Kollaborative Sensorsteuerung mit TensorFlow Lite

Ilja Grauer

Hochschule für Angewandte Wissenschaften
Berliner Tor 7, 20099 Hamburg, Deutschland
ilja.grauer@haw-hamburg.de

Zusammenfassung. Machine Learning mit TensorFlow ist ein hochaktuelles Forschungsgebiet. Gerade im Zeitalter von IoT[1] und Smartwatches ist es naheliegend, dass auch mobile und embedded Geräte hohes Interesse gewinnen. TensorFlow Lite kann normale TensorFlow Modelle auf dieser Art von Geräten betreiben, indem es äußerst Performance- und Speichereffizient ist. Sensoren wie Beschleunigungsmesser und Gyroskop ermöglichen hierbei nützliche Software, mit der beispielsweise Gestensteuerung realisiert werden kann. In dieser Arbeit soll mit Hilfe dieser Sensoren eine solche Gestensteuerung am Beispiel eines Schlagzeugs implementiert werden, ohne das entsprechende Musikinstrument zu besitzen. Hierbei sollen mit TensorFlow Lite und den entsprechenden Sensoren auf einem mobilen oder embedded Gerät die Schlagbewegungen eines Drumsticks zunächst aufgezeichnet werden. Die ermittelten Daten sollen anschließend über einen Clustering-Algorithmus kategorisiert werden und manuell mit einem Label (Hi-Hat, Snare,...) gekennzeichnet werden. Sobald man dadurch ausreichende Trainingsdaten generiert hat, soll damit ein geeignetes RNN[2] trainiert werden. Das trainierte Modell soll dann auf zwei mobilen Geräten laufen (Smartwatch oder verkabeltes Raspberry Pi). Die Bewegungen des Nutzers sollen dann erkannt werden und der passende Ton des Schlagzeugs in angemessener und benutzerfreundlicher Latenz ausgespielt werden. Forschungsziel der Arbeit ist es zwei Kernfragen zu beantworten:

1. Wie können zwei neuronale Netze zweier Geräte (die Drumsticks) miteinander kollaborativ agieren?
2. Wie erreicht man durch Predictions eine gute Latenz bei dem konkreten Anwendungsfall einer Schlagzeug-App?

Schlüsselwörter: machine learning, tensorflow, tensorflow mobile, sensor data, classification, prediction, drum, app, android, raspberry pi, recurrent neural network, long short term memory, automatic drum transcription

[1] Internet Of Things
[2] Recurrent Neural Network

1 Einleitung

Machine Learning mit TensorFlow ist ein hochaktuelles Forschungsgebiet. Gerade auf mobilen und embedded Geräten ist es ein stark wachsendes Thema der Künstlichen Intelligenz, insbesondere da IoT und Smartwatches in der Gesellschaft immer verbreiteter werden. Gerade hierbei wird deutlich, wie diverse Anwendungsgebiete der Technologie einer großen Zielgruppe aus Menschen den Alltag erleichtern. Seien es Spracherkennungssoftware wie Siri, Alexa und Google Now, Sicherheitskomponenten wie Fingerabdrucksensor oder Gesichtserkennung oder die personalisierte Fitness-App. Darunter sind viele Anwendungen, die insbesondere auf die Bewegungssensoren wie Beschleunigungsmesser und Gyroskop zugreifen und dadurch einzigartige Softwarelösungen wie Gestensteuerung bieten.

Aus diesem Grund ist Machine Learning für viele Software-Entwickler interessant, da sie mit Hilfe der aktuellen Forschung beeindruckende Anwendungen implementieren können. Diesen Gedanken habe ich aufgenommen und mich entschlossen mit Hilfe von Sensoren und Machine Learning eine solche App zu implementieren. Konkret will ich eine Schlagzeug-App programmieren, mit der man ohne das entsprechende Musikinstrument qualitativ gute Ergebnisse erzielen kann. Es gibt bereits einige Hersteller wie Freedrum [1], die bereits ein ähnliches Konzept auf dem Markt haben. Da es aber nicht sicher ist, ob der Hersteller Machine Learning verwendet hat, um sein Produkt zu entwickeln, will ich es mit Machine Learning versuchen.

Dabei will ich mit variablen Hardware-Komponenten und Methoden die beste Lösung finden, um eine solche App zu entwickeln. Die Hardware-Komponenten können dabei Smartwatches oder Raspberry Pis mit oder ohne Verkabelung sein, während die Methoden zum einen unterschiedliche Wege beinhalten die Trainingsdaten zu erzeugen sowie unterschiedliche KNNs[3] zu nutzen. Ziel ist es am Ende eine Evaluierung der verwendeten Komponenten und Methoden für eine solche Produktentwicklung bereitzustellen.

Nebenher sollen zwei wesentliche Kernfragen beantwortet werden, die maßgeblich Einfluss auf die Qualität der App nehmen.

