School of Engineering

Lab 13: Deep Learning-basierte Keyword Detektion

Ein herzliches Dankeschön für den Entwurf des Labs geht an Philipp Schmid, ZHAW-ISC, heute Sonova.

1. Einleitung

Mit Hilfe von Deep Neural Networks lassen sich vielfältige Klassifizierungsprobleme lösen. In diesem Praktikum wird ein Keyword-Detektor trainiert und in einer Echtzeitanwendung eingesetzt. Keyword-Detektoren werden beispielsweise in Sprachassistenten eingesetzt, um diese zu aktivieren. Bekannte Beispiele hierfür sind „Hey Google“ beim Google Assistant oder „(Hey) Siri“ bei Apple’s Siri.

Dieses Praktikum verwendet eine Teilmenge des Speech Commands Dataset [1] von Google. Die Version 1 des Datensatz besteht aus ca. 65’000 Beispielen von 1900 verschiedenen Sprechern und 30 verschiedenen Wörtern (ein Wort pro .wav Datei). Wir werden versuchen zehn verschiedene Worte (z.B. «Yes», «No», «Up», «Down», …) zu detektieren. Dafür werden wir ein Convolutional Neural Network trainieren, welches 1s lange Audiosegmente als Input nimmt und dann die Wahrscheinlichkeiten ausgibt, dass ein gegebenes Segment eines der Kommandos, ein unbekanntes Wort oder nur ein Hintergrundgeräusch ist (siehe Abbildung 1).

Obwohl das Zeitsignal direkt als Input verwendet werden könnte (siehe z.B. https://pytorch.org/tutorials/intermediate/speech_command_classification_with_torchaudio_tutorial.html), transformiert man das 1D Zeitsignal oftmals in eine geeignetere Darstellung (z.B. 2D Mel-Spektrogramm). Durch eine geschickte Wahl der Signaldarstellung kann häufig die Grösse (Anzahl Parameter) des neuronalen Netzes bei gleicher Performance reduziert werden.

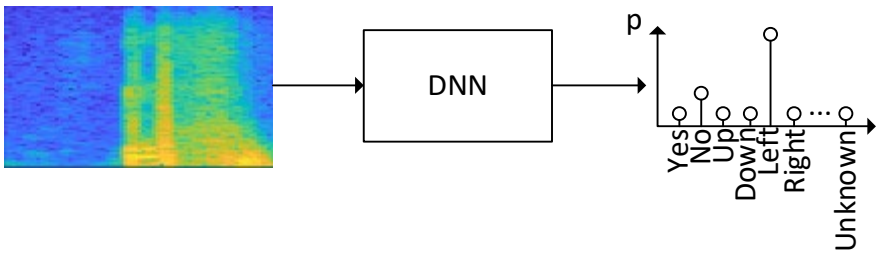

Abbildung 1: Konzept der Deep Neural Network (DNN)-basierten Keyword Detektion.

Für eine real-time Detektion der Keywords werden wir das trainierte DNN in eine GUI einbauen. Um dann ein Keyword zu detektieren, werden wir den Audiostream in **überlappende Segmente** unterteilen und diese einzeln mit dem DNN klassifizieren. Um die Robustheit der Klassifikation zu erhöhen, wird zusätzlich eine «Abstimmung» über mehrere Segmente implementiert.

Das heutige Praktikum werden wir hauptsächlich mit **MATLAB** durchführen. Wenn in einem Projekt oder Team MATLAB verwendet wird, eignet sich die Deep Learning Toolbox von MATLAB (https://ch.mathworks.com/products/deep-learning.html) sehr gut zur Entwicklung von Deep Learning Modellen. Die beliebtesten und dominanten Deep Learning Frameworks mit Python-Frontend sind jedoch **TensorFlow** und **PyTorch**.

2. Aufgabenstellung

2.0 Inspektion Daten

Eine ausreichend grosse Menge gut gelabelter Daten ist eine Grundvoraussetzung für erfolgreiches Machine Learning. Aus diesem Grund sollte ein Datensatz ausführlich untersucht werden. Gehen Sie darum durch die folgenden Punkte:

a) Entpacken Sie das File `daten.zip` im Keyword Detektion Ordner. Im Ordner «`daten`» sollte der Ordner «`google_speech`» sichtbar sein.
b) Machen Sie sich mit dem Datensatz vertraut.
    1. Welche Wörter sind im Datensatz enthalten?
    2. Wie viele Samples und wie viele Sekunden sind z.B. in der Klasse «Up» enthalten?
    3. Wie viele Sekunden Background-Geräusche hat es? Welche Sampling-Frequenz haben die Daten?
    4. Hören Sie sich einige Samples z.B. aus dem Ordner «`House`» an.

Diese Aufgabe können Sie auch gerne mit Python lösen.

