Mozilla DeepSpeech: Speech-to-Text Schritt für Schritt

Originalartikel

Backup

<html> <header class=„article-header“><h1 class=„articleheading“>Mozilla DeepSpeech: Speech-to-Text Schritt für Schritt</h1> Pascal Moll </header><figure class=„aufmacherbild“><figcaption class=„akwa-caption“>(Bild: petrmalinak/Shutterstock.com)</figcaption></figure> Dieses Tutorial zeigt anhand eines Praxisbeispiels, wie sich ein Sprachassistent mit DeepSpeech auf einem Raspberry Pi erstellen lässt. Intelligente Lautsprecher oder Sprachassistenten sind auf dem Vormarsch. Mehr als ein Drittel [1] aller Deutschen nutzt sie. Doch was passiert eigentlich mit den dadurch erzeugten Daten? Es gibt verschiedene Vermutungen, dennoch lässt sich nicht sicher sagen, was mit den gesprochenen Informationen geschieht. Um die volle Kontrolle über diese Daten zu behalten, entschied sich der Autor, selbst eine Offline-Anwendung mit Java zu entwickeln. Dieses Tutorial soll den Speech-to-Text (STT)-Anteil des Entwicklungsprozesses ausschnittsweise vorstellen. Es folgen Einblicke in die Installation der DeepSpeech Engine mithilfe eines Raspberry Pi. Diese Komponenten nehmen Gesprochenes entgegen und überführt das Gesprochene in Text. Im nächsten Schritt erfolgt die Einführung des zugehörigen Java-Programmcodes, um mit diesem Text zu interagieren und Sprachbefehle zu designen. Der Ausblick zeigt zahlreiche Erweiterungsmöglichkeiten und Einsatzgebiete. <h3 class=„subheading“ id=„nav_einführung_in0“>Einführung in Mozilla DeepSpeech</h3> Mozillas DeepSpeech ist eine freie Speech-to-Text-Engine. Sie arbeitet mit einem durch Maschine Learning erstellten Sprachmodell, basierend auf den Forschungsergebnissen von Baidu’s Deep Speech Research Paper [2] und ermöglicht somit, Gesprochenes in Text umzuwandeln. Hier kommt unter anderem Googles Machine-Learning-Framework TensorFlow zum Einsatz, was die Implementierung vereinfachen soll [3]. TensorFlow Lite eignet sich besonders für mobile Geräte, da es deutlich kleinere Sprachmodelle bei vergleichbaren Ergebnissen und Geschwindigkeit mitbringt. Mozilla arbeitet daran, verschiedene Modelle in anderen Sprachen zu generieren. Dafür nutzen sie das Projekt Common Voice. Das Ziel dieses Crowdsourcing-Projekts ist, eine freie Datenbank mit verschiedenen Sprachen und Sprechern aufzubauen. Mitwirkende können entweder andere Spracheingaben validieren oder selbst Sprachaufnahmen beisteuern. Zur Teilnahme wird lediglich ein Mikrofon

benötigt. Für die englische Sprache ist dieses Projekt schon weit entwickelt, so ist die Fehlerquote sehr gering und selbst für nicht Muttersprachler mit Dialekt vielseitig anwendbar. Das deutsche Sprachmodell funktioniert ebenfalls gut, hat aber noch Schwierigkeiten bei Wörtern, die ähnlich klingen. Beispielsweise wird „Licht an“ oft als „Lichter“ erkannt. <h3 class=„subheading“ id=„nav_benötigte1“>Benötigte Komponenten und Voraussetzungen</h3> Für das STT-Projekt sind diese Komponenten notwendig: <ul class=„rtelist rtelist–unordered“>
Raspberry Pi 4 (ein PI3 B+ ist ausreichend, kann aber zu Verzögerungen bei der Ergebnisrückgabe führen)
Java 8 mit Maven
Python 3.5 auf dem Raspi
Mikrofon (je nach Entfernung empfiehlt sich ein Ringmikrofon)
<h3 class=„subheading“ id=„nav_deepspeech_als2“>DeepSpeech als Server aufsetzen</h3> Zunächst sollte das Standardbetriebssystem „Raspberry PI OS“ installiert sein. Anschließend empfiehlt es sich, die gesamte Speicherkarte als Speicher über das Konfigurationsmenü bereitzustellen. Somit entfallen mögliche Speicherprobleme zu einem späteren Zeitpunkt, beispielsweise bei der Verwendung eines größeren Sprachmodells. <figure class=„a-inline-image a-u-inline“> <figcaption class=„a-caption“>Aufruf der Config mit sudo raspi-config (Abb. 1)</figcaption> </figure> Über Advanced Options (Punkt 7) und A1 Expand Filesystem lässt sich die gesamte Größe der Speicherkarte nutzen. Darüber hinaus ist das Aktivieren des Secure File Transfer Protocol (SSH) oder Virtual Network Computing (VNC) zu empfehlen, um nicht immer wieder externe Peripherie für die Bedienung anschließen zu müssen. Nach erfolgreichem Abschluss der Grundinstallation gilt es, DeepSpeech zu installieren, was über den Konsolenbefehl

```
pip3 install deepspeech
```

