

**DHBW KARLSRUHE**  
APPLIED COMPUTER SCIENCE

STUDIENARBEIT

# **Klassifikation und Analyse aus Stromdaten im Haushalt mit neuronalen Netzen**

Tim Schrodi  
6857027, TINF15B1

01.10.2017 - 14.05.2018  
BETREUT VON DANIEL LINDNER  
PROF. DR. JOHANNES FREUDENMANN, STUDIENGANGSLEITER

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>2</b>
<b>2</b>	<b>Grundlagen</b>	<b>4</b>
2.1	Maschinelles Lernen (Machine Learning) . . . . .	4
2.1.1	Künstliche Intelligenz . . . . .	4
2.1.2	Einführung . . . . .	4
2.1.3	Lernprozess . . . . .	5
2.2	Neuronale Netze . . . . .	6
2.2.1	Aufbau . . . . .	6
2.2.2	CNN - Convolutional Neural Network . . . . .	10
2.2.3	RNN - Recurrent Neural Network . . . . .	10
2.3	Physikalische Grundlagen zur Netzaktivität . . . . .	10
2.4	Erhebung der Messdaten . . . . .	10
2.4.1	Klassifikation der Messdaten . . . . .	11
2.5	Visualisierung . . . . .	12
<b>3</b>	<b>Ausführung</b>	<b>14</b>
3.1	Manuelle Analyse . . . . .	14
3.2	Vorbereiten der Daten . . . . .	18
3.3	Neuronales Netz . . . . .	19
3.4	Trainingsprozess . . . . .	23
3.5	Auswertungsalgorithmus . . . . .	24
<b>4</b>	<b>Ergebnis</b>	<b>28</b>
4.0.1	Auswertung der Ergebnisse . . . . .	31
4.0.2	Ergebnisglättung . . . . .	33
4.0.3	Zusammenfassung . . . . .	34
<b>5</b>	<b>Produktivbetrieb</b>	<b>35</b>
5.0.1	Implementierung . . . . .	35
5.0.2	Deployment . . . . .	36

5.0.3	API-Spezifikation . . . . .	37
<b>6</b>	<b>Wirtschaftlichkeit</b>	<b>39</b>
<b>7</b>	<b>Ausblick</b>	<b>40</b>

## **Zusammenfassung**

Das Ziel dieser Studienarbeit ist es mithilfe von verschiedenen Machine Learning-Methoden aus Stromdaten eines Haushalts verschiedene Geräte zu klassifizieren. Diese Klassifikation beinhaltet die Bestimmung der Laufzeit dieser Geräte innerhalb einer Zeitreihe. Die Bestimmung der Laufzeit bedeutet, dass bestimmt werden soll zu welchen Zeiten welche Geräte aktiv waren. Dazu wurden mit einem Messgeräte die allgemeine Spannung, Frequenz und verschiedene Oberwellen eines üblichen Stromnetzwerks eines privat Haushalts über mehrere Monate erfasst. Hinzu wurden manuell verschiedene Geräte wie eine Kaffeemaschine oder eine Mikrowelle klassifiziert. Anhand der Stromverläufe und den dazu klassifizierten Geräten wurden verschiedene neuronale Netze trainiert und miteinander verglichen.

Auch wird die Wirtschaftlichkeit sowie der produktive Einsatz der Ergebnisse beachtet. Die neuronalen Netze, welche die besten Ergebnisse erzielten werden außerdem im produktiv Betrieb getestet und eingesetzt.

<b>MSSQL</b>	Microsoft SQL-Server
<b>CI</b>	Continuous Integration
<b>CD</b>	Continuous Delivery
<b>E2E</b>	end to end
<b>RNN</b>	Recurrent Neural Network
<b>CNN</b>	Convolutional Neural Network

# Kapitel 1

## Einleitung

Was bis vor kurzer Zeit nur ein wissenschaftlicher Teil der Informatik war, erhält nun immer größere Bedeutung und Einfluss in vielen weiteren wirtschaftlichen und wissenschaftlichen Themen. Seit große IT Firmen wie Google oder Facebook große Fortschritte mit maschinellem Lernen und künstlicher Intelligenz erzielen, wird maschinelles Lernen auch produktiv eingesetzt und immer mehr Nutzer kommen damit im täglichen Leben in Berührung. So liegt es Nahe dies auch auf bisher unberührten Branchen wie Automobil oder Elektronik Branche auszuweiten.

Eine dieser neuen Branchen ist die elektrische Energiewirtschaft, welche unter anderem die elektronische Infrastruktur und somit auch die Grundversorgung an elektrischer Energie bereitstellt. Die Versorgung von privaten Haushalten sowie Firmen mit elektrischer Energie ist mit der immer schneller wachsenden Nachfrage nach diesem Rohstoff, eine nicht mehr wegzudenkende Kernindustrie, welche immer noch große Wachstumschancen hat. In dieser Arbeit wird Data-Mining auf elektrotechnische Größen angewendet um weiterführende semantische Aussagen über diese Werte zu erhalten. Es werden verschiedene Netzdaten und dazugehörige Verläufe aufgezeichnet und mithilfe von verschiedenen Methoden von Maschinellern analysiert. Hauptbestandteil ist die Mustererkennung in den aufgenommenen Verläufen und deren Zuordnung zu verschiedenen Geräten in einem Haushalt. Hierbei sollen verschiedene normale Haushaltgeräte, wie Kaffeemaschinen oder Fernseher, welche innerhalb einem Stromnetzes eines privaten Haushalts betrieben werden, erkannt werden. Mit den damit erhobenen Daten können somit Aussagen über Laufleistungen getroffen werden. Außerdem wird auf die wirtschaftliche Nutzung des resultierenden Modells, sowie deren Produktivbetrieb eingegangen.

Für die maschinelle Analyse wird die Keras-API<sup>1</sup> mit Tensorflow im Backend verwendet. Die Datenverarbeitung, Visualisierung sowie die Trainings- und Testphasen werden mit Python<sup>2</sup> umgesetzt.

---

<sup>1</sup><https://keras.io/>

<sup>2</sup><https://www.python.org/>

# Kapitel 2

## Grundlagen

### 2.1 Maschinelles Lernen (Machine Learning)

Wenn man Maschinelles Lernen oder Künstliche Intelligenz hört, denkt die Mehrzahl an Roboter mit eigenem Bewusstsein und Denken wie in vielen Science-Fiction-Filmen dargestellt. Jedoch ist Maschinelles Lernen mittlerweile keine Zukunftstechnologie mehr. Bereits in den 60er Jahren gab es erste Versuche der Wissenschaft Künstliche Intelligenz zu erschaffen. Doch was ist Maschinelles Lernen wirklich? Und was bedeutet es für einen Computer zu lernen?

Dieses Kapitel beschäftigt sich mit diesen Fragen und gibt einen kurzen Überblick über heutige Verfahren von Maschinellern Lernen.

#### 2.1.1 Künstliche Intelligenz

Bevor Maschinelles Lernen erklärt werden kann sollte Künstliche Intelligenz im allgemeinen geklärt werden.

#### 2.1.2 Einführung

Nimmt man den Begriff Maschinelles Lernen wörtlich beschreibt er das Lernen einer Maschine, also die Fähigkeit einer Maschine intelligenter zu werden. Von Maschinellern Lernen spricht man, falls eine Maschine auf Basis von Erfahrung und Fakten „ohne speziell programmiert worden zu sein“ [1, 20], neues Wissen oder neue Zusammenhänge generieren kann. Wenn eine Maschine nachdem sie etwas gelernt hat bei der Ausführung einer Aktivität besser geworden ist, hat die Maschine maschinell gelernt [1, 20]. Das reine auswendig lernen von Fakten, wie beispielsweise das abspeichern einer Wikipedia-Seite



auf die lokale Festplatte eines Computers, ist kein Wissenserwerb.

Ein Beispiel für Maschinelles Lernen ist der Spamfilter bei Emails. Hier lernt ein Computer auf Basis von bisherigen Spammails neue Emails als Spam zu erkennen.

Der Einsatz von Maschinellern Lernen hat meist Vorteile gegenüber herkömmlichen Statistischen Methoden falls große Datenmengen ausgewertet werden müssen oder kein bekanntes Modell zur Problemlösung bekannt ist. Durch Maschinelles Lernen können Zusammenhänge erkannt werden, die durch andere statistische oder algorithmische Verfahren nicht erkannt oder abgebildet werden können.

### **2.1.3 Lernprozess**

Grundlegend besteht Maschinelles Lernen aus einer Trainingsphase und einer Test- beziehungsweise Validierungsphase. Hierzu werden zunächst alle vorhandenen Daten in Trainings- sowie Testdaten meist im Verhältnis 80:20 eingeteilt. In der Trainingsphase wird auf Basis der Trainingsdaten, wie zum Beispiel bisherige Emails eines Benutzers, ein Modell erstellt, welches dann in der Validierungsphase auf seine Genauigkeit und Validierungsgenauigkeit überprüft wird. Genauigkeit bedeutet, dass in der Validierungsphase alle Daten, welche auch im Trainingsprozess verwendet wurden, von dem neuen Modell klassifiziert werden.

Dieses Ergebnis wird dann mit der bekannten Klassifizierung der Daten abgeglichen und eine Übereinstimmungswahrscheinlichkeit errechnet, welche dann als Genauigkeit des Modells gilt. Um die Validierungsgenauigkeit zu berechnen wird derselbe Prozess auch mit den Testdaten durchgeführt. Die Validierungsgenauigkeit ist wichtig um herauszufinden ob das entstandene Netz die Trainingsdaten nur auswendig gelernt hat oder wirklich neues Wissen generiert hat. Dies ist der Fall, sobald während des Trainings die Validierungsgenauigkeit immer weiter sinkt, dann spricht man von Overfitting des Netzes.

Aus dem Ergebnis der Validierungsphase sowie der sonstigen Wissensbasis kann nun eine neue Trainingsphase durchgeführt werden(vgl. 2.1). Dieser Prozess wird Epoche genannt und kann nun beliebig oft wiederholt werden und das Modell weiter verbessert werden. Das entstandene Modell stellt das neu generierte Wissen dar.

