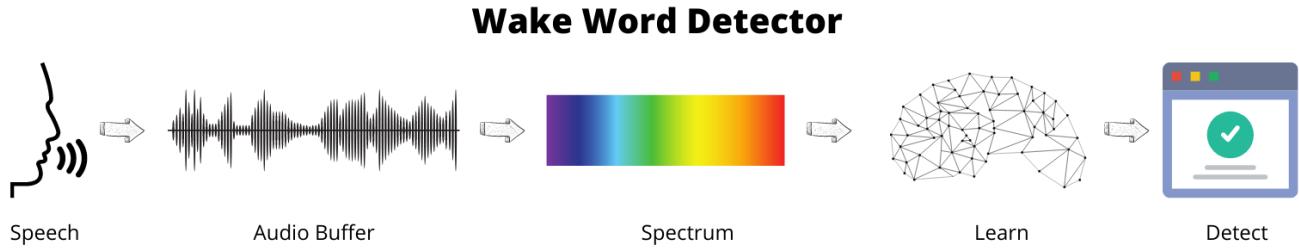


# Wake Word Detector

---



## Table of Contents

---

- [Background](#)
- [Introduction](#)
- [Related Work](#)
- [Implementation](#)
  - [Preparing labelled dataset](#)
  - [Word Alignment](#)
  - [Fix data imbalance](#)
  - [Extract audio features](#)
  - [Audio transformations](#)
  - [Define model architecture](#)
  - [Train model](#)
  - [Test Model](#)
  - [Inference](#)
    - [Using Pyaudio](#)
    - [Using web sockets](#)
    - [Using onnx](#)
    - [Using tensorflowjs](#)
    - [Using tflite](#)
- [Demo](#)
- [Slides](#)
- [Conclusion](#)
- [Enhancements](#)

## 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.

## Related Work

---

- Firefox Voice
  - Model was trained using Mozilla Common Voice dataset, used Pytorch (refer paper [Howl](#)) library to extract audio features and to train model on res8. Custom logic MeydaMelSpectrogram was used to train the model.
  - Used [Meyda: an audio feature extraction library for the Web Audio API](#) for audio feature extraction at client side. Mel-frequency cepstral coefficients (MFCCs) is extracted from audio stream.
  - Used [Honkling](#) (Purely written in Javascript) to do inference on model created using TensorFlow.js and copied above Pytorch model weights to the model created in tensorflow js.
- This project
  - Model was trained using MCV dataset and generated data using Google Speech to Text. Used Pytorch library to extract audio features and to train model on 2 layer CNN. Used Log MelSpectrogram to train the model.
  - Server side inference - Used websockets to stream audio from browser to backend and did inference on that model.
  - Client side inference
    - Used [magenta-js](#) for audio feature extraction, Log MelSpectrograms are extracted from audio stream.
    - Converted Pytorch Model to [Open Neural Network Exchange \(ONNX\)](#) model, used [microsoft/onnxjs](#) to do inference on onnx model at client side.
    - Converted ONNX model to tensorflow model, used [tensorflow js](#) to do inference on tensorflow model
    - Converted tensorflow model to tflite model, used [tflite js](#) to do inference on tensorflow lite model.

