Stochastic Adaptive Neural Architecture Search for Keyword Spotting

Tom Véniat [†], Olivier Schwander [†], and Ludovic Denoyer [†, *]

[†]Sorbonne Université, LIP6, F-75005, Paris, France
[*]Facebook AI Research
{tom.veniat, olivier.schwander}@lip6.fr
denoyer@fb.com

Abstract

The problem of keyword spotting *i.e.* identifying keywords in a real-time audio stream is mainly solved by applying a neural network over successive sliding windows. Due to the difficulty of the task, baseline models are usually large, resulting in a high computational cost and energy consumption level. We propose a new method called SANAS (Stochastic Adaptive Neural Architecture Search) which is able to adapt the architecture of the neural network on-the-fly at inference time such that small architectures will be used when the stream is easy to process (silence, low noise, ...) and bigger networks will be used when the task becomes more difficult. We show that this adaptive model can be learned end-to-end by optimizing a trade-off between the prediction performance and the average computational cost per unit of time. Experiments on the Speech Commands dataset [16] show that this approach leads to a high recognition level while being much faster (and/or energy saving) than classical approaches where the network architecture is static.

1 Introduction and Related Work

Neural Networks (NN) are known to obtain very high recognition rates on a large variety of tasks, and especially over signal-based problems like speech recognition [1], image classification [7, 11], etc. However these models are usually composed of millions of parameters involved in millions of operations and have high computational and energy costs at prediction time. There is thus a need to increase their processing speed and reduce their energy footprint.

From the NN point of view, this problem is often viewed as a problem of network architecture discovery and solved with Neural Architecture Search (NAS) methods in which the search is guided by a trade-off between prediction quality and prediction cost [6, 8, 15]. Recent approaches involve for instance Genetic Algorithms [10, 11] or Reinforcement Learning [20, 21]. While these models often rely on expensive training procedures where multiple architectures are trained, some recent works have proposed to simultaneously discover the architecture of the network while learning its parameters [8, 15] resulting in models that are fast both at training and at inference time. But in all these works, the discovered architecture is static *i.e.* the same NN being re-used for all the predictions.

When dealing with streams of information, reducing the computational and energy costs is of crucial importance. For instance, let us consider the keyword spotting[1] problem which is the focus of this paper. It consists in detecting keywords in an audio stream and is particularly relevant for virtual assistants which must continuously listen to their environments to spot user interaction requests. This

[1]See Section 3 for a formal description.

Preprint. Work in progress.

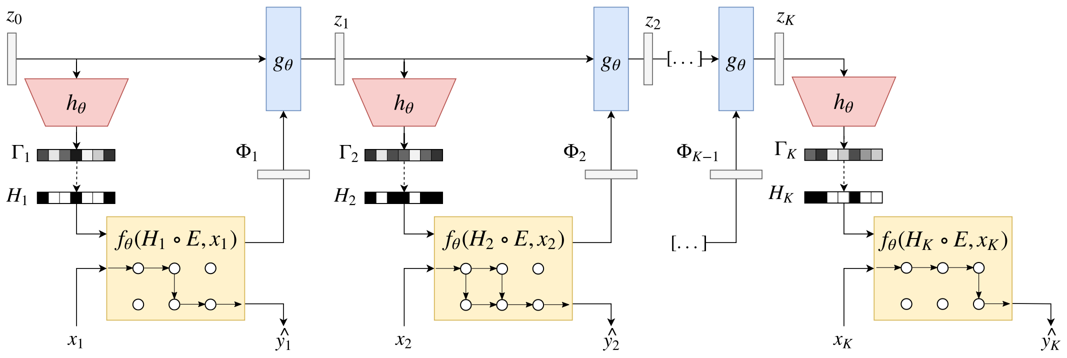

Figure 1: **SANAS Architecture**. At timestep t, the distribution Γ_t is generated from the previous hidden state, $\Gamma_t = h(z_t, \theta)$. A discrete architecture H_t is then sampled from Γ_t and evaluated over the input x_t. This evaluation gives both a feature vector $\Phi(x_t, \theta, E \circ H_t)$ to compute the next hidden state, and the prediction of the model $\hat{y}_t$ using $f(z_t, x_t, \theta, E \circ H_t)$. Dashed edges represent sampling operations. At inference, the architecture which has the highest probability is chosen at each timestep.

requires detecting when a word is pronounced, which word has been pronounced and able to run quickly on resource-limited devices. Some recent works [2, 12, 13] proposed to use convolutional neural networks (CNN) in this streaming context, applying a particular model to successive sliding windows [12, 13] or combining CNNs with recurrent neural networks (RNN) to keep track of the context [2]. In such cases, the resulting system spends the same amount of time to process each audio frame, irrespective of the content of the frame or its context.

