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

Improving Keyword Spotting and Language Identification via Neural Architecture Search at Scale

Hanna Mazzawi[1], *Xavi Gonzalvo*[1], *Aleks Kracun*[2], *Prashant Sridhar*[2], *Niranjan Subrahmanya*[2], *Ignacio Lopez Moreno*[2], *Hyun Jin Park*[2], *Patrick Violette*[2]

[1]Google Research
[2]Google Speech

{mazzawi, xavigonzalvo, yak, psridhar, sniranjan, elnota, hjpark, pdv}@google.com

Abstract

In this paper we present a novel Neural Architecture Search (NAS) framework to improve keyword spotting and spoken language identification models. Even with the huge success of deep neural networks (DNNs) in many different domains, finding the best network architecture is still a laborious task and very computationally expensive at best with existing searching approaches. Our search approach efficiently and robustly finds better model sequences with respect to hand-designed systems. We do this by constructing architectures incrementally, using a custom mutation algorithm and leveraging the power of parameter transfer between layers. We demonstrate that our approach can automatically design DNNs with an order of magnitude fewer parameters that achieves better performance than the current best models. It leads to significant performance improvements: up to 4.09% accuracy increase for language identification (6.1% if we allow an increase in the number of parameters) and 0.3% for phoneme classification in keyword spotting with half the size of the model.

1. Introduction

Voice control is emerging as an attractive method of interaction in all types of smart devices. Its incorporation in many new devices is pushing researchers to provide better models for the well known, long studied speech processing problems that are shared across devices. In this study, we focus on two tasks: language identification [1], and "Ok Google" keyword spotting, a task that enables a hands free speech recognition experience across Google [2, 3, 4].

In the last two decades, machine learning (ML) practitioners have been switching to using DNNs as a template solution for a large variety of problems (from translation [5] to speech recognition [6]).

However, two main problems remain: designing a neural network is a hard task and our understanding of neural network generalization is limited. In addition, with growing evidence that the architecture matters significantly as seen in areas like image classification (from AlexNet [7] to ResNet [8]), to the most recent Transformer [9, 10], a new category of ML algorithms is emerging, the so-called automated ML or AutoML. AutoML systems aim to create high-quality ML models with minimum human intervention [11].

AutoML involves different techniques, from feature manufacturing to model search. Neural architecture search (NAS) is a prominent example of the latter. Recently, several successful approaches have been presented for solving the NAS problem. In [12], the authors use reinforcement learning (RL) to search for new architectures and the result is two building blocks, one for convolution and one for downsampling. Similarly [13] presents an evolutionary approach where mutation is applied to the search space achieving faster convergence. The main problem of these two approaches is the amount of resources required to perform proper exploration of the search space. One possible solution is presented in [14] where transfer learning is applied to the search controller. Additionally, in [15], a differentiable search space is used by defining a continuous relaxation of the architecture representation. For keyword spotting, some preliminary results are shown in [16] where a Stochastic Adaptive NAS is presented in order to adapt the architecture of the neural network on-the-fly at inference time.

In addition to the problem of the above solutions being computational expensive, there is also a problem of bias towards human expertise. The aforementioned search spaces are designed to find specific building blocks to fit in a larger architecture that is very specific to a problem in mind (e.g., image classification) and that means that the final design might be sub-optimal as shown in [17]. The same goes for recent high performing architectures in the text domain [9].

The main contribution of this paper is to propose a novel approach to search for DNN architectures aimed at resolving the two search problems mentioned above by: (a) defining an incremental search; (b) using a transferable training; and (c) using a set of generic neural network blocks.

Our proposed algorithm narrows down the search space compared to the exhaustive search approaches mentioned above. It does so by: (1) using predefined blocks which simplifies the search space dramatically; (2) using less expensive greedy algorithms for the search in order to alleviate sequential data training issues which are inherently slower on GPUs; and (3) structuring the search in a way that enables the greedy search to imitate the RL "exploration vs. exploitation" duality by creating a two-phase learning.

Finally, in addition to searching for an architecture with significantly fewer parameters, we also ensemble the best architecture found. Ensembling is a well studied field [18], it is domain agnostic and it is a natural way to increase the size of the network given a good performing model architecture. Furthermore, by using ensembling over the final architecture, we shrink down the search problem even further, and speed up the training of the models as they have fewer parameters.

We show that our algorithm surpasses human designed networks for keyword spotting (specifically, the task of phoneme classification) and language identification problems. For keyword spotting, we improve upon the human design network from [2] by 0.3% percent using a model that is half the size. As for language identification, we improve upon the best design model from [19] by 4.09% percent, without the use of the custom designed loss function. If we don't restrict the number of parameters, our system increases accuracy by 6.1%.

http://dx.doi.org/10.21437/Interspeech.2019-1916

This paper is organized as follows. Section 2 describes our algorithm, starting with the notation (Section 2.1), followed by describing the search part (Section 2.3) and the ensembling module (Section 2.4). Finally, Section 3 presents the results and Section 4 some conclusions.