## Implementation

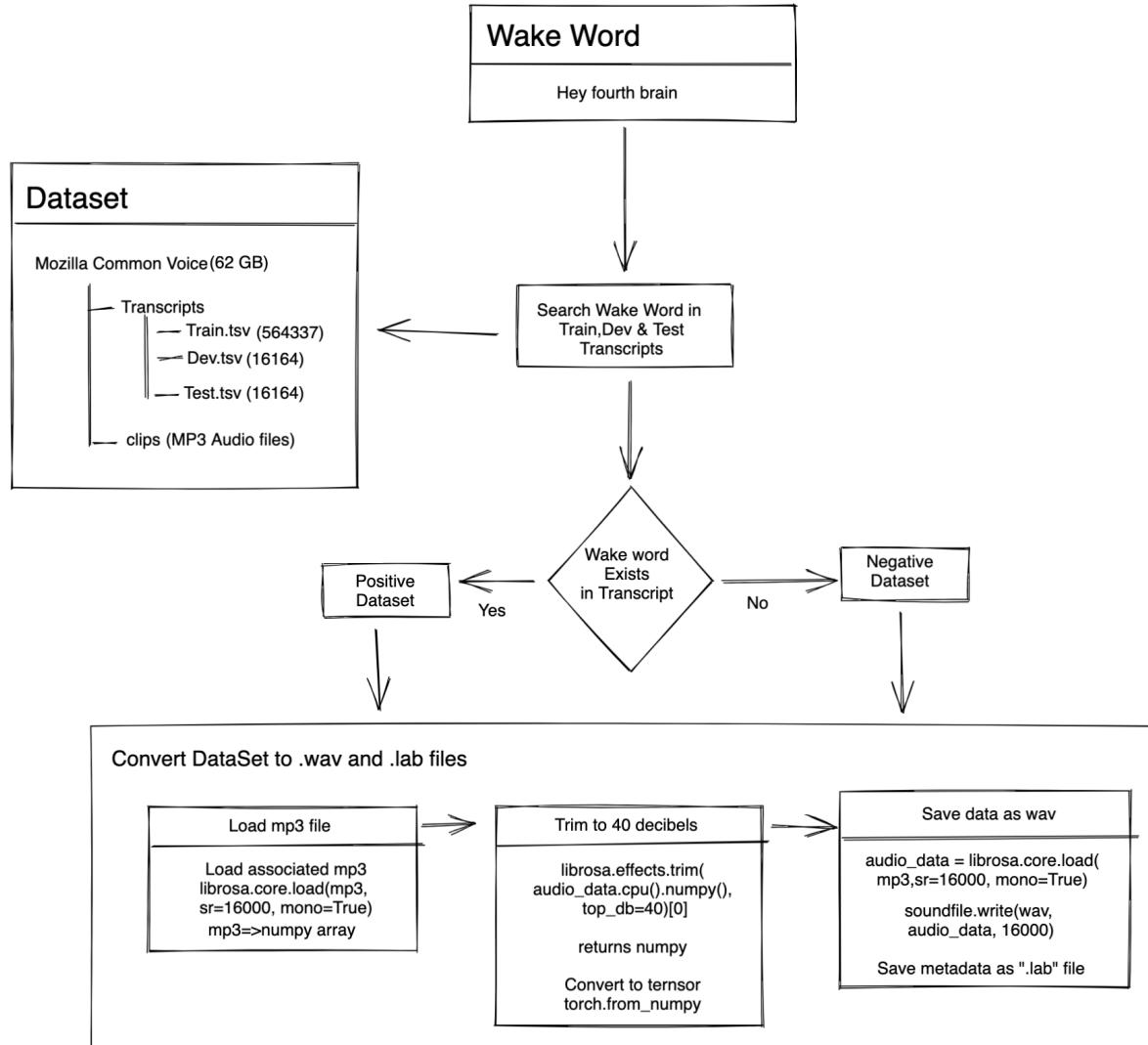
---

### 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

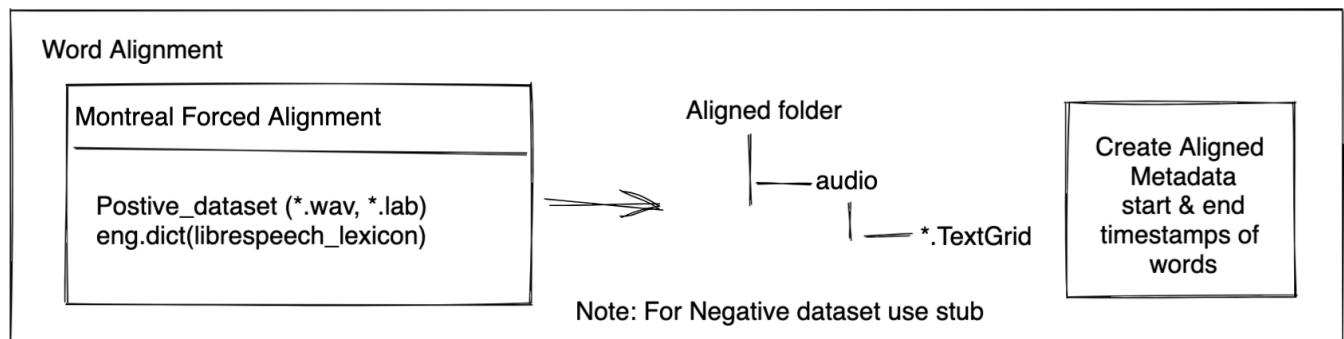
```

- Known issues in MFA

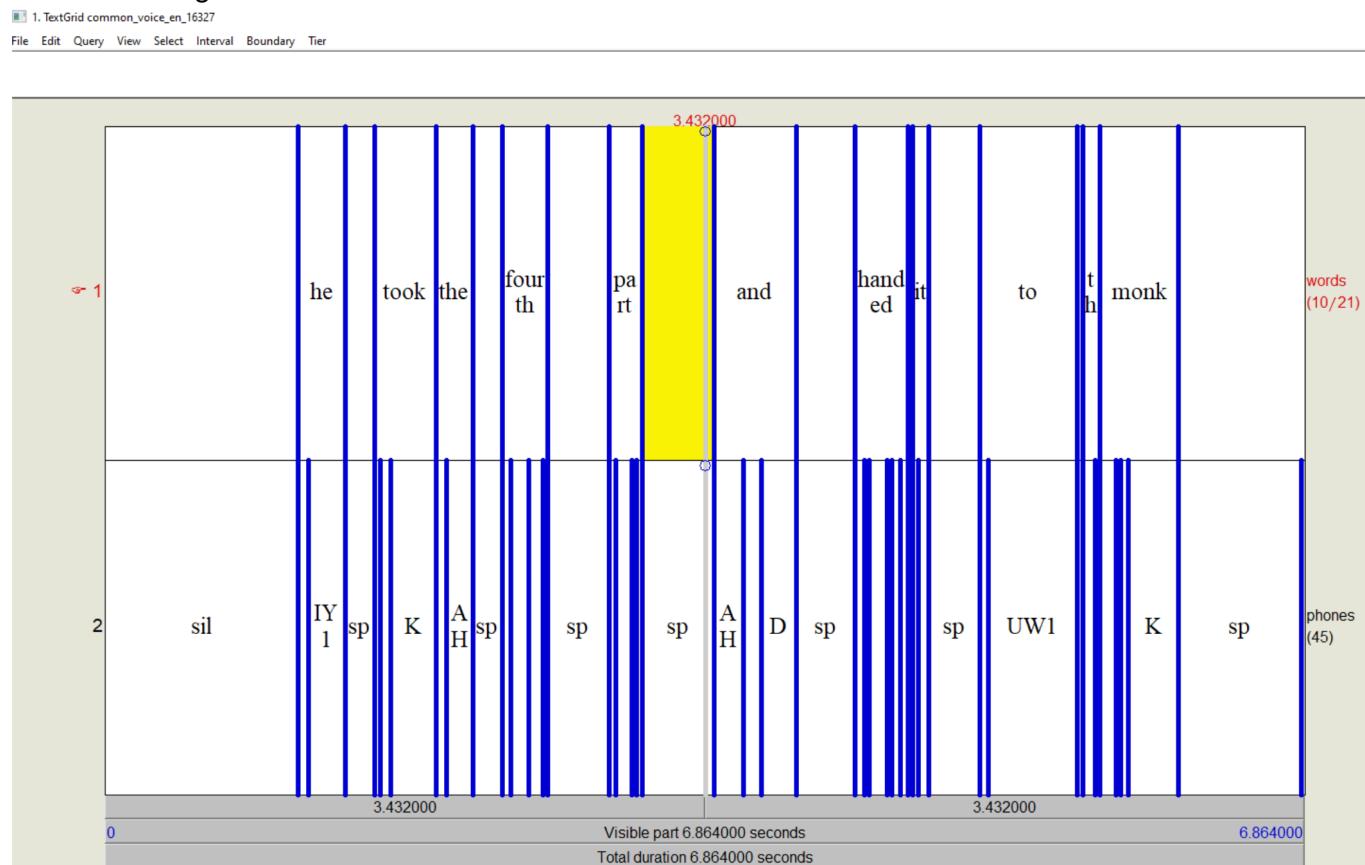
```
# known mfa issue https://github.com/MontrealCorpusTools/Montreal-
Forced-Aligner/issues/109
cp montreal-forced-aligner/lib/libpython3.6m.so.1.0 montreal-forced-
aligner/lib/libpython3.6m.so
cd montreal-forced-aligner/lib/thirdparty/bin && rm libopenblas.so.0
&& ln -s ../../libopenblas-p0-8dca6697.3.0.dev.so 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.

