Extended Abstract

In recent years, virtual reality (VR)/augmented reality (AR) headset has gradually become popular and prevalent. VR/AR has emerged as an increasingly significant area of research and development, with applications spanning across multiple domains including entertainment, education, healthcare, and engineering. Moreover, researchers and developers have explored a lot about the interactivity enhancement techniques of VR/AR Headsets. Better interactivity contributes to more human-centered, comfortable and easier user experiences, opening up new possibilities for training, simulation, and so on. For AR glasses, even with wide application areas, few works focus on its interactivity.

Therefore, this project uses hand-scratching gestures on AR glasses surface as the input, enhancing interactive experiences by classifying audio of these gestures. The project introduces a low-income approach to achieve the goal that the user bounds a matte tape on the glasses arm for hand-scratching. In total, there are three gestures: double click, slide forward and slide back and forth. To achieve this, we built an Andriod application to execute the training data recording and real-time gesture classification functions of Madgaze AR glasses. A CNN (Convolutional Neural Network) model with MFCC (Mel Frequency Cepstral Coefficients) features of audios as input was trained by noise and data recorded ourselves in Python for better data processing and modelling performance. Finally, the deliverable application should connect to the AR glasses to use and has two functions: First is to record a 2 seconds audio and store locally for training use; Second is to start a real-time classification process and show the current gesture detected on the screen. With real-time hand-scratching gesture recognition, these gestures are transformed into a new interactive method on AR glasses.

Acknowledgments

I would like to express my sincere appreciation to those people who assisted or supported me during the whole process of the project. It's your help that makes me complete the project.

Deep gratitude goes to Dr ZHU Kening, my Final Year Project supervisor, for assisting me throughout the whole process through his insightful and in-time feedback and suggestions on my previous reports and application development.

At last, my thanks are extended to my parents and friends for their selfless help and encouragement during the development of the project.

Table of Contents

1.Introduction

1.1 Backgrounds and Motivations

As a part of the HCI (Human Computer Interaction) field, virtual reality (VR)/augmented reality (AR) glasses or headsets are widely discussed in previous research, including in medical treatment [1], human-robot collaboration [2], the automotive industry [3], collective workspace [4], and management and education [5]. These devices have great potential in multiple areas to enhance productivity or facilitate work with large markets [1, 5].

In common, high-end headsets, such as HTC Vive, and Playstation VR, often with extra controllers, are more expensive [6]. To make VR/AR techniques fit in a low-budget context in these application areas, reducing the cost has been a non-neglectable topic in VR/AR glasses for an extended period. Since the release of the Google Board, various low-cost designs have been proposed [6]. Without extra controllers or devices in other high-end headsets, based on low-cost VR/AR equipment, many techniques have been developed to enhance its interactivity, such as extra touch panels on the surface [7], voice-based [8] and magnet-based interaction [9], and gesture and acoustic-based interactions (GestOnHMD) [10]. Some previous work focused on extra equipment for low-cost headsets [7, 8, 9], which needs extra costs and troublesome settings. However, GestOnHMD designed a completely software-based program on Google Cardboard with recognition of acoustic features [10].

For AR glasses, the interactivity is limited in products with arranged budgets, and the operation can be complicated and not human-centred (focusing on human). Few researchers [6] pay attention to the low-cost hardware interactivity in the AR glasses field. Its interactive operation space is still significant and remains unexplored. Therefore, to increase the interactivity of AR glasses with small costs and meet the demands in large application areas in a limited cost

context, we are working on transferring the acoustic idea of GestOnHMD to AR glasses with a purely software-based application in Android.

1.2 Problem Statement, Project Objectives and Scope

Various techniques for higher interactivities were applied in many headsets [6, 7, 8, 9, 10], but low-cost interactivity enhancement in the AR glasses field remains unclear. Under a shrinking budget but high interactivity demands scenario, a simple and easy-to-perform solution is necessary.

The project aims to address the low interactivity issue of some AR glasses with a low-cost solution. It plans to detect a particular pattern of the sounds caused by users' scratching on the rough surface of one of glasses arm, and optimal patterns are also designed through exploration. It should distinguish the difference and enable to trigger correct operations within a relatively short period without interrupting operations. With these functions, this project should offer a feasible solution to the interactivity expansion in a low-cost background with simple processes.

The project focuses on using Madgaze AR glasses to develop the application on the Android platform. It contains several parts: First, an audio recording function is necessary to connect to the AR glasses and generate acoustic data for future training. Then a machine learning model is designed to distinguish the different acoustic patterns with training data. Finally, the deployment of the model to the Android phone is needed with real-time pattern detection.

1.3 Report Structure

The remained paper is structured as follows: in section 2, we first review previous VR/AR headsets work, including low-cost headsets, their corresponding interactivity technique. We then introduce related concepts of audio processing and CNN (Convolutional Neural Network).

Section 3 introduces the methods and design applied in the project, and section 4 describes our low-cost AR glasses application's structures and algorithms. In section 5, we evaluate the function and accuracy of the current system and demonstrate the results. A summary with limitation and future work of the whole project is provided in section 6.

2. Literature Review

2.1 VR/AR Headset and Interaction

Various low-cost headsets come to the market, mainly depending on smartphones, such as Google Cardboard, Google's Daydream and VR PRO [11]. Typically, the motion sensors equipped in the smartphone detect the head rotation as the input [10]. Moreover, Yuhang et al. designed several head motion gesture sets with measurement [12], and Tomasz et al. deeply evaluated the head motion recognition rate by classifiers [13]. Specifically, specific application areas, for instance, text entry [14] or self-motion [15], were explored within the head rotation operation field of headsets.

