University of Applied Sciences Berlin

Report

AI in der Robotik

by

Tim Oelkers
s0568352

Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Supervisor: **Patrick Baumann, M.Sc.**

XX. Month, 2021

Abstract

AI in der Robotik

This should be a single paragraph of not more than 500 words, which concisely summarizes the entire proposal.

Contents

Chapter 1

Introduction

1.1 Motivation

Speech recognition systems gained immense popularity over the last years with almost every big tech company (Amazon, Google, Apple, Microsoft, etc.) developing their own systems. And since spoken language is very complex and intend can be given in many different ways, these systems are hugely complex and rely on the power of the cloud for evaluation. While these systems require huge amount of ressources and expertise to develop, command recognition is a simpler subset of speech recognition that should achievable more realisticly. It is also important, that all current speech recognition systems listen for a specific command ("Hey Google", "Alexa", etc.) as a trigger to start analyzeing, so its usefullness is still present in these complex sytems. But still, human voice is incredibly diverse and even the voice of one speaker or pronounciations of specific commands can change very rapidly, so the usefulness of these simpler systems has to evaluated in the first place. And if successful, this projects can help to cut the dependency on data collecting companies by running the command recognition locally.

1.2 Project Objective

This projects aims to develop a human command recognition system using machine learning. The scope should be limited to simple and short commands and its usefulness should be evaluated in a realistic, imperfect environment. This includes testing the model with a real microphone in a noisy environment with multiple speakers. It should be also evaluated for multiple people with different voice tones. The system should be able to run on a middle class computer. The main evaluation aspect is the accuracy of the trained model, the training itself (e.g. training time) should not be evaluated.

1.3 Proceeding and Structure of the Work

The first step will be focused on research on the field of command recognition. A specific type of machine learning has to be selected and the design has to be considered before the implementation. Ideally, a existing model has already been developed and this projects builds on top of it. The implementation will focus on two different parts. The first one on the training of the model and the second one on the building of an evaluation environment and the evaluation itself. Here, a libary for getting audio data from the microphone has to be found. The data found during the evaluation has to be presented in an intuitive and expressive way.

Chapter 2

Fundamentals

2.1 Robot Operating System

ROS (Robot Operating System) is an open-source meta operating system. It runs on top of Unix based systems and provides abstraction over process management, low-level communication and hardware. It also provides tools and libraries to develop applications in the field of robotics and distributed systems. ROS is build on the concept of creating a (possibly distributed) peer-to-peer network of nodes. A node is a single, modular process that should perform a dedicated task, which is implemented using C++ (roscpp) or Python (rospy). All nodes register themselfs at the ROS master, which serves as a nameserver and processer manager to the rest of the network. Nodes communicate which each other using ROS messages, which are data structures that define the type of data that is passed. The communication is usually based on ROS topics, which implement an asynchronous publish-subscribe pattern so that nodes are truly decoupled. If direct, synchronous communication with an immediate response is needed, ROS services can be used.

2.2 Neural Networks

Neural networks are a emulation of a human brain with inter-connected nodes (neurons) with weighted edges (synapses). The learning part comes from the process of altering each weight after each evaluation to better map inputs to their expected outputs. This is called supervised learning and inputs with known outputs have to be available for that. During the forward pass, the data passes through the network while the weights are applied. To achieve non-linear mappings, an activation function is applied to inputs between each layer. After that, the error to the expected value is calculated with a

predefined loss function and the weights are updated accoring to a predefined optimizer in a process called back-propagation. The process is repeated for a certain number of epochs so that the output gets closer and closer to the expected value. After the training, the model can be used on unknown input.

2.2.1 Fully Connected Neural Network

A fully connected network is a type of network where each node of one layer is connected to each node on the following layer. It is often divided into a input layer, multiple hidden layer and one output layer, which has to have the same number of nodes as the amount of classes in the classification.