geschieht. Nach wenigen Minuten Wartezeit sollte die Erfolgsmeldung erscheinen: <figure class=„a-inline-image a-u-inline“> <figcaption class=„a-caption“>Erfolgsmeldung: DeepSpeech erfolgreich installiert (Abb. 2)</figcaption> </figure> Mit

```
pip3 install deepspeech-server
```

wird nun der Server installiert. Dieser Vorgang kann einige Minuten in Anspruch nehmen. <figure class=„a-inline-image a-u-inline“> <figcaption class=„a-caption“>Benötigte Sprachmodellkomponenten (Abb. 3)</figcaption> </figure> Anschließend ist DeepSpeech installiert, jedoch ist noch kein Sprachmodell hinterlegt. Pre-Trained-Modelle lassen sich via GitHub herunterladen [4]. Für das deutsche Modell empfiehlt sich jedoch diese Adresse [5]. Dieses Modell beinhaltet 1582 Sprach-Stunden. Benötigt werden das Modell TensorFlow lite und der Scorer. <figure class=„a-inline-image a-u-inline“> <figcaption class=„a-caption“>Scorer und Tflite Datei in /home/pi/.local/bin abgelegt (Abb. 4)</figcaption> </figure> Beide Dateien gilt es herunterzuladen und in der Konfiguration zu hinterlegen. Am einfachsten ist dies mit dem VNC Viewer möglich. Im letzten Schritt ist es notwendig, die Konfiguration zu hinterlegen. Hierzu dient eine neue Datei namens config.json. <pre class=„rtetx–listing listing“><code>Config.json{ „deepspeech“: { „model“ :„output_graph_de.tflite“, „scorer“ :„kenlm_de.scorer“, „beam_width“: 500, „lm_alpha“: 0.931289039105002, „lm_beta“: 1.1834137581510284 }, „server“: { „http“: { „host“: „0.0.0.0“, „port“: 8080, „request_max_size“: 1048576 } }, „log“: { „level“: [{ „logger“: „deepspeech_server“, „level“: „DEBUG“}] }}</code></pre> Zum Abschluss der Konfiguration ist nun ein Neustart des

Raspi erforderlich. <h3 class=„subheading“ id=„nav_starten_des3“>Starten des DeepSpeech-Servers</h3> Um den Server zu starten, ist im Verzeichnis cd /home/pi/.local/bin der Steuerbefehl

```
deepspeech-server --config config.json
```

einzugeben. <figure class=„a-inline-image a-u-inline“> <figcaption class=„a-caption“>DeepSpeech-Server erfolgreich gestartet (Abb. 5)</figcaption> </figure> Sollte der Fehler Original error was: libf77blas.so.3: cannot open shared object file: No such file or directory auftauchen, hilft der Befehl:

```
sudo apt-get install libatlas-base-dev
```

. DeepSpeech-Aufruf Um DeepSpeech per Curl aufzurufen, wird die IP des Raspberry-Servers benötigt sowie der Curl-Befehl:

```
curl -X POST --data-binary @testfile.wav http://IP:8080/stt
```

<h3 class=„subheading“ id=„nav_deepspeech4“>DeepSpeech verlangt eine Sounddatei</h3> Aktuell existieren keine nativen Java Libraries, um DeepSpeech zu nutzen. Es gibt lediglich eine Android-Bibliothek, die für Desktop-Anwendungen jedoch wenig hilfreich ist. Daher ist es erforderlich, DeepSpeech als Server aufzusetzen. Somit lässt sich eine Verbindung über HTTP aufbauen und ein eigenes Java-Programm entwickeln. Der Curl-Befehl lässt es schon vermuten: DeepSpeech verlangt eine Sounddatei im WAV-Format, die entgegengenommen, interpretiert und dann als Text zurückgeliefert wird. Das nachfolgende Programm geht genauso vor. <h3 class=„subheading“ id=„nav_blick_in_die5“>Blick in die Bibliothek</h3> Zunächst sind zwei Bibliotheken notwendig. Beide befinden sich im Maven Central Repository und lassen sich einfach ins Project Object Model (POM) einbinden: <pre class=„rtetx–listing listing“><code><dependencies> <dependency> <groupId>org.apache.httpcomponents</groupId> <artifactId>httpclient</artifactId> <version>4.5.13</version> </dependency> <dependency> <groupId>commons-io</groupId> <artifactId>commons-io</artifactId> <version>2.8.0</version> </dependency></dependencies></code></pre><h3 class=„subheading“ id=„nav_ans_eingemachte6“>Ans Eingemachte mit Java</h3> Der nachfolgende Code ist vereinfacht dargestellt: Exception Handling wurde bewusst weitestgehend vernachlässigt. Der Client besteht aus den drei Methoden

```
transcribe()
```

