DHBW Karlsruhe
Applied Computer Science

Studienarbeit

Klassifikation und Analyse aus Stromdaten im Haushalt mit neuronalen Netzen

Tim Schrodi
TINF15B1

Bearbeitungszeitraum:
01.10.2017 - 06.05.2018
betreut von Daniel Lindner
Prof. Dr. Johannes Freudenmann, Studiengangsleiter

Selbstständigkeitserklärung

(gemäß §5(3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 29. 9. 2015) Ich versichere hiermit, dass ich meine Projektarbeit mit dem Thema: „Klassifikation und Analyse aus Stromdaten im Haushalt mit neuronalen Netzen“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Tim Schrodi	Ort, Datum

Zusammenfassung

Das Ziel dieser Studienarbeit ist es mithilfe verschiedener Machine Learning-Methoden aus Stromdaten eines Haushalts Geräte zu klassifizieren. Diese Klassifikation beinhaltet die Bestimmung der Laufzeit dieser Geräte innerhalb einer Zeitreihe. Die Bestimmung der Laufzeit stellt dar, welche Geräte zu welchen Zeiten aktiv waren. Dazu wurden mit einem Messgerät die allgemeine Spannung, Frequenz und verschiedene Oberwellen eines üblichen Stromnetzwerks eines Privathaushalts 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 Produktivbetrieb getestet und eingesetzt.

Inhaltsverzeichnis

MSSQL	Microsoft SQL-Server
CI	Continuous Integration
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
CRM	Customer Relationship Management

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 die Automobil- oder Elektronikbranche auszuweiten.

Dazu gehört auch 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 eine nicht mehr wegzudenkende Kernindustrie. Durch neue Datencenter und den Ausbau von digitalen Systemen wächst die Nachfrage nach diesem Rohstoff immer schneller wodurch für diese Branche eine enormer Wachstumchance besteht.

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 maschinellem Lernen 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 eines Stromnetzes eines privaten Haushalts betrieben werden, erkannt werden. Mit den damit erhobenen Daten können somit Aussagen über Laufleistungen getroffen werden.

Außerdem wird auf die Wirtschaftlichkeit des resultierenden Ergebnisses, sowie deren Produktivbetrieb eingegangen.

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

[1] https://keras.io/
[2] 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 maschinellem 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 maschinellem Lernen spricht man, falls eine Maschine auf Basis von Erfahrung und Fakten „ohne speziell programmiert worden zu sein“ [4, 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 diese etwas neues gelernt [4, 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 maschinellem Lernen hat meist Vorteile gegenüber herkömmlichen statistischen Methoden wenn 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 mit der bekannten Klassifizierung der Daten abgeglichen und eine Übereinstimmungswahrscheinlichkeit errechnet, welche 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.

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

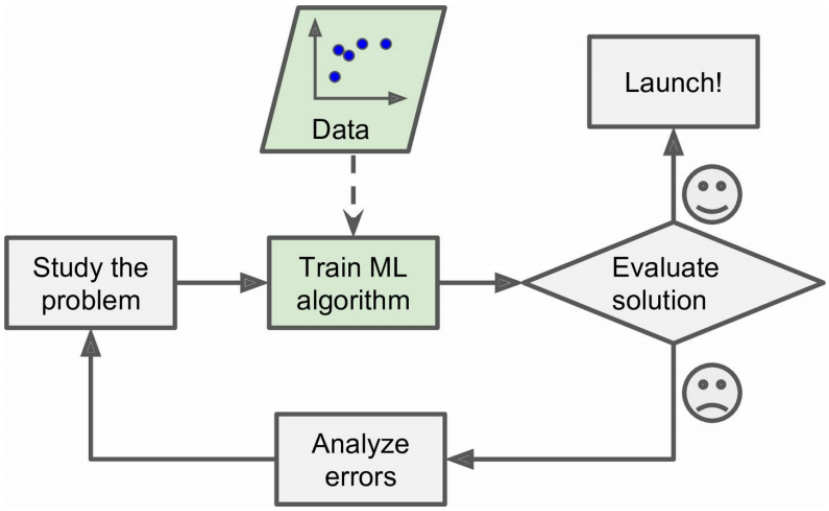

Abbildung 2.1: Trainingsprozess ([4, Figure 1-2])

2.2 Neuronale Netze

[10, 11, Vgl. im Folgenden]
Neuronale Netze sind eine Form von maschinellem Lernen und die heute am meisten eingesetzte Technik für Bilderkennung, Spracherkennung oder Zeitreihenanalyse.

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 Eingangs-, Aktivierungs- sowie Ausgabefunktion geleitet werden. Die Ausgabe eines Neurons kann je nach Eingabe und Art entweder angeregt („gefeuert“) werden oder nicht.

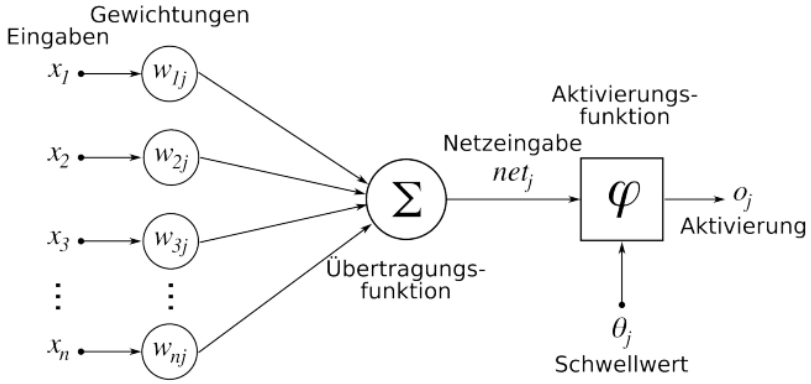

Abbildung 2.2: Aufbau eines Neurons

Die Eingangsfunktion berechnet aus 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:

$$net = \sum_{i=0}^{N} x_i w_i \tag{2.1}$$

Die Aktivierungsfunktion dient zur eigentlichen Auswertung der Eingabe bzw. Netzaktivität. Diese 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 heuzutage 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 Verbindungen zwischen Neuronen ergeben sich verschiedene Topologien von Netzen:

Vorwärts-verkettete Netze (feed-forword) 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 (Recurrent Neural Network) können aus mehreren Neuronen-Schichten bestehen, wobei hier Verbindungen in tiefere Schichten möglich sind. Dies bedeutet, dass Ergebnisse von vorherigen Durchläufen des Netzes als Eingang mit einbezogen werden können.

Voll-vernetzte Netze (fully connected) sind neuronale Netze, bei denen alle Neuronen miteinander vernetzt sind. Wobei hierbei keine wirkliche Struktur erkennbar ist und ein sehr großer Rechenaufwand besteht.

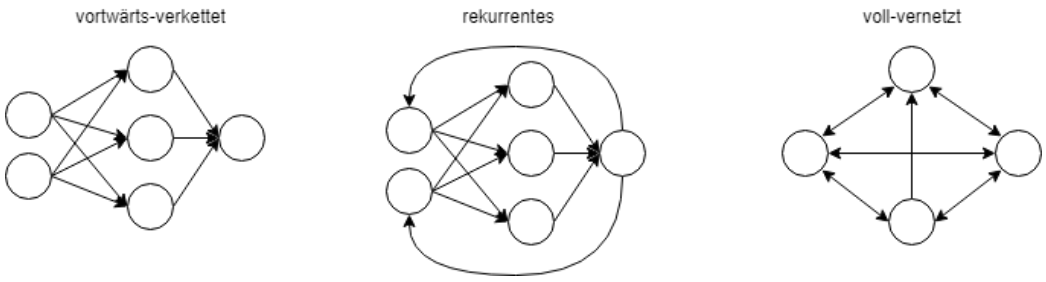

Abbildung 2.3: Netztopologien [6]

Lernen

Bei neuronalen Netzen ist das vorhandene Wissen durch die Kantengewichte repräsentiert. Daraus folgt, dass die verschiedenen Klassifikationen von neuronalen Netzen mit derselben Topologie, nur von den Kantengewichten

abhängt. Dementsprechend lernt ein neuronales Netz durch sequentielles Anpassen der Gewichte. Dazu gibt es verschiedene Verfahren wie diese Gewichte angepasst werden können.

Beim **Backpropagation Lernen** werden in der Trainingsphase die Ergebnisse 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 Ausgabefehlerfunktion zu minimieren. Hierbei wird für jedes Gewicht mithilfe verschiedener Gradientenverfahren eine Änderung berechnet.