2.2.2 Convolutional Neural Networks

Convolulation neural networks are often used in classification problems and are a specialisation of full connected neural networks. They are escpecially used in image and audio classification. It tries to find features or patterns in the provided data by removing noise and extracting data relevant for the actual target. They are constructed with a convolutional layer and a pooling layer. The convolutional parts moves a kernel filter over the input matrix and calculates the inner product of surrounding datapoints. This creates an overlap and helps neighboring neurons react to similiar data (e.g. similiar frequencies in audio data). In the pooling layer, the amount of data gets reduced, for example by using max-pooling, a process where only the most active neuron in a certain area is kept and the rest being discarded. This prevents the model from overfitting (valueing data not actualy relevant for the classification task) and improves training times. The data, now reduced to its features, is then passed through a fully connected layer for the actual classification.

Chapter 3

Conception

3.1 Dataset

For this projects, the "SpeechCommands"[**warden2018speech**] dataset will be used. This dataset was created by Pete Warden, working for Google, with the intent of creating a fundamental set of simple commands spoken in English with a lot of variance. The data was gather publicly and thus contains audio from different speakers with normal day-to-day audio quality and possibly even background noise. Even though the data comes from a public source, its correctness was checked manually and with various alghorhithms to remove too quiet or interrupted words. In total, the data contains 105.829 audio files (3.4 GB) for 35 words spoken by 2.618 different people. All audio files are exacly one seconds long and the data is alligned so that words all start and end roughly at the same time. The audio comes in the WAVE format with 16bit single channel (mono) values, sampled at a rate of 16kHz. The dataset also contains audio files containing common background noise, that can be used for evaluation. The dataset is split into multiple subsets for training, testing and validation, specified by specific file lists in the same folder.

3.2 Model

The model will be a deep convolutional neural network with 5 convolutional layers described as M5[**dai2016deep**]. This was choses as they claim a reasonable high accurary of over 60% in a 10 class audio classification with a training time of a minute per epoch. They use a very high sampling rate of 32kHz and a very aggresive reduction of the temporal resolution in the first layer with a kernel size (or receptive field) of 80, a stride of 4 and a ouput size of 128. After each layer, a batch normalization and a

maxpooling of size 4 is performed, further reducing the feature maps. After the first layer, the kernel size is reduced to a more normal size of 3 and the stride reduced to 1. The heavy reduction of resolution is complemented by a doubling of the amount of filters each layer. They use ReLU as the activation function. While most classification problems use multiple connected layers after the convolutional layers, the authors claim to use a "fully convolutional design" without fully connected layers. Instead, they use a global average pooling to reduce the feature maps to a single dimension. At the end, a softmax function is applied.

3.3 Audio Capture

As the quality should be evaluated with a real microphone, the audio_capture ROS package is used for this project. It can be started as a ROS node and exposes the two topics /audio and /audio_info. The first contains the raw audio data as unsigned 8 bit arrays, the second contains metadata about how the data should be interpreted. Concretly, it contains the number of channels, the sample rate, the bitrate, the sample format and the encoding format. How the data correlates to the sampled data is controlled via the encoding format (e.g. 16LE means that two bytes in the little endian format make up one sample). The data for each channel is stored in an interleaved format and contains no header information. The sample rate specifies how often the analog audio signal from the microphone is converted to a digital signal. The higher it is, the better the quality (e.g. CD audio is sampled at 44kHz). Of couse, the bit rate, mainly specified in kilo bits per second (kbps), has to be high enough to allow for that amount of information to be processed, otherwise the higher sample rate is nullified. For uncompressed audio, the bit rate can be calculated by multiplying sample rate, byte count per sample and number of channels.

3.4 Evaluation Architecture

The evaluation architecture will be developed as a ROS application as it helps with development and models a potential real world use case as close as possible. The following image shows a graph of the architecture and how the nodes interact with each other.

