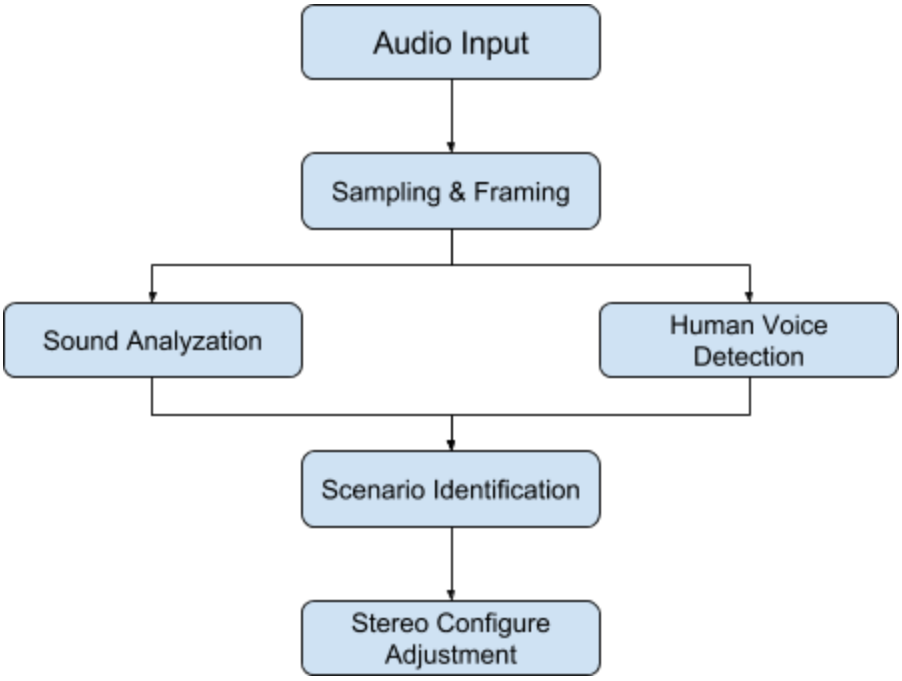
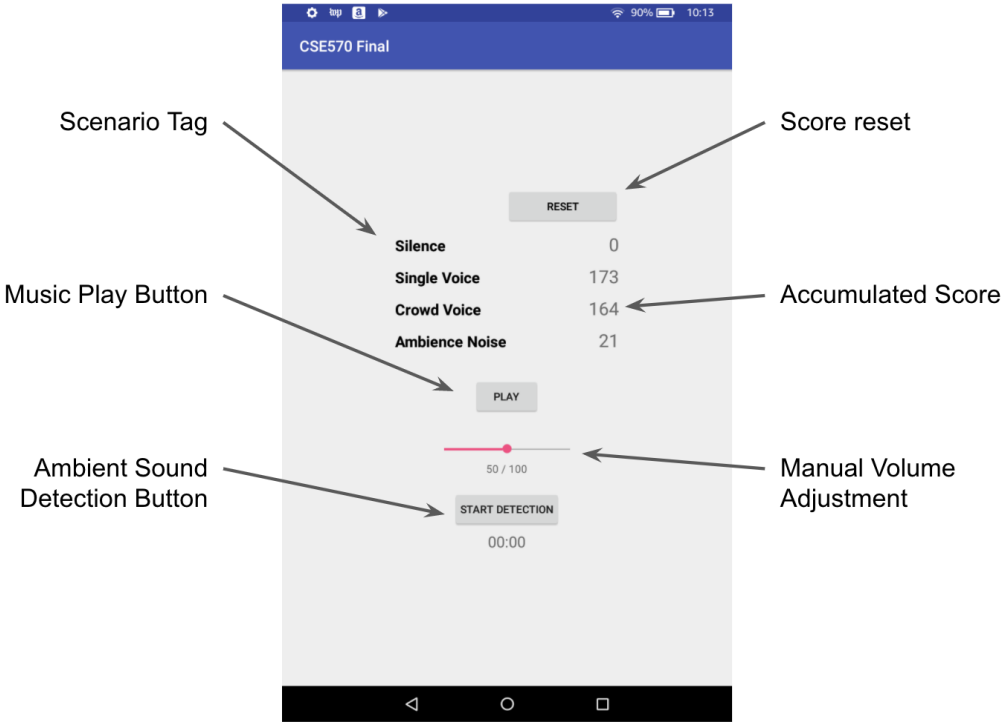
Mechanism