Researchers have developed abundant techniques to increase the interaction for low-cost mobile VR without extra controllers. EyeSpy VR uses the front camera of smartphones and senses eye tracking as a modal for the input [16]. Its evaluation proved the effectiveness of this method in offering a new interaction. Similarly, Junichi et al. installed Electrooculography sensors on Google Cardboard to capture eye movement [17]. Hand-based VR was also introduced in the low-cost mobile VR area. Majed et al. invented PAWdio with one Degree of Freedom hand input [18]. An earphone held in the user's hand connects to the phone, and the software receives the acoustic signal to calculate the position of the user's hand. FistPointer can distinguish the thumb-up position hand gesture with a smartphone's built-in back camera [19]. However, hand-

based selection is regarded as the worst technique for selection in low-cost mobile VR, and both head and eye-based techniques may cause neck pain [20].

More interaction approaches in low-cost mobile VR were generated. MeCap can examine the user's whole body information through two mirrored spheres hung behind the back-facing camera [6]. ScratchVR detects the sliding of fingers on the headset surface through magnetic sensing [9]. Taizhou et al. focused on acoustic features produced by surface sliding and generated GestOnHMD with a VGG-19-based classifier, a model of a Convolutional Neural Network [10]. GestOnHMD needs no extra devices or accessories, reducing the interactivity enhancement cost.

In the AR devices field, interactivity enhancement also has a wide research history. The latest practice proposed an interface to imitate objects by hand directly, such as joysticks [21]. Various 2D hand input techniques, such as on-skin input [22], and thumb-to-finger gesture detection [23], have been examined by researchers. In terms of AR glasses, Weng et al. created FaceSight, adding an extra infrared camera on the nose bridge to detect the hand-to-face gesture and increase the interactivity [24]. Also, many Ar glasses eye-tracking enhancements were proposed. [25, 26, 27]. However, to our best knowledge, most of these current AR glasses interactivity techniques require extra devices, which demands extra costs.

2.2 Audio Processing

Mel Frequency Cepstral Coefficients (MFCC) are widely used for feature extraction in audio processing tasks, especially for speech recognition [28], which is derived from " a type of cepstral depiction of the audio clip" [29]. With the frequency bands equally placed on the Mel scale in the MFCC method, the mel-frequency cepstrum is more similar to the human hearing system than the regular cepstrum [29]. MFCCs are obtained through some steps, starting with

spliting the audio signal into small segments, typically with a duration of 20-40 milliseconds, and then applying the discrete Fourier transform (DFT) to each segment to convert it from the time area to the frequency area [28].

Moreover, a filterbank that mimics the human auditory system is applied to the DFT magnitudes, which maps the signal onto a mel scale, where the distance between frequencies is perceived as equal as humans' [28]. The logarithm of the filterbank outputs is then taken, followed by a final DCT (discrete cosine transform) step to obtain the MFCC coefficients [28]. The number of MFCC coefficients usually ranges from 10 to 40, and these coefficients represent the spectral characteristics of the audio signal in a compressed form [30].

Another approach is Gammtone Frequency Cepstral Coefficients (GFCC), which conveys the spectrum through a Gammatone filter bank and then further processes. GFCC have advantages under noisy conditions, which is, however, not the scenario for AR glasses used in the project. For better environment support and easier use, we select MFCC to extract audio features.

We built the CNN (Convolutional Neural Network) model to classify the sounds with MFCC features as input, which is widely used and gets good results [30]. Based on introduction of [30, 31], CNN is a kind of popular neural network, that processes data arrays, especially images. One of the important operations, the convolutional layer, processes the data with convolution with a trained filter and bias. The detailed computing is as follows:

$$CFM_{x,y} = f\left(b + \sum_{i=1}^{k_h} \sum_{j=1}^{k_w} K_{i,j} \circledast I_{x+i,y+j}\right) \qquad (1)$$

CFM is the convoluted feature map as the result of the operation. f() is the activation function like ReLU(rectified linear unit); b is for Bias; kh,kw are the length and width of kernels and the

* with a circle represents two-dimensional convolution. Another important operation is the subsampling layer, such as max-pooling and average-pooling. The detailed computation is:

Max-pooling:

$$h_j^n(x, y) = max_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Average-pooling:

$$h_j^n(x, y) = 1/K \sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

The max-pooling use the maximum to represent all data points within N(x) and N(y), and average-pooling computes their average value as the representation. Subsampling transforms the data array to a receptive size based on the output size needed. A dropout layer is usually followed by the previous two-layer operations, randomly dropping some data with a set probability to avoid overfitting the model.

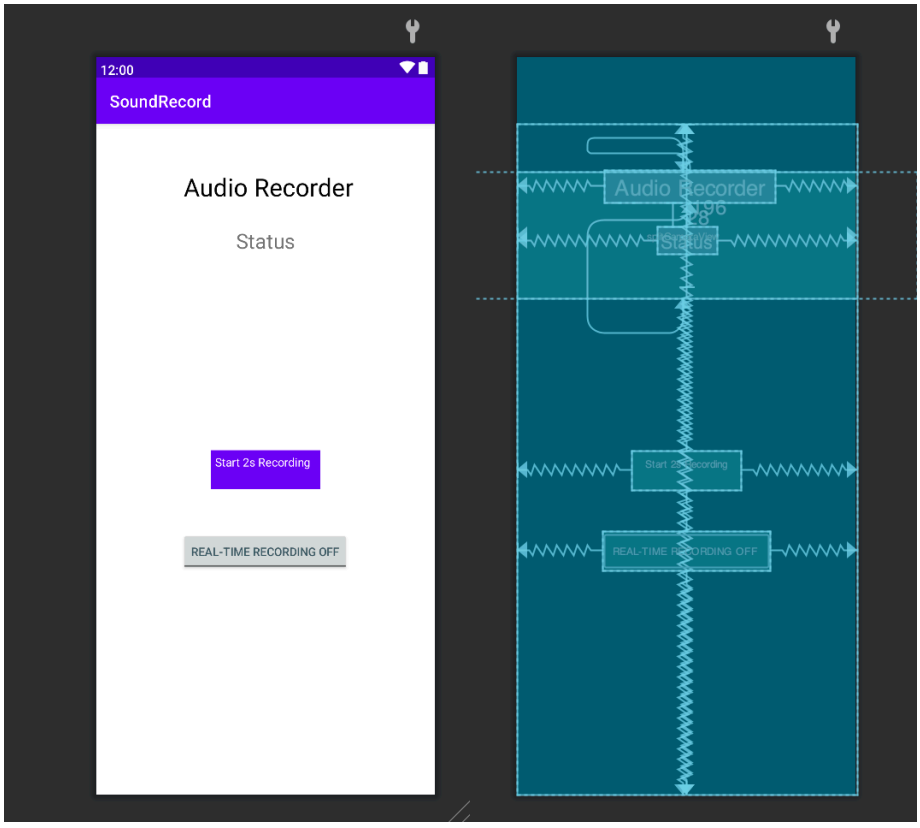
3. Proposed Design, solution, and system

