Wake Word Detector

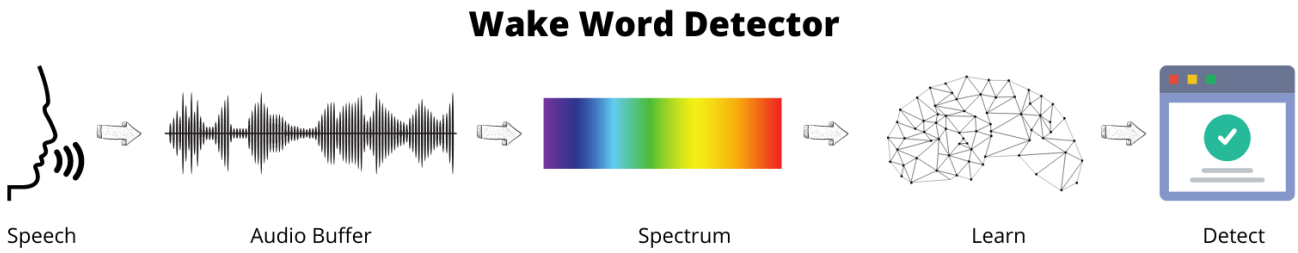


Table of Contents

Background

Personal Assistant devices like Google Home, Alexa and Apple Homepod, will be constantly listening for specific set of wake words like “Ok, Google” or “Alexa” or “Hey Siri”, and once these sequence of words are detected it would prompt to user for next commands and respond to them appropriately.

Introduction

To create a open-source custom wake word detector, which will take audio as input and once the sequence of words are detected then prompt to the user.

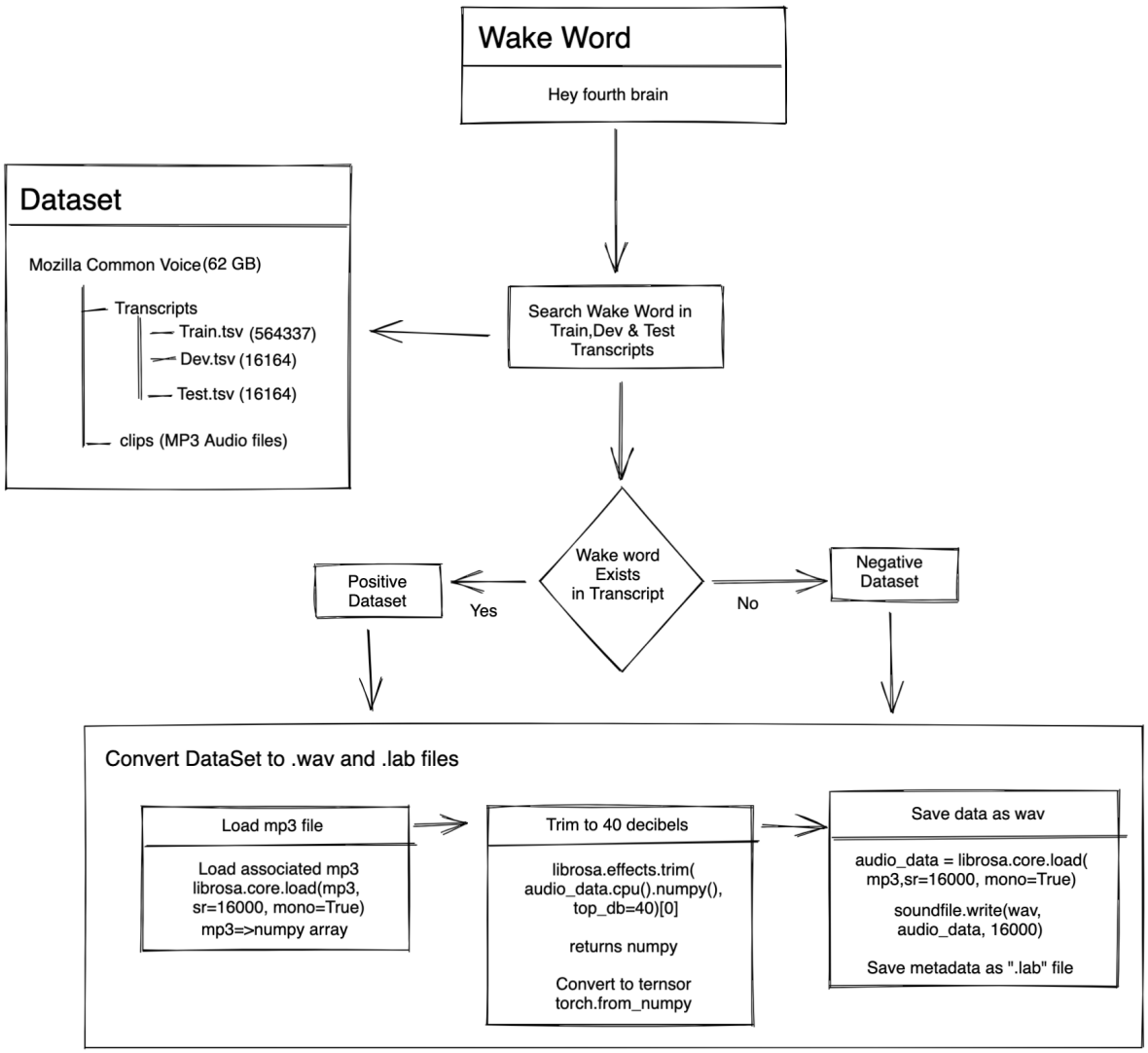
Goal is to provide configurable custom detector so that anyone can use it on their own application to perform operations, once configured wake words are detected.