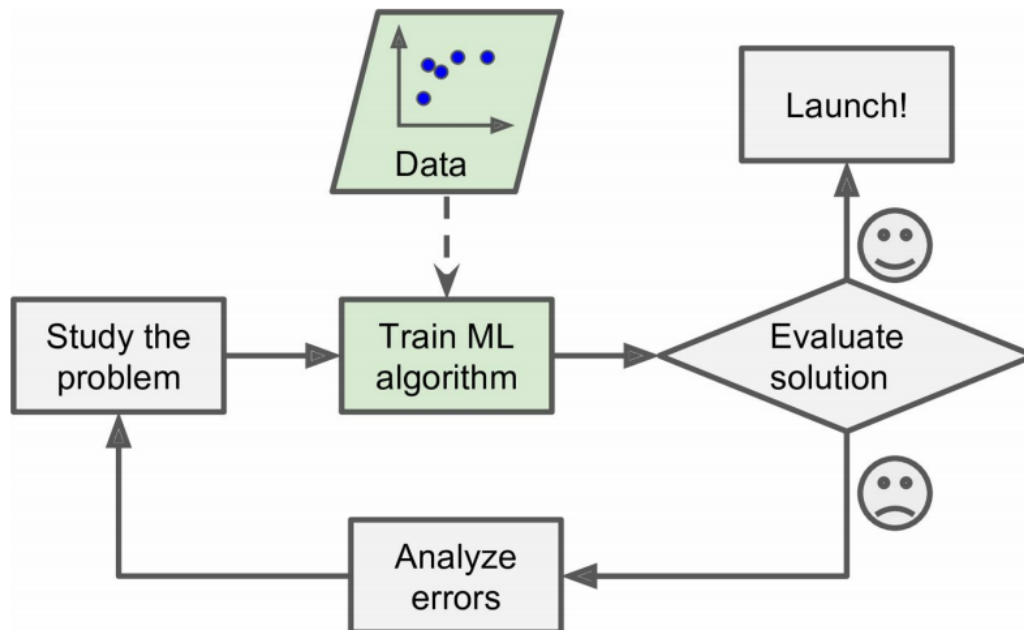


Abbildung 2.1: Trainingsprozess ( [1, Figure 1-2])

## 2.2 Neuronale Netze

[2,3, Vgl. im Folgenden] Neuronale Netze sind eine Form von Maschinellern Lernen und die heute am meisten eingesetzte Technik für Bilderkennung, Spracherkennung oder Zeitreihenanalyse.

### 2.2.1 Aufbau

Ein künstliches neuronales Netz ist an das menschliche Gehirn angelehnt und soll dessen neuronales Netz sowie dessen Verhalten abbilden. Dementsprechend besteht ein solches Netz aus mehreren Neuronen und Schichten von Neuronen welche über Synapsen miteinander verbunden sind.

#### Neuronen

Neuronen bestehen aus Eingängen, auch Dendriten genannt, welche durch eine Eingangsfunktion, eine Aktivierungsfunktionen sowie eine Ausgabefunktion geleitet werden. Die Ausgabe eines Neurons besteht daraus dass dieses je nach Eingabe und Art entweder angeregt(„gefeuert“) wird oder nicht.

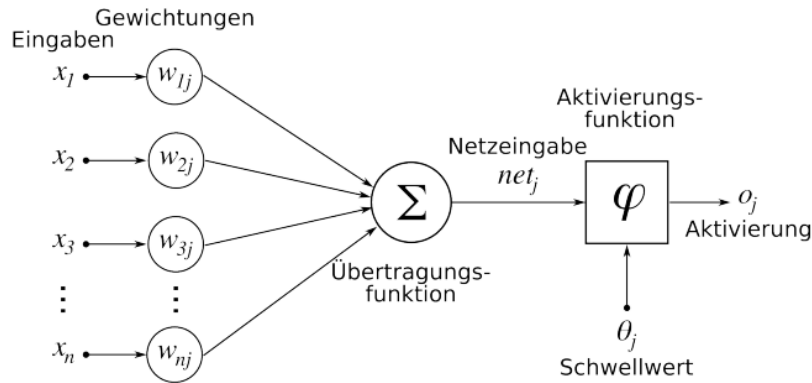


Abbildung 2.2: Aufbau eines Neurons

Die Eingangsfunktion berechnet aus den verschiedenen Eingängen den effektiven Eingang welcher dann weiter von der Aktivierungsfunktion verarbeitet wird. Diese effektive Eingabe ist dann die Netzeingabe in das Neuron. Meist wird die Netzaktivität  $net$  einfach als Summe aller Eingänge  $x$  und deren Gewichtungen  $w$  errechnet:

$$net = \sum_{i=0}^N x_i w_i \quad (2.1)$$

Die Aktivierungsfunktion dient zur eigentlichen Auswertung der Eingabe bzw. Netzaktivität. Die Aktivierungsfunktionen erzeugt ein Aktivierungspotential welches von der Ausgangsfunktion ausgewertet wird.

Die Ausgangsfunktion entscheidet schlussendlich ob das Neuron angeregt wird oder nicht. Hierzu wird geprüft ob das Aktivierungspotential der Aktivierungsfunktion einen bestimmten Schwellenwert übersteigt. Dieser Schwellenwert wird mithilfe einer Schwellenwertfunktion errechnet. Die Schwellenwertfunktion muss monoton wachsend sein, kann jedoch verschiedene Formen annehmen. Sie kann linear Verlaufen wie es beim Perzeptron der Fall ist, nicht-linear oder eine Sprungfunktion sein. Durch eine nicht-lineare Funktion wie die Sigmoid-Funktion lassen sich mächtigere neuronale Netze entwickeln weshalb heutzutage meist eine solche zum Einsatz kommt.

## Topologie

Bei einem neuronalen Netz sind dessen Neuronen über Synapsen, welche bestimmte Kantengewichte bzw. Biases haben, verbunden. Durch verschiedene Kantengewichte können Eingänge oder bestimmte Features priorisiert werden. Dies bedeutet, dass Eingänge welche größeren Einfluss auf das erwartete Ergebnis haben, höher gewichtet werden können und somit stärker in das Ergebnis mit einbezogen werden.

Durch verschiedene Verbindungen zwischen Neuronen ergeben sich verschiedene Topologien von Netzen:

**Vorwärts-verkettete Netze**, auch feed-forward Netze genannt, bestehen aus verschiedenen Neuronen-Schichten(hidden layers), welche nur mit der nächst höheren Schicht verbunden sind. Somit verlaufen die Daten nur in eine Richtung, vom Eingang zum Ausgang.

**Rekurrente Netze** können auch aus mehreren Neuronen-Schichten bestehen wobei hier auch Verbindungen in tiefere Schichten möglich sind. Dies bedeutet, dass auch Ergebnisse von vorherigen Durchläufen des Netztes als Eingang mit einbezogen werden können.

**Voll-vernetzte Netze** sind neuronale Netze, bei denen alle Neuronen mit allen anderen Neuronen vernetzt sind. Wobei hierbei keine wirkliche Struktur erkennbar ist und ein sehr großer Rechenaufwand besteht.

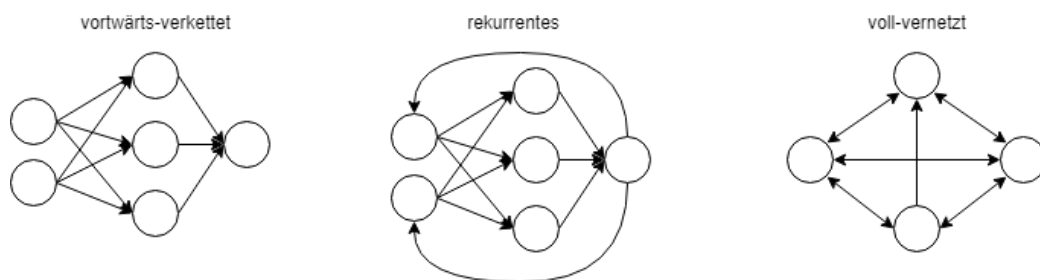


Abbildung 2.3: Netztopologien

## Lernen

Bei neuronalen Netzen ist das vorhandene Wissen durch die Kantengewichte repräsentiert. Daraus folgt, dass verschiedene Klassifikationen von neuronalen Netzen, mit derselben Topologie, sich nur von den Kantengewichten abhängt. Dementsprechend lernt ein neuronales Netz durch das sequentielle Anpassen

der Gewichte. Dazu gibt es verschiedene Verfahren wie diese Gewichte angepasst werden können.

Beim **Backpropagation Lernen** werden in der Trainingsphase die Ergebnissen eines Durchlaufs mit den erwarteten Ergebnissen verglichen und der Ausgabefehler berechnet. Mithilfe dieses Fehlers wird nun Schichtweise versucht den Fehler auf einzelne Neuronen bzw. Gewichte bis hin zur Eingabeschicht zurückzuführen. Danach werden nun bei verschiedenen Neuronen Änderungen der Gewichte durchgeführt um den Fehler der Ausgabefunktion zu minimieren. Hierbei wird für jedes Gewicht mithilfe verschiedener Gradientenverfahren eine Änderung berechnet.

Beim **Batch Lernen** wird anders als beim Backpropagation Lernen nicht nach jeden Durchlauf des Netzes die Gewichte angepasst sondern erst nachdem das Trainingsset komplett durchlaufen wurde. Dies führt zu weniger Genauigkeit das nicht auf spezielle Eigenschaften einzelner Einträge eingegangen. Jedoch Ist dieses Verfahren sehr viel weniger Rechenaufwand und es können mehr Trainingsdaten in gleicher Zeit einbezogen werden.

## Perzeptron

Ein sehr einfaches neuronales Netz ist das Perzeptron, welches 1958 von Frank Rosenblatt entwickelt wurde. Das Perzeptron besteht aus zwei Eingabeparametern, welche in ein Neuron gegeben werden und von diesem mithilfe der Kantengewichte und der Aktivierungsfunktion eine Ausgabe liefert.

Ein Beispiel ist die Abbildung des booleschen „Und“-Operators welche wie in Abbildung 2.4 mit einem Neuron dargestellt werden kann. Jedoch können mit einem Neuron nur sehr einfache Funktionen abgebildet werden. Komplexere Funktionen können über mehrere Neuronen bzw. mehrere Schichten von Neuronen abgebildet werden.

