Interspeech 2024
1-5 September 2024, Kos, Greece

RepTor: Re-parameterizable Temporal Convolution for Keyword Spotting via Differentiable Kernel Search

Eunik Park, Daehyun Ahn, Hyungjun Kim

SqueezeBits Inc., Republic of Korea
{eunik.park, daehyun.ahn, hyungjun.kim}@squeezebits.com

Abstract

Keyword spotting (KWS) is an essential in speech recognition, particularly for smart device interfaces. However, it is challenging to design KWS models that are both efficient and accurate, especially for mobile devices with limited resources. In this paper, we propose **RepTor**-family models, where a structural re-parameterization technique was applied to temporal convolutions in order to improve the efficiency of KWS systems in various environments. The re-parameterized RepTor models have a single-branch plain feed-forward architecture without any residual connections and, therefore, can be efficiently computed on mobile devices. We further propose a latency-aware differentiable kernel search method to find the optimal kernel size for each convolution layer. The proposed RepTor-A∼E models achieve 97.9% top-1 accuracy on the Google Speech Command dataset v2 with a latency of 183μs on the Galaxy S10. They outperform state-of-the-art KWS models regarding the accuracy-latency trade-off.

Index Terms: keyword spotting, efficient neural network, neural architecture search, structural re-parameterization

1. Introduction

Speech recognition has gained momentum due to its potential in human-computer interaction. While deep learning advancements improve performance, they also lead to computationally expensive models, which hinder their use in always-on scenarios on resource-constrained devices. Keyword spotting (KWS) emerges as a solution for efficient speech recognition on edge devices. It identifies specific keywords to trigger actions like "Hey Siri" [1] or "Okay Google" [2]. To be effective, KWS models require both high accuracy and low latency.

Recent research has actively explored the design of efficient KWS systems. Convolutional neural network (CNN)-based models have garnered significant attention due to their promising trade-off between accuracy and latency on mobile devices. These models can be broadly categorized into two approaches. The first is traditional 2D convolution-based models [2–4], representing the audio features in the form of time×frequency. The second is temporal convolution-based models [5–7], which reshape the 2D audio feature into 1D feature maps with the frequency axis as the channel direction and a fixed height of 1. It is difficult for 2D convolution-based models to capture informative features from a large range of frequencies due to the limited size of the convolution kernel size. In contrast, temporal convolution-based models can easily capture such features with much less computation. Therefore, temporal convolution-based models are widely chosen for efficient KWS systems.

Structural re-parameterization [8–12] presents a novel technique to enhance CNN efficiency. This approach decouples the model structure for training and inference, enabling a multi-branch model for enhanced training capacity during the training phase. At inference time, the model is re-parameterized into a single-branch architecture for faster speed. This process involves fusing convolution layers, where the largest kernel size dictates both computational cost (latency) and representational power. Therefore, it is important to find which kernel size is optimal for the largest kernel.

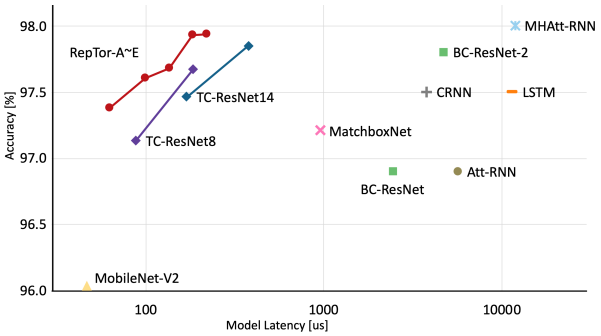


Figure 1: *Accuracy vs. latency of various KWS models on GSC dataset v2. Latency was measured on Galaxy S10. Note that the x-axis is in log scale.*