Beim **Batch Lernen** werden anders als beim Backpropagation Lernen nicht nach jedem Durchlauf des Netzes die Gewichte angepasst, sondern erst nachdem das Trainingsset komplett durchlaufen wurde. Dies kann zu weniger Genauigkeit führen, da nicht auf spezielle Eigenschaften einzelner Einträge eingegangen wird. Vorteil dieses Verfahren ist, dass sehr viel weniger Rechenaufwand benötigt wird und mehr Trainingsdaten in gleicher Zeit einbezogen werden können.

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 boolschen „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, beziehungsweise mehrere Schichten von Neuronen abgebildet werden.

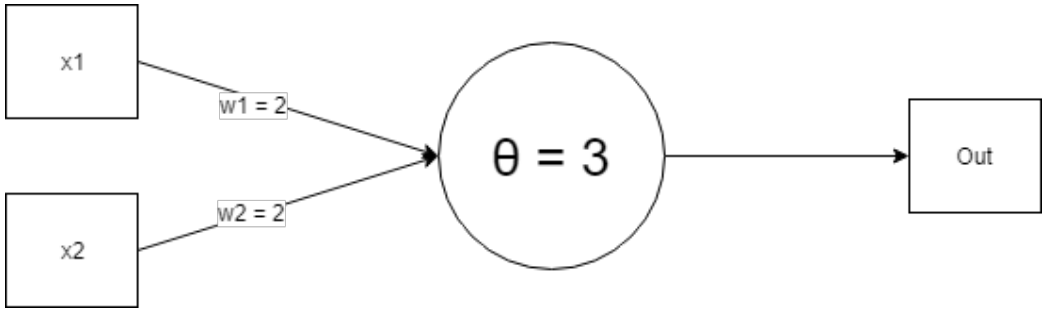

Abbildung 2.4: UND-Operator als Perzeptron [6]

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

CNN - Convolutional Neural Network [7, Vgl. im Folgenden]
Eine weitere Form eines neuronalen Netzes ist ein Convolutional Neural Network. Diese Form findet häufig Anwendung in der Bilderkennung oder Bildklassifizierung.

Die Struktur eines Convolutional Neural Network besteht aus mehreren „Convolutional Layern“ und einem „Pooling Layer“. Bei einem Convolutional Neural Network wird durch verschiedene Faltungsmatrizen eine mehrdimensionale Matrix ständig „verkleinert“ (vgl. Abbildung 2.5). Dieser Prozess beschreibt eine Generalisierung der Eingabematrix, sodass als Ausgabe für jede Klassifizierungs-Klasse eine generalisierte, kleinere Matrix entsteht.
Beim Lernen wird versucht diese Matrix so zu erzeugen, dass essentielle Eigenschaften für die jeweilige Klassifikation abgebildet werden.

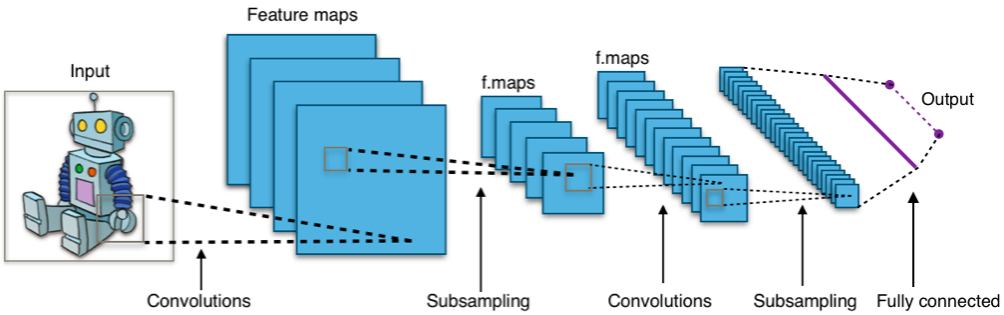

Abbildung 2.5: Typischer Aufbau eines Convolutional Neural Network [1]

2.3 Physikalische Grundlagen

Zur Messung der physikalischen Werte eines Stromnetzes wird ein WeSense-Messgerät[2] verwendet. Dieses Messgerät misst sekündlich nach DIN EN50160 [1] 9 verschiedene Parameter. Diese Parameter beinhalten die Spannung, die Frequenz sowie die ungeraden harmonischen Spannungsoberwellen bis zur 15ten [1, S.2, Kapitel 1.2].

Die DIN EN50160 [5] gibt an wie sich die sieben ungeraden harmonischen Oberwellen aus der Spannung zusammensetzen. Harmonische Oberwellen sind reine sinusförmige Schwingungen, welche das n-fache einer Grundfrequenz sind. Addiert ergeben diese verschiedenen Oberwellen den gemessenen Wert (vgl. Abbildung 2.6).
Die harmonischen Spannungsoberwellen werden als Oberschwingungspegel der Grundfrequenz in Prozent angegeben. Die Grundfrequenz f, mit Geschwindigkeit V und Wellenlänge λ, ergibt sich aus $f = \frac{V}{\lambda}$.

[1] `https://upload.wikimedia.org/wikipedia/commons/6/63/Typical_cnn.png`
[2] http://www.wesense-app.com/home-en/

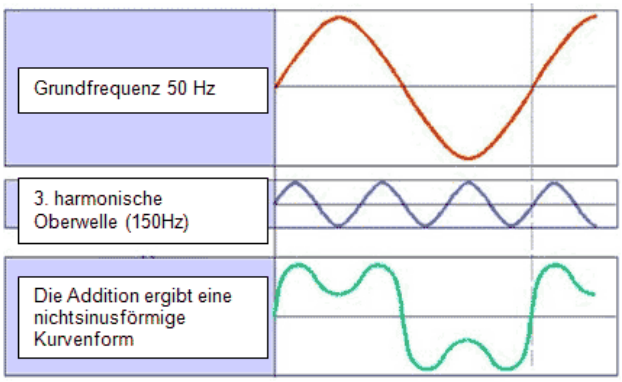

Abbildung 2.6: Beispiel für Addition von Oberwellen [3]

Netzteile, welche in verschiedenen Haushaltsgeräten eingebaut sind, verbrauchen je nach Nutzung mehr oder weniger Strom. Durch diesen Verbraucher sinkt oder steigt die Spannung im Netz. Auch beeinflusst jedes Netzteil die harmonischen Oberwellen auf eine andere Art und Weise.
Durch diese zwei Einwirkungen auf das gesamte Stromnetz eines Haushalts sollte es folglich möglich sein Unterschiede zu erkennen.
Jedoch könnten Komplikationen auftreten, da das Messgerät nur sekündlich Daten misst und möglicherweise Veränderungen der Parameter so minimal sind, das diese in der Zeitspanne nicht erkannt werden.

2.4 Erhebung der Messdaten

Um aussagekräftige Analysen und Klassifikationen über ein Stromnetz, beziehungsweise die Geräte in einem Stromnetz, mit maschinellem Lernen machen zu können, werden viele Trainings- und Testdaten benötigt. Die Daten bestehen aus verschiedenen physikalische Größen, die zu einem bestimmten Zeitpunkt in einem Stromnetz auftreten. Zu diesen Größen gehört die allgemeine Netzspannung, die Netzfrequenz und 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.

Das Messgerät, welches in Kapitel 2.3 beschrieben wird, misst alle benötigten Werte und sendet diese über einen MQTT-Broker[4] an einen Service, welcher

[3] `http://www.energie.ch/harmonische-oberschwingungen-netzqualitaet`
[4] Message Queuing Telemetry Transport

die Daten aufbereitet und in einer MSSQL Datenbank abspeichert. Die Werte werden sekündlich gemessen und in die Datenbank gespeichert oder an registrierte Websocket-Verbindungen gesendet.

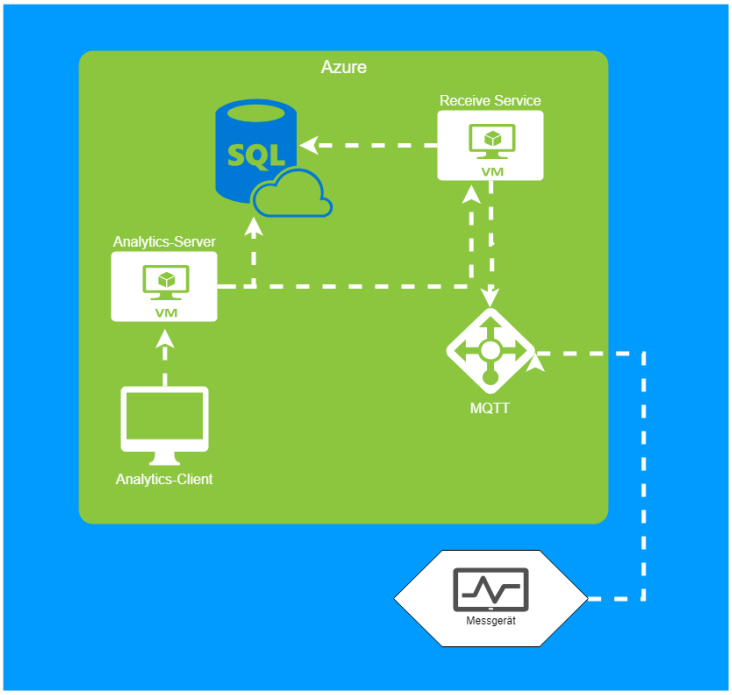

Abbildung 2.7: Architektur zur Abspeicherung der Daten [6], [3]

Klassifikation der Messdaten

Wie in Abschnitt 2.4 beschrieben, sind die rohen Messdaten in einer Datenbank abgespeichert. Zusätzlich werden 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 zugewiesenen Daten werden im weiteren Verlauf gelabelte Daten genannt.