Our conjecture is that, when dealing with streams of information, a model able to adapt its architecture to the difficulty of the prediction problem at each timestep – i.e. a small architecture being used when the prediction is easy, and a larger architecture being used when the prediction is more difficult – would be more efficient than a static model, particularly in terms of computation or energy consumption. To achieve this goal, we propose the SANAS algorithm (Section 2.3): it is, as far as we know, the first architecture search method producing a system which dynamically adapts the architecture of a neural network during prediction at each timestep and which is learned end-to-end by minimizing a trade-off between computation cost and prediction loss. After learning, our method can process audio streams at a higher speed than classical static methods while keeping a high recognition rate, spending more prediction time on complex signal windows and less time on easier ones (see Section 3).

2 Adaptive Neural Architecture Search

2.1 Problem Definition

We consider the generic problem of stream labeling where, at each timestep, the system receives a datapoint denoted x_t and produces an output label y_t. In the case of audio streams, x_t is usually a time-frequency feature map, and y_t is the presence or absence of a given keyword. In classical approaches, the output label y_t is predicted using a neural network whose architecture[2] is denoted $\mathcal{A}$ and whose parameters are θ. We consider in this paper the recurrent modeling scheme where the context $x_1, y_1, ..., x_{t-1}, y_{t-1}$ is encoded using a latent representation z_t, such that the prediction at time t is made computing $f(z_t, x_t, \theta, \mathcal{A})$, z_t being updated at each timestep such that $z_{t+1} = g(z_t, x_t, \theta, \mathcal{A})$ - note that g and f can share some common computations.

For a particular architecture $\mathcal{A}$, the parameters are learned over a training set of labeled sequences $\{(x^i, y^i)\}_{i\in[1,N]}$, N being the size of the training set, by solving:

$$\theta^* = \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} [\sum_{t=1}^{\#x^i} \Delta(f(z_t, x_t, \theta, \mathcal{A}), y_t)]$$

where $\#x^i$ is the length of sequence x^i, and Δ a differentiable loss function. At inference, given a new stream x, each label $\hat{y}_t$ is predicted by computing $f(x_1, \hat{y}_1, ..., \hat{y_{t-1}}, x_t, \theta^*, \mathcal{A})$, where $\hat{y}_1 \dots \hat{y_{t-1}}$

[2] a precise definition of the notion of architecture is given further.

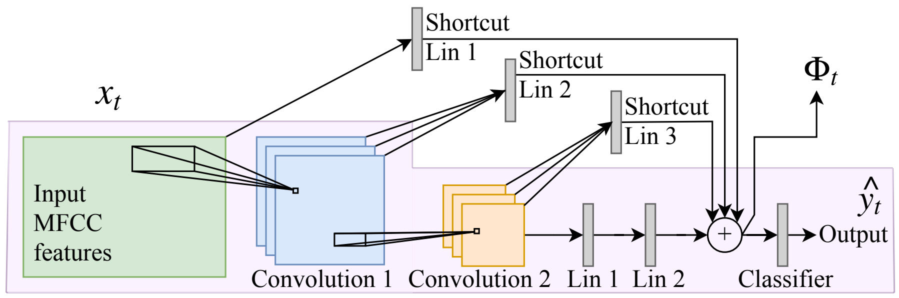

Figure 2: SANAS architecture based on *cnn-trad-fpool3* [12]. Edges between layers are sampled by the model. The highlighted architecture is the base model on which we have added shortcut connections. Conv1 and Conv2 have filter sizes of (20,8) and (10,4). Both have 64 channels and Conv1 has a stride of 3 in the frequency domain. Linear 1,2 and the Classifier have 32, 128 and 12 neurons respectively. Shortcut linears all have 128 neurons to match the dimension of the classifier.

are the predictions of the model at previous timesteps. In that case, the computation cost of each prediction step solely depends on the architecture and is denoted $C(\mathcal{A})$.