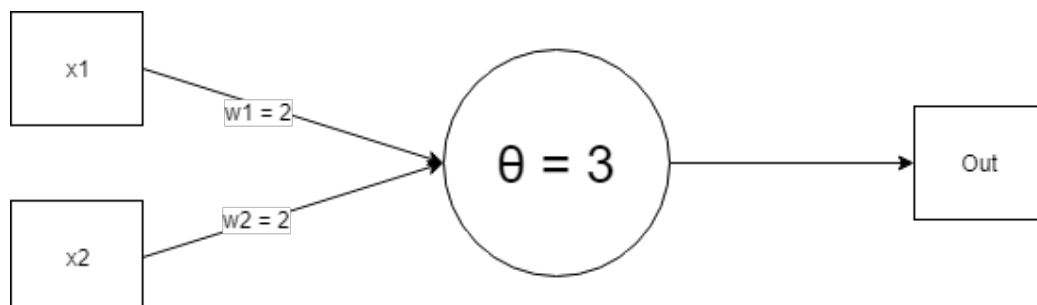


Abbildung 2.4: UND-Operator als Perzeptron

x1	x2	$(x_1 * w_1) + (x_2 * w_2)$	out
0	0	$(0 * 2) + (0 * 2) = 0 < 3$	0
0	1	$(0 * 2) + (1 * 2) = 2 < 3$	0
1	0	$(1 * 2) + (0 * 2) = 2 < 3$	0
1	1	$(1 * 2) + (1 * 2) = 4 < 3$	1

Tabelle 2.1: Wahrheitstabelle des Perzeptron

### 2.2.2 CNN - Convolutional Neural Network

### 2.2.3 RNN - Recurrent Neural Network

## 2.3 Physikalische Grundlagen zur Netzaktivität

## 2.4 Erhebung der Messdaten

Um aussagekräftige Analysen und Klassifikationen über ein Stromnetz bzw. die Geräte in einem Stromnetz mit Maschinellern Lernen machen zu können, werden viele Trainings- und Testdaten benötigt. Die Daten bestehen aus verschiedenen physikalischen Größen, die zu einem bestimmten Zeitpunkt in einem Stromnetz auftreten. Zu diesen Größen gehört die allgemeine Netzspannung, die Netzfrequenz sowie sieben harmonischen Oberwellen (vgl. 2.3). Um einen allgemeinen Überblick über den Verlauf der Netzaktivität zu erhalten sowie verschiedene Zeiten und Geräte vergleichen zu können, müssen Daten über lange Zeiträume erhoben werden.

Zur Erhebung der Werte zur Netzaktivität wurde ein WeSense-Messgerät<sup>1</sup> verwendet. Dieses Gerät misst alle benötigten Werte und sendet diese über einen MQTT-Broker<sup>2</sup> an einen Service, welcher dann die Daten aufbereitet und in einer MSSQL Datenbank abspeichert. Die Werte werden sekundlich gemessen und in die Datenbank gespeichert oder an registrierte Websocket-Verbindungen gesendet.

---

<sup>1</sup><http://www.wesense-app.com/home-en/>

<sup>2</sup>Message Queuing Telemetry Transport

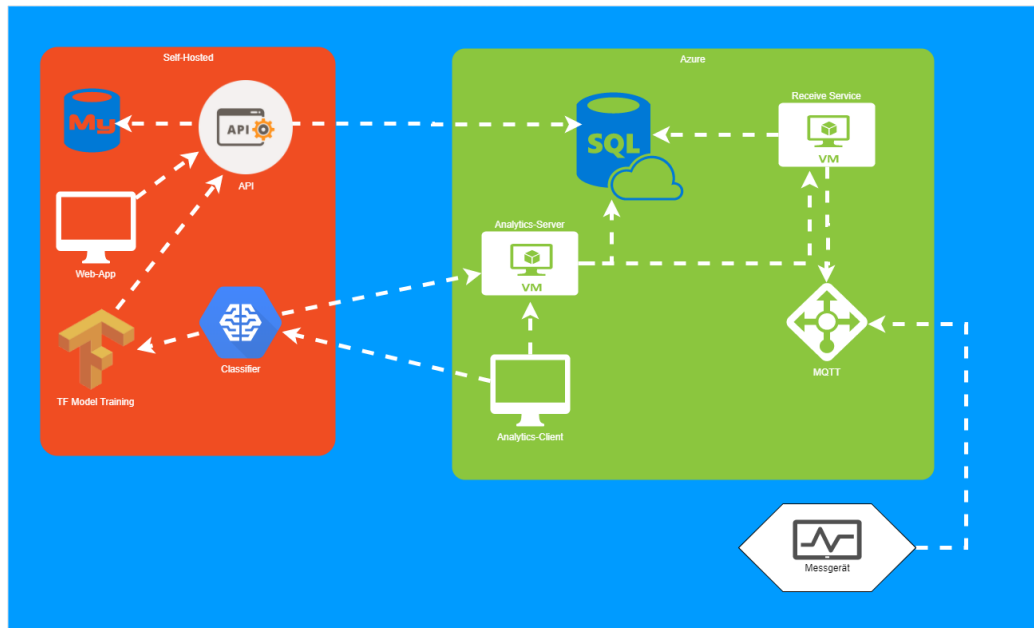


Abbildung 2.5: Complete Architecture

### 2.4.1 Klassifikation der Messdaten

Durch die oben beschriebene Erhebung sind die physikalischen Werte zu bestimmten Zeitpunkten bestimmt worden. Zusätzlich wird nun zur Identifikation der Geräte sowie zum Maschinellen Lernen, genau definierte Zeiträume benötigt in denen bestimmte Geräte aktiv waren. Dies bedeutet, dass jedem Zeitpunkt ein oder mehrere Geräte zugewiesen werden. Die einem Gerät zugewiesene Daten werden im weiteren Verlauf gelabelte Daten genannt.

Um diese gelabelten Daten zu erheben gibt es verschiedene Möglichkeiten. Die Daten können entweder durch eine Person, welche Zeiten zu denen sicher Geräte aktiv waren manuell erfasst werden, oder durch eine Maschine automatisch erhoben werden. Zur automatischen Erhebung wird ein Messgerät benötigt, welches zwischen dem zu messenden Gerät und dem Stromnetz zwischengeschaltet wird und sobald Strom fließt Daten an den Service übermittelt. Die automatisierte Methode ermöglicht es präzisere Zeiten als auch mehr Zeiten und Haushaltsgeräte gleichzeitig zu erfassen.

Da jedoch für die automatisierte Methode ein weiteres Gerät benötigt wird, werden die Daten manuell durch eine Person erfasst.

Für die manuelle Datenerfassung wurde eine progressiv Web-App(vgl. Ab-

bildung 2.6) mit einer einfachen MySQL-Datenbank im Backend erstellt, mit der die Daten sehr einfach erfasst und abgespeichert werden können.

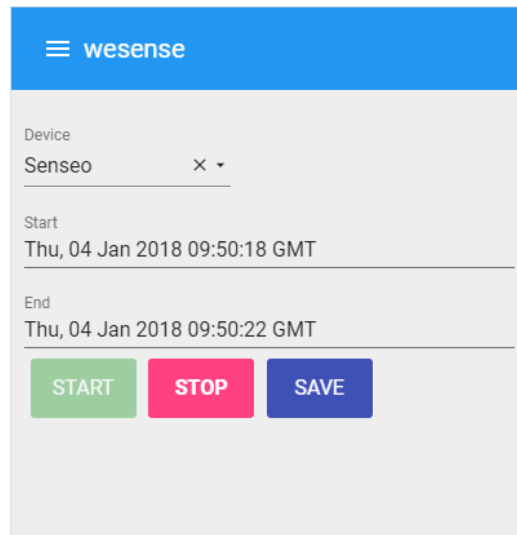


Abbildung 2.6: Screenshot der progressive Web-App

## 2.5 Visualisierung

Zusätzlich zur manuellen Erhebung der Daten wurden zur besseren Analyse der Daten verschiedene Visualisierungsmöglichkeiten implementiert. Zum einen können die verschiedenen Physikalischen Größen eines Gerätes zu einem bestimmten Zeitpunkt miteinander verglichen werden. Außerdem können auch bestimmte Größen zu verschiedenen gelabelten Zeiträumen eines Gerätes verglichen und analysiert werden. Durch diese Visualisierung können sehr gut und genau Gemeinsamkeiten in verschiedenen Größen oder Zeiten erkannt werden.

Es werden verschiedene Diagramme sowie Normalisierungen der Daten zur Analyse bereitgestellt. Es besteht die Möglichkeit die Daten in einem Liniendiagramm sekundlich oder in frei wählbaren zusammengefassten Datenpunkten, sogenannten Klassen, anzuzeigen. Des weiteren können Histogramme mit verschiedenen Klassen gewählt werden.





Abbildung 2.7: Screenshot eines gelabelten Zeitraumes aus der Web-App

# Kapitel 3

## Ausführung

Dieses Kapitel beschreibt die Vorgehensweise von der ersten manuellen Analyse der Daten bis zu einem funktionierenden neuronalen Netz. Außerdem werden verschiedene neuronale Netze miteinander verglichen um die beste Vorgehensweise zur Klassifikation von Geräten zu finden. Als Beispielprodukt für die Klassifizierung wurde eine Senseo Kaffeemaschine gewählt, da durch häufige Benutzung viele Daten erhoben werden können. Außerdem wurde eine Mikrowelle als zweite Gerät mit wenigen Daten gewählt um verschiedene Parameter und Kennzahlen zu vergleichen. Hierbei soll die Anzahl der Daten sowie die Klassifikation von mehreren Geräten innerhalb eines Neuronalen Netzes untersucht werden.

### 3.1 Manuelle Analyse

Zur manuellen Analyse der Daten wird das in 2.5 beschriebene Tool verwendet. Zunächst werden sehr aussagenkräftige Größen wie die Spannung oder die Frequenz der Kaffeemaschine verwendet und mit anderen aktiven Zeiträumen der Kaffeemaschine verglichen. Hierbei kann ein sehr spezifischer Verlauf der Spannung erkannt werden.

