Topic Title: Application of Keyword Spotting System on accurately predict memory test result

A. **Problem statement**

Memory tests assess the ability of memory in the short and medium term which could help doctors to detect possible memory loss and its associated disease such as Alzheimer's in early stages which is crucial in treatment. The result, however, needs human resource and time to count. An automate solution is needed to speed up the counting process as well as saving the resources and time it takes. This thesis develops a novel approach to the problem by using a keyword spotting system to count the result automatically. The system will take a folder as input which contains the information of all the patients as well as the wav file of their responses of the memory test. It will then extract all the keywords from each patient's wav file and output the analysis of the result.

B. **Objective**

Review relevant literature in the field of KWS. Choose one KWS model to evaluate the memory test result. Analyze the performance of the model and optimize the model by studying different training parameters.

C. **My solution**

KWS using Residue Network
Slice the dataset using speech recognition model
Train the model using different split ratio and dataset size

D. **Contributions** (at most one per line, most important first)

Demonstrate the effectiveness of KWS on predicting memory test result
Demonstrate why choose KWS over SR
Figure out the optimized training parameter

E. **Suggestions for future work**

Integrate the dataset generation process into the system
Increase noise robustness of the system
Further classify the dataset (gender age) and study the result
Provides easy user interface so that self-detection dementia can be realized

While I may have benefited from discussion with other people, I certify that this report is entirely my own work, except where appropriately documented acknowledgements are included.

Pointers

List relevant page numbers in the column on the left. Be precise and selective: Don't list all pages of your report!

Literature: (up to 5 most important references)

	[21] Raphael Tang, "Howl: A Deployed, Open-Source Wake Word Detection System," 2020.
	[7] Raphael Tang, Lin, "Deep Residual Learning for Small-Footprint Keyword Spotting," 2018.
	[28] Rácz, "Effect of Dataset Size and Train/Test Split Ratios in QSAR/QSPR Multiclass Classification," 2021.

Abstract

Memory tests assess the ability of memory in the short and medium term which could help doctors to detect possible memory loss and its associated disease such as Alzheimer's in early stages which is crucial in treatment. The result, however, needs human resource and time to count. An automate solution is needed to speed up the counting process as well as saving the resources and time it takes.

This thesis develops a novel approach to the problem by using a keyword spotting system to count the result automatically. The system will take a folder as input which contains the information of all the patients as well as the wav file of their responses of the memory test. It will then extract all the keywords from each patient's wav file and output the analysis of the result. The system was evaluated using the data from the NHMRC funded Maintain Your Brain (MYB) project and shows an average accuracy of 87% is achieved in detection of 4085 males and females aged between 55 and 77.

This report provides solution for medical staff to count memory test result and potentially detect memory loss associated disease using only digital devices, leading to possibilities of self-detection of dementia in the future.

Acknowledgements

I would like to thank my supervisor, Dr. Beena, for her guidance and support. Without her knowledge this Thesis A would not be possible.

I would also like to express sincere appreciation to my assessor Dr. Elias for his constructive feedback on my seminar and poster and Mr. Mostafa for his instructions on machine learning as well as the keyword spotting infrastructure.

Abbreviations

KWS	Keyword spotting system
HMM	Hidden Markov Models
MFCC	Mel-frequency cepstral coefficients
DNN	Deep neural network
CNN	Convolutional neural network
RNN	Recurrent neural network
TDNN	Time delay neural networks
CRNN	Convolutional recurrent neural network
DS-CNN	Depth wise separable convolutional neural network
ReLU	Rectified Linear Unit
GRU	Gated recurrent unit
LSTM	Long Short-Term Memory
BRNNs	Bidirectional Recurrent Neural Network.
SR	Speech Recognition

Table of Figures

Contents

1. Introduction

1 Context

1.1.1 Keyword Spotting System overview

With increasing demands for voice assistant like Siri from Apple and Alexa from Amazon, keyword spotting, namely trigger word detection system, has gained more attention. Keyword spotting refers to the process of detecting a specific word of interest from a continuous stream of audio. [1]

Most of the Keyword Spotting System is implemented in devices that require users to preface their commands with a keyword such as “Hey, Siri” to trigger actions. These devices are usually under lots of constraints such as limited RAM and computational power. They often exist in environment with all sorts of interference such as background noise and sounds coming from devices’ own speaker in which the system is embedded. Because the output of the KWS systems has crucial impact on the states of the device, the KWS systems are required to have high detection accuracy, robustness to noise and low latency to ensure the voice commands are correctly executed and to guarantee good user experience.

1.1.2 Memory test overview

Although Keyword Spotting System has been applied widely in the voice assistant industry, it is hardly applied in the medical field. This thesis discusses possibilities to utilize the Keyword Spotting system in predicting memory test result. Memory tests assess the short and medium term of memory ability to assist doctors in detection and treatment of dementia such as Alzheimer’s in early stages. Take the NHMRC funded Maintain Your Brain (MYB) project [2] as an example, participants need to listen to the same 15 target words, then repeat immediately as many words as possible. Doctors can then assess the participants’ memory based on their performance.

1.1.3 Reason on using the Keyword Spotting System

The result from the memory test usually requires manual counting and transcript of words repeated by the participants, which is highly inefficient. Since the content of the result is an audio file consisting of multiple short words, the Keyword Spotting System may automate the processing of these memory test results. KWS could potentially be applied to predict whether the participants have given the correct response.

While there already are plenty of architectures for keyword spotting systems, they are not properly tested in the context of memory test result analysis. The quality of the audio in memory testing is sometimes lower, as some of the tests are conducted through phone calls that have more noise due to signal interference. Additionally, Keyword Spotting System usually focuses on simple command like “yes” or “no” or short phrases such as “Hey Siri”, which are quite different from potential vocabulary used in memory test results. All these factors contribute difficulties and uncertainties in developing training models to ensure accuracy of the Keyword Spotting System in its application in the medical field.

1.1 Thesis Goals

The goal of this thesis is to study the feasibility of application of KWS systems in predicting whether the participants provide the correct memory test response:

- Find the suitable KWS model for the purpose of predicting the memory test result.
- Experiment the KWS model with the memory test dataset and analyze its performance
- Find out appropriate training parameters such as the size of the dataset and split ratio to optimize the model's accuracy in predicting memory test result.

2. Literature Review

2.1 High-level structure of Keyword Spotting System

The Keyword Spotting System (KWS) takes a speech signal as an input. The goal of the system is to output whether a specific keyword have been spoken in that speech signal. To achieve this the system will first divide the speech signal of length *L* into multiple frames that are overlapped with each other. The frame has a length *l* with a stride *s*. A total number of frames $F_{total} = \frac{L-l}{s} + 1$ is obtained. The system then obtains the acoustic features from each frame, *F* by extracting sets of feature vectors from the input. After the feature extraction phase, the extracted speech feature matrix is fed into a classifier which will compute a set of probabilities on which word or phrase was spoken in the input. These probabilities will then output to a posterior handling module to average the output probabilities of each output to improve the Prediction confidence (Figure 1).

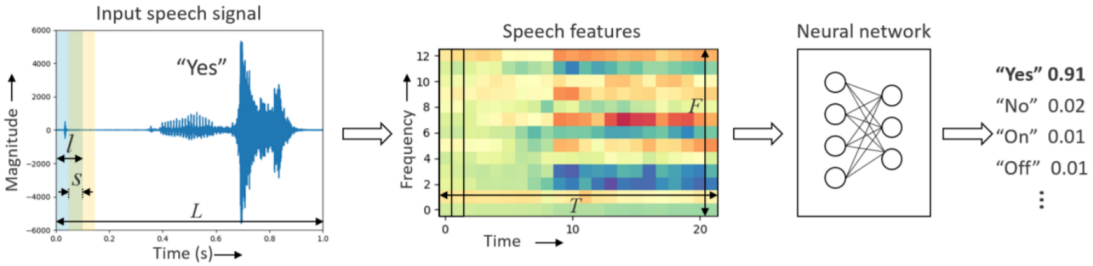

Figure 1 Overall structure on a KWS system [3]

The architecture used by the KWS system must be implemented and trained before piping in the input. Training involves training datasets which contains large number of phrases and words spoken by different people. During the training, the dataset is the input of the architecture, and the architecture will learn the different features of different words and gradually increase the accuracy.

2.2 Feature Extraction

Feature extraction is a process that has been widely used in Keyword Spotting Systems to extract a set of desired feature vectors from the speech signal. It

reduces the dimension of the raw data set to a more manageable group for signal processing by creating new features from existing ones [4]. Feature extraction is various in method and techniques [5]. Mel-frequency cepstral coefficients (MFCC) is the most used in KWS. MFCC uses Mel Scale which is a scale of pitches evaluated by perception of listeners. In order to calculate the coefficients, the signal splits into frames (F), the estimation of the spectral density of each frames is calculated. The spectrum is then passed in a Mel-spaced filter bank and the energies at each Mel frequency is computed. Apply the discrete cosine transform, the MFCC is obtained. The key parameters during the feature extraction process that impact the size of the model, accuracy and number of operations are:

- Number of MFCC per frame (*nMFCC*)
- Stride of the frame (S)

Number of MFCC per frame influence the number of weights in the recurrent layer. The stride decides the number of frames that needs to process per inference. To max out the efficiency and accuracy, having a small $\mathrm{nMFCC} \times \mathrm{S}$ is required [3]

2.3 Neural Network Architectures for KWS

In the past KWS systems the Hidden Markov Models was used to represent both keyword and the background sound. [6] However, with the fast development of neural networks, architecture such as DNN and RNN based KWS system out-performed the HMM models and become the mainstream method to implement the system. [7] This section provides an overview of different Neural Network architectures that have been used in implementing the KWS systems.

2.3.1 Deep Neural Network (DNN) based Keyword Spotting System

DNN is a feed-forward Neural Network consists of fully connected layers and activation layers. A typical DNN applied to a KWS system usually contain 3 parts as shown in Figure 2: (i) model for feature extraction (ii) DNN to encode sound features to abstract parameters (iii) some posterior handling method (e.g. CTC method) [8]. DNN requires a flattened feature matrix input. It will then pass

the input into fully connected hidden layers. Usually, each hidden layer is followed by a linear unit (ReLU) based activation function. [3] The output then feed into a SoftMax layer to compute the probabilities of the keywords which are used for posterior handling.

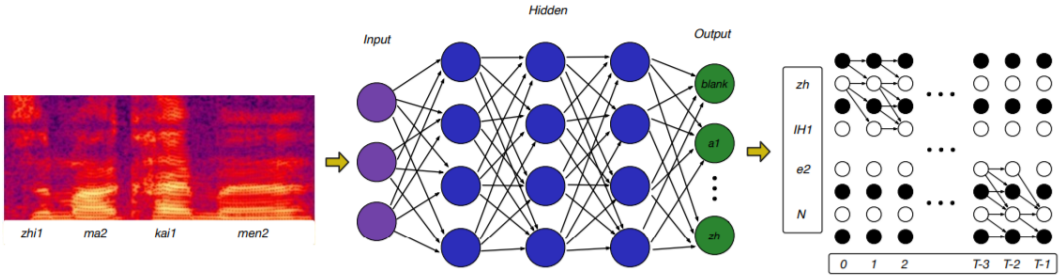

Figure 2 DNN architecture in KWS

2.3.2 Convolutional Neural Network (CNN) based Keyword Spotting System

Unlike DNN, CNN performs a 2-D convolution over the input time domain and the spectral domain features [9]. The output of convolution layer pass into batch normalization [10], activation functions and max pooling layers which reduces features' dimensions as shown in Figure 3 [9]. The batch normalization parameters can be combined as the weights of convolution layer. Batch normalization technique will be discussed further in section 3.1.4.

