1. Introduction

Automatic speech recognition (ASR) systems are being used more widely on consumer devices such as smartphones, smart speakers and computer operating systems. Many of these devices can be controlled by voice only, without touching the screen or performing other operations. Some systems, including Google Assistant and Apple’s Siri, start up when a specific “magic word” is detected, allowing the devices to be operated verbally, without touching them. Magic Word Detection (MWD), also known as “wake-up word detection”, is the technology which makes this possible, and it has become essential for the hands-free operation of electronic devices.

MWD systems attempt to detect these magic words from continuous audio streams of speech, thus MWD systems function as discriminators for the hands-free operation of such devices. Because MWD systems need to run constantly in order to detect Magic Words at any time, many studies have focused on the development of a small-footprint system. In this paper, we propose a novel, small-footprint MWD method which uses a convolutional Long Short-Term Memory (LSTM) neural network to capture frequency and time domain features over time. As a result, the proposed method outperforms the baseline method while reducing the number of parameters by more than 80%. An experiment on a small-scale device demonstrates that our model is efficient enough to function in real time.

Index Terms: keyword spotting, convolutional neural network, recurrent neural network, convolutional LSTM, small footprint

2. Network Architectures

In this section, we explain the architectures of LSTM and Convolutional LSTM networks.

2.1. LSTM

As demonstrated in [11], the simple architecture of RNNs makes them incapable of capturing long-term relationships within input. Long Short-Term Memory (LSTM) networks work out these relationships using cells and gates [12, 13]. The cells use gate structures to choose the necessary information within current and recurrent states, allowing the LSTM to handle relatively long sequences of data. At time frame $t$ ($t = 1, 2, ..., T$), when current input vector $x_t$, recurrent input vector $h_{t-1}$, and cell state $c_{t-1}$ are given, an LSTM update can be formulated as:

$$
\begin{align*}
\dot{i}_t & = \sigma(w_{ix} x_t + w_{ih} h_{t-1} + w_{ic} c_{t-1} + b_i) \\
\dot{f}_t & = \sigma(w_{fx} x_t + w_{fh} h_{t-1} + w_{fc} c_{t-1} + b_f) \\
c_t & = \dot{f}_t \odot c_{t-1} + \dot{i}_t \odot \tanh(w_{cx} x_t + w_{ch} h_{t-1} + b_c) \\
\dot{h}_t & = \sigma(w_{cx} x_t + w_{ch} h_{t-1} + w_{co} c_{t-1} + b_o) \\
o_t & = \sigma(w_{ox} x_t + w_{oh} h_{t-1} + w_{oc} c_{t-1} + b_o) \\
h_t & = o_t \odot \tanh(c_t)
\end{align*}
$$

where $i_t, f_t, c_t$ and $o_t$ denote the input gate, forget gate, cell update gate, and output gate, respectively, and $\odot$ denotes the Hadamard product.
2.2. Convolutional LSTM

Convolutional LSTMs (CLSTMs) [10] use previous and current spatial information, such as current matrix $X_t$ and recurrent matrix $H_{t-1}$. This allows the CLSTM to propagate spatial information to the next time frame, making it possible to capture temporal changes in the input matrix. At time frame $t$ ($t = 1, 2, \ldots, T$), when current input matrix $X_t$, recurrent input matrix $H_{t-1}$, and cell states $C_{t-1}$ are given, a CLSTM can be formulated as:

\[ I_t = \sigma(W_{xi} \ast X_t + W_{hi} \ast H_{t-1} + W_{ci} \ast C_{t-1} + b_i) \]
\[ F_t = \sigma(W_{xf} \ast X_t + W_{hf} \ast H_{t-1} + W_{cf} \ast C_{t-1} + b_f) \]
\[ C_t = F_t \odot C_{t-1} + I_t \odot \tanh(W_{xc} \ast X_t + W_{hc} \ast H_{t-1} + b_c) \]
\[ H_t = O_t \odot \tanh(C_t) \]

where $I_t$, $F_t$, $C_t$ and $O_t$ denote the input gate, forget gate, cell update gate, and output gate, respectively, and $\odot$ and $*$ denote the Hadamard product and the convolutional operation, respectively. In comparison with Eq.(1), weighted calculations are simply replaced by convolutions. Also, note that the matrix size should remain unchanged between the CLSTM input and output. Thus, we have to prevent the size of the matrix $H_T$ from changing since the CLSTM uses $H_t$ for the subsequent time frame ($t+1$).