Wie in Abbildung reffig:Spannungsverlauf zu sehen ist, ist die Kurve zu Beginn start fallend und verbleibt dann eine gewissen Zeit auf diesem Tief. nach einem längeren steigenden Abschnitt fällt die Kurve wieder bis der Zubereitungsvorgang beendet wurde und wieder steigt.

Diesem Verlauf können nach mehreren Beobachtungen bestimmte Vorgänge einer Kaffeezubereitung zugeordnet werden. Zu Beginn der Kaffeezubereitung wird die Kaffeemaschine manuell eingeschaltet. Dies führt automatisch zum erwärmen des Brühwassers, welches dem ersten Fallen der Kurve zuge-

ordnet werden kann. Da dort viel Energie benötigt um das Wasser zu erhitzen steigt der Stromverbrauch der Kaffeemaschine stark an und somit fällt die Netzspannung stark ab. Die Netzspannung bleibt solange auf einem gewissen Tiefpunkt mit minimaler Netzschwankung bis das Wasser erwärmt wurde und ein weiterer manueller Schritt zum fortfahren des Prozesses notwendig ist. Nach Wahl der Tassengröße wird dann der Brühvorgang gestartet. Dabei wird das erhitzte Wasser mit einem gewissen Druck durch einen Kaffeepad gepresst. Da dieser Druck bei der Senseo Kaffeemaschine durch eine elektronische Pumpe erzeugt wird, sinkt demnach die Netzspannung wird ab bis der komplette Kaffee durch gelaufen ist. Somit kann die Zweite Tiefpunktphase dem "Pressvorgang" der Kaffeemaschine zugeordnet werden.

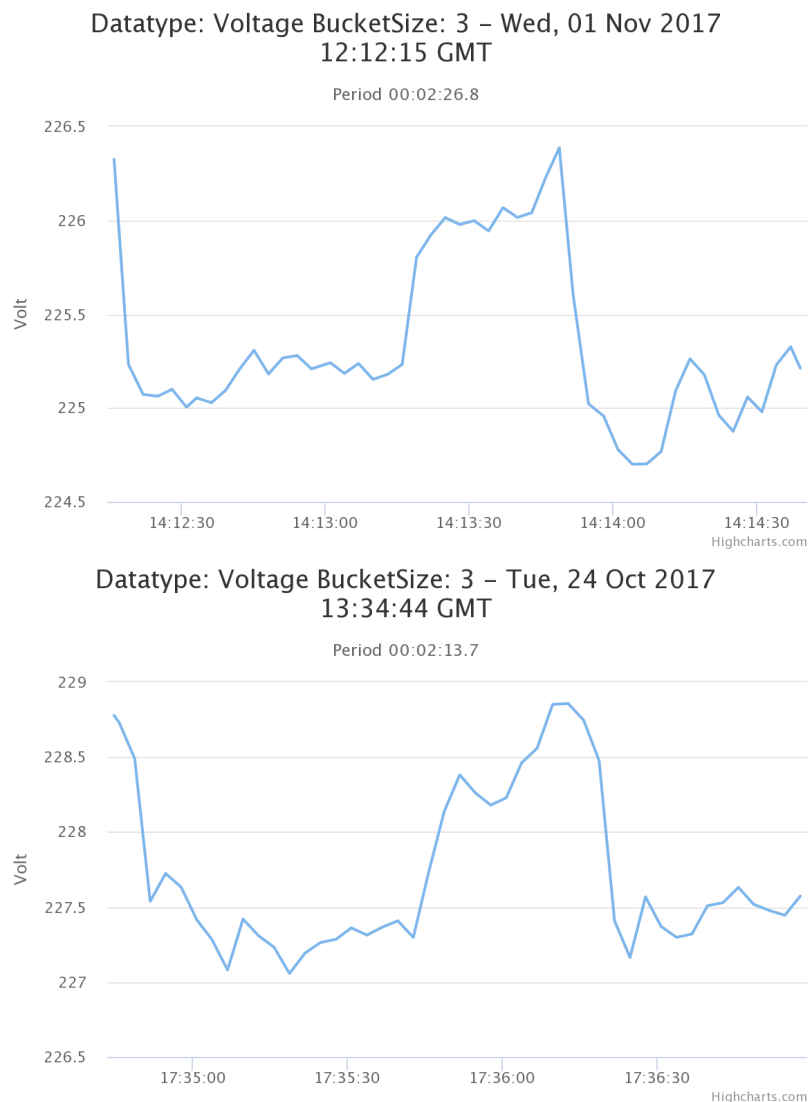


Abbildung 3.1: Spannungsverlauf der Senseo Kaffeemaschine mit einer Klasse von je 3 Datenpunkten zu 2 verschiedenen Zeitpunkten

Bei der manuellen Analyse der Mikrowelle können bei Analyse des Spannungs- oder Frequenzverlaufs leider keine hervorstechenden Merkmale oder Gemeinsamkeiten erkannt werden. Somit wird die Mikrowelle demnach einen anderen Einfluss auf die Netzaktivität ausüben. Wie bei dem Vergleich der verschiedenen harmonischen Oberwellen zu sehen ist (siehe Abbildung 3.2), besteht eine große Ähnlichkeit der Kurven der dritten harmonischen Welle.

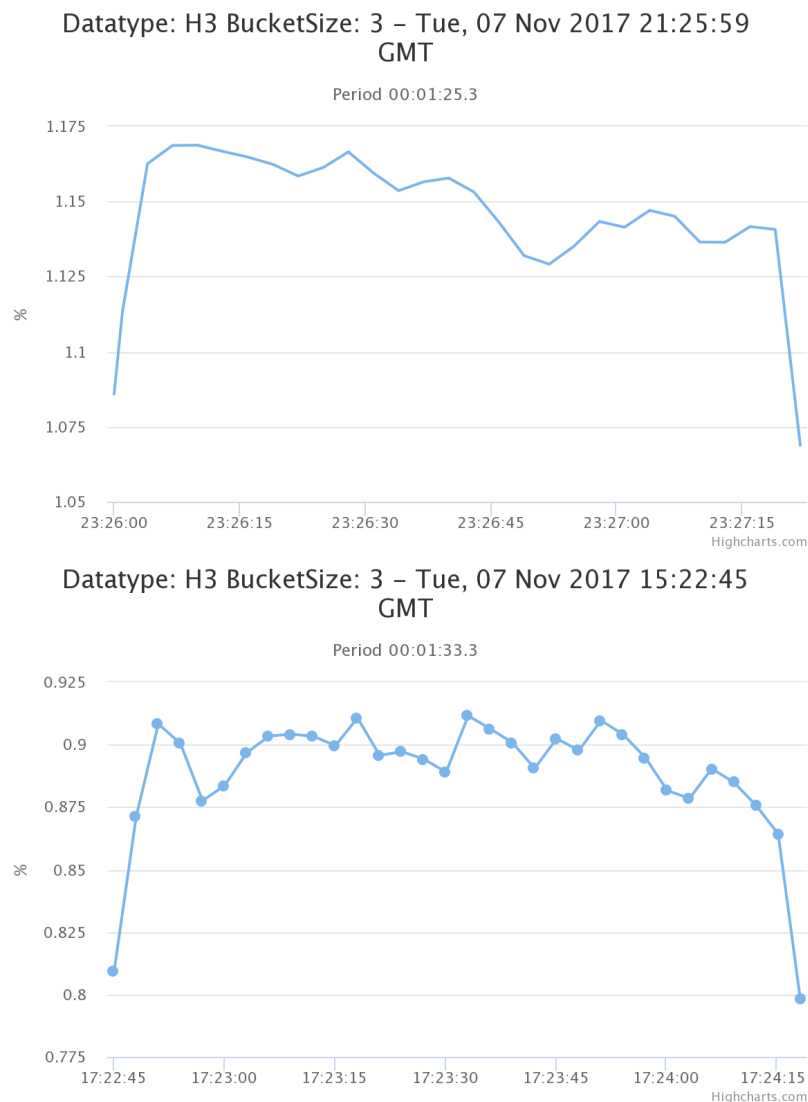


Abbildung 3.2: Verlauf der 3. harmonischen Oberwelle einer Mikrowelle mit einer Klasse von je 3 Datenpunkten zu 2 verschiedenen Zeitpunkten

Wie dieser Abschnitt zeigt können schon mit bloßen Auge bestimmte Auswirkungen der verschiedenen Geräte erkannt werden. Auch wenn einige Störungen auftreten und den Verlauf verfälschen, sollte es dennoch möglich sein diese Geräte herauszufiltern. Weiterführend wird dieser Prozess automatisiert und verbessert um auch trotz großer Störungen, Geräte präzise klassifizieren zu können.

## 3.2 Vorbereiten der Daten

Um die Klassifizierung der Geräte auf neuronalen Netzen abzubilden müssen die vorhandenen Daten in ein geeignetes Format transformiert werden. Dabei gibt es verschiedene Voraussetzungen zu beachten, sowie verschiedene Möglichkeiten die Daten zu transformieren. Die Implementierung zu diesen Kapitel lässt sich in [https://github.com/schrodit/wesense-ml/blob/master/data/data\\_manager.py](https://github.com/schrodit/wesense-ml/blob/master/data/data_manager.py) finden.

Bei konventionellen neuronalen Netzen werden Eigenschaften Objekten zugewiesen, wobei alle Objekte mit denselben Eigenschaften betitelt werden. Ein Beispiel dafür sind Tiere, bei denen anhand von Reproduktion oder Atmung in Säugetiere, Vögel, Reptilien, etc. eingeteilt wird. So können spezielle Tiere wie ein Hund eine dieser Klassen zugewiesen werden. Alle klassifizierten Tiere werden hierbei dieselben Eigenschaftsklassen mit unterschiedlichen Werten zugewiesen. Das heißt, dass einem Hund oder einer Taube die Anzahl der Beine und die Reproduktion zugewiesen werden, jedoch mit jeweils unterschiedlichen Werten.