1. Wie können zwei neuronale Netze zweier Geräte (die Drumsticks) miteinander kollaborativ agieren? Konkret bedeutet das inwiefern 2 getrennte RNNs miteinander vernetzt werden können, um komplexere Gestensteuerung zu ermöglichen.
2. Wie erreicht man durch Predictions eine gute Latenz bei dem konkreten Anwendungsfall einer Schlagzeug-App? Da davon auszugehen ist, dass es zu Verzögerungen in der Tonausgabe aufgrund von Bluetooth kommen kann, müssen Lösungen gefunden werden wie diese Latenzprobleme zu minimieren sind. Hierbei ist die Idee durch Vorhersagen bereits den folgenden Ton frühzeitig auszuspielen.

[3] Künstliches Neuronales Netz

2 Problemstellungen

Die zuvor genannten Kernfragen erzeugen am Anwendungsbeispiel der Schlagzeug-App auch entsprechende konkrete Problemstellungen. Diese zu lösen ist ein kritischer Faktor, um eine zufriedenstellende App zu implementieren.

2.1 Winkelerkennung

Der Winkel zueinander, in der die Drumsticks gespielt werden ist entscheidend, da die Schläge - wenn sie in der selben Höhe liegen - nicht oder nur sehr schwer von ähnlichen unterschieden werden können. Daher muss wie in Abbildung 1 zu sehen, der Winkel zwischen den Sticks als zusätzliches Trainings-Feature berücksichtigt werden. Die Himmelsrichtung eines Drumsticks muss daher jederzeit die Himmelsrichtung des anderen kennen, um den korrekten Winkel zueinander berechnen zu können. Nur so kann eine genaue Positionsbestimmung stattfinden.

2.2 Prediction

Die bereits erwähnte Problematik der Performance tritt ein, wenn ein Ton erst verzögert gespielt wird, nachdem der Stick auf das Objekt geschlagen hat. Die Ursache dafür können eine (schlechte) Bluetooth-Verbindung oder ein langsames Neuronales Netz sein. Da diese technischen Probleme schwer mit einfachen Mitteln zu lösen sind, ist die Idee über die Software eine Vorhersage zu treffen, wann ein Schlag auftrifft und den entsprechenden Ton schon vorher zu spielen. Siehe Abbildung 2.

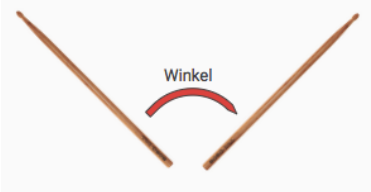

Abb. 1: Himmelsrichtung der Drumsticks soll durch Magnetometer bestimmt werden

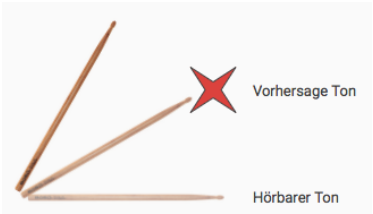

Abb. 2: Vorhersage einer Bewegung soll den Ton früher abspielen, um Latenz zu verbessern.

3 Grundlagen

3.1 Sensoren

Sensoren sind in Geräten wie Smartphone oder Smartwatch bereits integriert und können - falls vom Hersteller bereitgestellt - über eine SDK [4] angesprochen werden können, um die Rohdaten auszuwerten und zu verarbeiten.

Beschleunigungsmesser: Der Beschleunigungsmesser misst die Beschleunigung einer Bewegung des Geräts im dreidimensionalen Raum in einer eingestellten Frequenz. Diese Daten werden in der Regel in Verbindung mit dem Gyroskop genutzt.

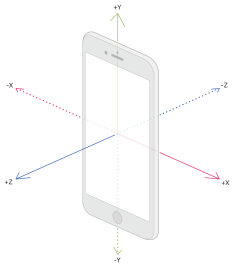

Abb. 3: Beschleunigungsmesser [2]

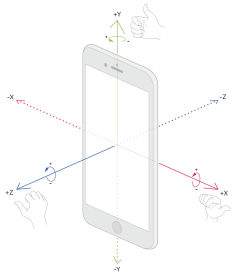

Abb. 4: Gyroskop [2]

Gyroskop: Das Gyroskop misst die Positionsveränderung im Raum und kann dadurch die aktuelle Lage des Geräts genau bestimmen. In Kombination mit dem Beschleunigungsmesser lassen sich so sehr genaue Aussagen über die Bewegung eines Gerätes machen.