2.3. Advantages of Convolutional LSTM

An LSTM is a kind of fully-connected network of feed-forward DNNs across time, as mentioned in Section 1, so they are unable to capture input context in either the time or frequency domains [5]. In contrast, CLSTMs are able to capture input context in both the time and frequency domains. We believe this will allow a CLSTM to effectively capture input context expanded complexity in both the time and frequency domains over the time frames or segments. Thus, we expect a CLSTM to achieve better results on MWD tasks.

3. Experimental Setup

In this section we explain our experimental settings, such as the data set, model architectures and training settings.

3.1. Dataset

We constructed an original data set for this study for training and testing. We set the Japanese word “ゴーギモン” (Romanization: “goemon”) as our Magic Word, which could be used to activate a home robot to perform tasks when it detects this word in human speech. Goemon is a popular ninja or bandit character name in Japan. We created five suitable scenarios involving the use of the Magic Word, each of which consisted of a dialog between two subjects, A and B. During recording, we combined

the subjects’ monaural audio streams, which were captured by lapel microphones, into a stereo stream using a PCM recorder. The quantization and sampling rate of the recorded speech data were set to 24 bit and 48 kHz, respectively. After recording, the speech data were converted to 16 bit and 16 kHz and split into monaural speech data. We collected a total of 200 samples of monaural data from 20 pairs of subjects (40 subjects in total). Next, we converted the speech into log power spectrograms using FFT and a Hamming window. Window size and shift width were set at 256 points (16 ms) and 160 points (10 ms), respectively. Then we applied segmentation in order to create a matrix of stacked frames, as shown in Fig. 1. The segment size and shift width were set to 20 frames (206 ms) and 10 frames (100 ms), respectively. Finally we added Positive or Negative labels to each segment; Positive (1) for segments containing the Magic Words, and Negative (0) for all the others. Segments containing 70% period from the end of the Magic Word were labelled as Positive. As a result, we obtained 187,184 Negative segments (10.711 hours) and 500 Positive segments (103 seconds). This data set was then divided into training and test sets at a ratio of 9:1.

3.2. Model Architectures and Training Settings

We compared several variations of our models, all of which included a CNN block and stacked RNN blocks. The details of each variant’s architecture are as shown in Tables 1. As in [3, 4], given the discrepancy in input dimensionality and the training data, we optimized all of the models using a limit on the number of parameters of 250,000 to allow for a fair comparison.

The baseline models (Table 1) used a CRNN architecture similar to the one proposed in [4]. In this paper, the baseline models are called lstm_RNN because the LSTM layers are stacked (the architecture shown on the left of Fig. 2). According to [9], BiCLSTMs are unsuitable for real time processing; therefore, we did not use one. Our proposed models have uni directional CLSTM based architectures (the architecture shown on the right of Fig. 2). Our RNN block can contain either an LSTM, CLSTM or CLSTM NIN. In Table 1, the notation clstm_lstm means the RNN block contains CLSTM followed by an LSTM, while clstm_clstm contains two stacked CLSTMs, and clstm_nin contains a CLSTM followed by a CLSTM NIN. A CLSTM NIN is a CLSTM with 1 x 1 kernel, and this layer functions as a Network-in-Network (NIN) [14].
An NIN is actually a multilayer perceptron rather than a CNN, which is able to express more complicated projections than typical CNNs. However, an NIN is the equivalent of a CNN with a 1 × 1 kernel. In our proposed model architecture, convolution is applied in a temporal direction, so the NIN is also adopted in a temporal direction. The clstm nin 2 is larger model in the number of NIN kernel in clstm nin. In order to evaluate whether each model is effective for small-footprint MWD systems, the sml variants were re-optimized by limiting the number of parameters of about 30,000.

