SMALL-FOOTPRINT SLIMMABLE NETWORKS FOR KEYWORD SPOTTING

Zuhaib Akhtar[1,2], *Mohammad Omar Khursheed*[1], *Dongsu Du*[1], *Yuzong Liu*[1]

Alexa Perceptual Technologies, Amazon[1] New York University, NY[2]

ABSTRACT

In this work, we present Slimmable Neural Networks applied to the problem of small-footprint keyword spotting. We show that slimmable neural networks allow us to create super-nets from Convolutional Neural Networks and Transformers, from which sub-networks of different sizes can be extracted. We demonstrate the usefulness of these models on in-house voice assistant data and Google Speech Commands, and focus our efforts on models for the on-device use case, limiting ourselves to less than 250k parameters. We show that slimmable models can match (and in some cases, outperform) models trained from scratch. Slimmable neural networks are therefore a class of models particularly useful when the same functionality is to be replicated at different memory and compute budgets, with different accuracy requirements.

Index Terms— *Slimmable Neural Networks, Transformers, Keyword Spotting, Convolutional Neural Networks*

1. INTRODUCTION

Keyword spotting, or wakeword detection [1], is the act of identifying the presence of a keyword within a stream of audio. This can be thought of as a binary classification problem in the simplest case, detecting if there's a keyword or not. Wakeword detection has become the first point of interaction that a user has with voice assistant systems that are now ubiquitous in everyday life, such as Amazon's Alexa and Apples' Siri. With the prevalence of voice assistants in different types of devices such as earbuds, mobile devices, and smart speakers, there is a constant need to develop on-device keyword spotting models with tradeoffs between model accuracy and on-device resource constraints such as model size and CPU computation. As a result, researchers need to develop deep learning models with different resource constraints, which is expensive in compute and research time.

In recent work, there have been promising approaches towards developing deep learning models with different accuracy-resource tradeoffs for edge devices. The key idea is to train a single network that allows one to derive sub-networks with different tradeoffs between accuracy and resource constraint. One example is the class of networks called slimmable neural networks [2], which train a neural network executable at different widths via weight-sharing and switchable Batch Normalization. Dynamic neural networks are another paradigm in which the network dynamically adapts its computation graph and parameters to different inputs and permits tradeoff between accuracy and inference efficiency [3]. Another notable work Once-for-All (OFA) network was proposed in [4], which allows one to train one super-network once and derive multiple sub-networks with different resource contraint requirements. OFA also mitigates the large computational cost in conventional neural architecture search (NAS) by decoupling the network training and search.

In this paper, we propose the use of slimmable neural networks for small-footprint on-device keyword spotting. Unlike the original slimmable neural networks which were applied to moderate-size (3-25M parameters) networks, our focus is to explore the feasibility of training one small-footprint ($<$250k parameters) neural network to derive sub-networks that achieve tradeoffs between accuracy and device resource constraints. Our contributions are summarized as follows: 1) We propose lightweight CNN-based keyword spotting models using slimmable architectures; 2) We extend the original slimmable networks for keyword spotting with detailed design on slimmable self-attention module; 3) We demonstrate the effectiveness of the proposed method on both a large-scale in-house voice assistant dataset and the public Google Speech Commands dataset [5]. To our knowledge, this is the first work to apply the slimmable networks to multiple architectures including CNN and Transformers on a large-scale keyword spotting dataset under tight on-device resource constraints (10k-250k parameters).

2. RELATED WORK

Keyword spotting models [6, 7, 8, 9, 10] are deployed on-device and typically use a small-footprint deep learning model with minimal latency. With the growing need to deploy deep learning applications to edge devices, it takes significant manual efforts from researchers to develop models to accommodate different on-device budget constraints. There is a plethora of work on developing one single super-network with multiple sub-networks that can be executed on their own and meet the on-device computation budget, including slimmable networks [2, 11, 12], dynamic neural network [13, 14, 3], once-for-all network training [4]. More

Zuhaib Akhtar performed this work during an internship at Amazon recently in speech processing applications, there has been further work exploring the feasibility of this concept. In [15], a dynamic sparsity neural network (DSNN) was proposed for speech recognition. The idea of DSNN is to train one single over-parameterized super-network and apply pruning masks defined from a fixed set of sparsity ratio values. In [16], an Omni-sparsity DNN was proposed for on-device speech recognition. Similar to [15], layer-wise pruning masks were applied to sample sub-networks with different sparsity ratios during super-network training. After the super-network has been trained, an evolutionary searching algorithm was adopted to find the optimal sub-network for any sparsity requirement. In speech representation pretraining [17], an OFA [4] was applied to find the optimal sub-network in a computationally-intensive HuBERT model. A closely related idea is that of Neural Architecture Search [18] [19] which has been used for the problem of speech recognition.

