CS5590 Python and Deep Learning - Project

Spring 2019, UMKC

Team Members

- Ruthvic Punyamurtula
- Sai Charan Kotthapalli

We have implemented to a light weight wake word detection model which can be used to perform detection on low power and low computational devices like raspberry pi. We have achieved high accuracy (98%) with training data about 4000 samples of a single speaker.
Also we have obtained an accuracy of 90% with training data of just 400 samples with the same architecture when trained on a single speaker voice. In the following sections, we explain in depth about our project.

‘Recept’ - Wake word Detection

We are going to construct a audio speech dataset and implement an algorithm for Wake word detection which also called as keyword detection, or wakeword detection. Trigger word detection is the technology that allows devices like Amazon Alexa, Google Home, Apple

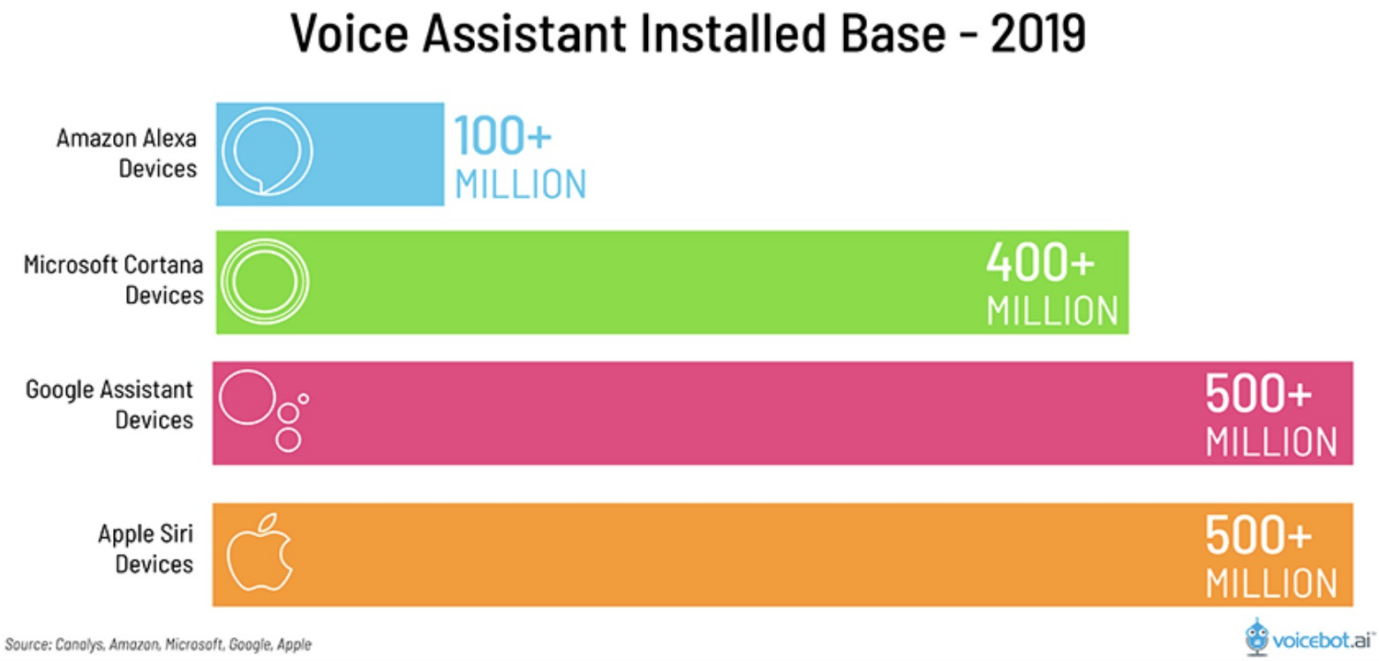
Siri, etc to wake up, turn on upon hearing a certain word.

Our trigger word will be “Activate.” so every time you say “activate,” the device is going to make a “chiming” sound. We will also be able to record a clip of yourself talking in the audio microphone, and have the algorithm trigger a chiming sound when it detects saying “activate.”