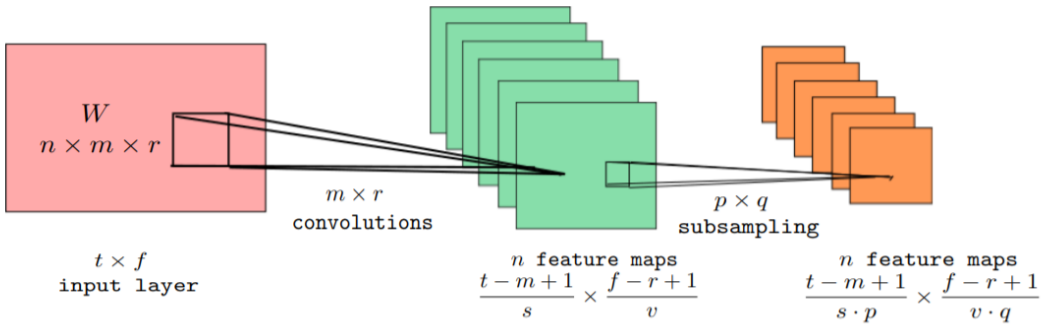

Figure 3 CNN architecture consisting of a convolutional and max-pooling layer. [9]

2.3.3 Recurrent Neural Network (RNN) based Keyword Spotting System

Similar to CNN, RNN study the local temporal and spectral correlation in the speech features. 'Gating' mechanism in RNN can help to capture dependencies. The input of RNN at each time step *t* is the feature vector $f_t \in R^F$ concatenate with the previous output at *t-1*. [1] RNN models have less parameters compared

to CNNs as the weights in RNN networks is reused across all the time steps. Layer normalization in RNN is similar to the batch normalization in CNNs which adds efficiency in training process.

2.3.4 Bidirectional Recurrent Neural Network based Keyword Spotting System

One of the drawbacks of standard RNN architectures is that they only exploit the previous context. In KWS systems, the whole utterances are transcribed at once, future context requires to be considered as well. Bidirectional Recurrent Neural Network (BRNNs) achieve this by adopting two sperate hidden layers and process data in both forward and backward directions. The output of this is then pipe into the same output layer as demonstrated in Figure 4 [11]. A BRNN architecture takes a forward sequence $\vec{h}$ and a backward sequence $\overleftarrow{h}$ connects to the same output sequence *y* by iterating both backward layer and forward layer from $t = T$ to 1 and $t = 1$ to T respectively and then compute the output layer as follows [12] [1]:

$$\overrightarrow{h_t} = H\left(W_{x\vec{h}}x_t + W_{\vec{h}\vec{h}}\vec{h}_{t-1} + b_{\vec{h}}\right)$$

$$\overleftarrow{h_t} = H\left(W_{x\overleftarrow{h}}x_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t-1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{x\vec{h}}x_t\vec{h}_t + W_{x\overleftarrow{h}}x_t\overleftarrow{h}_t + b_{\mathrm{o}}$$

The bidirectional LSTM is obtained by combining BRNN with LSTM [13] which can access long-range context in both forward and backward directions [12].

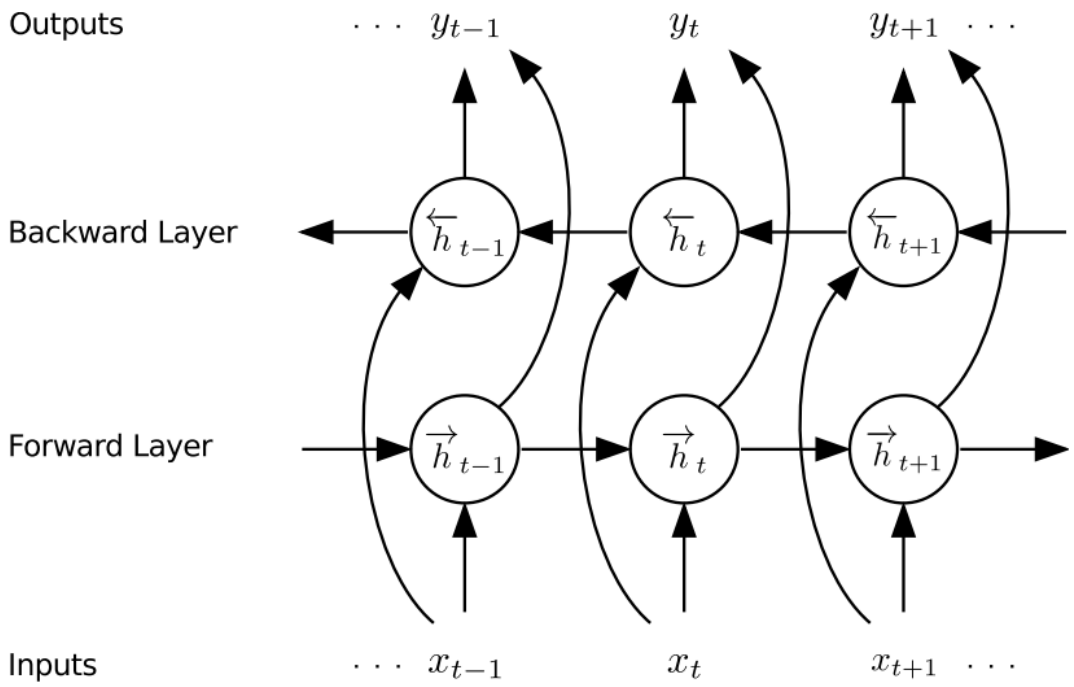

Figure 4 Bidirectional Recurrent Neural Network [13]

2.3.5 Convolutional Recurrent Neural Network (CRNN) based Keyword Spotting System

The hydration of RNN and CNN is the CRNN which consider both the local correlation and the global dependencies in speech signal using recurrent layers. [14] The model starts with a convolution layer, the output is piped to an RNN to map the information as shown in Figure 5 and 6. Due to the bi-directional and multistage nature, the network learning capability is better than the RNN and CNN. Instead of using LTSMs, it adopts the Gated recurrent units (GRU) as the base component for the recurrent layers which uses fewer parameters and have provide better convergence in the experiments. [3]

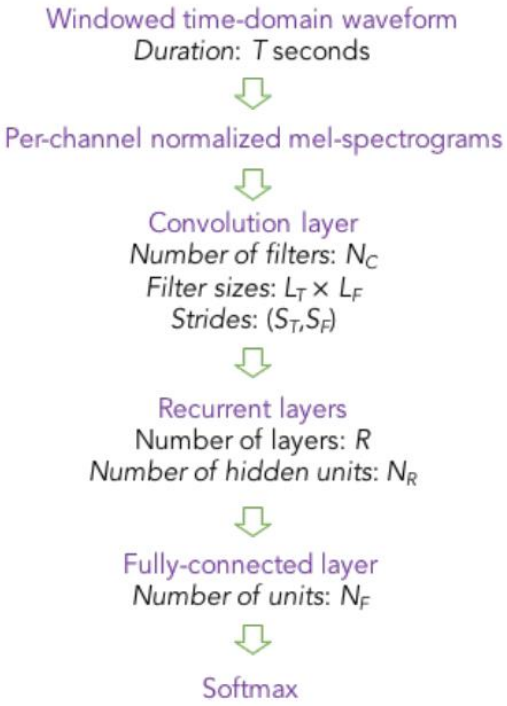

Figure 5 End-to-end CRNN architecture for KWS [9]

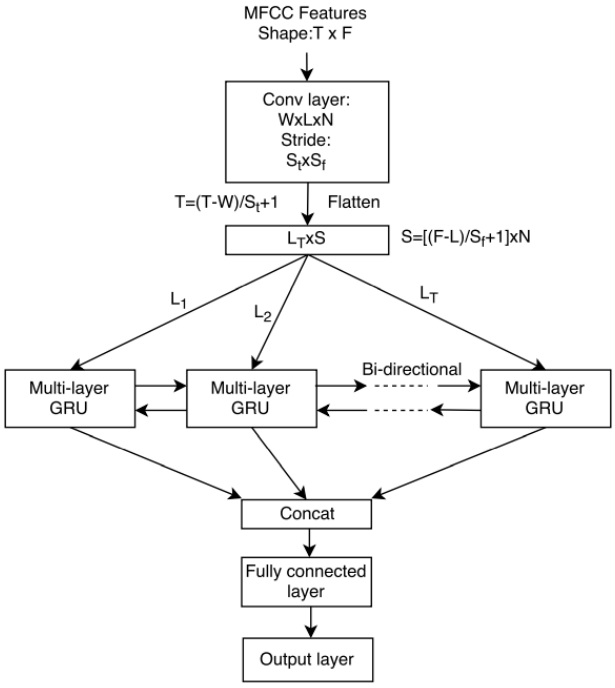

Figure 6 CRNN model [3]

2.3.6 Depth wise Separable Convolutional Neural Network (DS-CNN) based Keyword Spotting System

DS-CNN use a 2-D filter to convolve each channel in the input feature map and combine the output by using pointwise convolutions as shown in Figure 7 [3]. This approach enables a deeper architecture to be implemented in a resource constrained device such as a microcontroller through decomposing the 3-D convolution into 2-D by 1-D which reduce the number of parameters and operations and add efficiency in the computation phase [15].

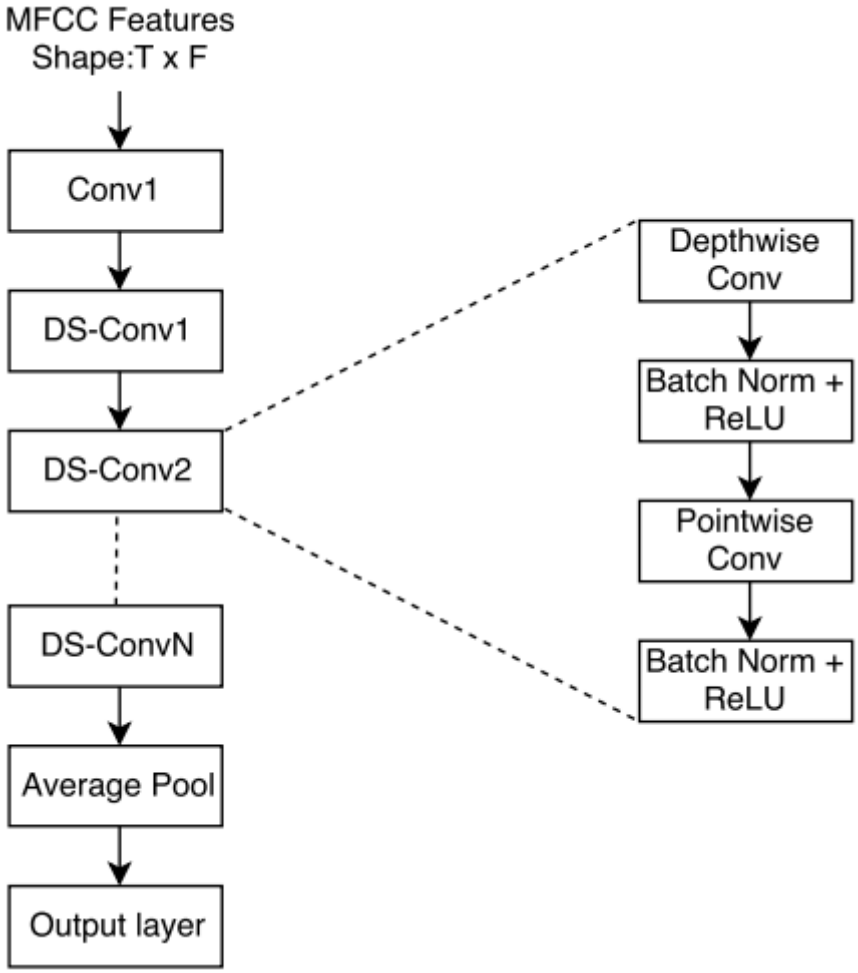

Figure 7 Depth wise separable CNN model [3]

2.3.7 Time delay neural networks (TDNN) based Keyword Spotting System

A TDNN can be pictured as a DNN model moving along the time. [16] At each time step, the contiguous features *w* is the input of the network which is a T by D matrix. T is the length of feature sequence and D is the feature dimension. TDNN output the vector and move forward *k* steps to obtain the result of every time steps. [17] Figure 8 demonstrate the structure of TDNN. The input matrix *w* input to the first layer: TDNN-SUB which is the subsampling layer. This layer

reduces input sequence length. The second layer is SWSA which has a feedforward structure as shown in the right side of Figure 8. Finally, after 2 TDNN, the SoftMax at top will output the posteriors of each keyword.