2.1 Feature Extraktion

Bevor wir das neuronale Netz trainieren, werden wir die Daten vorverarbeiten. Dazu berechnen wir das sogenannte **Mel-Spektrogramm** der Audiosignale. Im Gegensatz zum klassischen Spektrogramm sind die Frequenzbins beim Mel-Spektrogramm nicht äquidistant, sondern logarithmisch auf einer sogenannten Mel-Skala verteilt. Diese Skala ist vom menschlichen Gehör inspiriert und bewirkt, dass die Frequenzbänder bei tiefen Frequenzen eine hohe Auflösung haben und schmal sind, während sie bei höheren Frequenzen breiter werden. Dadurch kann die Anzahl der Frequenzbins reduziert werden (z.B. von 256 auf 16), was den Rechenaufwand des neuronalen Netzes erheblich verringert. Ein Mel-Spektrogramm M kann berechnet werden, indem ein Spektrogramm S mittels

$$\boldsymbol{M = \mathbf{Filterbank} \cdot |\mathbf{S}|^2}$$

auf die Mel-Skala gemappt wird. Dabei ist $\mathbf{S}$ die Short-time Fourier Transformation (STFT) des Signals. Die Filterbank besteht aus mehreren Dreieckfiltern. Weitere Informationen zum Mel-Spektrogramm sind unter [2] und [3] zu finden. Als nächstes berechnen wir das Mel-Spektrogramm eines Audiobeispiels und bereiten dann eine Feature-Extraktionsfunktion für das Training des neuronalen Netzes vor.

a) Öffnen Sie das Skript `test_mel_spec.m`
b) Berechnen Sie mit Hilfe des Befehls `stft` und den Argumenten `'Window'`, `'FFTLength'`, `'OverlapLength'` und `'FrequencyRange'` das Spektrogramm des Signals y.
c) Erzeugen Sie mit dem Befehl `designAuditoryFilterBank` und den Argumenten (`'FrequencyScale','mel'`), `'FFTLength'`, `'NumBands'` und (`'Normalization'`, `'none'`) eine Mel-Filterbank und plotten Sie diese.
d) Verwenden Sie die Filterbank, um das Spektrogramm in ein Mel-Spektrogramm zu transformieren. Plotten Sie das Spektrogramm und das Mel-Spektrogramm (z.B. mit `surf`, die `EdgeColor` sollte dann `none` sein.). Vergewissern Sie sich, dass so die Dimension deutlich reduziert werden kann.

Im nächsten Schritt werden wir die Mel-Spektrogramm Berechnung nutzen, um die Audiosamples aus dem Speech Command Datensatz zu transformieren.

a) Implementieren Sie die Mel-Spektrogramm-Berechnung in der Funktion `extractFeatures.m` im `helpers`-Ordner. Verwenden Sie dazu die Funktion `melSpectrogram` und schalten die `WindowNormalization` aus. Für die Parameter des melSpectrogram-Befehls sollen der config-Struct verwendet werden (definiert im Script `train_network.m`).

Zur Info: In Python kann das Mel-Spektrogramm entweder «von Hand» z.B. mittels Scipy oder mit speziellen Audio-Libraries wie Librosa implementiert werden. Deep Learning Frameworks wie TensorFlow oder PyTorch stellen eigene Bibliotheken für Audio-Signalverarbeitung zur Verfügung. Für PyTorch gibt es beispielsweise die Library `torchaudio` (https://pytorch.org/audio/stable/index.html), welche in PyTorch integriert ist und Berechnung auf GPUs unterstützt.

2.2 Vorbereiten der Daten

Nachdem wir die Feature-Extraktionsfunktion vorbereitet haben, können wir unsere Trainingsdaten berechnen. Hierfür sind die Funktionen `preprocess_data` und `get_background` geben. Diese Funktionen lesen die Trainings- und Validationsdaten ein und rufen `extractFeatures` auf jedem Audiosample auf. Wörter, welche nicht in der `config.commands`-Liste sind, werden der Klasse «unknown» zugeordnet.

a) Öffnen Sie das Skript `train_network.m`
b) Lesen Sie die Daten ein. Wir haben ein Trainings- und Validationset. Das Validationset wird verwendet, um zu prüfen, ob das DNN auch auf ungesehenen Daten gut performt oder ob es sich nur die Trainingsdaten «gemerkt» hat bzw. overfitting vorhanden ist. Um schneller experimentieren zu können, können Sie `config.reduceDataset` im `train_network.m` Skript auf `true` setzen, dann werden nur 5 % der Daten geladen. Für Aufgabe 2.3, also dem Training des DNNs, sollen aber wieder alle Daten verwendet werden.
c) Plotten Sie einige der berechneten Mel-Spektrogramme.
d) Sind von allen Klassen gleich viele Samples in den Daten? Plotten Sie die Histogramme der Trainings- und Validationsdatenlabels.