In this project we are going to acheive

- Structure a speech recognition project
- Synthesize and process audio recordings to create train/dev datasets
- Train a trigger word detection model and make predictions

Motivation

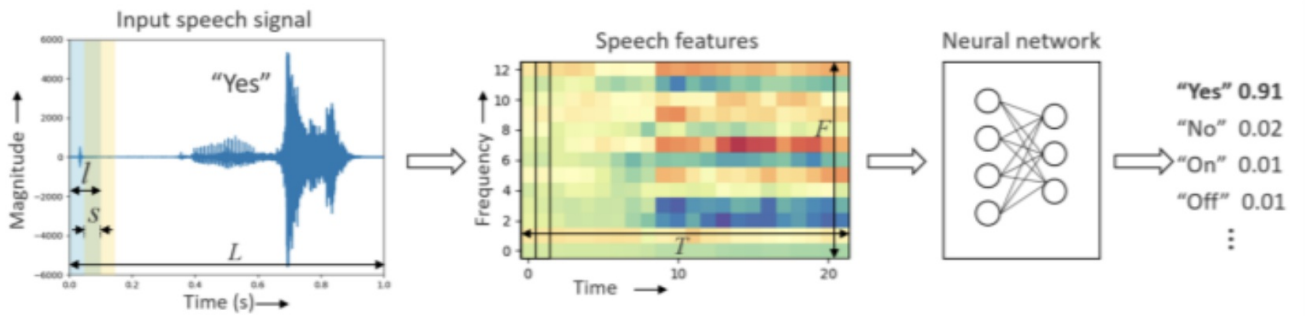


Amazon Alexa, Google Voice Assistant, Apple Siri are biult on high level GPU’s and trained on over 250 million hours of audio recordings, which

is not open source and non Customisable

We have biult the trigger word dettection on a light weight keras based model which is open source and can be easily deployed on a embedded device.

Hello Edge: Keyword Spotting on Microcontrollers



NN model	S(80KB, 6MOps)			M(200KB, 20MOps)			L(500KB, 80MOps)		
	Acc.	Mem.	Ops	Acc.	Mem.	Ops	Acc.	Mem.	Ops
DNN	84.6%	80.0KB	158.8K	86.4%	199.4KB	397.0K	86.7%	496.6KB	990.2K
CNN	91.6%	79.0KB	5.0M	92.2%	199.4KB	17.3M	92.7%	497.8KB	25.3M
Basic LSTM	92.0%	63.3KB	5.9M	93.0%	196.5KB	18.9M	93.4%	494.5KB	47.9M
LSTM	92.9%	79.5KB	3.9M	93.9%	198.6KB	19.2M	94.8%	498.8KB	48.4M
GRU	93.5%	78.8KB	3.8M	94.2%	200.0KB	19.2M	94.7%	499.7KB	48.4M
CRNN	94.0%	79.7KB	3.0M	94.4%	199.8KB	7.6M	95.0%	499.5KB	19.3M
DS-CNN	94.4%	38.6KB	5.4M	94.9%	189.2KB	19.8M	95.4%	497.6KB	56.9M

	Open Source	**Language**	**Model Trainer**	**License**
Precise	Yes	Python	Open source script	Permitted
Snowboy	No	Node, Java, Python, GO, Perl, iOS, Android	Web API	License Free
Porcupine	No	C, Python	Closed Binary	License Free
PocketSphinx	Yes	Any Linux based	Open source script	Permitted
(Recept) Our Approach	Yes	Python	Open source script	License Free