As described in section 3.3, the raw audio data and metadata get sent to two topics that are subscribed by the microphone node. It is responsible for recording the data when a specific user input has been made. Once another input was made, the data has to be aggregated and sent to the audio_data topic. It is subscribed by the processor

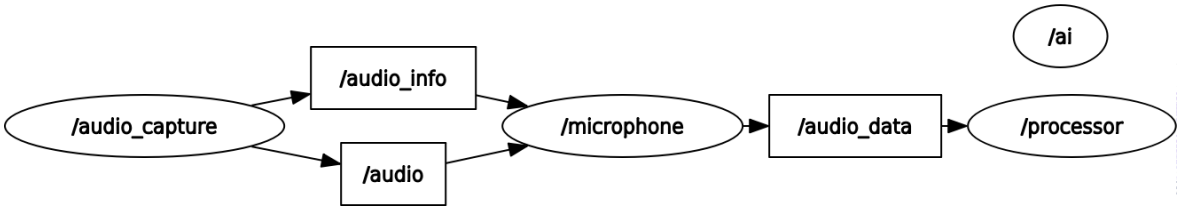

Figure 3.1: ROS architecture

node, which handles the necessary conversions so that it is compatible with the input of the model. It then also invokes the ai service which predicts the command and sends it back, together with the probability. To do that, the ai node has to load the pretrained model and metadata about the used model like the labels it can predict and the used sample rate. The predicted command will then be shown on the console for now.

Chapter 4

Implementation

4.1 Model Training

The implementation for the model training is based on a tutorial on command recognition by the pytorch team[**tutorial**]. The first step is to create a training and testing subsets of the whole dataset, that got downloaded and saved locally if needed. The testing and validation filenames are stored in specific files, everything not listed in these files is supposed to be for training. The iteration and downloading of these files is done by the SPEECHCOMMANDS class provided by torchaudio. The dataset class also has some methods to get the labels, the amount of files and the sample rate of the subset. As these methods require iteration over all files and thus take some time to compute, the results were memoized by implementing a file based cache with the help of pythons object serialization (pickle). This speeds up consecutive model trainings. One datapoint then contains the waveform, the sample rate, the label, a speaker id and the label number. As I am only able to do the training on the CPU, the audio gets resampled to 8kHz to improve training time. We then use a standard torch DataLoader and specify a batch count, the number of workers and that the data should be shuffled after each epoch so that the order is not affecting the model in any way. A so called collate function is also defined that pads the waveforms with zeroes so that all have the same length, encodes the label as the index in the label array and throws away the rest of the unnwanted data in the dataset (like the sample rate). The model is implemented according to section 3.2 with a input size equal to the number of labels. We use the Adam optimizer with a relatively high learning rate of 0.01 and a small weight decay of 0.0001 as in the original paper [**dai2016deep**]. The loss function used is called NLLLoss, which is often used in 1D classification problems. The model is then trained and tested for 6 epochs. After that, the model itself and metadata about it

(labels and sample rate) are saved to disk, together with a graph showing the accuracy and the loss for each epoch.

4.2 Evaluation Environment

All nodes are started via the following launch file. It describes where the nodes are located and how they should be configured. For the audio_capture node, we have to match the setup from the original dataset as closely as possible. Thats why we use the WAVE audio format, as it stores the data uncompressed. We also use mono audio and store the data in the 16 bit little endian format. Even though the audio will be resampled to 8kHz to fit the models sample rate, the initial sample rate should be as high as possible as during resampling, the average is calculated and we don´t want the higher variation during lower sample rates. The default bitrate set by the audio_capture package has to be overwritten to match the expected amount of data.

```
1  <launch>
2    <node name="ai" pkg="ros_ai_report" type="ai.py" output="screen"/>
3    <node name="microphone" pkg="ros_ai_report" type="microphone.py"
         output="screen"/>
4    <node name="processor" pkg="ros_ai_report" type="processor.py"
         output="screen"/>
5    <node name="audio_capture" pkg="audio_capture" type="audio_capture"
         output="screen">
6      <param name="bitrate" value="256"/>
7      <param name="channels" value="1"/>
8      <param name="sample_rate" value="16000"/>
9      <param name="format" value="wave"/>
10     <param name="sink" value="appsink"/>
11     <param name="sample_format" value="S16LE"/>
12   </node>
13 </launch>
```

The microphone node then stores the received data in memory when a specific key is pressed and joins it together when pressed again. It also calculates the sample width by extracting the information from the sample format. The processor node receives the aggregated data and writes it to file using the wave module of the python standard library. This is sadly needed, as torchaudio does not support loading of file-like objects and only expects a string as a file path. The needed metadata like sample rate and sample width are received from the microphone node. The ai node loads the

model and its metadata from disk. When a request comes in, it loads the file previously created, resamples the audio to the models sample rate and predicts the command using the model. As the model applies a softmax function at the end, the data represents the logarithm of the probability of being a certain command. That means, that the highest number is the predicted output and by applying the exponential function, the probability is calculated. As the highest element is just the index of the command, the actual command has to be found in the label array stored in the models metadata. The entire process is repeated until the user terminates the program.

