Journal of Ambient Intelligence and Humanized Computing (2022) 13:1525–1535
https://doi.org/10.1007/s12652-021-03022-1

ORIGINAL RESEARCH

EdgeCRNN: an edge-computing oriented model of acoustic feature enhancement for keyword spotting

Yungen Wei[1,2] · Zheng Gong[1] · Shunzhi Yang[1,2,3] · Kai Ye[1,2,3] · Yamin Wen[3]

Received: 12 October 2020 / Accepted: 1 March 2021 / Published online: 14 March 2021

Abstract
Keyword Spotting (KWS) is a significant branch of Automatic Speech Recognition (ASR) and has been widely used in edge computing devices. The goal of KWS is to provide high accuracy with a low False Alarm Rate (FAR), while reducing the costs of memory, computation, and latency. However, limited resources are challenging for KWS applications on edge computing devices. Lightweight models and structures for deep learning have achieved good results in the KWS branch while maintaining efficient performances. In this paper, we present a new Convolutional Recurrent Neural Network (CRNN) architecture named EdgeCRNN for edge computing devices. EdgeCRNN, which is based on depthwise separable convolution and residual structure, uses a feature enhanced method. On the Google Speech Commands Dataset, the experimental results depict that EdgeCRNN can test 11.1 audio data per second on Raspberry Pi 3B+, which is 2.2 times than that of Tpool2. Compared with Tpool2, the accuracy of EdgeCRNN reaches 98.05% whilst its performance is also competitive.

Keywords Edge computing · Keyword spotting · Convolutional recurrent neural network · Feature enhancement · Lightweight structure

1 Introduction

Keyword Spotting (KWS) is a branch of Automatic Speech Recognition and focuses on detecting predefined keywords from a continuous audio stream. The wake-up words are the critical applications of KWS on edge computing devices, such as Apple's "Hey Siri" and Google's "OK Google". The device is awakened to execute the appropriate command if the KWS system detects a predefined keyword in a dialogue.

Traditional methods of KWS usually use the Keyword/Filler Hidden Markov Model (HMM) (Wilpon et al. 1991; Silaghi and Bourlard 1999; Silaghi 2005). However, depending on an HMM topology, these systems require Viterbi decoding, which is computationally expensive. These approaches are not suitable for edge computing with limited resources. In KWS branch, Deep Neural Networks (DNN) has been shown to produce an efficient and reliable solution. DNN (Chen et al. 2014) was the first deep learning model applied to KWS. Its model parameters are 224M, which is smaller than the 373M of GMM-HMM model, and its performance also is better than HMM model. However, these model parameters and computation costs are still relatively expensive and not suitable for edge computing devices.

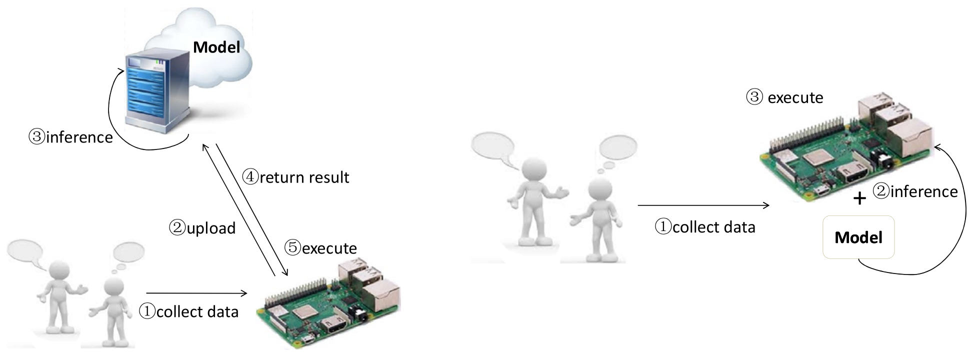
A preliminary version of this paper has been published at ML4CS 2020, Springer LNCS, this is the full-length version. This paper is supported by the National Natural Sciences Foundation of China (No. 62072192), National Cryptography Development Fund (No. MMJJ20180206), the Project of Science and Technology of Guangzhou (No. 201802010044), Guangdong Basic and Applied Basic Research Foundation (No. 2019A1515011797), the Opening Project of GuangDong Province Key Laboratory of Information Security Technology(No. 2020B1212060078), the Project of Guangdong Province Innovative Team(2020WCXTD011) and the Research Team of Big Data Audit from Guangdong University of Finance and Economics.

✉ Yamin Wen
wenyamin@gdufe.edu.cn

[1] School of Computer Science, South China Normal University, Guangzhou, China

[2] Computer Engineering Technical College, GuangDong Polytechnic of Science and Technology, Guangzhou, China

[3] School of Statistics and Mathematics, Guangdong University of Finance and Economics, Guangzhou, China



(a) server-client pattern **(b)** client pattern

Fig. 1 KWS system deployment pattern

In addition, the KWS system uses a server-client pattern, where the client collects and uploads data on the terminal and the cloud server processes it (Fig. 1a). When the client receives the result returned by the cloud server, it will execute related commands and operations. With the rapid growth of data, pressures of computing and storage at the server will increase exponentially. Eventually, the user experience will become very bad. Moreover, there is a problem of user privacy leakage, which may lead to violations of law (Gaff et al. 2014; Custers et al. 2019). Consequently, we adopt a new pattern that the client collects and processes data on terminal as show in Fig. 1b. The client does not need to upload data to the cloud server. This pattern not only diminishes the burden of cloud servers and network bandwidth, and protects user privacy, but also provides positioning and high-quality services.

However, the high requirements of model on hardware and limited resources pose a challenge to the application of this pattern in edge computing devices. The hardware acceleration and designing lightweight models have been used to solve this problem. Hardware acceleration provides powerful computational power by adding hardware. In (Benelli et al. 2018), Benelli et al. used a Neural Compute Stick (NCS) to increase the speed, which reduced the model latency by 50%. Dinelli et al. (2019) proposed a Convolutional Neural Network (CNN) based on a field-programmable gate array (FPGA), which is nearly 10 times faster than NCS. However, these methods based on NCS or FPGA are costly, which are not used in edge computing devices. The method of designing lightweight models can reduce calculation costs and model parameters by designing or automatically selecting lightweight algorithms, structure, or model calculation methods. So we choose the approach of designing lightweight model. In other words, the lightweight model can solve the problem of insufficient resources and is an edge-computing oriented model.