Project Idea

Data Preprocessing : Creating a speech Dataset

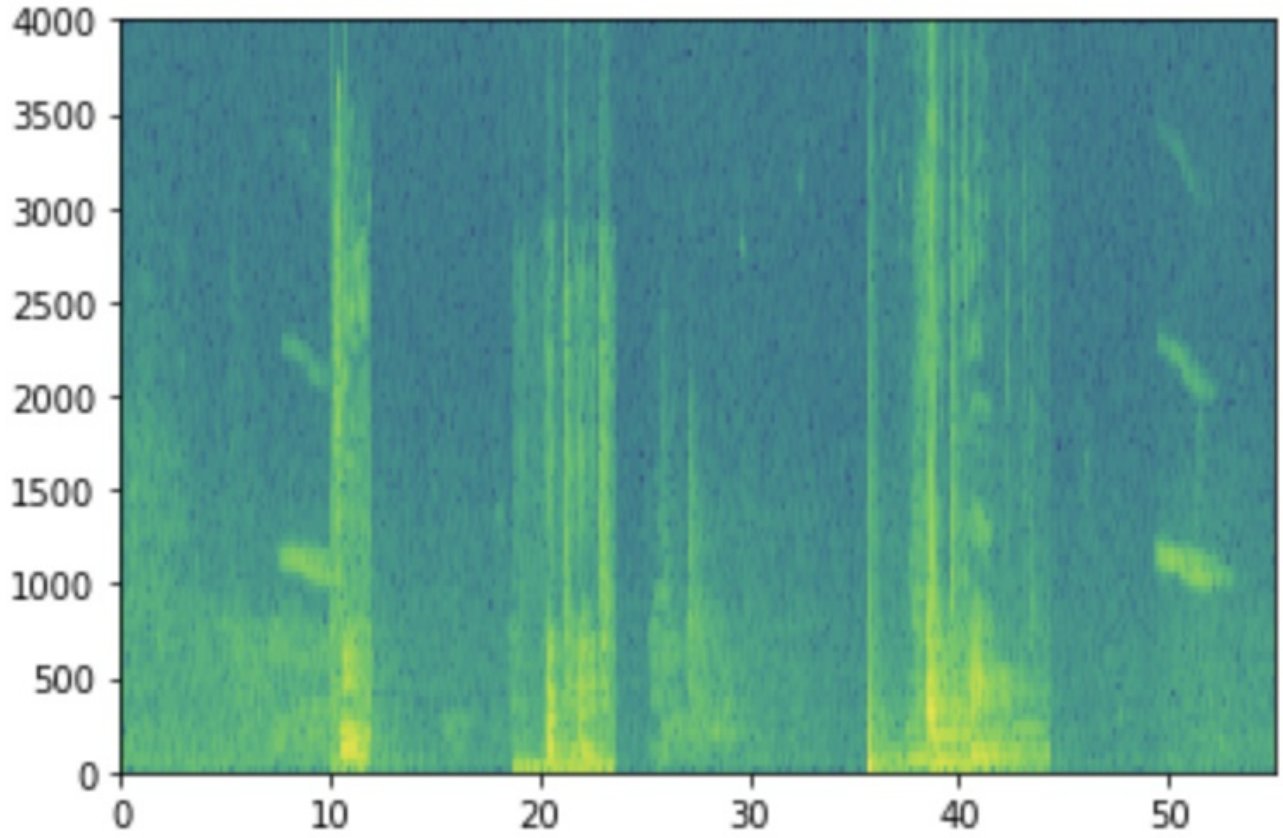
Start by building a dataset for your trigger word detection algorithm, We need to create recordings with a combination of positive words (in our case “activate”) and negative words (random words other than “activate”) on different background sounds.

We started by recording 10 second audio clips with varying accents containing the trigger word of about 4000 samples.

Audio Recordings to Spectograms

In order to help our sequence model more easily learn to detect triggerwords, we will need to compute a spectrogram of the audio. The spectrogram tells us how much different frequencies are present in an audio clip at a moment in time.

A spectrogram is computed by sliding a window over the raw audio signal, and calculates the most active frequencies in each window using a Fourier transform.



The graph above represents how active each frequency is (y axis) over a number of time-steps (x axis).

Generating a training Sample

With 10 seconds being our default training example length, 10 seconds of time can be discretized to different numbers of value. We have observed 441000 (raw audio) and 5511 (spectrogram). In the former case, each step represents 10/441000 approx 0.000023 seconds. In the second case, each step represents 10/5511 approx 0.0018 seconds.

For the 10sec of audio, the key values you will see in this assignment are:

$441000 raw audio

$5511 = T_x which is the spectrogram output, and dimension of input to the neural network
10000 used by the pydub module to synthesize audio
$1375 = T_y which is the number of steps in the output of the GRU you'll build

To obtain a single training example we need:

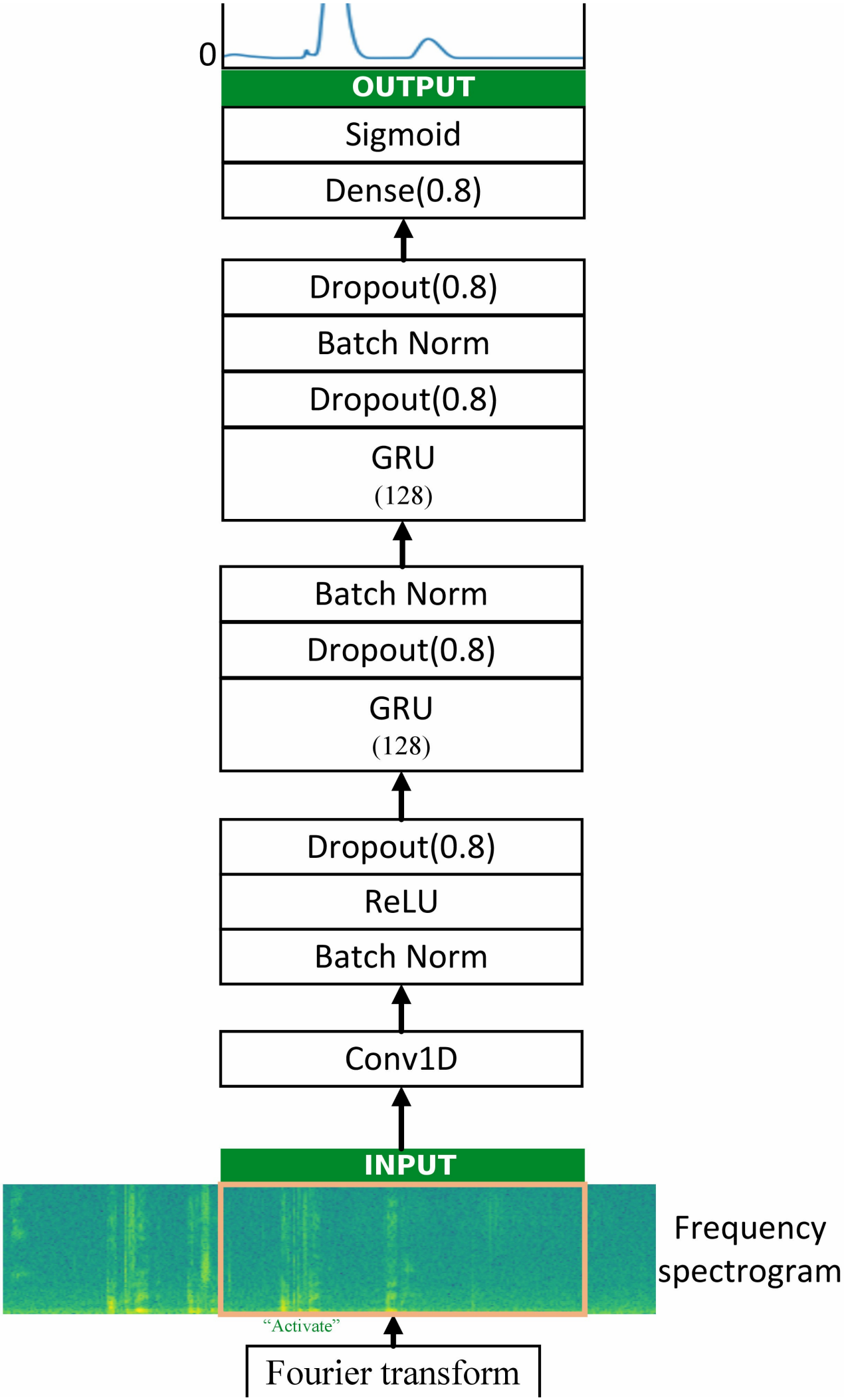
- Pick a random 10 second background audio clip
- Randomly insert 0-4 audio clips of "activate" into this 10sec clip
- Randomly insert 0-2 audio clips of negative words into this 10sec clip