Durch die in Kapitel 2.4 beschriebene Erhebung der Daten, werden die Stromdaten in einem bestimmten Format von der API bereitgestellt. Es werden 2 verschiedene Listen zurückgegeben wobei eine die reinen Messdaten enthält und die andere die Labels für diese Daten.

Die reinen Messdaten werden zusammengetragen, indem zu den gelabelten Zeiträumen jeweils alle sekundlich gemessenen Werte des Stromnetzes als eine Matrix eingefügt werden.

Zusätzlich zu den manuell klassifizierten Daten werden noch zufällig ausgewählte Zeitabschnitte hinzugefügt um die Möglichkeit „kein Gerät war aktiv“ abzubilden. Diese Abschnitte werden gewählt, indem zufällig Daten aus der Datenbank gewählt werden bei denen mit großer Wahrscheinlichkeit kein zu klassifizierendes Gerät aktiv war. Eine dieser Zeiträume könnte zum Beispiel von 00:00 bis 5:00 Uhr, da zu dieser mit hoher Wahrscheinlichkeit kein Gerät aktiv war.

Somit ergibt sich eine drei dimensionale Matrix welche in der ersten Dimension die Zeiträume abbildet, in der zweiten Dimension die sekundliche Zeitreihe und die dritte die vorhandenen physikalischen Größen.

Listing 3.1: Datenstruktur der gelabelten Messdaten, die von der API bereit

gestellt wird

```
[
  [
    [u, f, h3, h5, h7, h9, h11, h13, h15],
    [u, f, h3, h5, h7, h9, h11, h13, h15],
    [u, f, h3, h5, h7, h9, h11, h13, h15],
    ...
  ]
]
```

Da Neuronale Netze feste Eingabegrößen benötigen, werden die Daten in Batches aufgeteilt. Batches bedeutet, dass die Zeitreihe in gleich große Abschnitte aufgeteilt wird. Dementsprechend wird die Matrix  $A[m, n, 9]$ , zu einer Matrix mit einheitlicher Größe transformiert, sodass die entstehende Matrix zum Beispiel die Form  $B[m, 20, 9]$  annimmt.

Um diese neue Matrix zu erhalten wird folgender Algorithmus auf die alte Matrix angewandt:

---

**Algorithm 1** Batch Generierung

---

```
1: function GENERIEREBATCHES( $A, b$ )
2:                                      $\triangleright A$  - float[...],  $b$  - int (Batchgröße)
3:    $B = \text{Array}[\dots]$ 
4:   for  $i = 0$  to  $\text{length}(A)$  do
5:     for  $j = 0$  to  $\text{length}(A[i]) - b$  do
6:        $start = j$ 
7:        $end = j + b$ 
8:        $I_b = \text{length}(B) + 1$ 
9:        $B[I_b] = A[i][start : end]$ 
10:    end for
11:  end for
12:  return  $B$ 
13: end function
```

---

### 3.3 Neuronales Netz

Wie Abschnitt 3.1 zeigt ist es möglich mit herkömmlichen manuellen Methoden verschiedene Geräte innerhalb eines Verlaufs zu klassifizieren. Somit

sollte dies auch mit maschinellem Lernen möglich sein.

Um nun die Geräte maschinell zu klassifizieren, werden drei verschiedene „neuronale Netz“ Modelle erstellt und miteinander verglichen. Die drei Modelle beinhalten ein „Recurrent Neural Network(RNN)“, ein „Convolutional Neural Network(CNN)“ sowie eine Mischung aus diesen Ansätzen. Bei allen Modellen wurde als Gradientenverfahren die Adam-Optimierung gewählt, da diese bei mehr Rechenaufwand bessere Ergebnisse erzielt.

## CNN

Grundlegend besteht das Convolutional Neural Network aus einem „Input Layer“, drei „Convolutional Layers“, einem „Dropout Layer“ sowie einem „Output Layer“ (vgl. [https://github.com/schrodit/wesense-ml/blob/master/classification/cnn\\_classification.py](https://github.com/schrodit/wesense-ml/blob/master/classification/cnn_classification.py)).

Das „Input Layer“ ist der Eingang zum neuronalen Netz, welcher die Rohdaten annimmt und an das eigentliche Netz weiterleitet. Es ist ein weiteres „Convolutional Layer“ und nimmt einen Eingabe-Tensor (Matrix innerhalb eines neuronalen Netzes), welcher an die Batchgröße angepasst ist. Dementsprechend hat der Eingabetensor die Form  $A[m, b, 9]$ , mit m: Trainingsdatengröße und b: Batchgröße.

Die Eingabe wird dann weitergeleitet an die drei „Convolutional Layers“, welche mit unterschiedlicher „Hidden Layer“-Größe initialisiert werden. Wobei die „Hidden Layer“ größer werden je näher sie topologisch an dem „Output Layer“ liegen.

Das „Dropout Layer“ wird eingefügt um die Tensordimension zu verkleinern und somit das Ergebnis zu generalisieren und Overfitting zu vermeiden.

Das „Output Layer“ wandelt nun die Erkenntnisse der vorherigen Schichten in ein lesbares Format um. Hierzu wird ein „Dense Layer“ verwendet, welcher ein eindimensionaler Tensor mit einem Feld pro klassifizierbaren Objekt ist und die Softmax-Funktion als Aktivierungsfunktion beinhaltet. Dieser eindimensionale Tensor repräsentiert pro Feld die Ausgabe pro zu klassifizierendem Gerät.



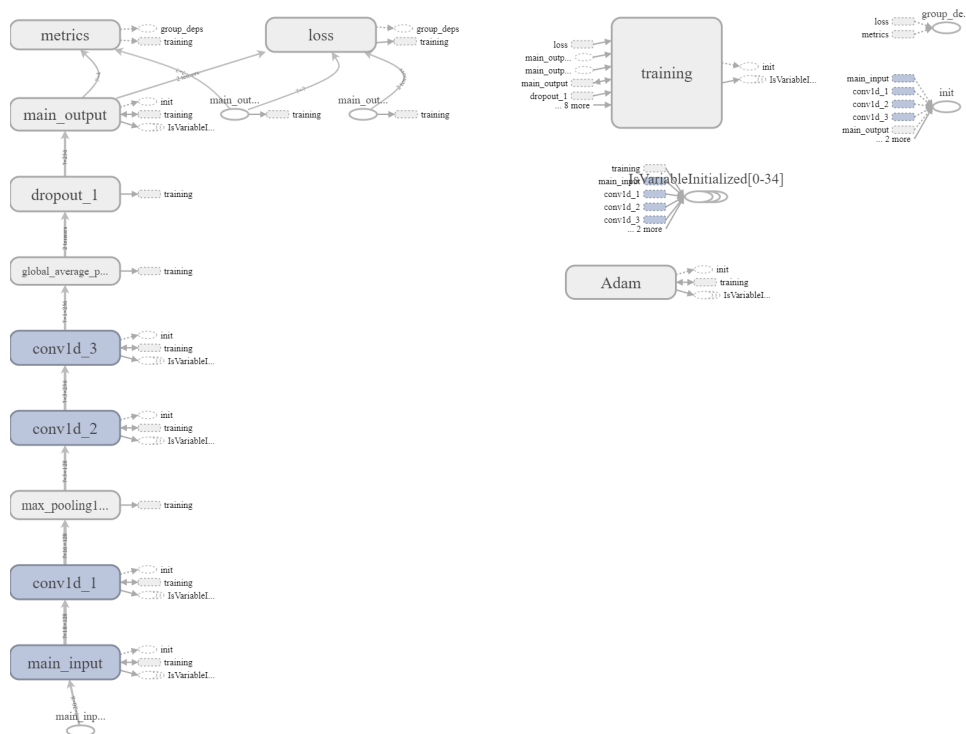


Abbildung 3.3: Convolutional Neural Network

## RNN

Das „Recurrent Neural Network(RNN)“-Model besteht aus einem „Input Layer“, zwei darauffolgende „LSTM(Long short-term memory) Layer“ und einem „Output Layer“ (vgl. [https://github.com/schrodit/wesense-ml/blob/master/classification/rnn\\_classification\\_keras.py](https://github.com/schrodit/wesense-ml/blob/master/classification/rnn_classification_keras.py)).

Das „Input Layer“ ist ein weiteres „LSTM Layer“ welche denselben Eingabetensor annimmt, wie das CNN-Model.

Es folgen zwei weitere „LSTM Layer“ einer „Hidden Layer“-Größe von 64 pro Schicht. Dieselbe Größe wird auch im Eingabe-„LSTM Layers“ verwendet.

Das „Output Layer“ ist, wie beim Convolutional Neural Network-Model, ein „Dense Layer“, der aus der Berechnung des Netzes eine verwertbare Ausgabe erzeugt.





Diese Standardparametern werden wie folgt festgelegt:

Lernrate = 0.001

Datentypen = ['u', 'h3', 'h5', 'h7', 'h9', 'h11', 'h13', 'h15']

Batchgröße = 20

Epochen = 30

Um die Auswirkung einzelner Parameter auf das Trainingsergebnis sehen zu können wird jeweils ein Parameter verändert und verschiedene Werte dieses Parameters miteinander verglichen.

Als Maßstab der Beeinflussung eines Parameters auf das Ergebnis eines Models wird die Genauigkeit sowie die Validierungsgenauigkeit in Prozent gewählt. Diese Werte werden nach dem Training eines Models, anhand von Test- bzw. Validierungsdaten errechnet (vgl. Abschnitt 2.1.3). Somit kann mit der Genauigkeit das Erlernen der Trainingsdaten überprüft werden, sowie mit der Validierungsgenauigkeit Over- bzw. Underfitting des Models nachgewiesen werden.

Nach einigen Trainingsversuchen (vgl. Tabelle 3.1) mit den Rohdaten wurde festgestellt, dass mit diesen Daten keine aussagekräftigen Ergebnisse erzielt werden können. Als Lösung dieses Problems, werden nach einigen Versuchen zwei verschiedene Verfahren gewählt mit denen die Rohdaten transformiert werden.

Zum einen wird eine Normalisierung der Daten in Werte zwischen 0-1 durchgeführt, sodass größere Werte wie die Spannung, welche bei ca. 230V liegt, keinen höheren Einfluss auf das Ergebnis haben kann als die Frequenz.