Magnetometer: Das Magnetometer ist im Prinzip ein Kompass mit dem man die magnetische Feldstärke messen kann. Dieser Sensor ist dafür gedacht die Himmelsrichtung des Geräts zu bestimmen, in die es zeigt. Das ist dahingehend von Bedeutung, da der Beschleunigungsmesser und das Gyroskop zusammen zwar sehr genau die Bewegung messen können, jedoch die Komponente der Himmelsrichtung fehlt. Dieser Sensor hilft dabei zu bestimmen, ob eine Bewegung in z.B. Richtung Norden oder Osten ausgeführt wird. Problematisch können hierbei störende Magnetfelder sein, die ein auf das Grad genaues Messen der Richtung erschweren. Ebenso ist die Frage, wie exakt diese Sensoren sind.

Mikrofon: Das Mikrofon dient primär zur Kommunikation zweier oder mehr Personen. In dieser Arbeit soll das Mikrofon gegebenenfalls dabei helfen die Schläge zu klassifizieren.

[4] **Software Development Kit:** Ermöglicht das Erstellen von Apps zu einer spezifischen Software

3.2 TensforFlow Lite

TensorFlow Lite ist ein von Google [3] entwickeltes Subset von TensorFlow [4], was darauf ausgelegt ist Machine Learning Modelle auf effiziente Art zu speichern und in mobile Systeme einzubetten. Dabei wird das auf einem leistungsfähigen System erzeugte Modell - bzw. der Graph - zunächst eingefroren, um das Verändern der Knotengewichte des KNN zu unterbinden. Dadurch kann man es nicht mehr trainieren, aber immer noch anwenden. Im nächsten Schritt wird der eingefrorene Graph in eine `.tflite`-Datei konvertiert. Diese benutzt im Gegensatz zu den ProtocolBuffers [5] sogenannte FlatBuffers [6], die schneller und ressourceneffizienter sind. Die generierte Datei kann dann im Zielsystem abgelegt werden und durch entsprechende Wrapper in Android, iOS oder Raspberry Pi angesprochen und genutzt werden. In Abbildung 5 kann man eine vereinfachte Form dieser Architektur sehen.

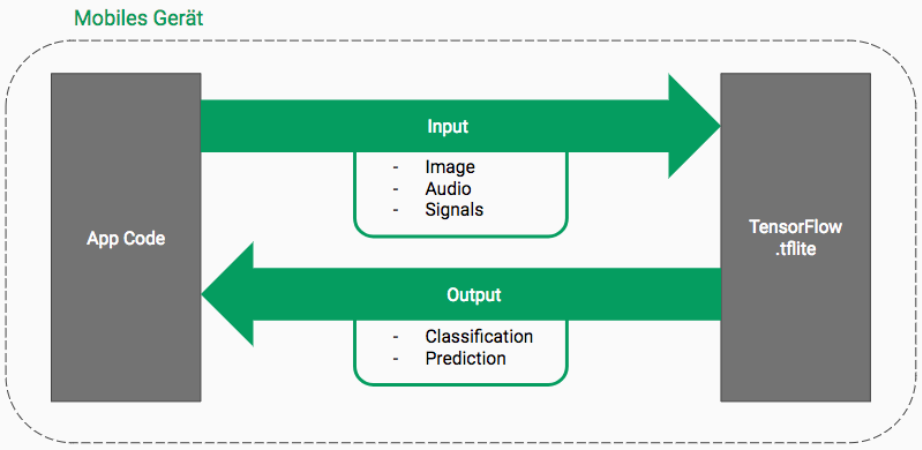

Abb. 5: Architektur von TensorFlow Lite mit Inputs und Outputs

Dabei liegt der Fokus darauf ein fertiges Modell in einer Anwendung zu nutzen. Das Training findet in der Regel auf effizienten Grafikarten statt (siehe Abbildung 6). Hierbei ist eine weitere wichtige Rolle des mobilen Gerätes die, dass sie neben der Anwendung auch das Sammeln der Daten übernimmt. Da viele Personen eine App nutzen, können diese Personen eine entsprechende Menge an Daten generieren, die wiederum für das Training und Verbessern des KNN genutzt werden kann.