3. PROPOSED APPROACH

We show that Convolutional Neural Networks and Transformer models are slimmable via the general procedure described in **Algorithm 1**. Slimmable training works by defining a width list over which the network will run. During training, the larger network is set to a particular width which is used as a normal neural network to get predictions over a batch, following which losses and gradients are computed. The gradients are saved and weights of the network are not changed at this stage. Following this, the same batch of data is passed again to a model with a smaller width, with the parameters shared from the first larger model. This is done till all the gradients are calculated for every sub-network. Finally, all the gradients are summed up and the weights are updated for the full network.

Algorithm 1 Slimmable Network Training

```
 1: $width\_list = [1.0, 0.75, 0.50, 0.25]$
 2: $model = get\_model(width\_list)$
 3: $grad\_list = []$
 4: for $batch \in \{dataset\}$ do
 5:     for $width \in \{width\_list\}$ do
 6:         $slim\_network(width)$
 7:         $predictions = model(batch)$
 8:         $loss = loss\_function(predictions, labels)$
 9:         $gradient = get\_gradients(loss, model.weigths)$
10:         $grad\_list.append(gradient)$
11:     end for
12:     $overall\_gradients = sum\_gradient(grad\_list)$
13:     $apply\_gradients(model, overall\_gradients)$
14: end for
```

3.1. Slimming CNNs

Slimming in a convolution neural network works by sharing weights between different widths. As the weights are kernels in a convolution layer, when the network switches to a different width, a corresponding number of kernels are dropped to reduce the weight of the network according to the width the network has been set. As the intermediate tensor output is changing between different layers during slimming, the size of individual kernels are also reduced for the convolutional layer. Both of these techniques to reduce kernels makes slimming of the convolution possible. We also used switchable BatchNorm which assigns a separate BatchNorm layer [2] for each width. **Fig. 1** shows slimming of convolution neural network by our approach.

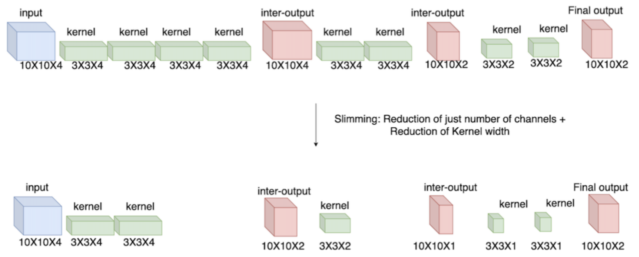

Fig. 1: *The figure shows of an example how end-to-end slimming is done for a convolutional neural network. For the first layer, kernels are dropped. For intermediate layers, kernels are dropped as well as reduced in width. For the last layer, kernels are reduced in width.*

3.2. Slimming Transformers

For slimming of transformers [20], all the dense layers are slimmed within every transformer block. Each intermediate input gets reduced to a dimension determined by the width of the sub-network under consideration via a slimmable dense layer. During the slimming of dense layers in intermediate sub-blocks, a fraction of weights are switched off according to the width, and this is followed by the usual operations while taking the reduced dimension into account. A dense layer at the beginning of each block projects the input to a lower dimension which makes the output compatible with the residual connection. The final dense layer of the transformer ensures that it always produces output of same dimension (number of classes) irrespective of input dimension (which are reduced during slimming). The slimmable transformer is designed in such a way that LayerNorm acts as a switch. For each width, a separate LayerNorm layer is used such that a particular layer that is assigned to that width is selected at run-time. This is implemented in a fashion similar to the switchable BatchNorm layer in CNNs.
Equation 1, 2 and 3 shows the working of slimmable attention. Query (Q), Key (K) and Value (V) tensors are of dif-

ferent sizes according to the width of the transformer sub-network. Attention operation such as dot-product takes place on a lower dimension for the sub-networks according to the width. This makes lower-width models compute and memory efficient. These lower-dimension tensors are generated by slimming down Q, K and V weight matrices. **Fig. 2** shows the slimming of a transformer by our approach.