In this paper, we propose a novel approach for building efficient KWS systems using structural re-parameterization and Neural Architecture Search (NAS). While existing methods focus on re-parameterizing 2D convolutions [9–11], we explore its application to temporal convolutions with varying kernel sizes for KWS tasks. We investigate the optimal maximum kernel size for the entire model and for each individual layer, considering the accuracy-latency trade-off. To efficiently navigate the vast search space of possible configurations, we introduce a differentiable kernel search method inspired by NAS. This method leads to the discovery of RepTor-(A∼E) models, a family of KWS models that are both compact due to re-parameterization and latency-efficient thanks to NAS. As shown in Figure 1, our RepTor models achieve superior accuracy-latency performance compared to prior KWS solutions. Our contributions are as follows:

- We propose RepTor-family models that leverage structural re-parameterization on temporal convolutions to find the optimal kernel size for fast and accurate KWS.
- We propose a differentiable kernel search technique to efficiently explore the optimal kernel size for each layer in RepTor models.
- RepTor-family models show better accuracy-latency trade-off on various mobile environments.

10.21437/Interspeech.2024-233

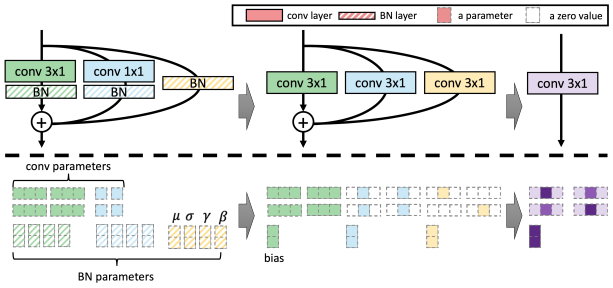


Figure 2: *Structural re-parameterization of a RepTorBlock-3 when stride = 1 and $C_{in} = C_{out} = 2$. The re-parameterization process in the view of model architecture is depicted above the horizontal dashed line. The re-parameterization process in the view of model parameters is depicted below the horizontal dashed line.*

2. Structural re-parameterization of Temporal Convolution

2.1. Re-parameterization process of temporal convolution

The key idea of the structural re-parameterization technique is to decouple the train-time graph and inference-time graph of a neural network. At training time, multiple convolution layers followed by batch-normalization layers form one layer in parallel. These convolution layers have different kernel sizes but share the same input feature maps, and the outputs of the convolution layers are accumulated. In the case of temporal convolution, convolution layers with $k \times 1$ kernel ($k \in \{9, 7, 5, 3, 1\}$) and an identity path can compose a RepTorBlock as shown in Figure 2. For RepTorBlock-k_{max}, convolution layers with kernel size less than or equal to k_{max} are used in parallel. The identity path, which consists of only a batch-normalization layer, is additionally used when the shapes of the input and output of a RepTorBlock are the same.

After training, all the branches in each RepTorBlock are fused into a single convolution layer with $k_{max} \times 1$ kernel through a re-parameterization technique. Figure 2 shows the structural re-parameterization process of RepTorBlock-3. In the RepTorBlock-3, two branches with convolution layers and one identity branch are used in parallel at training time. Two convolution layers have 3×1 and 1×1 kernel, each followed by a batch-normalization layer. Since the forward computation of a batch normalization layer is affine transformation at inference time, it can be fused to the preceding convolution layer. Four parameters $(\mu, \sigma, \gamma, \beta)$ in each batch normalization layer are fused to weights and biases of the preceding convolution layer. Then, all the branches are extended to convolution layers with a kernel size of $k_{max} \times 1$, which is 3×1 in this case, by zero-padding the convolution kernels. When multiple convolution layers use the same size of kernel, use the same input feature map and the outputs of them are accumulated, they can fused into a single convolution layer by adding all the weights and biases. As a result, multi-branch complex RepTorBlock-3 can be simplified to a single convolution layer with 3×1 kernel at inference time.

As k_{max} increases, the representation power of a RepTorBlock increases since more convolution layers with larger kernel sizes can be used during training. However, the compute cost of the RepTorBlock also increases depending on k_{max}. Therefore, finding optimal k_{max} for RepTorBlocks is essential to design efficient model architecture.