1. Workflow diagram and APP configure





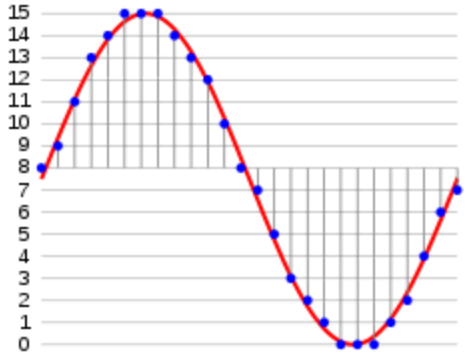
Four labels — “silence”, “single voice”, “crowd voice”, and “ambient noise” are set to represent the four scenarios that are able to describe user’s most common environment. The difference among these four scenarios is basically upon the sound type, therefore the environment types (e.g. indoor or outdoor) is not taken into consideration. Some scenarios may be hard to be identified from others (e.g. crowd voice from single voice). Following are the detailed definition of four labels:

- **Silence**: In default most quiet places are classified as the silence scenario.
- **Single Voice**: This label is for the scenario where only one or two persons are talking around the user. Usually this happens in someplace like the user stays in the study room with friends or listens to a speech in the lecture hall.
- **Crowd Voice**: This label is for the scenario where multiple human voices are occurring continuously around the user. Some cases like a shuttle full with people is a typical example for this scenario.
- **Ambient noise**: This label is for the scenario where there is no human voice but a specific sound that almost never stops disturbing the user. A good example is the NCS commuter lounge room. The vendor machine over there periodically generate noise that is hard to ignore.

2. Collect background sound data in Android devices

The Android media framework provides basic media services, including two major tasks in our app — collecting audio signals from devices’ audio source, usually from microphone, and playing various format media files as well. Here, our app use the AudioRecord APIs to collect background sound and use the MediaPlayer APIs to play music. Like a human being, the AudioRecord module is the ear of an app, which is used to hear surrounding voices from its microphone, and the MediaPlayer moulde is the mouth of an app, which sings a beautiful song.

Since we need to get the raw data of input audio signals for further analysis, we use the AudioRecord API, which manages the audio resources for Java applications to record audio from the audio input hardware of the platform. The collected audio data is encoded in PCM (Pulse-code modulation) format. From the PCM-encoded audio data, we can easily retrieve the amplitude and frequency of the sound.



Pulse-code modulation (PCM) in audio

First of all, we need to acquire users permission before initializing the AudioRecord. We define the uses-permission in AndroidManifest.xml.

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

When initializing, we need to setup parameters of the AudioRecord module, including:

- Audio source: microphone
- Sample Rate: 16000 Hz
- Audio format: PCM 16 bits

For power consumption and system performance concern, we collect and analyze audio source about every 200 millisecond. The reason for choosing such a high frequency is due to the limitation of TensorFlow. If we choose longer duration, e.g. 1000 millisecond, the recognition result from TensorFlow model would be terrible. However, if we choose shorter duration, the recognition module would cost most of the system resource, thus making the system very lag. Also, power consumption is also an issue.

3. Analyze the profile of collected sound data

Once sufficient sound data collected, our app thereafter calculates the average value of sound volume in dB and implements fast-fourier transform (FFT) to obtain the frequency-domain data.

When measuring sound, we use the following logarithmic formula to determine the sound pressure level (SPL) in decibels.

$$SPL = 20 \log_{10}\left(\frac{p}{p_{ref}}\right) \text{dB}$$

The reference sound pressure Pref = 2, which maps to SPL = 100 dB. The sound pressure from PMC encoding equals to the root mean square of every byte. Therefore, the equation we use in our implementation is:

$$db = 100 + 20 \cdot log(\frac{avg}{2})$$

For FFT implementation, we found there is free source code in Java. [3]

The average sound volume is mostly employed as a threshold to sort the collected sound into silence or non-silence classes. The data in frequency-domain can indicate whether the sound comes from single or multiple source. We assume that the sound from single source may mostly distribute in a relatively narrow frequency band when comparing to the frequency distribution of multi-source sound.

4. Human voice detection

This function is executed whenever the sound data is not regarded as silence scenario. At this stage, we will use the third-party deep-learning toolkit, TensorFlow, as our deterministic function to decide whether the sound source is human or not. TensorFlow is an open-source library based on deep neural networks. We leverage its tutorial of human voice commands recognition to develop our detecting function. [1] We assume there should be high-frequency

words in most human daily talks, and usually such words are either single or two syllables like “Yes”, “No”, “Go”, “Up”, etc.. Therefore we use TensorFlow to train a model that can detect these words on the fly. If the model indicates that a sound data contains anyone of the word bank, this data would be regarded as either single voice and multiple voice; otherwise it would be ambient noise.