Various lightweight architectures for deep learning have been successfully applied to KWS problems, such as Tpool2 (Tang et al. 2018) and CNN (Sainath and Parada 2015). Compared with DNN (Chen et al. 2014), CNN (Sainath and Parada 2015) offers between a 27 and 44% relative improvement in false alarm rate (FAR) with a limited number of multiplications and parameters. However, CNN ignores the global time and spectral correlation owing to the size of the convolution kernel. Recurrent Neural Network (RNN) can leverage a longer temporal context, which makes up for this question of CNN. Recently, RNN (Sun et al. 2016) and convolutional recurrent neural network (CRNN) (Arik et al. 2017; Du et al. 2018; Zeng and Xiao 2019) are used in KWS. CRNN is a hybrid of CNN and RNN. In CRNN, convolution layer extracts local temporal/spatial correlation and recurrent layer extracts global temporal features dependency in time sequence. The accuracy rate reaches 97.71% in the Talk-Type dataset (Arik et al. 2017). However, the CNN model in CRNN (Arik et al. 2017) did not use a lightweight structure.

In this paper, we design a new CRNN model called EdgeCRNN. Its CNN adopts EdgeCRNN Block based on depthwise separable convolution and residual structure. Besides, we propose feature enhancement in EdgeCRNN, which uses the LFBE-Delta feature instead of the Mel-Frequency Cepstrum Coefficient (MFCC) as input features. The LFBE-Delta consists of three features. EdgeCRNN can recognize 12 classes keywords by training on the Google Speech Commands Dataset (Warden 2018). The experiment results show that EdgeCRNN not only reduces model parameters and Floating-point Operations Per Second (FLOPs), but also decreases latency. The test cases can run normally and test 11.1 audio data per second on edge computing device Raspberry Pi 3B+ without stuttering. Besides, accuracy rate has also improved and reaches 98.05%.

This paper is organized as follows. Section 2 introduces the related work of the lightweight KWS model. We describe our approach and EdgeCRNN architecture in Sect. 3. In Sect. 4, we explain the experiment steps and results. Section 5 gives a conclusion.

2 Related work

There are three main methods for designing lightweight KWS models: (1) model compression, (2) automatic neural network architecture design based on Neural Architecture Search (NAS), (3) artificial design of lightweight neural network.

The model compression is to further diminish the size of the model by removing redundant layers, quantizing high-precision weight parameters, and decomposing complex operations. According to the redundancy of neural networks in different aspects, it can be divided into quantizing weight, low rank decomposition, and knowledge distillation (Nakkiran et al. 2015; Mishchenko et al. 2019). In (Tucker et al. 2016), George et al. used low-rank weight matrice and knowledge distillation throughout DNN, which obtained a 23.9% relative reduction in frame error rate. TDNN (Sun et al. 2017) used singular value decomposition (SVD) to lower model complexity and obtained a 37.6% reduction in area under the curve (AUC) compared to the DNN (Chen et al. 2014).

The Neural Architecture Search can automatically design high-performance neural networks, which are gradually applied in the fields of computer vision (Howard et al. 2019) and speech recognition (Tan et al. 2018). It consists of search space, search strategy, and performance evaluation strategy. Based on the search strategy, NAS automatically designs a model suitable for specific applications in a predefined search space (Mazzawi et al. 2019; Anderson et al. 2020). SANAS is able to adapt the architecture of the neural network on-the-fly at inference time based on the difficulty of the task with a high recognition level (Véniat et al. 2019). AUTOKWS adopts differentiable neural structure search to find more effective networks and it attains 97.2% top-1 accuracy (Zhang et al. 2020).

The artificially designed lightweight neural network mainly reduces the amount of calculation by optimizing the calculation method of convolution and designing more efficient convolution operation. The lightweight structure mainly includes deep residual structure (He et al. 2016), depthwise separable convolution (Sifre and Mallat 2014), dilated convolution (Coucke et al. 2019) and attention mechanism (Luo et al. 2019). DS-CNN (Zhang et al. 2017) is a lightweight model based on depthwise separable convolution, its accuracy rate reaches 95.4% with limited memory and computational capability. With maintaining performance, CNNs combine residual learning and dilated convolution (Coucke et al. 2019; Tang and Lin 2018). These models can effectively extract speech features repeatedly and reduce the computational cost of the model. The accuracy rate reaches 95.8% on paper (Tang and Lin 2018).

Both model compression and automatic design methods consume resources and time costly. The artificial design of lightweight neural network requires designers to have professional knowledge, but it consumes fewer resources and is mature in technology. Therefore, we use the artificial design neural network method to design a lightweight KWS model for edge computing devices.

3 EdgeCRNN

In this section, first we propose two feature enhanced approaches. And then the architecture of EdgeCRNN is designed based on EdgeCRNN Block.

3.1 Feature enhancement

In order to extract acoustic features more efficiently, we propose two enhanced methods Input Feature Enhancement and First Convolution Layer Feature Enhancement. Table 3 depicts that the accuracy is increased by 3% after adding feature enhancement.

3.1.1 Input feature enhancement

The traditional method MFCC only extracts the envelope information on spectrum and it loses sound details. MFCC is suitable for voice data longer than 2 s, because the data contains enough envelope features. In the KWS task, the overall features of speech commands with an average length of 1–2s are few. If only the envelope features in the frequency spectrum are extracted, it is not conducive to the inference of the neural network. We find that the Log-Mel filterbank energies (LFBE) contains more features, such as low-frequency and spectral details. Many proposals had adopted LFBE as feature extracted method (Sainath and Parada 2015; Coucke et al. 2019). Besides, the first derivative (Delta) and second derivative (Delta-Delta) features on the time axis of MFCC can better represent correlation among frames.