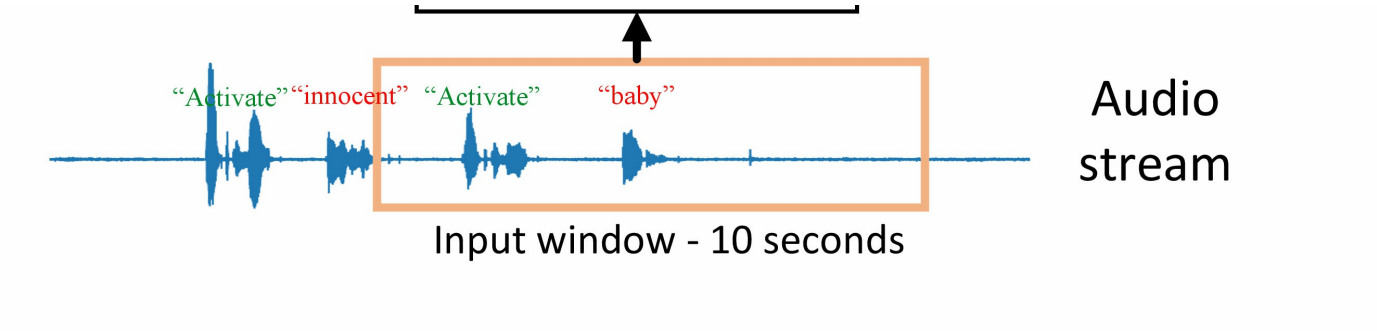
Full training set

After we implement the code needed to generate a single training example. We used this process to generate a large training set. To save time, we've already generated a set of training examples.

Our dataset now consists of 4000 audio samples with 2 positives and 1 negative word for each audio clip.
We one hot encode the positive values with 1's and negative words and background noise with 0's

System Architecture


0
OUTPUT
Sigmoid
Dense(0.8)
Dropout(0.8)
Batch Norm
Dropout(0.8)
GRU
(128)
Batch Norm
Dropout(0.8)
GRU
(128)
Dropout(0.8)
ReLU
Batch Norm
Conv1D
INPUT
Frequency
spectrogram
"Activate"
Fourier transform



The 1D convolutional step inputs 5511 timesteps of the spectrogram 10 seconds audio file in our case, outputs a 1375 step output. It extracts low-level audio features similar to how 2D convolutions extract image features. Also helps speed up the model by reducing the number of timesteps.

The two GRU layers read the sequence of inputs from left to right, then ultimately uses a dense+sigmoid layer to make a prediction. Sigmoid make the range of each label between 0~1. Being 1, corresponding to the user having just said "activate".

Model Summary

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 5511, 101)	0
conv1d_1 (Conv1D)	(None, 1375, 196)	297136
batch_normalization_1 (Batch	(None, 1375, 196)	784
activation_1 (Activation)	(None, 1375, 196)	0
dropout_1 (Dropout)	(None, 1375, 196)	0
gru_1 (GRU)	(None, 1375, 128)	124800
dropout_2 (Dropout)	(None, 1375, 128)	0
batch_normalization_2 (Batch	(None, 1375, 128)	512
gru_2 (GRU)	(None, 1375, 128)	98688
dropout_3 (Dropout)	(None, 1375, 128)	0
batch_normalization_3 (Batch	(None, 1375, 128)	512
dropout_4 (Dropout)	(None, 1375, 128)	0
time_distributed_1 (TimeDist	(None, 1375, 1)	129

Total params: 522,561

Technical Stack

We have used the following technical specs for this project

- Python 3.4 and above
- Pydub package for reading audio files
- Jupyter notebook

  To implement Metrics and plot graphs
- numpy

- keras
- h5py
- pydub
- scipy
- matplotlib

Implementation

The implementation is described in two notebooks. As the Notebooks size was high, we did not combine the implementation in this main report. The notebooks are described below

1. Training your model - https://github.com/Ruthvicp/CS5590_PyDL/blob/master/Project/Do
2. Real-time demo of the wake word recognition - https://github.com/Ruthvicp/CS5590_PyDL/blob/master/Project/Do

Contribution

We have togethor worked in preparing the data and processing it. Also tried different ways of training the model and its testing as well. So take equal contribution in the work

- Ruthvic : 50%
- Charan : 50%

References

- Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas

Chandra. Hello edge: Keyword spotting on microcontrollers. arXiv preprint arXiv:1711.07128, 2017.

- Alice Coucke, Mohammed Chlieh, Thibault Gissel-brecht, David Leroy, Mathieu Poumeyrol, Thibaut Lavril, “Efficient keyword spotting using dilated convolutions and gating”, 2018.
- https://github.com/MycroftAI/mycroft-precise
- https://www.dlology.com/blog/how-to-do-real-time-trigger-word-detection-with-keras/
- https://medium.com/snips-ai/federated-learning-for-wake-word-detection-c8b8c5cdd2c5