A New Lightweight CRNN Model for Keyword Spotting with Edge Computing Devices

Yungen Wei, Zheng Gong, Shunzhi Yang, Kai Ye and Yamin Wen

April 23, 2020

A New Lightweight CRNN Model for Keyword Spotting with Edge Computing Devices*

Yungen Wei[1], Zheng Gong[1], Shunzhi Yang[1], Kai Ye[1], and Yamin Wen[2]**

[1] School of Computer Science, South China Normal University, China
{genty1314, cis.gong}@gmail.com
[2] School of Statistics and Mathematics, Guangdong University of Finance and Economics, China
wenyamin@gdufe.edu.cn

Abstract. Keyword Spotting (KWS) is a significant branch of Automatic Speech Recognition (ASR), which has been widely used in edge computing devices. The goal of KWS is to provide high accuracy at 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 high accuracy, low computational costs, and low latency. In this paper, we present a new Convolutional Recurrent Neural Network (CRNN) architecture named EdgeCRNN for edge computing devices. EdgeCRNN is based on a depthwise separable convolution (DSC) and residual structure, and it uses a feature enhancement method. The experimental results on Google Speech Commands Dataset depict that EdgeCRNN can test 11.1 audio data per second on Raspberry Pi 3B+, which are 2.2 times 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, which 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 commands if the KWS system detects a predefined keyword in a dialogue.

* This paper is supported by the National Natural Sciences Foundation of China(No. 61572028), National Cryptography Development Fund (No. MMJJ20180206), the Project of Science and Technology of Guangzhou (No. 201802010044) and Guangdong Basic and Applied Basic Research Foundation (No. 2019A1515011797).
** Corresponding author.

Traditional methods of KWS usually use the Keyword/Filler Hidden Markov Model (HMM) [1, 2]. However, depending on an HMM topology, these systems require Viterbi decoding and are 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 [3] was the first deep learning model to be applied to KWS with a model parameter size of 224M, which is smaller than the 373M of the GMM-HMM , and its performance exceeds that of the HMM model. However, these model parameters are still not suitable for edge computing devices.

In addition, the KWS system uses a server-client pattern, where the client collects data on the terminal and the cloud server processes it. 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. Consequently, we adopt a new pattern that the client collects and processes data on the terminal. This model not only diminishes the burden of cloud servers and network bandwidth, but also provides positioning and high-quality services.

However, the model's high requirements for hardware resources and limited resources pose a challenge on applying KWS to edge computing devices. The hardware acceleration and designing lightweight models are used to solve this problem. In [4], Benelli et al. used a Neural Compute Stick (NCS) to accelerate and lower latency by 50%. Dinelli et al. [5] proposed a Convolutional Neural Network (CNN) based on a field-programmable gate array (FPGA), which is nearly 10 times faster than NCS. However, the hardware acceleration is costly and is not used in edge computing devices. So we choose the approach of designing lightweight model.

Various lightweight architectures for deep learning have been successfully applied to KWS problems, such as Tpool2 [6] and CNN [7]. Compared with DNN [3], CNN [7] offers a 27-44% relative improvement in false alarm rate (FAR). However, CNN ignores the global time and spectral correlation owing to the size of the convolution kernel. Recurrent Neural Network (RNN) could leverage a longer temporal context, which makes up for this question of CNN. Recently, RNN [8] and convolutional recurrent neural network (CRNN) [9] 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 [9].

In this paper, we design a new CRNN model called EdgeCRNN. Its CNN adopts depthwise separable convolution (DSC) and residual structure. Besides, we propose two feature enhancement methods LFBE-Delta and first convolution feature enhancement, and use the LFBE-Delta feature instead of the Mel-Frequency Cepstrum Coefficient (MFCC) as input features. EdgeCRNN can recognize 12 class keywords by training on the Google Speech Commands Dataset [10]. 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 Raspberry Pi 3B+ without stuttering. Besides, accuracy rate is state of the art, reaches 98.05%. The source codes of EdgeCRNN and its test samples are available at GitHub repository [1].

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 Section 3. In Section 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. In [11], George et al. use low-rank weight matrices throughout DNN, which obtains a 23.9% relative reduction in frame error rate.

The NAS can automatically design high-performance neural networks, which are gradually applied in the fields of speech recognition [12]. NAS is based on a search strategy to automatically design a model suitable for a specific application within a predefined search space [13].

The artificial design of lightweight neural network mainly reduces the amount of calculation by optimizing the calculation method of convolution and designing more efficient convolution operations. DS-CNN [14] proposes a lightweight model based on DSC and the accuracy rate reaches 95.4% with limited memory and compute capability.

The model compression and automatic design methods based on network architecture search 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 neural network method to design a lightweight KWS model for edge computing devices.

3 EdgeCRNN

In this section, first we propose a Feature Enhancement approach. And then the architecture of EdgeCRNN is designed from EdgeCRNN Block.

