**Wake Word Detection API Project Report**
https://github.com/evgenymun/wake_word_detection

Roma Shusterman
Zan Butt
Evgeny Mun: https://www.linkedin.com/in/evgenymun

# Spotify Wake Word Detection Project

The wake word is a phrase that usually consists of two or more words. It is used to engage always listening AI into a complete power processing. Some well-known wake words are 'OK Google', 'Hey Alexa', or 'Hey Siri'. In this project, we will design a machine learning model to wake from a custom wake word like 'Hey Fourth Brain'.

Table of Contents

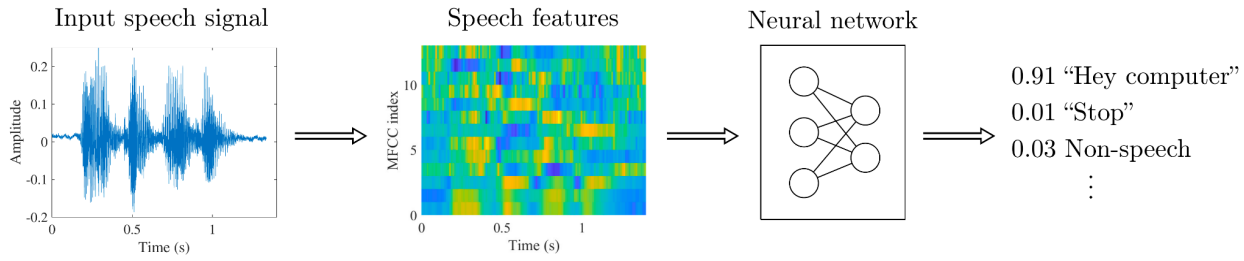


## Background and Significance of Project

Wake word detection is a critical part of the design of the system where AI is always listening to the incoming command before waking up to full power. The goal is to correctly identify the phrase, preserve power, and avoid possible privacy issues. Lack of training data can be one of the challenges in ML development.

## Related Work

According to Howl (https://www.aclweb.org/anthology/2020.nlposs-1.9.pdf) there are several open-source ecosystems available that provide a voice recognition modeling toolkit. Some of them are Snowboy and Porcupine. However, these systems still keep their models private. Howl does provide the access to all of the codes (https://github.com/castorini/howl) and an ability to train your own wake word model. In this project our plans were not so ambitious and we started with a simpler task of developing pytorch CNN and deploying it on AWS. The overall structure of keyword-spotting algorithms is illustrated below (adapted from https://wiki.aalto.fi/display/ITSP/Wake-word+and+keyword+spotting).




Adapted from Yundong Zhang, Naveen Suda, Liangzhen Lai and Vikas Chandra, "Hello Edge: Keyword Spotting on Microcontrollers"

## Explanation of Data sets

We used MCV corpus 8 en https://commonvoice.mozilla.org/en/datasets - 70Gb, 80K voices, MP3 data as a source for positive (hey, fourth & brain) and negative wav files. The data has MP3 files in the clips folder and .tsv files with transcriptions of the audio. There are train, test, and validation audio sets in the MCV. We used MFA to create timestamps for each of the MP3 files. The timestamps are needed to extract the wake words from the audio files later. To address the imbalanced data we used google cloud API to generate additional positive wav files. Then we recorded two hundred short audio files with background noise. The final positive data had about 6K audio files and  4K negative data.

## Explanation of Processes

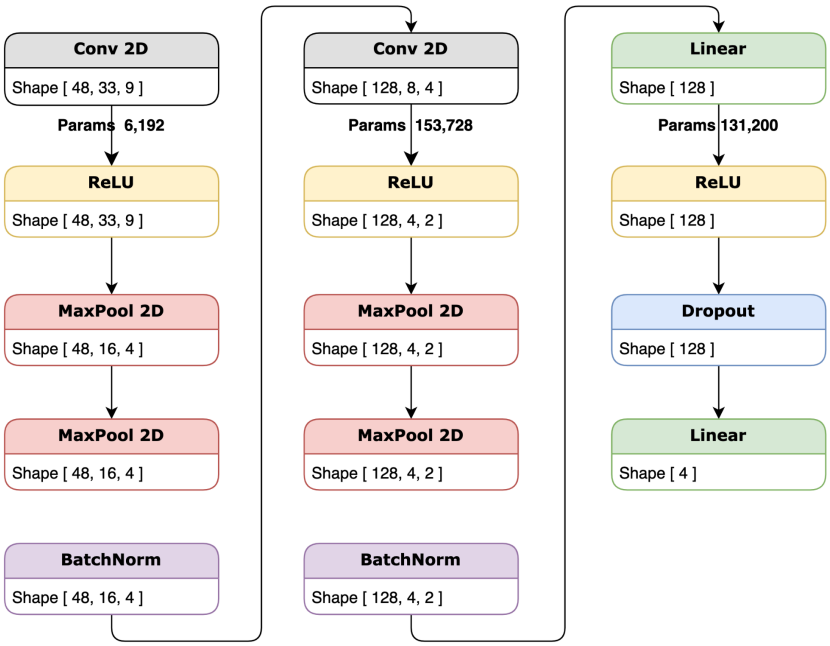
Since the MCV data is pretty big we used the personal machine to preprocess the data before moving to Google Colab. In the preprocessing, we unzipped the data and ran MFA to create the timestamps. The rest of the work was done on Google Colab. Here are the data processing steps that we performed on Colab before building the model:

- Created dataloader class to handle batch data processing
- For ML to understand the data we converted wave and frequency data into an array of numbers by applying MEL Spectogram to convert data suitable for audio classification.
- Make audio length similar between positive and negative samples. Just like images are converted to an array we converted sound into an array of equal sizes.
- Finally added the noise. MCV audio is pretty clean and doesn't have much background noise. In a real-life situation, there is always background noise.

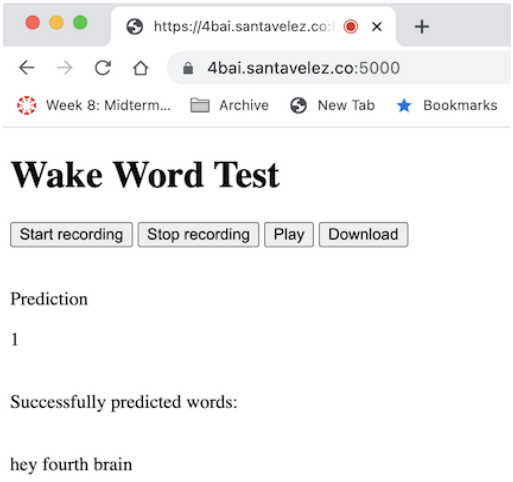
For the modeling part we:

- Used pytorch and CNN to predict the label for each of the wake words instead of the complete phrase. If the expected sequence of the individual wake words is present we confirm the detection event.
- Design # 1: 4 conv blocks / flatten / linear / softmax => Didn’t work

- Design # 2 was borrowed from https://github.com/shlbatra/TriggerWakeWordDetection: Overall we achieved a good accuracy result with this architecture but ran into some issues during a real-life test. Here is a diagram of the architecture.




Once the model was built we used HTML, JavaScript, Uvicorn and FastAPI deployed on AWS EC2 instance to test wake word detection.



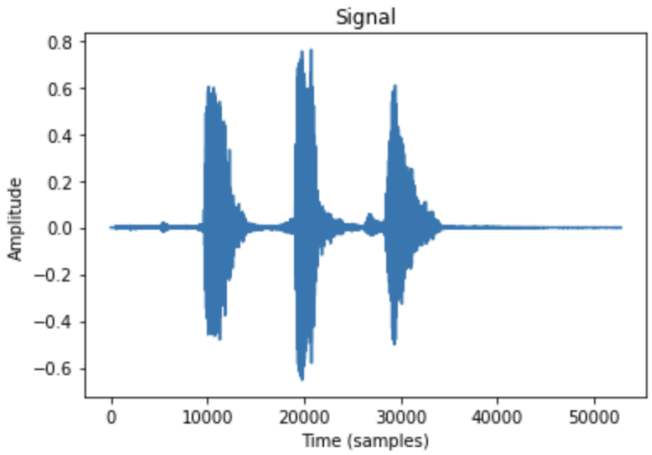

## Explanation of Outcomes

We trained the model for 20 epochs while tracking the cross entropy loss. Running the model on test data produced an Accuracy of 0.96 with an F1 score above 0.93 for wake words.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| brain | 0.99 | 0.95 | 0.97 | 604 |
| fourth | 0.99 | 0.88 | 0.93 | 337 |
| hey | 0.97 | 0.99 | 0.98 | 837 |
| negative | 0.78 | 0.99 | 0.87 | 162 |
| accuracy | | | 0.96 | 1940 |
| macro avg | 0.93 | 0.95 | 0.94 | 1940 |
| weighted avg | 0.96 | 0.96 | 0.96 | 1940 |

However, as mentioned before we ran into some challenges while testing live voice detection. Instead of using stream detection, we split the recorded audio into chunks of wav file based on the silence gap and the amplitude of the frequency. Here is the shape of one of the test audios with three words:

```
Original audio shape
```




As you can see it's not straightforward when the word ends. We also noticed that MFA produced blank mp3 files due to the incorrect transcription of the audio. Thus we ended up with some false positive data in the training set, which possibly contributed to the lower performance during the live test. Overall the results are still encouraging and indicate that CNN can detect voice phrases.

## System Design and Ethical Considerations

In this project, we focused more on designing a wake word API for testing and proving purposes. Thus our design doesn't consider possible issues with recording the audio before detecting the wake word phrase. As one can imagine recording the audio can lead to privacy and compliance concerns. In future improvements, we will move from recording audio to detecting the wake word in the stream.