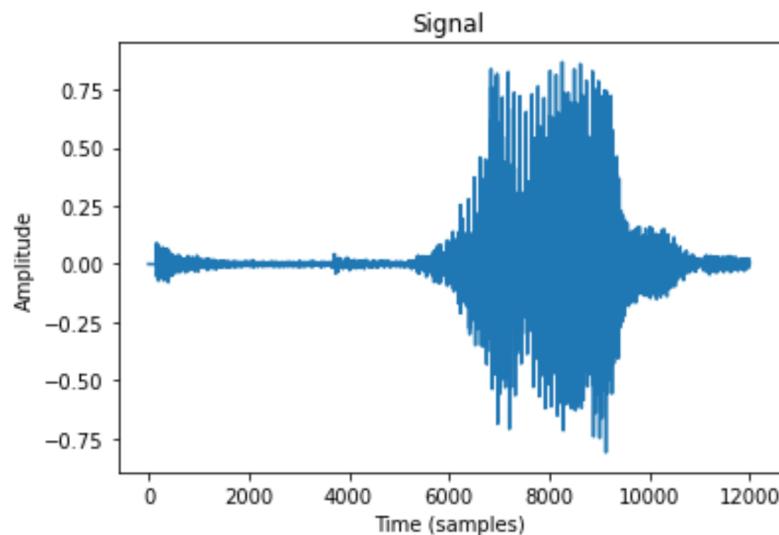
- Used Google Text To Speech Api, set environment variable `GOOGLE_APPLICATION_CREDENTIALS` with your key.
- Used various speed rates, pitches and voices to generate data for wake words.
- Code [generate\\_dataset\\_google\\_tts.py](#)

## 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]

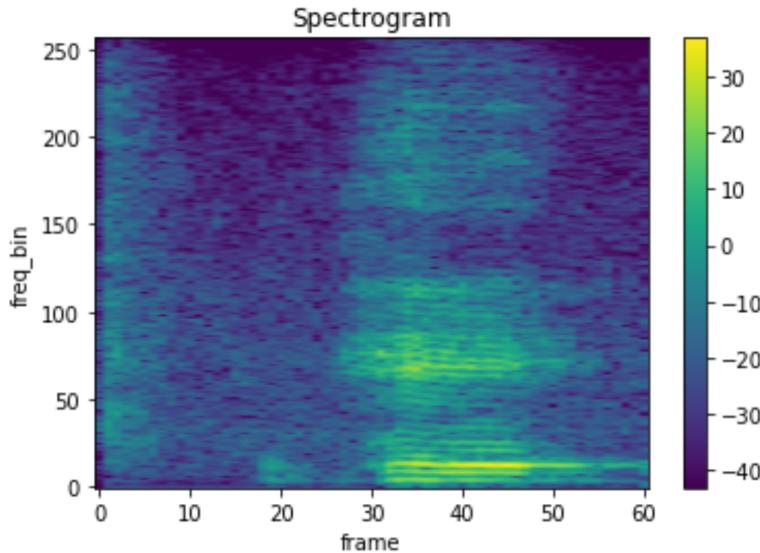
# plotting the signal in time series
plt.plot(sounddata)
plt.title('Signal')
plt.xlabel('Time (samples)')
plt.ylabel('Amplitude')
```



- 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)

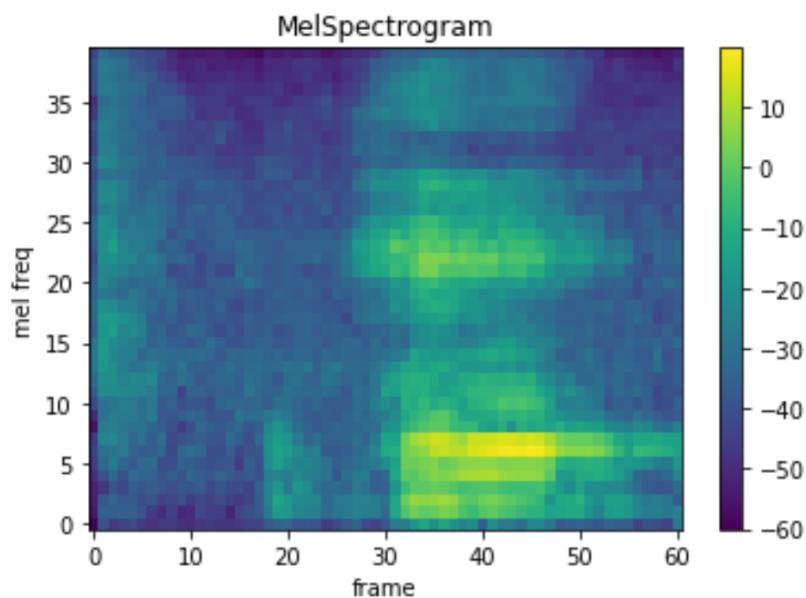
inp = torch.from_numpy(sounddata).float().to(device)
hey_spectrogram = spectrogram(inp.float())
plot_spectrogram(hey_spectrogram.cpu(), title="Spectrogram")
```