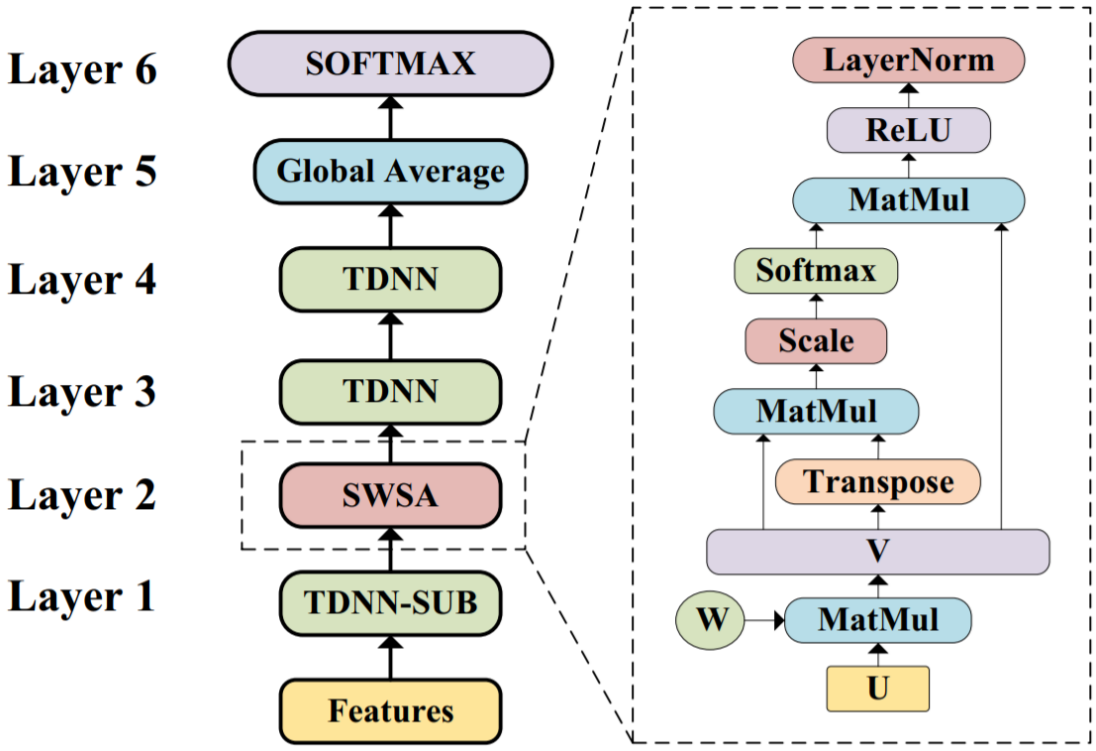

Figure 8 TDNN model. [16]

2.3.8 Deep Residual Network (ResNet) based Keyword Spotting System

Research has shown that network depth is crucial in terms of the network performance. However, stacking more layers into the network may lead to degradation: with the network depth increasing, accuracy will get saturated and then degrades rapidly. Adding more layers into the neural network architecture may eventually leads to a higher training error which shows in He and Xiangyu's experiments in Figure 9. In order to solve this issue, the ResNet is constructed. The architecture of the ResNet is shown in Figure 10. Instead of directly extend the block to create a deeper network, He and Xiangyu created a shortcut connections which skips one or more layer which performs an identity mapping and their output are added to the output of the stack layers.

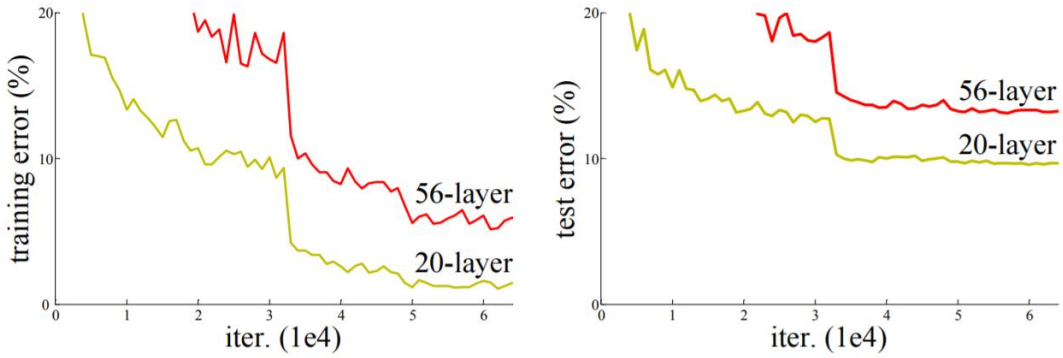

Figure 9 Training error with deep network

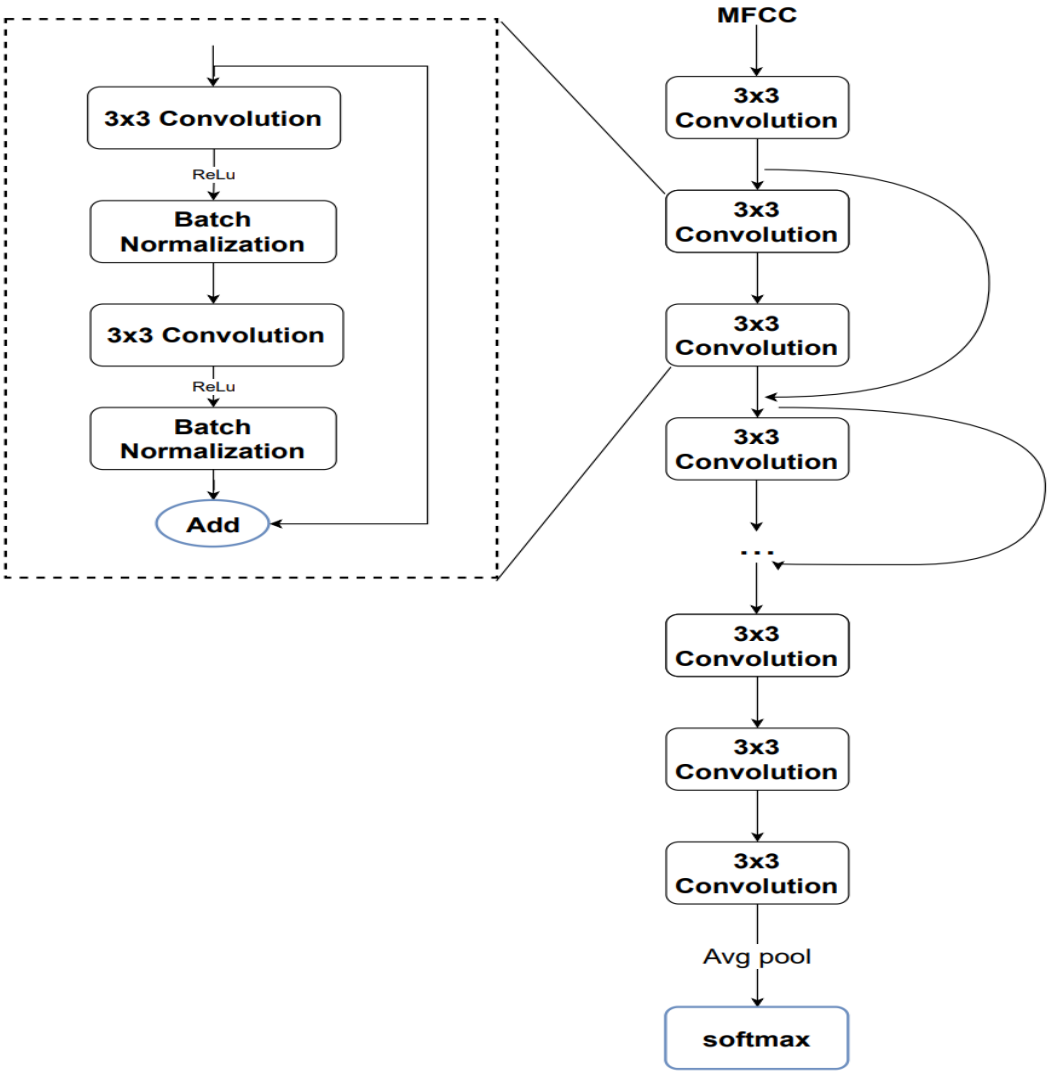

Figure 10 ResNet architecture

2.4 Posterior Handling

2.4.1 Posterior smoothing

The Neural network will produce raw posteriors which is usually noisy. Smoothing technique needs to be applied over a fixed time ω_{smooth}. Let the raw posterior to be p_{ij} and after smoothing signal be p'_{ij} then the smoothing formula would be:

$$p'_{ij} = \frac{1}{j - h_{smooth} + 1} \sum_{k=h_{smooth}}^{j} p_{ik}$$

where $h_{smooth} = \max\{1, j - \omega_{smooth} + 1\}$ which is the index of the first frame. [18]

2.4.2 Confidence

Assume having a sliding window of size ω_{max}, the confidence score at j^{th} frame is calculated as:

$$confidence = \sqrt[n-1]{\prod_{i=1}^{n-1} \max_{h_{max} \le k \le j} p'_{ik}}$$

$h_{max} = \max\{1, j - \omega_{max} + 1\}$ is the sliding window first index. [18]

Other posterior handling method is described in detail in section 3.

2.5 Datasets

2.5.1 Training Datasets

2.5.1.1 Google Speech Commands

Google Speech Commands Dataset [19] is chose to be the training dataset as it contains 65,000 one-second-long utterances files which are recorded by thousands of different people and labeled with 30 target categories [10]. The dataset is large enough to train the model to be accurate and have sufficient variety.

2.5.1.2 Mozilla Common Voice

Each entry in the Mozilla Common Voice dataset consists of a unique MP3 and corresponding text file. Many of the 13,905 recorded hours in the dataset also include demographic metadata like age, sex, and accent that can help train the accuracy of speech recognition engines. The dataset currently consists of 11,192 validated hours in 76 languages.

