INTERSPEECH 2019
September 15–19, 2019, Graz, Austria

Temporal Convolution for Real-time Keyword Spotting on Mobile Devices

Seungwoo Choi[*], *Seokjun Seo*[*], *Beomjun Shin*[*], *Hyeongmin Byun*,
Martin Kersner, *Beomsu Kim*, *Dongyoung Kim*[†], *Sungjoo Ha*[†]

Hyperconnect, Seoul, South Korea

{seungwoo.choi, seokjun.seo, beomjun.shin, hyeongmin.byun}@hpcnt.com
{martin.kersner, beomsu.kim, dongyoung.kim, shurain}@hpcnt.com

Abstract

Keyword spotting (KWS) plays a critical role in enabling speech-based user interactions on smart devices. Recent developments in the field of deep learning have led to wide adoption of convolutional neural networks (CNNs) in KWS systems due to their exceptional accuracy and robustness. The main challenge faced by KWS systems is the trade-off between high accuracy and low latency. Unfortunately, there has been little quantitative analysis of the actual latency of KWS models on mobile devices. This is especially concerning since conventional convolution-based KWS approaches are known to require a large number of operations to attain an adequate level of performance.

In this paper, we propose a temporal convolution for real-time KWS on mobile devices. Unlike most of the 2D convolution-based KWS approaches that require a deep architecture to fully capture both low- and high-frequency domains, we exploit temporal convolutions with a compact ResNet architecture. In Google Speech Command Dataset, we achieve more than **385x** speedup on Google Pixel 1 and surpass the accuracy compared to the state-of-the-art model. In addition, we release the implementation of the proposed and the baseline models including an end-to-end pipeline for training models and evaluating them on mobile devices.

Index Terms: keyword spotting, real-time, convolutional neural network, temporal convolution, mobile device

1. Introduction

Keyword spotting (KWS) aims to detect pre-defined keywords in a stream of audio signals. It is widely used for hands-free control of mobile applications. Since its use is commonly concentrated on recognizing wake-up words (e.g., “Hey Siri” [1], “Alexa” [2, 3], and “Okay Google” [4]) or distinguishing common commands (e.g., “yes” or “no”) on mobile devices, the response of KWS should be both *immediate* and *accurate*. However, it is challenging to implement fast and accurate KWS models that meet the real-time constraint on mobile devices with restricted hardware resources.

Recently, with the success of deep learning in a variety of cognitive tasks, neural network based approaches have become popular for KWS [5, 6, 7, 8, 9, 10]. Especially, KWS studies based on convolutional neural networks (CNNs) show remarkable accuracy [6, 7, 8]. Most of CNN-based KWS approaches receive features, such as mel-frequency cepstral coefficients (MFCC), as a 2D input of a convolutional network. Even though such CNN-based KWS approaches offer reliable accuracy, they demand considerable computations to meet a performance requirement. In addition, inference time on mobile devices has not been analyzed quantitatively, but instead, indirect metrics have been used as a proxy to the latency. Zhang *et al.* [7] presented the total number of multiplications and additions performed by the whole network. Tang and Lin [8] reported the number of multiplications of their network as a surrogate for inference speed. Unfortunately, it has been pointed out that the number of operations such as additions and multiplications, is only an indirect alternative for the direct metric such as latency [11, 12, 13]. Neglecting the memory access costs and different platforms being equipped with varying degrees of optimized operations are potential sources for the discrepancy. Thus, we focus on the measurement of actual latency on mobile devices.

In this paper, we propose a temporal convolutional neural network for real-time KWS on mobile devices, denoted as *TC-ResNet*. We apply *temporal convolution*, i.e., 1D convolution along the temporal dimension, and treat MFCC as input channels. The proposed model utilizes advantages of temporal convolution to enhance the accuracy and reduce the latency of mobile models for KWS. Our contributions are as follows:

- We propose *TC-ResNet* which is a *fast* and *accurate* convolutional neural network for real-time KWS on mobile devices. According to our experiments on Google Pixel 1, the proposed model shows **385x** speedup and a 0.3%p increase in accuracy compared to the state-of-the-art CNN-based KWS model on Google Speech Commands Dataset [14].
- We release our models[1] for KWS and implementations of the state-of-the-art CNN-based KWS models [6, 7, 8] together with the complete benchmark tool to evaluate the models on mobile devices.
- We empirically demonstrate that temporal convolution is indeed responsible for reduced computation and increased performance in terms of accuracy compared to 2D convolutions in KWS on mobile devices.