5. Scoring mechanism

Both the sound profile analyzation and human voice detection functions collaboratively figure out what the sound type is for the sound collected in a constant time. Since the time frame of sound collection is actually very short (less than 1 second), while the detection mechanism is not perfectly reliable, it may annoy the user if APP adjusts the music volume whenever the scenario changes, because it may happen frequently.

To avoid this problem, we set up a scoring mechanism to enhance soundness of the program. Once the collected sound data is marked with one scenario label, APP adds one point to that label. The scenario detection and scoring last for a constant time (e.g. 30 seconds), and when the time is up, APP checks which label gets the highest score and adjust the music volume if the scenario around the user changes. This mechanism ensures that the volume adjustment would not be triggered too frequently, and the misjudgement can be considerably reduced.

6. Utilize our prediction

Based on the prediction, APP will switch the current user mode into its corresponding mode. For example, at first a user stays in a quiet place listening to the music. When the user leaves the bed room and walks on the road, we detect the background noise become larger. Then, the app will increase the volume.

Criteria of Scenario Identification

Below are the classification rules for these scenarios:

- If sound average volume is lower than 40 dB, the scenario at the moment (< 0.2 sec) is regarded as silence environment.
- If sound cannot get enough score in the human voice detection but the volume is large enough, it is the environment with ambient noise.
- If TensorFlow model identifies that sound is full of human voice, and the volume never exceeds the 75dB and is closed to positive skewed distribution, it is regarded as single voice environment.
- If sound is full of human voice while the volume is negative skewed or normalized distribution, it is crowd voice environment.

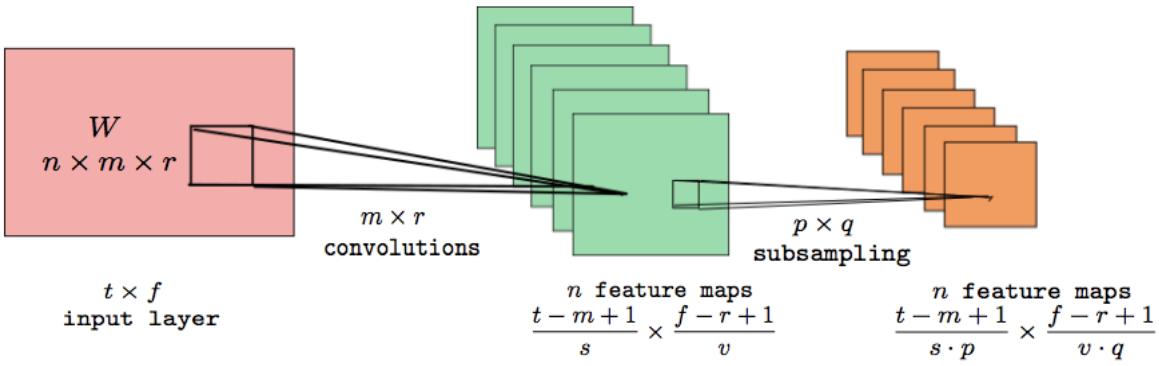
Each time the identification process continuously classifies the collected sound for 1 minutes, and after the time APP adjust the music volume in accordance of the output label. In terms of silence and single voice labels, the music would be turn down; while the crowd voice label would make it louder. The default label is ambience noise.