The deep learning model has powerful learning and presentation capabilities and it can extract more robust features from input features (Abdel-Hamid et al. 2014). We propose a new feature extraction method LFBE-Delta as input feature. LFBE-Delta is 39 dimensions and computed every 30 ms with a 10 ms frame shift by LibROSA package (McFee et al. 2015). It contains three features LFBE, Delta, and Delta-Delta. The dimension of each feature is 13. Compared with MFCC, it contains more types of features as illustrates on

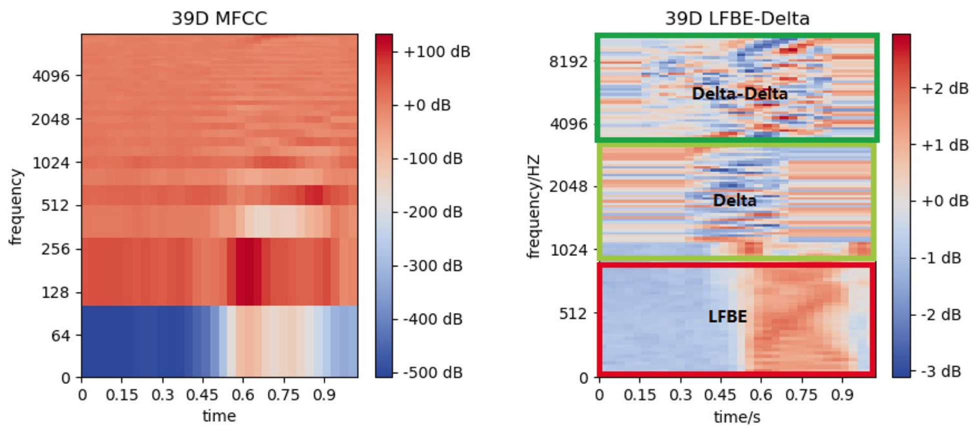


Fig. 2 Input Feature. 39 dimensions (39D) MFCC and 39D LFBE-Delta (LFBE-Delta denotes the concatenation of 13D LFBE, 13D Delta and 13D Delta-Delta)

Table 1 Convolution operation parameters

Convolution	Kernel size	Stride	padding
Conv2D	3×3	2×2	1×1
Conv2D_enhance	3×3	$\mathbf{1 \times 1}$	1×1

Conv2D stands for standard convolution, Conv2D_enhance stands for enhanced convolution with setting Stride = 1

Fig. 2, which is helpful for the model to extract more useful features and improve accuracy.

3.1.2 First convolution layer feature enhancement

The convolution kernel enhances features by multiplying input signals with sliding, then it outputs a small-size map feature. The convolution operation parameters include stride, kernel size, and padding, as described in Table 1. By setting stride = 1, the size of the output map remains the same. Therefore, repeating multiple convolution operation is equivalent to adding features. Compared to large-size input feature, small-size inputs could relatively save computational costs.

Compared with the input data dimensions of $3 \times 224 \times 224$ in the computer vision (Zhang et al. 2018; Howard et al. 2017), the acoustic feature of 39 dimensions LFBE-Delta is too small to effectively extract valid features. In the convolution layer, maintaining the output map size unchanged by setting stride could extract more efficient features. So we maintain the output map size by setting stride = 1 to achieve feature enhancement. the convolutional operation calculates equation as follows:

$$\left\{\frac{t-m+2\times p}{s}+1\right\}\times\left\{\frac{f-n+2\times p}{s}+1\right\} \tag{1}$$

where t and f represent the input feature dimension in time and frequency domain, respectively. The $m \times n$, p, and s are kernel size, padding, and stride of the convolution operation. The Conv2D output features map is $\frac{d}{2}$ dimension(s) with setting stride = 2, while the Conv2D_enhance is d dimension(s) for stride = 1. This means that the input feature has been enhanced. As the stride decreases, model calculation will increase by a corresponding multiple. In the experiment, we find that this method only once is the best balance between computational costs and improving performance.

3.2 The building blocks of EdgeCRNN

In this section, we first describe the core approaches (i.e., depthwise separable convolution and residual structure), on which EdgeCRNN Block is built. We then introduce the EdgeCRNN Block and RNN.

3.2.1 Depthwise separable convolution

The idea of depthwise separable convolution is based on that depth and spatial dimension of a filter can be separated. Hence, it splits a kernel into two separate parts, Depthwise Convolution (DWConv) and Pointwise Convolution (PConv). The DWConv performs independently over each channel of input signal, then PConv projects the output channels by the DWConv onto a new channel space.

According to Howard et al.'s research (Howard et al. 2017), the FLOPs of Depthwise Convolution and Pointwise Convolution are $D_K^2 \times M \times D_F^2$ and $N \times M \times D_F^2$, and the FLOPs of the depthwise separable convolution is $\frac{1}{N} + \frac{1}{D_k^2}$ times that of the standard convolutional operation, where M and N are the number of input and output channels, D_k is the

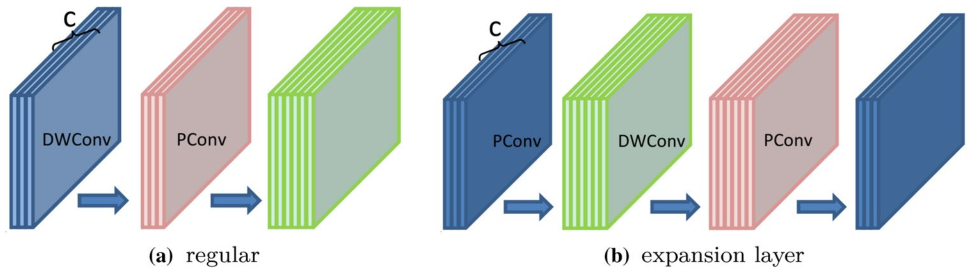


Fig. 3 Convolution block. The C denotes channel number. **a** Is regular form of Depthwise Separable Convolution. **b** Is expansion form which added a PConv layer before DWConv layer

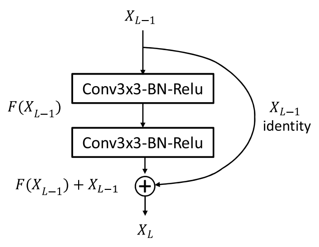


Fig. 4 Residual structure of a building block. A basic CNN layer consists of standard convolution layer of 3×3 kernel size, Batch Normalization (BN) (Ioffe and Szegedy 2015), and rectified linear units (Relu) (Glorot et al. 2011) function

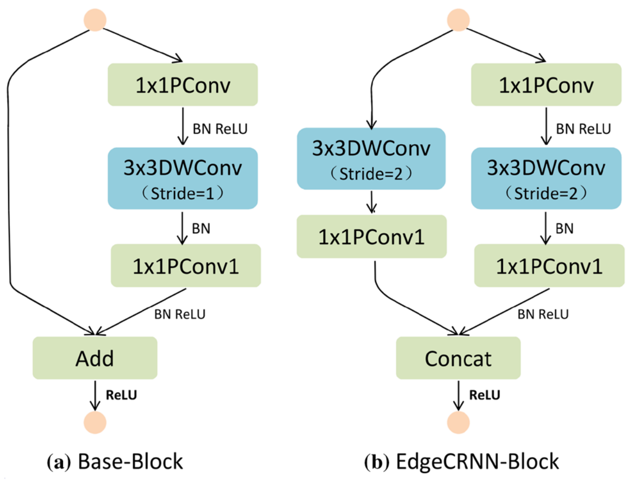


Fig. 5 EdgeCRNN Block. **a** Is the basic block with the output operate by "Add"; **b** Is the downsampling module with the output operate by "Concat"

kernel size, and D_F is the spatial width and height of a square input feature map. It gradually replaces standard convolution kernels in many lightweight model studies. Most EdgeCRNN's convolution kernel size is 1×1, so the EdgeCRNN computational cost than full size 3×3 convolution layer is about 9 times less. This proves that the EdgeCRNN model can reduce computational costs and model parameters.