2.5.2 Researching and testing Datasets

The dataset used for researching and test whether the Keyword Spotting System can accurately predict memory test result is from the NHMRC funded Maintain Your Brain (MYB) project [2]. It contains the 14296 telephone audio recordings across 4 trials from 4085 participants aged between 55 and 77. Each trial consist of an episodic verbal memory test result. The participants need to listen to the same 15 target words and then repeat as many as possible promptly. The final and the fourth trial is the delayed trial. The participants were asked to recall the target words after 20 minutes without listening to them again. [20]

3. Model of choice

The KWS model this essay has adopt is the “Howl” model. It is an open-source wake word detection toolkit with native support for open speech datasets, like Mozilla Common Voice and Google Speech Commands [21]. The code base of the model can be found on Github.

3.1 Pipeline and control flow of the model

The high-level block diagram of the model is shown in Figure 11. First, the user needs to input a dataset which will be fed into the preprocessing unit. Next, the user can (optionally) select which augmentation modules to use. After that the dataset will be passed into the trainer with the hyperparameters the user chooses to use. Finally, users can run the model using the command line interface [21].

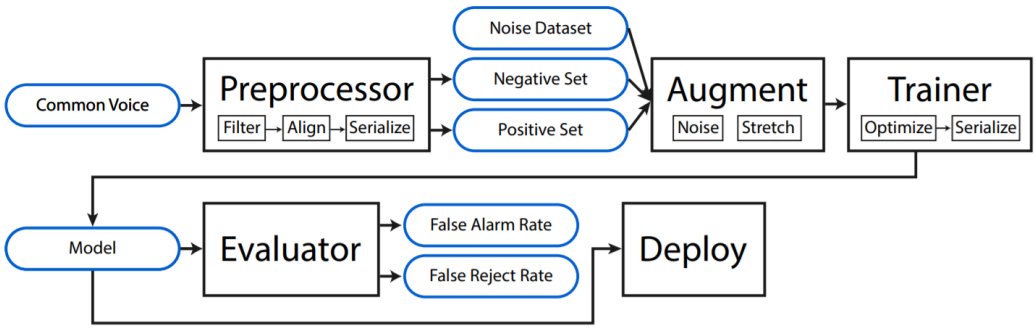

Figure 11 pipeline of the model

3.2 Implementation of the model

Howl provides implementations of CNNs, RNNs and residual CNNs. All of these models are lightweight and maintain a small footprint.

3.3 Choice of neural network and the reason behind it

The neural network this thesis chose to use is the residual CNNs. It allows user to train the model using deeper networks and has been applied into other tasks such as image recognition, speaker identification and automated speech

recognition [22]. Residual network can also maintain a high accuracy with a relatively small depth which results in a high training speed. According to Tang's experiments on the residual network the depth won't affect the test accuracy too much. Tang compare the residual network's performance against three CNN variants proposed by Sainath and Parada: trad-fpool3, tpool2 and one-stride. The accuracies of these models are shown in Figure 12, which indicates that residue network has the highest accuracy among all these variance and can maintain a decent performance only using 8 layers of the network [22].

Model	Test accuracy	Par.	Mult.
trad-fpool3	90.5% ± 0.297	1.37M	125M
tpool2	91.7% ± 0.344	1.09M	103M
one-stride1	77.9% ± 0.715	954K	5.76M
res15	95.8% ± 0.484	238K	894M
res15-narrow	94.0% ± 0.516	42.6K	160M
res26	95.2% ± 0.184	438K	380M
res26-narrow	93.3% ± 0.377	78.4K	68.5M
res8	94.1% ± 0.351	110K	30M
res8-narrow	90.1% ± 0.976	19.9K	5.65M

Figure 12 ResNet with other variance performance [22]

4. Experimental Design, Method, and Procedure

4.1 Data preprocessing

In order to start testing the performance of Keyword Spotting System on the memory test result dataset. The dataset needs to be preprocessed so that the system can be trained and evaluated.

The dataset consists of multiple wav files from all the participants. Each wav file is a 10 -30 seconds long of audio with multiple short words. Transcription of the audio file has been provided along with the audio dataset. The overall structure of the transcription is shown in Figure 13.

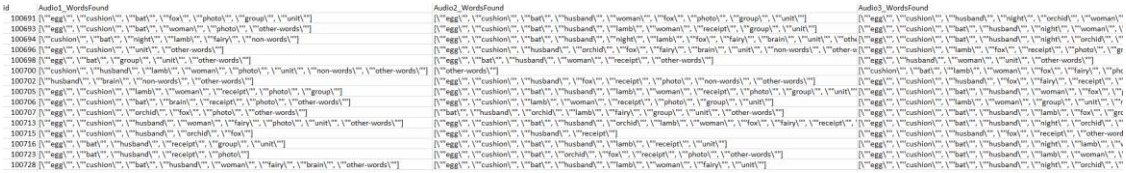

Figure 13 format of the transcription of the audio dataset

In order to allow the KWS system to properly trained and identify these words, the data set was divided in half. First half is for performance verification which doesn't need any preprocessing and is used to test the performance of the KWS model. The other half need to be truncated. Each keyword in the second half of the data needs to be extracted from the dataset and needs to be put in separated folder so that the preprocessed dataset will have the same format with the google command dataset. As the KWS model, 'howl' has built-in support for google command dataset format.

Figure 14 training set format.

To achieve this, an automated system that can divide the audio in word-level boundary is required.

4.1.1 Overall structure of the audio division system

The audio division will take the entire memory test dataset folder as input and output multiple folders which contains only one keyword (e.g. bat folder will

contain various short audio clips that only have the word ‘bat’ in it).So that these single word folders can be used for training the KWS model.

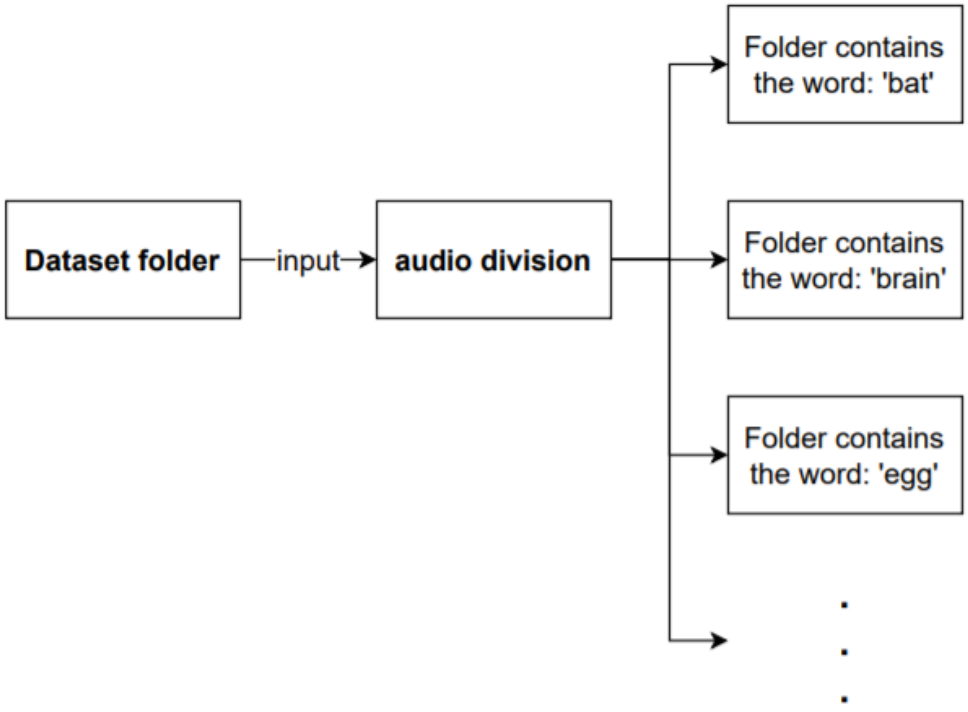

Figure 15 block diagram of the audio division system

4.1.2 Implementation of the audio division system

The Speech recognition (SR) system is used to solve this problem. Since the dataset is relatively large, a light-weight speech recognition model: ‘Vosk’ is used to increase the processing speed. Vosk is a speech recognition toolkit which supports multiple languages and is applied in chatbots, smart home appliances etc. and is compatible with python. The SR model can identify each word in the audio file and transcribe them to text and output as shown in Figure 16.

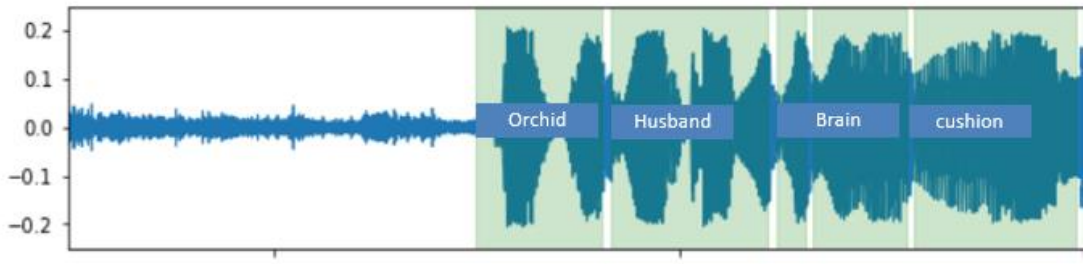

Figure 16 Speech recognition model's functionality

The SR model is imported to this [python script] in Appendix A. This script will record the word as well as the timestamps of the words so that the words can be extracted and distributed to each folder accordingly. The results is shown in Figure 17.

```
conf,end,start,word
0.618949,1.11,0.84,orchid
1.0,1.32,1.116314,husband
1.0,1.59,1.32,brain
0.411941,1.77,1.59,cushion
```

Figure 17 Sample output result

4.2 Experiment 1: Verification of proposed Keyword Spotting Model's accuracy

In order to perform tasks on the KWS model, the accuracy of the model needs to be verified. This can be achieved by reproducing the experiment proposed by Tang [21] , one of the authors of the model. The system was first examined with the google command dataset using the vocabulary: yes, no, up, down, left, right, on, off, stop and go with a batch size of 64, learning rate of 0.01 and a learning rate decay of 0.8. Then it is assessed by Hey Firefox dataset with a vocabulary of hey, fire, fox with a batch size of 16, inference threshold of 0, 300 of epochs, and an inference sequence of [0,1,2] as the system's training parameters. Noise was also added in the Firefox dataset.

4.3 Experiment 2: Performance difference between Keyword Spotting Model and Speech Recognition model

To analyze the performance of the model. The KWS needs to be trained using the preprocessed dataset.

4.3.1 Keyword Spotting Model training

The training procedure is similar to training the google command dataset. The whole dataset has a size of t and was divided into 2 sections. First fraction with a size g used for training is referred to as “training set” and the remaining fraction with a size of f = t – g is used as “validation set” [23]. The split ratio = $\frac{training\ dataset\ size}{validation\ dataset\ size} = \frac{g}{f} = \frac{8}{2}$. The split ratio will be further illustrated in section 3.4.

After dividing the set, the model is trained with the vocabulary of bat, brain, cushion, egg, fairy, fox, group, husband, lamb, night, orchid, photo, receipt, unit, woman with a learning rate of 0.01 and a learning rate decay of 0.8.