3.1 Feature Enhancement

To extract acoustic features more efficiently, we propose two enhancement methods Input and First Convolution Layer Feature Enhancement.

[1] https://github.com/genty1314/KWS.git

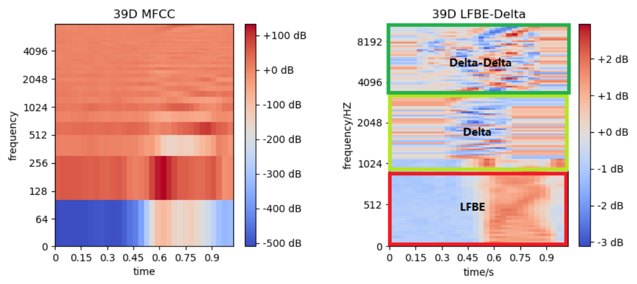

Fig. 1. Input Feature. 39 dimensions (39D) MFCC and 39D LFBE-Delta (LFBE-Delta denotes the concatenation of 13D LFBE, 13D Delta and 13D Delta-Delta).

Input Feature Enhancement The traditional method MFCC only extracts the envelope information on spectrum and it loses sound details. However, the Log-Mel filterbank energies (LFBE) contains more features, such as low-frequency and spectral details. Many proposals have adopted LFBE as feature extract method [7, 15]. Besides, the first derivative (Delta) and second derivative (Delta-Delta) features on the time axis of MFCC can better represent correlation among frames. We propose a new feature extraction method LFBE-Delta, which is 39 dimensions and computed every 30ms with a 10ms frame shift by LibROSA package [16]. LFBE-Delta contains three features with 13 dimensions, which include LFBE, Delta, and Delta-Delta (Figure 1).

First Convolution Layer Feature Enhancement The convolution kernel enhances features by multiplying input signals with sliding, then it outputs a small-size map feature. By setting convolution kernel 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 inputs, small-size inputs could relatively save computational costs. Compared with the input data dimensions of $3 \times 224 \times 224$ in the computer vision [17, 18], 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.

3.2 The Building Blocks of EdgeCRNN

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

Depthwise Separable Convolution According to Howard et al.'s research [18], the FLOPs of the DSC is $\frac{1}{N} + \frac{1}{D_k^2}$ times than the standard convolutional

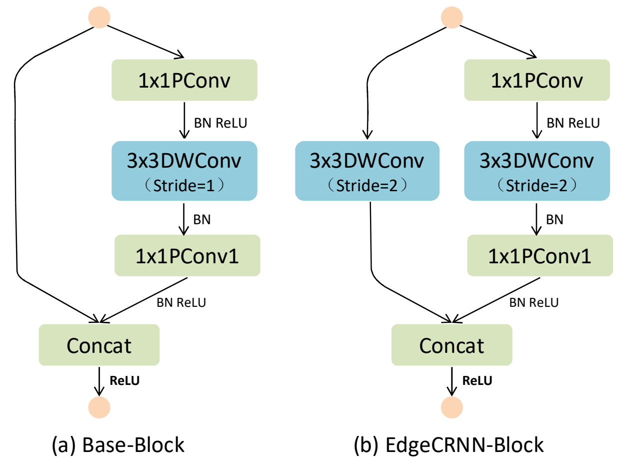

Fig. 2. EdgeCRNN Block. a) is the basic block, two branch outputs are "Concat" operation ;b) is the downsampling module with the output operate by "Concat".

operation, where N is the number of output channels, D_k is the kernel size. The number of channels is usually large, so the $\frac{1}{N}$ value can be ignored. It consists of depthwise convolution(DWConv) and point convolution(PConv) and gradually replaces standard convolution kernels in many lightweight model studies. Most EdgeCRNN's convolution kernel sizes are 3×3 and 1×1, so the computation cost of EdgeCRNN can less about 9 times than full convolution layer. This proves that the DSC can reduce computational costs and model parameters.

Residual Structure In theory, deeper networks are more capable of learning. However, with the numbers of network layer increases, the structure becomes more complicated and requires expensive computational cost. Therefore, He et al. [19] proposed ResNet based on the Residual Structure, which uses the identity mapping of shortcut connections, the input and output of different blocks are concatenated by an element-wise. It increases the training speed of the model. The Residual Structure was applied in the KWS task, and the accuracy rate was state-of-the-art at that time and reached 95.8% [20].

RNN The RNN uses a loop structure to connect early state information to the later state, which can well extract sequence data context features. However, standard RNN has short-term memory problem. The long short-term memory (LSTM) [21] and gated recurrent unit (GRU) [22] of variant RNN were created as the solution to the 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% [23]. Hence, we use LSTM on EdgeCRNN model.