DWConv cannot change the channels number of input features (Fig. 3a), and the features cannot be extracted well if the number of channels is small. The acoustic feature has only one channel number, the extraction effect will is poor. However, PConv can change the number of channels, and the calculation amount and model parameter are relatively small from above describe. Therefore, we increase the number of channels by adding a PConv layer before DWConv layer as describe on Fig. 3b, the model feature extraction effect will is better.

3.2.2 Residual structure

In theory, deeper networks are more capable of learning. However, with the numbers of network layer increases, the model structure becomes more complicated and requires expensive computational costs, even has the problem of vanishing/exploding gradients. Therefore, He et al. (He et al. 2016) proposed ResNet to solve the above problems. It is based on the residual structure and uses the shortcut connections. Shortcut connections are skipping one or more layers on neural network with an identity mapping function.

$$X_L = F(X_{L-1}) + X_{L-1} \tag{2}$$

The Eq. 2 can be implemented by the L_{L-1}^{th} feed forward neural networks with shortcut connections (Fig. 4), where $F(\cdot)$ is a composite function of operations, such as Convolution (Conv), BN, and ReLU function. The X_L denotes the output of L *th* layer. In our work, the shortcut connections simply perform identity mapping, and their input and output of different layers are added by an element-wise or concatenated through the channel. Identity shortcut connections do not add extra parameters or computational complexity. The residual structure increases the training speed of model. In the model design process, we add a residual structure to improve the efficiency of the model and reduce redundant hidden layer.

3.2.3 RNN

The RNN uses a loop structure to connect previous state information to the current state, which can well extract sequence data context features. However, standard RNN has short-term memory problems. The long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997) and gated recurrent unit (GRU) (Cho et al. 2014) of variant RNN were created as the solution to this problem. They have internal mechanisms called memory cells that can store the flow of information. BiLSTM can obtain time series features well and achieve the accuracy of 96.6% (Zeng and Xiao 2019). Hence, we use LSTM on EdgeCRNN model.

3.2.4 EdgeCRNN Block

We design the EdgeCRNN Block based on depthwise separable convolution and residual structure, which is similar to the paper (Ma et al. 2018). As discussed above, EdgeCRNN Block uses the expansion layer (Fig. 3b) to increase channel number, and it includes Base-Block and EdgeCRNN-Block (Fig. 5). EdgeCRNN Block consists of two PConv layers and one DWConv layer, and it selects the ReLU non-linearity and uses BN to normalize input data. Assuming that x is input data, the input data of the two branches is the same as x in EdgeCRNN-Block, and the x is divided into two halves as the input of the two branches in Base-Block. EdgeCRNN-Block is used for downsampling to halve the input signal size by setting Stride = 2 on the DWConv layer, and then it uses the Concat operation to double the number of channels. The Concat denotes combining multiple data sources together. Base-Block is the basic block and adding features by Add operation, the input signal size and channels remain unchanged with setting Stride = 1. EdgeCRNN-Block is on the first unit of each Stage (see more detail in Sect. 3.3), and Base-Block follows it.

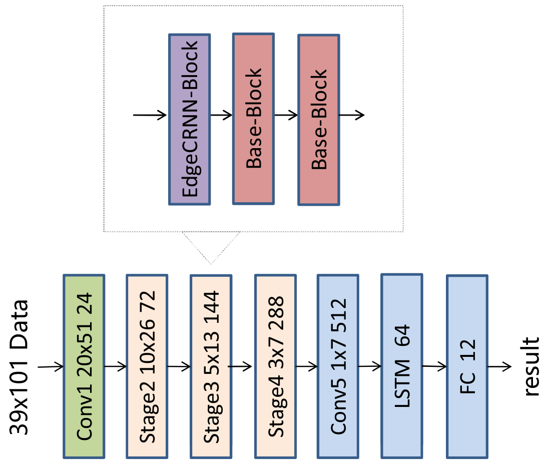


Fig. 6 EdgeCRNN 1.0× model. Where 20 × 51 denotes output map size, 24 represents channel number

3.3 The architecture of EdgeCRNN

The EdgeCRNN architecture is a hybrid model of CNN and RNN, where CNN is mainly composed of a stack of EdgeCRNN Block and RNN selects the LSTM model which consists of one hidden layer with 64 nodes. Besides, CNN is divided into one first convolution layer feature enhancement layer called Conv1, three Stage, and one standard convolution layer named Conv5. Conv1 and Conv5 contain the variant Pool operator which is a sample-based discretization process with the goal of downsampling the input representation (Sun et al. 2016). Conv1 is MaxPool and Conv5 uses GlobalPool. The Fig. 6

Table 2 EdgeCRNN architecture

Layer	Output	K	S	R	Output channels			
					0.5×	1.0×	1.5×	2.0×
Audio	39 × 101	–	–	–	1			
Conv1	39 × 101	3 × 3	**1**	1	16	24	24	24
MaxPool	20 × 51	3 × 3	2					
Stage2	10 × 26	3 × 3	2	1	32	72	116	160
	10 × 26	3 × 3	1	1				
Stage3	5 × 13	3 × 3	2	1	64	144	232	320
	5 × 13	3 × 3	1	2				
Stage4	3 × 7	3 × 3	2	1	128	288	464	640
	3 × 7	3 × 3	1	1				
Conv5	3 × 7	1 × 1	1	1	256	512	1024	1024
GlobalPool	1 × 7	3 × 1	1					
RNN	–	–	–	–	64			
FC	–	–	–	–	12			
MFLOPs					4.10	14.54	34.89	57.65
MWeights					0.15	0.59	1.15	1.68

Stage* is EdgeCRNN Block, K and S are the size and stride of the convolutions kernel, R represents the number of module and padding = 1 illustrates that each Stage are two units in EdgeCRNN. The first unit consists of a downsampling block EdgeCRNN-Block with a convolution kernel stride of 2. The second unit consists of the Base-Block module, which is located behind the EdgeCRNN-Block and its number is determined by R in Table 2.