4.3.2 Performance verification on Keyword Spotting

After the model is trained. It is imported in python and its performance is verified using this python script in Appendix B. First, using this script in Appendix C to extract all the result from the transcription file and count the number of instances of each keyword as shown in Figure 18.

```
{'bat': 8919,
 'brain': 4694,
 'cushion': 12159,
 'egg': 14513,
 'fairy': 7940,
 'fox': 8145,
 'group': 7091,
 'husband': 12361,
 'lamb': 6795,
 'night': 4176,
 'orchid': 6193,
 'photo': 9318,
 'receipt': 7753,
 'unit': 11961,
 'woman': 11287}
```

Figure 18 result from transcription

Next, the script containing the KWS model is used. The script take half of the memory test dataset result as input (because the other half is used for training, mentioned in section 3.1) and count the number of responses the KWS has detected.

Finally, the overall accuracy is calculated by:

$$\frac{numer\ of\ responses\ detected\ by\ KWS}{Total\ number\ of\ responses\ in\ testing\ dataset}.$$

4.3.3 Performance verification on Speech Recognition (SR) system

Speech recognition model: Vosk is used to perform the same task as the KWS to compare the performance. The performance verification is similar to the KWS model except training is not needed as Vosk has its own pretrained model. I use this script in Appendix D which takes the test dataset folder as input and output the number of detected responses for each keyword. The overall accuracy is calculated by $\frac{numer\ of\ responses\ detected\ by\ KWS}{Total\ number\ of\ responses\ in\ testing\ dataset}$.

4.4 Experiment 3: Impact of word difference, Split ratio and dataset size on system's performance

This experiment use Control variates methods with 3 factors: word difference, split ratio and dataset size.

4.4.1 Impact of word difference

The experiment aim to find out whether different words can impact the overall performance of the system. The experiment is performed using the same amount dataset for each words in training and investigate the accuracy of each keyword.

4.4.2 Impact of Split ratio

Another important factor that affects the performance of the model is the balance of training set and validation set. The balance of these two sets is determined by the Split Ratio. Split ratio is defined as $\frac{training\ dataset\ size}{validation\ dataset\ size}$. Training dataset refers to the set of data used to determine the parameters of the model [24]. While validation dataset refers to the set that evaluates the hyperparameters of the model [25]. The experiment used the keyword "night" as the testing keyword and 2000 as the training set size and study the performance using split ratio of: 5:5, 6:4, 7:3 and 8:2.

4.4.3 Impact of dataset size

Another element that may determine the accuracy of the system is the size of the dataset [26]. In order to study this factor, The experiment is performed by using 8:2 ratio with the keyword night and dataset size 100, 500, 1000, 2000

5. Experimental Results

5.1 Experiment 1: Verification of proposed keyword spotting model's accuracy

5.1.1 Results and Analysis

The model reached an accuracy of 97% using the google command dataset as shown in Figure 19 which is similar with the paper's result.

```
model:  res8
dev_acc:  0.9679317446307738
test_acc:  0.9724945135332845
```

Figure 19 results using google command dataset

The model was then assessed using the Hey Firefox dataset which achieved an accuracy of 96% which is similar to thesis's result. The false rejection rate of the model has similar data as the paper indicated.

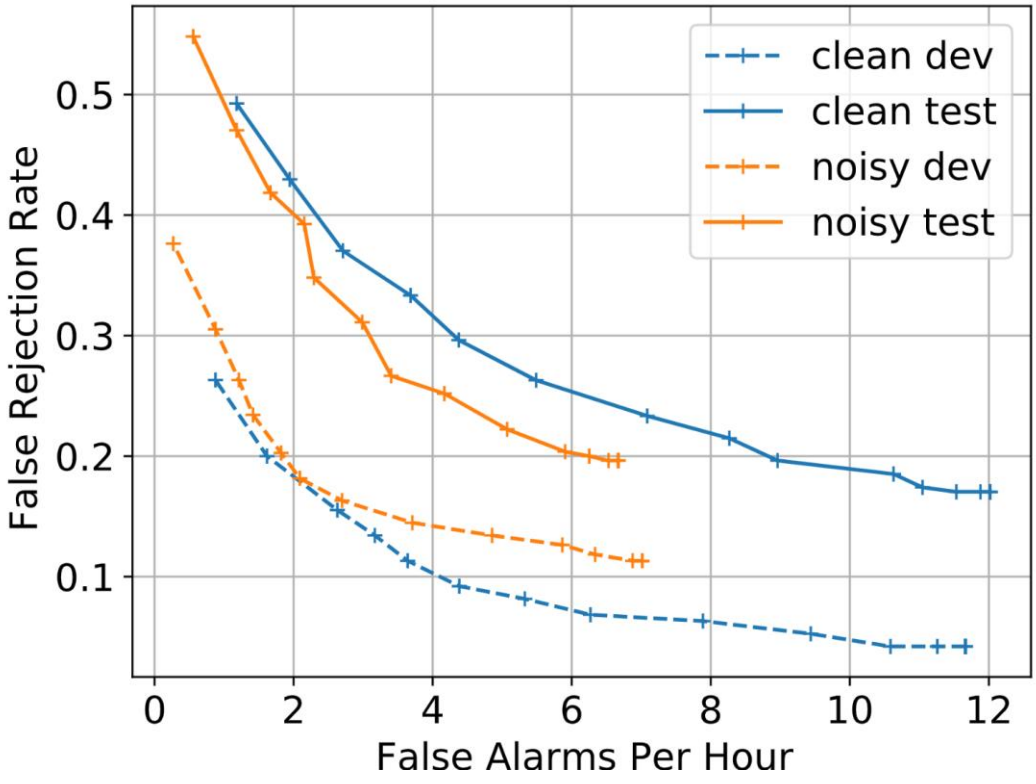

Figure 20 Firefox data result

5.1.2 Concluding Remarks

The performance of the model matches the author's result and performs reasonably well which indicates that the model is capable for handling keyword spotting tasks.

5.2 Experiment 2: Performance difference between Keyword Spotting Model and Speech Recognition model

5.2.1 Results and Analysis

5.2.1.1 Speech Recognition model identification rate

The Speech recognition system achieve an average accuracy of 28.26% as shown in Figure 21 which is not an ideal performance.

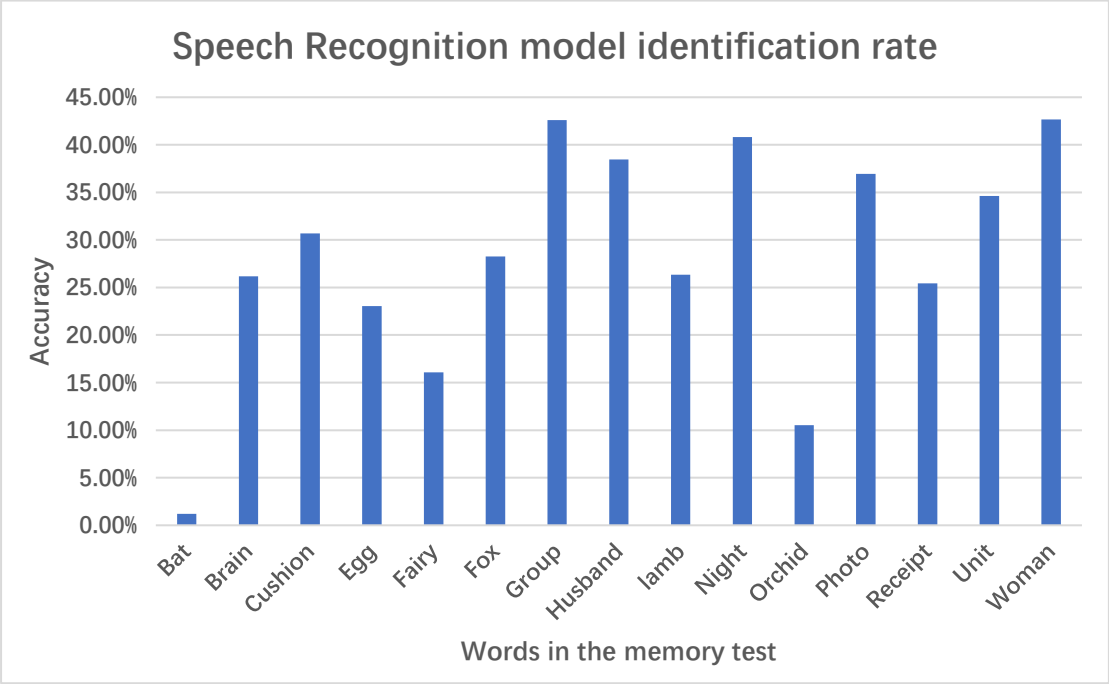

Figure 21 SR performance

5.2.1.2 Keyword Spotting System identification rate

The Speech recognition system achieve an average accuracy of 92.02% as shown in Figure 22 which is a decent performance.

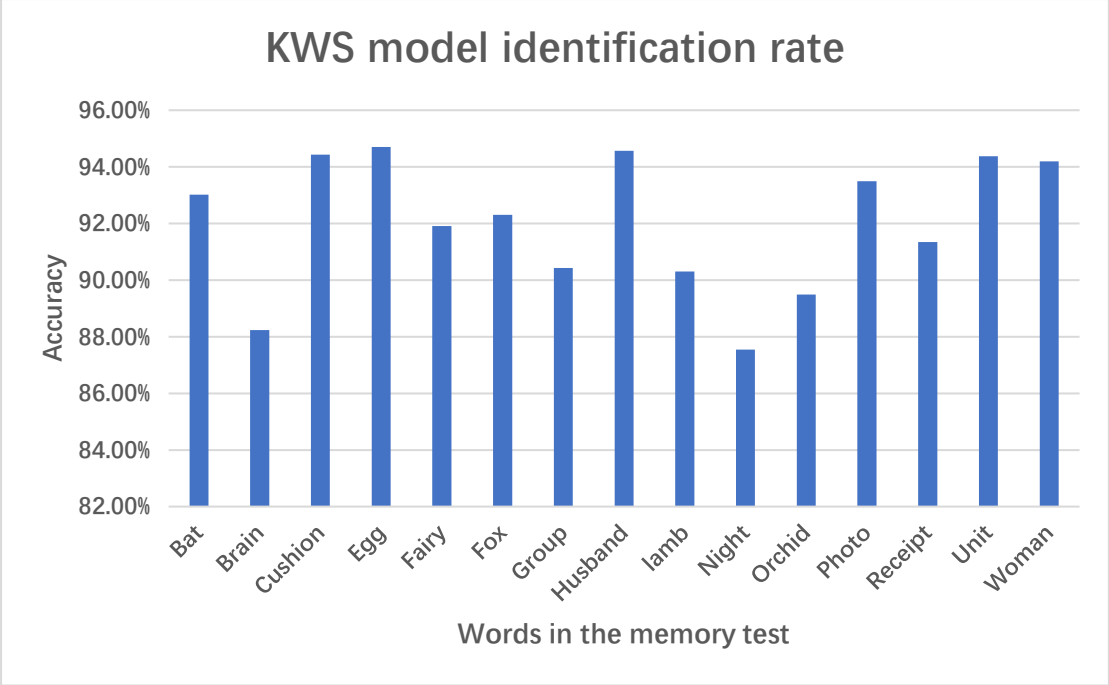

Figure 22 KWS performance

5.2.1.3 Keyword Spotting System vs Speech Recognition identification rate

As shown in Figure 23, KWS accuracy is much higher than the SR model identification rate. The reason why SR perform so poorly may be SR systems typically have more general purposes such as speech to text, real time translation etc. [27]. It has been trained using a broader dataset and the output rely on the context of the speech which is not suitable for handling this task: spotting these 15 words from a stream of audio. The audio only consists random words (has no context) and the vocabulary of the speech is quite low which makes speech recognition a weak choice in handling such tasks. The Keyword Spotting however, only used these 15 words to train its only purpose is to solve this task, which allows the model to have a better accuracy and a stronger performance.