2. Network Architecture

2.1. Temporal Convolution for KWS

Figure 1 is a simplified example illustrating the difference between 2D convolution and temporal convolution for KWS approaches utilizing MFCC as input data. Assuming that stride is one and zero padding is applied to match the input and the output resolution, given input $\mathbf{X} \in \mathbb{R}^{w \times h \times c}$ and weight $\mathbf{W} \in \mathbb{R}^{k_w \times k_h \times c \times c'}$, 2D convolution outputs $\mathbf{Y} \in \mathbb{R}^{w \times h \times c'}$.

[*] Equal contributions, listed in alphabetical order.
[†] Shared corresponding authors.

[1] Source code can be found at the following link: https://github.com/hyperconnect/TC-ResNet

http://dx.doi.org/10.21437/Interspeech.2019-1363

MFCC is widely used for transforming raw audio into a time-frequency representation, $\mathbf{I} \in \mathbb{R}^{t\times f}$, where t represents the time axis (x-axis in Figure 1a) and f denotes the feature axis extracted from frequency domain (y-axis in Figure 1a). Most of the previous studies [7, 8] use input tensor $\mathbf{X} \in \mathbb{R}^{w\times h\times c}$ where $w = t$, $h = f$ (or vice versa), and $c = 1$ ($\mathbf{X_{2d}} \in \mathbb{R}^{t\times f\times 1}$ in Figure 1b).

CNNs are known to perform a successive transformation of low-level features into higher level concepts. However, since modern CNNs commonly utilize small kernels, it is difficult to capture informative features from both low and high frequencies with a relatively shallow network (colored box in Figure 1b only covers a limited range of frequencies). Assuming that one naively stacks n convolutional layers of 3×3 weights with a stride of one, the receptive field of the network only grows up to $2n+1$. We can mitigate this problem by increasing the stride or adopting pooling, attention, and recurrent units. However, many models still require a large number of operations, even if we apply these methods, and has a hard time running real-time on mobile devices.

In order to implement a *fast* and *accurate* model for real-time KWS, we reshape the input from $\mathbf{X_{2d}}$ in Figure 1b to $\mathbf{X_{1d}}$ in Figure 1c. Our main idea is to treat per-frame MFCC as a time series data, rather than an intensity or grayscale image, which is a more natural way to interpret audio. We consider $\mathbf{I}$ as one-dimensional sequential data whose features at each time frame are denoted as f. In other words, rather than transforming $\mathbf{I}$ to $\mathbf{X_{2d}} \in \mathbb{R}^{t\times f\times 1}$, we set $h = 1$ and $c = f$, which results in $\mathbf{X_{1d}} \in \mathbb{R}^{t\times 1\times f}$, and feed it as an input to temporal convolution (Figure 1c). The advantages of the proposed method are as follows:

Large receptive field of audio features. In the proposed method, all lower-level features always participate in forming the higher-level features in the next layer. Thus, it takes advantage of informative features in lower layers (colored box in Figure 1c covers a whole range of frequencies), thereby avoiding stacking many layers to form higher-level features. This enables us to achieve better performance even with a small number of layers.

Small footprint and low computational complexity. Applying the proposed method, a two-dimensional feature map shrinks in size if we keep the number of parameters the same as illustrated in Figure 1b and 1c. Assuming that both conventional 2D convolution, $\mathbf{W_{2d}} \in \mathbb{R}^{3\times 3\times 1\times c}$, and proposed temporal convolution, $\mathbf{W_{1d}} \in \mathbb{R}^{3\times 1\times f\times c'}$, have the same number of parameters (i.e., $c' = \frac{3\times c}{f}$), the proposed temporal convolution requires a smaller number of computations compared to the 2D convolution (② is smaller than ① in Figure 1). In addition, the output feature map (i.e., the input feature map of the next layer) of the temporal convolution, $\mathbf{Y_{1d}} \in \mathbb{R}^{t\times 1\times c'}$, is smaller than that of a 2D convolution, $\mathbf{Y_{2d}} \in \mathbb{R}^{t\times f\times c}$. The decrease in feature map size leads to a dramatic reduction of the computational burden and footprint in the following layers, which is key to implementing fast KWS.