This section will demonstrate the design, solution, and system, and we divide the whole system into four parts, following the sequence of how data is generated and used. We will first focus on developing the AR glasses sound recording function to enable our device to get acoustic data. Then we will describe how the acoustic data is collected and processed for training use. The system also includes the real-time gesture sound pattern recognition function to distinguish various hand-scratching command patterns. Finally, the interface design of the final gesture recognition application will be also included.

3.1 AR Glasses Sound Recording

The AR glasses we used, MadGaze GLOW PLUS, have no feasible sound recording application, and we also require high control of the recording function to collect data. Therefore, we developed a function in Android application to control the AR glasses through a Type-C to Type-C cable provided in the MadGaze GLOW PLUS set.

The application can use the AR glasses sound recording function through the "Start 2s Recording" button. We restricted the recording time to 2 seconds to guarantee the same length for each record. Once the button is clicked, it will start recording and end when it's 2 seconds. The application can save the sounds as PCM files located in the file system of the connected Android device. The PCM files generated by the recording function will be transferred to WAV files to facilitate us to listen the records directly for checking and further processing.
The function is purely for development use, and the application is designed as follows:



3.2 Data Collection and Processing

The acoustic records of different gesture patterns are required to train a model to distinguish the patterns. We use a low-cost method to make sounds and then increase the interactivity: a matte tape is bound on one of the glasses arm, allowing users to scratch and make sounds. Based on the previous work [10] on hand-scratching gestures on headset devices, we select three hand-scratching gestures as operations, including double-click, scratch forwards, and scratch back and forth. Each operation is exerted on the tape-bonded area of the glasses. We should ask two indivisuals to repeat each operation 100 times and then get 600 WAV files with a 2 seconds length in total.

The environment noises should be also included, to increase the accuracy and prevent our model from wrong recognition by imparting them to the model with recorded pattern data. We will generate 2 second noise WAV files randomly selected from online EFC50 datasets (https://github.com/karolpiczak/ESC-50/tree/master/audio), which contains a variety of environment noise.

The data processing part should process a large amount of data, and normalize all data to 2-second single-channel format WAV files. The shorter files should be filled with zero amplitude, and the exceeding parts are eliminated because the main pattern sounds are all included within 2 seconds through manual checking. Data Augumentation techniques should be used to augment the relatively small dataset to get better performance in various situations.

3.3 Feature Extracting and Pattern Recognition

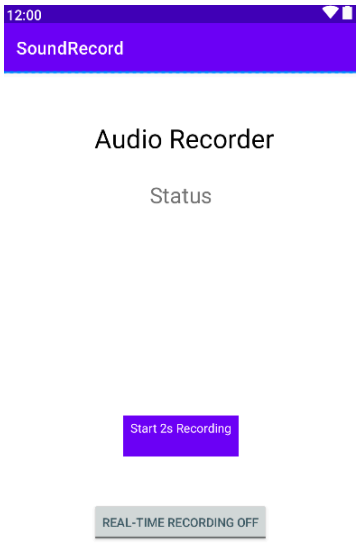
For feature extracting, the MFCC features of WAV sound files are used as further input. A deep learning model based on CNN should be used for sound pattern recognition, which modifies its

parameters itself through training data collected before. The model should classify new coming data into one of three patterns or noise with an acceptable error rate.

After model building, we will deploy the model to the Android device. The system should have a real-time data processing function to recognize sounds in a real-time scenario, which will be controlled by a button and react in a short time. It should have a reasonable running time for each record and display the result to the user in time.

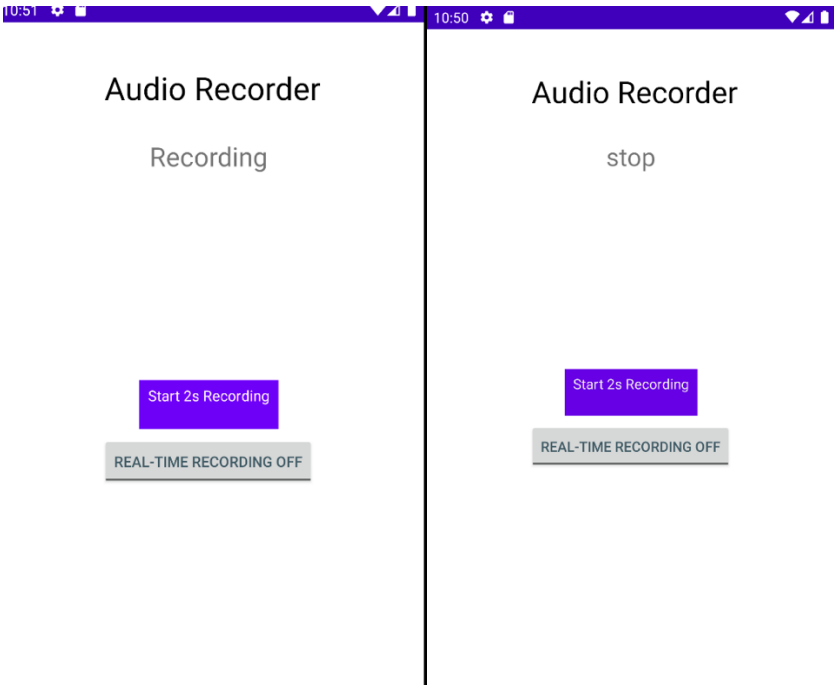
3.4 Interface Design

The interface is expected to explicitly show the result of current recognition and be able to turn on or off the function buttons. The current status of the application is necessary for the user to indicate the current action to them. We should also consider the design principles for user experience to create an excellent and user-friendly interface. The interface is designed as follows:



Under the title of our application "audio recorder", the status TextView handles the function of displaying all information about what this application is doing to the user. The purple button,

"Start 2s Recording", takes the responsibility of recording a two seconds audio from the connected AR glasses, mainly for development use. Once the button is clicked, the status TextView will be transformed to "Recording" to indicate the application is recording audio. After that, the TextView will become "Stop".



The gray button with "REAL-TIME RECORDING OFF" is for start the real-time gesture recognition function. Once it get clicked, the text on it will become " REAL-TIME RECORDING ON" with green line below to show the working status.The status TextView above will be changed to "Record first 2s…". After 2 seconds, the status TextView will show the current recoginition result for the user and keep updating as more real-time audio comes.

3.5 Test and Experiment

We will be using accuracy, precision, and recall as evaluation metrics to assess the performance of our model. The results should be acceptable and able to ensure normal use in general situations. For real-time performance, we should perform event mocking such as testing the function by feeding long audio to mimic the real-time situation. An integration real-time test

should be also applied, which requires the tester uses the application for a relatively long time and check the accuracy of its gesture recognition.

4. Detailed Methodology and Implementation

In this section, we display the methodology of the project in detail. We also show the actual implementation details and algorithm behind. First, the details about Android application development are introduced. Then two main functions, two second audio recording and real-time gesture recognition function with data processing, are discussed. Finally, we talk about how the UI related elements are designed.

4.1 Android Application

All sound recording and real-time classification functions are implemented in an Android application called SoundRecord, developed by Android Studio 2021.2.1 Patch 1. The APIs between the Android app and AR glasses and Android SDKs are provided by MadGaze GLOW PLUS development support (https://sdk.madgaze.com/glow/android-sdk). However, the GitHub link offered on its official website is missing. The packages in the project are downloaded from other's GitHub project directly. The packages includes glow-development-kit-v1.1.0, glow-uvc-library and usbserial-v6.1.0.

The Android app builds the connection by SplitUSBSerial.getInstance(this).setConnectionCallback(new USBSerial2.ConnectionCallback()

{…} and check the connection status by SplitUSBSerial.getInstance(this).isDeviceConnected(). To test the audio function of AR glasses, we developed a glasses camera video-catching application which is completely supported by the official website used before. The result shows the audio capturing system of glasses is intact.

4.2 AR Glasses Sound Recording

The video capturing function introduced on AR glasses official website is integrated, but the audio function is lacked and missing. Therefore, we developed the audio-related functions based on the AudioRecord package in Java for audio recording, generating and saving. In terms of the project structure, the AudioHelper class file handles all sound recording functions. To record the audio and avoid stopping the main thread, an extra thread, the Audio_Thread which extends the Thread class and is developed by us, is needed.

We select 44100 as the sample Hz, 16 bit per sample as the audio data format and multi-channels (*CHANNEL_IN_MONO*) for our audio record setting, which are the most widely accepted configuration in Android phone. In fact, for our develop phone, Honor , only 16bit audio format and multi-channel can sucessfully generate audio. To record through the AR glasses audio input, we simply set the sound source to "MediaRecorder.AudioSource.*MIC*", and the AudioRecord object will record the sound from the glasses. The sound source parameter is derived from the source code of official package. In the video function codes, these audio recording codes are within the video-related methods, and the audio related methods in official document are just empty.

The recording is stopped automatically by a handler with handler.postDelayed(Runnable, delay mileseconds) method. The Handler class evokes a new thread and therefore maintains UI update in the main thread. Without the handler, users cannot be informed of record finishing by

showing the text "stop". The runnable contains the record saving method "save_recording()". In this method, recorded data is transferred to the buffer by audioRecord.read(buf, 0, buf_sz) and then saved as pcm files by OutputStream. Here we selected 44100*4 as the buffer read size in bytes to guarantee 2 second's data because it's 16 bit (2 bytes) per sample (44100*2 bytes for 1 second).

The files stored are PCM files. Before storing, we transfer the PCM format files to WAV files by methods in the PCMtoWAV class. The main change is adding some extra heads and byte shifting to PCM files. WAV format benefits further audio recognition and enables us to check the recorded files directly by hearing from it.

4.3 Data Collection, Processing, Model Building

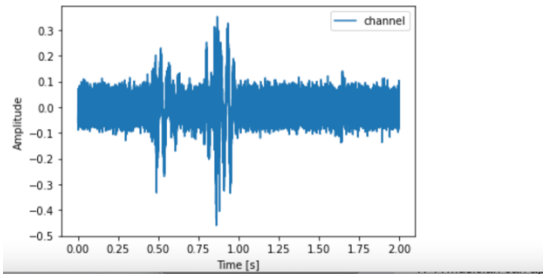
The data is collected following previous design and gets 600 hadn-scratching gesture records in total. The data processing, and model building are all done by Python 3.9 in Google Colab. The WAV files are stored in Google Drive, achieved in Colab by: from google.colab import drive and adding os path.