Implementation

Preparing labelled dataset

Used Mozilla Common Voice dataset,

- Go through each wake word and check transcripts for match
- If found then it will be in positive dataset
- If not found then it will be in negative dataset
- Load appropriate mp3 files and trim the silence parts
- save as .wav file and transcript as .lab file
- Code reference fetch_dataset_mcv.py



Word Alignment

- For positive dataset, used Montreal Forced Alignment to get timestamps of each word in audio.
- Download the stable version

```
wget https://github.com/MontrealCorpusTools/Montreal-Forced-
Aligner/releases/download/v1.0.1/montreal-forced-aligner_linux.tar.gz
tar -xf montreal-forced-aligner_linux.tar.gz
rm montreal-forced-aligner_linux.tar.gz
```

- Download the Librispeech Lexicon dictionary

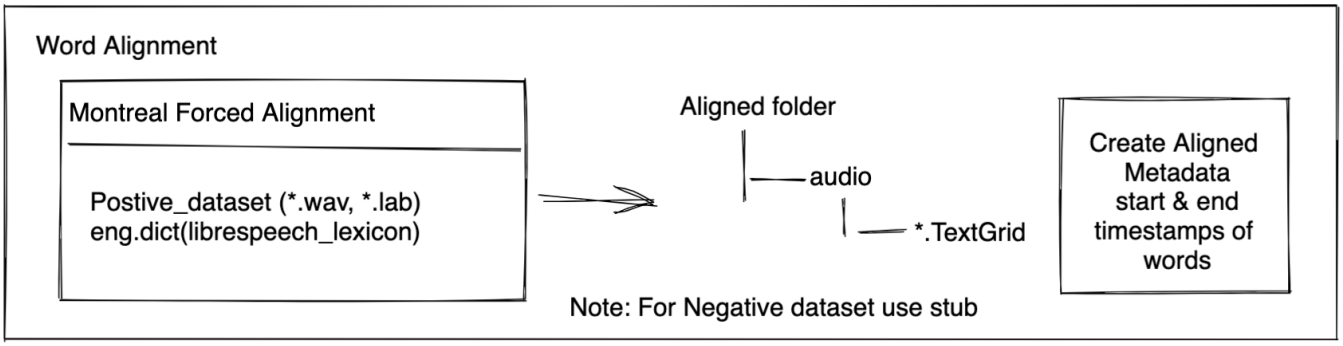
```
wget https://www.openslr.org/resources/11/librispeech-lexicon.txt
```

- Known issues in MFA

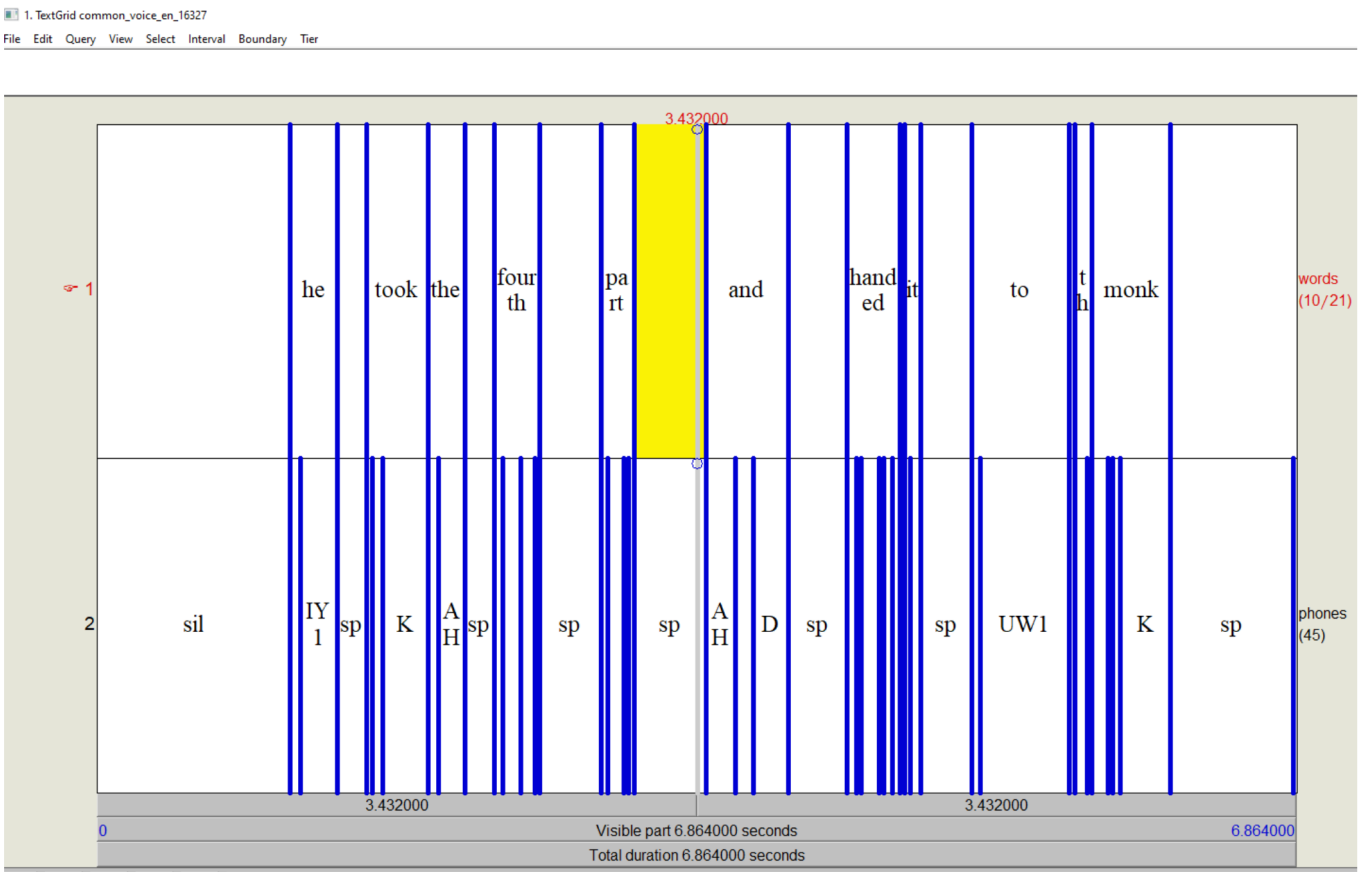
```
# known mfa issue https://github.com/MontrealCorpusTools/Montreal-
Forced-Aligner/issues/109
cp montreal-forced-aligner/lib/libpython3.6m.so.1.0 montreal-forced-
aligner/lib/libpython3.6m.so
cd montreal-forced-aligner/lib/thirdparty/bin && rm libopenblas.so.0
&& ln -s ../../libopenblasp-r0-8dca6697.3.0.dev.so libopenblas.so.0
```