Zusätzlich werden die Ableitungen über alle Werte der Rohdaten berechnet. Dies führt dazu, dass vom neuronalen Netz der Verlauf der Kurve berücksichtigt wird um Geräte zu erkennen und nicht die reinen Rohwerte, welche je nach Netzwerk und aktueller Netzaktivität stark abweichen können.

Model	Genauigkeit	Validierungsgenauigkeit
CNN	85,69%	85,78%
RNN	87,99%	86,68%
MIX	48,13%	48,42%

Tabelle 3.1: Ergebnis: Ableitung

## 3.5 Auswertungsalgorithmus

Nach der Klassifikation der Batches mit einem der neuronalen Netze, sind für diese klassifizierten Batches mit n Datenpunkten für jeden Punkt n Klas-

sifizierungen vorhandenen(vgl. Algorithmus 3.2). Somit muss diese Klassifikation ausgewertet werden um genau sagen zu können, zu welchen Datenpunkten(Zeitpunkten) welche Geräte aktiv waren und welches Gerät am wahrscheinlichsten aktiv war. Hierzu wird der Algorithmus in 3.5 verwendet, welcher die Treffer sowie die genaue Wahrscheinlichkeit pro Klasse zu einem bestimmten Zeitpunkt berücksichtigt. Dieser Algorithmus bestimmt die Anzahl der Gewinnerklassen(Klasse mit der höchsten Wahrscheinlichkeit) aller Batches in denen der bestimmte Datenpunkt vorhanden ist, und bestimmt nun aus dieser Anzahl wiederum die Gewinnerklasse(Klasse mit den meisten Treffern). Außerdem wird die Wahrscheinlichkeit aller Klassen in allen Batches des Datenpunktes summiert und wiederum die Gewinnerklasse bestimmt. Somit erhält man zwei verschiedene Gewinnerklassen, welche wiederum miteinander verglichen werden um die endgültige Gewinnerklasse zu bestimmen.

Der Vorteil dieses Verfahrens ist die Berücksichtigung aller Klassifikationen sowie deren einzel Wahrscheinlichkeit, sodass sich eine genauere Klassifizierung ergibt, wie Abbildung 4.2 zeigt.

---

**Algorithm 2** Ergebnisauswertung

---

```
1: function EVALUATEPREDICTION( $P, b$ )
2:    $\triangleright P$  -  $\text{int}[]$  (Ergebnis eines neuronalen Netzes),  $b$  -  $\text{int}$  (Batchgröße)
3:
4:    $A = \text{int}[]$ 
5:   for  $i = 0$  to  $\text{length}(P)$  do
6:
7:     // Hit-Punkte aller Klassen für einen Zeitpunkt
8:      $\text{ClassesHit} = \text{int}[]$ 
9:     // Wahrscheinlichkeit aller Klassen für einen Zeitpunkt
10:     $\text{ClassesPerc} = \text{float}[]$ 
11:
12:    if  $i < b$  then
13:       $\text{start} = 0$ 
14:       $\text{end} = i$ 
15:    else
16:       $\text{start} = i - b$ 
17:       $\text{end} = i$ 
18:    end if
19:
20:    for  $d = \text{start}$  to  $\text{end}$  do
21:       $\text{maxClassIndex} = \text{argmax}(\text{pred}[d])$ 
22:       $\text{ClassesHit}[\text{maxClassIndex}] ++$ 
23:      for  $w = 0$  to  $\text{pred}[d]$  do
24:         $\text{ClassesPerc}[w] = \text{ClassesPerc}[w] + \text{pred}[d][w]$ 
25:      end for
26:    end for
27:
28:     $A.\text{append}(\text{EvaluateClasses}(\text{ClassesHit}, \text{ClassesPerc}))$ 
29:
30:  end for
31:  return  $A$ 
32: end function
```

---

---

**Algorithm 3** Ergebnis-Klassen-Auswertung

---

```
1: function EVALUATECLASSES(ClassesHit, ClassesPerc)
2:                                     ▷ ClassesHit - int[], ClassesPerc - float[]
3:
4:   MaxHitClass = argmax(ClassesHit)
5:   MaxPercClass = argmax(ClassesPerc)
6:
7:   if MaxHitClass == MaxPercClass then
8:     return MaxHitClass
9:   else
10:
11:     MaxHitClassPerc = ClassesHit[MaxHitClass]/Sum(ClassesHit)
12:     MaxPercClassPerc = MaxPercClass[MaxHitClass]/Sum(ClassesPerc)
13:
14:     if
15:         MaxHitClassPerc < MaxPercClassPerc
16:     then
17:         return MaxPercClass
18:     else
19:         return MaxHitClass
20:     end if
21:   end if
22: end function
```

---

# Kapitel 4

## Ergebnis

Wie die nachfolgenden Ergebnisse des Trainingsprozesses zeigen, unterscheiden sich die Ergebnisse der verschiedenen Netze stark.

Generell zeigen die Ergebnisse, dass ein Convolutional Neural Network bessere beziehungsweise genauere Ergebnisse ohne Overfitting erzielt als ein Recurrent Neural Network bei gleichen Parametern. Dies führt auch dazu, dass das „Mixed-Modell“ Ergebnisse zwischen den beiden reinen Modellen liefert. Auch sieht man sehr gut, dass ein Model mit einem Recurrent Neural Network schneller trainiert, jedoch auch sehr früh Overfittet (vgl. Tabelle 4.2).

**Batch** Bei der Wahl der Batchgröße müssen zwei Faktoren für die Bewertung beachtet werden. Wie bei den anderen Parametern sind die Genauigkeiten ein wichtiger Faktor. Hierbei ist zu sehen, dass mit steigender Batchgröße die Genauigkeit prinzipiell steigt, jedoch ein Overfitting bei zu großen Werten eintritt. Dies ist auf die Laufzeit der einzelnen Geräte zurückzuführen. Der zweite Bewertungsfaktor der Batchgröße ist die Laufzeit(Runtime) der zu klassifizierenden Geräte. Die Batchgröße darf nicht größer sein als die kürzeste Laufzeit eines zu klassifizierenden Gerätes, da sonst zu der Ungenauigkeit des neuronalen Netzes die Ungenauigkeit der Batchgröße hinzukommt. Somit sollte trotz besserer Ergebnisse mit großen Batches eine eher kleine Batchgröße gewählt werden.

**Epoche** Mit höherer Epochenanzahl, also mehreren Trainingsdurchläufen mit allen Trainingsdaten sowie Backpropagation lernen, werden alle Modelle besser trainiert und erzielen bessere Ergebnisse. Jedoch tritt bei Modellen mit einem Recurrent Neural Network bei zu großen Trainingsdurchläufen, wie oben beschreiben, Overfitting auf. Dies bedeutet, dass bei Recurrent



Neural Network-Modellen weniger Epochen durchlaufen werden sollten als bei Convolutional Neural Network-Modellen.

**Lernrate** Die Lernrate hat bei 20 Epochen nur wenig Einfluss auf das Ergebnis der neuronalen Netze. Jedoch besteht bei hoher Lernrate die Gefahr, das Overfitting schneller auftritt oder bestimmte Features durch Zufall zu stark in das Ergebnis einfließen. Hier sollte eine kleine Lernrate mit höherer Epochenzahl für besserer Ergebnisse gewählt werden.

Zusammenfassend kann somit festgestellt werden, dass die besten Ergebnisse erzielt werden können, wenn die Lernrate niedrig ist, viele Epochen trainiert wird und eine kleinere Batchgröße gewählt wird. Zur Erkennung von Mustern innerhalb einer Zeitserie mit mehreren Features eignet sich im Fall dieser Arbeit ein Convolutional Neural Network am besten.

### Batchgröße

Lernrate = 0.001

Datentypen = ['u', 'h3', 'h5', 'h7', 'h9', 'h11', 'h13', 'h15']

Epochen = 30

Model	Batchgröße	Genauigkeit	Validierungsgenauigkeit
CNN	20	96,62%	95,16%
RNN	20	84,68%	84,70%
MIX	20	89,53%	89,24%
CNN	30	97,97%	95,05%
RNN	30	89,31%	89,02%
MIX	30	94,28%	92,96%
CNN	40	98,85%	98,94%
RNN	40	92,85%	92,05%
MIX	40	96,90%	96,92%
CNN	50	99,21%	96,95%
RNN	50	95,82%	96,21%
MIX	50	98,74%	98,79%
CNN	60	99,49%	91,63%
RNN	60	96,61%	96,39%
MIX	60	99,17%	98,99%

Tabelle 4.1: Ergebnis: Batchgröße

## Epochen

Lernrate = 0.001

Datentypen = ['u', 'h3', 'h5', 'h7', 'h9', 'h11', 'h13', 'h15']

Batchgröße = 20

Model	Epochen	Genauigkeit	Validierungsgenauigkeit
CNN	10	87,38%	87,36%
RNN	10	82,01%	81,22%
MIX	10	83,28%	82,76%
CNN	20	93,93%	92,09%
RNN	20	84,41%	84,00%
MIX	20	86,68%	86,15%
CNN	30	96,75%	95,72%
RNN	30	84,88%	84,57%
MIX	30	88,23%	87,87%
CNN	40	98,18%	96,76%
RNN	40	85,18%	84,46%
MIX	40	90,90%	89,72%
CNN	50	98,86%	97,50%
RNN	50	85,80%	84,80%
MIX	50	91,93%	91,10%
CNN	60	99,15%	98,58%
RNN	60	85,67%	85,66%
MIX	60	94,09%	90,53%
CNN	70	99,39%	99,02%
RNN	70	86,63%	86,09%
MIX	70	94,52%	92,94%
CNN	80	99,57%	99,27%
RNN	80	86,22%	86,81%
MIX	80	95,56%	93,98%
CNN	90	99,89%	99,63%
RNN	90	87,64%	86,67%
MIX	90	95,96%	94,42%
CNN	100	99,49%	99,37%
RNN	100	87,29%	86,43%
MIX	100	97,39%	96,46%

Tabelle 4.2: Ergebnis: Epochen

## Lernrate

Datentypen = ['u', 'h3', 'h5', 'h7', 'h9', 'h11', 'h13', 'h15']

Batchgröße = 20

Epochen = 30

Model	Lernrate	Genauigkeit	Validierungsgenauigkeit
CNN	0.0001	97,13%	95,55%
RNN	0.0001	84,54%	84,91%
MIX	0.0001	88,45%	88,43%
CNN	0.001	96,65%	93,85%
RNN	0.001	84,98%	84,34%
MIX	0.001	90,17%	89,02%
CNN	0.01	96,91%	93,42%
RNN	0.01	84,67%	84,84%
MIX	0.01	89,71%	89,41%
CNN	0.1	96,51%	95,50%
RNN	0.1	85,06%	84,62%
MIX	0.1	89,46%	88,51%

Tabelle 4.3: Ergebnis: Lernrate

### 4.0.1 Auswertung der Ergebnisse

Nach Anwendung des in 3.5 beschriebenen Algorithmus ergeben sich die nachfolgenden Abbildungen der verschiedenen neuronalen Netze. Die Abbildungen zeigen die jeweiligen klassifizierten Geräte jeden Zeitpunktes von Abbildung 4.1. Hierbei zeigt die x-Achse den Zeitpunkt und die y-Achse das Gerät, wobei  $y: 0 \Rightarrow$  „Senseo Kaffeemaschine“,

$y: 1 \Rightarrow$  „Mikrowelle“,

$y: 2 \Rightarrow$  „Bosch Vollautomat“,

und  $y: 3 \Rightarrow$  „Kein aktives bekanntes Gerät“.