2.2. Raptor-k Network Architecture

To investigate the effect of the re-parameterization technique, we propose RepTor-k models, which are slightly modified versions of TC-ResNet8 [5]. First, we removed all the skip connections in TC-ResNet8. Then, all the temporal convolution layers are replaced by RepTorBlock-k. We set all the RepTorBlocks to have the same k value while varying k from 3 to 9. Total 6 RepTorBlock-k are followed by the first convolution layer, and a fully-connected layer with global average pooling was used for the classifier. Channel size was set to {16, 24, 24, 32, 32, 48, 48} in order while feature width was halved at the second, fourth, and sixth convolution layers with stride=2.

To evaluate RepTor-k models with different scales, we further modified the width of each layer as proposed in [13]. We increased the channel size of each convolution layer in a RepTor-k model by τ times and denoted the model as RepTor-k-τ.

3. Differentiable search for optimal kernel size of RepTorBlocks

In the previous section, we observed that the re-parameterization technique greatly improves the accuracy and/or latency of KWS models. However, it is difficult to choose proper k_{max} for each RepTorBlock. As maximum kernel size increases, the accuracy of the model improves while the latency of the model also increases. To design accurate and efficient KWS models for resource-constrained environments, it is important to find the optimal k_{max} for each RepTorBlock in a model. While we fixed k_{max} for all RepTorBlocks in the previous section, here we propose to find optimal k_{max} value for each RepTorBlock by differentiable NAS approach to achieve better accuracy-latency trade-off.

We basically followed DARTS [14] to search for the optimal kernel size using gradient descent. Let O be the set of candidate operations per layer. For every j-th candidate operation of i-th layer, a continuous architecture parameter $\alpha_{i,j}$ is assigned. During the search stage, each $\alpha_{i,j}$ is trained by back-propagation. The output of i-th layer (y_i) can be represented as

$$y_i = \sum_{j=1}^{|O_i|} \left(\frac{\exp(\alpha_{i,j})}{\sum_{k=1}^{|O_i|} \exp(\alpha_{i,k})} \cdot o_{i,j}(x_i) \right) \quad (1)$$

where $o_{i,j}$ denotes the output of j-th candidate operation of i-th layer and $|O_i|$ denotes the number of elements in the set O_i. An example of search space of two consecutive layers is shown in Figure 3. For each layer, RepTorBlock-$\{5, 7, 9\}$ are included in the search space. Since RepTorBlock-3 showed poor accuracy (Figure 4), we omitted RepTorBlock-3 from the search space.

The optimal architecture search is conducted by alternately updating model weights and architecture parameters. First, the training dataset is used to update model weights and biases while architecture parameters are fixed. Then, the validation dataset is used to update the architecture parameters while model weights are fixed. To stabilize the search process and mitigate overfitting, we also adopted perturbation-based regularization to stabilize the search process and prevent overfitting as introduced in [15]. When updating model weights, a small amount of noise (ϵ) is added to architecture parameters. ϵ values are randomly sampled from a uniform distribution, and the range of the uniform distribution is increased from $U(-0.03, 0.03)$ to $U(-0.3, 0.3)$ linearly during training. This perturbation noise aids in identifying more robust architectures and plays a crucial role in preventing overfitting in NAS.

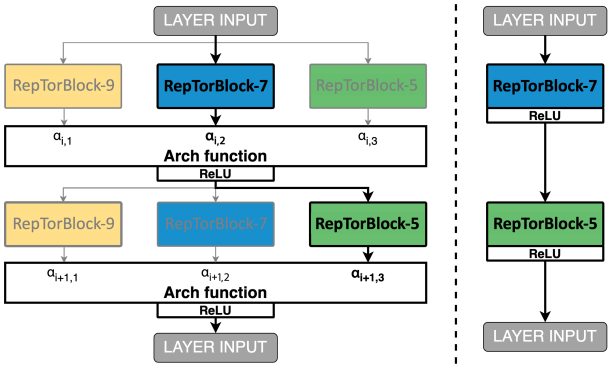


Figure 3: ***Left:*** *i-th and (i+1)-th layers with various candidate operations. Each candidate operation (RepTorBlock-k) has a dedicated architecture parameter (α).* ***Right:*** *The resulting architecture of differentiable kernel search. It was assumed that RepTorBlock-$\{7, 5\}$ was selected at $\{i, i+1\}$-th layer.*