The panda package handles the function of most data processing work. The WAV files are loaded to a DataFrame by Librosa package, which is the most popular Python tool for audio processing. Even though the audio data length is fixed by previous recording algorithm, the project applies a normalization function to ensure that all audio records have the same length: For short records, we fill them with zero from behind. For long records, we delete the exceeded part (After manual checking by listening directly, there is no gesture sound in this part). Then we augment the data by jittering (add random noise to each sample in each record) and scaling to simulate the real complicated environment. Other audio data processing methods like magnitude warping and rotation are discarded for not salient behaviors or unreasonable scenario they may

simulate in our case (sounds come from a fixed position, the matte on the glasses arm). Finally, we have 1800 audio gesture records. To balance the dataset, we randomly selected 1200 records in EFC50 dataset as noise data. In total, 3000 records are used for further training. A visualization function is designed for check specific records. Matplotlib.pyplot package is imported for the image display. The y-axis represents the amplitude of the audio, and the x-axis represents the time flow. We are therefore able to check how it sounds like without searching and directly listening.

```
import numpy as np
import matplotlib.pyplot as plt
length = data.shape[0] / sr
#print(data.shape[0],length)
time = np.linspace(0., length, data.shape[0]) #data.shape
plt.plot(time, list(data), label="channel")
plt.legend()
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.show()
```

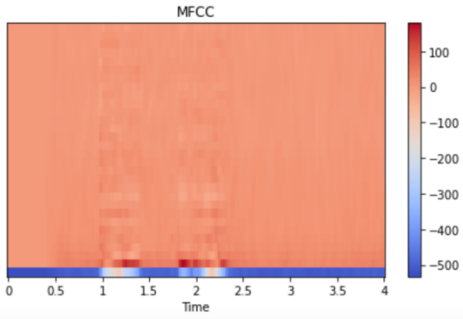
```
[-0.03752857 -0.06307802  0.03295134 ... -0.01261457  0.0238
  0.0258853 ]
```



Then Librosa package helps extract the MFCC features from audio records for further training. The parater n_mfcc in librosa.feature.mfcc method determines the shape of the MFCC data and also the details of its spectrum. After testing n_mfcc=64, n_mfcc=30, n_mfcc=20 and n_mfcc=10, we found that with the parameter equaling 20 or 30, the model gets the best performance. For n_mfcc=10, the small value may lose too many details to distinguish the audio. Also, the spectrums of MFCC are visualized by matplotlib and Librosa's function as well:

```
#    Vis MFCCS
#====================
h = all_data["mfccs"][501]
#h = trydata["mfccs"][1]

from matplotlib import cm
librosa.display.specshow(h, x_axis='time')
plt.colorbar()
plt.title('MFCC')
plt.tight_layout()
```



To split data for train and test, refering to one of the widely used proportion 8:1:1 for train data: validation data: test data, 10% of record dataset are randomly sampled for the test data, and 10% are for validation data used to test the model during the training stage. The validation can provide a strong indication of how the train model works on non-training data and examine whether there is overfitting occuring. The evaluation standard is "accuracy" here, which will be explained further in the test section. The dimensions of data are extended for the tensor format requirement of CNN input. The CNN model is built by tensorflow.keras with Adam as the optimizer and Cross Entropy as the loss. We has modified the structure including adding more layers to make it more deep or using more parameters. However, it easily raises the overfitting issue, which works well on offline test with our dataset but quite bad on real-time data. Finally, we determined current structure. The network structure is as followed:

```
input_shape=(20,173,1)
CNNmodel = models.Sequential()
CNNmodel.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
CNNmodel.add(layers.MaxPooling2D((2, 2)))
CNNmodel.add(layers.Dropout(0.2))

CNNmodel.add(layers.Conv2D(64, (3, 3), activation='relu'))
CNNmodel.add(layers.MaxPooling2D((2, 2)))
CNNmodel.add(layers.Dropout(0.2))

#CNNmodel.add(layers.Conv2D(128, (3, 3), activation='relu'))
#CNNmodel.add(layers.Conv2D(64, (3, 3), activation='relu'))
#CNNmodel.add(layers.MaxPooling2D((2, 2)))
#CNNmodel.add(layers.Dropout(0.2))

#CNNmodel.add(layers.Conv2D(128, (3, 3), activation='relu'))


CNNmodel.add(layers.Flatten())

#CNNmodel.add(layers.Dense(128, activation='relu'))
#CNNmodel.add(layers.Dropout(0.2))
CNNmodel.add(layers.Dense(64, activation='relu'))
CNNmodel.add(layers.Dropout(0.2))
CNNmodel.add(layers.Dense(32, activation='relu'))
CNNmodel.add(layers.Dense(4, activation='softmax'))

CNNmodel.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),metrics=['accuracy'])
```

The model is then transformed to tensorflow format, tflite, by TensorFlow's convertor for Java use.

4.4 Real-time Classification in Android

Getting the Java-capable CNN model, the Android application should be functional to utilize it. The tfOperation.java takes the role of the model predicting and real-time monitoring supports. The model is physically saved in assets with a size of 875KB and stored in a MappedByteBuffer object. The predict function in the tensorflow package in Android requires inputs, outputs, and the model itself to generate the results (version: implementation 'org.tensorflow:tensorflow-lite:2.11.0' implementation 'org.tensorflow:tensorflow-lite-support:0.4.3').

Another Thread subclass, tf_thread, takes the role of the real-time function with a while loop. In each loop, the latest audio data will be saved as WAV file through previous "save_recording" function. Next, in next called BufferInput method, the 2 second audio records are buffered and transferred to TensorAudio and then TensorBuffer object as inputs. The records generated will be overwritten by later newly generated records, so the storage is controlled. The output is empty four float number length array, waiting for storing the results. Both output array and model loading are complete before the loop to reduce the unnecessary workload.