- Creating aligned data

```
montreal-forced-aligner\bin\mfa_align -q positive\audio librispeech-
lexicon.txt montreal-forced-aligner\pretrained_models\english.zip
aligned_data
```



Generated textgrid file



Fix data imbalance

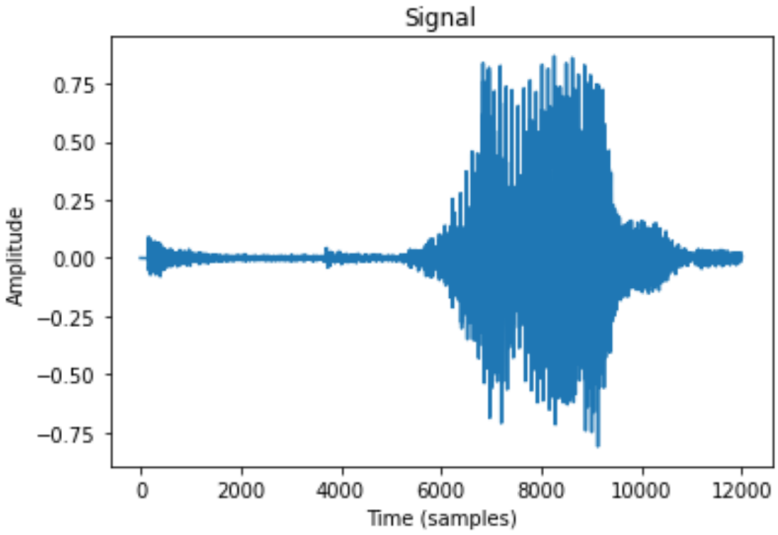
Check for any data imbalance, if the dataset does not have enough samples containing wake words, consider using text to speech services to generate more samples.

- Used Google Text To Speech Api, set environment variable `GOOGLE_APPLICATION_CREDENTIALS` with your key.
- Used various speed rates, pitches and voices to generate data for wake words.
- Code generate_dataset_google_tts.py

Extract audio features

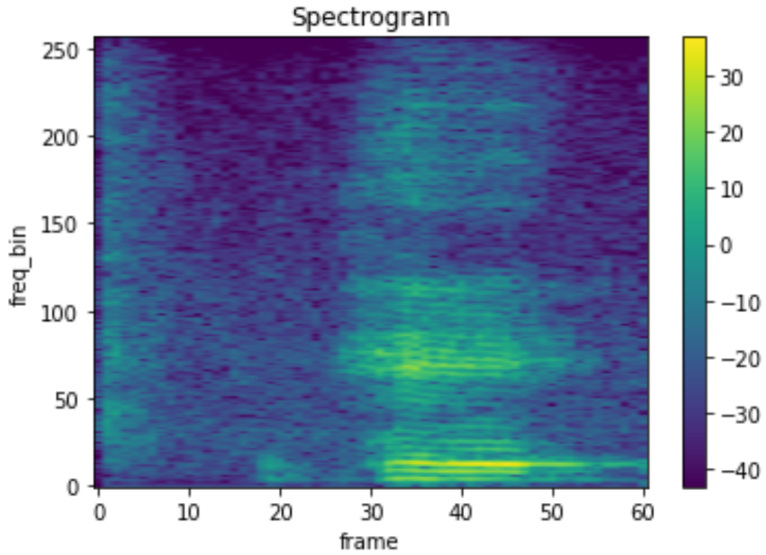
- Below is how sound looks like when plotted on time (x-axis) and amplitude (y-axis)

```
import librosa
sounddata = librosa.core.load("hey.wav", sr=16000, mono=True)[0]

# plotting the signal in time series
plt.plot(sounddata)
plt.title('Signal')
plt.xlabel('Time (samples)')
plt.ylabel('Amplitude')
```



- When Short-time Fourier transform (STFT) computed, below is how spectrogram looks like