If we use cross-entropy loss only to search for optimal architecture, the largest kernel size will always be chosen. To consider the latency of the searched model, we used an additional loss term related to the measured latency. The total loss term can be expressed as

$$L_{tot} = L_{CE} + \mu \sum_{i=1}^{N} \left(\sum_{j=1}^{|O_i|} \left(\frac{\exp(\alpha_{i,j})}{\sum_{k=1}^{|O_i|} \exp(\alpha_{i,k})} \cdot R_{i,j} \right) \right) \quad (2)$$

where L_{CE} denotes cross-entropy loss, μ denotes latency loss coefficient, N denotes the number of searching layers, and $R_{i,j}$ denotes the latency of j-th candidate operation of i-th layer. In our setting, 10^{-4} was used for μ. Latency of RepTorBlock-k with channel size c is the same as that of $k \times 1$ convolution layer with channel size c. Therefore, the latency of each candidate operation was easily measured on the target hardware.

4. Experiments

4.1. Experimental Setup

RepTor-family models were evaluated for accuracy and latency using Google Speech Command dataset (GSC) v2 [16], a dataset of 105,829 1-second speaker audio samples from 2,618 individuals. A total of 12 classes are used for keyword spotting. These include 10 out of 35 keywords (*"yes", "no", "up", "down", "left", "right", "on", "off", "stop",* and *"go"*), an *"unknown"* class which consists of data sampled from the other keywords, and a *"silence"* class containing empty audio data. Note that both the *"unknown"* and *"silence"* classes contain the similar amount of data as the other 10 classes to balance the number of samples across all keywords. The dataset with the 12 classes is divided into 36,922 training, 4,443 validation, and 4,888 test samples. Raw audio inputs are sampled at 16 kHz where the window size is 30 ms and the stride size is 10 ms. We used 40 MFCC features per frame. Data augmentation included time shifting, noise injection, and SpecAugment [17] with frequency masking, time masking, and time warping.

The optimal architecture was searched for 100 epochs using stochastic gradient descent (SGD) with a momentum of 0.9, weight decay of 3e-4, and batch size of 64. The initial learning rate was set to 0.025 and reduced following cosine annealing scheduler. After 100 epochs of searching, the optimal architecture was obtained based on architecture parameters in each layer.

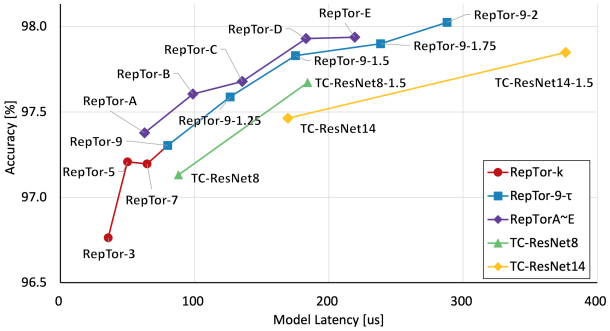


Figure 4: *Test accuracy vs. latency of TC-ResNet-family and RepTor-family models.*

Table 1: *Test accuracy and model latency of TC-ResNet-family and the proposed RepTor-k-family models.*