2.3 Training des DNN

Wir haben bereits eine DNN-Architektur vorbereitet. Studieren Sie den Code. Sie können das definierte Netzwerk mit den vorverarbeiteten Daten trainieren. Das trainierte DNN wird zusammen mit der Konfiguration in ein .mat-File gespeichert. Das DNN ist ein Convolutional Neural Network (CNN). Die verschiedenen Layers sind im Folgenden kurz erklärt:

- `imageInputLayer`: Dieser ermöglicht dem DNN ein Bild (in unserem Fall das Mel-Spektrogramm) als Input zu nehmen.
- `convolution2dLayer`: Dieser Layer macht ein DNN zu einem CNN. Dieser Layer führt eine 2D-Faltung (bekannt aus der Bildverarbeitung) durch. Die Grösse des Faltungskernels kann gewählt werden. Die Gewichte des Kernels werden mit Stochastic Gradient-Descent (Backpropagation) trainiert.
- `batchNormalizationLayer`: Dieser Layer macht eine Normalisierung (Mittelwert abziehen und durch Standardabweichung teilen), wobei der Mittelwert und die Standardabweichung des Mini-Batches während des Trainings berechnet werden.
- `reluLayer`: Layer, um ReLU-Funktion auf jedes Element anzuwenden.
- `maxPooling2dLayer`: Reduziert die Dimension der Daten bzw. «downsampled» diese, in dem jeweils von einem bestimmten Bereich das Maximum gesucht wird und nur dieses an den nächsten Layer weitergegeben wird.
- `dropoutLayer`: Layer, um Overfitting zu verhindern. Setzt während dem Training zufällig Datenpunkte auf 0 (mit wählbarer Wahrscheinlichkeit). Das DNN muss so redundante Features lernen und kann sich nicht einfach die Daten “merken”.
- `fullyConnectedLayer`: Normaler Dense-Layer
- `softmaxLayer`: Aktivierungsfunktion, um aus einem Vektor eine Wahrscheinlichkeitsdichtefunktion zum machen (Summe=1, jeder Wert im Bereich [0,1])
- `weightedClassificationLayer`: Gewichtete Cross-Entropy-Loss-Funktion. So kann ausgeglichen werden, wenn nicht von allen Klassen gleich viele Samples im Trainingsset sind. Ohne diese Gewichtung würden Klassen mit mehr Samples einen grösseren Einfluss auf den Loss haben. Häufig ist dies nicht gewünscht.

Im Gegensatz zur Vorlesung wird hier anstelle des Standard Stochastic Gradient Descent (SGD) der sogenannte Adam-Optimierer verwendet. Dieser und Varianten davon sind heute die Standardwahl und bringt oft eine schnellere und bessere Konvergenz als SGD.

2.4 Berechnung des Fehlers auf dem Testset

Um die Performance eines DNNs auf ungesehenen Daten zu bestimmen, wird ein vom Training möglichst unabhängiges Testset verwendet. Es wird dann angenommen, dass die Performance auf dem Testset die Performance in einer echten Anwendung widerspiegelt. Eigentlich sollte das Testset auch unabhängig vom Validationset sein. Aus Gründen der begrenzten Datenlage, verwenden wir jedoch das Validationset zum Testen.

a) Im Skript werden mit Hilfe des `classify`-Befehls bereits die Klassen des Trainings- und des Validationsets vorhergesagt. Bestimmen Sie die Genauigkeiten bzw. den Anteil richtig klassifizierter Samples.

b) Als zweites wird eine sogenannte Konfusionsmatrix geplottet. Diese zeigt an, als was die verschiedenen Samples klassifiziert werden. z.B. zeigt die Zeile «down» an, als was die verschiedenen Samples der Klasse «down» klassifiziert wurden (es gut wäre, wenn alle als «down» klassifiziert werden, es gibt aber natürlich Fehlklassifikationen). Was bedeuten die Zahlen rechts? Welcher Anteil der «Up»-Samples wurde richtig klassifiziert?

2.5 Realtime Applikation

Im Skript `real_time_test.m` wird das in den vorherigen Schritten trainierte DNN in Echtzeit auf Ihrem Mikrofon angewendet. Es soll 20-mal in der Sekunde der DNN Klassifikator aufgerufen werden, um die Klasse und die Wahrscheinlichkeiten zu bestimmen. Diese werden in einen Buffer geschrieben. Eine Klasse wird dann ausgegeben, wenn eine bestimmte Anzahl mal in diesem Buffer eine Klasse detektiert wurde und die vorhergesagte Wahrscheinlichkeit über einem Threshold ist. Mit dem Variablen `countThreshold` und `probThreshold` können diese Thresholds konfiguriert werden. Erledigen Sie alle ToDos im Skript. Testen Sie ihr DNN mit verschiedenen Schwellwerten.

2.6 (Optional) Verbesserung der Performance

Versuchen Sie die Performance des DNNs zu verbessern. Möglich Ansätze zur Verbesserung der Performance sind:

- Andere Architekturen (evtl. grösser)
- Robustheit gegenüber Hintergrundgeräusche: Hintergrundrauschen zu den Trainingsdaten hinzufügen (mit zufälligem SNR)
- Mehr Daten sammeln (evtl. mehr «Unknown»-Daten hinzufügen)

3. Referenzen

[1]	Warden, Pete. (2018). Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. https://arxiv.org/abs/1804.03209
[2]	https://de.wikipedia.org/wiki/Mel
[3]	https://www.mathworks.com/help/audio/ref/melspectrogram.html