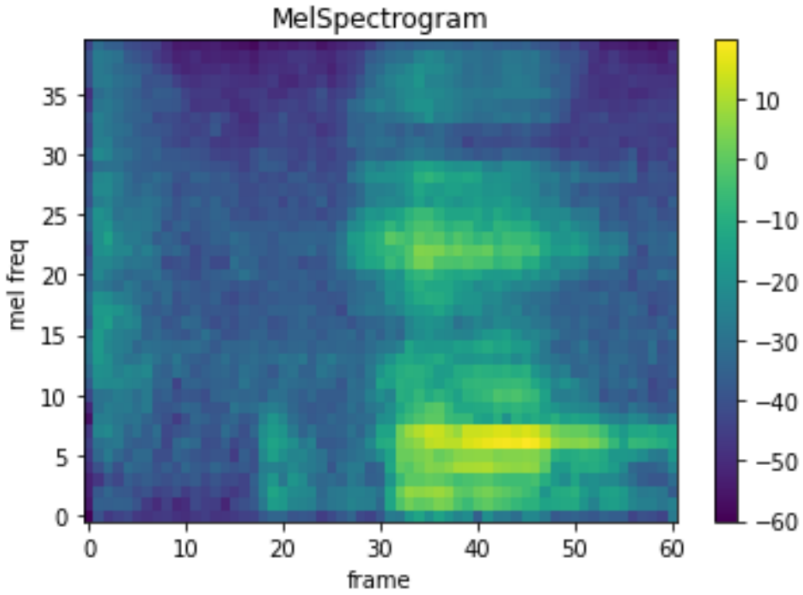
```
from torchaudio.transforms import Spectrogram
spectrogram  = Spectrogram(n_fft=512,hop_length=200)
spectrogram.to(device)

inp = torch.from_numpy(sounddata).float().to(device)
hey_spectrogram = spectrogram(inp.float())
plot_spectrogram(hey_spectrogram.cpu(), title="Spectrogram")
```



- A mel spectrogram is a spectrogram where the frequencies are converted to the mel scale.

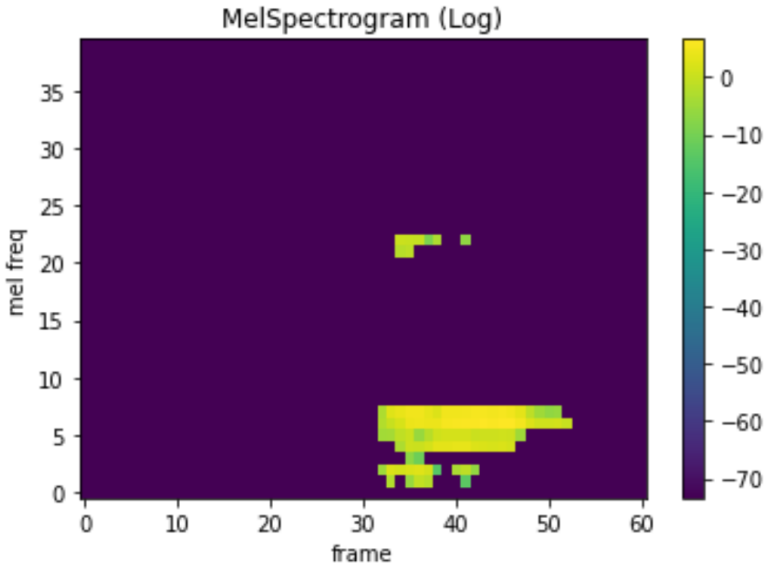
```
from torchaudio.transforms import MelSpectrogram
mel_spectrogram  = MelSpectrogram(n_mels=40,sample_rate=16000,
                                  n_fft=512,hop_length=200,
                                  norm="slaney")

mel_spectrogram.to(device)
inp = torch.from_numpy(sounddata).float().to(device)
hey_mels_slaney = mel_spectrogram(inp.float())
```

```
plot_spectrogram(hey_mels_slaney.cpu(), title="MelSpectrogram",
ylabel='mel freq')
```



- After adding offset and taking log on mels, below is how final mel spectrogram looks like

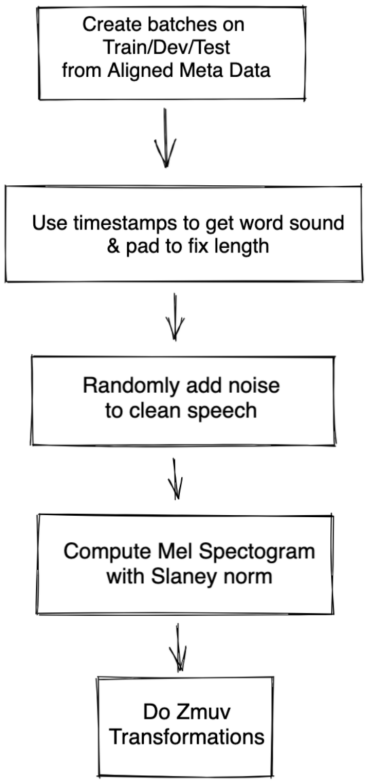
```
log_offset = 1e-7
log_hey_mel_specgram = torch.log(hey_mels_slaney + log_offset)
plot_spectrogram(log_hey_mel_specgram.cpu(), title="MelSpectrogram
(Log)", ylabel='mel freq')
```



Audio transformations

- Used MelSpectrogram from Pytorch audio to generate mel spectrogram
- Hyperparameters

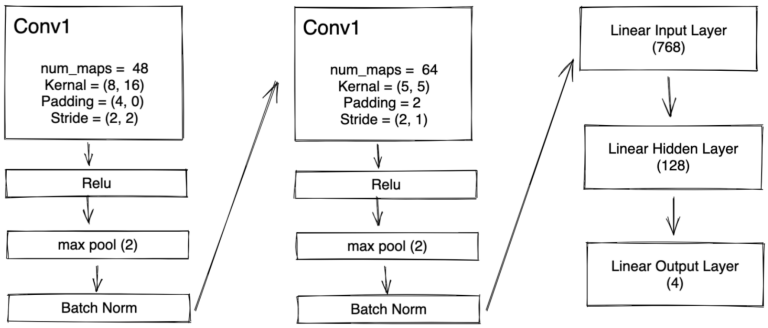
```
Sample rate = 16000 (16kHz)
Max window length = 750 ms (12000)
Number of mel bins = 40
Hop length = 200
Mel Spectrogram matrix size = 40 x 61
```



- Used Zero Mean Unit Variance to scale the values
- Code transformers.py and audio_collator.py