We used ReLU in the CNN and FC layers in all models. Batch-normalization was applied to CNN block's output, followed by a max pooling layer with 4 × 2 kernel (stride size was the same as kernel size). In addition, in all of the proposed variants the last CLSTM or CLSTM NIN block is followed by a max pooling layer with 2 × 2 kernel (stride size was the same as kernel size). For example, if RNN block is an LSTM, the CLSTM was followed by a max pooling layer. All of the proposed models were able to obtain good results, and the number of parameters was reduced by following the last CLSTM or CLSTM NIN layer with a max pooling layer. Furthermore, Dropout [15] was applied to all of the RNN Blocks. Dropout is also effective for CNNs [15], so we also applied it to the CLSTM architectures. The Dropout rate for all of the RNN blocks was set to 0.25.

We used Chainer [16] for NN implementation, training and testing. All of the models were classification models, so the softmax function was applied to the output layer. The loss functions of all of the models, including the baseline model [4], were set using a Softmax cross entropy loss. We used Adam [17] as our optimization method. The learning rate was initialized at 0.001 and the batch size was set to 60. All models were trained using the back-propagation through time (BPTT) method. BPTT length was set to T = 1.5 sec.

4. Experimental Results

In this section, we explain and discuss our experimental results. We chose the best performing model in iterations and used it for our evaluation experiment. MWD performance was measured using a Receiver Operating Characteristic (ROC) curve. The vertical and horizontal axes represent the False Rejection rates (FRR) and False Alarms per hour (FAh), respectively. The lower the FRR per FAh, i.e., the closer the curve is to the bottom-left of the graph, the better a model’s Magic Word detection performance.

4.1. Comparison of all models

ROC curves for all of the models described in Table 1 are shown in Fig.3. As shown in Fig.3 (A), clstm nin 2 was the best performing model, followed by clstm lstm and clstm lstm sml. The clstm lstm model did not obtain results as good as clstm lstm or clstm nin, although the FRR of the clstm lstm model was better than the baseline CRNN (lstm lstm) at the FAh ≤ 21. The FRR of our proposed clstm nin 2 improved 57.5% from the FRR of lstm lstm at the FAh = 25. Regarding the small models, as shown in Fig.3 (B), clstm lstm sml and clstm nin 2 sml outperformed the other models. Therefore, when comparing the performance of the baseline and CLSTM models, based on our results we believe that the CLSTM models outperform the models using only LSTM.

4.2. Improvement of CLSTM NIN model performance

When comparing lstm lstm and clstm nin, we can see that clstm lstm has a much larger number of the recurrent units. So, to compare them more accurately, we adjusted the number of parameters to increase the weights in NIN (clstm nin 2, clstm nin 2 sml), keeping the total number of parameters around the previous limits of 250k or 30k. As shown in Fig.3, when comparing clstm nin, clstm nin sml, clstm nin 2 and clstm nin 2 sml, we can see that clstm nin 2 and clstm nin 2 sml achieved superior performance. Therefore, we think that a structure with many the number of weights is more effective for the CLSTM NIN following CLSTM.

4.3. Impact of CLSTM on reducing footprint

We ranked all of the models by FRR at 25 FAh, and the five, top-performing models are listed in Table 2. ROC curves for the top five models are shown in Fig.4. As shown in Tables 1 and 2, models clstm lstm sml and clstm nin 2 sml achieved higher performance than lstm lstm, while reducing the number of parameters by more than 80%. In addition, we compared sml models in order to dispove the suspicion that the

Table 1: Specifications of Baseline and Proposed Models: The kernel size of CNN Block was fixed (F_c, F_v) = (7, 5). Also, the kernel size and stride of CLSTM Block was fixed (F_i, c, f, v) = (3, 3) and (S_i, c, f, v) = (1, 1), respectively.