2.2 Stochastic Adaptive Architecture Search: Principles

We propose now a different setting where the architecture of the model can change at each timestep depending on the context of the prediction z_t. At time t, in addition to producing a distribution over possible labels, our model also maintains a distribution over possible architectures denoted $P(\mathcal{A}_t|z_t,\theta)$. The prediction y_t being now made following[3] $f(z_t,x_t,\theta,\mathcal{A}_t)$ and the context update being $z_{t+1}=g(z_t,x_t,\theta,\mathcal{A}_t)$. In that case, the cost of a prediction at time t is now $C(\mathcal{A}_t)$, which also includes the computation of the architecture distribution $P(\mathcal{A}_t|z_t,\theta)$. It is important to note that, since the architecture $\mathcal{A}_t$ is chosen by the model, it has the possibility to learn to control this cost itself. A budgeted learning problem can thus be defined as minimizing a trade-off between prediction loss and average cost. Considering a labeled sequence (x,y), this trade-off is defined as :

$$\mathcal{L}(x,y,\theta)=\mathbb{E}_{\{\mathcal{A}_t\}}\big[\sum_{t=1}^{\#x}[\Delta(f(z_t,x_t,\theta,\mathcal{A}_t),y_t)+\lambda C(\mathcal{A}_t)]\big]$$

where $\mathcal{A}_1,...,\mathcal{A}_{\#x}$ are sampled following $P(\mathcal{A}_t|z_t,\theta)$ and λ controls the trade-off between cost and prediction efficiency. Considering that $P(\mathcal{A}_t|z_t,\theta)$ is differentiable, and following the derivation schema proposed in [5] or [15], this cost can be minimized using the Monte-Carlo estimation of the gradient. Given one sample of architectures $\mathcal{A}_1,...,\mathcal{A}_{\#x}$, the gradient can be approximated by:

$$\nabla_\theta\mathcal{L}(x,y,\theta)\approx\big(\sum_{t=1}^{\#x}\nabla_\theta\log P(\mathcal{A}_t|z_t,\theta)\big)\mathcal{L}(x,y,\mathcal{A}_1,...,\mathcal{A}_{\#x},\theta)$$
$$+\sum_{t=1}^{\#x}\nabla_\theta\Delta(f(z_t,x_t,\theta,\mathcal{A}_t),y_t)$$

where

$$\mathcal{L}(x,y,\mathcal{A}_1,...,\mathcal{A}_{\#x},\theta)=\sum_{t=1}^{\#x}[\Delta(f(z_t,x_t,\theta,\mathcal{A}_t),y_t)+\lambda C(\mathcal{A}_t)]$$

In practice, a variance correcting value is used in this gradient formulation to accelerate the learning as explained in [17, 18].

2.3 The SANAS Model

We now instantiate the previous generic principles in a concrete model where the architecture search is cast into a sub-graph discovery in a large graph representing the search space called *Super-Network* as in [15].

[3] f is usually a distribution over possible labels.

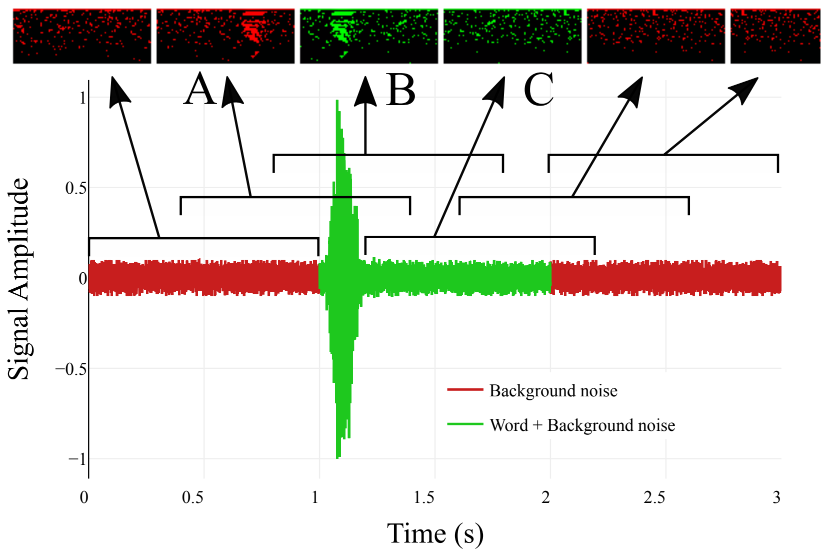

Figure 3: Example of labeling using the method presented in section 3. To build the dataset, a ground noise (red) is mixed with randomly located words (green). The signal is then split in 1s frames every 200ms. When a frame contains at least 50% of a word signal, it is labeled with the corresponding word (frame B and C – frame A is labeled as *bg-noise*). Note that this labeling could be imperfect (see frame A and C).