2.2. TC-ResNet Architecture

We adopt ResNet [15], one of the most widely used CNN architectures, but utilize $m \times 1$ kernels ($m = 3$ for the first layer and $m = 9$ for the other layers) rather than 3×3 kernels (Figure 2). None of the convolution layers and fully connected layers have biases, and each batch normalization layer [16] has trainable parameters for scaling and shifting. The identity shortcuts can be directly used when the input and the output have matching dimensions (Figure 2a), otherwise, we use an extra *conv-BN-ReLU* to match the dimensions (Figure 2b). Tang and Lin [8] also adopted the residual network, but they did not employ a temporal convolution and used a conventional 3×3 kernel. In addition, they replaced strided convolutions with dilated convolutions of stride one. Instead, we employ temporal convolutions to increase the effective receptive field and follow the original ResNet implementation for other layers by adopting strided convolutions and excluding dilated convolutions.

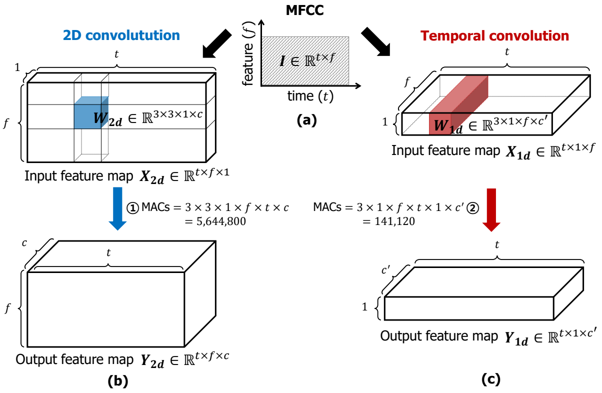

MFCC
2D convolutution
Temporal convolution
feature (f)
$\boldsymbol{I} \in \mathbb{R}^{t\times f}$
time (t)
(a)
t
1
f
$\boldsymbol{W_{2d}} \in \mathbb{R}^{3\times3\times1\times c}$
Input feature map $\boldsymbol{X_{2d}} \in \mathbb{R}^{t\times f\times 1}$
f
1
$\boldsymbol{W_{1d}} \in \mathbb{R}^{3\times1\times f\times c'}$
Input feature map $\boldsymbol{X_{1d}} \in \mathbb{R}^{t\times 1\times f}$
① MACs = $3\times3\times1\times f\times t\times c$ = 5,644,800
MACs = $3\times1\times f\times t\times1\times c'$ ② = 141,120
c
Output feature map $\boldsymbol{Y_{2d}} \in \mathbb{R}^{t\times f\times c}$
(b)
c'
Output feature map $\boldsymbol{Y_{1d}} \in \mathbb{R}^{t\times 1\times c'}$
(c)

Figure 1: *A simplified example illustrating the difference between 2D convolution and temporal convolution. (a) MFCC. (b) 2D convolution for conventional CNN-based KWS approaches. (c) Proposed temporal convolution. Note that both the parameters of a conventional 2D convolution and that of the temporal convolution have the same size in this example by setting $t = 98$, $f = 40$, $c = 160$, and $c' = 12$.*

We select ***TC-ResNet8*** (Figure 2c), which has three residual blocks and $\{16, 24, 32, 48\}$ channels for each layer including the first convolution layer, as our base model. ***TC-ResNet14*** (Figure 2d) expands the network by incorporating twice as much residual blocks compared to *TC-ResNet8*.