Die Abbildungen zeigen unterschiedliche Bewertungen beziehungsweise Klassifizierungen der Zeitpunkte. Gut können die Gemeinsamkeiten aller drei Klassifizierungen erkannt werden. Trotz der Einbeziehung vieler verschiedener Ergebnisse (vgl. Algorithmus 3.5), gibt es noch viele kleine Ausreißer. Diese Ausreißer sind bei allen Modellen unterschiedlich verteilt und auch mit unterschiedlicher Länge. Jedoch kann man sehr gut sehen, dass alle drei einen großen gemeinsamen Abschnitt haben (ca.  $y = [30, 190]$ ), wo die „Sen-

seo Kaffeemaschine“ erkannt wurde. Somit kann mit hoher Wahrscheinlich davon ausgegangen werden, dass zu diesem Zeitpunkt dieses Gerät aktiv war.

Die anderen Geräte wurden jedoch nie erkannt, was darauf zurückzuführen ist, dass für die anderen Geräte sehr wenige Daten vorhanden sind. Wie die Klassifikation der „Senseo Kaffeemaschine“ zeigt, können mit genügend Daten von Geräten, diese auch klassifiziert werden. Auch können durch mehr Daten die Ausreißer, welche in allen Abbildungen zu sehen sind, vermindert werden und die Bestimmung von aktiven Geräten noch präziser durchgeführt werden.

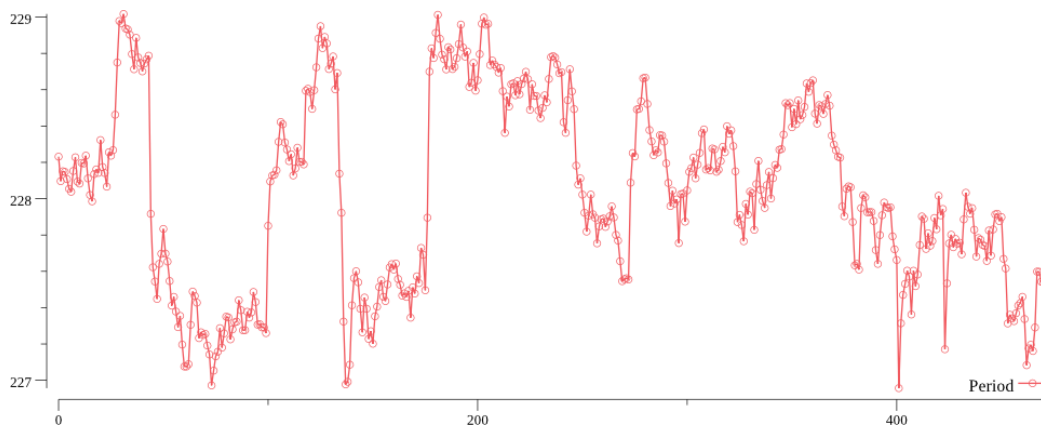


Abbildung 4.1: Spannungsverlauf in Volt von „24.10.2017 15:29:44“ bis „24.10.2017 15:41:57“

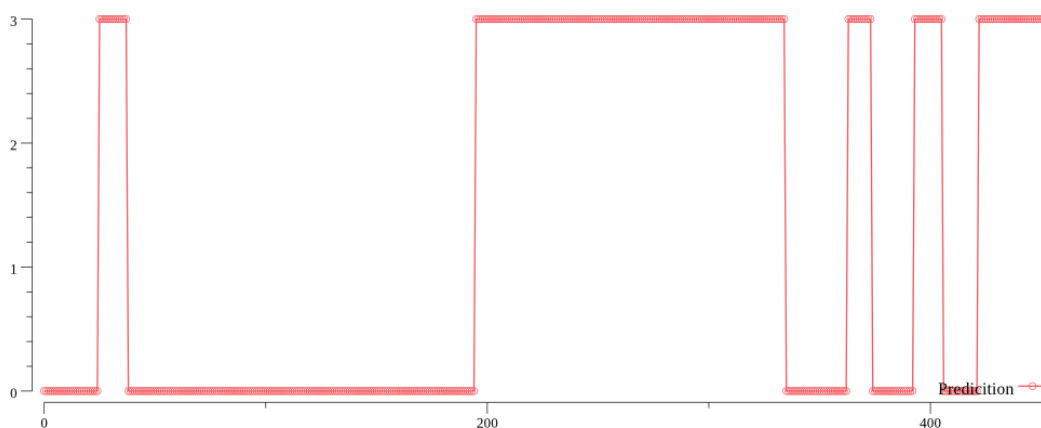


Abbildung 4.2: Klassifizierung mit CNN

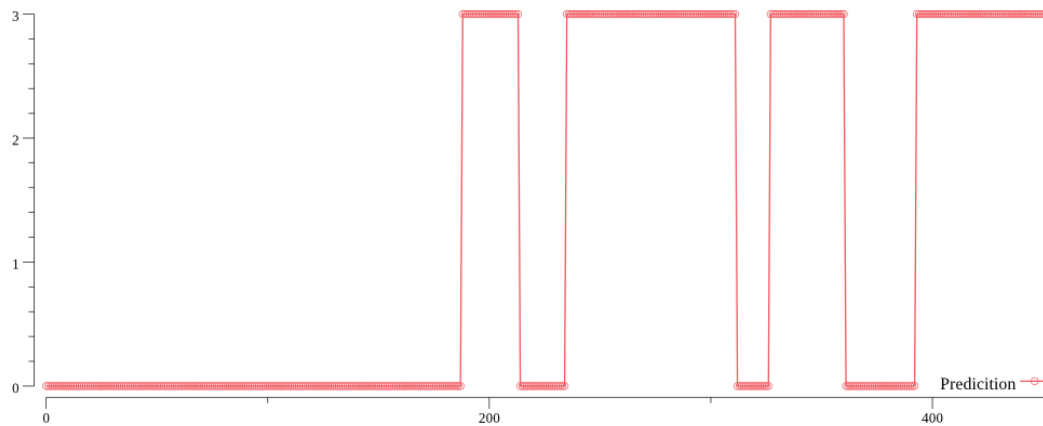


Abbildung 4.3: Klassifizierung mit RNN

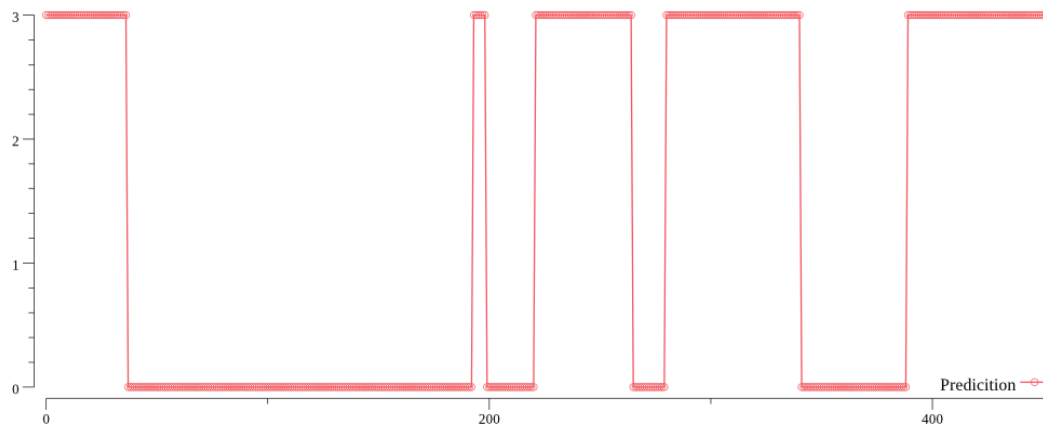


Abbildung 4.4: Klassifizierung mit Mixed RNN und CNN

### 4.0.2 Ergebnisglättung

Eine Möglichkeit die Ungenauigkeit des neuronalen Netzes zu verbessern ist die Auswertung der Ausreißer. Durch die Einbeziehung einer weiteren Eigenschaft der Geräte können einige Ausreißer erkannt und beseitigt werden. Wie bereits im Abschnitt 4 in Paragraph „Batch“ erwähnt wurde, wird die Größe der Batches klein gehalten damit Geräte mit kleineren Laufzeiten erkannt werden können. Diese Laufzeiten können auch für eine weitere Verbesserung verwendet werden. Durch die Datenbank(vgl. Kapitel 2.4), welche die manuell klassifizierten Geräte beinhaltet, wird die kürzeste Laufzeit jedes Gerätes ermittelt. Diese Laufzeit gibt an wie lange ein Gerät mindestens aktiv war. Somit wird diese Eigenschaft eingesetzt um Ausreißer, welche kürzer dauern

als die kürzeste Laufzeit, zu beseitigen. Daraus ergibt sich für die Abbildung 4.2 eine geglättete Klassifikation, siehe Abbildung 4.5.