- 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")

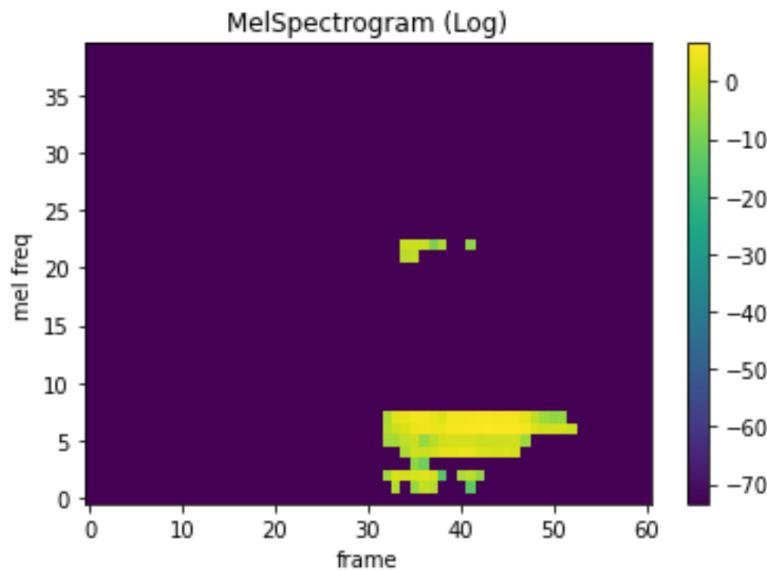
mel_spectrogram.to(device)
inp = torch.from_numpy(sounddata).float().to(device)
hey_mels_slaney = mel_spectrogram(inp.float())
plot_spectrogram(hey_mels_slaney.cpu(), title="MelSpectrogram",
                  ylabel='mel freq')
```



- 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

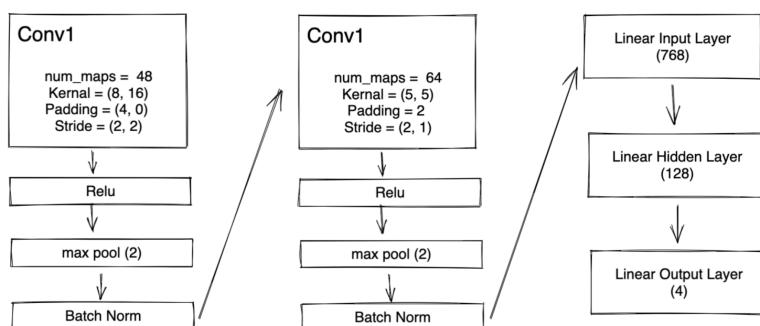
- Used [MelSpectrogram](#) from Pytorch audio to generate mel spectrogram
- Hyperparameters

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

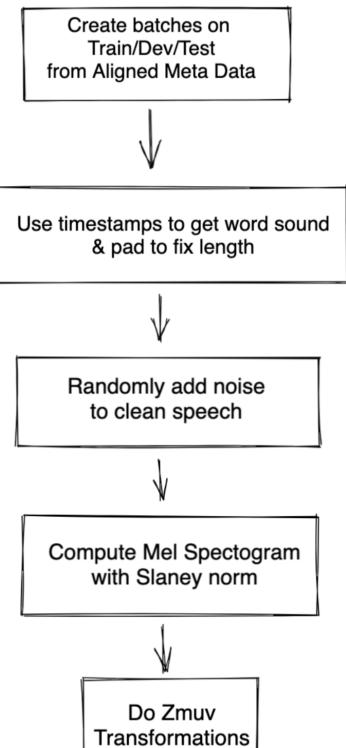
- Used Zero Mean Unit Variance to scale the values
- Code [transformers.py](#) and [audio\\_collator.py](#)

## 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](#)



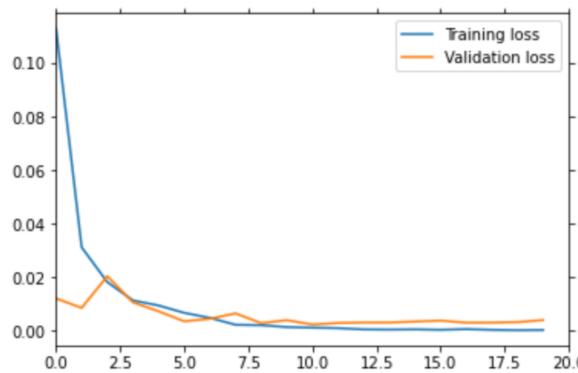
- Below is the CNN model summary

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

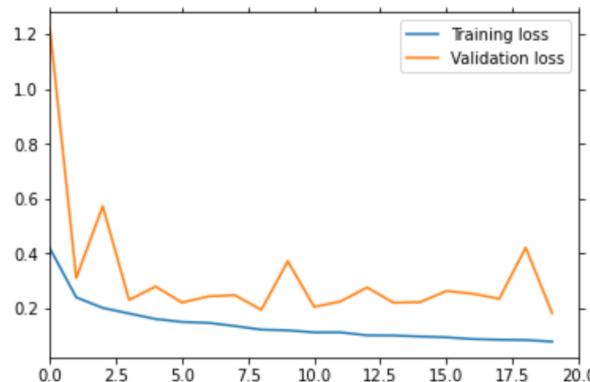
Layer (type)	Output Shape	Param #
<hr/>		
Conv2d-1	[ -1, 48, 31, 13]	6,192
ReLU-2	[ -1, 48, 31, 13]	0
MaxPool2d-3	[ -1, 48, 15, 6]	0
MaxPool2d-4	[ -1, 48, 15, 6]	0
BatchNorm2d-5	[ -1, 48, 15, 6]	96
Conv2d-6	[ -1, 64, 8, 6]	76,864
ReLU-7	[ -1, 64, 8, 6]	0