,

```
createAudioOutputStream(...)
```

und

```
sendDataToServer(...)
```

. Übertragen des Gesprochenen Die Methode

```
transcribe()
```

öffnet den Mikrofoneingang und legt gleichzeitig einen Buffer für das Gesprochene an. <pre class=„rtetx–listing listing“><code>AudioFormat format = new AudioFormat(16000, 16, 1, true, false);</code></pre> Die Abtastrate des Eingangssignals wird hiermit auf 16000, die Samplesize auf 16 und die Kanäle auf Mono festgelegt. Der <code>true</code>-Wert gibt an, dass das Eingangssignal signed ist, es kann also sowohl positive als auch negative Werte beinhalten. Das ist notwendig, da Sprachsignale eben solche Werte beinhalten. Der <code>false</code>-Wert steht für <code>littleEndian</code> und gibt die Speicherreihenfolge an;<code>true</code> steht für <code>bigEndian</code>. Diese Einstellungen sind erforderlich, da die hinterlegten Sprachmodelle von DeepSpeech ebenfalls mit diesen Werten abgelegt wurden. Ein Wechsel von mono auf stereo beeinflusst die Erkennung negativ. Schließlich wird das Mikrofon mit dem vorgegebenen Audioformat geöffnet und ist bereit Signale zu empfangen. <pre class=„rtetx–listing listing“>

```
DataLine.Info info = new DataLine.Info(TargetDataLine.class,
format);if ( ! AudioSystem.isLineSupported(info)) {        throw new
Exception("Mikrofon ist nicht verf&amp;#252;gbar!");}TargetDataLine line = (
TargetDataLine )AudioSystem.getLine(info);line.open(format,
line.getBufferSize());
```

</pre> Zum Speichern des Eingangssignales wird noch ein Buffer zur Verfügung gestellt: <pre class=„rtetx–listing listing“><code>ByteArrayOutputStream out = new ByteArrayOutputStream();byte[] data = new byte[line.getBufferSize() / 4];</code></pre> Nachdem alle Vorbereitungen abgeschlossen sind, lässt sich das Signal aufzeichnen und abspeichern: <pre class=„rtetx–listing listing“>

```
System.out.println("StartLine");line.start();long startTime =
System.currentTimeMillis();while ( ( System.currentTimeMillis() - startTime
) &amp;lt; 2000) {        int numBytesRead = line.read(data, 0, data.length);
out.write(data, 0, numBytesRead);}System.out.println("ende");
```

</pre> Der Eingang hört zwei Sekunden zu und speichert das Gesprochene in dem zuvor festgelegten Buffer. In einem nächsten Schritt wird der

```
ByteBuffer
```

an die Methode

```
createAudioOutputStream(...)
```

weitergegeben. Das Ergebnis davon geht an

```
sendDataToServer(...)
```

. <pre class=„rtetx–listing listing“><code>ByteArrayOutputStream byos = createAudioOutputStream(line, out);return sendDataToServer(byos);</code></pre> Ausgabe in die Datei Die Methode <code>createAudioOutputStream</code> wandelt den Inhalt des

<code>ByteBuffer</code> in eine WAV-Datei um. Dazu berechnet sie die benötigte WAV-Größe, liest den <code>ByteBuffer</code> ein und schreibt schließlich eine WAV als <code>ByteArrayOutputStream</code> raus: <pre class=„rtetx–listing listing“>

```
private ByteArrayOutputStream createAudioOutputStream(TargetDataLine line,
ByteArrayOutputStream out)    throws IOException {    int frameSizeInBytes =
line.getFormat().getFrameSize();    byte audioBytes[] = out.toByteArray();
AudioInputStream audioInputStream = new              AudioInputStream(new
ByteArrayInputStream(out.toByteArray()),
line.getFormat(),                          audioBytes.length /
frameSizeInBytes);        ByteArrayOutputStream byos = new
ByteArrayOutputStream();        AudioSystem.write(audioInputStream,
AudioFileFormat.Type.WAVE, byos);        audioInputStream.close();
return byos;}
```