$$Slim\_Attn(Q, K, V) = softmax(\frac{Q[:i]K[:i]^T}{\sqrt{d_k[:i]}})V[:i] \tag{1}$$

$$i \in [d_k * width[0], d_k * width[1], \ldots, d_k * width[n]] \tag{2}$$

$$width \in [1, (n-1)/n, (n-2)/n, \ldots, (1/n)], n > 1(integer) \tag{3}$$

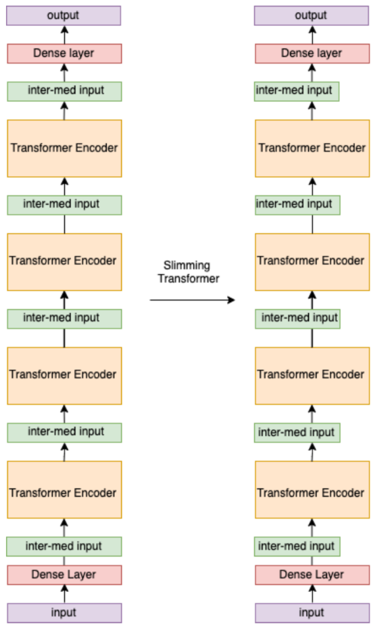

Fig. 2: *The figure shows of an example how end-to-end slimming is done for transformers.*

CNN Layers	Kernel size	Channel	Striding	Pooling
1	(5, 4)	32	(1, 2)	(2, 1)
2	(3, 4)	32	(1, 3)	(2, 1)
3	(4, 4)	40	(1, 2)	(2, 1)
4	(7, 4)	128	(1, 1)	(1, 1)
5	(1, 1)	160	(1, 1)	(1, 1)
Transformer model	**dim**	**mlp-dim**	**heads**	**layer**
Transformer (in-house data)	64	128	1	3
Transformer-GoogleSpeech	64	64	1	2

Table 1: *CNN and Transformer Architectures*

Model Name	Relative False Accepts	Parameters	Multiplies
CNN-Scratch-Width-1	Baseline (1)	199k	3.5M
CNN-Scratch-Width-0.75	1.11	122k	2.1M
CNN-Scratch-Width-0.5	1.28	50k	1.1M
CNN-Scratch-Width-0.25	1.78	13k	0.35M
CNN-Slim-Width-1	1.05	199k	3.5M
CNN-Slim-Width-0.75	1.20	122k	2.1M
CNN-Slim-Width-0.5	1.35	50k	1.1M
CNN-Slim-Width-0.25	1.72	13k	0.35M

Table 2: *CNN Results on in-house Dataset*

Model Name	Accuracy	Parameters	Multiplies
CNN-Scratch-Width-1	0.8915	243k	4.8M
CNN-Scratch-Width-0.75	0.8878	138k	2.9M
CNN-Scratch-Width-0.5	0.8636	62k	1.4M
CNN-Scratch-Width-0.25	0.8455	16k	0.48M
CNN-Slim-Width-1	0.9047	243k	4.8M
CNN-Slim-Width-0.75	0.8967	138k	2.9M
CNN-Slim-Width-0.5	0.8726	62k	1.4M
CNN-Slim-Width-0.25	0.8446	16k	0.48M
Res26	0.95	438k	N/A

Table 3: *CNN Results on Google Speech Commands Dataset*

4. EXPERIMENTS AND RESULTS

4.1. Data and Baseline models

We trained all our models on 2 different tasks. The first task was to detect a unique keyword in our in-house voice assistant dataset, and the second is the Google Speech Commands 35-class classification task using TensorFlow [21].
We use 76 input frames for our CNN models and 182 frames for transformer models, computing Log Mel Filter Bank Energies (LFBEs) every 10 ms over a window of 25 ms for our in-house dataset. For Google Speech Commands data, we use 98 frames of audio for both model types. We align the wake-word using a DNN-HMM based spotter model that places the keyword in the center of the context window being fed into the model for training. We train our models on over 20000 hours of de-identified in-house data and the full training set from Google Speech Commands, and evaluate on 100 hours for in-house data and the Google Speech Commands test set for the latter. Our baseline CNN architecture is a 5-layer CNN with 199k parameters (**Table 1**) for both in-house and Google Speech Commands datasets. Our baseline transformer architecture is a smaller version of the KWT [22] architecture with 120k parameters for in-house data and 67k parameters for Google Speech Commands (**Table 1**). For all experiments on the in-house dataset, we report relative improvement over a baseline model in terms of False Accepts at a fixed miss rate, and we provide absolute accuracy for google speech commands. We also report parameter counts and multiplication operations, since these provide a sense of the memory and compute requirements of our keyword spotting models.