Die gelabelten Daten können manuell oder automatisiert erhoben werden.
Die manuelle Erhebung der Daten erfordert eine Person, welche Zeiten zu denen sicher Geräte aktiv, waren manuell kennzeichnet.
Um Daten automatisiert zu erheben wird ein weiteres Messgerät pro Haushaltsgerät benötigt. Dieses Messgerät müsste zwischen dem zu messenden

Gerät und dem Stromnetz zwischengeschalten werden, um das Gerät zu klassifizieren sobald Strom fließt. Die automatisierte Methode ermöglichst es präzisere Klassifikationen, als auch mehr Zeitspannen und Haushaltsgeräte gleichzeitig zu erfassen, und ist somit die bevorzugte Methode.
Jedoch müsste für die automatisierte Methode ein weiteres Gerät entwickelt werden, weshalb für diese Arbeit diese manuelle Methode gewählt wird.

Für die manuelle Datenerfassung wurde eine progressive Web-Applikation(vgl. Abbildung 2.9, API: [8], GUI: [9]) mit einer einfachen MySQL-Datenbank im Backend erstellt. Mithilfe dieser App können die Daten sehr einfach über ein mobiles Endgerät erfasst und abgespeichert werden.

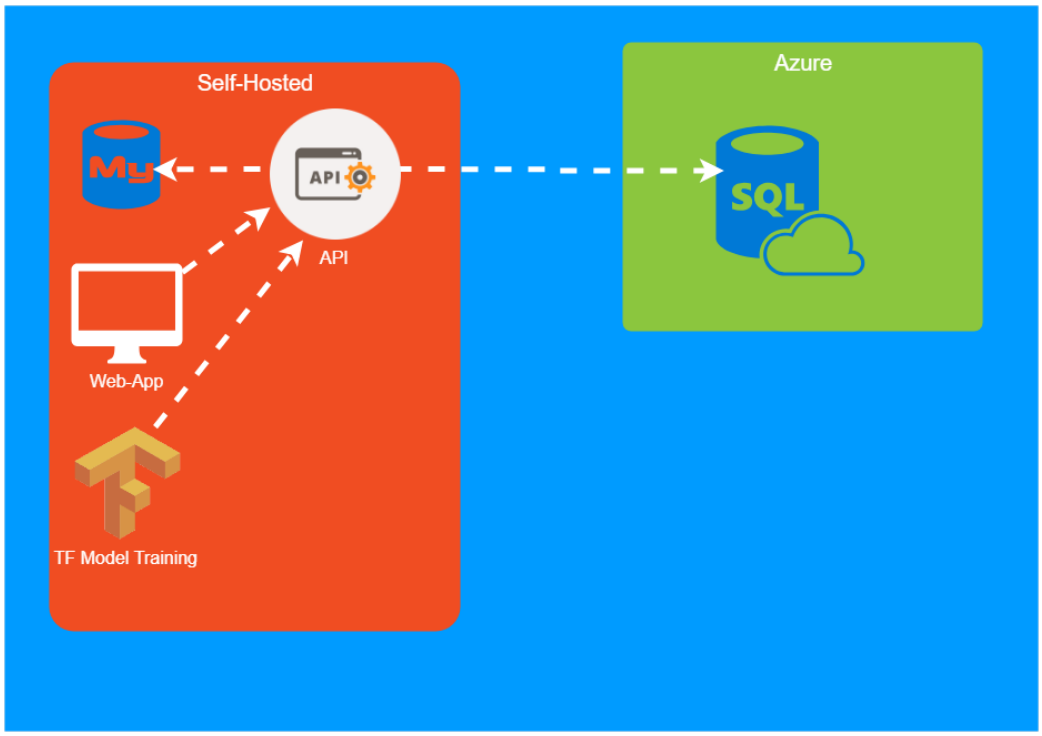

Abbildung 2.8: Architektur zum Training der neuronalen Netze [6], [3]

Abbildung 2.8 zeigt die Einbindung der GUI und des APIServers in die Architektur der Messdatenspeicherung. Die entwickelte Applikation besteht aus einer GUI, welche eine Website ist, und eine Visualisierung für den APIServer darstellt. Der APIServer bildet das Verbindungsstück zwischen den klassifizierten Zeiträumen und den eigentlichen Daten des Messgerätes. Die Website,

welche auf dem PolymerFramework[5] basiert, ist somit nur eine Anzeige für die Funktionen des API-Servers. Sie bietet mehrere Seiten zur manuellen Klassifikation der Geräte, sowie verschiedene Visualisierungen(vgl. Anbschnitt 2.5) zur Analyse der klassifizierten Zeiträume.

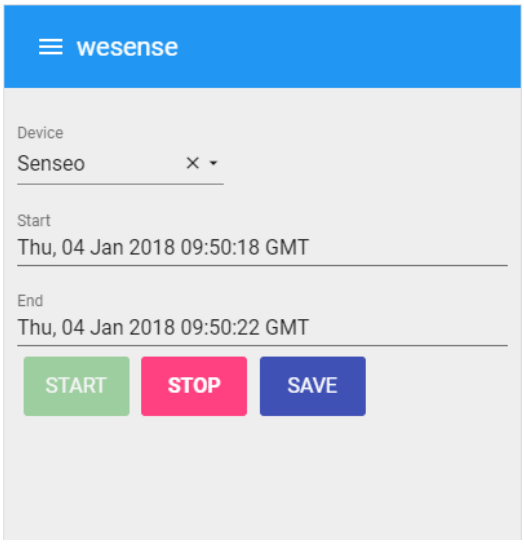

Abbildung 2.9: Screenshot der progressive Web-App

2.5 Visualisierung

Zusätzlich zur manuellen Erhebung der Daten, sowie zur besseren Analyse der Daten, wurden verschiedene Visualisierungsmöglichkeiten implementiert. Einerseits können die verschiedenen physikalischen Größen eines Gerätes zu einem bestimmten Zeitpunkt miteinander verglichen werden. Andererseits können auch bestimmte Größen zu gelabelten Zeiträumen eines Gerätes verglichen und analysiert werden. Durch diese Visualisierung können Gemeinsamkeiten bestmöglich in unterschiedlichen Größen oder Zeiten erkannt werden.

Es werden verschiedene Diagramme sowie Normalisierungen der Daten zur Analyse bereitgestellt. Die Analyse der Daten können in verschiedenen Diagrammen und Normalisierungen dargestellt werden:

- sekündlich in einem Liniendiagramm

[5] `https://www.polymer-project.org/`

- in Datenpunkten (Klassen), welche frei wählbar sind
- Histogramme mit verschiedenen Klassen

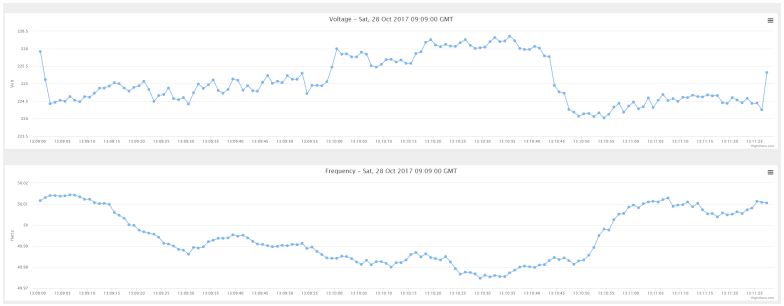

Abbildung 2.10: Screenshot eines gelabelten Zeitraumes aus der Web-App

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 zweites 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

Kaffeemaschine Zur manuellen Analyse der Daten wird das in 2.5 beschriebene Tool verwendet. Zunächst werden aussagekräftige Größen wie die Spannung oder die Frequenz der Kaffeemaschine verwendet und mit anderen aktiven Zeiträumen der Kaffeemaschine verglichen. Hierbei kann ein spezifischer Verlauf der Spannung erkannt werden.
Wie in Abbildung 3.1 zu sehen ist, ist die Kurve zu Beginn stark fallend und verbleibt dann eine gewissen Zeit auf diesem Tief. Nach einem länger steigenden Abschnitt fällt die Kurve wieder, bis der Zubereitungsvorgang beendet wurde und die Spannung wieder steigt.