MaxPool2d-8	[ -1, 64, 4, 3]	0
MaxPool2d-9	[ -1, 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
<hr/>		
Total params: 182,228		
Trainable params: 182,228		
Non-trainable params: 0		
<hr/>		
Input size (MB): 0.01		
Forward/backward pass size (MB): 0.46		
Params size (MB): 0.70		
Estimated Total Size (MB): 1.17		
<hr/>		

## Train model

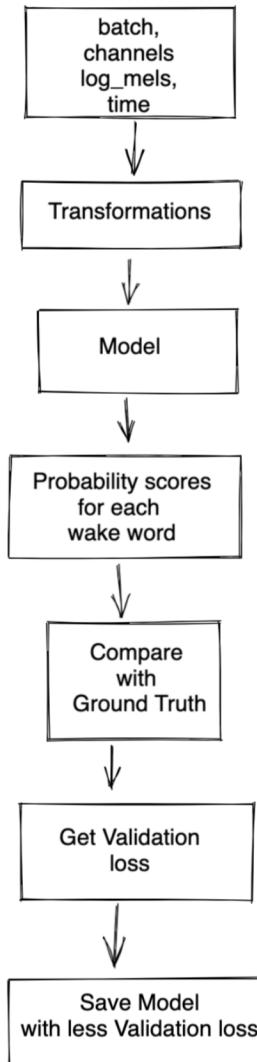
- Used batch size as 16, Tensor of size [16, 1, 40, 61] will be fed to Model
- Used 20 epochs, below is how the train vs validation loss looks like without noise



- 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

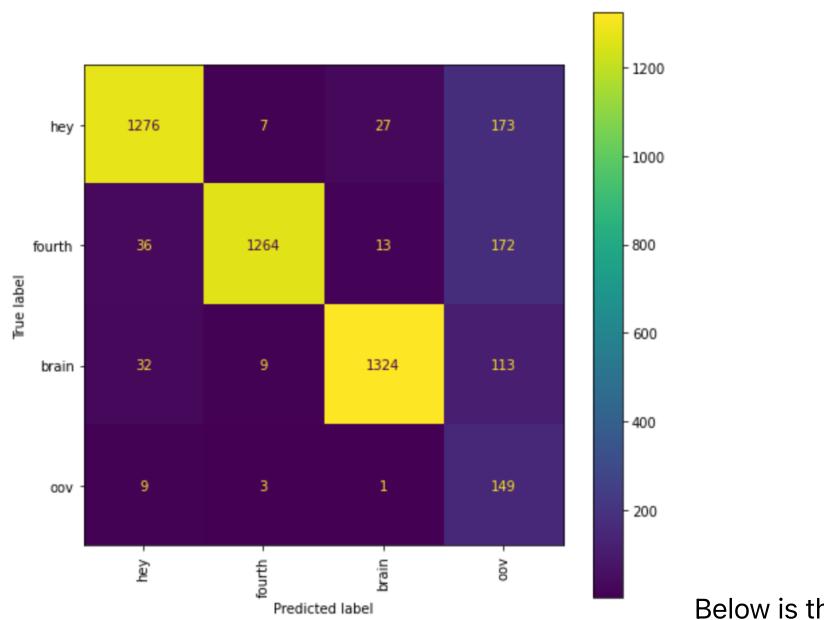
Below is how model performed on test dataset, model achieved 87% accuracy

	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

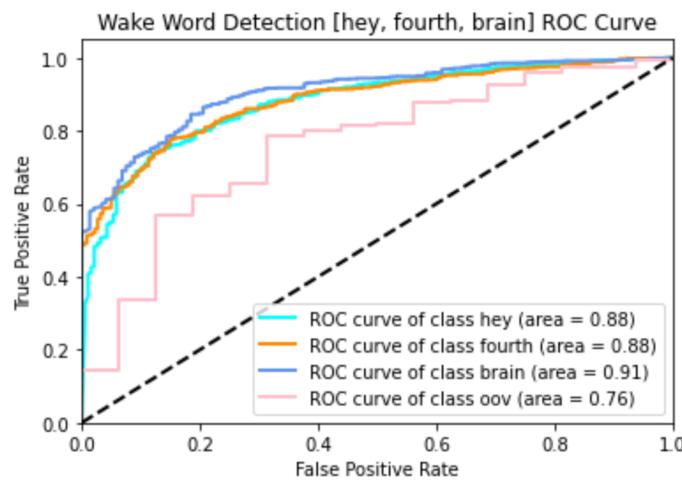
Test Loss: 0.606232

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)



Below is the ROC curve



## Inference

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

### Using Pyaudio

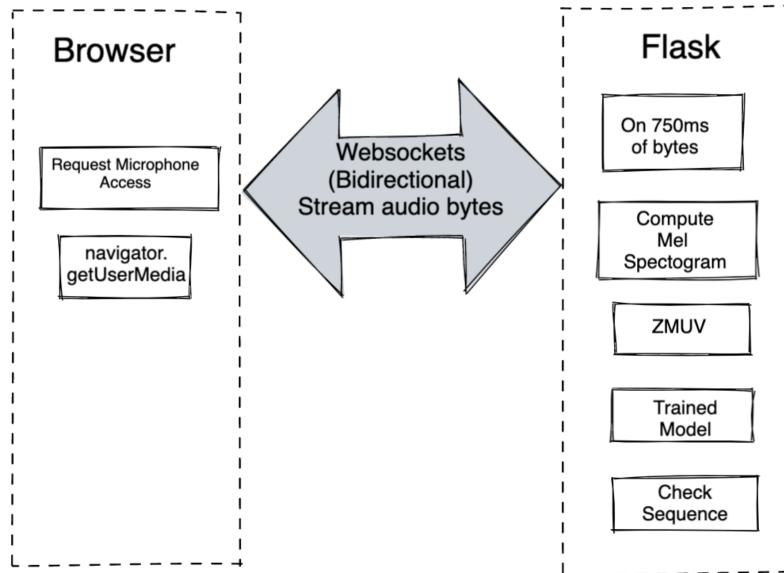
- Used [Pyaudio](#), to get input from microphone
- Capture 750ms window of audio buffer
- After n batches, do transformations and infer on model
- Code - [infer.py](#)
  - \* listening ..