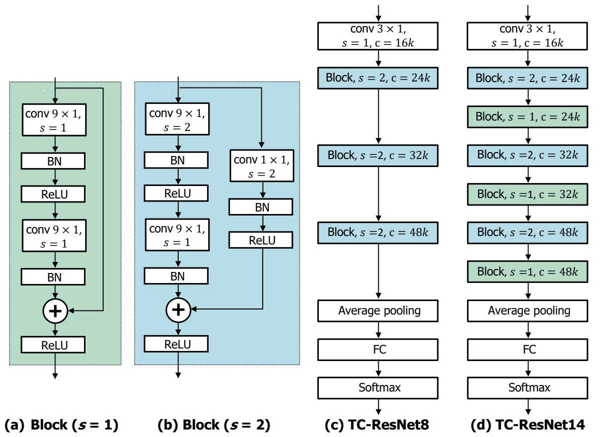
We introduce width multiplier [17] (k in Figure 2c and Figure 2d) to increase (or decrease) the number of channels at each layer, thereby achieving flexibility in selecting the right capacity model for given constraints. For example, in *TC-ResNet8*, a width multiplier of 1.5 expands the model to have $\{24, 36, 48, 72\}$ number of channels respectively. We denote such a model by appending a multiplier suffix such as ***TC-ResNet8-1.5***. ***TC-ResNet14-1.5*** is created in the same manner.

3. Experimental Framework

3.1. Experimental Setup

Dataset. We evaluated the proposed models and baselines [6, 8, 7] using *Google Speech Commands Dataset* [14]. The dataset contains 64,727 one-second-long utterance files which are recorded and labeled with one of 30 target categories. Following Google's implementation [14], we distinguish 12 classes: *"yes", "no", "up", "down", "left", "right", "on", "off", "stop", "go", silence*, and *unknown*. Using SHA-1 hashed name of the audio files, we split the dataset into training, validation, and test sets, with 80% training, 10% validation, and 10% test, respectively.

Data augmentation and preprocessing. We followed Google's preprocessing procedures which apply random shift and noise injection to training data. First, in order to generate background noise, we randomly sample and crop background



(a) Block (s = 1) (b) Block (s = 2) (c) TC-ResNet8 (d) TC-ResNet14

Figure 2: *The building block (denoted Block) of TC-ResNet when (a) stride = 1 and (b) stride = 2. (c) Architecture for TC-ResNet8 and (d) TC-ResNet14. Each of them utilizes ResNet8 and ResNet14 as the backbone-CNN, respectively. BN and FC denote batch normalization and fully connected layer. Note that 's', 'c', and 'k' indicates stride, channel size, and width multiplier, respectively.*

noises provided in the dataset, and multiply it with a random coefficient sampled from uniform distribution, $U(0, 0.1)$. The audio file is decoded to a float tensor and shifted by s seconds with zero padding, where s is sampled from $U(-0.1, 0.1)$. Then, it is blended with the background noise. The raw audio is decomposed into a sequence of frames following the settings of the previous study [8] where the window length is 30 *ms* and the stride is 10 *ms* for feature extraction. We use 40 MFCC features for each frame and stack them over time-axis.

Training. We trained and evaluated the models using TensorFlow [18]. We use a weight decay of 0.001 and dropout with a probability of 0.5 to alleviate overfitting. Stochastic gradient descent is used with a momentum of 0.9 on a mini-batch of 100 samples. Models are trained from scratch for 30k iterations. Learning rate starts at 0.1 and is divided by 10 at every 10k iterations. We employ early stopping [19] with the validation split.

Evaluation. We use *accuracy* as the main metric to evaluate how well the model performs. We trained each model 15 times and report its average performance. *Receiver operating characteristic (ROC) curves*, of which the x-axis is the false alarm rate and the y-axis is the false reject rate, are plotted to compare different models. To extend the ROC curve to multi-classes, we perform micro-averaging over multiple classes per experiment, then vertically average them over the experiments for the final plot.

We report the number of operations and parameters which faithfully reflect the real-world environment for mobile deployment. Unlike previous works which only reported the numbers for part of the computation such as the number of multiply operations [8] or the number of multiplications and additions only in the matrix-multiplication operations [7], we include *FLOPs* [20], computed by TensorFlow profiling tool [21], and the number of *all* parameters instead of only trainable parameters reported by previous studies [8].