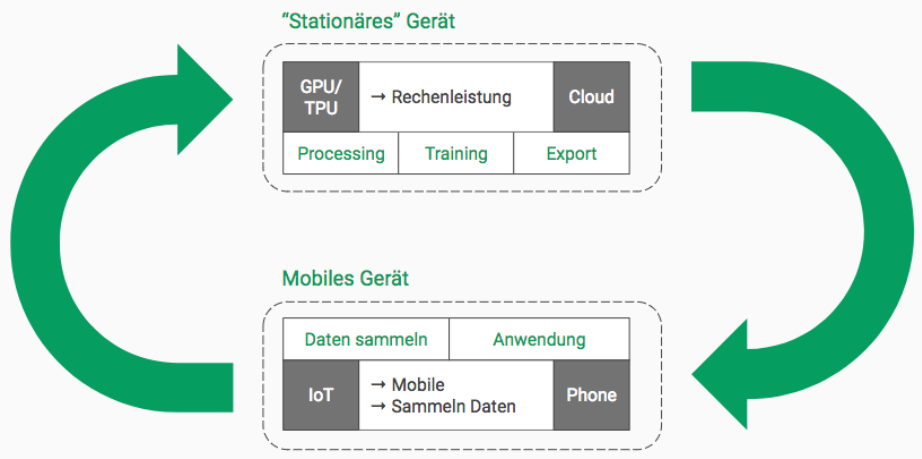

Abb. 6: Gängiges Setup von stationären und mobilen Machine Learning Geräten

3.3 Recurrent Neural Network (RNN)

RNNs sind eine Form von KNNs, bei der die Knoten eines Layers mit sich selber verknüpft sind und dadurch eine rekursive Verbindung aufbauen. In Abbildung 7 kann man ein solches Netzwerk nach einer Sequenz t sehen. Das bedeutet, dass ein solcher Knoten nicht nur die Outputs der Knoten im vorherigen Layer als Input bekommt, sondern auch seinen eigenen Output. Dadurch kann sich das Netz Inputs merken, wodurch es dazu geeignet ist sequentielle Inputs, also Zeitfolgen, zu berücksichtigen. Beispiele für solche Sequenzen sind Sprache, Börsenpreise oder Gesten.

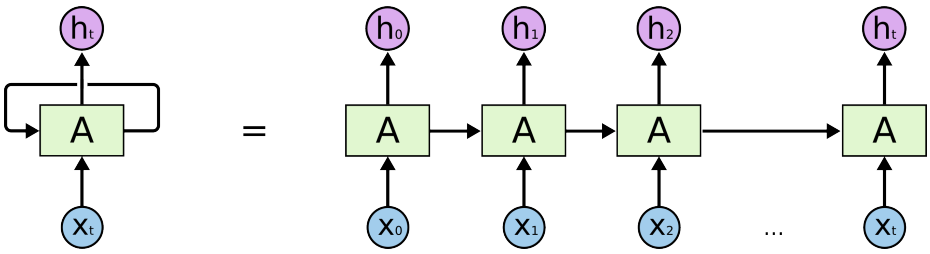

Abb. 7: An unrolled recurrent neural network. [7]

Der große Nachteil von RNNs ist, dass in längeren Sequenzen ihre Beschränkung ersichtlich wird. In Abbildung 8 sieht man eine Gegenüberstellung. Aufgrund der vielen Zeitschritte verschwimmen die Inputs. Sie haben in späteren Zeitfolgen nicht mehr die Relevanz, um den Output maßgebend zu beeinflussen. Das wird auch Vanishing Gradient Problem genannt siehe auch [8].

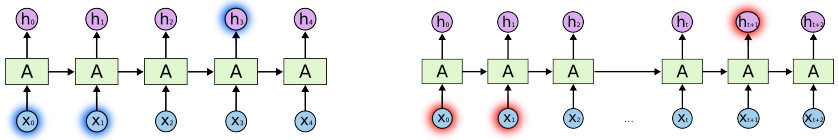

Abb. 8: Eine Gegenüberstellung der Vor- und Nachteile eines RNNs [7]

3.4 Long Short Term Memory (LSTM)

LSTMs sind eine Form von RNNs. Diese haben das Ziel den großen Nachteil der RNNs durch ein - wie der Name schon sagt - Langzeitgedächtnis zu lösen. Im Gegensatz zu herkömmlichen RNNs sind die Neuronen bzw. Zellen deutlich komplexer (siehe Abbildung 9). Im Wesentlichen besitzen diese Zellen je eine zusätzliche Input- sowie Output-Schnittstelle. Die Kernidee ist einen neuen Datenfluss zu implementieren, der Inputs über mehrere Zeitfolgen hinweg unverändert weitergibt.

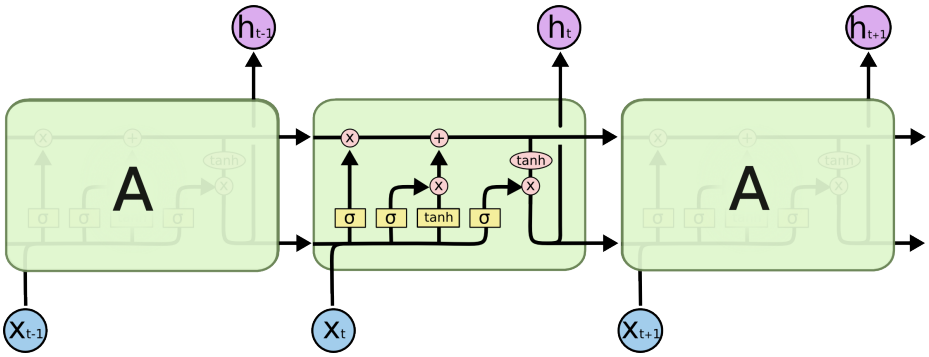

Abb. 9: Long Short Term Memory Cell [7]

Zur Steuerung dieses Datenfluss wird durch sogenannte Gates erreicht, die den Zustand der Zelle bestimmen. Dadurch können die Langzeitdaten auch vergessen werden, wenn sie nicht mehr gebraucht werden.