```
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

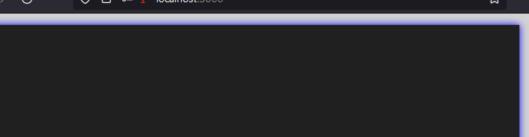
- Used [Flask Socketio](#) at server level to capture audio buffer from client.
- At Client, used [socket.io](#) at client level to send audio buffer through socket connection.
- Capture audio buffer using [getUserMedia](#), convert to array buffer and stream to server.
- Inference will happen at server, 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
```

```
127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9EGKs1d=11947"
127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9EVGs1d=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9EZGs1d=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9EuGs1d=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9F7Ts1d=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9FDGs1d=12888
predicted label 0 - hey
predicted label 0 - hey
predicted label 1 - fourth
127.0.0.1 -- [25/Aug/2021 17:55:48] "GET /socket.io/?EIO=4&transport=polling&t=Nj_z8UT&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9FL&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9Fr&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9G3&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9G8&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9GX&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9Gc&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9Gr&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9Gv&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9H6&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9H9&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9HD&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9HT&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9Hv&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9Ho&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9I5&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9T9&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9Tm&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9TT&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9Tk&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9Tp&sid=127.0.0.1 -- [25/Aug/2021 17:55:48] "POST /socket.io/?EIO=4&transport=polling&t=Nj_z9B8&sid=
```



Listening for wake words ...  
hey  
fourth  
brain  
Wake word hey fourth brain detected



- 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

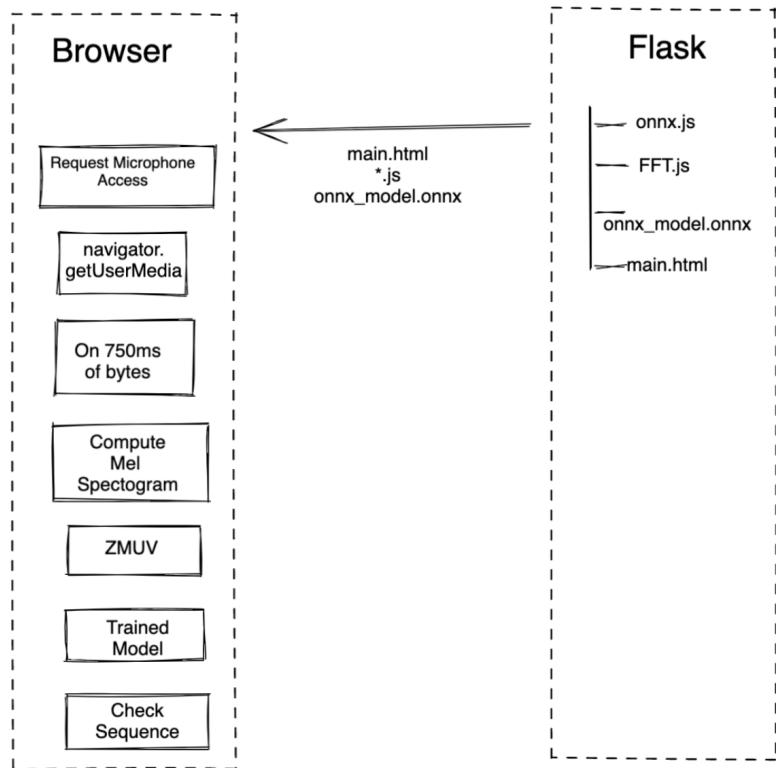
```
eb create wakebot-app --instance_type t2.large --max-instances 1
```

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

## Using ONNX

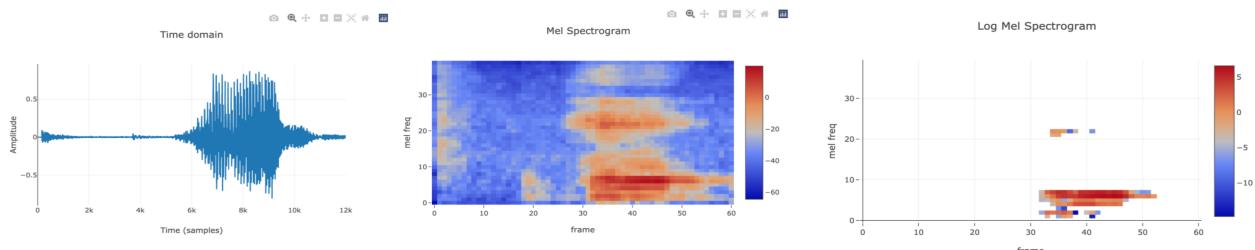
- Used Pytorch onnx to convert pytorch model to onnx model
  - Pytorch to onnx convert code - [convert\\_to\\_onnx.py](#)
  - Once converted, onnx model can be used at client side to do inference
  - Client side, used [onnx.js](#) to do inference at client level
  - Capture audio buffer at client using [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

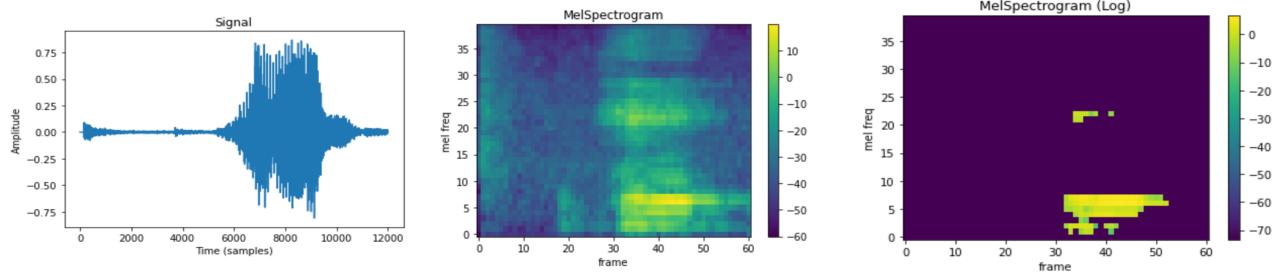


- Below is the comparison of client side vs server side audio transformations

#### 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
```

