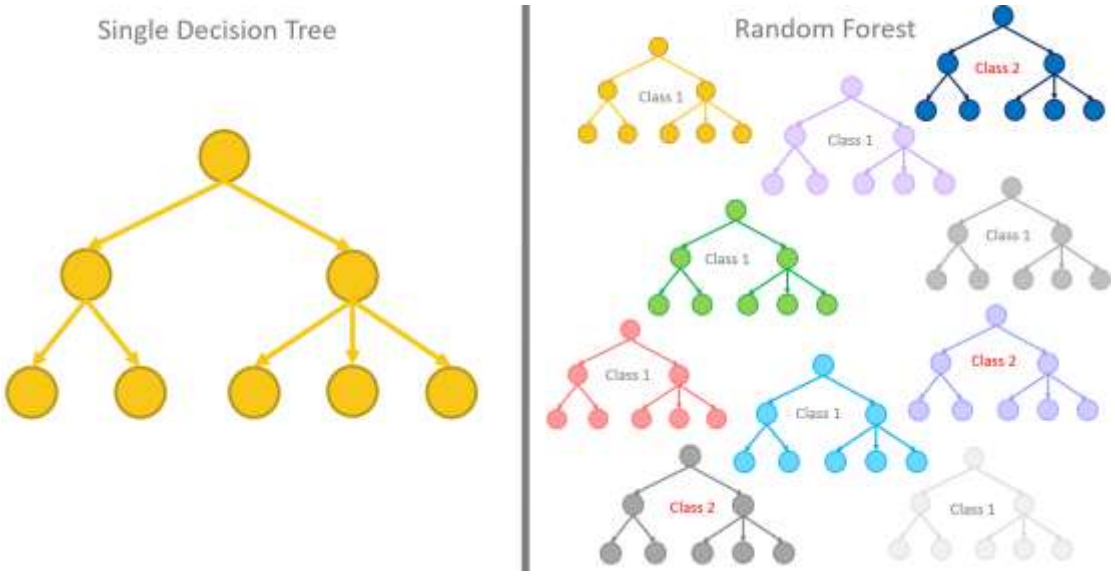


Abbildung 8: Unterschied Decision Tree und Random Forest [8]

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 Ergebisse 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

Deep Learning ist ein Teilbereich des maschinellen Lernens (engl. Machine Learning) und der künstlichen Intelligenz, wobei mit Hilfe von Algorithmen, die der Struktur und Funktionsweise des menschlichen Gehirns ähneln (künstliche neuronale Netze), insbesondere „kognitive Probleme“ gelöst werden können.
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

Diese Form von Modell ist einer der einfacheren aus der „Deep Learning“ Familie und impliziert mit seinem Namen „Feed Forward“ (deutsch: vorwärts) eine Bewegung der Daten nur in eine Richtung, wie in Abbildung 9 zu sehen. Die Informationen wandern somit nur vom *input layer* zum *output layer.* [13]

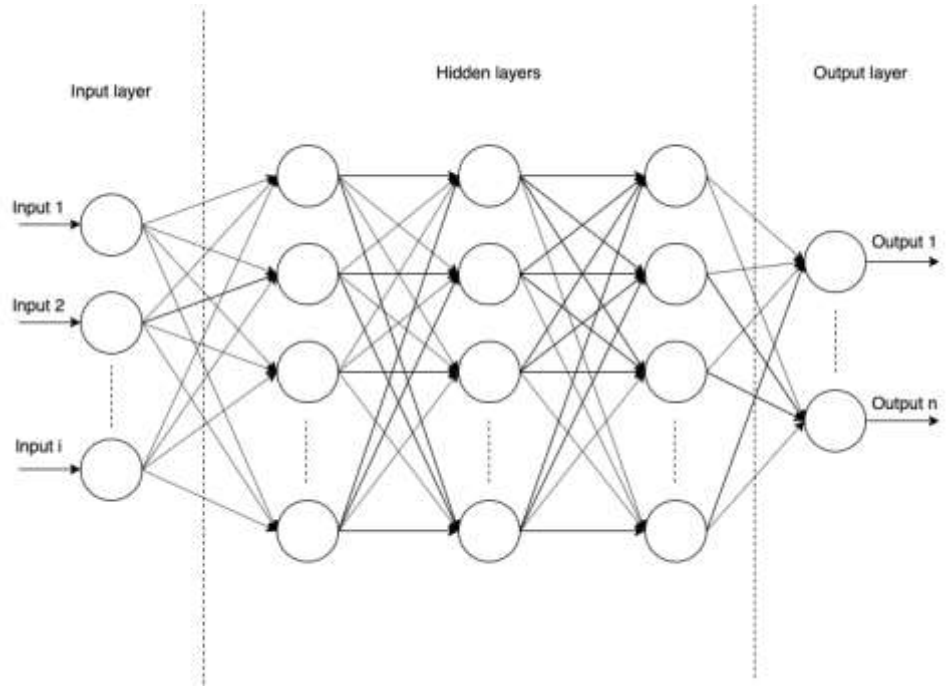


Abbildung 9: Feed Forward Modell (Pavlovski, 2019)

Jede Schicht besitzt mehrere Knoten in denen Berechnungen stattfinden, vergleichbar mit einem Neuron im Gehirn. Abbildung 10 zeigt die genaue Funktionsweise eines Neurons in diesem Modell.

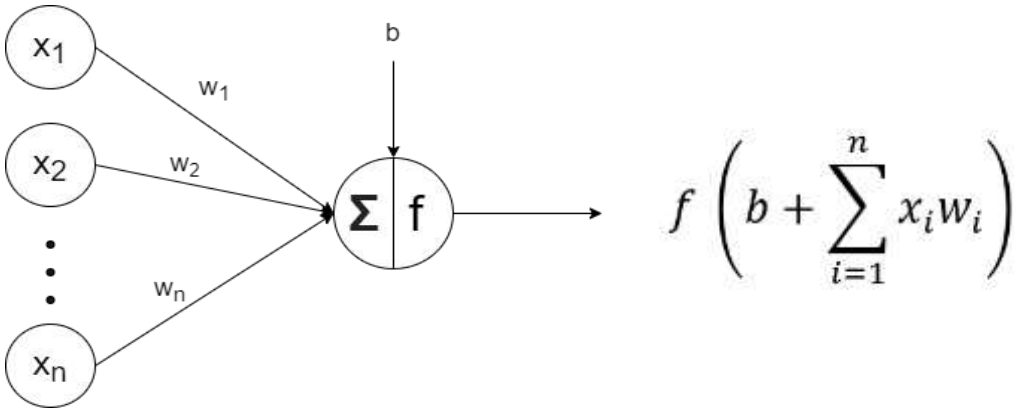


Abbildung 10: Funktionsweise eines Neurons (Pavlovski,2020)

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

Abbildung 10 enthält drei (Input)Neuronen aus einer Schicht, die mit einem Neuron aus der nächsten Schicht verbunden sind. Jeder Inputwert wird mit dem Gewicht der entsprechenden Kante multipliziert und anschließend zu den noch fehlenden Multiplikationen addiert, wie in Formel 1 dargestellt. Hinzu kommt noch die Addition einer zusätzlichen Variable *bias*. Das Modell kann nicht alle Faktoren in die Berechnung einbeziehen, weswegen diese Variable alle nicht beobachtbaren Bestände darstellt. [13]

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

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*.

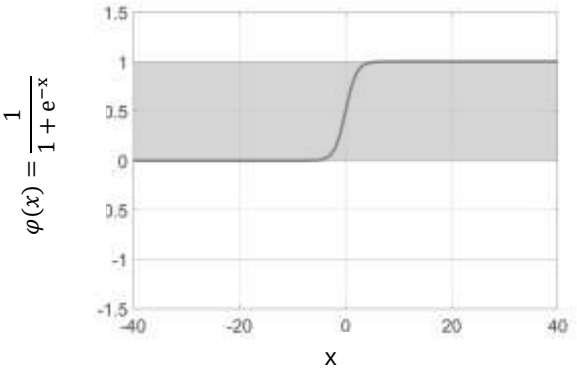


Abbildung 11: Sigmoid Funktion [3]

Abbildung 11 zeigt die *Sigmoid* Funktion was 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)

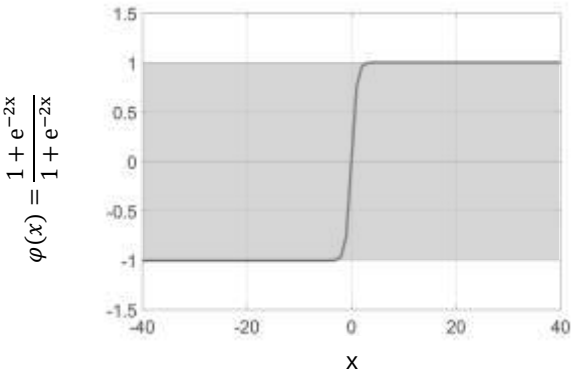


Abbildung 12: tanh Funktion [3]

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].

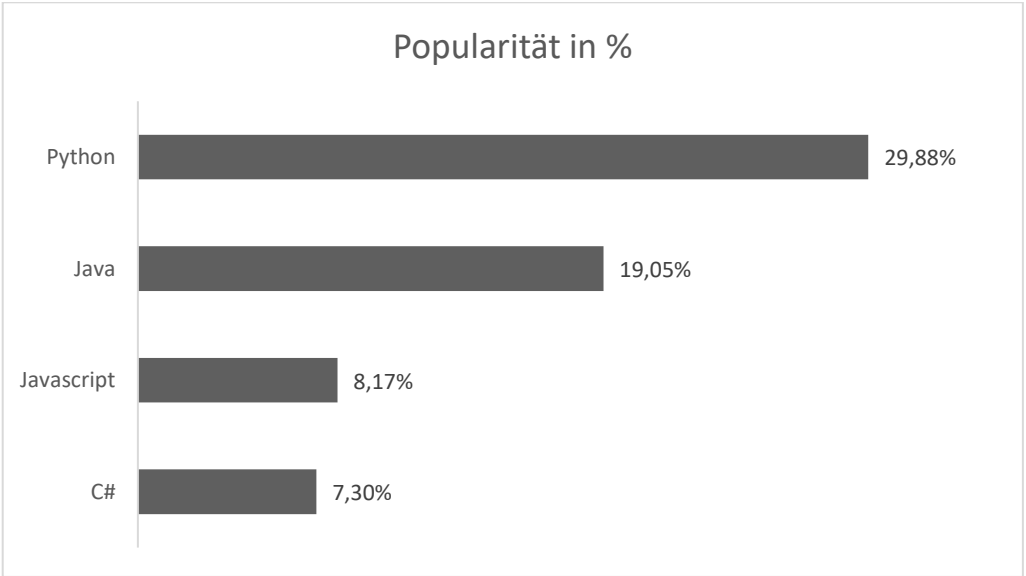


Abbildung 13: Prozentuale Verteilung der populärsten Programmiersprachen (Pavlovski 2020, [9])

Obwohl diese beiden Programmiersprachen die populärsten scheinen, bestehen einige grundlegende Unterschiede in deren Aufbau und der Funktionsweise, wie in Tabelle 2 zu erkennen.

	Java	**Python**
Übersetzer	Compiler	Interpreter
Typisierung	Statisch	Dynamisch
Programmierparadigma	Objektorientiert	Objektorientiert, aspektorientiert, prozedural
Plattformunabhängig?	Ja	

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).

Java	**Python**
1. `int a = 4;` 2. `String car = "Nissan";`	1. `>>> a = 5` 2. `>>> car = 'Nissan`

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].
In Bezug auf *Machine Learning*, wo ein großer Teil darin besteht große Datensätze einzulesen und anzupassen, ist die Nutzung von Python vorteilhaft. In Tabelle 4 ist das Einlesen einer Textdatei, einmal in Java und in Python, dargestellt. Wo Java für das Ausgeben einer Textdatei sieben Zeilen Code benötigt, braucht Python gerade einmal zwei Zeilen. (Tabelle 4).