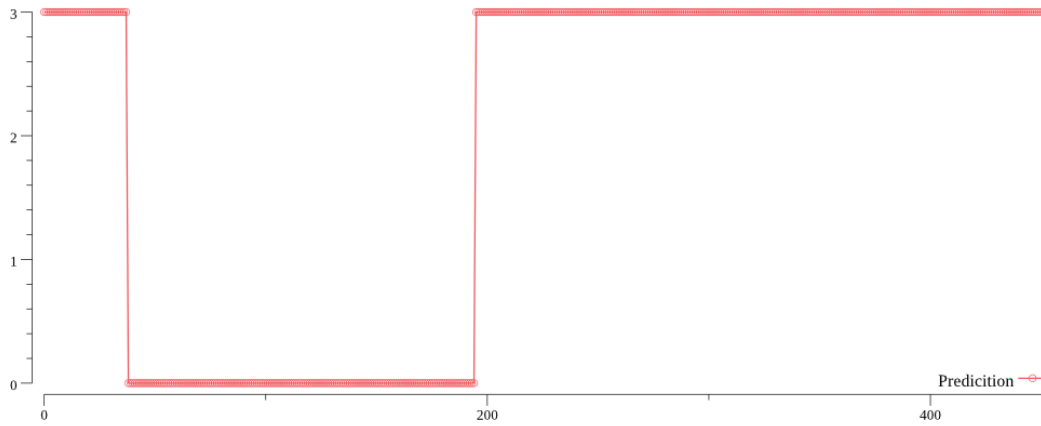


Abbildung 4.5: Geglättetes Ergebnis des CNN

### 4.0.3 Zusammenfassung

Das Ergebnis ist ein neuronales Netz welches mithilfe verschiedener Algorithmen die aktive Laufzeit verschiedener Geräte bestimmen kann. Dies bedeutet, dass bestimmt werden kann zu welchen Zeiten welche Geräte aktiv waren und benutzt wurden.

Um weitere Geräte klassifizieren zu können oder die Genauigkeit der bisherigen Geräte zu verbessern, ist es notwendig viel mehr klassifizierte Daten zu erheben.

# Kapitel 5

## Produktivbetrieb

Das in Kapitel 4 resultierende neuronale Netz sowie die Auswertung dessen Ergebnis kann bisher nur begrenzt genutzt werden. Um diese Funktionen produktiv einsetzen zu können wird eine Schnittstelle zur einfachen Verwendung benötigt.

An diese Schnittstelle werden folgende weitere Voraussetzungen gestellt.

**Echzeitanalyse** Es soll möglich sein Daten beziehungsweise Batches in Echtzeit zu analysieren.

**Periodenanalyse** Es soll möglich sein für beliebige historische Zeiträume Geräte zu klassifizieren.

**Performance** Es sollte möglich sein lange Zeitreihen mit sehr vielen Daten auch noch in annehmbarer Zeit auszuwerten. Zudem sollte es möglich sein Zeitreihen in Echtzeit zu analysieren.

**Tests** Um Fehler zu verhindern und ein stabile und sicheres Programm zu entwickeln sollen wichtige fehleranfällige Funktionalitäten getestet werden.

### 5.0.1 Implementierung

Als Schnittstelle wird eine Socket.IO-API anstatt einer gebräuchlicheren REST-API aufgrund von besserer Performance und Einheitlichkeit mit der Daten-API gewählt. Die Spezifikation dieser Schnittstelle wird, wie in Abschnitt ?? beschrieben, festgelegt. Somit ist die Echtzeitanalyse sowie die Periodenanalyse mit möglichst wenig Mehraufwand abgebildet.

Durch das Verwenden von Tensorflow und dessen Unterstützungseinschränkungen<sup>1</sup>

---

<sup>1</sup><https://www.tensorflow.org/install/>

stehen grundsätzlich zwei verschiedene Ansätze zur Wahl. Es ist möglich die Architektur auf Python, C oder GoLang aufzubauen. Da die Architektur die vorher beschriebenen Anforderungen erfüllen muss und eine einfache Benutzung der neuronalen Netze erlauben sollte, werden mögliche Architekturen auf diese Anforderungen geprüft.

Da der komplette Trainingsprozess bereits in Python implementiert wurde, werden erste Implementierung einfach dort getestet <sup>2</sup>. Jedoch ist Python schlecht geeignet für viele Anfragen mit großem Rechenaufwand, wie für die Vorbereitung der Daten erforderlich ist. Auch konnte das Auslagern der aufwendigen Rechenoperationen keine Daten in Echtzeit auswerten. Somit wird entschieden den produktiv Betrieb in GoLang zu implementieren.

Der Einsatz von GoLang kann alle benötigten Anforderungen erfüllen auch wenn die Einbindung sowie der Support von Tensorflow nicht so gut Ausfällt wie es bei Python der Fall ist. Jedoch kann mit GoLang die essentielle Performanceanforderung ohne Probleme erfüllt werden und auch die Stabilität sowie die Fehleranfälligkeit ist durch den Einsatz von GoLang erheblich verbessert. Die Implementierung (siehe <sup>3</sup>) besteht aus drei essentiellen Bausteinen welche in Module(Packages) aufgeteilt sind. Diese drei Grundbausteine sind die Datentransformation, die Klassifikation sowie dem Socket.IO-Server.

Da im Datentransformation-Modul die essentiellen und Fehleranfälligen Funktionalitäten implementiert sind, sind Tests für dieses Modul besonders wichtig. Hier wird eine Testabdeckung von mehr als 70% erreicht um die Funktionsfähigkeit dieser Funktionen sicherzustellen.

Um eine stabile, zuverlässige und fehlerfreie Applikation zu schaffen wird eine Continuous Integration-Umgebung eingeführt. Diese Continuous Integration-Umgebung führt bei jedem Commit alle Tests aus, kompiliert das Programm und baut ein Docker-Container (siehe <sup>4</sup>) in einer isolierten gleichartigen Umgebung. Somit kann eine gleichbleibende gute Qualität der Applikation sichergestellt werden.

## 5.0.2 Deployment

Die Bereitstellung der API wird als Docker-Container realisiert. Diese Deployment-Strategie bietet viele Vorteile wie Abstraktion von anderen Programmen ei-

---

<sup>2</sup><https://github.com/schrodit/wesense-ml/tree/master/prod>

<sup>3</sup><https://github.com/schrodit/wesense-ml/tree/master/go>

<sup>4</sup><https://github.com/schrodit/wesense-ml/blob/master/.drone.yml>



nes Systems bei wenig Performance-Verlust, welcher bei der Echtzeitanalyse von Nöten ist. Außerdem ermöglicht es ein automatisches Deployment mit automatischer Lastverteilung in weitere Docker-Umgebungen. Somit wird zur Bereitstellung der API nur eine Dockerumgebung benötigt und kann sehr einfach auf weitere Server portiert werden.

### 5.0.3 API-Spezifikation

#### Single Batch

##### Input:

```
1 Topic: "single-batch"
2 {
3     "data": [
4         {
5             "u": 230,
6             "f": 230,
7             "h3": 230,
8             "h5": 230,
9             "h7": 230,
10            "h9": 230,
11            "h11": 230,
12            "h13": 230,
13            "h15": 230,
14        },
15        .. x Batchsize
16    ]
17 }
```

##### Output:

```
1 Topic: 'single-prediction'
2 {
3     "data": float //- Prediction of Senseo
4 }
```

#### Period

##### Input:

```
1 Topic: "period"
2 {
3     "data": {
```

```
4         "start": "DateTime2",
5         "end": "DateTime2"
6     }
7 }
```

#### Output:

---

```
1 Topic: 'period-prediction'
2 * Series of Predictions where 0: Senseo, 1:
   Microwave, 2: Bosch, 3: Undefined, for every
   second
3 {
4     "data": [
5         Int,
6         ... end - start
7         Int
8     ]
9 }
```

# Kapitel 6

## Wirtschaftlichkeit

# Kapitel 7

## Ausblick

# Abbildungsverzeichnis

2.1	Trainingsprozess ( [1, Figure 1-2]) . . . . .	6
2.2	Aufbau eines Neurons . . . . .	7
2.3	Netztopologien . . . . .	8
2.4	UND-Operator als Perzeptron . . . . .	9
2.5	Complete Architecture . . . . .	11
2.6	Screenshot der progressive Web-App . . . . .	12
2.7	Screenshot eines gelabelten Zeitraumes aus der Web-App . . .	13
3.1	Spannungsverlauf der Senseo Kaffeemaschine mit einer Klasse von je 3 Datenpunkten zu 2 verschiedenen Zeitpunkten . . . .	16
3.2	Verlauf der 3. harmonischen Oberwelle einer Mikrowelle mit einer Klasse von je 3 Datenpunkten zu 2 verschiedenen Zeit- punkten . . . . .	17
3.3	Convolutional Neural Network . . . . .	21
3.4	Recurrent Neural Network . . . . .	22
3.5	Convolutional Neural Network mit Recurrent Neural Network	23
4.1	Spannungsverlauf in Volt von „24.10.2017 15:29:44“ bis „24.10.2017 15:41:57“ . . . . .	32
4.2	Klassifizierung mit CNN . . . . .	32
4.3	Klassifizierung mit RNN . . . . .	33
4.4	Klassifizierung mit Mixed RNN und CNN . . . . .	33
4.5	Geglättetes Ergebnis des CNN . . . . .	34

# List of Algorithms

1	Batch Generierung . . . . .	19
2	Ergebnisauswertung . . . . .	26
3	Ergebnis-Klassen-Auswertung . . . . .	27

# Literaturverzeichnis

- [1] Aurélien [VerfasserIn] Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. OReilly, Beijing, first edition edition, March 2017.
- [2] Felix Riese Sina Keller. VI 5 — neuronale netze, 2017. DHBW Karlsruhe - Wissenbasierte Systeme.
- [3] Florian Wenzel. Einführung in neuronale netze. *Neurorobotik, Institut für Informatik, Humboldt-Universität zu Berlin*, 2012.