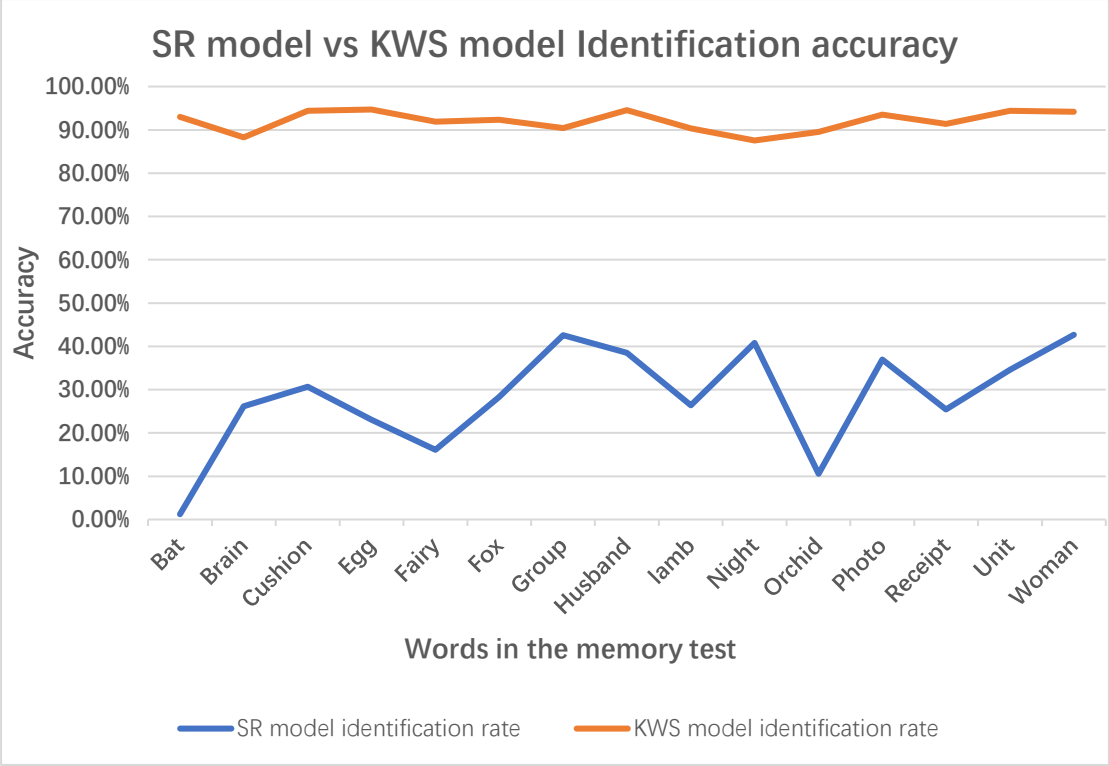

Figure 23 KWS vs SR performance

5.3 Experiment 3: Impact of word difference, Split ratio and dataset size on system's performance

5.3.1 Results and Analysis on the Impact of word difference

The result as shown in Figure 24, all the words have similar accuracy. The words with relatively low accuracy are bat, egg, lamb. These words have less phonemes which may result to less accuracy. However, the memory test dataset itself may lead to large uncertainty as it contains noises (the results were recorded through phones) and the tone of the speaker may var the result.

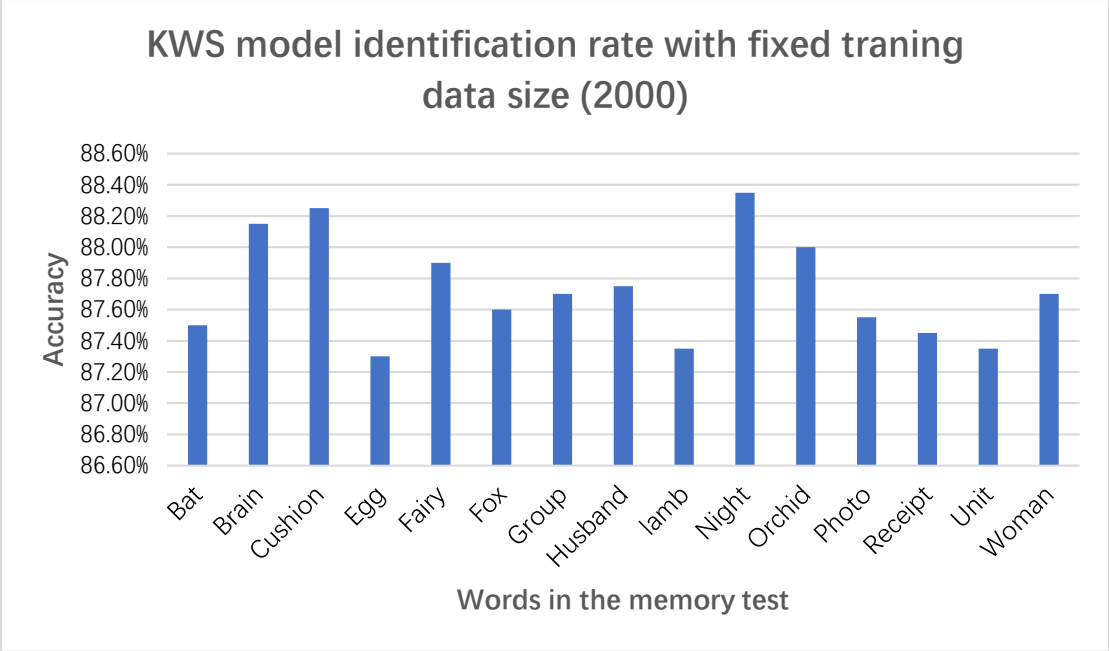

Figure 24 word difference result

5.3.2 Results and Analysis on the Impact of Split Ratio

As the result shown in Figure 25. When the split ratio is at 70:30 and 80:20 the model tends to have a better performance. This result meets the research by Anita which also indicates that 70 : 30 and 80:20 split ratio performed better when using a large dataset [28] as shown in Figure 26.

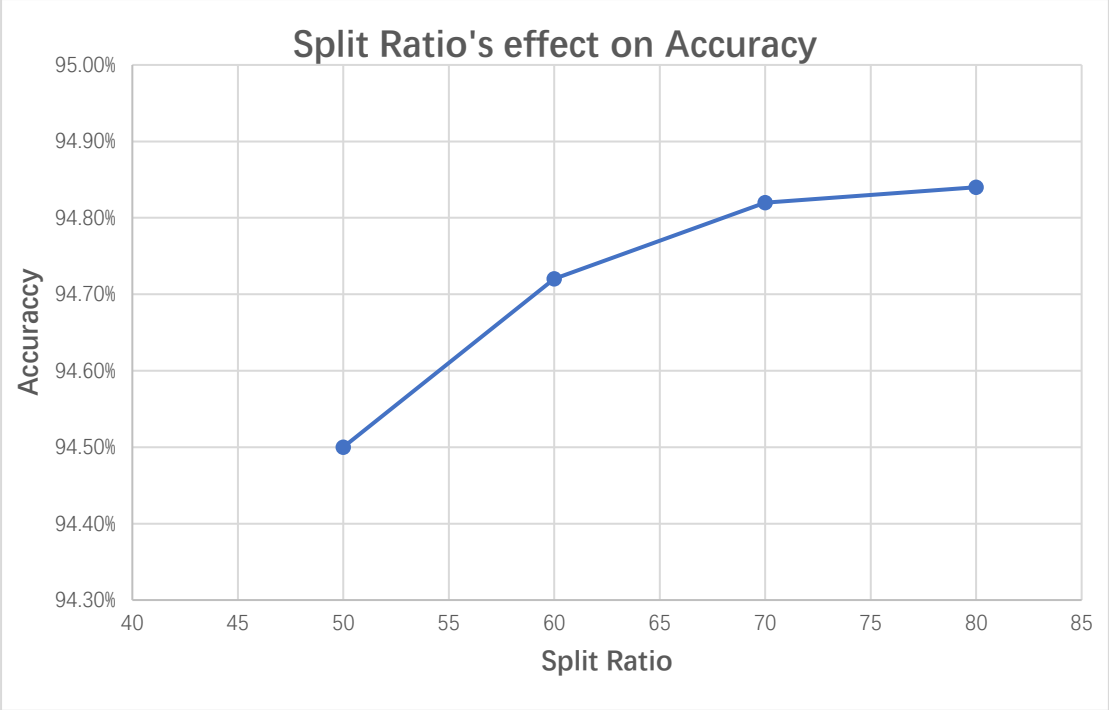

Figure 25 Split ratio's effect

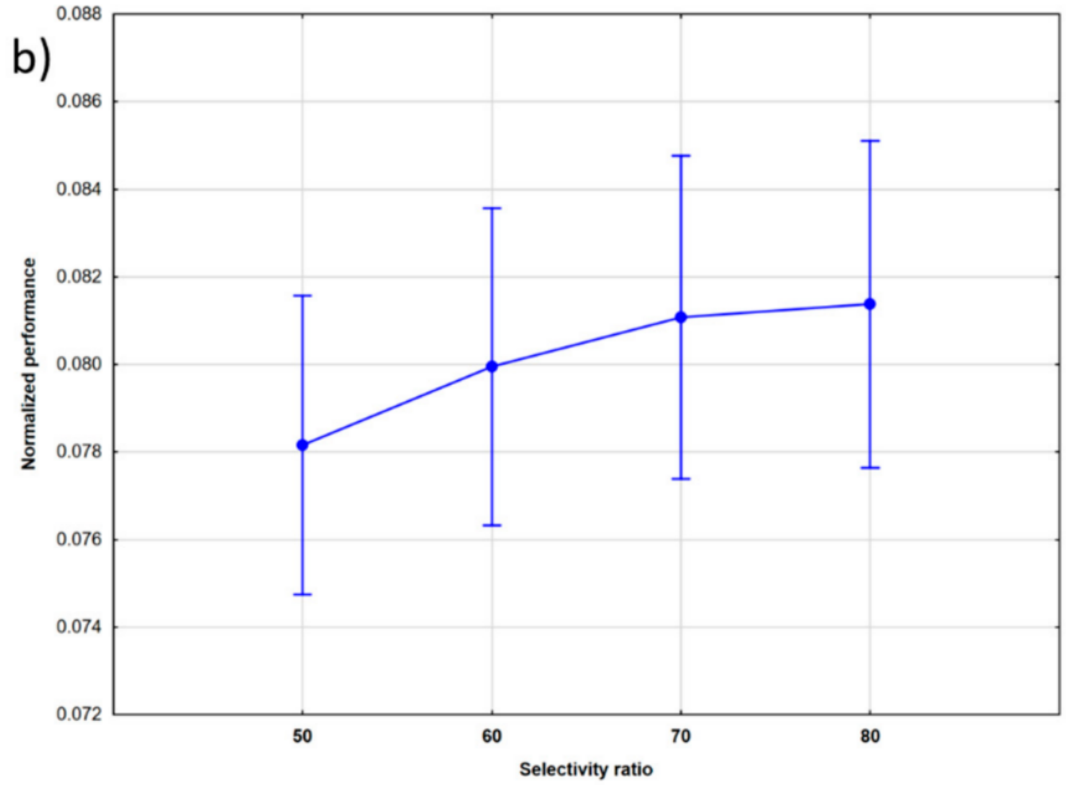

Figure 26 Anita's research result on split ratio

5.3.3 Results and Analysis on the Impact of Split Ratio and dataset size

As the result shown in Figure 27. The colored line represents the different split ratio. According to the result, the higher the dataset size, the higher the accuracy. What's more, the results show that the split ratios had a more significant effect on the modeling at bigger dataset sizes, the 80:20 split ratio's performance drastically increase when the dataset size increases.