Define model architecture

- Given above transformations, Mel spectrogram of size `40x61` will be fed to model
- Below is the CNN model used



- Code model.py
- Below is the CNN model summary

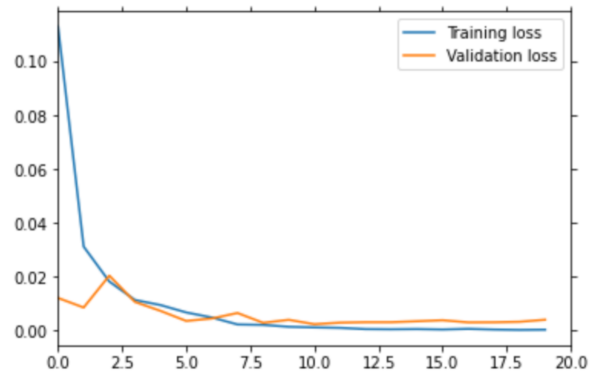
```
summary(model, input_size=(1,40,61))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 48, 31, 13]           6,192
              ReLU-2          [-1, 48, 31, 13]               0
         MaxPool2d-3           [-1, 48, 15, 6]               0
         MaxPool2d-4           [-1, 48, 15, 6]               0
       BatchNorm2d-5           [-1, 48, 15, 6]              96
            Conv2d-6            [-1, 64, 8, 6]          76,864
              ReLU-7            [-1, 64, 8, 6]               0
         MaxPool2d-8            [-1, 64, 4, 3]               0
         MaxPool2d-9            [-1, 64, 4, 3]               0
      BatchNorm2d-10            [-1, 64, 4, 3]             128
           Linear-11                 [-1, 128]          98,432
             ReLU-12                 [-1, 128]               0
          Dropout-13                 [-1, 128]               0
           Linear-14                   [-1, 4]             516
================================================================
Total params: 182,228
Trainable params: 182,228
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.46
Params size (MB): 0.70
Estimated Total Size (MB): 1.17
----------------------------------------------------------------
```

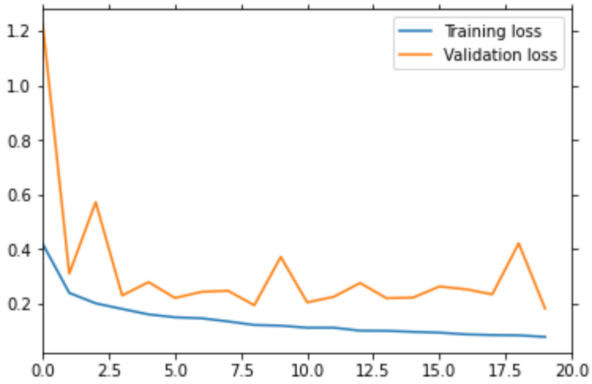
Train model

- Used batch size as 16, Tensor of size `[16, 1, 40, 61]` will be fed to Model

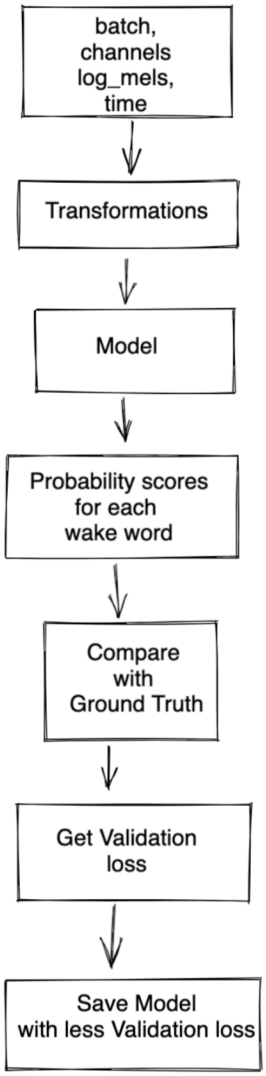
- Used 20 epochs, below is how the train vs validation loss looks like without noise



- As you can see, without noise, there is overfitting problem
- Its resolved after adding noise, below is how the train vs validation loss looks like



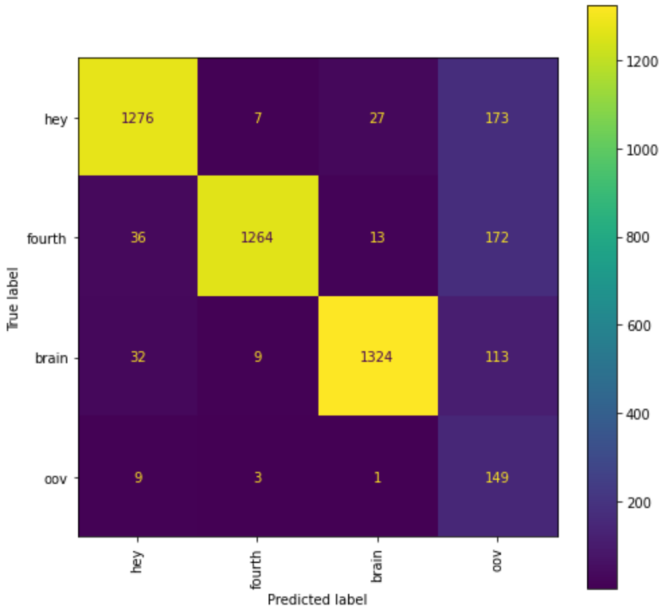
- Code - train.py



Test Model

Below is how model performed on test dataset, model acheived 87% accuracy

```
              precision    recall  f1-score   support

       brain       0.97      0.90      0.93      1478
      fourth       0.99      0.85      0.91      1485
         hey       0.94      0.86      0.90      1483
         oov       0.25      0.92      0.39       162

    accuracy                           0.87      4608
   macro avg       0.79      0.88      0.78      4608
weighted avg       0.94      0.87      0.90      4608

Test Loss: 0.606232

Test Accuracy of   hey: 86% (1276/1483)
Test Accuracy of fourth: 85% (1264/1485)
Test Accuracy of brain: 89% (1324/1478)
Test Accuracy of   oov: 91% (149/162)

Test Accuracy (Overall): 87% (4013/4608)
```



Inference

Below are the methods used on live streaming audio on above model.