4.2. Slimmable CNNs and Transformers

When slimming a 199k parameter CNN to 4 different widths (1, 0.75, 0.5 and 0.25), we find an accuracy and compute trade-off. While the slimmed models are slightly worse compared to single models of the same size trained from scratch on our in-house dataset as shown in **Table 2**, the training time is reduced by a factor of 4 per step, which is a significant savings in spite of the extra epochs (20 vs 30) it takes to improve slimmable models when compared to their counterparts trained from scratch. However, for the smallest width (0.25), the slimmable model outperforms the equivalent model trained from scratch.

We also see that training time for multiple widths does not scale linearly. Instead, as **Table 4** shows, there is only a 3.72x increase in training time when the number of widths goes from 1 to 40. This points to the possibility of training a single super-network, from which sub-networks of a variety of sizes can be extracted, in over 10 times less time than individually training each width from scratch.

Total Widths	Training Time Per Step (sec)
1	1.01
2	1.53
3	1.60
4	1.63
5	1.69
10	2.01
20	2.29
40	3.72

Table 4: *Slimmable NNs Training Time*

When slimming down a 120k parameter transformer model to the same 4 widths, we see a similar pattern emerge, with the slimmable models being slightly worse than those trained from scratch on the in-house dataset as shown in **Table 5**.

When we repeat this experiment on Google Speech Commands data (35 classes), we see that the slimmable models are actually slightly better in terms of accuracy than models trained from scratch, for all CNNs except the smallest, as shown in **Table 3**. This indicates that the slimmable NN has a gain of performance because of the inherent distillation present in the model due to the shared weights [2] across different widths. For the slimmable transformer, this pattern also holds true (**Table 6**)

Model Name	Relative False Accepts	Parameters	Multiplies
Transformer-Scratch-Width-1	Baseline (1)	120k	18M
Transformer-Scratch-Width-0.75	1.08	76k	10.1M
Transformer-Scratch-Width-0.5	1.15	43k	4.5M
Transformer-Scratch-Width-0.25	1.36	24k	1.1M
Transformer-Slim-Width-1	1.19	120k	18M
Transformer-Slim-Width-0.75	1.19	76k	10.1M
Transformer-Slim-Width-0.5	1.27	43k	4.5M
Transformer-Slim-Width-0.25	1.34	24k	1.1M

Table 5: *Transformer Results on in-house dataset*

Model Name	% Accuracy	Parameters	Multiplies
Transformer-Scratch-Width-1	0.8623	67k	7.2M
Transformer-Scratch-Width-0.75	0.8463	44k	4.5M
Transformer-Scratch-Width-0.5	0.8467	26k	2.4M
Transformer-Scratch-Width-0.25	0.7295	15k	0.9M
Transformer-Slim-Width-1	0.8787	67k	7.2M
Transformer-Slim-Width-0.75	0.8766	44k	4.5M
Transformer-Slim-Width-0.5	0.8588	26k	2.4M
Transformer-Slim-Width-0.25	0.7840	15k	0.9M

Table 6: *Transformer Results on Google Speech Commands Dataset*

5. CONCLUSION AND FUTURE WORK

In this work, we show that slimmable networks are a viable option for small-footprint keyword spotting. We show that this method works across different types of architectures. Our work shows that slimmable NNs generalize to different datasets and situations. We explore the possibility of training multiple models at different budgets at once, and show, through analysis of training times, that slimmable NNs are several times more efficient when compared to training individual models from scratch.

The use of slimmable models can be used to test different sizes of a single architecture, and find the optimal network for a certain dataset. When one trains a supernetwork of a number of widths and evaluates on each of the sub-networks, it can be used to determine the minimum capacity of a model required to do well on that dataset. Considering that inference is usually a lot cheaper and faster to perform than training, this could enable models to be released on a variety of platforms with different memory, compute and performance requirements. Many companies that provide keyword spotting solutions as part of an AI assistant. Using slimmable models would allow for training and deploying models at scale. Since these AI assistants run on a variety of platforms, creating a single model that can run at different budgets (or have different budget sub-models extracted from itself) is a driver of efficiently deploying models across products, which vary from smartphones to earbuds to TVs.