Once the real-time function is triggered by clicking the ToggleButton, the loop will start. For the first 2 seconds, with the text "Record first 2s..." shown, the application records the audio for the first 2 seconds to generate the first output. There is a thread sleeping process made by Thread.sleep() to stop the tf_thread before entering the while loop, guaranting the first 2 seconds record is complete. A sliding window algorithm is set with 2 second as the window size and 500ms as the step. We executed the algorithm by setting two buffer array to store previous three quarters of the recording data and the newly coming quarter of recording data. It's because the step is a quarter of the window size. By keeping generating audio data, the records are feeded to the CNN model in a sliding-window manner. Due to constant and overlapped recording data feeding, the application therefore avoid the probability of splitting a complete gesture sound to two records and making the recognition fail.

To avoid replicating two similar saving_record functions, here we add a flag called supplyOrNot to determine whether it needs to store three quarters of previous audio data. We make it be True after each time's call to execute the storing and buffering next turn. For normal 2 seconds recording, we change the flag back to the Flase when stop_recording method is called.

In this case, the CNN model will output four float number to the output array. The index with the highest value represents the label corresponding to this index gets the highest probability, which is regarded as the final result and get shown on screen.

The loop will stop the ToggleButton gets clicked again at any time. The stop_recording method triggered by this operation will interrupt the thread by sending the signal and call the setText method to change the text on screen to "stopped". The interruption signal is handled by Thread.*currentThread*().interrupt() method to stop the thread without terminating the whole

application. In the while loop, there is a 100 milliseconds thread sleep for handling the interruption signal. Otherwise, the program will get failed to stop.

4.5 UI

The UI change in Android application is in the main thread. However, if there are other massive codes behind the UI modification command, UI modification event will queue after all of massive process. Therefore, for 2 second recording, we developed a handler to start a new thread and leave the main thread has nothing to execute but the UI change code. Similarly, in real-time recording and tfOperation class, all massive codes, such as the loop, are executed in a single thread to avoid to affect UI events.

Only MainActivity class can change UI directly. Thus, we designed a method called "setText(final TextView text, final String value)" with *runOnUiThread* method within to put the textview change process in the UI Thread (Main Thread).

5. Testing Procedure and Results

5.1 Offline Test

In pattern recognition and other classification task, accuracy, precision and recall are popular to check the model behavior as the test. Suppose we use TP for True Positive (the number of predicting the positive class successfully), TN for True Negative (the number of predicting the negative class successfully), FP for False Positive (the number of failing to predict the positive class) and FN for False Negative (the number of failing to predict the negative class). Then the definition of accuracy is:

$$Acc=(TP+TN) / (TP+TN+FP+FN)$$

It represents all successful prediction of the test regardless of P or F. Then we get a good result in our offline test using recorded data:

```
7/7 [==============================] - 0s 29ms/step - loss: 0.1881 - accuracy: 0.9600
[0.18808302283287048, 0.9599999785423279]
```

Precision quantifies how many of the positive predictions made by the model are actually correct. It shows the proportion of successful predict of positive class. The definition of Precision is:

$$Pre = TP / (TP+FP)$$

Then Recall is to measures the proportion of actual positive samples that are correctly identified by the model among all positive cases in the data set, which is also known as sensitivity or true positive rate. A high recall score indicates that the model is able to correctly identify most of the positive instances, while a low recall score indicates that many positive instances are being missed by the model. Recall is important for tasks where identifying all positive instances is crucial, such as making sure all gestures are correctly captured and distinguished in this project. The definition of Recall is as follows:

$$Recall = TP / (TP+FN)$$

The F1 score takes into account both precision and recall. Accuracy measures how many predictions the model gets right, but the F1 score evaluates the model's performance in terms of both false positives and false negatives. By considering both precision and recall, the F1 score is balanced assessment of the model's ability to correctly classify data. The F1 score is given by:

$$F1 = 2*Pre*Recall / (Pre+Recall) = 2TP / 2TP+FP+FN$$

And we have:

```
1 from sklearn.metrics import classification_
2
3
4 y_pred = CNNmodel.predict(X_test)
5 y_pred_label = np.argmax(y_pred, axis=1)
6
7 print(classification_report(y_test, y_pred_
```

```
7/7 [==============================] - 1s 78ms
              precision    recall  f1-score   s

           0       0.92      0.92      0.92
           1       0.92      0.96      0.94
           2       0.98      0.93      0.95
           3       1.00      1.00      1.00

    accuracy                           0.96
   macro avg       0.95      0.95      0.95
weighted avg       0.96      0.96      0.96
```

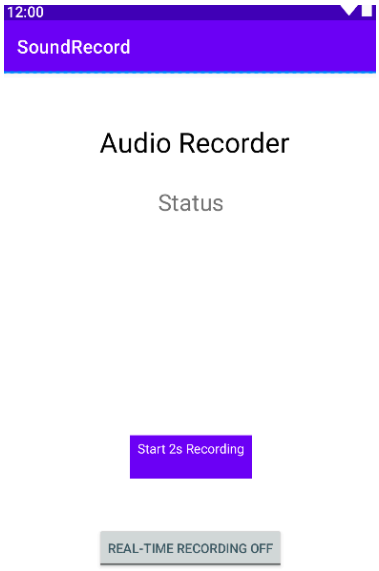
The offline test accuracy, precision, recall and f1-score obtained in the testing stage of the CNN model in Colab are all highly accurate.

5.2 Real-time Test

We first conduct an integration test by starting the real-time function and keep using the application for 10 minutes for three times. The result shows there's no apparent issue occurred. Then we conduct a simulation test by feeding a 15 minute audio with 180 gestures inside recorded by us to the real-time model in Java. Appoxiamately 85.5% of the gestures are predicted successfully, which is acceptable. Throughout the entire runtime, no instances of lagging or crashing were observed.

5.3 Application Development Results

As shown in section 4, the application handles two functions and has simple design for users to use:



The button size and position is based on Fitt's Law to give a better reaction and finger moving experiences for the users.