Inference speed can be estimated by FLOPs but it is well known that FLOPs are not always proportional to speed. Therefore, we also measure *inference time* on a mobile device using the TensorFlow Lite Android benchmark tool [22]. We measured inference time on a Google Pixel 1 and forced the model to be executed on a single little core in order to emulate the always-on nature of KWS. The benchmark program measures the inference time 50 times for each model and reports the average. Note that the inference time is measured from the first layer of models that receives MFCC as input to focus on the performance of the model itself.

3.2. Baseline Implementations

We carefully selected baselines and verified advantages of the proposed models in terms of accuracy, the number of parameters, FLOPs, and inference time on mobile devices. Below are the baseline models:

- ***CNN-1*** and ***CNN-2*** [6]. We followed the implementations of [7] where window size is 40 *ms* and the stride is 20 *ms* using 40 MFCC features. *CNN-1* and *CNN-2* represent *cnn-trad-fpool3* and *cnn-one-fstride4* in [6], respectively.
- ***DS-CNN-S***, ***DS-CNN-M***, and ***DS-CNN-L*** [7]. *DS-CNN* utilizes depthwise convolutions. It aims to achieve the best accuracy when memory and computation resources are constrained. We followed the implementation of [7] which utilizes 40 *ms* window size with 20 *ms* stride and only uses 10 MFCCs to reduce the number of operations. *DS-CNN-S*, *DS-CNN-M*, and *DS-CNN-L* represent small-, medium-, and large-size model, respectively.
- ***Res8***, ***Res8-Narrow***, ***Res15***, and ***Res15-Narrow*** [8]. *Res*-variants employ a residual architecture for keyword spotting. The number following *Res* (e.g., 8 and 15) denotes the number of layers and the *-Narrow* suffix represents that the number of channels is reduced. *Res15* has shown the best accuracy with Google Speech Commands Dataset among the KWS studies which are based on CNNs. The window size is 30 *ms*, the stride is 10 *ms*, and MFCC feature size is 40.

We release our end-to-end pipeline codebase for training, evaluating, and benchmarking the baseline models and together with the proposed models. It consists of TensorFlow implementation of models, scripts to convert the models into the TensorFlow Lite models that can run on mobile devices, and the pre-built TensorFlow Lite Android benchmark tool.

4. Experimental Results

4.1. Google Speech Command Dataset

Table 1 shows the experimental results. Utilizing advantages of temporal convolutions, we improve the inference time measured on mobile device dramatically while achieving better accuracy compared to the baseline KWS models. *TC-ResNet8* achieves 29x speedup while improving 5.4%p in accuracy compared to *CNN-1*, and improves 11.5%p in accuracy while maintaining a comparable latency to *CNN-2*. Since *DS-CNN* is designed for the resource-constrained environment, it shows better accuracy compared to the naive CNN models without using large number of computations. However, *TC-ResNet8* achieves 1.5x / 4.7x / 15.3x speedup, and improves 1.7%p / 1.2%p / 0.7%p accuracy compared to *DS-CNN-S* / *DS-CNN-M* / *DS-CNN-L*, respectively. In addition, the proposed models show better accuracy and speed compared to *Res* which shows the best accuracy among baselines. *TC-ResNet8* achieves 385x speedup while improving 0.3%p accuracy compared to deep and complex *Res*

Table 1: *Comparison of the baseline models and the proposed models. The numbers marked with $\star$ are taken from the paper. The best result (accuracy and latency) among different approaches are displayed in bold.*