CNN and LSTM cell are used to extract frequency and time domain features on spectrogram respectively, and fully connected layers (FC) is used for posterior probability classification. The EdgeCRNN uses Width Multiplier α similar to (Howard et al. 2017). The role of the α is to thin a network uniformly at each layer. For example, the 0.5× denotes $\alpha = 0.5$ and the number of output channels N becomes $0.5 \times N$ in Table 2.

4 Experiments on EdgeCRNN

In this section, we introduce the datasets, experiment steps, and how to train the model. We then investigate the effects of feature enhancement and EdgeCRNN Block and compare results to the full convolution model. Finally, we compare performances between EdgeCRNN and popular KWS models.

4.1 Experimental step on EdgeCRNN

We evaluate our model by using Google Speech Commands Dataset (Warden 2018), which consists of 65,000 1-s utterances of 30 words by thousands of different people. The sampling frequency is 16KHz. Our task is to discriminate among 12 classes "yes", "no", "up", "down", "left", "right", "on", "off", "stop", "go", unknown, and silence. The unknown class is used to simulate the model to learn the difference between keywords and non-keywords. The silence class represents low loudness audio. Under the SNR of random sampling between [−5 db, + 10 dB], we manually add car noise and cafeteria to the dataset. It can improve the performance of the KWS system in practice, which gets closer to the background of continuous noise. The dataset is then randomly split into training, validation, and test set in the ratio of 80:10:10. EdgeCRNN is trained in the training and validation set, and the experimental results are obtained from the test set.

We use the Tpool2 (Tang et al. 2018) as the baseline model, which consists of two convolutional layers followed by two linear layers and one DNN layer. In our experiment, the input features are 39 dimensions LFBE-Delta. The EdgeCRNN uses the Relu activation function, the Adam optimizer, and Cross Entropy (CE) loss function on each of the convolution layers. In Equation 3, the learning rate is decayed every 50 rounds. The initial learning rate LR_init is 1E−3, the end learning rate LR_end equals 1E−4, total epochs $total\_epoch$ are 500, $current\_epoch$ denotes the current epoch, and batch sets 128.

$$LR = LR\_init - \frac{current\_epoch}{total\_epoch} \times (LR\_init - LR\_end) \quad (3)$$

4.2 Model training on EdgeCRNN

Accuracy, FLOPs, and model parameters are our primary metric of quality, which are measured as the fraction of correct classification decisions. We also plot receiver operating characteristic (ROC) curves, where the x and y axes denote FAR and false reject rate (FRR), respectively. Curves for each of the keywords are computed and then averaged vertically to produce the overall ROC. The lower the curve, the better the model performance.

4.2.1 Training based on feature enhancement

We compare the model performances that adopt feature enhancement and non-use. The accuracy of EdgeCRNN-Mel is 3% higher than EdgeCRNN-M for LFBE-Delta containing three features in Table 3. Meanwhile, EdgeCRNN-Mel-F and EdgeCRNN-M-F also have a similar relationship in accuracy. The Fig. 7b illustrates the EdgeCRNN-Mel-F gives a 69.5% relative improvement over the EdgeCRNN-M-F at the operating point of 0.1 FAR. Compared with one feature, input features including three feature types can improve the accuracy of the model.

The first convolution layer feature enhancement can repeatedly extract features and improve accuracy. The EdgeCRNN-Mel-F is 0.8% higher than EdgeCRNN-Mel. However, FLOPs of the EdgeCRNN-Mel-F is almost 10M more than that of EdgeCRNN-Mel. We have find that it is most appropriate to reuse it only once. So EdgeCRNN uses the first convolution layer feature enhancement only once. The

Table 3 Accuracy of features enhancement

Model	FLOPs	Parameter	Accuracy (%)
EdgeCRNN-M	4.60M	0.59M	94.15
EdgeCRNN-M-F	14.54M	0.59M	94.97
EdgeCRNN-Mel	**4.60M**	0.59M	97.05
EdgeCRNN-Mel-F	14.54M	0.59M	**97.89**

Where M and Mel denote MFCC as feature extraction and LFBE-Delta, respectively. F represents first convolution layer feature enhancement, the α defaults to 1.0×

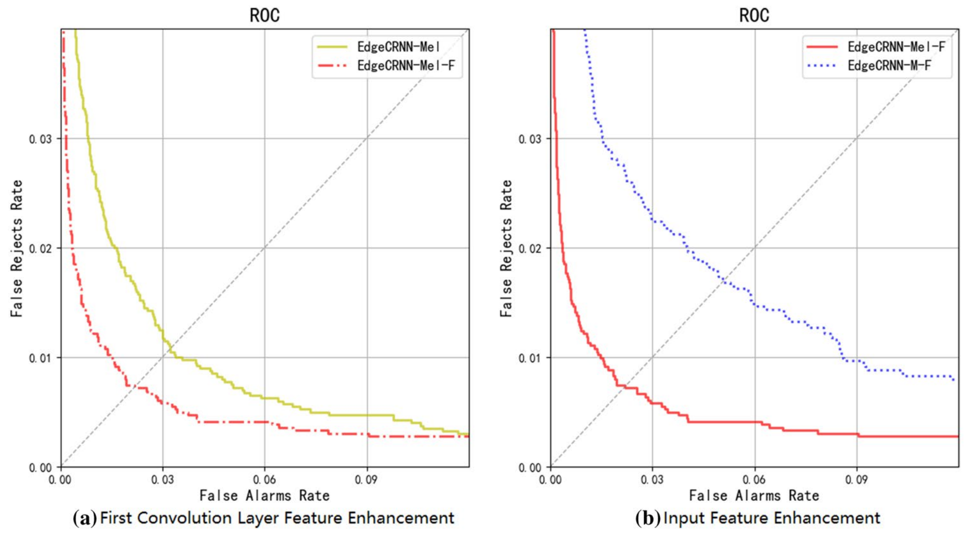


Fig. 7 ROC curves for feature enhancement

Table 4 Table caption

RNN model	FLOPs	Parameter	Accuracy (%)
RNN	14.54M	**0.38M**	97.77
GRU	14.54M	0.53M	97.78
LSTM	14.54M	0.59M	**97.89**

FRR of EdgeCRNN-Mel-F has a 34.2% relative improvement over the EdgeCRNN-Mel at the operating point of 0.1 FAR from Fig. 7a, which depicts that EdgeCRNN extracts feature more robust by feature enhancement.