</pre> Kommunikation mit DeepSpeech Nachdem nun die WAV-Datei als

```
ByteArrayOutputStream
```

vorliegt, lässt sich eine Verbindung zum DeepSpeech Server aufbauen und die Datei dorthin senden: <pre class=„rtetx–listing listing“><code>private String sendDataToServer(ByteArrayOutputStream byos) throws IOException, ClientProtocolException { HttpClient httpclient = HttpClients.createDefault(); HttpPost httppost = new HttpPost(SERVERADRESS); ByteArrayEntity bae = new ByteArrayEntity(byos.toByteArray()); httppost.setEntity(bae); HttpResponse response = httpclient.execute(httppost); HttpEntity entity = response.getEntity(); if (entity != null) { try (InputStream instream = entity.getContent()) { return IOUtils.toString(instream, „UTF-8“); } } return „Kein Ergebnis“;}</code></pre> Die gezeigte Methode baut zunächst eine Serververbindung auf und setzt anschließend einen <code>Post</code>-Befehl ab. Die Serveradresse sieht beispielsweise so aus: http://localhost:8080/stt. Das zuvor erstellte <code>ByteArray</code> wird nun in eine Entity verpackt und an den Server gesendet. Zum Abschluss erfolgt das Auslesen der zurückgegebenen Response. <figure class=„a-inline-image a-u-inline“> <figcaption class=„a-caption“>DeepSpeech-Ergebnis von „Hallo Welt“ (Abb. 6)</figcaption> </figure><figure class=„a-inline-image a-u-inline“> <figcaption class=„a-caption“>Eclipse Konsolenausgabe von „Hallo Welt“ (Abb. 7)</figcaption> </figure><h3 class=„subheading“ id=„nav_fazit_und7“>Fazit und Ausblick</h3> Der hier vorgestellte Prozess ist nur ein Ausschnitt von dem, was mit DeepSpeech möglich ist. Der eigene Sprachassistent des Autors ist mittlerweile in der vierten Iteration und ermöglicht es ihm, das Licht zu steuern, Termine aus dem Kalender abzufragen und YouTube zu nutzen. Problematisch sind englische Begriffe und Worte, die ähnlich klingen. Auch wurde hier bewusst das Reagieren auf unterschiedliche Lautstärkepegel vernachlässigt. Mit dem Audiodispatcher und Silencelistener der Tarsos-Bibliothek lässt sich diese Problematik lösen, da der Sprachassistent erst dann reagiert, wenn ein bestimmter Lautstärkepegel überschritten wird und auch erst dann aufhört zuzuhören, wenn dieser Pegel wieder unterschritten wird. Damit ist eine fixe Zeitangabe für das Zuhören nicht mehr notwendig. Unberücksichtigt blieben eigene Sprachmodelle. Es besteht auch die Möglichkeit, eigene Wörter oder eigene Modelle zu entwickeln und mit TensorFlow zu trainieren. Dabei lassen sich Entfernungen zum Mikrofon oder Umgebungsgeräusche kompensieren. Die Möglichkeiten sind grenzenlos und stehen den gängigen Sprachassistenten auf dem aktuellen Markt in nichts nach. Mit diesem Vorgehen lassen sich die

Sprachkompetenzen von DeepSpeech auch für Java nutzen und nicht mehr nur für Python. Der vorgestellte Code findet sich auf GitHub [6]. Auf dem eigenen Blog zeigt der Autor [7] unter anderem die Möglichkeiten Philips Hue Leuchten mit Java zu steuern. Für den Sommer plant er seinen Sprachassistenten auf GitHub zur Verfügung zu stellen. Pascal Moll
ist freiberuflicher Berater und Java-Entwickler. Seine Schwerpunkte liegen im Bereich des Test-Managements und Testautomatisierung von Web - und Desktopapplikationen insbesondere SAP. Neben seiner Beratertätigkeit arbeitet er auch als freiberuflicher Trainer für Java, Cucumber und Selenium Schulungen. Mehr Informationen finden sich auf https://pmo-it.de [8]. () <hr /> URL dieses Artikels:
<small>

```
https://www.heise.de/-6048698
```

</small> Links in diesem Artikel:
<small>

```
<strong>[1]</strong> https://www.heise.de/meldung/Fast-jeder-Dritte-in-Deutschland-nutzt-Sprachassistenten-4443365.html
```

</small>
<small>

```
<strong>[2]</strong> https://arxiv.org/abs/1412.5567
```

</small>
<small>

```
<strong>[3]</strong> https://deepspeech.readthedocs.io/en/r0.9/
```

</small>
<small>

```
<strong>[4]</strong> https://github.com/mozilla/DeepSpeech/releases
```

</small>
<small>

```
<strong>[5]</strong> https://drive.google.com/drive/folders/1oO-N-VH_0P89fcRKWEUlVDm-_z18Kbkb
```

</small>
<small>

```
<strong>[6]</strong> https://github.com/PMO-IT
```

</small>
<small>

```
<strong>[7]</strong> http://pmo-it.de/blog/
```

</small>
<small>

[8] https://pmo-it.de

</small>
<small>

[9] mailto:mdo@ix.de

</small>
 Copyright © 2021 Heise Medien </html>