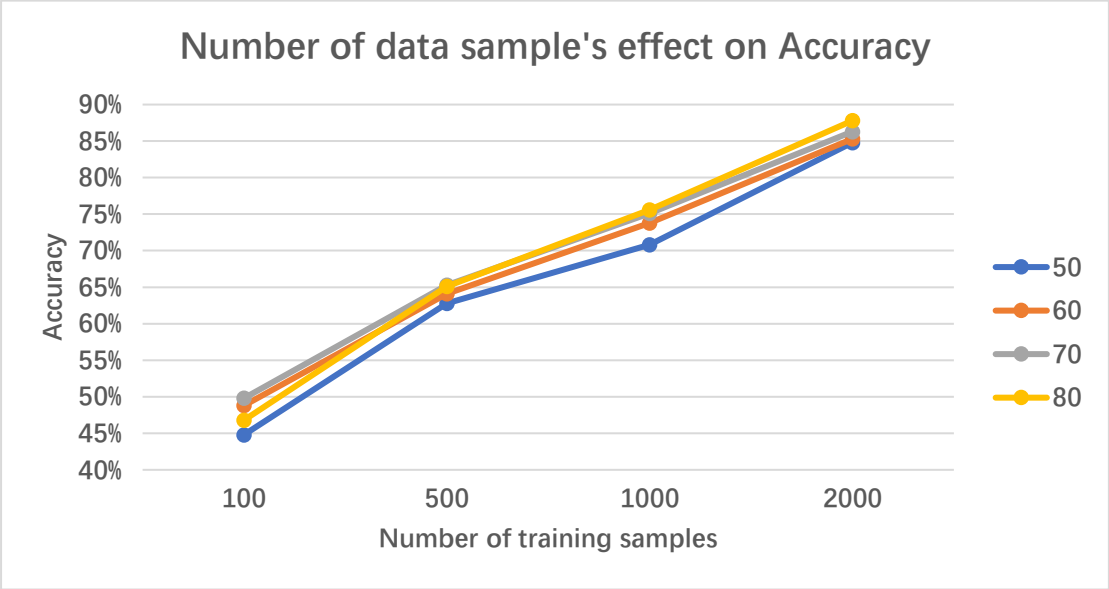

Figure 27 Impact of split ratio and dataset size

5.4 Concluding Remarks and Summary

Experiment 1 showed the model has capability in handling the keyword spotting tasks. Experiment 2 demonstrated that the keyword spotting system perform better than speech recognition model therefore it is a better choice in predicting memory test result. Experiment 3 revealed that dataset size and split ratio have significant effects during the training phases. Therefore, having a large training set and an appropriate split ratio can allow the model to predict the memory test result more accurately.

6. Conclusion

Though Keyword Spotting System is widely used on smart assistance devices like Alexa and Siri, it hardly has any medical application. This thesis details the research, design and implementation of adapting KWS for accurately predicting the memory test result.

6.1 Result for thesis aim 1: Find the suitable KWS model for the purpose of predicting the memory test result.

Research was conducted into the current neural network architecture of KWS, how they were constructed and commonly design as well as the common techniques for feature extractions and posterior handling methods. Among all these neural networks and models. Howl was chosen for its efficiency and high accuracy.

6.2 Result for thesis aim 2: Experiment the KWS model with the memory test dataset and analyze its performance

Results from the experiments indicates that KWS can perform decently on predicting the memory test result. It is a more suitable choice compared to other speech processing model such as Speech Recognition system. All the words occurred in the memory test have similar accuracy.

6.3 Result for thesis aim 3: Find out the appropriate training parameters such as the size of the dataset and split ratio to optimize the model's accuracy in predicting memory test result.

During the training process, both number of samples and the split ratio have significant impact on the performance of the model. Using a 70:30 or 80:20 split ratio with a large dataset will increase the accuracy of the model.

6.4 Future work

The work accomplished in this thesis is a tip of an iceberg of the application of KWS. In order to have a more precise and efficient system, more research and work is needed including:

- Integrate the dataset generation process into the system
- Increase noise robustness of the system
- Further classify the dataset (gender age) and study the result
- Provides easy user interface so that self-detection dementia can be realized

Bibliography

[1] A. G. a. J. S. S. Fernández, "An Application of Recurrent Neural Networks to Discriminative Keyword Spotting," 2007.

[2] e. a. 9. Heffernan, "Maintain Your Brain: Protocol of a 3-Year Randomized Controlled Trial of a Personalized Multi-Modal Digital Health Intervention to Prevent Cognitive Decline Among Community Dwelling 55 to 77 Year Olds," Journal of Alzheimer's Disease: JAD, 2018.

[3] N. S. L. L. a. V. C. Y. Zhang, "Hello Edge: Keyword Spotting on Microcontrollers," 2018.

[4] P. P. Ippolito, "Feature Extraction Techniques," 2019.

[5] H. L. Tomi Kinnunen, "An Overview of Text-Independent Speaker Recognition:," 2009.

[6] M. W. M. S. S. P. G. F. a. S. V. G. Tucker, "Model Compression Applied to Small-Footprint Keyword Spotting," 2016.

[7] R. T. a. J. Lin, "Deep Residual Learning for Small-Footprint Keyword Spotting," 2018.

[8] X. L. J. Z. Zhiming Wang, "SMALL-FOOTPRINT KEYWORD SPOTTING USING DEEP NEURAL NETWORK AND CONNECTIONIST TEMPORAL CLASSIFIER," 2017.

[9] C. P. Tara N. Sainath, "Convolutional Neural Networks for Small-footprint Keyword Spotting," 2015.

[10] S. I. a. C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[11] A. M. A. a. H. G. Graves, "Speech recognition with deep recurrent neural networks," ProcICASSP, 2013.

[12] A. G. a. N. Jaitly, "Towards End-to-End Speech Recognition with Recurrent Neural Networks," 2015.

[13] M. a. N. H. Bisani, "Open vocabulary speech recognition with flat hybrid models," INTER-SPEECH, 2005.

[14] Ö. A. e. al, "Convolutional Recurrent Neural Networks for Small-Footprint Keyword Spotting," Interspeech , 2017.

[15] F. Chollet., " Deep learning with depthwise separable convolutions," 2016.

[16] J. ,. J. T. Z. W. Ye Bai, "A Time Delay Neural Network with Shared Weight Self-Attention for Small-Footprint Keyword Spotting," 2019.

[17] D. S. Y. G. Ming Sun, "Compressed time delay neural network for small-footprint keyword spotting," 2017.

[18] C. P. a. G. H. G. Chen, "Small-footprint keyword spotting using deep neural networks," 2014.

[19] P. Warden, "Launching the speech commands," 2017.

[20] B. A. M. I. a. M. V. Kaavya Sriskandaraja, "Subject Independent Dementia Risk Prediction Models Using Paralinguistic and Memory Test Features with Feature Warping," 2020.

[21] J. L. A. R. J. C. Raphael Tang, "Howl: A Deployed, Open-Source Wake Word Detection System," 2020.

[22] R. T. a. J. Lin, "DEEP RESIDUAL LEARNING FOR SMALL-FOOTPRINT KEYWORD SPOTTING," Sep. 2018.

[23] I. Guyon, "A scaling law for the validation-set training-set size ratio," California, 1997.

[24] B. D. Ripley, "Patter Recognition via Neural Netwoeks," Cambridge university press., 2007.

[25] J. D. W. T. H. &. R. T. Gareth, "An introduction to statistical learning: with applications in R. Spinger.," 2013.

[26] J. G. A. Barbedo, " Impact of dataset size and variety on the effectiveness of deep learning and transfer learning for plant disease classification," 2018.

[27] D. A. &. R. R. C. Reynolds, "Robust text-independent speaker identification using Gaussian mixture speaker models.," 1995.

[28] A. D. B. a. K. H. Rácz, "Effect of Dataset Size and Train/Test Split Ratios in QSAR/QSPR Multiclass Classification," 2021.

[29] M. J. S. G. K. a. R. B. L. H. Buschke, "Diagnosis of early dementia by the Double Memory Test: Encoding specificity improves diagnostic sensitivity and specificity," 1997.

[30] C. L. a. A. Hannun, "An End-to-End Architecture for Keyword Spotting and Voice

Activity Detection," 2016.

[31] M. a. P. K. K. Schuster, "Bidirectional Recurrent Neural Networks," IEEE Transactions on Signal Processing, 1997.

[32] F. S. N. a. S. J. Gers, "Learning Precise Timing with LSTM Recurrent Networks," Journal of Machine Learning Research, 2002.

[33] G. P. P. B. Y. Z. a. Y. B. Cesar Laurent, "Batch normalized recurrent neural networks," 2015.

[34] R. A. E. B. C. C. J. C. B. C. J. C. M. C. A. C. G. D. E. E. J. E. L. F. C. F. T. H. A. H. B. J. P. L. Dario Amodei, "END-TO-END SPEECH RECOGNITION IN ENGLISH AND MANDARIN," 2016.

[35] A. G. a. J. S. S. Fernández, "An Application of Recurrent Neural Networks to Discriminative Keyword Spotting," Artificial Neural Networks, 2007.

[36] V. S. T. Samuel Myer, "Efficient keyword spotting using time delay neural networks," 2018.