Principle of Human Voice Detection

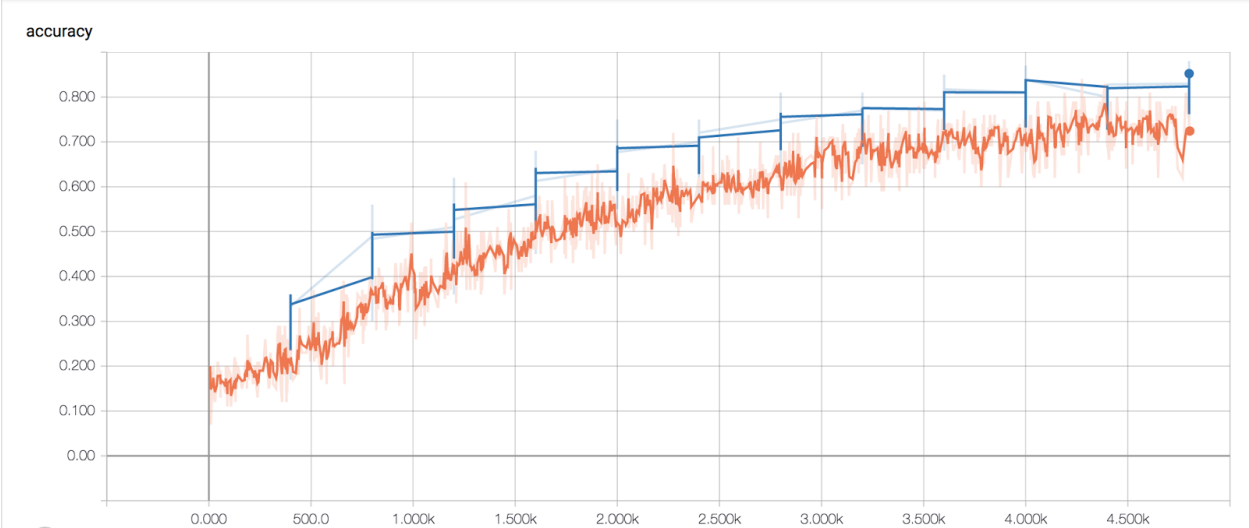
➢ **Deep Learning Implementation**

The mechanism to detect keywords is achieved by means of TensorFlow training tool with multi-layers convolutional neural networks architecture. This is a quick and effective solution that deploying the training task on a dual-core CPU only takes about 5 days, let alone using GPU. The training and verifying datasets are contributed by Google and released under a CC BY license. The word bank consists of “Yes”, “No”, "up", "down", "left", "right", "on", "off", "stop", and "go". Each word has over 2000 sample sound tracks, and the time length of each tracks is about 5 seconds.

The architecture used in this model is based on convolutional neural network (CNN). [2] Under this architecture a set of 1-D sound discrete signals are converted into a 2-D image data which the dimension is corresponding to the time of signal duration by the range of frequency. Below is the illustration of CNN architecture. The weight matrix ***W*** consists of ***n*** local patches which are truncated from the input images. The input size is ***t x f*** (i.e. time x frequency) and thus the dimension of feature maps ***m*** and ***r*** is less or equal to ***t*** and ***f***. Since there are total ***n*** hidden units in the weight matrix, when the convolution is implemented with the filter of ***s x v*** (i.e. the sampling interval in the time and frequency domains), totally ***n*** feature maps with a set of specific size shown in the figure would be generated. Given a fact that there exists a substantial variance in one feature because of the speak’ tone and accent, the pooling is performed to reduce the tremendous size of features but to still maintain translation-invariant among all features. [4] This process can be also regarded as subsampling shown in the right part of figure.



In our training we implement two convolutional layers, which has been proved to work well on large-vocabulary continuous speech recognition (LVCSR). [5] In each convolution layer the input size is 17(***m***) X 40(***r***). The first filter size is 8(***s***) X 20(***v***) and the amount is 64. The second filter size is 4(***s***) X 10 (***v***) with the same amount as the first one. The parameter of pooling is 2(***p***) X 2(***q***). The required training steps is set to 15,000. Below illustrates the graph of accuracy increasing versus training steps. The accuracy rate is sampled every 100 steps.



➢ **Bandpass Filter**

The keyword detection method is especially limited in distinguishing scenarios of single voice and crowd voice. We assume in a place with crowd the dominant sound frequency may change significantly; while in the single voice scenario such as 2 people in a meeting room the dominant frequency may be comparatively stable. Hence, we implement a bandpass filter to periodically check whether the dominant frequency is located in the range of human voice or not. If in a constant period most dominant frequencies are located in human voice range, the scenario is identified as single voice; while if most dominant frequencies are out of the range, we consider should be ambience noise. Besides these two conditions, we define all other conditions belong to the crowd voice scenario.

References

1. https://www.tensorflow.org/versions/master/tutorials/audio_recognition
2. http://www.isca-speech.org/archive/interspeech_2015/papers/i15_1478.pdf
3. https://introcs.cs.princeton.edu/java/97data/FFT.java
4. http://ufldl.stanford.edu/tutorial/supervised/Pooling/
5. T. N. Sainath, A. Mohamed, B. Kingsbury, and B. Ramabhadran, “Deep Convolutional Neural Networks for LVCSR,” in Proc. ICASSP, 2013.