Model	Acc. (%)	Latency (us)	MACs	Params
RepTor-3	96.76	36	638k	22.5K
TC-ResNet8	97.13	88	1.60M	65.2K
RepTor-5	97.21	50	929k	35.7K
RepTor-7	97.20	65	1.22M	48.9K
RepTor-9	97.30	80	1.51M	62.1K
RepTor-A	**97.38**	63	1.19M	42.9K
TC-ResNet14	97.46	169	3.17M	136K
RepTor-9-1.25	**97.59**	126	2.30M	96.1K
RepTor-B	97.60	99	1.80M	66.1K
TC-ResNet8-1.5	97.67	184	3.43M	144K
RepTor-C	97.68	135	2.53M	94.4K
RepTor-9-1.5	**97.83**	175	3.25M	138K
TC-ResNet14-1.5	97.85	376	6.94M	303K
RepTor-D	97.93	183	3.39M	128K
RepTor-9-1.75	97.90	239	4.36M	186K
RepTor-E	97.94	219	4,37M	165K
RepTor-9-2.0	**98.02**	288	5.64M	243k

All RapTor-family models, including the aforementioned searched model, were trained with SGD optimizer with momentum of 0.9, weight decay of 1e-4 and batch size of 128 for 300 epochs. The initial learning rate of 0.02 was used and was modified by 5 warm-up steps and cosine annealing schedule. Dropout with a ratio of 0.2 was applied to the input of the classifier layer.

The models were trained using Pytorch and then converted to TensorFlow Lite (TFLite). Latency was measured using TFLite with XNNPACK delegate on a mobile CPU single thread in Galaxy S10. The average latency of the 300 runs is reported using TFLite benchmark tool [18]. Note that speech feature extraction processes such as MFCC were not included in latency measurement. The average latency of the 300 runs is reported.

4.2. Results on structural re-parameterization

Table 1 shows accuracy-latency trade-off of the proposed RepTor family models and baseline TC-ResNet models. All four RepTor-$\{3, 5, 7, 9\}$ models showed lower latency than TC-ResNet8 model which is the fastest model among TC-ResNet family. While RepTor-3 model showed 96.76% test accuracy which is lower than TC-ResNet8, RepTor-$\{5, 7, 9\}$ models showed higher test accuracy than TC-ResNet8 with lower latency. For example, RepTor-9 achieved 0.91× lower latency

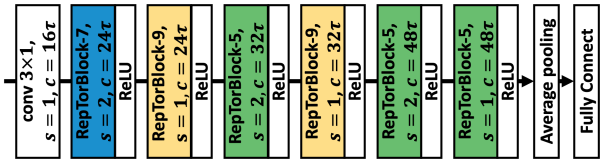


Figure 5: *Model architecture obtained by differentiable kernel search. The model is used as base model for RepTor-A∼E.*

with 0.17% higher accuracy compared to TC-ResNet8. We further compared RepTor-9-τ models ($\tau \in \{1.25, 1.5, 1.75, 2.0\}$) and larger TC-ResNet models. As τ increases, both latency and accuracy of RepTor-9 model increase. While RepTor-9-1.5, TC-ResNet8-1.5, and TC-ResNet14 models showed similar latency about 175 us, RepTor-9-1.5 model achieved 0.16% and 0.37% higher accuracy than TC-ResNet8-1.5 and TC-ResNet14 models, respectively. Table 1 also shows the number of MACs and the number of parameters measured by open-source Thop library [19]. While some previous works sacrificed another metric to improve one metric [20, 21], the proposed RepTor models achieved better accuracy-latency trade-off without sacrificing any other metric significantly.

4.3. Results on the searched models with differentiable method

Figure 5 shows the obtained RepTor model using the proposed search method. We denoted the model as RepTor-A, and scaled the model to RepTor-B∼E by multiplying $\tau = \{1.25, 1.5, 1.75, 2\}$ to the channel size. The accuracy-latency trade-off of the hand-crafted RepTor-k models and the searched RepTor-A∼E models is shown in Figure 4. The differentiable kernel search method further improved the accuracy-latency trade-off of RepTor models. For example, RepTor-A model achieved 1.27x speed-up and 0.08% accuracy improvement compared to RepTor-9 model. Note that RepTor-k models already achieved lower latency and higher accuracy than previous models, and RepTor-A∼E models are even better.