Java	Python
1. File file = new File("test.txt"); 2. FileReader reader = new FileReader(file); 3. BufferedReader in = new BufferedReader(reader); 4. 5. String readLine; 6. 7. while ((readLine = in.readLine()) != null) { 8. 9. System.out.println(readLine); 10. } 11. 12. in.close();	1. myFile = open("test.txt") 2. print myFile.read();

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

4.4 Anwendung in PUL-Anfahr (Java)

Die Implementierung von PUL-Anfahr erfolgte in Java. Resultierend sind der Deep Learning Algorithmus sowie der Machine Learning Algorithmus ebenso in Java implementiert. Als zusätzliche Bibliothek wurde PUL-Anfahr mit „deeplearning4j“ und „quickdt“ ergänzt. „deeplearning4j“ ist eine freie, plattformübergreifende Programmbibliothek für künstliche Intelligenz bzw. maschinelles Lernen und läuft unter der Apache 2.0 Lizenz [11].
„quickdt“ läuft unter der GNU Lesser General Public License version 3 Lizenz und wurde seit 2011 entwickelt. [12]

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)

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

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.

Klasse	Fehlerart
0	Bewegung
1	Schienenstoß
2	Schienenbruch
3	Periodischer Schienenfehler
4	Punktuelle Instabilität im frühen Zustand (verändert)
5	Punktuelle Instabilität im finalen Zustand
6	Auf und Abfahrt Brücke
7	Gleislagefehler
8	Punktuelle Instabilität im frühen Zustand (unverändert)

Abbildung 14: Fehlerklassifizierungstabelle (Pavlovski, 2020)

Jede klassifizierte Datei beinhaltet danach in der ersten Spalte, mit dem Spaltenamen „raw“, die tatsächlichen Messdaten und in der zweiten Spalte, mit dem Spaltennamen „cla“, die Klassifizierung.

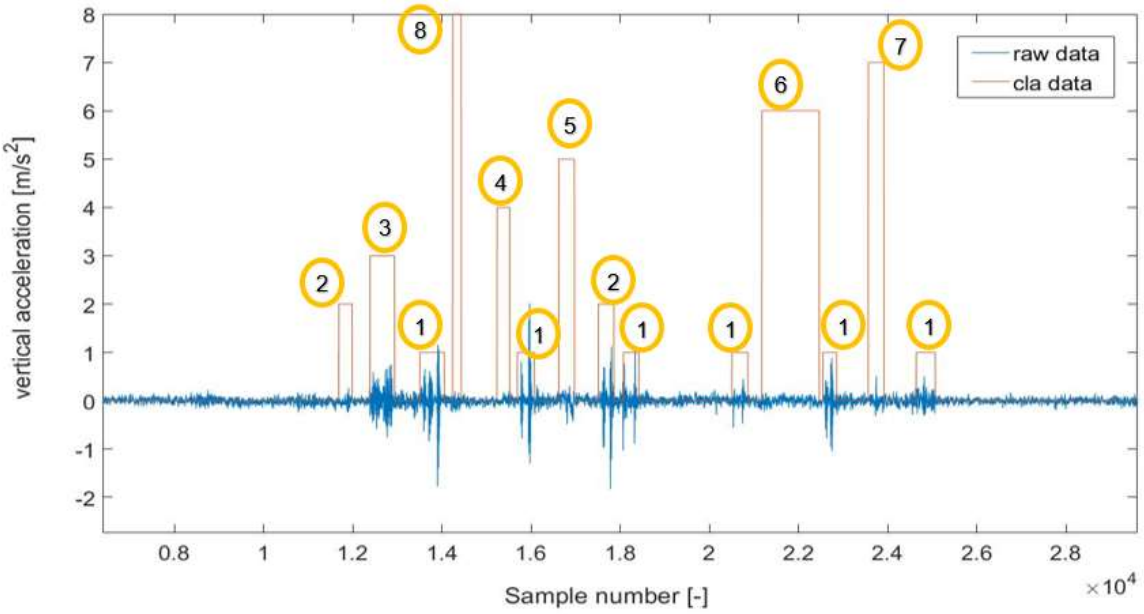


Abbildung 15: Visualisierung von einer Runde Messdaten (Pavlovski, 2020)

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.

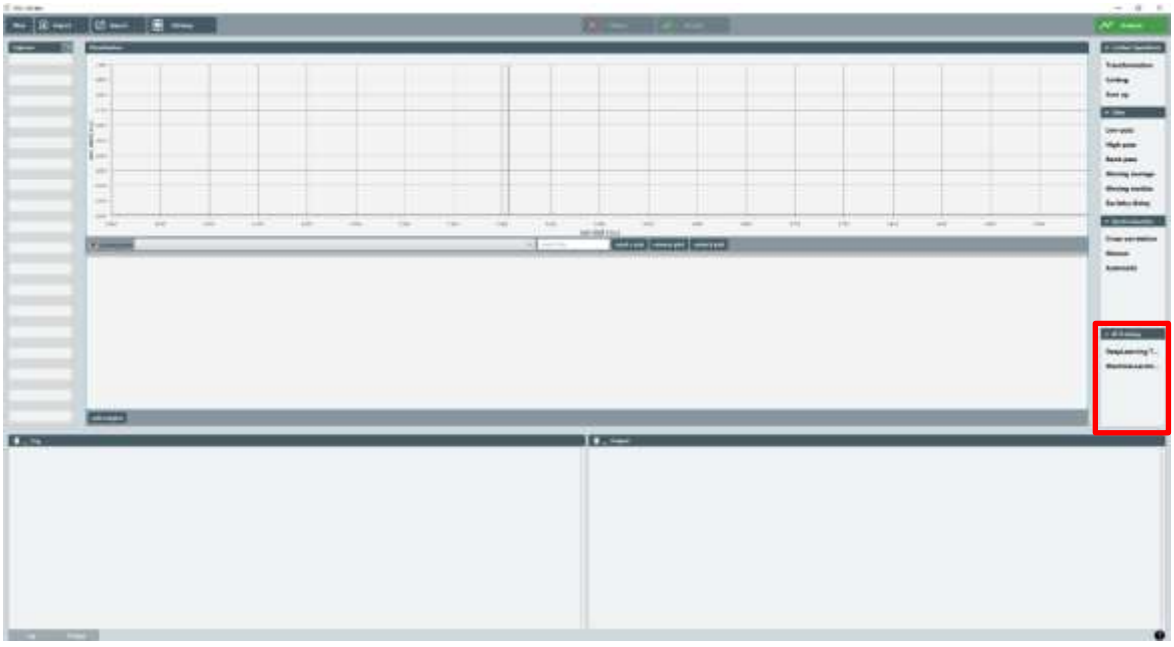
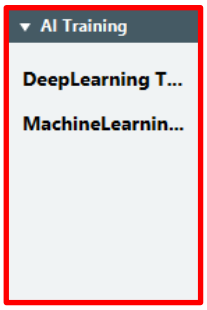


Abbildung 16: Screenshot Hauptfenster (Pavlovski, 2020)



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 Datenneustrukturierung 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).

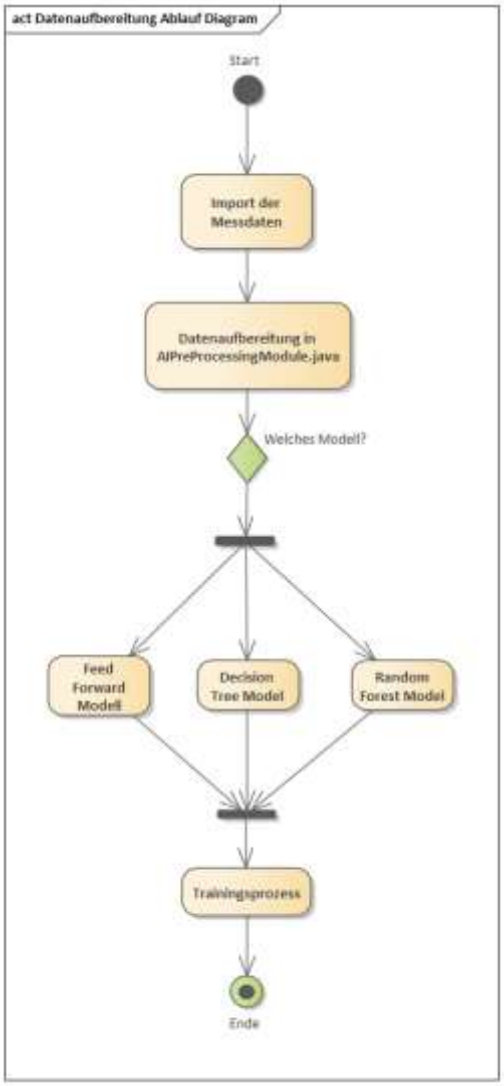


Abbildung 17: Ablaufdiagramm Datenaufbereitung (Pavlovski, 2020)

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

5.1.4.1

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 Schrittweite).

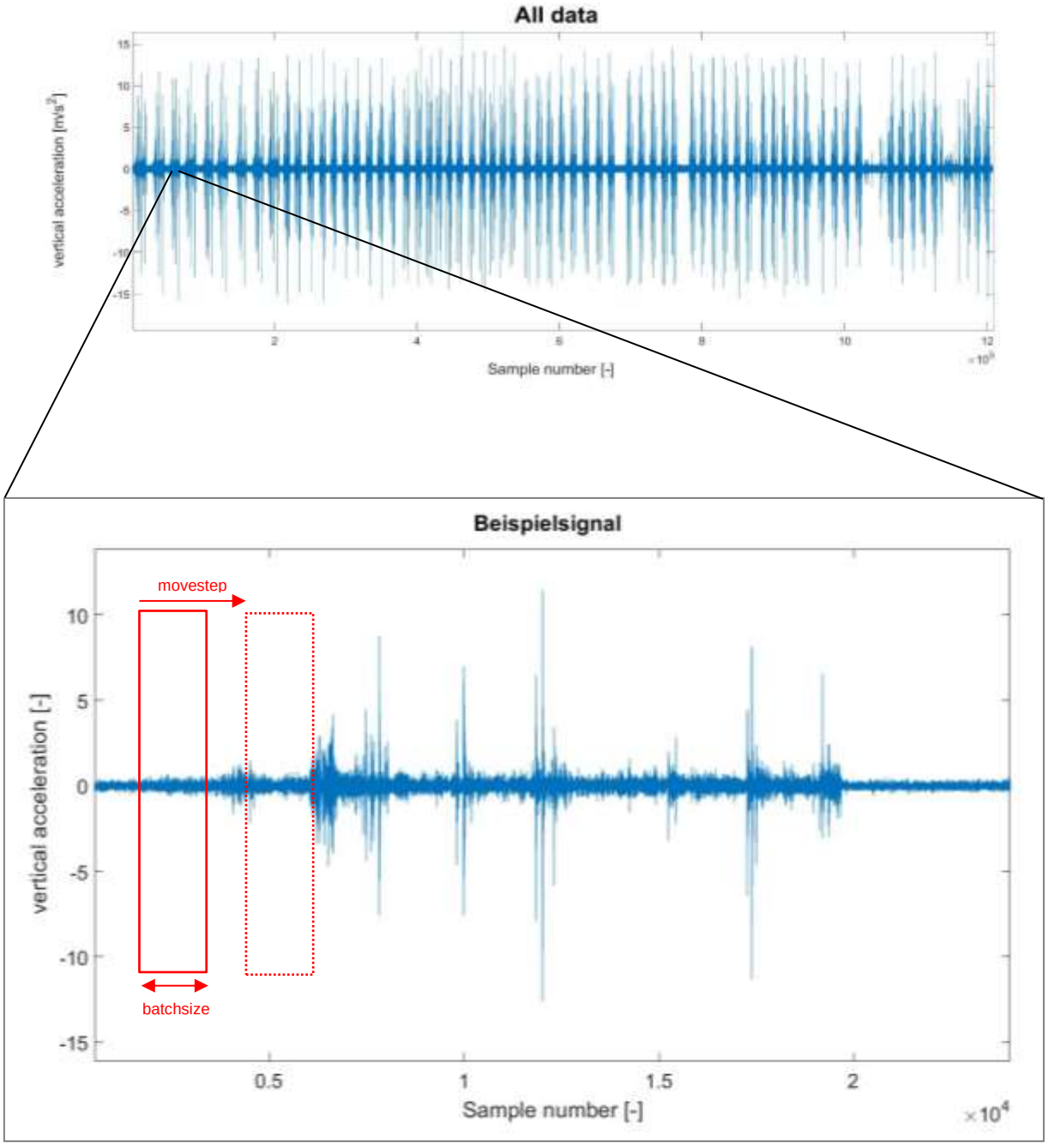


Abbildung 18: Erklärung der Batchgröße und Schrittweite (Pavlovski, 2020)

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).