Chapter 5

Evaluation

First of all, the accuracy of the model was tested agaist the testing subset provided by the used dataset. The following groups of images show the loss (left) and the accuracy (right) at each epoch.

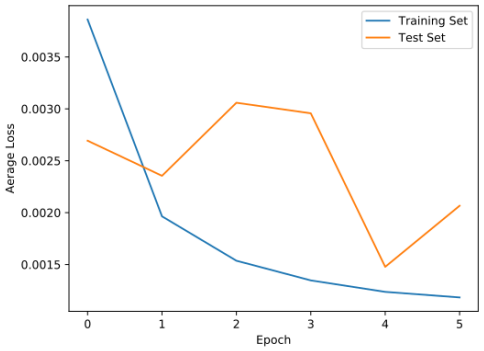

Figure 5.1: Default Model Losses

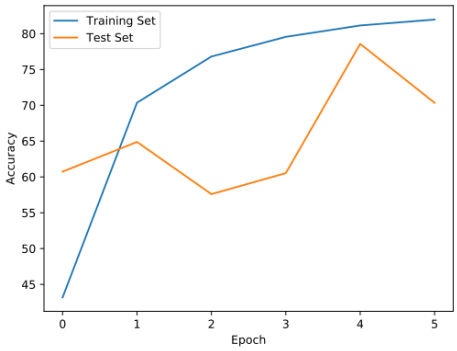

Figure 5.2: Default Model Accuracies

Interestingly, the testing accuracy seems to fluctuate and even decreases by a bit at the end while the training accuracy still keeps growing. But still, with the testing dataset reaching a 78.1% accuracy and the training dataset reaching a accuracy of 82.5%, the goal is definetly reached. For reference, the same model is tested against the validation dataset to see if there are differences in data quality.

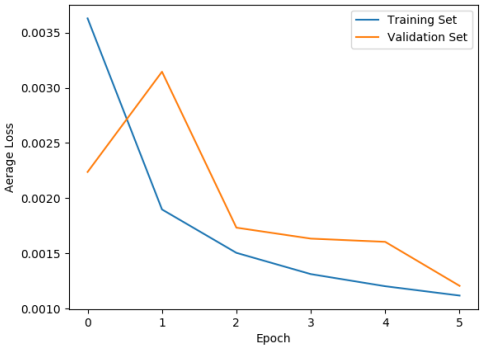

Figure 5.3: Default Model Losses

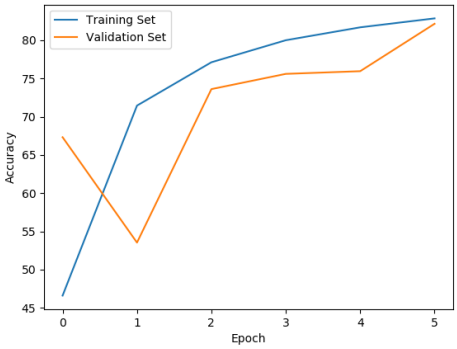

Figure 5.4: Default Model Accuracies

Here, a bit smoother curve for the validation dataset can be seens with the training set reaching 82.9% and the validation set reaching 82.2%. With the relativly small epoch count and it being relativly close to the testing accuracy, it is concluded that no difference in data quality exists.

Chapter 6

Conclusion

* length is really important for training, original data is aligned and even padded with zeroes so that all have the same length, that is not the case for the data coming from the microphone, * the model took the waveform at specifc times into consideration, not fully based on the overall structure of the wave

6.1 Summary

TODO

6.2 Future Work

TODO

List of Figures

List of Tables

Source Code Content

Appendix A

About Appendices

Appendices are optional and should only be used if necessary.