Comparison with previous state-of-the-art KWS models is shown in Figure 1. Except for TC-ResNet family and MobileNet-V2, recent works achieved high accuracy while sacrificing latency a lot. MatchboxNet-3x2x64 [7] achieved 97.21% test accuracy with 951 us latency. LSTM [21], BC-ResNet [20], MHAtt-RNN [21], CRNN [22] and Att-RNN [23] took more than 1000 us which is 5x slower than the largest RepTor-E model. MHAtt-RNN model showed the highest accuracy (98.0%) but is 54× slower than RepTor-E that achieved 97.94% accuracy. MobileNet-V2 [24] was the fastest among all models. However, RepTor-A achieved 1.34% higher accuracy with only 16 us latency overhead.

5. Discussion

5.1. Effect of re-parameterization technique

We evaluated the effect of re-parameterization technique by comparing RepTor-9 model and RepTor-9-vanilla model. RepTor-9-vanilla model has the same architecture as RepTor-9 at inference phase, but the same architecture was also used at training phase. In other words, all the RepTorBlock-k in RepTor-9 model were replaced by plain Conv $k \times 1$ layers for the RepTor-9-vanilla model. By comparing the accuracies of the two models, we can evaluate the effect of the re-parameterization technique.

Table 2 shows the test accuracies of RepTor-9 and RepTor-9-vanilla models with different width multipliers (τ). Since RepTor-9 model exploits multi-branch convolution layers and skips connection during training, it shows higher test accuracy compared to RepTor-9-vanilla model within the same width multiplier. At inference time, RepTor-9 and RepTor-9-vanilla have exactly the same architecture. However RepTor-9 is more efficient in terms of accuracy-latency trade-off.

Table 2: *Test accuracy (%) of RepTor-9 and RepTor-9-vanilla*

width multiplier	1	1.25	1.5	1.75	2
RepTor-9	97.30	97.59	97.83	97.90	98.02
RepTor-9-vanilla	97.05	97.38	97.60	97.75	97.68
Improvement	+0.25	+0.21	+0.23	+0.15	+0.32

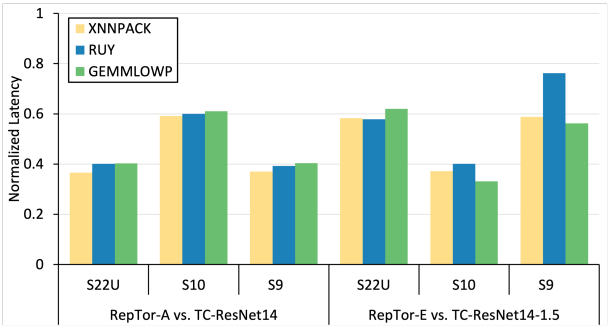


Figure 6: *Normalized latency of RepTor-A and RepTor-E models measured on different devices with various backend libraries. "S22U", "S10"" and "S9" indicates Galaxy S22 Ultra, Galaxy S10 and Galaxy S9, respectively.*

5.2. Hardware and backend library dependency

We further evaluated the latency of the proposed RepTor models compared to TC-ResNets on different mobile devices with different backend libraries. We compared the latency between RepTor-A and TC-ResNet14, and RepTor-E and TC-ResNet14-1.5. We measured the latencies on Galaxy S22 Ultra, Galaxy S10, and Galaxy S9 devices. When using TFLite for inference on Android phones, different backend libraries can be used; XNNPACK, ruy, and Gemmlowp. The latency of a RepTor model was normalized to that of the counterpart TC-ResNet as shown in Figure 6. Although there exist slight differences depending on which device and backend library are used, RepTor models showed almost a half shorter latency than TC-ResNet models which show similar accuracy in all environments.