Layer	Output	K	S	R	Output Channels			
					0.5x	1.0x	1.5x	2.0x
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

Table 1. EdgeCRNN architecture. Stage* include EdgeCRNN Block, K and S are the size and stride of the convolutions kernel, R represents the number of modules and padding=1.

EdgeCRNN Block We design EdgeCRNN Block based on DSC and residual structure, which is similar to the ShuffleNetV2 [24]. It includes Base-Block and EdgeCRNN-Block (Figure 2). EdgeCRNN Block consists of two PConv layers and one DWConv layer, which selects the rectified linear unit (ReLU) nonlinearity and it uses Batch Normalization (BN) to normalize input data. 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. Base-Block is the basic block and adding features by Concat operation, the input signal size and channels remain unchanged. EdgeCRNN-Block is on the first layer of each stage (see more detail in Session 3.3), and Base-Block follows it.

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 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 [8]. Conv1 is MaxPool and Conv5 uses GlobalPool. There are two units in each Stage. 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 1. The Edge-CRNN uses Width Multiplier α similar to MobilenetV1 [18]. The role of the α is to thin a network uniformly at each layer (Table 1).

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. Finally, we compare performances between EdgeCRNN and popular KWS models.

4.1 Experimental Step on EdgeCRNN

We evaluate our models by using Google Speech Commands Dataset [10], which consists of 65,000 one-second 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 background noise. The dataset is then randomly split into training, validation, and test sets 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 [6] as the baseline model, which consists of two convolutional 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 loss function on each of the convolution layers.

4.2 Model Training on EdgeCRNN

Accuracy, FLOPs and model parameters are our primary metric of quality. We also plot receiver operating characteristic (ROC) curves, where the x and y axes denote FAR and false rejects 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.

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 2. Meanwhile, EdgeCRNN-M-F and EdgeCRNN-Mel-F also have a similar relationship in accuracy. The Figure 3(b) illustrates the EdgeCRNN-Mel-F gives a 69.5% relative improvement over the EdgeCRNN-M-F at the operating point of 0.1 FAR. This means that input features contain three feature types which can improve model accuracy more than containing one feature.

The first convolution layer feature enhancement can repeatedly extract features and improve accuracy. EdgeCRNN-Mel-F is 0.8% higher than EdgeCRNN-Mel. However, FLOPs of the EdgeCRNN-Mel-F is almost 10M more than that

Model	MFLOPs	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**

Table 2. Accuracy of features enhancement. Where M denotes MFCC as feature extraction and Mel represents LFBE-Delta, F represents first convolution layer feature enhancement, the α defaults to 1.0x.

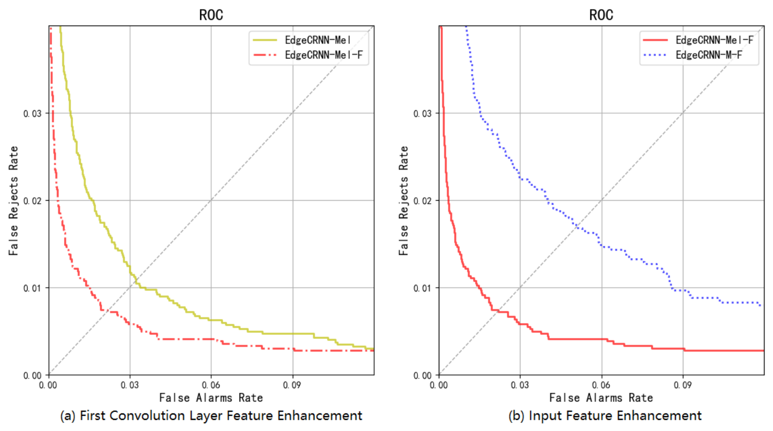

Fig. 3. ROC curves for feature enhancement.

of EdgeCRNN-Mel. We have found that it is most appropriate to reuse it only once. So EdgeCRNN uses the first convolution layer feature enhancement only once. The EdgeCRNN-Mel-F curve is lower compared with the EdgeCRNN-Mel from Figure 3(a), which depicts that EdgeCRNN extracts feature more robust by feature enhancement.

4.3 Result on EdgeCRNN

First, we compare accuracy between previous KWS models [6, 8, 9, 14, 23] and EdgeCRNN (Table 3), these models are trained on the Google Speech Commands Dataset [10] (except CRNN [9], which uses a private TalkType dataset, and the data of LSTM from literature [14]). The parameter of EdgeCRNN 1.0x is not the smallest, while it is relatively lightweight and less than 0.6M. Besides, the accuracy of EdgeCRNN is higher than other KWS models from Table 3, 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 4. EdgeCRNN 0.5x can read 11.1 audio data per second on

Model	FLOPs	Parameter	Accuracy
Tpool2[6]	103M	1.09M	91.97%
LSTM[8]	48.4M	0.26M	94.81%
CRNN[9]	19.3M	**0.22M**	97.71%
DS-CNN[14]	56.9M	0.47M	95.38%
DenseNet-BiLSTM[23]	-	0.24M	97.50%
EdgeCRNN 1.0x	**14.54M**	0.59M	**97.89%**

Table 3. Accuracy of the related KWS models.