2. Description of the algorithm

In the following section we describe our algorithm to construct DNN architectures via a search method over different types of neural network layers (e.g., convolution or recurrent).

2.1. Definitions and notation

We consider the standard supervised learning scenario and assume that training and test examples are drawn i.i.d. according to some distribution and denote by $S = \{(x_1, y_1), \ldots, (x_s, y_s)\} \subseteq \mathcal{X} \times \mathcal{Y}$ a training set of size s.

We define a neural network *building block*, as any network architecture that inputs and outputs a tensor. Examples of valid blocks are a recurrent or a convolutional layer.

Let B_1 and B_2 be building blocks. For a given input example $x \in \mathcal{X}$ we denote by $B_2(B_1(x))$ the network architecture that is the result of stacking block B_2 on top of block B_1.

Assume that $\mathcal{B}$ is the family of all possible n blocks. In general, we define $B^{(\mathbf{a})}$ as a *tower*, a sequence of k blocks $B_i \in \mathcal{B}$ defined by vector $\mathbf{a} \in [n]^k$. That vector describes the type of block used on every layer, that is, $[B_{a_1}, \ldots, B_{a_k}]$. The architecture of a neural network is then defined as follows,

$$B^{(\mathbf{a})}(x) = B_{a_k}(B_{a_{k-1}}(\ldots B_{a_1}(x)). \tag{1}$$

For a set of building blocks $\mathcal{B}$, we denote by $\mathcal{C}$, the set of all architectures that are the result of stacking blocks from $\mathcal{B}$, i.e.,

$$\mathcal{C} = \left\{ B^{(\mathbf{a}_i)} : \forall i > 0, B^{(\mathbf{a}_i)} \in \mathcal{B}, \mathbf{a}_i \in [n]^{k_i}, k_i \in \mathbb{Z} \right\}.$$

For a tower $B^{(\mathbf{a})} \in \mathcal{C}$, we also denote by $\mathcal{L}(B^{(\mathbf{a})})$ the final loss on the test data of the architecture $B^{(\mathbf{a})}$. Given p towers, $\{B^{(\mathbf{a}_1)}, B^{(\mathbf{a}_2)}, \ldots, B^{(\mathbf{a}_p)}\}$, the neural network formed by averaging the logits (i.e., the pre-activation output) is defined as:

$$f = \sum_{i=1}^{p} w_i B^{(\mathbf{a}_i)}, \tag{2}$$

where in this paper we use $w_i = 1/p$.

2.2. Search space

Our algorithm is an optimized search over the family of building blocks $\mathcal{B}$ with the aim to output a weighted average of towers. Formally speaking, the family of functions that define an ensemble is:

$$\mathcal{H} = \left\{ x \mapsto \sum_{i=1}^{p} w_i B^{(\mathbf{a}_i)}(x) : w_i \in \mathbb{R}, \mathbf{a}_i \in [n]^{k_i}, k_i \in \mathbb{Z} \right\},$$

where $p \in \mathbb{Z}$ is the number of towers, $w_i \in \mathbb{R}$ and $\mathbf{a}_i$ are the weight and architecture definition of the i-th tower, respectively. Note that each tower has different depth, being k_i the number of blocks of the i-th tower.

As we will see next, our main algorithm can be divided into two equally important components: search and ensembling. The search component (denoted by $\mathcal{S}$ and described in Section 2.3) is in charge of exploration. The ensembling component ($\mathcal{A}$ in Section 2.4) is in charge of exploitation.

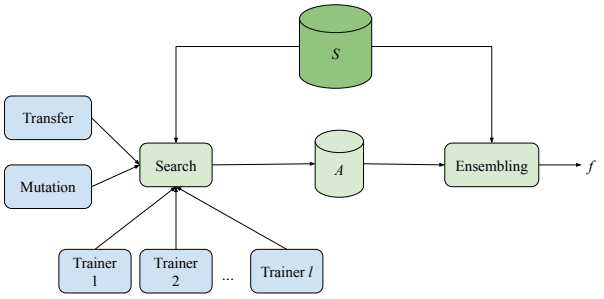

Figure 1: *Distributed search and ensembling. Each trainer runs independently and the search algorithm invokes mutation over the best architecture that is found so far. S is the set of training and validation examples and A are all the candidates used during training and search.*

2.3. Tower search algorithm

The goal of the search component $\mathcal{S}$ described in Algorithm 1 is to find the optimal DNN architecture formed by stacking k blocks. As can be seen in Figure 1, this algorithm works iteratively by mutating a set of candidate architectures $C \subset \mathcal{C}$. This search runs in a distributed asynchronous fashion using l parallel trainers.