Appendix A: python script for preprocessing audio file

```
import sys

import wave, os, glob

import subprocess

import json

import math

import vosk

import librosa

import numpy

import pandas

from pydub import AudioSegment

from pathlib import Path

from os.path import basename


wordlist = ["egg", "cushion", "bat", "husband", "night", "orchid", "lamb", "woman", "fox", "fairy", "brain",
"receipt", "photo", "group", "unit"]


def extract_words(res):

    jres = json.loads(res)

    if not 'text' in jres:

        return []

    words = jres['text']

    return words


def transcribe_words(recognizer, bytes):

    results = []


    chunk_size = 4000

    for chunk_no in range(math.ceil(len(bytes)/chunk_size)):
```

```
        start = chunk_no*chunk_size
        end = min(len(bytes), (chunk_no+1)*chunk_size)
        data = bytes[start:end]

        if recognizer.AcceptWaveform(data):
            words = extract_words(recognizer.Result())
            words = words.split(" ")
            results.extend(words)

    return results

def process_model(audio_path):

    vosk.SetLogLevel(-1)

    audio_path = audio_path

    model_path = 'vosk-model-en-us-0.22'
    sample_rate = 16000

    audio, sr = librosa.load(audio_path, sr=16000)

    # convert to 16bit signed PCM (pulse code modulation), as expected by VOSK
    int16 = numpy.int16(audio * 32768).tobytes()
    if not os.path.exists(model_path):
        raise ValueError(f"Could not find VOSK model at {model_path}")

    model = vosk.Model(model_path)
    recognizer = vosk.KaldiRecognizer(model, sample_rate)
```

```
    res = transcribe_words(recognizer, int16)
    # print("-----result for participant: {} is-----".format(audio_path))
    # print(res)
    participantName = audio_path.split('/')[1]
    print("-----For participant {} -----".format(participantName))
    for word in res:
        if word in wordlist:
            print(word + ' is spotted in the result')

    # output to txt as dictionary of dic
    # {id: {1 : [word1, 2 ,3]}
    #      {2: []}
    # }
    # use id as key

# -----Folder Processing-----
def nesting(path):
    """ counts how often `os.path.split` works on `path` """
    c = 0
    head = tail = path
    while head and tail:
        head, tail = os.path.split(head)
        c +=1
    return c

def longest_path( paths ):
        return max(paths, key=nesting)
```

```
def find_leafes( root , Plist):
    """ finds folders with no subfolders """
    for root, dirs, files in os.walk(root):
        if not dirs: # can't go deeper
            Plist.append(root)
            yield root

def main():
    pathList = []
    longest_path(find_leafes('test-folder', pathList))
    # print(pathList)

    for path in pathList:
        for filename in glob.glob(os.path.join(path, '*.wav')):
            # print(filename)
            process_model(filename)

if __name__ == '__main__':
    main()
```

Appendix B: python script for verifying performance of the system

```
import sys

import wave, os, glob

import subprocess

import json

import math

import librosa

import numpy

# import pandas

from pydub import AudioSegment

from pathlib import Path

from os.path import basename

import pandas as pd

import pprint

# import json


wordlist = ["egg", "cushion", "bat", "husband", "night", "orchid", "lamb", "woman", "fox",
"fairy", "brain", "receipt", "photo", "group", "unit"]

outputDict = {}


# Data generation

# format: {ID :[df1,df2,df3,df4] }


# -----excel processing-----

def formatWord(str):
```

```
    temp = '\\\"\"' + str + '\\\"\"'
    return temp

def parseList(lst):
    res = lst.split(',')
    # print(res)
    res[0] = res[0].strip('[')
    res[-1] = res[-1].strip(']')
    # print(res)
    return res



def extract_words(res):
    jres = json.loads(res)
    if not 'text' in jres:
        return []
    words = jres['text']
    return words

def transcribe_words(recognizer, bytes):
    results = []

    chunk_size = 4000
    for chunk_no in range(math.ceil(len(bytes)/chunk_size)):
        start = chunk_no*chunk_size
        end = min(len(bytes), (chunk_no+1)*chunk_size)
```

```
        data = bytes[start:end]

        if recognizer.AcceptWaveform(data):

            words = extract_words(recognizer.Result())
            words = words.split(" ")
            if 'eg' in words :
                words[words.index('eg')] = 'egg'
            if 'aig' in words:
                words[words.index('aig')] = 'egg'
            if 'folks' in words :
                words[words.index('folks')] = 'fox'
            results.extend(words)

    return results

countDict = {}
#  {word: count}

def process_model(audio_path):

    vosk.SetLogLevel(-1)

    audio_path = audio_path

    model_path = 'howl'
```

```
sample_rate = 16000


audio, sr = librosa.load(audio_path, sr=16000)


# convert to 16bit signed PCM (pulse code modulation), as expected by VOSK
int16 = numpy.int16(audio * 32768).tobytes()
if not os.path.exists(model_path):
    raise ValueError(f"Could not find VOSK model at {model_path}")


model = vosk.Model(model_path)
recognizer = vosk.KaldiRecognizer(model, sample_rate)


res = transcribe_words(recognizer, int16)
# print("-----result for participant: {} is-----".format(audio_path))
# print(res)


for word in res:
    count = 0
    if word in wordlist:
    #     print(word + ' is spotted in the result')
        count += 1
        if word in countDict.keys():
            countDict[word] += count
        else:
            countDict[word] = count
```

```
# -----Folder Processing-----
def nesting(path):
    """ counts how often `os.path.split` works on `path` """
    c = 0
    head = tail = path
    while head and tail:
        head, tail = os.path.split(head)
        c +=1
    return c

def longest_path( paths ):
        return max(paths, key=nesting)

def find_leafes( root , Plist):
    """ finds folders with no subfolders """
    for root, dirs, files in os.walk(root):
        if not dirs: # can't go deeper
            Plist.append(root)
            yield root



def main():
    pathList = []
```

```
    # longest_path(find_leafes('test-folder', pathList)) # TODO change folder perform final
test using > output.txt
    longest_path(find_leafes('participant', pathList))
    # print(pathList)


    for path in pathList:
        for filename in glob.glob(os.path.join(path, '*.wav')):
            # print(filename)
            participantName = filename.split('/')[1]
            wavName = filename.split('/')[-1]
            # print("-----For participant: {}, wavfile: {}-----".format(participantName, wavName))
            process_model(filename)
    pprint.pprint(countDict)


if __name__ == '__main__':
    main()
```

Appendix C python script for extracting keyword instances from csv file

```
import pandas as pd
import pprint
import json


def parseList(lst):
    res = lst.split(',')
    # print(res)
    res[0] = res[0].strip('[')
    res[-1] = res[-1].strip(']')
    # print(res)
    return res


# Data generation
# format: {ID :[df1,df2,df3,df4] }


data = pd.read_excel (r'dataRes.xlsx')


participants = pd.DataFrame(data, columns= ['id'])
df1 = pd.DataFrame(data, columns= ['Audio1_WordsFound'])
df2 = pd.DataFrame(data, columns= ['Audio2_WordsFound'])
df3 = pd.DataFrame(data, columns= ['Audio3_WordsFound'])
df4 = pd.DataFrame(data, columns= ['Audio4_WordsFound'])
outputDict = {}
```

```
# print(df1.values.tolist()[0])

# A single elem list of string

# -> [\""egg\"", \""cushion\"", \""bat\"", \""fox\"", \""photo\"", \""group\"", \""unit\""]

# print(parseList(df1.values.tolist()[0][0])[1])

# i= 0

# for elem in participants.values.tolist():

#     outputDict[elem[0]] = df1.values.tolist()[i]

#     outputDict[elem[0]].append(df2.values.tolist()[i])

#     outputDict[elem[0]].append(df3.values.tolist()[i])

#     outputDict[elem[0]].append(df4.values.tolist()[i])

#     i += 1


# Print the result

# pprint.pprint(outputDict)


# count number of words and output to txt

wordlist = ["egg", "cushion", "bat", "husband", "night", "orchid", "lamb", "woman", "fox",
"fairy", "brain", "receipt", "photo", "group", "unit"]

countDict = {}

for word in wordlist:

    countDict[word] = 0

# f = open('resCount.txt', 'a')

# f.write('readme')


# f.write('\n')

for word in wordlist:

    temp = '\\\"\"' + word + '\\\"\"'
```

```
# print(temp)
count = 0
# #traverse the dict
# for key in outputDict.keys:
#     for lst in outputDict[key]:
#         for elem in lst:
#             if temp in parseList(elem[0]):
#                 count += 1
# print(count)

for elem in df1.values.tolist():
    if temp in parseList(elem[0]):
        count += 1
countDict[word] += count

count = 0
for elem in df2.values.tolist():
    if temp in parseList(elem[0]):
        count += 1
countDict[word] += count

count = 0
for elem in df3.values.tolist():
    if temp in parseList(elem[0]):
        count += 1
countDict[word] += count
```

```
    count = 0
    for elem in df4.values.tolist():
        if temp in parseList(elem[0]):
            count += 1
    countDict[word] += count
pprint.pprint(countDict)
# f.write(json.dumps(countDict))
```

Appendix D SR model performance verification script

```
import sys

import wave, os, glob

import subprocess

import json

import math

import vosk

import librosa

import numpy

# import pandas

from pydub import AudioSegment

from pathlib import Path

from os.path import basename

import pandas as pd

import pprint

# import json


wordlist = ["egg", "cushion", "bat", "husband", "night", "orchid", "lamb", "woman", "fox",
"fairy", "brain", "receipt", "photo", "group", "unit"]

outputDict = {}


# Data generation

# format: {ID :[df1,df2,df3,df4] }


# -----excel processing-----

def formatWord(str):
```

```
    temp = '\\\"\"' + str + '\\\"\"'
    return temp

def parseList(lst):
    res = lst.split(',')
    # print(res)
    res[0] = res[0].strip('[')
    res[-1] = res[-1].strip(']')
    # print(res)
    return res



def extract_words(res):
    jres = json.loads(res)
    if not 'text' in jres:
        return []
    words = jres['text']
    return words

def transcribe_words(recognizer, bytes):
    results = []

    chunk_size = 4000
    for chunk_no in range(math.ceil(len(bytes)/chunk_size)):
        start = chunk_no*chunk_size
        end = min(len(bytes), (chunk_no+1)*chunk_size)
```

```
        data = bytes[start:end]

        if recognizer.AcceptWaveform(data):

            words = extract_words(recognizer.Result())
            words = words.split(" ")
            if 'eg' in words :
                words[words.index('eg')] = 'egg'
            if 'aig' in words:
                words[words.index('aig')] = 'egg'
            if 'folks' in words :
                words[words.index('folks')] = 'fox'
            results.extend(words)

    return results

countDict = {}
#  {word: count}

def process_model(audio_path):

    vosk.SetLogLevel(-1)

    audio_path = audio_path

    model_path = 'vosk-model-small-en-us-0.15'
```

```
sample_rate = 16000


audio, sr = librosa.load(audio_path, sr=16000)


# convert to 16bit signed PCM (pulse code modulation), as expected by VOSK
int16 = numpy.int16(audio * 32768).tobytes()
if not os.path.exists(model_path):
    raise ValueError(f"Could not find VOSK model at {model_path}")


model = vosk.Model(model_path)
recognizer = vosk.KaldiRecognizer(model, sample_rate)


res = transcribe_words(recognizer, int16)
# print("-----result for participant: {} is-----".format(audio_path))
# print(res)


for word in res:
    count = 0
    if word in wordlist:
    #     print(word + ' is spotted in the result')
        count += 1
        if word in countDict.keys():
            countDict[word] += count
        else:
            countDict[word] = count
```

```
# -----Folder Processing-----
def nesting(path):
    """ counts how often `os.path.split` works on `path` """
    c = 0
    head = tail = path
    while head and tail:
        head, tail = os.path.split(head)
        c +=1
    return c

def longest_path( paths ):
        return max(paths, key=nesting)

def find_leafes( root , Plist):
    """ finds folders with no subfolders """
    for root, dirs, files in os.walk(root):
        if not dirs: # can't go deeper
            Plist.append(root)
            yield root



def main():
    pathList = []
```

```
    # longest_path(find_leafes('test-folder', pathList)) # TODO change folder perform final
test using > output.txt
    longest_path(find_leafes('participant', pathList))
    # print(pathList)


    for path in pathList:
        for filename in glob.glob(os.path.join(path, '*.wav')):
            # print(filename)
            participantName = filename.split('/')[1]
            wavName = filename.split('/')[-1]
            # print("-----For participant: {}, wavfile: {}-----".format(participantName, wavName))
            process_model(filename)
    pprint.pprint(countDict)


if __name__ == '__main__':
    main()
```