Using Pyaudio

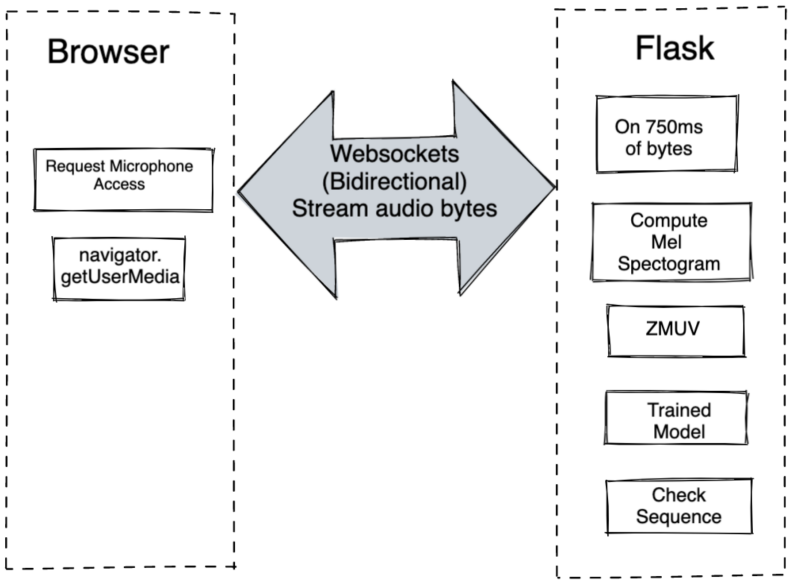
- Used Pyaudio, to get input from microphone
- Capture 750ms window of audio buffer
- After n batches, do transformations and infer on model
- Code - infer.py

```
* listening ..
C:\Users\rajas\Anaconda3\lib\site-pa
the builtin `float`. To silence this
ou specifically wanted the numpy sca
Deprecated in NumPy 1.20; for more d
['hey']
['hey', 'fourth']
['hey', 'fourth', 'brain']
Wake word hey fourth brain detected
```

Using web sockets

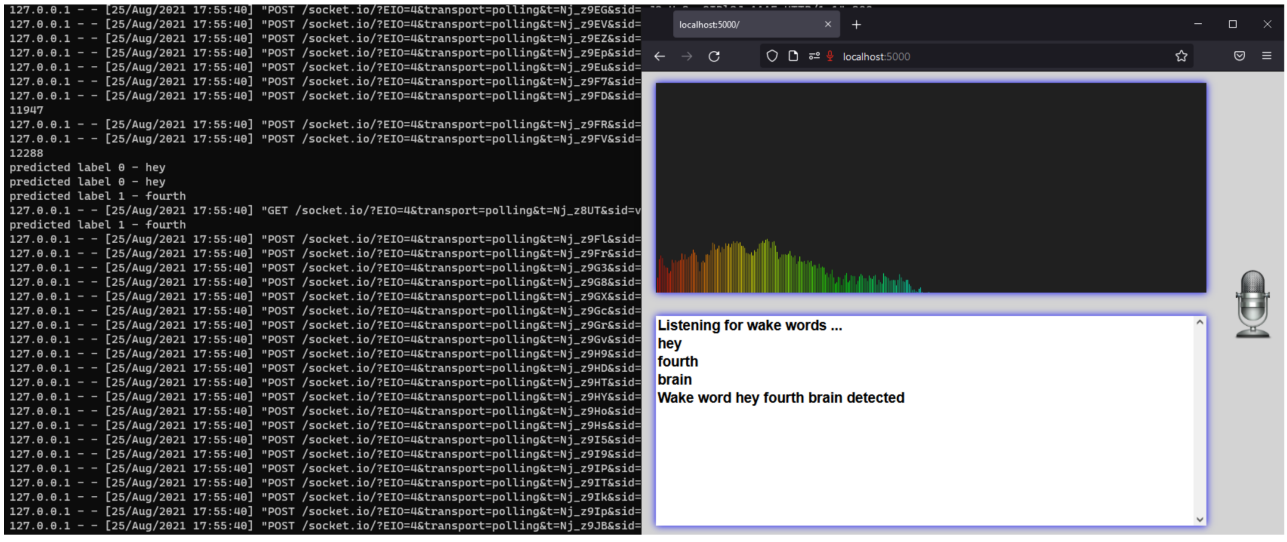
- Used Flask Socketio at server level to capture audio buffer from client.
- At Client, used socket.io at client level to send audio buffer through socket connection.
- Capture audio buffer using getUserMedia, convert to array buffer and stream to server.
- Inference will happen at server, after n batches of 750ms window

- If sequence detected, send detected prompt to client.