Diesem Verlauf können nach mehreren Beobachtungen bestimmte Vorgänge einer Kaffeezubereitung zugeordnet werden. Durch die Erwärmung des Brühwassers, beim Einschalten der Kaffeemaschine, kann ein erstes Fallen der Kurve beobachtet werden. Da dabei viel Energie benötigt wird um das Wasser zu

erhitzen, steigt der Stromverbrauch der Kaffeemaschine stark an; somit fällt die Netzspannung stark ab. Die Netzspannung bleibt solange auf einem gewissen Tiefpunkt mit minmaler Netzschwankung bis das Wasser erwärmt wurde und ein weiterer manueller Schritt zum Fortfahren des Prozesses notwendig ist.

Nach manueller Wahl der Tassengröße wird 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 wieder ab, bis der komplette Vorgang beendet ist. Somit kann die zweite Tiefpunktphase dem „Pressvorgang“ der Kaffeemaschine zugeordnet werden.

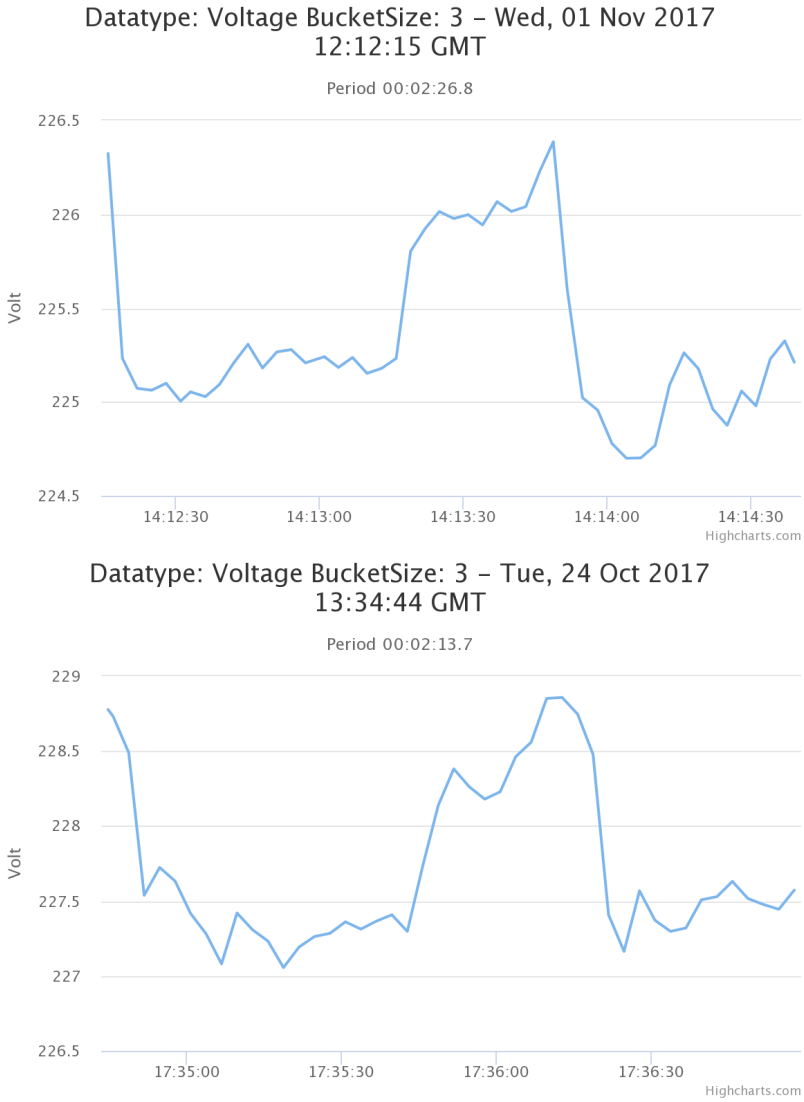

Abbildung 3.1: Spannungsverlauf der Senseo Kaffeemaschine mit einer Klasse von je 3 Datenpunkten zu 2 verschiedenen Zeitpunkten

Mikrowelle Bei der manuellen Analyse der Mikrowelle können bei dem Spannungs- oder Frequenzverlauf 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.

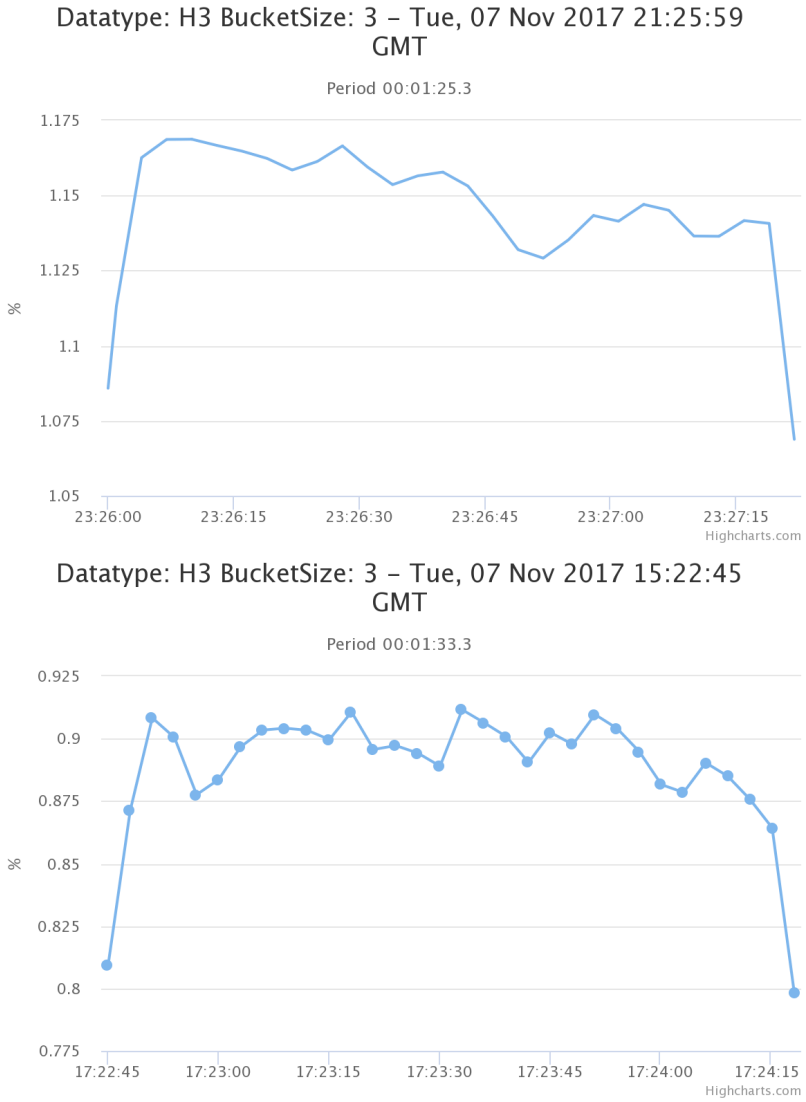

Abbildung 3.2: Verlauf der 3. harmonischen Oberwelle einer Mikrowelle mit einer Klasse von je 3 Datenpunkten zu 2 verschiedenen Zeitpunkten

Wie dieser Abschnitt zeigt, können schon mit bloßem 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. Damit die verschiedenen Formen der Transformation durchgeführt werden können, müssen bestimmte Voraussetzungen beachtet werden. Die Implementierung zu diesem Kapitel lässt sich in [1] finden.

Bei konventionellen neuronalen Netzen werden Objekten Eigenschaften zugewiesen, wobei alle Objekte mit denselben Eigenschaften beschrieben werden. Ein Beispiel dafür sind Tiere, bei denen anhand von Reproduktion oder Atmung, in Säugetiere, Vögel, Reptilien, etc. eingeteilt werden. So können speziellen Tieren, wie einem Hund, eine dieser Klassen zugewiesen werden. Allen klassifizierten Tieren 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 sekündlich gemessenen Werte des Stromnetzes als 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 willkürlich Daten aus der Datenbank abgerufen werden bei denen mit großer Wahrscheinlichkeit kein zu klassifizierendes Gerät aktiv war. Als einer dieser Zeiträume könnte zum Beispiel von 00:00 bis 5:00 Uhr gewählt werden, da zu dieser Zeit mit hoher Wahrscheinlichkeit kein Gerät aktiv war.