4 Stand der Forschung

In diesem Kapitel sollen Arbeiten gezeigt werden, die vergleichbar mit dieser sind und dabei helfen können einige Lösungsansätze für Problemstellungen zu liefern.

4.1 Automatic Drum Transcription Using Bi-Directional Recurrent Neural Networks [9]

Diese Arbeit beschäftigt sich mit dem Ansatz reale Schlagzeugaufnahmen direkt in entsprechende Noten umzuwandeln (Abbildung 10). Dabei werden je Schlagzeugkomponente ein eigenes neuronales Netz genutzt, um die Daten entsprechend zu verarbeiten. Da meine Arbeit auf Sensorbewegungen basiert, ist diese Arbeit interessant, wenn es darum geht die Trainingsdaten zu optimieren.

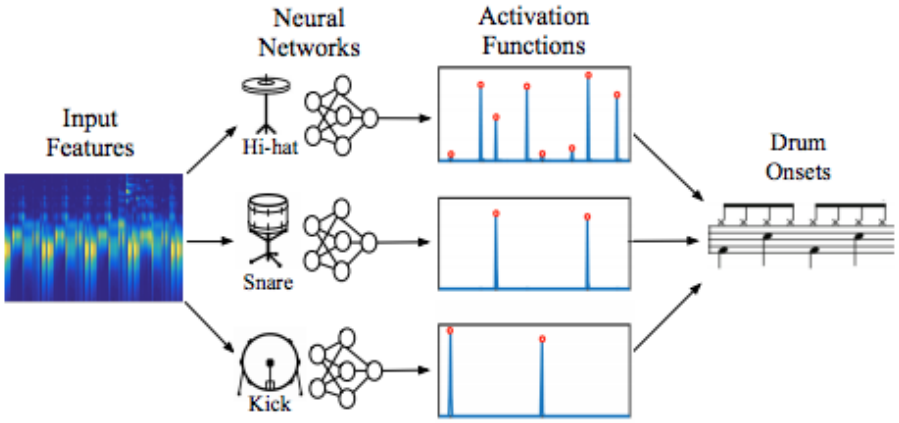

Abb. 10: Audio wird an je ein neuronales Netz pro Schlagzeugkomponente gesendet und durch entsprechenden Aktivierungsfunktionen in Noten umgewandelt. [9]

4.2 On the usage of smart devices to augment the user interaction with multimedia applications [10]

Diese Arbeit befasst sich mit der Gestenerkennung zur Interaktion mit Mediengeräten wie Fernseher oder Computer. Die Besonderheit dieser Arbeit ist, dass zunächst ein linearer Algorithmus erkennt, ob es sich überhaupt um eine Geste handelt. Erst dann, wird ein LSTM angesprochen, welches die konkrete Geste erkennt. Durch diesen effektiven Modellierung können die Schlagzeugschläge von nicht relevanten Bewegungen wie z.b. Ruhepausen differenziert werden, sodass das genutzte Netz entlastet wird.

4.3 Evaluation of Microgesture Recognition Using a Smartwatch [11]

Diese Arbeit nutzt ein in einer Smartwatch implementiertes SVM[5]-Model, um kleinste Bewegung des Handgelenks zu registrieren und in Gesten zu klassifizieren. In Abbildung 11 sieht man die erkannten Gesten.

Abb. 11: Beispiele der erkannten Gesten. [11]

Diese Arbeit ist insofern interessant, da ein Schlagzeugspiel auch aus entsprechenden Mikrogesten bestehen kann und das Erkennen dieser ein wichtiger Faktor ist.

4.4 Abgrenzung der eigenen Arbeit

In diesem Abschnitt sollen Abgrenzungen zu anderen Arbeiten hervorgehoben werden, wo meiner Ansicht nach Forschungsbedarf besteht.

Extraktion von Trainingsdaten: Die benötigten Trainingsdaten für ein Schlagzeugspiel sind äußerst vielfältig müssen an die entsprechende Hardware (Smartwatches, Raspberry Pi) angepasst sein, um gute Ergebnisse zu erzielen. Daher ist das Erzeugen von Trainingsdaten ein zentraler Bestandteil meiner Arbeit, die in anderen Forschungsarbeiten nicht behandelt wird.

Kollaborative Netzarchitektur: Verknüpfungen zwischen gleichartigen und verschiedenen neuronalen Netzen wird in wissenschaftlichen Arbeiten meistens als sequentielle Komponente genommen. D.h. zunächst läuft der Input durch ein Netz und anschließend durch ein weiteres. Ich möchte jedoch versuchen zwei Netze komplett miteinander zu verbinden, sodass in meinen Anwendungsbeispiel ein Drumstick mit der Position des anderen entsprechend interagieren kann.