<table>
<thead>
<tr>
<th>Baseline Model Name</th>
<th>CNN Block</th>
<th>RNN Blocks</th>
<th>FC # Params.</th>
<th># Mul.</th>
</tr>
</thead>
<tbody>
<tr>
<td>lstm lstm</td>
<td>N_c (S_v, S_f)</td>
<td>N_i, c, f, v</td>
<td>N_o / f</td>
<td>-</td>
</tr>
<tr>
<td>lstm lstm sml</td>
<td>16 (3, 1)</td>
<td>40 10</td>
<td>20 30k</td>
<td>211k</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proposed Model Name</th>
<th>CNN Block</th>
<th>RNN Blocks</th>
<th>FC # Params.</th>
<th># Mul.</th>
</tr>
</thead>
<tbody>
<tr>
<td>clstm lstm</td>
<td>N_c (S_v, S_f)</td>
<td>N_i, c, f, v</td>
<td>N_o / f</td>
<td>-</td>
</tr>
<tr>
<td>clstm lstm sml</td>
<td>16 (3, 1)</td>
<td>12 25</td>
<td>30k</td>
<td>981k</td>
</tr>
<tr>
<td>clstm clstm</td>
<td>N_c (S_v, S_f)</td>
<td>N_i, c, f, v</td>
<td>N_o / f</td>
<td>-</td>
</tr>
<tr>
<td>clstm clstm sml</td>
<td>16 (3, 1)</td>
<td>16 84</td>
<td>30k</td>
<td>1M</td>
</tr>
<tr>
<td>clstm nin</td>
<td>N_c (S_v, S_f)</td>
<td>N_i, c, f, v</td>
<td>N_o / f</td>
<td>-</td>
</tr>
<tr>
<td>clstm nin sml</td>
<td>16 (1, 1)</td>
<td>12 62</td>
<td>20k</td>
<td>51M</td>
</tr>
<tr>
<td>clstm nin 2</td>
<td>N_c (S_v, S_f)</td>
<td>N_i, c, f, v</td>
<td>N_o / f</td>
<td>-</td>
</tr>
<tr>
<td>clstm nin 2 sml</td>
<td>16 (1, 1)</td>
<td>8 25</td>
<td>30k</td>
<td>1.8M</td>
</tr>
</tbody>
</table>
shown in Table 3. It takes our system about 20 ms for wave-
form processing, so DNN processing should be accomplished
within 80 ms. In Table 3, the models whose execution times
are shown in bold (models clstm_lstm, clstm_lstm_sml and
clstm_nin_2_sml) were able to operate in real-time. We can
also see that the clstm_lstm_sml model was the most effi-
cient, making it the best candidate for high precision MWD with
real-time processing. Models clstm_nin and clstm_nin_2 had
difficulty operating in real-time. However, as shown in Ta-
bles 1 and 3, although the number of multiplication required
by clstm_nin_2_sml model was smaller than the number re-
quired by the clstm_lstm, clstm_nin_2_sml model still re-
quired more execution time than the clstm_lstm model. In
future work, it will be necessary to implement faster CLSTM
models.

### 5. Conclusions

In this paper, we explored suitable CLSTM architectures for
small-footprint MWD systems. Our proposed models com-
ined, convolutional LSTMs with another LSTM or a Network-
in-Netw, enabling them to outperform the best performance
of the baseline model while achieving a more than 80% reduc-
tion in the number of parameters required. In particular, the
combination of a convolutional LSTM with a second LSTM
with a small number of parameters allowed us simultaneously
achieve a high level of performance, a small-footprint and real-
time operation. As mentioned in Section 4.4, CLSTMs have
the drawback of requiring a large number of multiplication op-
erations. Despite this problem, some of our proposed models
were able to operate in real-time on a small-scale Raspberry Pi
3 Model B+ device. Therefore, we believe that CLSTM-based
models are effective for small-footprint MWD. In future work,
we will adjust the hyper-parameters of CLSTMs to reduce the
number of multiplication operations. As another goal, we would
like to develop a model that allows users to change their Magic
Word arbitrarily.
6. References