Model	MFLOPs	Parameters	Accuracy	CPU	ARM
Tpool2 [6]	103	1.09M	91.97%	27.6/s	5.0/s
EdgeCRNN 0.5x	**4.10**	**0.29M**	97.09%	**49.9/s**	**11.1/s**
EdgeCRNN 1.0x	14.54	0.59M	97.89%	25.6/s	5.0/s
EdgeCRNN 1.5x	34.89	1.29M	97.92%	17.3/s	3.1/s
EdgeCRNN 2.0x	57.65	1.72M	**98.05**%	13.5/s	2.3/s

Table 4. Performances of different width Multiplier and platforms. The CPU denotes the test speed on a platform of Intel(R) Core(TM) I3-8100 CPU. The ARM is the Raspberry Pi 3B+.

the Raspberry Pi 3B+, which is much faster than Tpool2's 5 per second. 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 speed can keep up with the speed of human speech in a resource-constrained environment.

Table 4 compares the effects of different Width Multiplier models, which have four multiples 0.5x, 1.0x, 1.5x, 2x from Table 1. The 2.0x 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 model.

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 enhancement methods with repeatedly extracting 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 1.0x, and there is still room for improvement in accuracy. Moreover, the model test platform is only on ARM. In the future, we will continue to reduce the computational costs, improve the accuracy, and apply the KWS system to different environments.

References

1. JG Wilpon, LG Miller, and P Modi, "Improvements and applications for key word recognition using hidden markov modeling techniques," in *1991 International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 1991, pp. 309–312.
2. Marius-Calin Silaghi, "Spotting subsequences matching an hmm using the average observation probability criteria with application to keyword spotting," in *AAAI*, 2005, pp. 1118–1123.
3. Guoguo Chen, Carolina Parada, and Georg Heigold, "Small-footprint keyword spotting using deep neural networks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 4087–4091.
4. Gionata Benelli, Gabriele Meoni, and Luca Fanucci, "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, 2018, pp. 267–272.
5. Gianmarco Dinelli, Gabriele Meoni, Emilio Rapuano, Gionata Benelli, and Luca Fanucci, "An fpga-based hardware accelerator for cnns using on-chip memories only: Design and benchmarking with intel movidius neural compute stick," *International Journal of Reconfigurable Computing*, vol. 2019, 2019.
6. Raphael Tang, Weijie Wang, Zhucheng Tu, and Jimmy Lin, "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, 2018, pp. 5479–5483.
7. Tara Sainath and Carolina Parada, "Convolutional neural networks for small-footprint keyword spotting," 2015.
8. Ming Sun, Anirudh Raju, George Tucker, et al., "Max-pooling loss training of long short-term memory networks for small-footprint keyword spotting," in *2016 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2016, pp. 474–480.
9. Sercan O Arik, Markus Kliegl, Rewon Child, et al., "Convolutional recurrent neural networks for small-footprint keyword spotting," *arXiv preprint arXiv:1703.05390*, 2017.
10. Pete Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.
11. George Tucker, Minhua Wu, Ming Sun, Sankaran Panchapagesan, Gengshen Fu, and Shiv Vitaladevuni, "Model compression applied to small-footprint keyword spotting.," in *INTERSPEECH*, 2016, pp. 1878–1882.
12. Yanqi Zhou, Siavash Ebrahimi, Sercan Ö Arık, et al., "Resource-efficient neural architect," *arXiv preprint arXiv:1806.07912*, 2018.
13. Andrew Anderson, Jing Su, Rozenn Dahyot, and David Gregg, "Performance-oriented neural architecture search," *arXiv preprint arXiv:2001.02976*, 2020.
14. Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra, "Hello edge: Keyword spotting on microcontrollers," *arXiv preprint arXiv:1711.07128*, 2017.
15. Alice Coucke, Mohammed Chlieh, Thibault Gisselbrecht, David Leroy, Mathieu Poumeyrol, and Thibaut Lavril, "Efficient keyword spotting using dilated convolutions and gating," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 6351–6355.
16. Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, 2015, vol. 8.

17. Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.
18. Andrew G Howard, Menglong Zhu, Bo Chen, et al., "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.
19. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
20. Raphael Tang and Jimmy Lin, "Deep residual learning for small-footprint keyword spotting," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5484–5488.
21. Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
22. Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, et al., "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.
23. Mengjun Zeng and Nanfeng Xiao, "Effective combination of densenet and bilstm for keyword spotting," *IEEE Access*, vol. 7, pp. 10767–10775, 2019.
24. Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 116–131.