Anzahl	Eigenschaft	Erklärung & Java Methode
1	Durchschnitt	Der Durchschnitt aller Werte berechnet Durch AIPreprocessingModule.calcMean()
1	Quadratisches Mittel	Das quadratische Mittel aller Werte berechnet durch AIPreprocessingModule.calcRMS()
1	Standardabweichung	Die Standardabweichung aller Werte berechnet durch AIPreprocessingModule.calcStandardDeviation()
8	Spektrale Hochpunkte	Auf die gegebenen Werte wird eine Frequenzanalyse durchgeführt und davon die vier größten Punkte genommen
1	Hauptkomponentenanalyse	Der größte Wert der Hauptkomponentenanalyse wird durch AIPreProcessingModule.calcPCA() berechnet
1	Schiefe	Beschreibt die Art und Stärke der Asymmetrie aller Werte berechnet durch AIPreProcessingModule.calcSkewness()
1	Varianz	Die Varianz der gegebenen Werte berechnet durch AIPreProcessingModule.calcVariance()
1	Erstes Moment	Das erste Moment der gegebenen Werte durch AIPreProcessingModule.calcFirstMoment()
1	Zweiter Moment	Das zweite Moment der gegebenen Werte durch AIPreProcessingModule.calcSecondMoment()

1	Dritter Moment	Das dritte Moment der gegebenen Werte durch AIPreProcessingModule.calcThirdMoment()
1	Vierter Moment	Das vierte Moment der gegebenen Werte durch AIPreProcessingModule.calcFourthMoment()

Tabelle 6: Charakteristika Signalabschnitt (Pavlovski, 2020)

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)

5.1.4.2 Methode 2: Statistische Darstellung der einzelnen Fehlermuster

Hier war der erste Ansatz die einzelnen Fehlermuster mit dessen Vertikalbeschleunigungen herauszuschneiden und diese Werte an das jeweilige Modell weiterzugeben. Dieses Konzept ist allerdings nicht umsetzbar, da viele Modelle nur Batches verarbeiten können, die dieselbe Länge besitzen. In Abbildung 19 ist links das Fehlermuster eines Schienenstoßes mit einer ungefähren Länge von 180 Datenpunkten und rechts ein periodischer Schienenfehler mit einer ungefähren Länge von 500 Datenpunkten zu erkennen.

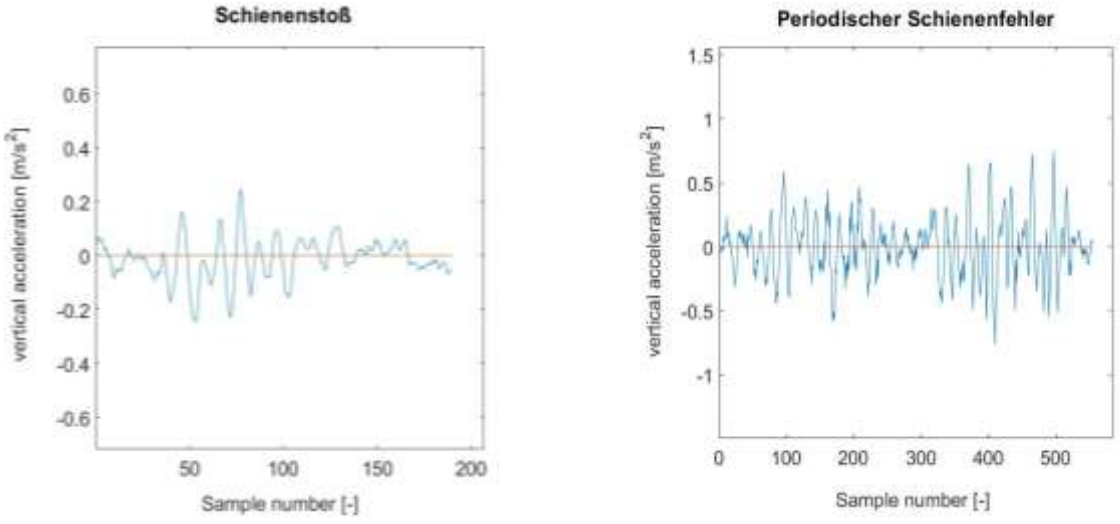


Abbildung 19: Vergleich der Fehlermusterlänge (Pavlovski, 2020)

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 *AIFailureVector* Objekt mit den vertikalen Beschleunigungswerten. Diese werden in einem AIFailureVectorList Objekt gesammelt. Danach werden die *AIFailureVector* Objekte in *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 Operationtab→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 Neural Net Builder, der eine Benutzeroberfläche für die Konfigurierung eines neuronalen Netzes öffnet. (s. Kapitel 5.2.2)

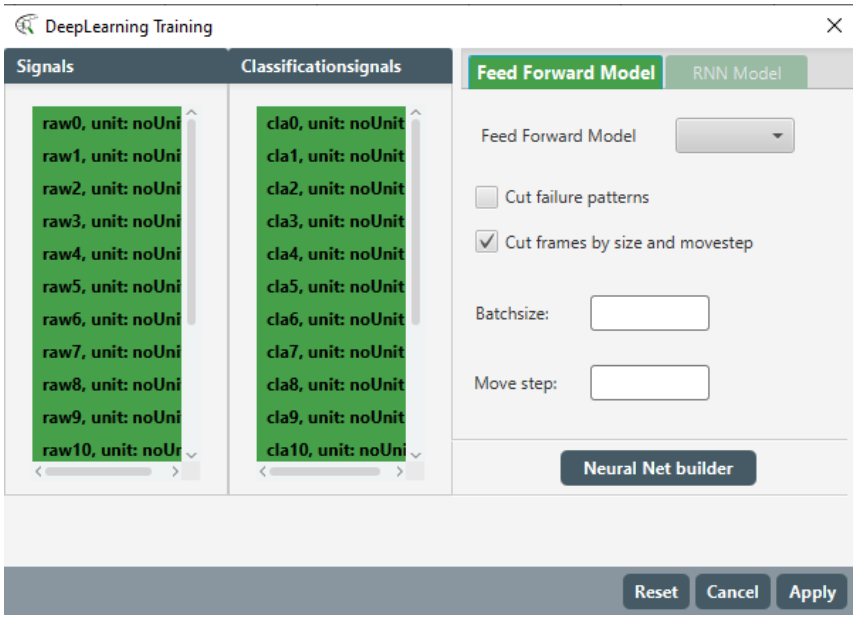


Abbildung 20: Trainingsfenster Feed Forward Modell (Pavlovski, 2020)

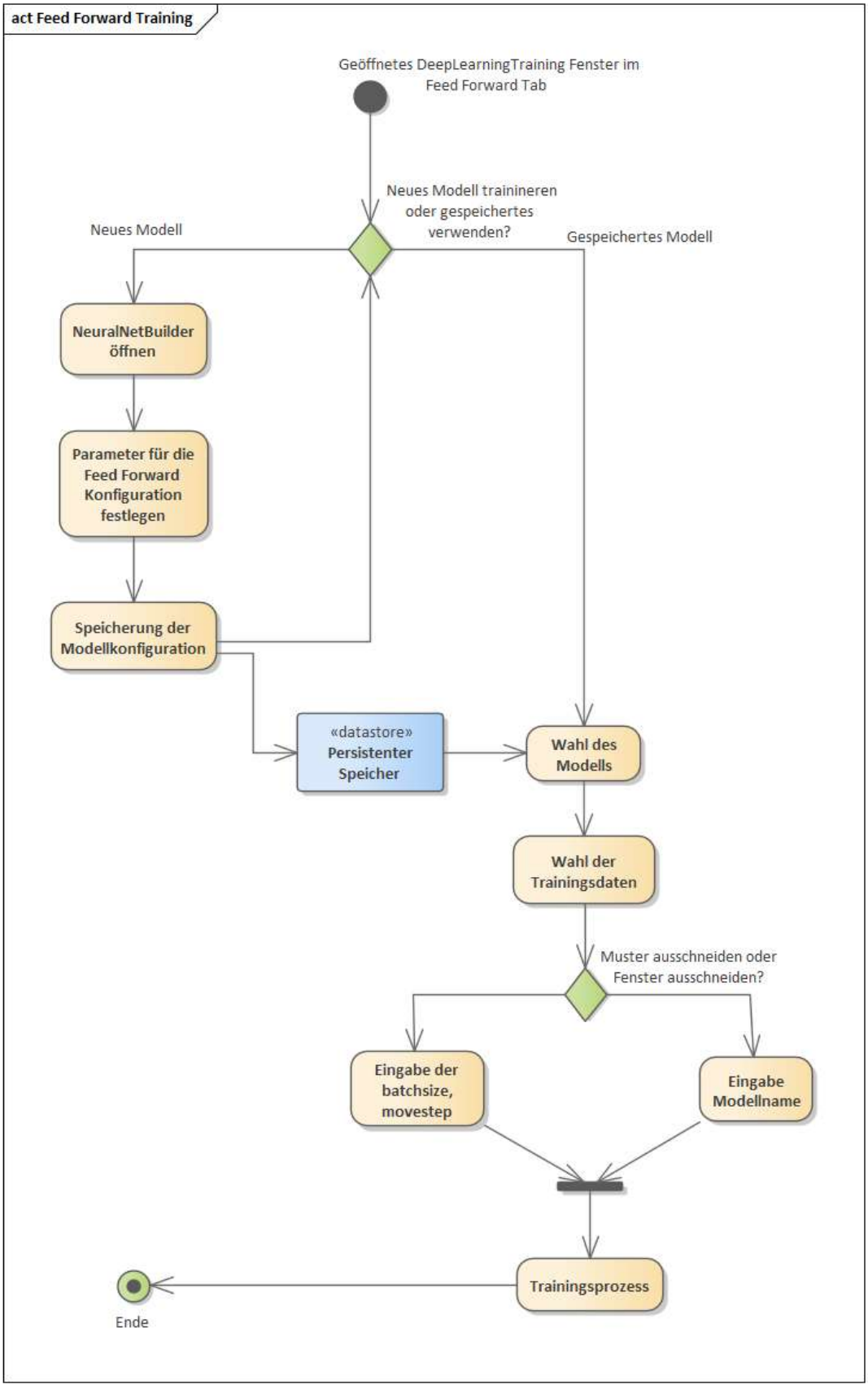


Abbildung 21: Trainingsablauf Feed Forward Modell (Pavlovski, 2020)

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 korrekt 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 korrekt klassifizierten Vorhersagen zu allen Vorhersagen in der jeweiligen Klasse
4. *F1 Score*
   Verhältnis von *Recall* und *Precision*