Somit ergibt sich eine dreidimensionale Matrix, welche in der ersten Dimension die Zeiträume abbildet, in der zweiten Dimension die sekündliche Zeitreihe und in der dritten die vorhandenen physikalischen Größen.

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

[1] `https://github.com/schrodit/wesense-ml/blob/master/data/data_manager.py`

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:                                   ▷ A - float[][][], b - int (Batchgröße)
3:      B = Array[][][]
4:      for i = 0 to length(A) do
5:          for j = 0 to length(A[i]) − b do
6:              start = j
7:              end = j + b
8:              I_b = 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 die Geräte maschinell zu klassifizieren, werden drei verschiedene „neuronale Netzmodelle“ 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. [2]).

Der „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 Tensor (Matrix innerhalb eines neuronalen Netzes) als Eingabe, 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. Die „Hidden Layer“ werden größer je näher sie topologisch an dem „Output Layer“ liegen.

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

Der „Output Layer“ wandelt nun die Erkenntnisse der vorherigen Schichten in ein lesbares Format um. Hierbei 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 der zu klassifizierenden Geräte.

[2] `https://github.com/schrodit/wesense-ml/blob/master/classification/cnn_classification.py`

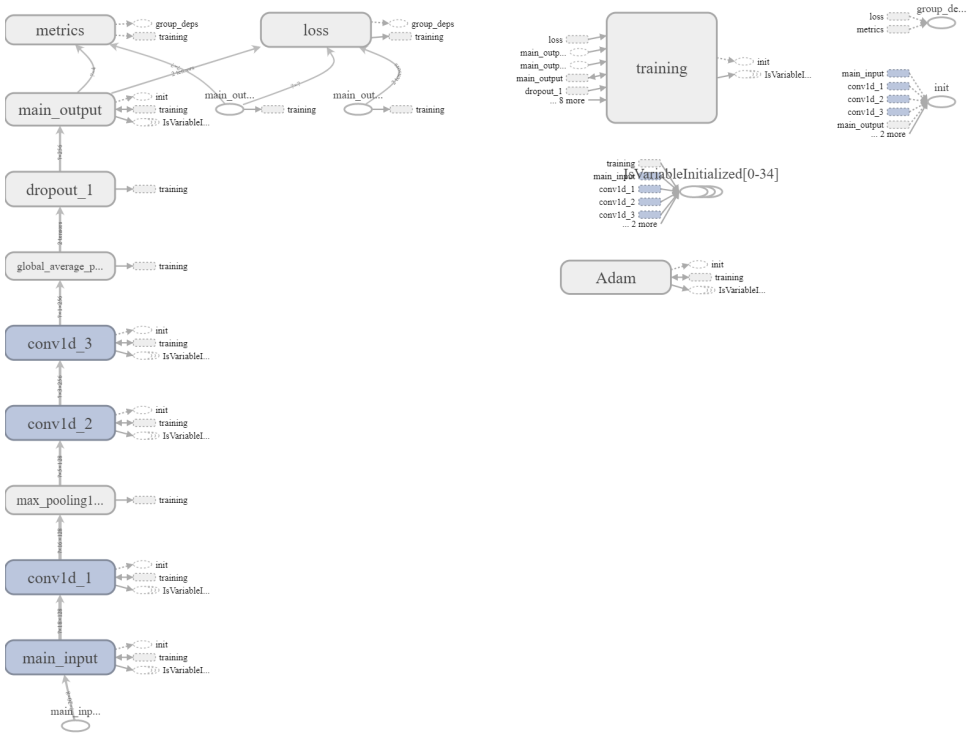

Abbildung 3.3: Convolutional Neural Network [2]

RNN

Das „Recurrent Neural Network(RNN) “- Model besteht aus einem „Input Layer“, zwei darauffolgenden „LSTM(Long short-term memory) Layer“ und einem „Ouput Layer“ (vgl. [3]).
Der „Input Layer“ ist ein weiterer „LSTM Layer“, welcher denselben Eingabetensor annimmt wie das CNN-Model.
Es folgen zwei weitere „LSTM Layer“ einer „Hidden Layer“-Größe von 64 Neuronen pro Schicht. Dieselbe Größe wird auch im Eingabe-„LSTM Layer“ verwendet.

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

[3] `https://github.com/schrodit/wesense-ml/blob/master/classification/rnn_classification_keras.py`

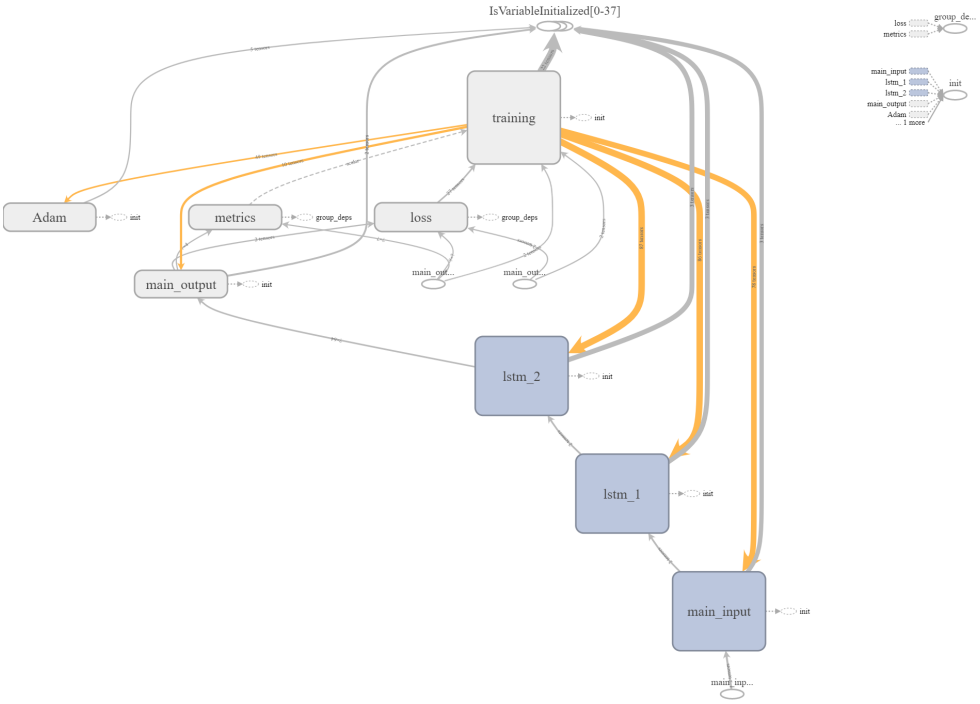

Abbildung 3.4: Recurrent Neural Network [2]

Mixed - CNN und RNN

Das dritte zu testende Model beinhaltet eine Mischung aus den zwei bisherigen Netzen. Dies soll zeigen, ob verschiedene Eigenschaften, von neuronalen Netzen, positive oder negative Auswirkungen auf das Training und die Genauigkeit des Models haben.

Da dieses Model eine Mischform aus Convolutional Neural Network und Recurrent Neural Network ist, besteht dieses Model aus zwei „Convolutional Layers“, welche die Eingabe und eine Vorprozessierung zum Generalisieren der Eingabe darstellt (vgl. [4]). Auf diese beiden Schichten folgen zwei rekurrente „LSTM Layer“ und dasselbe „Output Layer“ wie bei beiden anderen Modellen. Die Recurrent Neural Network-Schichten sollen das Vorprozessierte verarbeiten um mögliche Muster in der Zeitserie zu erkennen.