4.2.2 Training based on RNN

In addition to standard RNN, RNN also has variant models LSTM and GRU. We test the performance of EdgeCRNN combined with the variant RNN model. Due to the different internal mechanism of the variant RNNs, the parameter calculation method is different, which is why the Parameter values are different in Table 4. CSANN LAB (Dey and Salemt 2017) proposed the calculate method of parameter, the methods of RNN, LSTM, GRU are Eqs. 4–6, respectively.

$$P_{RNN} = n^2 + nm + n \tag{4}$$

$$P_{LSTM} = 4 \times (n^2 + nm + n) \tag{5}$$

$$P_{GRU} = 3 \times (n^2 + nm + n) \tag{6}$$

where n and m denote the output dimension and input. The Equation shows that the standard RNN has the least number

Table 5 EdgeCRNN block vs full convolution

Model	FLOPs	Parameter	Accuracy (%)
Conv EdgeCRNN 1.0×	104.19M	2.28M	96.96
EdgeCRNN 1.0×	**14.54M**	**0.59M**	**97.89**

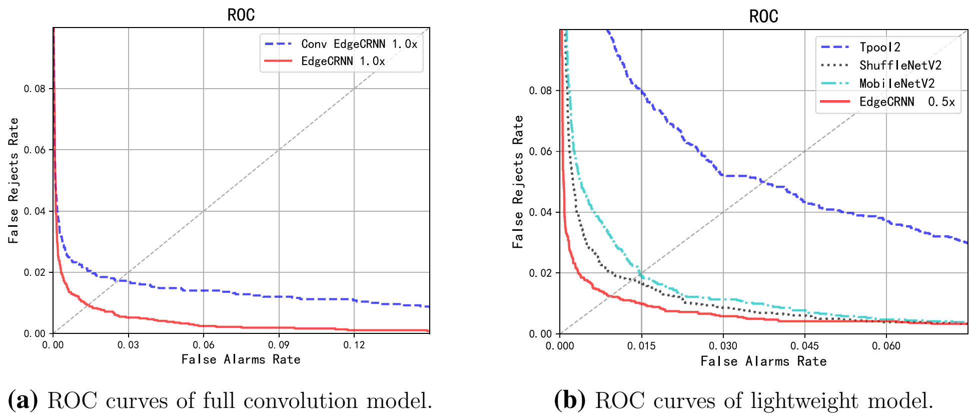


Fig. 8 ROC curves for others model

of model parameters and the LSTM has the largest. LSTM is 4 times that of RNN, and GRU is 3 times that of RNN.

The FLOPs of RNN are more complicated, so the FLOPs of RNN only roughly calculated in our work. The feature extraction uses LFBE-Delta and α is 1.0 multiple. The Table 4 shows that LSTM has the most parameters and the highest accuracy for its three gates, RNN has the fewest parameters. So we choose the LSTM as the RNN of EdgeCRNN in this paper.

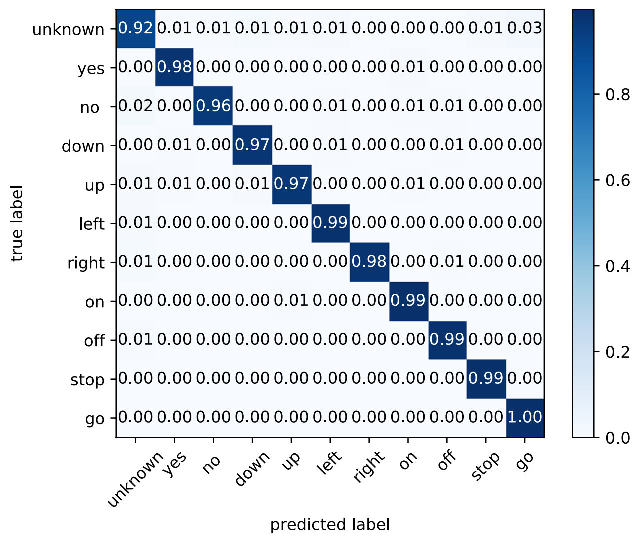
4.3 Result on EdgeCRNN

First, we test the performance of the EdgeCRNN with the EdgeCRNN Block and compare it with the fully convolutional model. Table 5 illustrates that compared with the full convolution model, the FLOPs based on the EdgeCRNN Block is reduced by 6.2 times and the model parameters are 2.9 times few. Meanwhile, the accuracy rate is improved by 0.93%. Even with fewer FLOPs and parameters, the performance of EdgeCRNN 1.0× is better than Conv EdgeCRNN 1.0×. For example, Fig. 8a shows that the improvement of over 47% relative compared to the Conv EdgeCRNN 1.0× at the operating point of 0.01 FAR, where the Conv EdgeCRNN 1.0× represents full convolutions model. This demonstrates that the EdgeCRNN model is lighter than the full convolution model while maintaining better performance.

To further detect which keyword recognizes good or bad, we delve into the confusion matrix of 12 classes (see more 12 classes detail in Session 4.1) on Google Speech Commands Dataset. Most misclassified situations are caused by real commands predicted as an unknown class. The diagonal is the accuracy of each keyword on the confusion matrix, Fig. 9 illustrates that most of keywords accuracy exceeds 96%. Due to the increased noise, the silence class is ignored. Due to the similar pronunciation of keywords, there are many misidentified classification, such as unknown and “go”, unknown and “no”. Besides, on the basis of 12 keyword tasks, we test the performance of EdgeCRNN 1.0× in 22 keyword tasks (added new keywords “dog”, “zero”, “one”, “bed”, “two”, “three”, “four”, “five”, “bird”, “six”). Table 6 shows that model parameters added about 0.1M, while accuracy reaches 95.77% and is still competitive. It means that EdgeCRNN is efficient at the KWS task.

Table 7 compares accuracy between previous lightweight KWS models (Tang et al. 2018; Sun et al. 2016; Arik et al. 2017; Zeng and Xiao 2019; Zhang et al. 2017) and EdgeCRNN, these models are trained on the Google Speech Commands Dataset (Warden 2018) (except CRNN (Arik et al. 2017), which uses a private TalkType dataset, and the data of LSTM from literature (Zhang et al. 2017). The parameter of EdgeCRNN 1.0× is not the smallest, while it is relatively lightweight and less than 0.6 M. Besides, the accuracy of EdgeCRNN is higher than other KWS models from Table 7, which reaches 97.89% with limited computational cost (only 14.54M). This indicates that EdgeCRNN can almost achieve the state of art accuracy in KWS task and is a lightweight model.