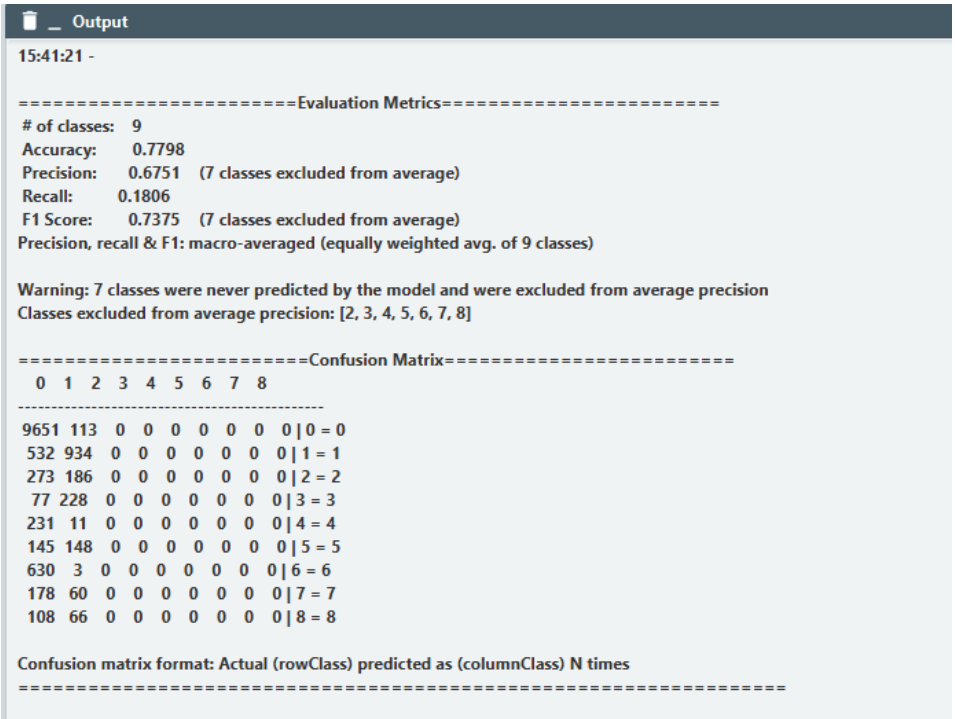


Abbildung 22: Beispielausgabe eines Trainingsprozesses mit Feed Forward (Pavlovski ,2020)

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.

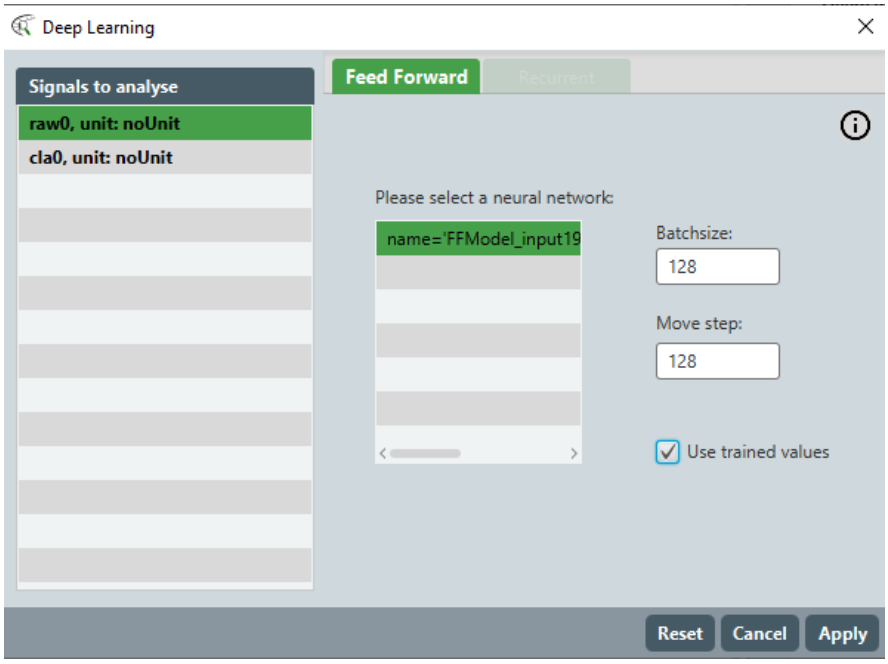


Abbildung 23: Analyse Feed Forward Modell

Abbildung 24 zeigt den Analyseverlauf.

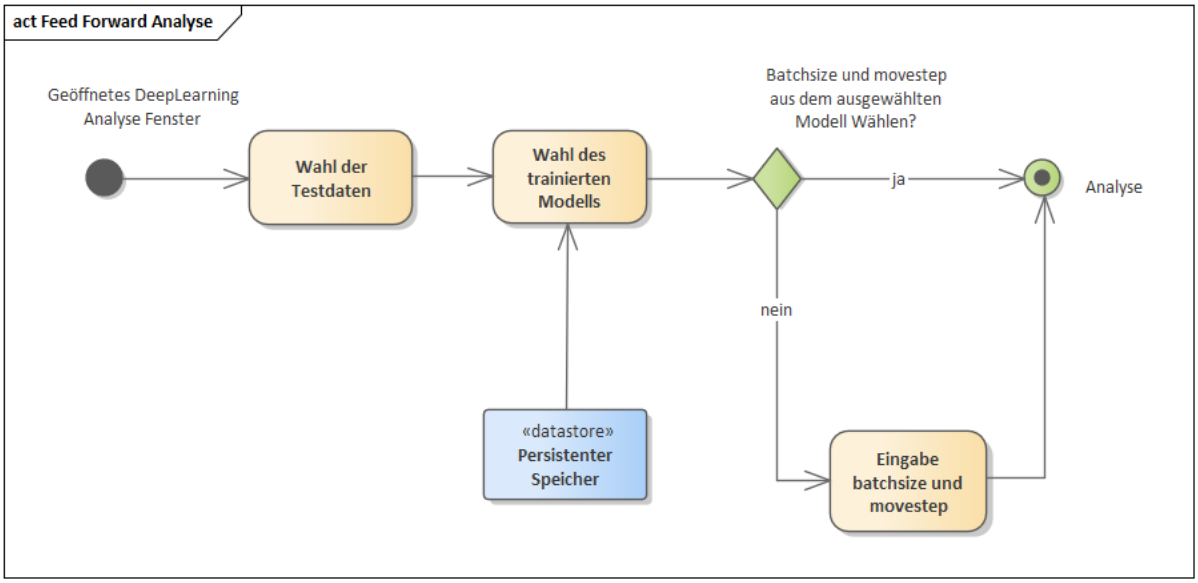


Abbildung 24: Analyse Verlauf Feed Forward (Pavlovski, 2020)

5.2.2 „Neural Net Builder“

Der „Neural Net Builder“ ist zugänglich über das „DeepLearning Training“ Fenster und ermöglicht dem Nutzer eine vollständige Konfiguration eines „feed forward“ -oder „recurrent“ Modells in einer UI-Umgebung. Es können alle Einstellungen, die die „deeplearning4j“ Bibliothek bereitstellt, angewandt, gespeichert und für späteren Gebrauch serialisiert werden.

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.

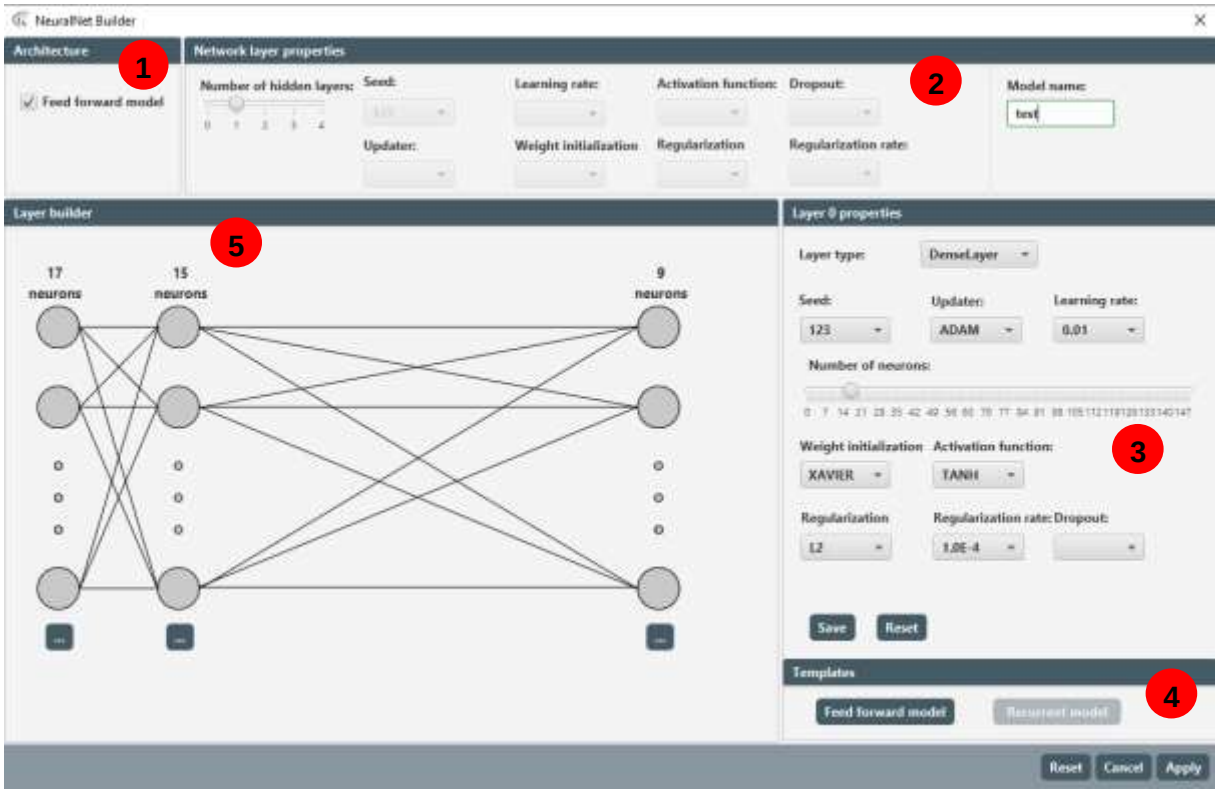


Abbildung 25: *Neural Net Builder* (Pavlovski, 2020)

3 *Layer* – Komponente

Hier befinden sich alle Einstellungsmöglichkeiten der jeweilig ausgewählten Schicht. Neben dem zuvor erwähnten Save - Button ist der Reset - Button, der alle Felder zurücksetzt. Alle Parameter sind in Tabelle 7 erklärt.

4 *Template* – Komponente

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

5 *Layer builder* – Komponente

Zur besseren Veranschaulichung des neuronalen Netzes und zur einfacheren Einstellung der Schichten findet sich hier ein Abbild des eingestellten Modells. Die Visualisierung bzw. die angezeigte Zahl der Zwischenschichten (*hidden layer*) ist an den Schieberegler in der *Network Layer* – Komponente 2 gekoppelt. Unter jeder angezeigten Schicht ist der Einstellungsbutton ..., der die *Layer* – Komponente 3 mit den gespeicherten Parametern aktualisiert.

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

Tabelle 7: Parameter und deren Funktion [13]

Der gesamte Prozess zur korrekten Konfiguration ist Abbildung 26 zu entnehmen.