[4] `https://github.com/schrodit/wesense-ml/blob/master/classification/mix_classification.py`

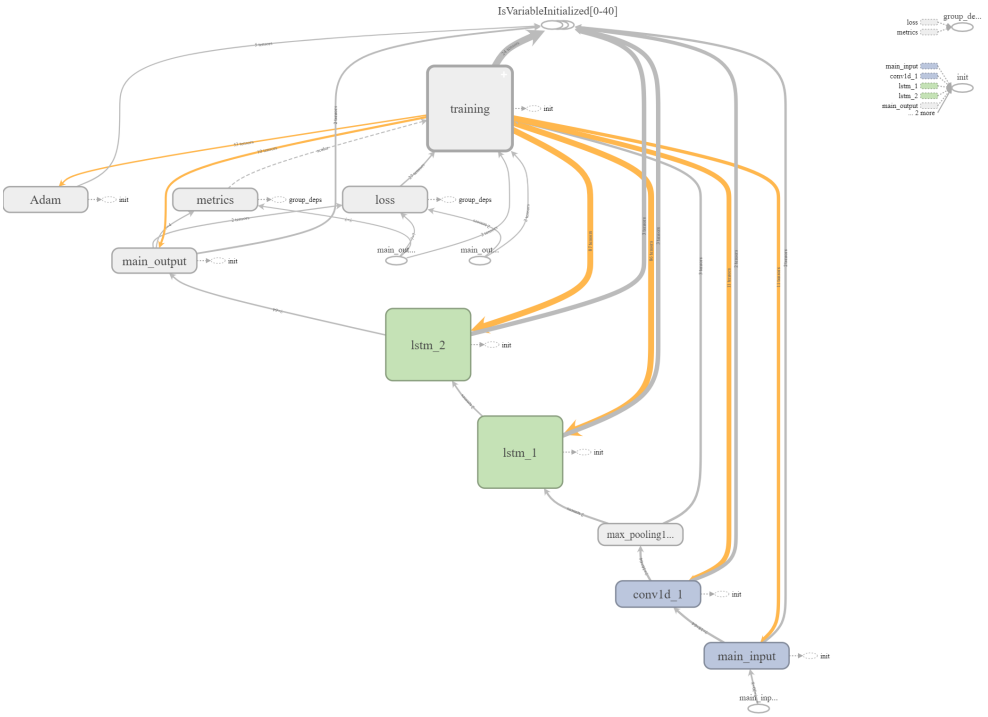

Abbildung 3.5: Convolutional Neural Network mit Recurrent Neural Network [2]

Die in diesem Abschnitt dargestellten Modelle stellen den grundlegenden Aufbau der neuronalen Netze dar. Diese beinhalten jedoch keinerlei Wissen und können somit noch nicht zum Klassifizieren verwendet werden.

3.4 Trainingsprozess

Um die drei verschiedene Modelle anwenden zu können, muss Diesen Wissen beigebracht werden. Dies geschieht durch Trainieren mit den in Abschnitt 3.2 vorbereiteten Daten(vgl. Abschnitt 2.4). Damit die Modelle bestmögliche Ergebnisse erzielen, wird der Trainingsprozess mit verschiedenen Trainingsparametern, sowie verschiedenen Daten durchlaufen. Die Trainingsparameter beinhalten die Batchgöße, die Lernrate, die Trainingsepochen, die Anzahl der zusätzlich gewählten Zeiträume, sowie die genutzten physikalischen Größen.

Die Trainingsparameter aller Netze werden zunächst mit Standardparametern versehen.

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 um verschiedene Werte dieses Parameters miteinander zu vergleichen.
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 Verfahren gewählt, mit denen die Rohdaten transformiert werden.
Es 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 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 Klassifi-

zierungen vorhandenen(vgl. Algorithmus 3.2). Somit muss diese Klassifikation ausgewertet werden, um genau sagen zu können, zu welchen Datenpunkten (Zeitpunkten) welche Geräte am wahrscheinlichsten aktiv waren.

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 die Gewinnerklasse bestimmt. Somit erhält man zwei verschiedene Gewinnerklassen, welche miteinander verglichen werden um die endgültige Gewinnerklasse zu bestimmen.
Der Vorteil dieses Verfahrens ist die Berücksichtigung aller Klassifikationen sowie deren Einzelwahrscheinlichkeit, sodass sich eine genauere Klassifizierung ergibt, wie Abbildung 4.2 zeigt.

Algorithm 2 Ergbnisauswertung

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

Kapitel 4

Ergebnis

Wie die nachfolgendenen 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 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 sehr früh „overfittet“ (vgl. Tabelle 4.2 und Abbildung 7.2).

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 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 (mehreren Trainingsdurchläufen mit allen Trainingsdaten sowie Backpropagation-Algorithmus) werden alle Modelle besser trainiert und erzielen bessere Ergebnisse. Jedoch tritt bei Modellen mit einem Recurrent Neural Network bei zu großen Trainingsdurchläufen „overfitting“ auf (siehe oben). Dies bedeutet, dass bei Recurrent Neural Net-

work-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, dass „overfitting“ schneller auftritt oder bestimmte Features durch Zufall zu stark in das Ergebnis einfließen. Hier sollte eine kleine Lernrate mit höherer Epochenzahl für bessere Ergebnisse gewählt werden.

Zusammenfassend kann somit festgestellt werden, dass die besten Ergebnisse erzielt werden, wenn die Lernrate niedrig ist, viele Epochen trainiert werden 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.

Die kompletten Ergebnisse sind in Kapitel 7 zu finden.

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	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 ⇒ „Senseo Kaffeemaschine“,
y: 1 ⇒ „Mikrowelle“,
y: 2 ⇒ „Bosch Vollautomat“, und
y: 3 ⇒ „Kein aktives bekanntes Gerät“.

Die Abbildungen zeigen unterschiedliche Bewertungen, beziehungsweise Klassifizierungen der Zeitpunkte. Dabei können die Gemeinsamkeiten aller drei Klassifizierungen gut erkannt werden.
Trotz der Einbeziehung vieler verschiedener Ergebnisse (vgl. Algorithmus 3.5), gibt es noch einige Ausreißer. Diese Ausreißer sind bei allen neuronalen Netzen unterschiedlich verteilt und auch mit unterschiedlicher Länge. Je-

doch kann man sehr gut sehen, dass alle drei neuronalen Netze einen großen gemeinsamen Abschnitt haben (ca. x = [30, 190]), wobei die „Senseo Kaffeemaschine“ erkannt wurde. Somit kann mit hoher Wahrscheinlichkeit davon ausgegangen werden, dass dieses Gerät zu diesem Zeitpunkt aktiv war.

Die anderen Geräte wurden nie erkannt, was darauf zurückzuführen ist, dass für diese Geräte wenige Daten vorhanden sind.
Wie die Klassifikation der „Senseo Kaffeemaschine“ zeigt, können mit genügend Daten diese Geräte 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.

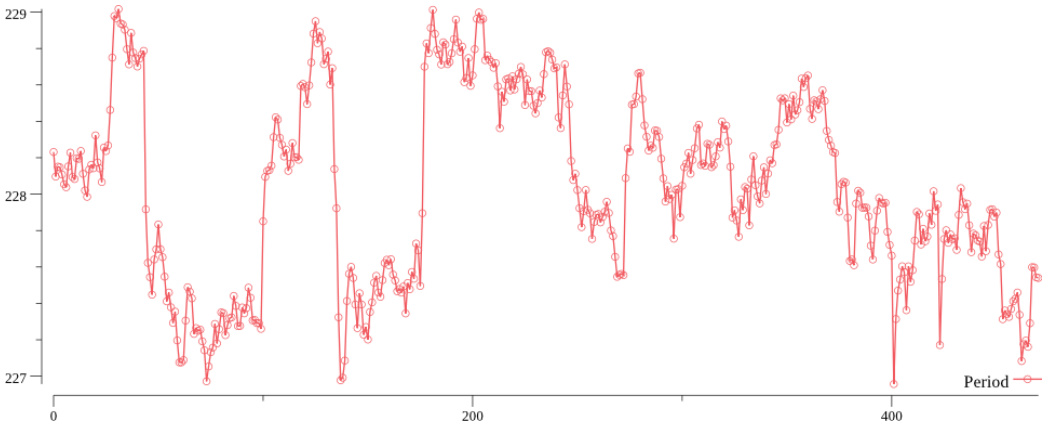

Abbildung 4.1: Spannungsverlauf in Volt von „24.10.2017 15:29:44“ bis „24.10.2017 15:41:57“