- Server Code - application.py
- Client Code - main.js
- To run this locally

```
cd server
python -m venv .venv
pip install -r requirements.txt
FLASK_ENV=development FLASK_APP=application.py .venv/bin/flask run --
port 8011
```



- Use Dockerfile & Dockerrun.aws.json to containerize the app and deploy to AWS Elastic BeanStalk
- Elastic Beanstalk initialize app

```
eb init -p docker-19.03.13-ce wakebot-app --region us-west-2
```

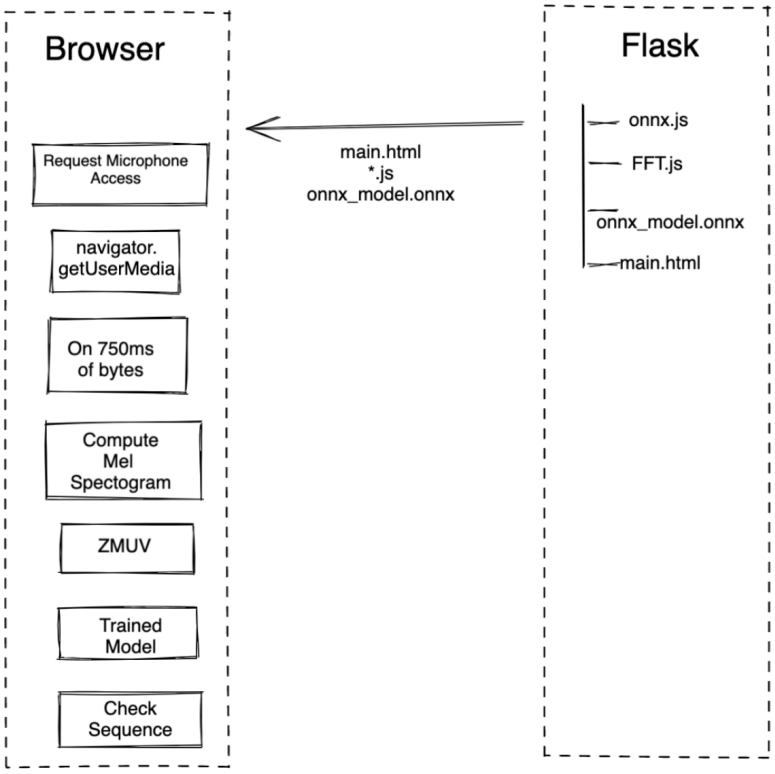
- Create Elastic Beanstalk instance

```
eb create wakebot-app --instance_type t2.large --max-instances 1
```

- Disadvantage of above method might be of privacy, since we are sending the audio buffer to server for inference

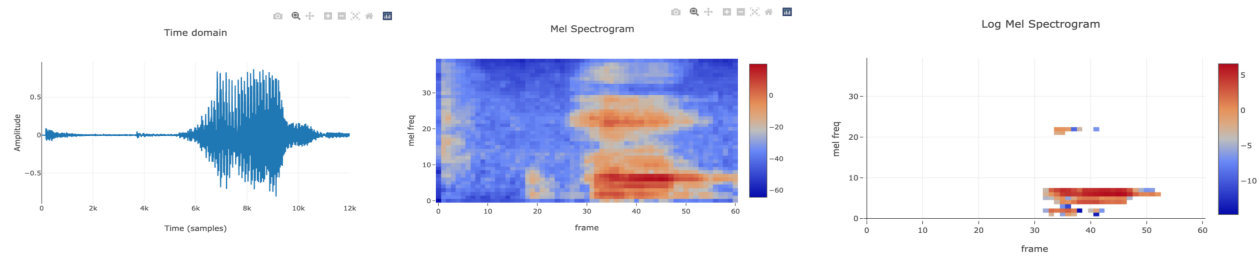
Using ONNX

- Used Pytorch onnx to convert pytorch model to onnx model
- Pytorch to onnx convert code - convert_to_onnx.py
- Once converted, onnx model can be used at client side to do inference
- Client side, used onnx.js to do inference at client level
- Capture audio buffer at client using getUserMedia, convert to array buffer
- Used fft.js to compute Fourier Transform
- Used methods from Meganta.js audio utils to compute audio transformations like Mel spectrograms

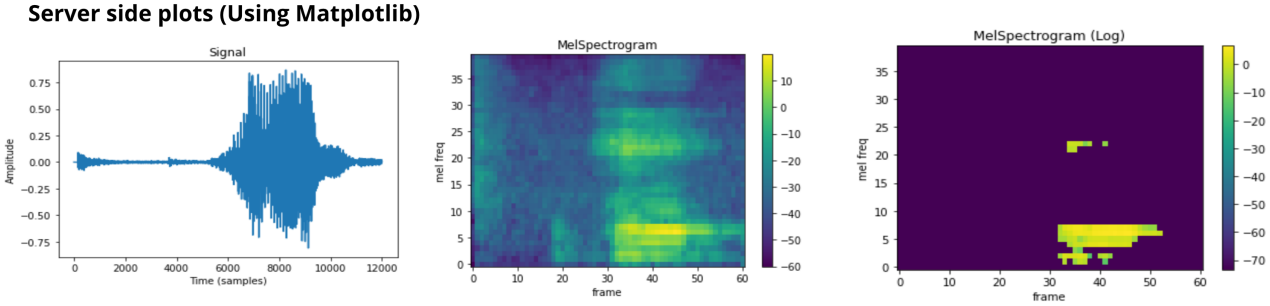


- Below is the comparision of client side vs server side audio transformations

Client side plots (Using Plotly.js)