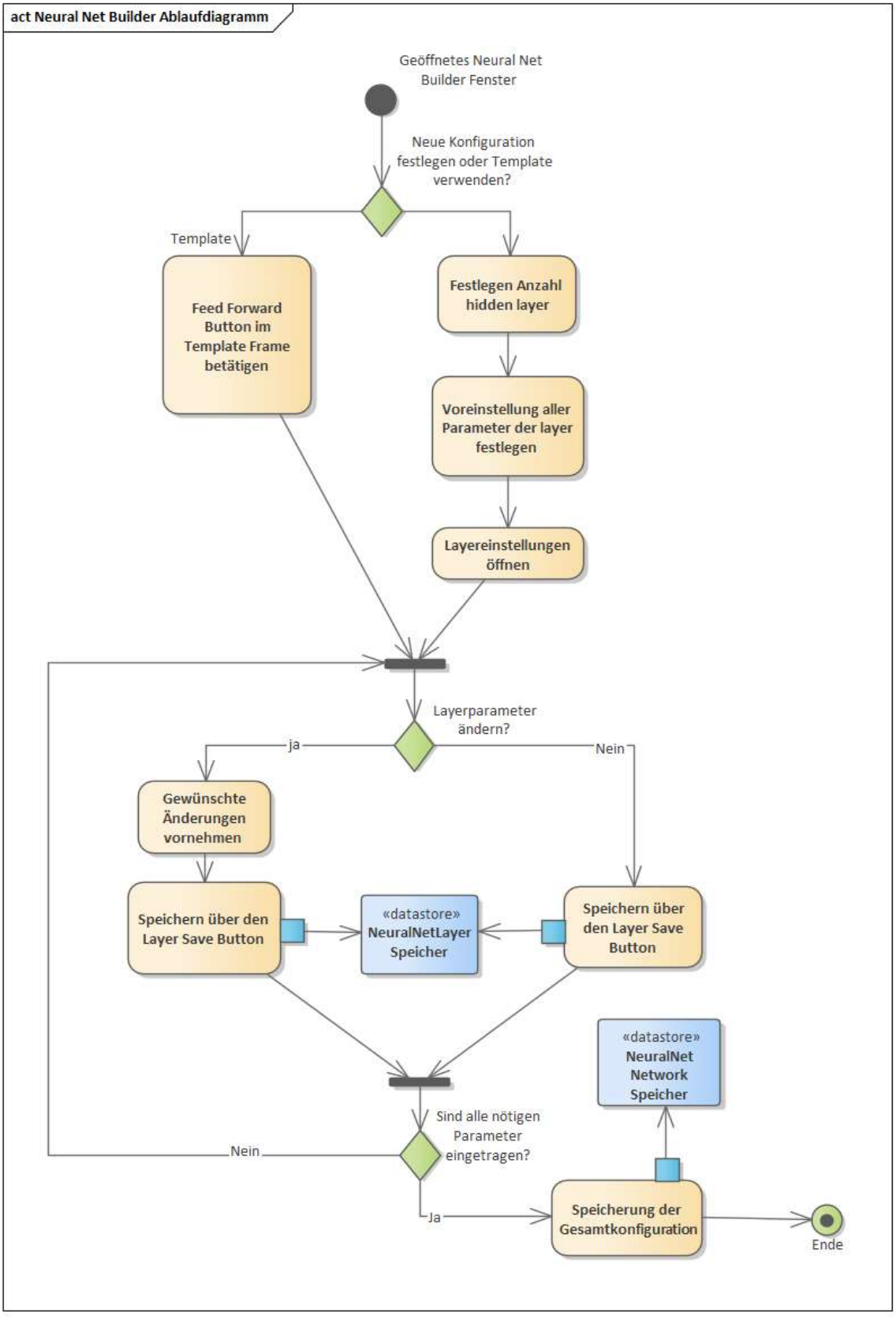


Abbildung 26: Ablauf einer Feed Forward Konfiguration mit Hilfe des Neural Net Builders (Pavlovski, 2020)

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).

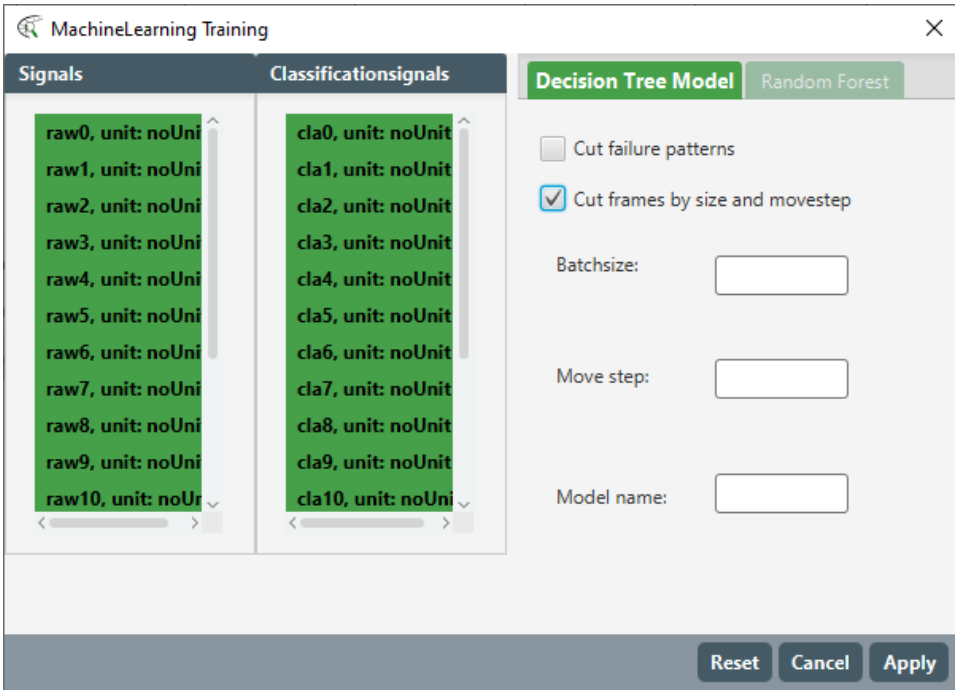


Abbildung 27: Machine Learning Trainingsfenster (Pavlovski,2020)

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.
Werden nämlich als Input nur die vertikalen Beschleunigungen genommen, erzielt das Modell sehr schlechte Ergebnisse. Grund dafür ist die Architektur eines *Decision Trees*, wie in Kapitel 2.1.1 erklärt. Es handelt sich um einen Binärbaum, der in jedem Knoten nur die Entscheidung „wahr“ oder „falsch“ trifft, abhängig von der Bedingung im Knoten. Somit sind Werte, die breit gestreut sind und wenig Zusammenhang aufweisen, keine gute Wahl. Ist die Entscheidung gefallen, welche Datenaufbereitungsmethode genommen wird, startet der Trainingsprozess über den Apply - Button. Der genaue Ablauf eines Trainingsprozesses mit Einstellung der möglichen Parameter zeigt Abbildung 28.

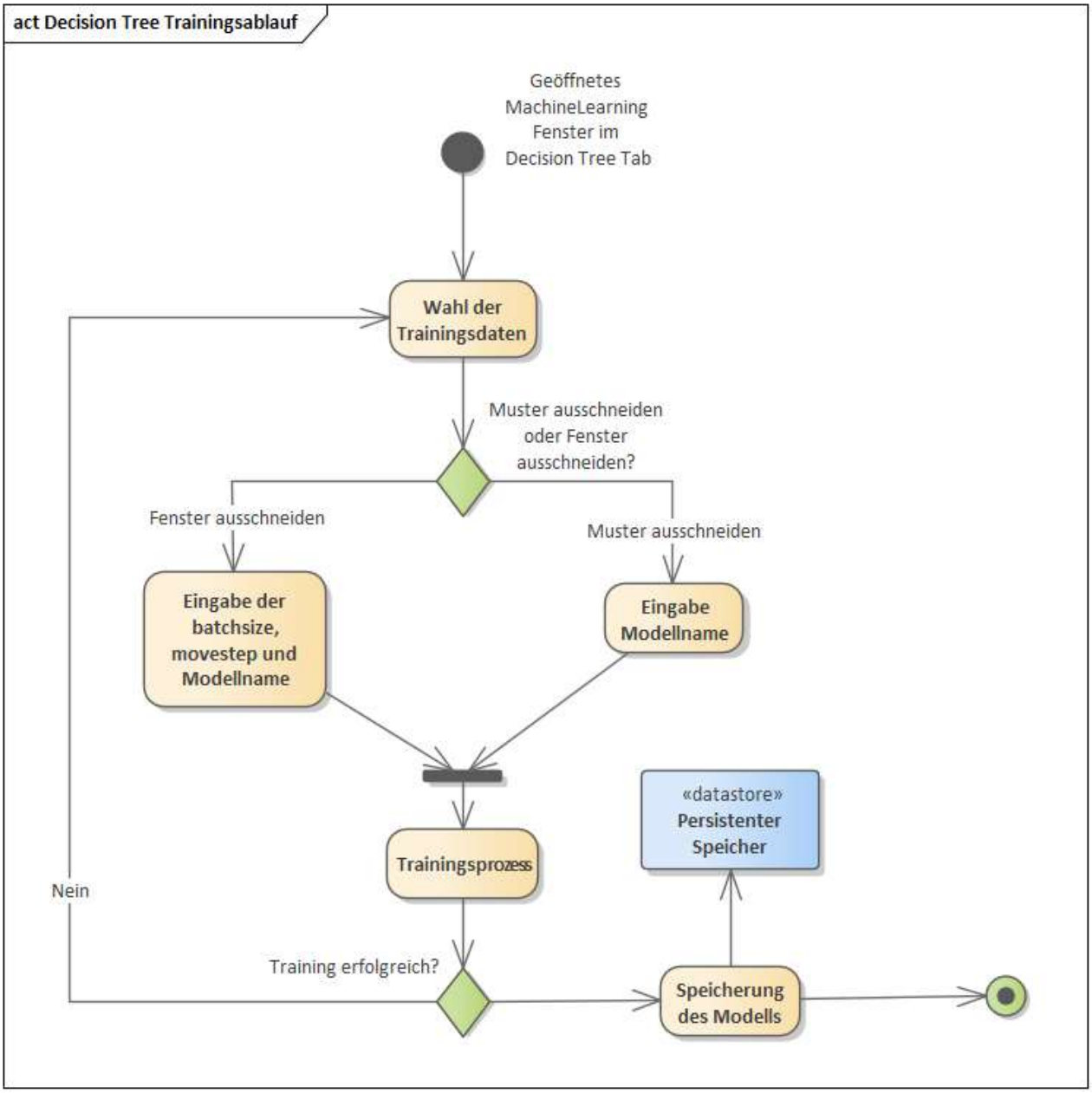


Abbildung 28: Ablaufdiagramm Decision Tree Training (Pavlovski, 2020)

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.

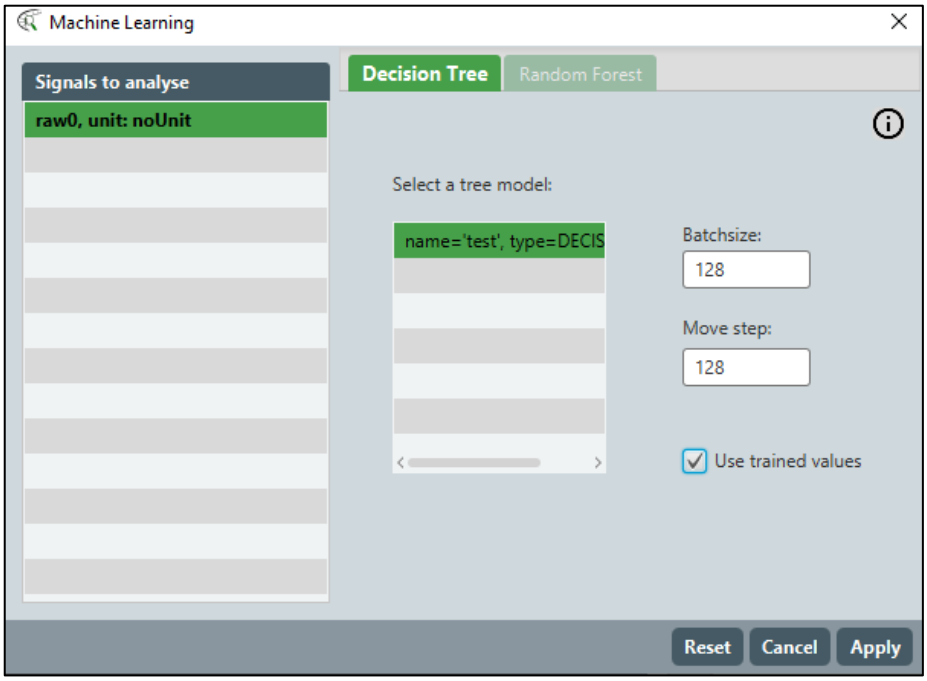


Abbildung 29: Analysefenster Decision Tree (Pavlovski, 2020)

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 Apply - Buttons startet die Analyse mit der Visualisierung. Der genau Ablauf kann Abbildung 30 entnommen werden.