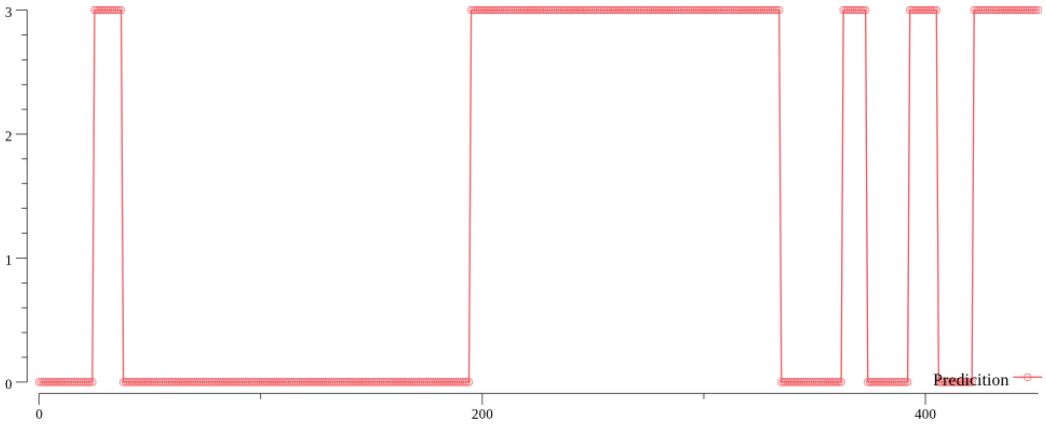

Abbildung 4.2: Klassifizierung mit CNN

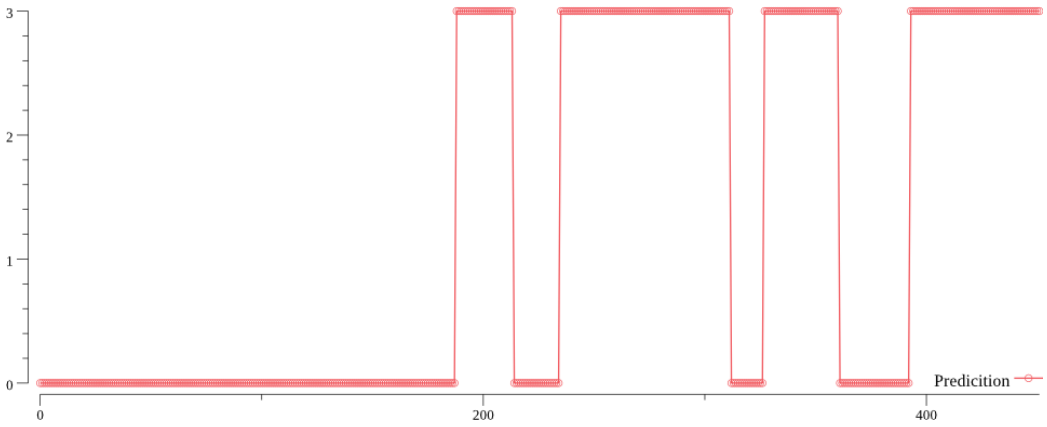

Abbildung 4.3: Klassifizierung mit RNN

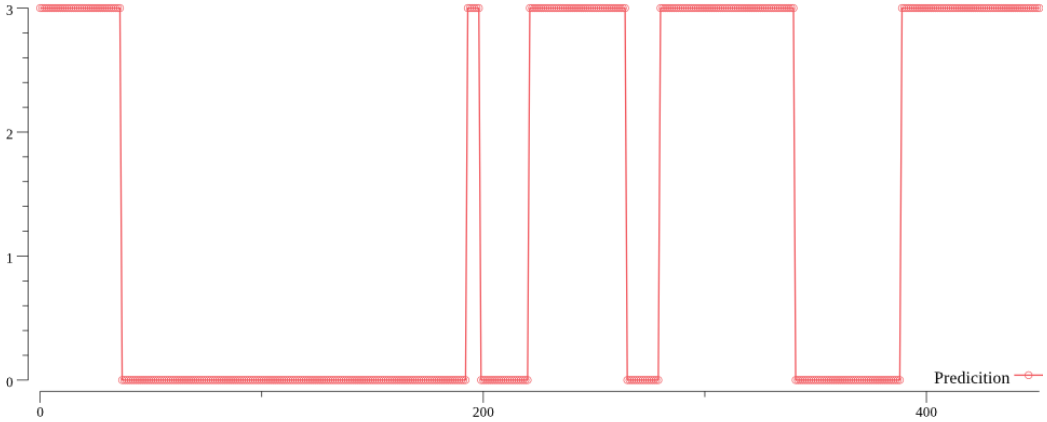

Abbildung 4.4: Klassifizierung mit Mixed RNN und CNN

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 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.

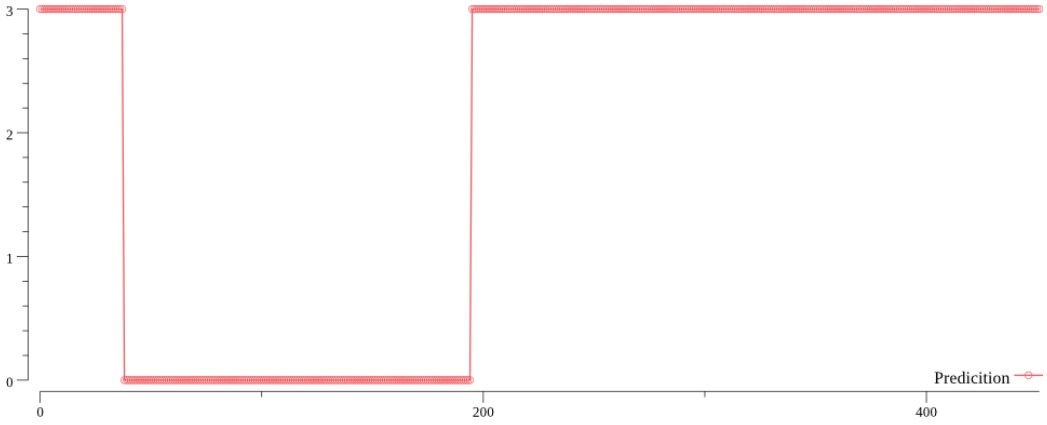

Abbildung 4.5: Geglättetes Ergebnis des CNN

4.0.3 Zusammenfassung

Zusammenfassend ist das Ergebnis eine Applikation mit einem neuronalen Netz als Kern. Dieses neuronale Netz kann mithilfe verschiedener Algorithmen die aktiven Laufzeiten von Geräten bestimmen. 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 mehr klassifizierte Daten zu erheben.

Kapitel 5

Produktivbetrieb

Das aus Kapitel 4 resultierende neuronale Netz sowie die Ergebnisauswertung dessen 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 Vorraussetzungen 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 stabiles und sicheres Programm zu entwickeln sollen wichtige fehleranfällige Funktionalitäten getestet werden.

5.0.1 Implementierung

Aufgrund besserer Performance und Einheitlichkeit mit der Daten-API (vgl. Analytics-Server in Abbildung 5.1) wird eine Socket.IO-API anstatt einer gebräuchlicheren REST-API gewählt. Die Spezifikation dieser Schnittstelle wird, wie in Abschnitt 7 beschrieben, festgelegt. Somit ist die Echtzeitanalyse sowie die Periodenanalyse mit möglichst wenig Mehraufwand abgebildet.

Durch das Verwenden von Tensorflow und dessen Unterstützungseinschränkungen[1]

[1] https://www.tensorflow.org/install/

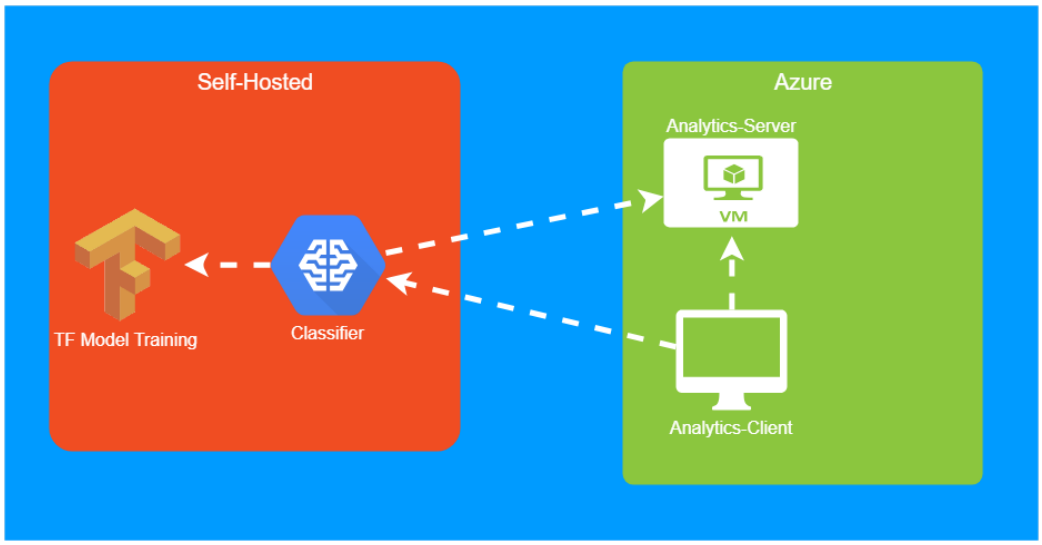

Abbildung 5.1: Architektur zum Produktivbetrieb [6], [3]

steht eine begrenzte Anzahl an Ansätzen 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 Implementierungen dort getestet [2]. Jedoch ist Python schlecht geeignet für viele Anfragen mit großem Rechenaufwand, die für die Vorbereitung der Daten erforderlich ist. Auch konnte das Auslagern der aufwendigen Rechenoperationen keine bessereren Ergebnisse liefern. Somit wird entschieden den Produktivbetrieb 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 wird dadurch erheblich verbessert. Die Implementierung (siehe [3]) besteht aus drei essentiellen Bausteinen, welche in Module (Packages) aufgeteilt sind. Diese drei Grundbausteine bestehen aus der Datentransformation, der Klassifikation sowie dem Socket.IO-Server.