6. Conclusion

The goal of this project is to develop a real-time audio-based gesture recognition system using deep learning techniques. In this study, we trained a CNN model to classify audio signals of hand gestures with promising results. However, due to the time constraints, several limitations still exist, including further improvement in real-time audio recognition performance, the need for a larger dataset, and the possibility of hidden problems that have not been discovered in long-term testing. Future improvements could include incorporating additional gesture types and feature extraction methods, testing other deep learning techniques, and exploring other machine learning methods and audio processing techniques. Overall, this project develops an audio-based hadn-scratching gesture recognition Android application and provides insights into areas for future research and development.

6.1 Summary

This report describes a project that enhances the interactive experiences of AR glasses by using hand-scratching gestures as input, which are classified through audio analysis. The project proposes a low-cost approach that involves attaching a matte tape on the glasses arm for hand-scratching. An Android application was developed for recording training data and real-time gesture classification using a CNN model with MFCC features. The deliverable application can record and save a 2-second audio for training and perform real-time classification of hand-scratching gestures. The test results have been good, as the real-time audio recognition can run smoothly and deliver decent outcomes.

6.2 Limitation

During the course of this project, a number of limitations were encountered as a result of time constraints. Although the test results were promising, there remains space for improvement in the real-time audio recognition function. For instance, while the system performs reasonably well, it could potentially be further optimized in terms of accuracy, and the operating speed could be increased. This may be achieved through refining the algorithm used for audio feature extracting or by changing the structure of model. Additionally, the dataset used for training the model could be expanded to encompass a broader range of environment or ways (like more indivisuals to scratch and record) to scratch the surface, as the current model may not perform as well in certain scenarios.

It is also worth noting that the testing conducted thus far has been relatively short-term in nature, and there may be potential issues that have not yet been identified. Therefore, it may be advisable to conduct longer-term testing to validate the system's performance over time. Overall,

while the results of the current testing are promising, further refinement and testing may be required.

6.3 Future Improvement

In the future, this project could incorporate additional gestures to increase the variety and interactivity of the operations. This would require a larger model and more data. One possible approach to support this would be to use a simple CNN model to first determine whether the input signal is noise before attempting to recognize specific gestures, which could potentially reduce unnecessary computational time of the larger model.

In addition, developing a suitable API to enable the real-time gesture recognition functionality to run in the background and act as an input control for other applications would be beneficial. This would allow for the increased interactivity offered by the project to be utilized across a wider range of fields.

Furthermore, I aim to further develop the application by incorporating additional feature extraction methods, such as GFCC, which has demonstrated promising performance in audio recognition similar to MFCC. I intend to compare and contrast the performance of these two methods to enhance the accuracy of the application. Moreover, I would like to explore other deep learning techniques beyond CNN, such as the latest Transformer, and compare their performance. It should be noted, however, that these methods may be too complex for the current small dataset and could impact the application's processing speed. Therefore, other machine learning methods and hard-coding audio processing methods should also be considered. The differences between various gesture audio signals are often quite noticeable and can even be distinguished visually through visualization techniques.

References

1. Krause, Q., and McCrory, B.: ‘Using Cost Efficient Augmented Reality Glasses in Anatomical Identification’, Proceedings of the Human Factors and Ergonomics Society Annual Meeting, 2021, 65, (1), pp. 1004-1008

2. Dianatfar, M., Latokartano, J., and Lanz, M.: ‘Review on existing VR/AR solutions in human–robot collaboration’, Procedia CIRP, 2021, 97, pp. 407-411\

3. Ciprian Firu, A., Ion Tapîrdea, A., Ioana Feier, A., and Drăghici, G.: ‘Virtual reality in the automotive field in industry 4.0’, Materials Today: Proceedings, 2021, 45, pp. 4177-4182

4. Mulder, J.D., and Boscker, B.R.: ‘A modular system for collaborative desktop VR/AR with a shared workspace’, in Editor (Ed.)^(Eds.): ‘Book A modular system for collaborative desktop VR/AR with a shared workspace’ (2004, edn.), pp. 75-280

5. Sosnilo, A., Kreer, M., and Petrova, V.: ‘AR/VR technologies in management and education’, Управление, 2021, 9, (2), pp. 114-124

6. Ahuja, K., Harrison, C., Goel, M., and Xiao, R.: ‘MeCap: Whole-Body Digitization for Low-Cost VR/AR Headsets’, in Editor (Ed.)^(Eds.): ‘Book MeCap: Whole-Body Digitization for Low-Cost VR/AR Headsets’ (Association for Computing Machinery, 2019, edn.), pp. 453–462

7. J. Gugenheimer, D. Dobbelstein, C. Winkler, G. Haas and E. Rukzio, "FaceTouch: Touch interaction for mobile virtual reality", *Conference on Human Factors in Computing Systems - Proceedings*, vol. 07, pp. 3679-3682.

8. J. Gu, Z. Yu and K. Shen, "Alohomora: Motion-Based Hotword Detection in Head-Mounted Displays", *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 611-620, 2020.

9. R. Li, V. Chen, G. Reyes and T. Starner, "ScratchVR: Low-cost calibration-free sensing for tactile input on mobile virtual reality enclosures", *Proceedings - International Symposium on Wearable Computers ISWC*, pp. 176-179, 2018.

10. Chen, T., Xu, L., Xu, X., and Zhu, K.: ‘GestOnHMD: Enabling Gesture-based Interaction on Low-cost VR Head-Mounted Display’, IEEE Transactions on Visualization and Computer Graphics, 2021, 27, (5), pp. 2597-2607

11. “BNEXT,” *Aniwaa*. [Online]. Available: https://www.aniwaa.com/brands/bnext/. [Accessed: 22-Oct-2022].

12. Y. Yan, C. Yu, X. Yi, and Y. Shi. Headgesture: hands-free input approach leveraging head movements for hmd devices. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 2(4):1–23, 2018.