Schlagzeuggesten erkennen: Wie in [12] und [9] erwähnt wird, gibt es im Schlagzeugspiel Passagen - insbesondere mit der Snare -, die sehr schwierig zu erkennen sind. Hier muss geprüft werden, inwieweit die Erkennung von Mikrogesten auf das Schlagzeugspiel übertragen werden kann.

[5]Support Vector Machines

Latenz: Die möglichen Verzögerungen durch die Hardware-Komponenten und der Lösungsansatz Predictions zu nutzen, um diese auszugleichen, wurde nicht genannt.

5 Methodik

Diese Kapitel soll die Methodik zeigen und wie Schritt für Schritt vorgegangen werden soll, um die Forschungsfrage bzw. Problemstellung zu beantworten.

5.1 Hardware-Setup

Für das Hardware-Setup bieten sich mehrere unterschiedliche Komponenten an, die ich an dieser Stelle noch nicht festlegen, aber priorisieren will. Gemein haben diese Setups, dass echte Drumsticks verwendet werden sollen und auf festen Oberflächen gespielt werden soll.

Smartwatch + Smartwatch: Ein Paar aus Fitnessarmbändern wäre ergonomisch und stylish die beste Lösung. Leider habe ich während meiner Recherchen kein geeignetes Produkt gefunden, das SDKs für die sensiblen Sensordaten liefert. Lediglich teure Modelle ab 200 Euro bieten entsprechende Schnittstellen an.

Arduino[6] **+ Arduino:** Zwei selbst zusammen gelötete Arduionos sind klein und lassen sich nach belieben konfigurieren. Mit Adafruit-Komponenten lassen sich die Platinen sogar in Stoff einnähen. Dadurch können dedizierte Armbänder hergestellt werden, die man für die Schlagzeug-App verwenden kann. Leider bieten Arduinos weder die nötige Leistung noch entsprechende Schnittstellen für TensorFlow an.

Raspberry Pi (kabellos oder verkabelt): Zwei Raspberry Pis sind relativ groß im Vergleich zu den Arduinos, können aber mit TensorFlow Lite laufen. Es gibt hierbei die Alternative sie kabellos über Bluetooth laufen zu lassen oder direkt zu verkabeln. Ein Problem könnte der Akku sein. Nichtsdestotrotz ist dies die flexibelste Lösung.

Smartwatch + Smartphone: Da Smwartwatches mit geeigneten SDKs, die Zugriff auf die benötigten Sensoren bieten, relativ teuer sind und die Wahrscheinlichkeit, dass eine Person zwei dieser Uhren gleichzeitig trägt gering ist, ist die Idee eine Kombination aus Smartwatch und Smartphone zu nutzen. Das Smartphone würde dann über einen Fitnesshalter am Unterarm befestigt werden. Problem an diesem Setup ist, dass das Nutzererlebnis durch die unterschiedlichen Größen und Gewichte leidet.

5.2 Traningsdaten

Sobald ein Hardware-Setup gefunden wurde und diese aufnahmefähig sind, sollen die Trainingsdaten generiert werden. Hierzu soll ein kleiner Aufbau aus Töpfen,

[6] Eine Hard- und Software Firma, die frei konfigurier- und programmierbare Hardware anbietet

Büchern und ähnlichen Gegenständen als Schlagzeug-Ersatz dienen. Dieser provisorische Aufbau soll als allgegenwärtige Alternative zu einem echten Schlagzeug dienen, um das Endergebnis auf diesem Setup zu testen.

Dauer der Trainingsaufnahmen: Die Dauer der Trainingsaufnahmen müssen noch evaluiert werden. Im ersten Zug soll eine Stunde Spiel aufgenommen werden. Wenn man von ca. 120 Schlägen pro Minute ausgeht, ergeben das 7200 Trainingssequenzen.

Datenanalyse: Zunächst soll bereits vor dem Aufbereiten der Daten untersucht werden, inwieweit die Rohdaten Auskunft über eine mögliche Klassifizierung geben. Dazu soll eine Sequence Clustering Technik genutzt werden, um bereits im Vorfeld Muster zu erkennen. Das **Hidden Markov Model** ist ein bewährtes Modell zum Erkennen solcher sequentiellen versteckten Daten soll auf das Datenset angewendet werden.

Datenextraktion: Falls die Datenanalyse keine hilfreichen Ergebnisse liefert, sollen über die Rohdaten der Sensoren (Beschleunigungsmesser, Gyroskop, Magnetometer, Mikrofon) die Schläge extrahiert werden. Dazu müssen Algorithmen entwickelt werden, die Peaks in den Daten erkennen und verlässlich die die Sequenzen zwischen höchstem Ausholpunkt und Schlag ermitteln. Einen aktuellen Stand der Peak Erkennung kann man in Abbildung 12 sehen.