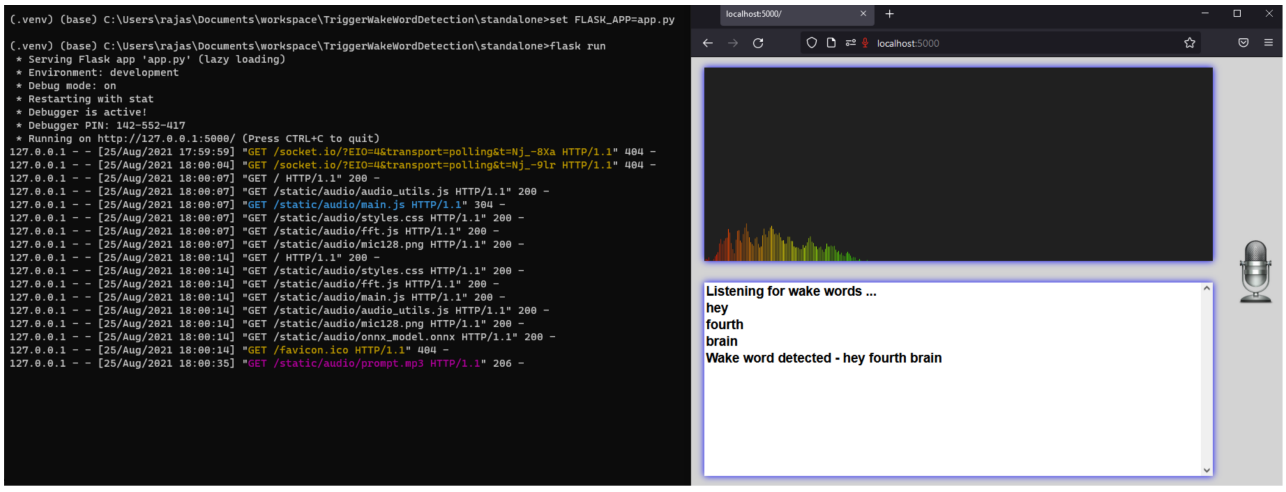


Server side plots (Using Matplotlib)



- Client side code - main.js
- To run locally

```
cd standalone
python -m venv .venv
pip install -r requirements.txt
FLASK_ENV=development FLASK_APP=application.py .venv/bin/flask run --
port 8011
```



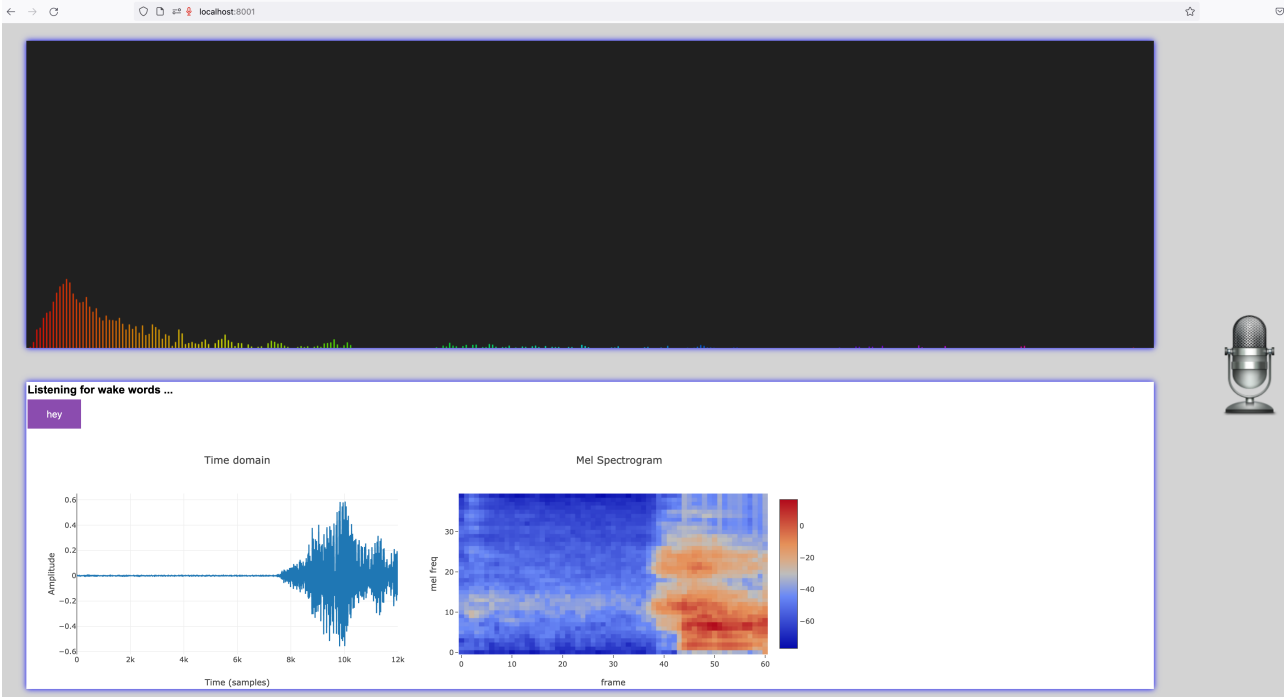
- To deploy to AWS Elastic Beanstalk, first initialize app

```
eb init -p python-3.7 wakebot-std-app --region us-west-2
```

- Create Elastic Beanstalk instance

```
eb create wakebot-std-app --instance_type t2.large --max-instances 1
```

- Refer standalone_no_flask for client version without flask, you can deploy on any static server, you can also deploy to IPFS
- Recent version will show, plots and audio buffer for each wake word which model infered for, click on wake word button to know what buffer was infered for that word.



Demo

- For live demo please refer this link
- Allow microphone to capture audio
- Model is trained on `hey fourth brain` - once those words are detected is sequence, for each detected wake word, a play button to listen to what sound was used to detect that word, and what mel spectrograms are used will be listed.

Slides

Please use this link for slides

Conclusion

In this project, we have went through how to extract audio features from audio and train model and detect wake words by using end to end example with source code. Go through wake_word_detection.ipynb jupyter notebook for complete walk through of this project.

Enhancements

- Explore different number of mels, in this project we used 40 as number of mels, we can use different number to see whether this will improve accuracy or not, this can be in range of 32 to 128.

- Use RNN or LSTM or GRU or attention to see whether we can get better results
- Check by computing MFCCs (which is computed after Mel spectrograms) and see if we see any improvements.
- Use different audio augmentation methods like TimeStrech, TimeMasking, FrequencyMasking