NAS with Super-Networks (static case): A Super-Network is a directed acyclic graph of layers $L = \{l_1, ...l_n\}$, of edges $E \in \{0,1\}^{n \times n}$ and where each existing edge connecting layers i and j ($e_{i,j} = 1$) is associated with a (small) neural network $f_{i,j}$. The layer l_1 is the input layer while l_n is the output layer. The inference of the output is made by propagating the input x over the edges, and by summing, at each layer level, the values coming from incoming edges. Given a Super-Network, the architecture search can be made by defining a distribution matrix $\Gamma \in [0,1]^{n \times n}$ that can be used to sample edges in the network using a Bernoulli distribution. Indeed, let us consider a binary matrix H sampled following Γ, the matrix $E \circ H$ defines a sub-graph of E and corresponds to a particular neural-network architecture which size is smaller than E ($\circ$ being the Hadamard product). Learning Γ thus results in doing architecture search in the space of all the possible neural networks contained in Super-Network. At inference, the architecture with the highest probability is chosen.

SANAS with Super-Networks: Based on the previously described principle, our method proposes to use a RNN to generate the architecture distribution at each timestep – see Figure 1. Concretely, at time t, a distribution over possible sub-graphs $\Gamma_t = h(z_t, \theta)$ is computed from the context z_t. This distribution is then used to sample a particular sub-graph represented by $H_t \sim \mathcal{B}(\Gamma_t)$, $\mathcal{B}$ being a Bernoulli distribution. This particular sub-grap $E \circ H_t = \mathcal{A}_t$ corresponds to the architecture used at time t. Then the prediction $\hat{y}_t$ and the next state z_{t+1} are computed using the functions $f(z_t, x_t, \theta, E \circ H_t)$ and $g(z_t, \Phi(x_t, \theta, E \circ H_t), \theta)$ respectively, where $g(z_t, ., \theta)$ is a classical RNN operator like a Gated Recurrent Unit[3] cell for instance and $\Phi(x_t, \theta, E \circ H_t)$ is a feature vector used to update the latent state and computed using the sampled architecture $\mathcal{A}_t$. The learning of the parameters of the proposed model relies on a gradient-descent method based on the approximation of the gradient provided previously, which simultaneously updates the parameters θ and the conditional distribution over possible architectures.

3 Experiments

We train and evaluate our model using the Speech Commands dataset [16]. It is composed of 65000 short audio clips of 30 common words. As done in [13, 14, 19], we treat this problem as a classification task with 12 categories: 'yes', 'no', 'up', 'down', 'left', 'right', 'on', 'off', 'stop', 'go', '*bg-noise*' for background noise and '*unknown*' for the remaining words.

Match	Correct	Wrong	FA	FLOPs per frame
cnn-trad-fpool3				
81.7%	72.8%	8.9%	0.0%	124.6M
***cnn-trad-fpool3* + shortcut connections**				
82.9%	77.9%	5.0%	0.3%	137.3M
SANAS				
61.2%	53.8%	7.3%	0.7%	519.2K
62.0%	57.3%	4.8%	0.1%	22.9M
86.5%	80.7%	5.8%	0.3%	37.7M
86.3%	80.6%	5.7%	0.2%	48.8M
81.7%	76.4%	5.3%	0.1%	94.0M
81.4%	76.3%	5.2%	0.2%	105.4M

Table 1: Evaluation of models on 1h of audio from [16] containing words roughly every 3 seconds with different background noises. We use the label post processing and the streaming metrics proposed in [16] to avoid repeated and noisy detections. Matched % corresponds to the portion of words detected, either correctly (Correct %) or incorrectly (Wrong %). FA is *False Alarm*.