We evaluate the performance of EdgeCRNN on edge computing device, which is depicted in Table 8. EdgeCRNN 0.5× can read 11.1 audio data per second on the Raspberry Pi 3B+, which is much faster than Tpool2 that is 5/s . It demonstrates that EdgeCRNN reduces latency and computational costs with an accuracy of 97.09%. From the keyword audio length of 1 second on Google Speech Commands Dataset, we know the speed of human speech is nearly one keyword per second. It means that EdgeCRNN processing



true \ predicted	unknown	yes	no	down	up	left	right	on	off	stop	go
unknown	0.92	0.01	0.01	0.01	0.01	0.01	0.00	0.00	0.00	0.01	0.03
yes	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00
no	0.02	0.00	0.96	0.00	0.00	0.01	0.00	0.01	0.01	0.00	0.00
down	0.00	0.01	0.00	0.97	0.00	0.01	0.00	0.00	0.01	0.00	0.00
up	0.01	0.01	0.00	0.01	0.97	0.00	0.00	0.01	0.00	0.00	0.00
left	0.01	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00
right	0.01	0.00	0.00	0.00	0.00	0.00	0.98	0.00	0.01	0.00	0.00
on	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.99	0.00	0.00	0.00
off	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00
stop	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00
go	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

Fig. 9 Confusion Matrix of EdgeCRNN 1.0×

Table 6 Performance based on 22 keyword tasks

Model	keyword	MFLOPs	Parameter	Accuracy (%)
EdgeCRNN 1.0×	12	14.54M	0.59M	97.89
EdgeCRNN 1.0×	22	14.54M	0.60M	95.77

Table 7 Accuracy of the related KWS models

Model	FLOPs	Parameter	Accuracy (%)
Tpool2 (Tang et al. 2018)	103M	1.09M	91.97
LSTM (Sun et al. 2016)	48.4M	0.26M	94.81
CRNN (Arik et al. 2017)	19.3M	**0.22M**	97.71
DenseNet-BiLSTM (Zeng and Xiao 2019)	–	0.24M	97.50
DS-CNN (Zhang et al. 2017)	56.9M	0.47M	95.38
EdgeCRNN 1.0×	**14.54M**	0.59M	**97.89**

Table 8 Performances of related KWS model on different platforms in terms of accuracy and speed

Model	MFLOPs	Accuracy (%)	CPU/s	ARM/s
Tpool2 (Tang et al. 2018)	103	91.97	27.6	5.0
ShuffleNetV2-M (Ma et al. 2018)	22.11	93.28	11.6	0.91
ShuffleNetV2 (Ma et al. 2018)	22.11	96.91	11.6	0.91
MobilNetV2 (Sandler et al. 2018)	36.69	96.80	3.6	0.15
EdgeCRNN 0.5×	**4.10**	**97.09**	**49.9**	**11.1**

The CPU denotes the test speed on a platform with a CPU of 3.6 GHz Intel(R) Core(TM) I3-8100 Processor Base Frequency. The ARM is the Raspberry Pi 3B+ of 1.2 GHz Processor Base Frequency and 1 GB memory

speed can keep up with the speed of human speech in a resource-limited environment.

To verify that EdgeCRNN is suitable for KWS, we compare the performance of ShuffleNetV2 (Ma et al. 2018) and MobileNetV2 (Sandler et al. 2018), which are the lightweight model in the field of computer vision. The experiment finds that the speed of EdgeCRNN 0.5x is 12.2 times and 74 times faster than them, and FLOPs is less than $\frac{1}{5}$ and $\frac{1}{9}$ on Raspberry Pi 3B+ from Table 8. ShuffleNetV2-M denotes input feature uses MFCC in Table 8, other models use LFBE-Delta. The accuracy rate of ShufflenetV2 is 96.91%, higher than ShuffleNetV2-M's 93.28%. This also proves that the LFBE-Delta feature can enhance features and improve accuracy on other models.

Table 9 compares the effects of different Width Multiplier models, which have four multiples 0.5×, 1.0×, 1.5×, 2.0× from Table 2. The 2.0× model has the highest accuracy 98.05%, and the 0.5x model processes 11.1 audio per second which is the fastest speed on Raspberry Pi 3B +. In practical applications, we should consider the trade-off between FLOPs and accuracy to choose the most appropriate multiplier model.

Table 9 Performances of different Width Multiplier

Model	MFLOPs	Parameters	Accuracy (%)	CPU/s	ARM/s
EdgeCRNN 0.5×	**4.10**	**0.29M**	97.09	**49.9**	**11.1**
EdgeCRNN 1.0×	14.54	0.59M	97.89	25.6	5.0
EdgeCRNN 1.5×	34.89	1.29M	97.92	17.3	3.1
EdgeCRNN 2.0×	57.65	1.72M	**98.05**	13.5	2.3

5 Conclusion

In the paper, we designed a new EdgeCRNN model for edge computing devices applied to KWS. We demonstrated how to improve EdgeCRNN's performance by using feature enhanced methods with repeatedly extracting features and extracting three types of features. The result shows that EdgeCRNN can process 11.1 audio per second on Raspberry Pi 3B+, and its accuracy rate reaches 98.05%. However, FLOPs are still relatively large on variant EdgeCRNN 1.0×, and there is still room for improvement in accuracy. Moreover, the model test platform is only on ARM CPU. In the future, we will continue to reduce the computational costs, improve the accuracy, and apply the KWS system to different environments.

References

Abdel-Hamid O, Ar Mohamed, Jiang H, Deng L, Penn G, Yu D (2014) Convolutional neural networks for speech recognition. IEEE/ACM Trans Audio Speech Lang Process 22(10):1533–1545

Anderson A, Su J, Dahyot R, Gregg D (2020) Performance-oriented neural architecture search. arXiv preprint arXiv:200102976

Arik SO, Kliegl M, Child R, Hestness J, Gibiansky A, Fougner C, Prenger R, Coates A (2017) Convolutional recurrent neural networks for small-footprint keyword spotting. arXiv preprint arXiv:170305390

Benelli G, Meoni G, Fanucci L (2018) A low power keyword spotting algorithm for memory constrained embedded systems. In: 2018 IFIP/IEEE international conference on very large scale integration (VLSI-SoC). IEEE, pp 267–272

Chen G, Parada C, Heigold G (2014) Small-footprint keyword spotting using deep neural networks. In: 2014 IEEE international conference on acoustics. speech and signal processing (ICASSP). IEEE, pp 4087–4091

Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:14061078

Coucke A, Chlieh M, Gisselbrecht T, Leroy D, Poumeyrol M, Lavril T (2019) Efficient keyword spotting using dilated convolutions and gating. In: ICASSP 2019–2019 IEEE international conference on acoustics. speech and signal processing (ICASSP). IEEE, pp 6351–6355

Custers B, Sears AM, Dechesne F, Georgieva I, Tani T, van der Hof S (2019) EU personal data protection in policy and practice. Springer, Berlin

Dey R, Salemt FM (2017) Gate-variants of gated recurrent unit (GRU) neural networks. In: 2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS). IEEE, pp 1597–1600

Dinelli G, Meoni G, Rapuano E, Benelli G, Fanucci L (2019) An FPGA-based hardware accelerator for CNNS using on-chip memories only: design and benchmarking with intel movidius neural compute stick. Int J Reconfigurable Comput 2019:7218758

Du H, Li R, Kim D, Hirota K, Dai Y (2018) Low-latency convolutional recurrent neural network for keyword spotting. In: 2018 Joint 10th international conference on soft computing and intelligent systems (SCIS) and 19th International Symposium on Advanced Intelligent Systems (ISIS). IEEE, pp 802–807

Gaff BM, Sussman HE, Geetter J (2014) Privacy and big data. Computer 47(6):7–9

Glorot X, Bordes A, Bengio Y (2011) Deep sparse rectifier neural networks. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics, pp 315–323

He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778

Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780

Howard A, Sandler M, Chu G, Chen LC, Chen B, Tan M, Wang W, Zhu Y, Pang R, Vasudevan V et al (2019) Searching for mobilenetv3. In: Proceedings of the IEEE international conference on computer vision, pp 1314–1324

Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H (2017) Mobilenets: efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:170404861

Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:150203167

Luo R, Sun T, Wang C, Du M, Tang Z, Zhou K, Gong X, Yang X (2019) Multi-layer attention mechanism for speech keyword recognition. arXiv preprint arXiv:190704536

Ma N, Zhang X, Zheng HT, Sun J (2018) Shufflenet v2: practical guidelines for efficient CNN architecture design. In: Proceedings of the European conference on computer vision (ECCV), pp 116–131

Mazzawi H, Gonzalvo X, Kracun A, Sridhar P, Subrahmanya N, Moreno IL, Park HJ, Violette P (2019) Improving keyword spotting and language identification via neural architecture search at scale. In: Proc Interspeech, vol 2019, pp 1278–1282

McFee B, Raffel C, Liang D, Ellis DP, McVicar M, Battenberg E, Nieto O (2015) Librosa: audio and music signal analysis in python. In: Proceedings of the 14th python in science conference, vol 8

Mishchenko Y, Goren Y, Sun M, Beauchene C, Matsoukas S, Rybakov O, Vitaladevuni SNP (2019) Low-bit quantization and quantization-aware training for small-footprint keyword spotting. In: 2019 18th IEEE international conference on machine learning and applications (ICMLA). IEEE, pp 706–711

Nakkiran P, Alvarez R, Prabhavalkar R and Parada C (2015) Compressing deep neural networks using a rank-constrained topology, In: Proceedings of annual conference of the international speech communication association (Interspeech). pp 1473–1477

Sainath TN, Parada C (2015) Convolutional neural networks for small-footprint keyword spotting. In: Proceeding of the Sixteenth Annual Conference of the International Speech Communication Association (Interspeech). pp 1478–1482

Sandler M, Howard A, Zhu M, Zhmoginov A, Chen LC (2018) Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4510–4520

Sifre L, Mallat S (2014) Rigid-motion scattering for image classification. Ph.D. Thesis

Silaghi MC (2005) Spotting subsequences matching an hmm using the average observation probability criteria with application to keyword spotting. In: AAAI, pp 1118–1123

Silaghi MC, Bourlard H (1999) Iterative posterior-based keyword spotting without filler models. In: Proceedings of the IEEE automatic speech recognition and understanding workshop. Citeseer, pp 213–216

Sun M, Raju A, Tucker G, Panchapagesan S, Fu G, Mandal A, Matsoukas S, Strom N, Vitaladevuni S (2016) Max-pooling loss training of long short-term memory networks for small-footprint keyword spotting. In: 2016 IEEE spoken language technology workshop (SLT). IEEE, pp 474–480

Sun M, Snyder D, Gao Y, Nagaraja VK, Rodehorst M, Panchapagesan S, Strom N, Matsoukas S, Vitaladevuni S (2017) Compressed time delay neural network for small-footprint keyword spotting. In: INTERSPEECH, pp 3607–3611

Tan M, Chen B, Pang R, Vasudevan V, Sandler M, Howard A, Le QV (2018) Resource-efficient neural architect. arXiv preprint arXiv:180607912

Tang R, Lin J (2018) Deep residual learning for small-footprint keyword spotting. 2018 IEEE International Conference on Acoustics. Speech and Signal Processing (ICASSP). IEEE, pp 5484–5488

Tang R, Wang W, Tu Z, Lin J (2018) An experimental analysis of the power consumption of convolutional neural networks for keyword spotting. In: 2018 IEEE international conference on acoustics. speech and signal processing (ICASSP). IEEE, pp 5479–5483

Tucker G, Wu M, Sun M, Panchapagesan S, Fu G, Vitaladevuni S (2016) Model compression applied to small-footprint keyword spotting. In: INTERSPEECH, pp 1878–1882

Véniat T, Schwander O, Denoyer L (2019) Stochastic adaptive neural architecture search for keyword spotting. In: ICASSP 2019–2019 IEEE international conference on acoustics. speech and signal processing (ICASSP). IEEE, pp 2842–2846

Warden P (2018) Speech commands: A dataset for limited-vocabulary speech recognition. arXiv preprint arXiv:180403209

Wilpon J, Miller L, Modi P (1991) Improvements and applications for key word recognition using hidden markov modeling techniques. In: 1991 international conference on acoustics, speech, and signal processing. IEEE, pp 309–312

Zeng M, Xiao N (2019) Effective combination of DenseNet and BiLSTM for keyword spotting. IEEE Access 7:10767–10775

Zhang B, Li W, Li Q, Zhuang W, Chu X, Wang Y (2020) Autokws: keyword spotting with differentiable architecture search. arXiv preprint arXiv:200903658

Zhang X, Zhou X, Lin M, Sun J (2018) Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 6848–6856

Zhang Y, Suda N, Lai L, Chandra V (2017) Hello edge: keyword spotting on microcontrollers. arXiv preprint arXiv:171107128