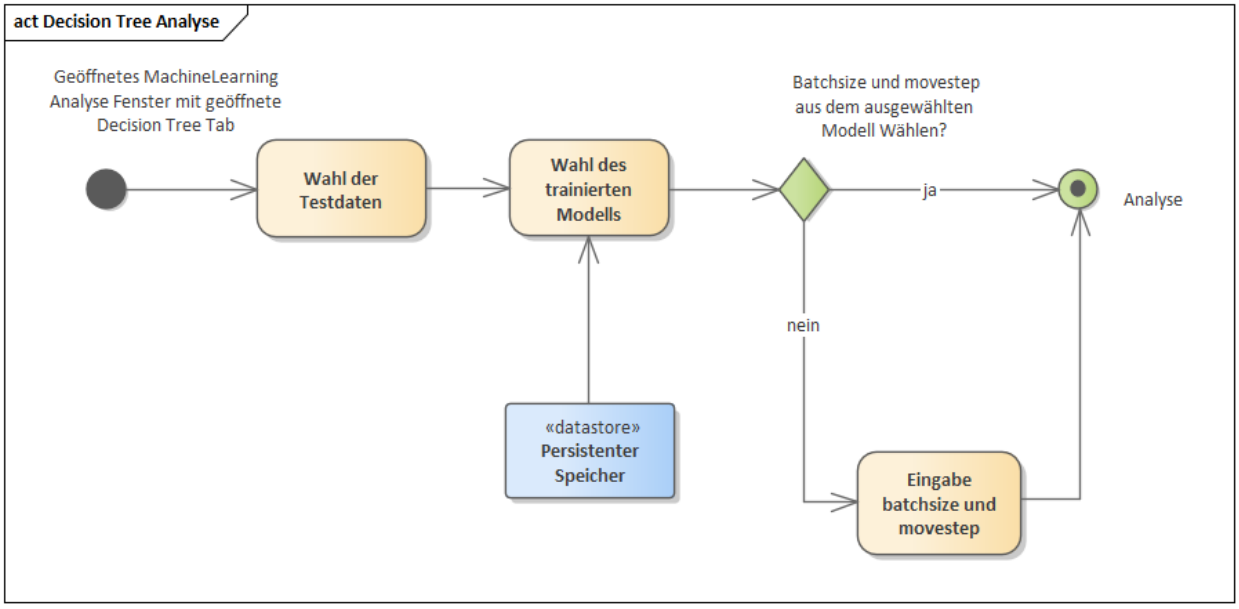


Abbildung 30: Analyseablauf Decision Tree (Pavlovski, 2020)

5.3.2 Random Forest Modell

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

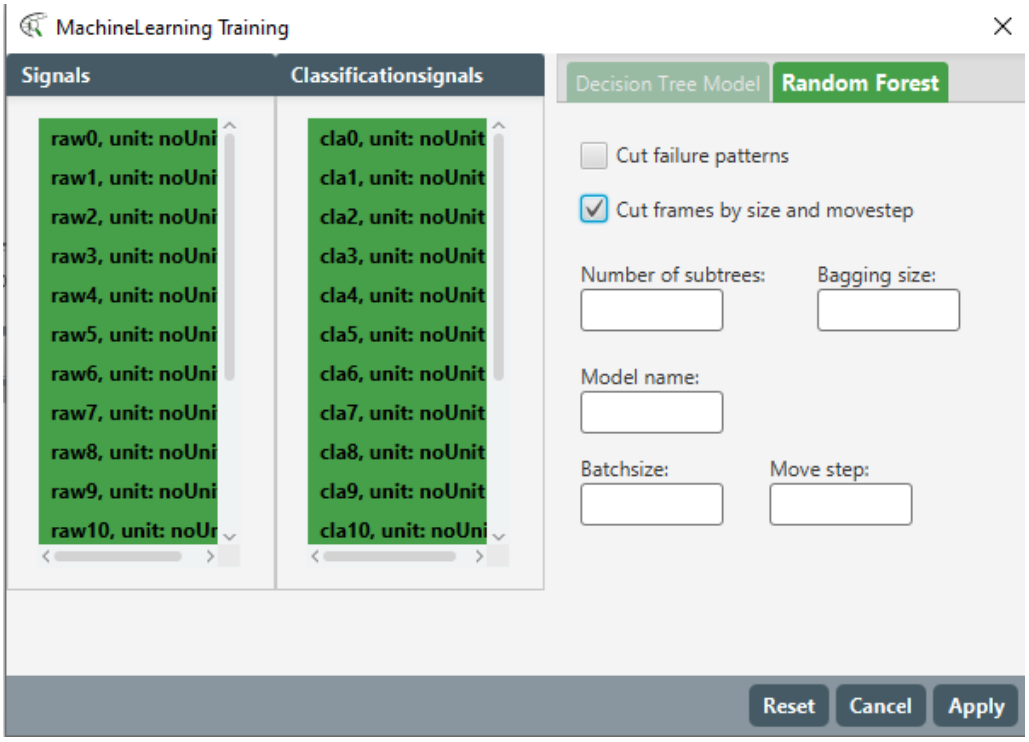


Abbildung 31: Random Forest Tab (Pavlovski, 2020)

Der Aufbau ist sehr ähnlich wie beim Decision Tree Model, bis auf die hinzugekommenen 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.

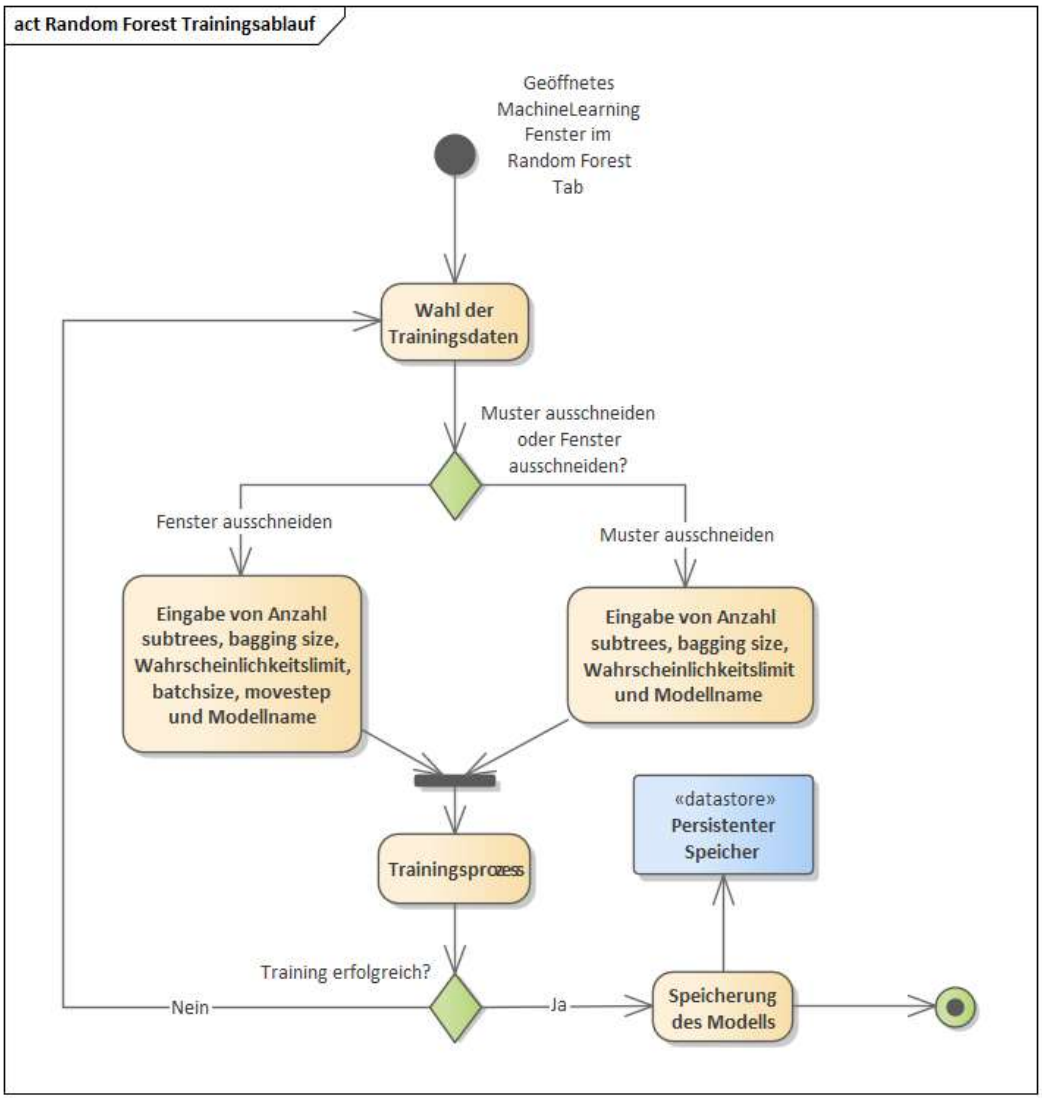


Abbildung 32: Random Forest Trainingsablauf (Pavlovski, 2020)

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.

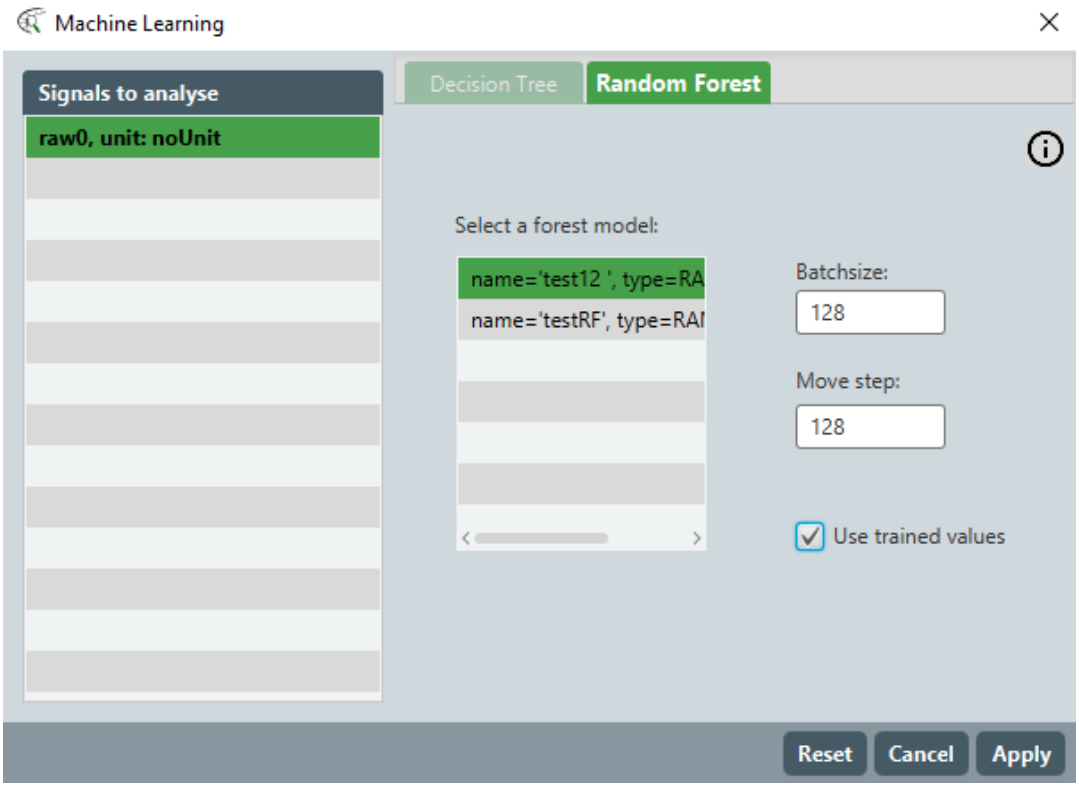


Abbildung 33: Random Forest Tab Analyse (Pavlovski, 2020)

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.