6. Conclusion

In this paper, we introduce RepTor-family models where the structural re-parameterization technique was applied to temporal convolutions for efficient KWS models. With the re-parameterization technique, RepTor-k models achieved better accuracy-latency trade-off compared to previous KWS models. We further proposed a differentiable kernel search method to find the optimal kernel size of the re-parameterized temporal convolutions. RepTor A∼E models achieved state-of-the-art results on GSC dataset v2 when measured on Android mobile devices compared to prior art. The proposed RepTor models showed that achieving state-of-the-art results on metrics such as model latency is possible without sacrificing other metrics with better network design and search techniques.

7. Acknowledgments

This work was supported by National IT Industry Promotion Agency (NIPA) grant funded by the Korea government(MSIT) (A1504-22-1010).

8. References

[1] S. Sigtia, R. Haynes, H. Richards, E. Marchi, and J. Bridle, "Efficient voice trigger detection for low resource hardware," *Proc. Interspeech 2018*, pp. 2092–2096, 2018.

[2] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 4087–4091.

[3] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," *Proc. Interspeech 2015*, pp. 1478–1482, 2015.

[4] R. Tang and J. 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.

[5] S. Choi, S. Seo, B. Shin, H. Byun, M. Kersner, B. Kim, D. Kim, and S. Ha, "Temporal convolution for real-time keyword spotting on mobile devices," *Proc. Interspeech 2019*, pp. 3372–3376, 2019.

[6] X. Li, X. Wei, and X. Qin, "Small-footprint keyword spotting with multi-scale temporal convolution," *Proc. Interspeech 2020*, pp. 1987–1991, 2020.

[7] S. Majumdar and B. Ginsburg, "Matchboxnet: 1d time-channel separable convolutional neural network architecture for speech commands recognition," *Interspeech 2020*, pp. 3356–3360, 2020.

[8] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, "Repvgg: Making vgg-style convnets great again," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 13 733–13 742.

[9] P. K. A. Vasu, J. Gabriel, J. Zhu, O. Tuzel, and A. Ranjan, "Mobileone: An improved one millisecond mobile backbone," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 7907–7917.

[10] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie *et al.*, "Yolov6: A single-stage object detection framework for industrial applications," *arXiv preprint arXiv:2209.02976*, 2022.

[11] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 7464–7475.

[12] X. Ding, X. Zhang, J. Han, and G. Ding, "Diverse branch block: Building a convolution as an inception-like unit," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10 886–10 895.

[13] 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.

[14] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=S1eYHoC5FX

[15] X. Chen and C.-J. Hsieh, "Stabilizing differentiable architecture search via perturbation-based regularization," in *International conference on machine learning*. PMLR, 2020, pp. 1554–1565.

[16] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *ArXiv e-prints*, Apr. 2018. [Online]. Available: https://arxiv.org/abs/1804.03209

[17] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "Specaugment: A simple data augmentation method for automatic speech recognition," *Proc. Interspeech 2019*, pp. 2613–2617, 2019.

[18] [Online]. Available: https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/tools/benchmark

[19] Thop: Pytorch-opcounter. [Online]. Available: https://github.com/Lyken17/pytorch-OpCounter

[20] B. Kim, S. Chang, J. Lee, and D. Sung, "Broadcasted residual learning for efficient keyword spotting," *Proc. Interspeech 2021*, p. 4538–4542, 2021.

[21] O. Rybakov, N. Kononenko, N. Subrahmanya, M. Visontai, and S. Laurenzo, "Streaming keyword spotting on mobile devices," *arXiv preprint arXiv:2005.06720*, 2020.

[22] S. Ö. Arık, M. Kliegl, R. Child, J. Hestness, A. Gibiansky, C. Fougner, R. Prenger, and A. Coates, "Convolutional recurrent neural networks for small-footprint keyword spotting," *Proc. Interspeech 2017*, pp. 1606–1610, 2017.

[23] 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.

[24] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.