Model	Acc. (%)	Time (*ms*)	FLOPs	Params
CNN-1	90.7*	32	76.1M	524K
CNN-2	84.6*	**1.2**	1.5M	148K
DS-CNN-S	94.4*	1.6	5.4M	24K
DS-CNN-M	94.9*	5.2	19.8M	140K
DS-CNN-L	95.4*	16.8	56.9M	420K
Res8-Narrow	90.1*	47	143.2M	20K
Res8	94.1*	174	795.3M	111K
Res15-Narrow	94.0*	107	348.7M	43K
Res15	**95.8***	424	1950.0M	239K
TC-ResNet8	96.1	**1.1**	3.0M	66K
TC-ResNet8-1.5	96.2	2.8	6.6M	145K
TC-ResNet14	96.2	2.5	6.1M	137K
TC-ResNet14-1.5	**96.6**	5.7	13.4M	305K

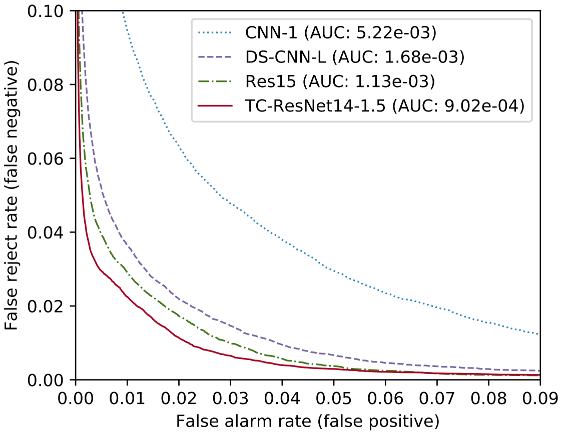


Figure 3: *ROC curves for selected models with corresponding values of AUC.*

baseline, *Res15*. Compared to a slimmer *Res* baseline, *Res8-Narrow*, proposed *TC-ResNet8* achieves 43x speedup while improving 6%p accuracy. Note that our wider and deeper models (e.g., *TC-ResNet8-1.5*, *TC-ResNet14*, and *TC-ResNet14-1.5*) achieve better accuracy at the expense of inference speed.

We also plot the ROC curves of models which depict the best accuracy among their variants: *CNN-1*, *DS-CNN-L*, *Res15*, and *TC-ResNet14-1.5*. As presented in Figure 3, *TC-ResNet14-1.5* is less likely to miss target keywords compared to other baselines assuming that the number of incorrectly detected keywords is the same. The small area under the curve (AUC) means that the model would miss fewer target keywords on average for various false alarm rates. *TC-ResNet14-1.5* shows the smallest AUC, which is critical for good user experience with KWS system.

4.2. Impact of Temporal Convolution

We demonstrate that the proposed method could effectively improve both accuracy and inference speed compared to the baseline models which treat the feature map as a 2D image. We further explore the impact of the temporal convolution by comparing variants of *TC-ResNet8*, named *2D-ResNet8* and *2D-ResNet8-Pool*, which adopt a similar network architecture and the number of parameters but utilize 2D convolutions.

Table 2: *Comparison of TC-ResNet variants, 2D-ResNet8 and 2D-ResNet8-Pool, which utilize 2D convolutions while retaining the architecture and the number of parameters of TC-ResNet8.*

Model	Acc. (%)	Time (*ms*)	FLOPs	Params
2D-ResNet8	96.1	10.1	35.8M	66K
2D-ResNet8-Pool	94.9	3.5	4.0M	66K

We designed *2D-ResNet8*, whose architecture is identical to *TC-ResNet8* except for the use of 3 × 3 2D convolutions. *2D-ResNet8* (in Table 2) shows comparable accuracy, but is 9.2x slower compared to *TC-ResNet8* (in Table 1). *TC-ResNet8-1.5* is able to surpass *2D-ResNet8* while using less computational resources.

We also demonstrate the use of temporal convolution is superior to other methods of reducing the number of operations in CNNs such as applying a pooling layer. In order to reduce the number of operations while minimizing the accuracy loss, *CNN-1*, *Res8*, and *Res8-Narrow* adopt average pooling at an early stage, specifically, right after the first convolution layer. We inserted an average pooling layer, where both the window size and the stride are set to 4, after the first convolution layer of *2D-ResNet8*, and named it *2D-ResNet8-Pool*. *2D-ResNet8-Pool* improves inference time with the same number of parameters, however, it loses 1.2%p accuracy and is still 3.2x slower compared to *TC-ResNet8*.