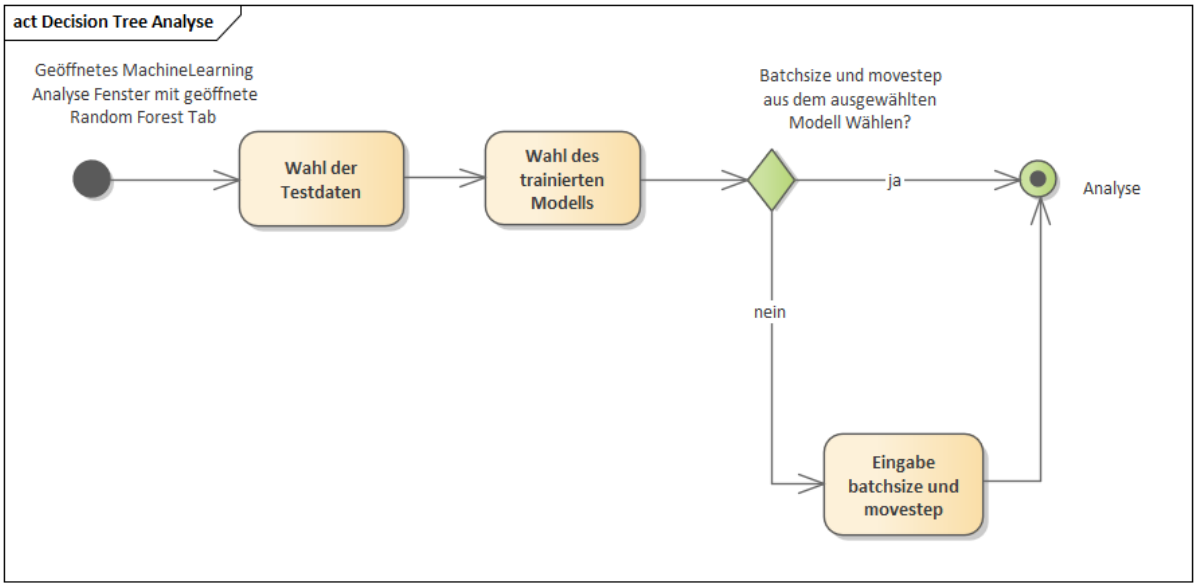


Abbildung 34: Analyseablauf (Pavlovski, 2020)

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.

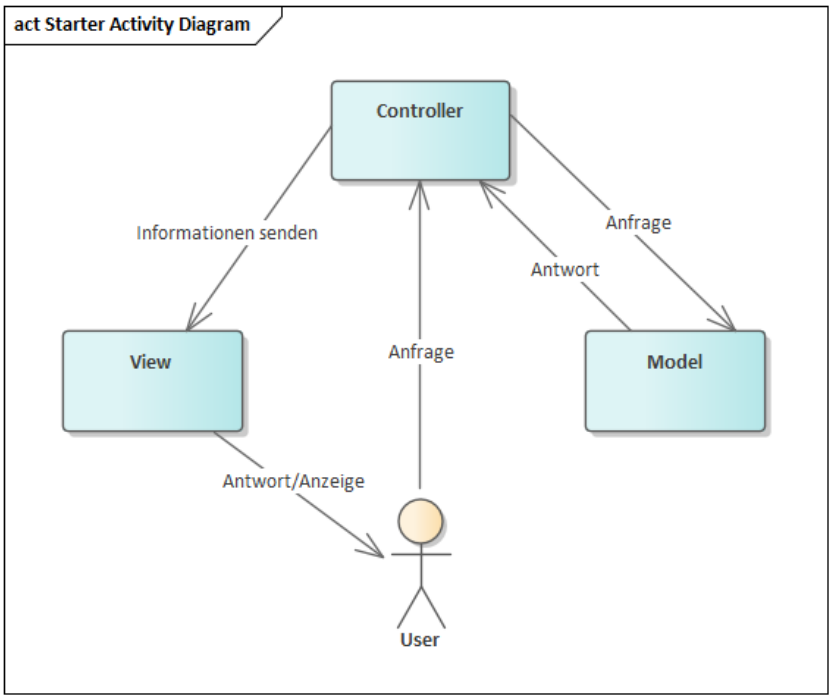


Abbildung 35: Model-View-Controller Konzept (Pavlovski,2020)

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 *PersistanceManager.jav*a 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.

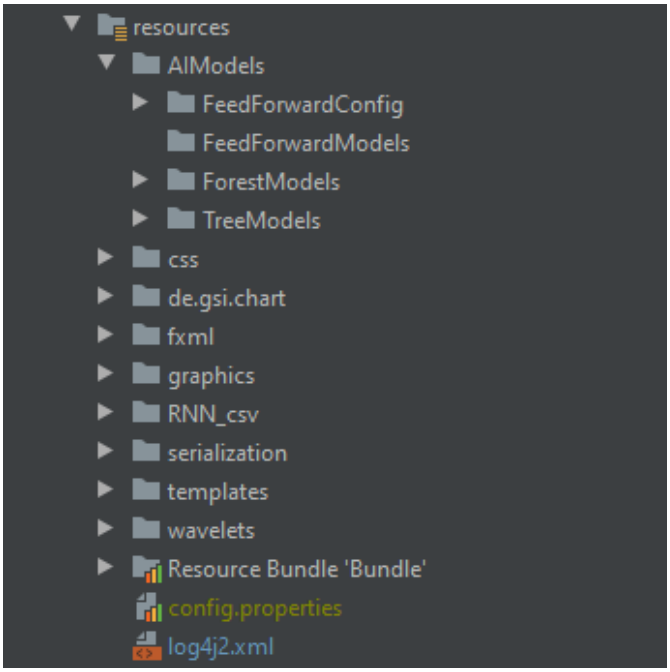


Abbildung 36: Ordnerstruktur Speicherort KI-Modelle (Pavlovski, 2020)

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

Das beste Ergebnis liefert das Random Forest Modell. Abbildung 37 zeigt hier ein Beispiel. Hier wurde das Modell mit den ersten 15 Trainingsdaten aus Tabelle 5 trainiert und mit der letzten Trainingsdatei getestet.

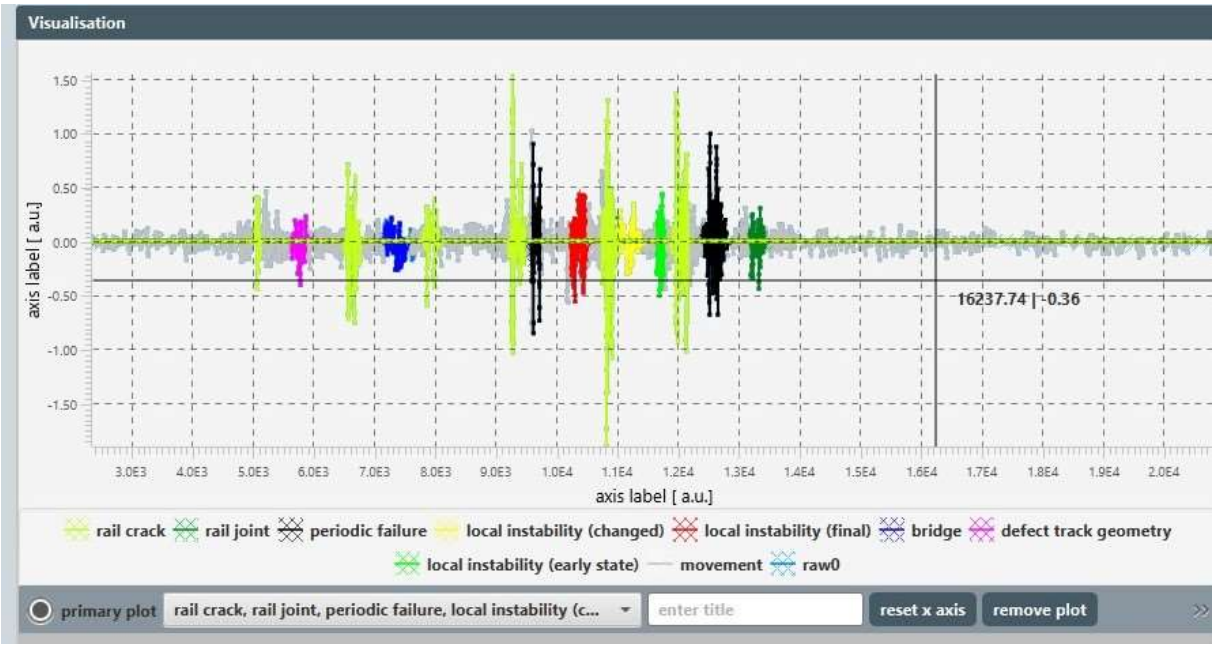


Abbildung 37: Random Forest Ergebnis (Pavlovski, 2020)

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 zeigt das Beispiel des Decision Tree Modells. Die Ergebnisse sind vergleichbar mit denen des Random Forest Modells, sind aber auffälliger für Fehlvorhersagen.

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

Diese Fehlvorhersagen sind mit 1 und 2 in Abbildung 38 gekennzeichnet.
Folgende Parameter wurden als die Geeignetsten empfunden:

- Batchsize = 128
- Movestep = 128

Abbildung 39 zeigt ein Beispielergebnis des Feed Forward Modells. Dieses Modell schneidet unter allen am schlechtesten ab. Jedoch existieren für das Modell deutlich mehr Parametereinstellungen, die das Ergebnis beeinträchtigen können.

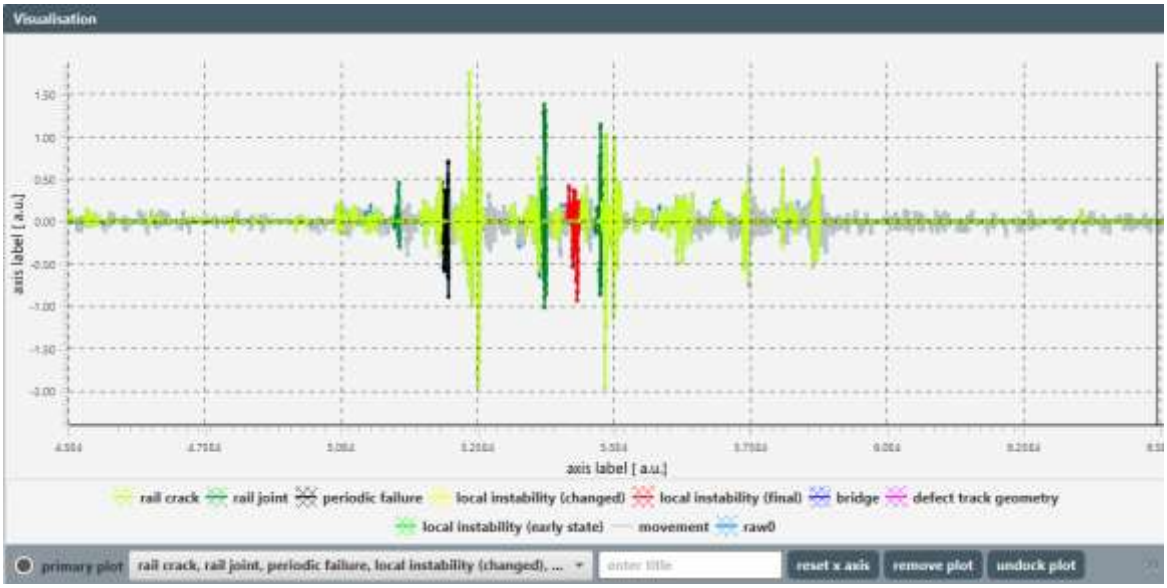


Abbildung 39: Feed Forward Beispiel Ergebnis (Pavlovski, 2020)

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

Die Ausgabe von Methode 1 der Datenaufbereitung (s. Kapitel 5.1.4.1) können die Modelle nicht gut genug verarbeiten. Dadurch dass die Fehler automatisch herausgeschnitten werden, muss in der Analyse die *batchsize* und der *movestep* manuell ausgewählt werden. Die Problematik liegt an der unterschiedlichen Länge jedes Fehlermusters.
Ein vielversprechender Ansatz ist kleine Teile aus den Messdaten herauszuschneiden und diese mit statistischen Eigenschaften darzustellen, die in Kapitel 5.1.4 erläutert werden.
Aber auch hier haben die Modelle die besten Ergebnisse, wenn die Messdaten in jeweils zwischen 128 und 256 Datenpunkten große Abschnitte geschnitten werden. Warum nun genau diese Längen die besten Ergebnisse liefert, liegt an der an den Durchschnittslängen aller Fehlerklassen. Tabelle 8 zeigt hier die Mediane einer jeder Fehlerklasse, ausgerechnet aus den Trainingsdaten, die im Kapitel 5.1.1 erklärt wurden. Die Funktion für diese Berechnung befindet sich, ebenso wie die anderen Methoden zur Datenaufbereitung, in der *AIPreProcessingModule.java* Klasse.