Instead of directly classifying 1 second samples, we use this dataset to generate between 1 and 3 second long audio files by combining a background noise coming from the dataset with a randomly located word (see Figure 3), the signal-to-noise ratio being randomly sampled with a minimum of 5dB. We thus obtain a dataset of about 30,000 small files[4], and then split this dataset in train, validation and test sets using a 80:10:10 ratio. The sequence of frames is created by taking overlapping windows of 1 second every 200ms. The input features for each window are computed by extracting 40 mel-frequency spectral coefficients (MFCC) on 30ms frames every 10ms and stacking them to create 2D time/frequency maps. For the evaluation, we use both the prediction accuracy and the number of operations per frame (FLOPs) value. The model selection is made by training multiple models, selecting the best models on the validation set, and computing their performance on the test set. Note that the 'best models' in terms of both accuracy and FLOPs are the models located on the pareto front of the accuracy/cost validation curve as done for instance in [4]. These models are also evaluated using the *matched, correct, wrong* and *false alarm* (FA) metrics as proposed in [16] and computed over the one hour stream provided with the original dataset. Note that these last metrics are computed after using a post-processing method that ensures a labeling consistency as described in the reference paper.

As baseline static model, we use a standard neural network architecture used for Keyword Spotting aka the *cnn-trad-fpool3* architecture proposed in [12] which consists in two convolutional layers followed by 3 linear layers. We then proposed a SANAS extension of this model (see Figure 2) with additional connections that will be adaptively activated (or not) during the audio stream processing. In the SANAS models, the recurrent layer g is a one-layer GRU [3] and the function h mapping from the hidden state x_t to the distribution over architecture Γ_t is a one-layer linear module followed by a sigmoid activation. The models are learned using the ADAM [9] optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, gradient steps between 10^{-3} and 10^{-5} and λ in range $[10^{-(m+1)}, 10^{-(m-1)}]$ with m the order of magnitude of the cost of the full model. Training time is reasonable and corresponds to about 1 day on a single GPU computer.

Results obtained by various models are illustrated in Table 1 for the one-hour test stream, and in Figure 4 on the test evaluation set. It can be seen that, at a given level of accuracy, the SANAS approach is able to greatly reduce the number of FLOPs, resulting in a model which is much more power efficient. For example, with an average cost of 37.7M FLOPs per frame, our model is able to match 86.5% of the words, (80.7% correctly and 5.8% wrongly) while the baseline models match 81.7% and 82.9% of the words with 72.8% and 77.9% correct predictions while having a higher budget of 124.6M and 137.3 FLOPs per frame respectively. Moreover, it is interesting to see that our model also outperforms both baselines in term of accuracy, or regarding the metrics in Table 1. This is due to the fact that, knowing that we have added shortcut connections in the base architecture, our model has a better expressive power. Note that in our case, over-fitting is avoided by the cost

[4]tools for building this dataset are available at `http://github.com/TomVeniat/SANAS` with the open-source implementation.

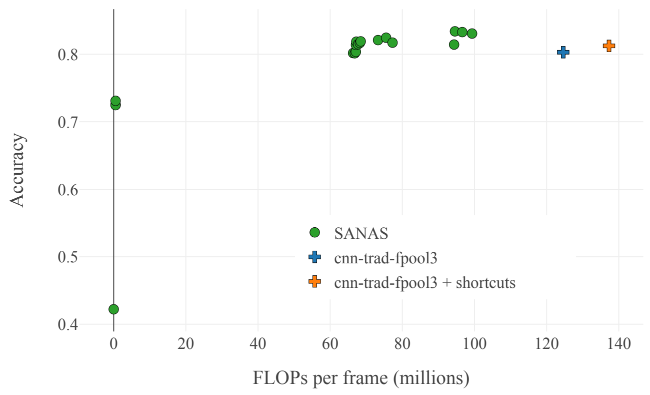

Figure 4: **Cost accuracy curves.** Reported results are computed on the test set using models selected by computing the Pareto front over the validation set. Each point represents a model.