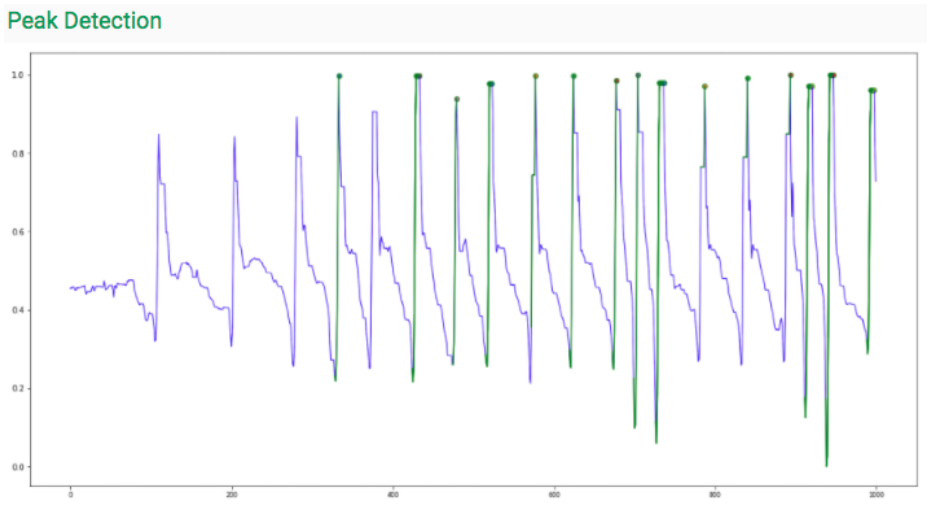

Abb. 12: Beschleunigungsmesser für die X-Achse. Maxima und Minima werden erkannt.

Clustering und Labeling: Als nächstes sollen erneut Clustering-Algorithmen angewendet werden, um die extrahierten Schläge zu kategorisieren und mit entsprechenden Lablen zu versehen, sodass diese Bereit sind als Trainingsset in das Model zu gehen. Hier soll gegebenenfalls das Mikrofon als Hilfe genutzt werden, um die Schläge zu labeln.

5.3 Model

Die genaue Form des Models soll durch Versuche evaluiert werden. Hinsichtlich der Prediction könnte ein solches Netz wie in Abbildung 13 aussehen. Dabei soll n Schritte bevor der Peak erreicht wird, der Sound erkannt werden. Der Aufbau und das Training verlaufen über TensorFlow und Keras.

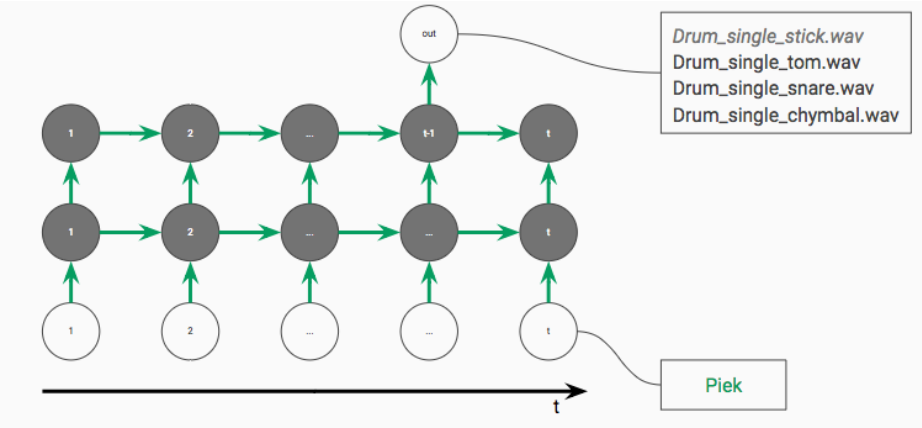

Abb. 13: Beispiel des verwendeten RNNs

5.4 Transfer Anwendung

Dieser Schritt wird es sein die Anwendung auf das mobile Gerät zu überführen und dort im Live-Betrieb zu testen. Für das leichtere Testen und Debuggen soll eine Pipeline eingerichtet werden, die das trainierte Netz automatisch auf die Mobilgeräte überträgt.

6 Risiken

Die angestrebte Arbeit hat viele Risiken, die in diesem Kapitel näher erläutert werden sollen. Da das Eintreten von Risiken eine Umstrukturierung der Arbeit zur Folge hat, sind die möglichen Maßnahmen aufgeführt.

6.1 Trainingsdaten

Wie so oft bei der Entwicklung von künstlichen neuronalen Netzen ist ein großes Risiko die unzureichende Qualität und Quantität der Trainingsdaten. In meinem Beispiel will ich Sensordaten selbst durch mehrere Stunden Schlagzeugspiel

erzeugen und anschließend mit Clustering-Algorithmen in unterschiedliche Bewegungsabläufe kategorisieren. Ob dieses Vorgehen am Ende hinreichende Trainingsdaten liefert, um ein zufriedenstellendes Resultat zu erreichen, lässt sich schwer einschätzen. Ebenso muss getestet werden inwieweit manuelle Optimierung der Daten die Ergebnisse verbessern können.