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

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 Fehlermuster herauszuschneiden, dies über statistische Eigenschaften anders darstellen und damit das gewählte Modell zu trainieren.

Im Datenaufbereitungsprozess werden zusätzlich alle Mediane der jeweiligen Fehlerklasse gespeichert.

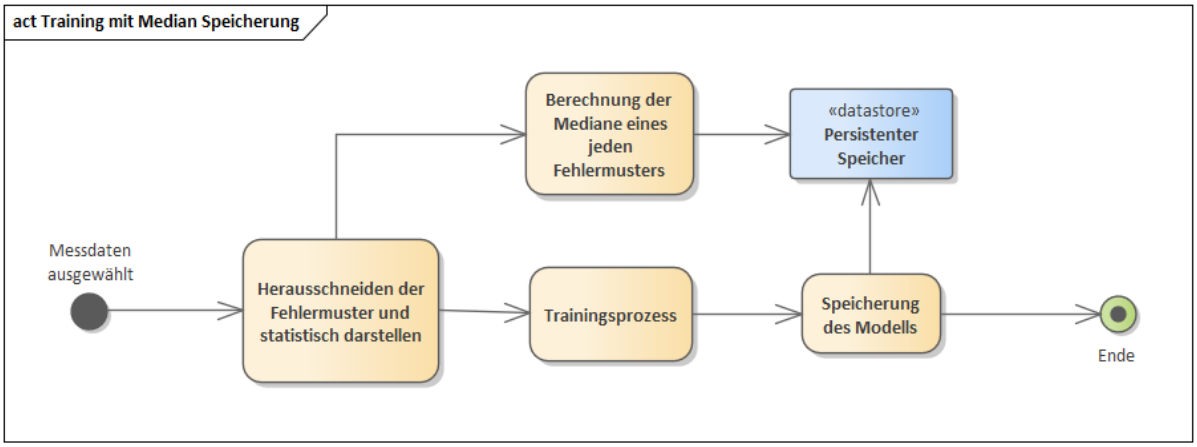


Abbildung 40: Training mit Median Speicherung (Pavlovski, 2020)

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.

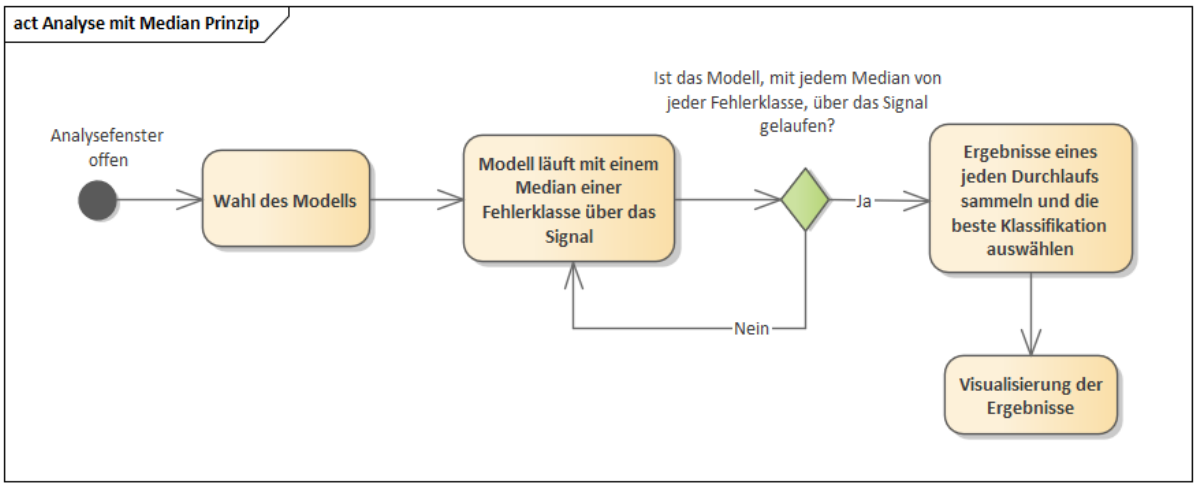


Abbildung 41: Analyseverlauf mit Hilfe des Medians (Pavlovski,2020)

6.3 Java vs. Python Fazit

In Bezug auf die zukünftige Forschungsplanung soll PUL-Anfahr in die Software PULTrack integriert werden. Beide Anwendungen sind in Java implementiert.
In Hinsicht auf die Wartung und Erweiterung, empfiehlt sich Java. Insbesondere muss der/diejenige nur Wissen über eine Programmiersprache besitzen, um die Gesamtsoftware PUL-Track vollständig zu verstehen.
Auf der anderen Seite, in Hinsicht auf die KI-Module, hätte es die Möglichkeit gegeben ein externes Modul in Python zu implementieren, was nicht nur von PUL-Anfahr bzw. PULTrack benutzt werden kann, sondern, durch eine entsprechende Schnittstelle, von jeder anderen Software ebenfalls. Zudem, wie schon aus Kapitel 4.3 ersichtlich, bietet Python in Bezug auf Datenimport und Datenmanipulation, kürzeren und verständlicheren Code.
Aufgrund der zu erwartenden Größe von PULTrack empfiehlt sich weiterhin Java zu verwenden.

6.4 Ausblick

- Mit dem heutigen Stand von PUL-Anfahr 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

[1] DB AG, „www.deutschebahn.de," 2019. [Online]. Available: https://www.deutschebahn.com/de/presse/pressestart_zentrales_uebersicht/Bilanz-2018-Neuer-Fahrgastrekord-Umsatz-gestiegen-Investitionsoffensive-fuer-bessere-Bahn--4045230.

[2] M. U. S. M. a. S. Rapp S., „Track-vehicle scale model for evaluating local track defects detection methods," Transportation Geotechnics, 2019.

[3] The Mathworks Inc., „Matlab R2019b".

[4] S. R. Y. Z. J. L. U. M. S Bahamon-Blanco, „Recognition of track defects through measured acceleration using a Recurrent Neural Network," Institute if Railway abd Transportation Engineering, Pfaffenwaldring 7, 70569 Stuttgart, 2019.

[5] S. Bahamon-Blanco, *Detection of local instabilities on a scale vehicle-track model through measured accelerations of the vehicle,* Stuttgart, 2018.

[6] R. Geelong, 2016. [Online]. Available: https://www.railgeelong.com/cache/corio-independent-goods-line/D569_6985_640.jpg.

[7] S. Rapp, „Erkennung von Schienen- und Gleislagefehlern anhand der kontinuierlich gemessenen Beschleunigung im Modell Auftaktworkshop," 2018.

[8] Quora, 2015. [Online]. Available: https://www.quora.com/How-does-the-locomotive-derails-in-terms-of-mechanical-engineering.

[9] P. S. PUL-Anfahr, „Benutzerhandbuch PUL-Anfahr," Uni Stuttgart, Stuttgart, 2019.

[10] J. FRIEDERIKE BÖGE, „faz.net," [Online]. Available: https://www.faz.net/aktuell/wirtschaft/kuenstliche-intelligenz/gesichtserkennung-kuenstliche-intelligenz-erobert-chinas-provinzen-16131110.html. [Zugriff am 22 06 2020].

[11] ecloudvalley, „ecloudvalley," 13 06 2020. [Online]. Available: https://www.ecloudvalley.com/mlintroduction/.

[12] K. M. Rosaria Silipo, „dataversity," 2 08 2019. [Online]. Available: https://www.dataversity.net/from-a-single-decision-tree-to-a-random-forest/.

[13] J. Rocca, „towardsdatascience," 23 April 2019. [Online]. Available: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205. [Zugriff am 2 05 2020].

[14] S. Skansi, Introduction to Deep Learning, Springer, 2018.

[15] P. Carbonnelle, „pypl.github.io,“ 2019. [Online]. Available: http://pypl.github.io/PYPL.html. [Zugriff am 5 Februar 2020].

[16] Snaplogic, 7 Februar 2020. [Online]. Available: https://www.snaplogic.com/glossary/python-vs-java-performance.

[17] MTheiler, 7 Februar 2020. [Online]. Available: https://de.wikipedia.org/wiki/Deeplearning4j.

[18] A. H. C. R. Ian Clarke, „github.com,“ [Online]. Available: https://github.com/Tradeshift/quickdt. [Zugriff am 28 06 2020].

[19] Konduit, „deeplearning4j,“ [Online]. Available: https://deeplearning4j.konduit.ai/. [Zugriff am 28 06 2020].

[20] S. Grasreiner, „CCM,“ 7 Februar 2020. [Online]. Available: https://de.ccm.net/contents/168-programmiersprachen.

[21] Python Software Foundation, „Python Documantation,“ 7 Februar 2020. [Online]. Available: https://docs.python.org/3/faq/general.html.

[22] T. Gupta, „towardsdatascience,“ 5 1 2017. [Online]. Available: https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7. [Zugriff am 5 2 2020].

8 Anhang

Parameter neuronales Netz	Mögliche Parametereinstellungen
Layer type	DenseLayer, OutputLayer
Updater	SGD, ADAM, ADAMAX, ADADELTA, NESTEROVS, NADAM, ADAGRAD, RMSPROP, NONE, CUSTOM
Seed	123, 123456
Dropout	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.0
Learning rate	0.00001, 0.0001, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0, 3.0, 10.0
Weight initialization	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
Activation function	CUBE, ELU, HARDSIGMOID, HARDTANH, IDENTITY, LEAKYRELU, RATIONALTANH, RELU, RELU6, RRELU, SIGMOID, SOFTMAX, SOFTPLUS, SOFTSIGN, TANH, RECTIFIEDTANH, SELU, SWISH, THRESHOLDEDRELU, GELU
Regularization	L1, L2, NONE
Regularization rate	0.00001, 0.0001, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0, 3.0, 10.0
Loss Function	MSE, L1, EXPLL, XENT, MCXENT, RMSE_XENT, SQUARED_LOSS, RECONSTRUCTION_CROSSENTROPY,

	NEGATIVELOGLIKELIHOOD, CUSTOM, COSINE_PROXIMITY, HINGE, SQUARED_HINGE, KL_DIVERGENCE, MEAN_ABSOLUTE_ERROR, L2, MEAN_ABSOLUTE_PERCENTAGE_ERROR, MEAN_SQUARED_LOGARITHMIC_ERROR, POISSON, WASSERSTEIN