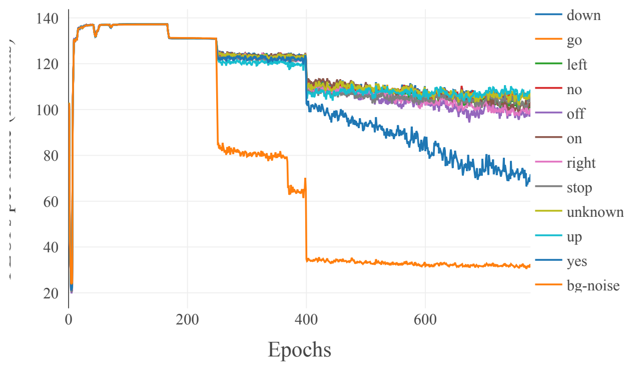

Figure 5: **Training dynamics.** Average cost per output label during training. The network is able to find an architecture that solves the task while sampling notably cheaper architectures when only background noise is present in the frames.

minimization term in the objective function, while it occurs when using the complete architecture with shortcuts (see Figure 4). Figure 5 illustrates the average cost per possible prediction during training. It is not surprising to show that our model automatically 'decides' to spend less time on frames containing background noise and much more time on frames containing words. Moreover, at convergence, the model also divides its budget differently on the different words, for example spending less time on the *yes* words that are easy to detect.

4 Conclusion

We have proposed a new model for keyword spotting where the recurrent network is able to automatically adapt its size during inference depending on the difficulty of the prediction problem at time t. This model is learned end-to-end based on a trade-off between prediction efficiency and computation cost and is able to find solutions that keep a high prediction accuracy while minimizing the average computation cost per timestep. Ongoing research includes using these methods on larger super-networks and investigating other types of budgets like memory footprint or electricity consumption on connected devices.

Acknowledgments

This work has been funded in part by grant ANR-16-CE23-0016 “PAMELA” and grant ANR-16-CE23-0006 “Deep in France”.

References

[1] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Y. Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. *CoRR*, abs/1512.02595, 2015. URL `http://arxiv.org/abs/1512.02595`.

[2] Sercan Ömer Arik, Markus Kliegl, Rewon Child, Joel Hestness, Andrew Gibiansky, Christopher Fougner, Ryan Prenger, and Adam Coates. Convolutional recurrent neural networks for small-footprint keyword spotting. *CoRR*, abs/1703.05390, 2017.

[3] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014. URL `http://arxiv.org/abs/1409.1259`.

[4] Gabriella Contardo, Ludovic Denoyer, and Thierry Artières. Recurrent neural networks for adaptive feature acquisition. In *Neural Information Processing - 23rd International Conference, ICONIP 2016, Kyoto, Japan, October 16-21, 2016, Proceedings, Part III*, 2016.

[5] Ludovic Denoyer and Patrick Gallinari. Deep sequential neural network. *CoRR*, abs/1410.0510, 2014. URL `http://arxiv.org/abs/1410.0510`.

[6] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. *CoRR*, abs/1711.06798, 2017. URL `http://arxiv.org/abs/1711.06798`.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

[8] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. *CoRR*, abs/1707.01213, 2017. URL `http://arxiv.org/abs/1707.01213`.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

[10] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Quoc V. Le, and Alex Kurakin. Large-scale evolution of image classifiers. *CoRR*, abs/1703.01041, 2017. URL `http://arxiv.org/abs/1703.01041`.

[11] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018. URL `http://arxiv.org/abs/1802.01548`.

[12] Tara N. Sainath and Carolina Parada. Convolutional neural networks for small-footprint keyword spotting. In *INTERSPEECH*, pages 1478–1482. ISCA, 2015.

[13] 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 2018, Calgary, AB, Canada, April 15-20, 2018*, 2018.

[14] 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 2018, Calgary, AB, Canada, April 15-20, 2018*, 2018.

[15] Tom Veniat and Ludovic Denoyer. Learning time/memory-efficient deep architectures with budgeted super networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[16] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *CoRR*, abs/1804.03209, 2018. URL `http://arxiv.org/abs/1804.03209`.

[17] Daan Wierstra, Alexander Förster, Jan Peters, and Jürgen Schmidhuber. Solving deep memory pomdps with recurrent policy gradients. In *Artificial Neural Networks - ICANN 2007, 17th International Conference, Porto, Portugal, September 9-13, 2007, Proceedings, Part I*, 2007.

[18] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992. doi: 10.1007/BF00992696. URL `https://doi.org/10.1007/BF00992696`.

[19] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. Hello edge: Keyword spotting on microcontrollers. *CoRR*, abs/1711.07128, 2017. URL `http://arxiv.org/abs/1711.07128`.

[20] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016. URL `http://arxiv.org/abs/1611.01578`.

[21] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017. URL `http://arxiv.org/abs/1707.07012`.