5. Related Works

Recently, there has been a wide adoption of CNNs in KWS. Sainath *et al.* [6] proposed small-footprint CNN models for KWS. Zhang *et al.* [7] searched and evaluated proper neural network architectures within memory and computation constraints. Tang and Lin [8] exploited residual architecture and dilated convolutions to achieve further improvement in accuracy while preserving compact models. In previous studies [6, 7, 8], it has been common to use 2D convolutions for inputs with time-frequency representations. However, there has been an increase in the use of 1D convolutions in acoustics and speech domain [23, 24]. Unlike previous studies [23, 24] our work applies 1D convolution along the temporal axis of time-frequency representations instead of convolving along the frequency axis or processing raw audio signals.

6. Conclusion

In this investigation, we aimed to implement *fast* and *accurate* models for real-time KWS on mobile devices. We measured inference speed on the mobile device, Google Pixel 1, and provided quantitative analysis of conventional convolution-based KWS models and our models utilizing temporal convolutions. Our proposed model achieved 385x speedup while improving 0.3%p accuracy compared to the state-of-the-art model. Through ablation study, we demonstrated that temporal convolution is indeed responsible for the dramatic speedup while improving the accuracy of the model. Further studies analyzing the efficacy of temporal convolutions for a diverse set of network architectures would be worthwhile.

7. References

[1] S. Sigtia, R. Haynes, H. Richards, E. Marchi, and J. Bridle, "Efficient voice trigger detection for low resource hardware," in *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2018.

[2] M. Sun, D. Snyder, Y. Gao, V. Nagaraja, M. Rodehorst, S. Panchapagesan, N. Strom, S. Matsoukas, and S. Vitaladevuni, "Compressed time delay neural network for small-footprint keyword spotting," in *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2017.

[3] G. Tucker, M. Wu, M. Sun, S. Panchapagesan, G. Fu, and S. Vitaladevuni, "Model compression applied to small-footprint keyword spotting," in *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2016.

[4] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014.

[5] Z. Wang, X. Li, and J. Zhou, "Small-footprint keyword spotting using deep neural network and connectionist temporal classifier," *arXiv preprint arXiv:1709.03665*, 2017.

[6] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2015.

[7] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," *arXiv preprint arXiv:1711.07128*, 2017.

[8] R. Tang and J. Lin, "Deep residual learning for small-footprint keyword spotting," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[9] D. C. de Andrade, S. Leo, M. L. D. S. Viana, and C. Bernkopf, "A neural attention model for speech command recognition," *arXiv preprint arXiv:1808.08929*, 2018.

[10] S. Ö. Arik, M. Kliegl, R. Child, J. Hestness, A. Gibiansky, C. Fougner, R. Prenger, and A. Coates, "Convolutional recurrent neural networks for small-footprint keyword spotting," in *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2017.

[11] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[12] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," *arXiv preprint arXiv:1807.11626*, 2018.

[13] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[14] P. Warden. (2017, August) Launching the speech commands dataset. [Online]. Available: https://ai.googleblog.com/2017/08/launching-speech-commands-dataset.html

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the Internal Conference on Machine Learning (ICML)*, 2015.

[17] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[18] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "TensorFlow: A system for large-scale machine learning." in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[19] L. Prechelt, "Early stopping-but when?" in *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[20] S. Arik, H. Jun, and G. Diamos, "Fast spectrogram inversion using multi-head convolutional neural networks," *arXiv preprint arXiv:1808.06719*, 2018.

[21] TensorFlow Profiler and Advisor. [Online]. Available: https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/profiler/README.md

[22] TFLite Model Benchmark Tool. [Online]. Available: https://github.com/tensorflow/tensorflow/tree/r1.13/tensorflow/lite/tools/benchmark/

[23] H. Lim, J. Park, K. Lee, and Y. Han, "Rare sound event detection using 1d convolutional recurrent neural networks," in *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2017 Workshop*, 2017.

[24] K. Choi, G. Fazekas, M. Sandler, and K. Cho, "Convolutional recurrent neural networks for music classification," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017.