Maßnahme: Falls die erzeugten Trainingsdaten nicht zu der gewünschten Qualität führen, muss die Methodik zu deren Erzeugung überdacht und gegebenenfalls mit zusätzlichen Hilfsmitteln wie Kamera- oder Tonaufnahmen verbessert werden. Hierzu können die Arbeiten von [9] und [12] Lösungsansätze bieten, um zusätzliche Hilfsmittel zum Labeln der Trainingsdaten zu haben. Hierbei nutzen die Autoren die Audio-Spektogramme von Schlagzeugspielen, die sie über verschiedene Methoden aus Bidirektionalen RNN oder CRNN[7] versuchen in Musiknoten zu überführen.

6.2 Latenz der Bluetooth-Verbindung

Ein weiteres Risiko können Latenzen der Bluetooth-Verbindung darstellen. Selbst eine geringe Verzögerung von knapp über einer Sekunde kann das Nutzererlebnis schmälern und dadurch das Ziel der Arbeit gefährden. Mein Lösungsansatz dieses Risiko zu reduzieren ist - wie bereits in einem vorherigen Kapitel erwähnt - Vorhersagen mit Hilfe von Machine Learning zu nutzen. Ob dieser Lösungsansatz auch sporadische Latenzeinbrüche regulieren kann, kann auch hier noch nicht gesagt werden. Ebenso ist nicht sichergestellt, ob die Vorhersage mit einer ausreichend hohen Performance funktioniert und wie oft falsche Vorhersagen auftreten.

Maßnahme: Falls in diesem Zusammengang sich während der Arbeit herausstellt, dass eine Bluetooth-Verbindung trotz Predictions nicht die erwartete Performance bringt, ist die Idee auf Kabelverbindungen auszuweichen, um schnellere und robustere Übertragungswege bereitzustellen.

7 Zusammenfassung

Diese Ausarbeitung stellt eine Entwicklung einer Schlagzeug-App vor, bei der der Nutzer ohne Schlagzeug, aber mit Drumsticks und entsprechenden Wearables auf haushaltsüblichen Objekten spielen kann. Dabei wurden zwei Kernfragen vorgestellt, die beantwortet und evaluiert werden sollen. Zum einen, wie zwei KNNs kollaborativ miteinander interagieren können und zum anderen ob mit Hilfe von Vorhersagen mit Latenzproblemen umgegangen werden kann. Diese Fragen wurden an konkreten Problemstellungen am Beispiel der Schlagzeug-App vorgestellt. Die vorgestellten Grundlagen und Publikationen zeigen die Technologien sowie die Abgrenzung zu bestehenden Arbeiten. Im anschließenden Kapitel wurde über die Methodik diskutiert und hervorgehoben, dass es variable Ansätze zum Lösen der Fragen geben soll. Diese verschiedenen Vorgangsweisen sollen dabei in

[7] Convolutional Recurrent Neural Network

der Entwicklung evaluiert werden, sodass die am besten geeigneten Methoden gewählt werden. Anschließend wurden die entscheidenden Risiken der Arbeit dargelegt und Maßnahmen vorgestellt, wie man damit umgehen kann.

Literatur

1. Freedrum. https://www.freedrum.rocks/. Accessed: 2018-08-30.
2. Smartphone sensors. https://developer.apple.com/documentation/coremotion/. Accessed: 2018-08-30.
3. Google. https://www.google.com/about/our-company/. Accessed: 2018-08-30.
4. Tensorflow. https://www.tensorflow.org/. Accessed: 2018-08-30.
5. Protocol buffers. https://developers.google.com/protocol-buffers/?hl=en. Accessed: 2018-08-30.
6. Flat buffers. https://google.github.io/flatbuffers/. Accessed: 2018-08-30.
7. Understanding lstms. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed: 2018-08-30.
8. Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media, 2017.
9. Jason Hockman Carl Southall, Ryan Stables. Automatic drum transcription using bi-directional recurrent neural networks. *Proceedings of the 17th ISMIR Conference*, pages 7–11, 2016.
10. D. Puccinelli A. Ferrari, V. Galli and S. Giordano. On the usage of smart devices to augment the user interaction with multimedia applications. *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–9, 2017.
11. S. Agarwal and S. Ghosh. Evaluation of microgesture recognition using a smartwatch. *017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 986–991, 2017.
12. Peter Knees et. al. Richard Vogl. Drum transcription via joint beat and drum modeling using convolutional recurrent neural networks. *Proceedings of the 18th ISMIR Conference*, pages 23–27, 2017.