(.venv) (base) C:\Users\rajas\Documents\workspace\TriggerWakeWordDetection\standalone>set FLASK\_APP=app.py  
(.venv) (base) C:\Users\rajas\Documents\workspace\TriggerWakeWordDetection\standalone>flask run  
\* Serving Flask app 'app.py' (lazy loading)  
\* Environment: development  
\* Debug mode: on  
\* Restarting with stat  
\* Debugger is active!  
\* Debugger PIN 142-552-417  
\* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)  
127.0.0.1 - - [25/Aug/2021 17:59:59] "GET /socket.io/?EIO=4&transport=polling&t=Nj\_-8Xa HTTP/1.1" 404 -  
127.0.0.1 - - [25/Aug/2021 18:00:04] "GET /socket.io/?EIO=4&transport=polling&t=Nj\_-9lr HTTP/1.1" 404 -  
127.0.0.1 - - [25/Aug/2021 18:00:07] "GET / HTTP/1.1" 200 -  
127.0.0.1 - - [25/Aug/2021 18:00:07] "GET /static/audio/audio\_utils.js HTTP/1.1" 200 -  
127.0.0.1 - - [25/Aug/2021 18:00:07] "GET /static/audio/main.js HTTP/1.1" 304 -  
127.0.0.1 - - [25/Aug/2021 18:00:07] "GET /static/audio/styles.css HTTP/1.1" 200 -  
127.0.0.1 - - [25/Aug/2021 18:00:07] "GET /static/audio/rfft.js HTTP/1.1" 200 -  
127.0.0.1 - - [25/Aug/2021 18:00:07] "GET /static/audio/mic128.png HTTP/1.1" 200 -  
127.0.0.1 - - [25/Aug/2021 18:00:07] "GET /static/audio/onnx\_model.onnx HTTP/1.1" 200 -  
127.0.0.1 - - [25/Aug/2021 18:00:14] "GET /favicon.ico HTTP/1.1" 404 -  
127.0.0.1 - - [25/Aug/2021 18:00:35] "GET /static/audio/prompt.mp3 HTTP/1.1" 200 -

Listening for wake words ...  
hey  
fourth  
brain  
Wake word detected - hey fourth brain

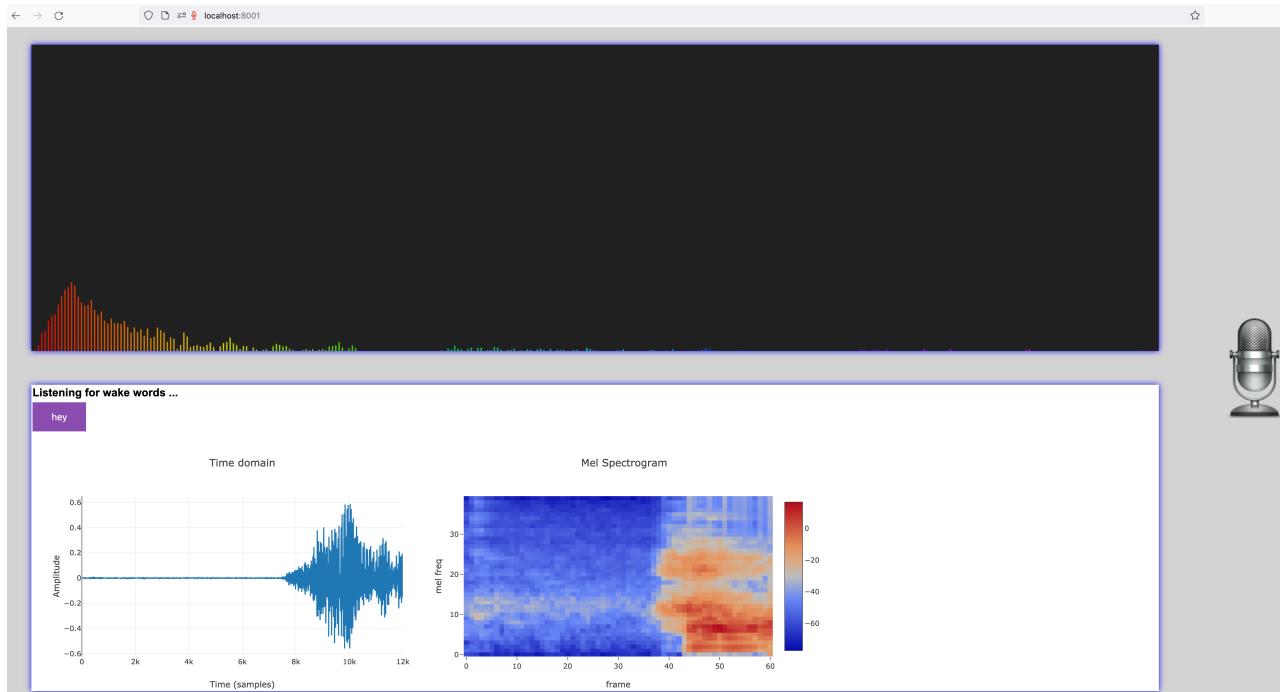
- To deploy to AWS Elastic Beanstalk, first initialize app

```
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\\_onnx](#) 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 inferred for, click on wake word button to know what buffer was inferred for that word.



## Using tensorflowjs

- Used [onnx-tensorflow](#) to convert onnx model to tensorflow model
- onnx to tensorflow convert code - [convert\\_onnx\\_to\\_tf.py](#)