We hope to extend this work by using more efficient automated slimming techniques such as AutoSlim [12], and extending slimmable models to include novel architectures, such as RNNs. There also exists the possibility to slim the depth of a network, rather than just the width. For the purposes of edge computing, profiling slimmable models on various chipsets used for edge computing could give us an improved sense of the memory and compute of these models.

6. REFERENCES

[1] Igor Szöke, Petr Schwarz, Pavel Matejka, Lukás Burget, Martin Karafiát, Michal Fapso, and Jan Cernockỳ, "Comparison of keyword spotting approaches for informal continuous speech.," in *Interspeech*, 2005, pp. 633–636.

[2] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang, "Slimmable neural networks," *ICLR*, 2019.

[3] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang, "Dynamic neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[4] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han, "Once for all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, 2020.

[5] Pete Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018.

[6] 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.

[7] Tara N. Sainath and Carolina Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Proc. Interspeech 2015*, 2015, pp. 1478–1482.

[8] Sankaran Panchapagesan, Ming Sun, Aparna Khare, Spyros Matsoukas, Arindam Mandal, Björn Hoffmeister, and Shiv Vitaladevuni, "Multi-task learning and weighted cross-entropy for DNN-based keyword spotting," *Interspeech 2016*, pp. 760–764, 2016.

[9] Ming Sun, David Snyder, Yixin Gao, Varun K. Nagaraja, Mike Rodehorst, Sankaran Panchapagesan, Nikko Strom, Spyridon Matsoukas, and Shiv Naga Prasad Vitaladevuni, "Compressed time delay neural network for small-footprint keyword spotting," in *Interspeech*, 2017.

[10] Mohammad Omar Khursheed, Christin Jose, Rajath Kumar, Gengshen Fu, Brian Kulis, and Santosh Kumar Cheekatmalla, "Tiny-CRNN: Streaming wakeword detection in a low footprint setting," in *2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2021, pp. 541–547.

[11] Jiahui Yu and Thomas S Huang, "Universally slimmable networks and improved training techniques," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1803–1811.

[12] Jiahui Yu and Thomas Huang, "AutoSlim: Towards one-shot architecture search for channel numbers," *arXiv preprint arXiv:1903.11728*, 2019.

[13] Lanlan Liu and Jia Deng, "Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018, vol. 32.

[14] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens Van Der Maaten, and Kilian Q Weinberger, "Multi-scale dense networks for resource efficient image classification," *arXiv preprint arXiv:1703.09844*, 2017.

[15] Zhaofeng Wu, Ding Zhao, Qiao Liang, Jiahui Yu, Anmol Gulati, and Ruoming Pang, "Dynamic sparsity neural networks for automatic speech recognition," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 6014–6018.

[16] Haichuan Yang, Yuan Shangguan, Dilin Wang, Meng Li, Pierce Chuang, Xiaohui Zhang, Ganesh Venkatesh, Ozlem Kalinli, and Vikas Chandra, "Omni-Sparsity DNN: Fast sparsity optimization for on-device streaming E2E ASR via supernet," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 8197–8201.

[17] Rui Wang, Qibing Bai, Junyi Ao, Long Zhou, Zhixiang Xiong, Zhihua Wei, Yu Zhang, Tom Ko, and Haizhou Li, "LightHuBERT: Lightweight and configurable speech representation learning with once-for-all hidden-unit BERT," in *Interspeech*, 2022.

[18] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter, "Neural architecture search: A survey," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.

[19] Abhinav Mehrotra, Alberto Gil CP Ramos, Sourav Bhattacharya, Lukasz Dudziak, Ravichander Vipperla, Thomas Chau, Mohamed S Abdelfattah, Samin Ishtiaq, and Nicholas Donald Lane, "NAS-Bench-ASR: Reproducible neural architecture search for speech recognition," in *International Conference on Learning Representations*, 2020.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. 2017, vol. 30, Curran Associates, Inc.

[21] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, USA, 2016, OSDI'16, p. 265–283, USENIX Association.

[22] Axel Berg, Mark O'Connor, and Miguel Tairum Cruz, "Keyword transformer: A self-attention model for keyword spotting," in *Interspeech 2021*. ISCA, 2021, pp. 4249–4253.