[2] https://github.com/schrodit/wesense-ml/tree/master/prod
[3] https://github.com/schrodit/wesense-ml/tree/master/go

Da im Datentransformations-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 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 einen Docker-Container (siehe [4]) in einer isolierten, gleichartigen Umgebung. Somit kann eine gleichbleibend 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 eines 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 Docker-Umgebung benötigt und kann sehr einfach auf weitere Server portiert werden.

[4] https://github.com/schrodit/wesense-ml/blob/master/.drone.yml

Kapitel 6

Wirtschaftlichkeit

Das Ergebnis und die Erkenntnis aus dieser Arbeit kann in verschiedenen Industriebranchen, sowie auch im privaten Gebrauch eingesetzt werden.

Hersteller von Haushaltsgeräten Falls Hersteller von diversen Haushaltsgeräten auf anonymisierte Strommessdaten aus privaten Haushalten zugreifen können, wäre es möglich Laufzeiten ihrer Geräten zu analysieren. Somit kann die Aktivität, beziehungsweise die Benutzung dieser Geräte in Gebieten bestimmt werden.
Durch präzisere Verbraucherinformationen kann der Customer Relationship Management-Prozess eines Unternehmens optimiert werden. Durch präzisere Messdaten können Fehlfunktionen klassifiziert werden und so „Predictive Maintenance“ für die Geräte angeboten werden.

Privater Gebrauch Für den privaten Gebrauch kann die Aktivität der Geräte dadurch von Vorteil sein, dass intelligent Strom gespart werden kann. Auch kann eine Art digitaler Marktplatz eingerichtet werden, indem private Kunden Geräte klassifizieren und diese Daten an Unternehmen verkaufen können.

Energieindustrie In der Energieindustrie können mithilfe dieser Technik einfache Haushaltsgeräte erkannt werden. Mit derselben Technik können auch Störungen sowie spezielle Merkmale eines Stromnetzes erkannt werden. Dadurch kann analysiert werden wann und warum bestimmte Störungen auftreten, wodurch Energiezulieferer besser und schneller auf diese reagieren können.

Wie gezeigt wurde, kann das Ergebnis dieser Arbeit vielfältig in verschiedenen Wirtschaftzweigen eingesetzt werden. Um auf dem Markt erfolgreich

zu sein, muss jedoch das Ergebnis dieser Arbeit mit mehr Daten trainiert und verbessert werden. Da dies jedoch für die Industrieunternehmen kein Problem darstellen sollte, kann diese Arbeit ein Denkanstoß sein.

Kapitel 7

Anlagen

Architektur

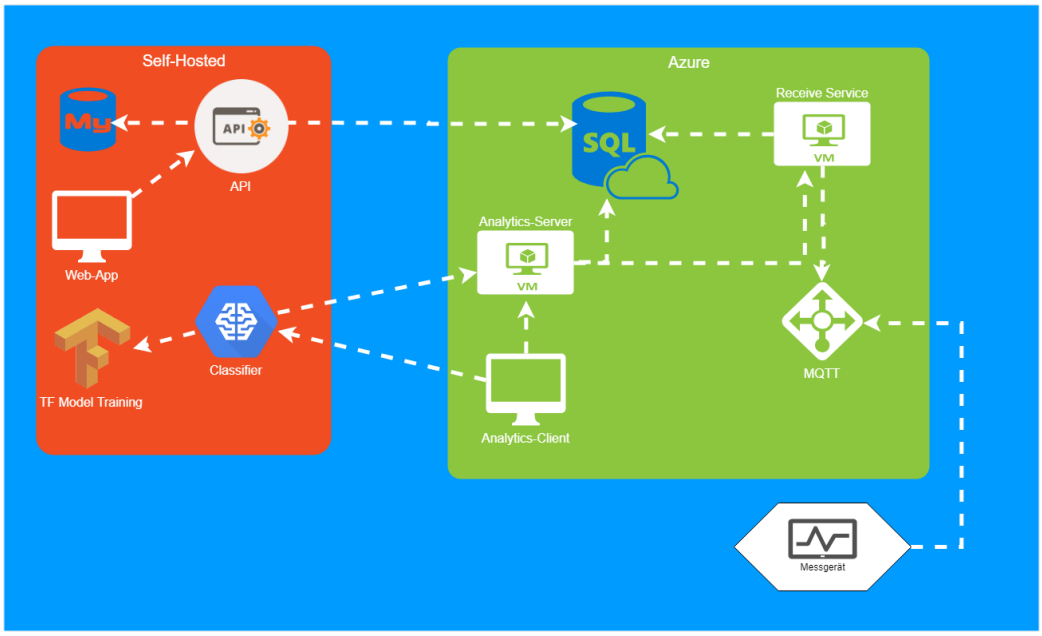

Abbildung 7.1: Komplette Architektur [6], [3]

Ergebnisse

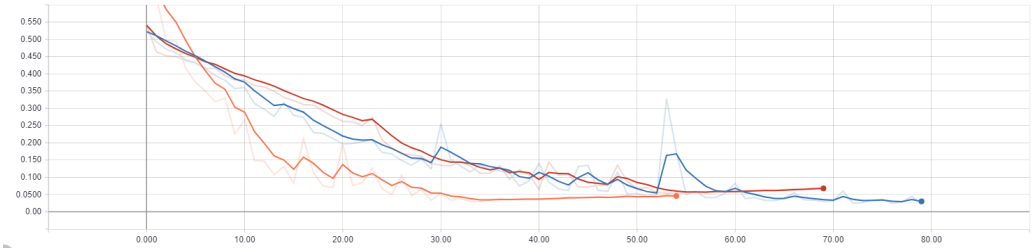

Abbildung 7.2: Validation Loss des Trainigs von CNN (blau), RNN (rot) und MIXED(orange) [3]

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 7.1: Ergebnis: komplette Epochen

API-Spezifikation

Single Batch

Input:

```
Topic: "single-batch"
{
    "data": [
        {
            "u": float,
            "f": float,
            "h3": float,
            "h5": float,
            "h7": float,
            "h9": float,
            "h11": float,
            "h13": float,
            "h15": float,
        },
        .. x Batchsize
    ]
}
```

Output:

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

Period

Input:

```
Topic: "period"
{
    "data": {
        "start": "DateTime2",
        "end": "DateTime2"
    }
}
```

Output:

```
Topic: 'period-prediction'
* Series of Predictions where 0: Senseo, 1:
   Microwave, 2: Bosch, 3: Undefined, for every
   second
```

```
{
    "data": [
        Int,
        ... end - start
        Int
    ]
}
```

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Literaturverzeichnis

[1] A. Eberle GmbH Co. KG, https://www.a-eberle.de/de/produktgruppen/pq-mobil/komponenten/wesense. *User Manual: WeSense for Android*, 2017.

[2] Google. Tensorboard. `https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard`, 2018.

[3] Google. Tensorflow. `https://www.tensorflow.org/`, 2018.

[4] 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.

[5] Wroclaw University of Technology Henryk Markiewicz Antoni Klajn. Power quality application guide voltage disturbances, standard en 50160, voltage characteristics in public distribution systems. Technical report, Copper Development Association, Juli 2004.

[6] JGraph. Draw.io. `https://www.draw.io/`, 2018.

[7] Dr. Shashank Pathak. mlp_engg, 2018. DHBW Karlsruhe - Robotics2.

[8] Tim Schrodi. wesense-labeling-api. `https://github.com/schrodit/wesense-labeling-api`, 2017.

[9] Tim Schrodi. wesense-labeling-gui. `https://github.com/schrodit/wesense-labeling-gui`, 2017.

[10] Felix Riese Sina Keller. Vl 5 — neuronale netze, 2017. DHBW Karlsruhe - Wissenbasierte Systeme.

[11] Florian Wenzel. Einführung in neuronale netze. *Neurorobotik, Institut fur Informatik, Humboldt-Universität zu Berlin*, 2012.

[12] Wikipedia. Cnn. `https://upload.wikimedia.org/wikipedia/commons/6/63/Typical_cnn.png`, 2018.