13. T. Hachaj and M. Piekarczyk. Evaluation of pattern recognition methods for head gesture-based interface of a virtual reality helmet equipped with a single IMU sensor. Sensors (Switzerland), 19(24):1–19, 2019. doi: 10 .3390/s19245408

14. C. Yu, Y. Gu, Z. Yang, X. Yi, H. Luo, and Y. Shi. Tap, dwell or gesture? exploring head-based text entry techniques for hmds. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, pp. 4479–4488, 2017.

15. Costes, A., and L'ecuyer, A.: 'The "Kinesthetic HMD": Enhancing Self-Motion Sensations in VR with Head-Based Force Feedback', ArXiv, 2021, abs/2108.10196

16. K. Ahuja, R. Islam, V. Parashar, K. Dey, C. Harrison, and M. Goel. EyeSpyVR: Interactive Eye Sensing Using Off-the-Shelf, Smartphone-Based VR Headsets. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 2(2):1–10, 2018. doi: 10.1145/3214260

17. J. Shimizu and G. Chernyshov. Eye movement interactions in google cardboard using a low cost EOG setup. In UbiComp 2016 Adjunct - Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing, pp. 1773–1776, 2016. doi: 10.1145/2968219.2968274

18. M. Al Zayer, S. Tregillus, and E. Folmer. PAWdio: Hand input for mobile VR using acoustic sensing. In CHI PLAY 2016 - Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play, pp. 154–158, 2016. doi: 10.1145/2967934.2968079

19. A. Ishii, T. Adachi, K. Shima, S. Nakamae, B. Shizuki, and S. Takahashi. FistPointer: Target Selection Technique Using Mid-air Interaction for Mobile VR Environment. Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems, p. 474, 2017. doi: 10.1145/3027063.3049795

20. S. Luo and R. J. Teather. Camera-based selection with cardboard HMDs. In 26th IEEE Conference on Virtual Reality and 3D User Interfaces, VR 2019 - Proceedings, pp. 1066–1067, 2019. doi: 10.1109/VR.2019.8797772

21. Pei, S., Chen, A., Lee, J., and Zhang, Y.: 'Hand Interfaces: Using Hands to Imitate Objects in AR/VR for Expressive Interactions'. Proc. Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, New Orleans, LA, USA2022 pp. Pages

22. Joanna Bergstrom-Lehtovirta, Kasper Hornbæk, and Sebastian Boring. 2018. It's a Wrap: Mapping On-Skin Input to Off-Skin Displays. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI 2018, Montreal, QC, Canada, April 21-26, 2018. ACM, 564. https://doi.org/10.1145/3173574.3174138

23. Mohamed Soliman, Franziska Mueller, Lena Hegemann, Joan Sol Roo, Christian Theobalt, and Jürgen Steimle. 2018. FingerInput: Capturing Expressive Single-Hand Thumb-to-Finger Microgestures. In Proceedings of the 2018 ACM International Conference on Interactive Surfaces and Spaces (Tokyo, Japan) (ISS '18). Association for Computing Machinery, New York, NY, USA, 177–187. https://doi.org/10.1145/3279778.3279799

24. Weng, Y., Yu, C., Shi, Y., Zhao, Y., Yan, Y., and Shi, Y.: ‘FaceSight: Enabling Hand-to-Face Gesture Interaction on AR Glasses with a Downward-Facing Camera Vision’. Proc. Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, Yokohama, Japan2021 pp. Pages

25. Meyer, J., Schlebusch, T., Spruit, H., Hellmig, J., and Kasneci, E.: ‘A Novel -Eye-Tracking Sensor for AR Glasses Based on Laser Self-Mixing Showing Exceptional Robustness Against Illumination’. Proc. ACM Symposium on Eye Tracking Research and Applications, Stuttgart, Germany2020 pp. Pages

26. Meyer, J., Schlebusch, T., Kuebler, T., and Kasneci, E.: ‘Low Power Scanned Laser Eye Tracking for Retinal Projection AR Glasses’. Proc. ACM Symposium on Eye Tracking Research and Applications, Stuttgart, Germany2020 pp. Pages

27. Meyer, J., Schlebusch, T., and Kasneci, E.: ‘A Highly Integrated Ambient Light Robust Eye-Tracking Sensor for Retinal Projection AR Glasses Based on Laser Feedback Interferometry’, Proc. ACM Hum.-Comput. Interact., 2022, 6, (ETRA), pp. Article 140

28. Ali, S., Tanweer, S., Khalid, S.S., and Rao, N.: ‘Mel frequency cepstral coefficient: a review’, ICIDSSD, 2020

29. Prabakaran, D., and Sriuppili, S.: ‘Speech processing: MFCC based feature extraction techniques-an investigation’, in Editor (Ed.)^(Eds.): ‘Book Speech processing: MFCC based feature extraction techniques-an investigation’ (IOP Publishing, 2021, edn.), pp. 012009

30. O. Sen, Al-Mahmud and P. Roy, "A Convolutional Neural Network Based Approach to Recognize Bangla Spoken Digits from Speech Signal," 2021 International Conference on Electronics, Communications and Information Technology (ICECIT), Khulna, Bangladesh, 2021, pp. 1-4, doi: 10.1109/ICECIT54077.2021.9641322.

31. Szeliski, Richard. "Computer vision: algorithms and application," Springer Nature, 2022.

APPENDIX A. Monthly Logs

Sep: Android learning, AR glasses video recording application development

Oct: Analyze AR glasses development kit and develop the needed recording functions in AR glasses for training data collection. Develop an Andriod app to handle this.

Nov: Complete the voice detecting and create the dataset of voice pattern recognition, and start the CNN sound recognition development

Dec: Complete the data processing of the sound dataset and add more noise data (wrong gesture sound) to the dataset

Jan: Complete CNN model building and test model structure to get a good performance

Feb: Transfer the CNN model used in Python to Java, complete the real-time classification function and test the application preliminarily.

March: Test the application and modify the model and application based on its result.

APPENDIX B. Class diagram of Android App