```
onnx_model = onnx.load("onnx_model.onnx") # load onnx model
tf_rep = prepare(onnx_model) # prepare tf representation

# Input nodes to the model
print("inputs:", tf_rep.inputs)

# Output nodes from the model
print("outputs:", tf_rep.outputs)

# All nodes in the model
print("tensor_dict:")
print(tf_rep.tensor_dict)

tf_rep.export_graph("hey_fourth_brain") # export the model
```

- Verify the model using below command

```
python .venv/lib/python3.8/site-
packages/tensorflow/python/tools/saved_model_cli.py show --dir
hey_fourth_brain --all
```

#### Output

```
MetaGraphDef with tag-set: 'serve' contains the following
SignatureDefs:

signature_def['__saved_model_init_op']:
The given SavedModel SignatureDef contains the following input(s):
The given SavedModel SignatureDef contains the following output(s):
  outputs['__saved_model_init_op'] tensor_info:
    dtype: DT_INVALID
    shape: unknown_rank
    name: NoOp
Method name is:

signature_def['serving_default']:
The given SavedModel SignatureDef contains the following input(s):
  inputs['input'] tensor_info:
    dtype: DT_FLOAT
    shape: (-1, 1, 40, 61)
    name: serving_default_input:0
The given SavedModel SignatureDef contains the following output(s):
  outputs['output'] tensor_info:
    dtype: DT_FLOAT
    shape: (-1, 4)
    name: PartitionedCall:0
Method name is: tensorflow/serving/predict
```

```

Defined Functions:
Function Name: '__call__'
    Named Argument #1
        input

Function Name: 'gen_tensor_dict'
```

- Refer [onnx\\_to\\_tf](#) for generated files
- Test converted model using [test\\_tf.py](#)
- Used [tensorflowjs\[wizard\]](#) to convert [savedModel1](#) to web model

```

(.venv) (base) → onnx_to_tf git:(main) ✘ tensorflowjs_wizard
Welcome to TensorFlow.js Converter.
? Please provide the path of model file or the directory that contains
model files.
If you are converting TFHub module please provide the URL.
hey_fourth_brain
? What is your input model format? (auto-detected format is marked
with *) Tensorflow Saved Model *
? What is tags for the saved model? serve
? What is signature name of the model? signature name:
serving_default
? Do you want to compress the model? (this will decrease the model
precision.) No compression (Higher accuracy)
? Please enter shard size (in bytes) of the weight files? 4194304
? Do you want to skip op validation?
This will allow conversion of unsupported ops,
you can implement them as custom ops in tfjs-converter. No
? Do you want to strip debug ops?
This will improve model execution performance. Yes
? Do you want to enable Control Flow V2 ops?
This will improve branch and loop execution performance. Yes
? Do you want to provide metadata?
Provide your own metadata in the form:
metadata_key:path/metadata.json
Separate multiple metadata by comma.
? Which directory do you want to save the converted model in?
web_model
converter command generated:
tensorflowjs_converter --control_flow_v2=True --
input_format=tf_saved_model --metadata= --saved_model_tags=serve --
signature_name=serving_default --strip_debug_ops=True --
weight_shard_size_bytes=4194304 hey_fourth_brain web_model

...
File(s) generated by conversion:
Filename                      Size(bytes)
group1-shard1of1.bin          729244
model.json                    28812
Total size:                  758056
```

- Once above step is done, copy the files to web application Example -

```

├── index.html
└── static
    └── audio
        ├── audio_utils.js
        ├── fft.js
        ├── main.js
        ├── mic128.png
        ├── model
        │   ├── group1-shard1of1.bin
        │   └── model.json
        ├── prompt.mp3
        └── styles.css

```

- Client side used [tfjs](#) to load model and do inference
- Loading the tensorflow model

```

let tfModel;
async function loadModel() {
  tfModel = await
  tf.loadGraphModel('static/audio/model/model.json');
}
loadModel()

```

- Do inference using above model

```

let outputTensor = tf.tidy(() => {
  let inputTensor = tf.tensor(dataProcessed, [batch, 1, MEL_SPEC_BINS,
  dataProcessed.length/(batch * MEL_SPEC_BINS)], 'float32');
  let outputTensor = tfModel.predict(inputTensor);
  return outputTensor
});
let outputData = await outputTensor.data();

```

## Using tflite

- Once tensorflow model is created, it can be converted to tflite, using below code

```

model = tf.saved_model.load("hey_fourth_brain")
input_shape = [1, 1, 40, 61]
func =
tf.function(model).get_concrete_function(input=tf.TensorSpec(shape=inp
ut_shape, dtype=np.float32, name="input"))
converter = tf.lite.TFLiteConverter.from_concrete_functions([func])

```

```
tflite_model = converter.convert()
open("hey_fourth_brain.tflite", "wb").write(tflite_model)
```

- Note: `tf.lite.TFLiteConverter.from_saved_model("hey_fourth_brain")` did not work, as it was throwing `conv.cc:349 input->dims->data[3] != filter->dims->data[3] (0 != 1)` on inference, so used above method.
- copy the tflite model to web application
- Used `tflite js` to load model and do inference
- Loading tflite model

```
let tfliteModel;
async function loadModel() {
    tfliteModel = await
tflite.loadTFLiteModel('static/audio/hey_fourth_brain.tflite');
}
loadModel()
```

## Demo

---

- For live demo
  - ONNX version -<https://wake-onnx.rajashekhar.blog>
  - Tensorflow js version - <https://wake-tf.rajashekhar.blog>
  - Tensorflow lite js version - <https://wake-tflite.rajashekhar.blog>
- Allow microphone to capture audio
- Model is trained on `hey fourth brain` - once those words are detected in 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 gone 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](#)