Algorithm 1 Search algorithm $\mathcal{S}$

1: **Input:** S_{train}, $S_{validation}$ (examples), A (all candidates)
2: **while** True **do**
3: $C \leftarrow$ SELECTCANDIDATES(A)
4: $B^{(\mathbf{a})}, B^{(\mathbf{b})} \leftarrow$ MUTATE(C)
5: $B^{(\mathbf{b}')} \leftarrow$ TRANSFER($B^{(\mathbf{a})}, B^{(\mathbf{b})}$)
6: $B^{(\mathbf{b}')} \leftarrow$ TRAIN($B^{(\mathbf{b}')}, S_{train}$)
7: $\mathcal{L}(B^{(\mathbf{b}')}) \leftarrow$ EVAL($B^{(\mathbf{b}')}, S_{validation}$)
8: $A \leftarrow A \cup \left\{ B^{(\mathbf{b}')}, \mathcal{L}(B^{(\mathbf{b}')}) \right\}$
9: **end while**

The mutation module (MUTATE) used in Algorithm 1 applies random mutations to a set of candidates C to produce a new architecture. As can be seen in Algorithm 2, the mutation takes into account the maximum number of blocks a tower can have, K, and the set of all possible blocks $\mathcal{B}$.

First, the mutation algorithm selects the best architecture that has the minimum loss from the set of candidates. It will grow this architecture in case it has performed enough exploration at this depth (line 3 of Algorithm 2) and it is within the allowed depth range. M is a special parameter that controls the amount of exploration needed to advance to deeper networks. Higher values of M prevent the algorithm from advancing to deep networks too fast, and forces it to do proper exploration for the current depth [1]. When the algorithm doesn't grow the network (lines 7-10), it applies a mutation on a random block. Here, the function z=RANDOM(j) generates a random number such that $z \in \mathbb{Z}, 0 < z \leq j$.

Once MUTATE selects the next architecture to train, and before actually training, we apply transfer learning from the original model before mutating. The details about how the module TRANSFER creates a new tower are of critical importance here. From a candidate tower $B^{(\mathbf{a})} \in \mathcal{C}$ and the new mutated architecture definition $\mathbf{b}$, we construct $B^{(\mathbf{b})}$ by transferring the

[1] A generic value of $M = 5$ is used across tasks and models.

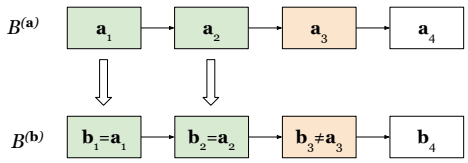

Figure 2: *Example of parameter transfer between two architectures. Blocks in green match and blocks in orange do not and so mark the boundary for parameter transfer between $B^{(\mathbf{a})}$ and $B^{(\mathbf{b})}$. Final transfer occurs in the intersecting blocks $\{b_1, b_2\}$.*

knowledge from $B^{(\mathbf{a})}$ to $B^{(\mathbf{b})}$ of those modules that are shared in both towers (see Figure 2). So, the final architecture $\mathbf{b}'$ constructed by parameter transfer taking into account $\mathbf{a}$ and $\mathbf{b}$ is defined as follows:

$$\mathbf{b}' = \{b_i : \forall j \leq i, a_j = b_j\}, \tag{3}$$

where a_j and b_j are the j-th block of tower $B^{(\mathbf{a})}$ and $B^{(\mathbf{b})}$, respectively. Hence, the training process in line 6 of Algorithm 1 will fine tune the parameters of the new tower $B^{(\mathbf{b}')}$.

Algorithm 2 Mutation Algorithm

1: **Input:** C (Candidates)
2: $B^{(\mathbf{a})} \leftarrow \arg\min_{B^{(\mathbf{a})} \in C} \mathcal{L}(B^{(\mathbf{a})})$
3: **if** $|\mathbf{a}| < \lfloor |C|/M \rfloor$ **and** $|\mathbf{a}| < K$ **then**
4: $\quad u \leftarrow \text{RANDOM}(|\mathcal{B}|)$
5: $\quad \mathbf{b} \leftarrow \text{EXTEND}(\mathbf{a}, u)$
6: **else**
7: $\quad i \leftarrow \text{RANDOM}(k)$
8: $\quad u \leftarrow \text{RANDOM}(|\mathcal{B}|)$
9: $\quad \mathbf{b} \leftarrow \mathbf{a}$
10: $\quad \mathbf{b}_i \leftarrow u$
11: **end if**
12: **return** $B^{(\mathbf{a})}$, $B^{(\mathbf{b})}$

Finally, in parallel to the search, we use bayesian optimization to tune different hyperparameters (e.g., learning rate).

2.4. Ensembling Algorithms

Once the search algorithm $\mathcal{S}$ has produced the best DNN architecture, the ensembling module $\mathcal{A}$ (see Algorithm 3) produces an average weighted ensemble of a number of repetitions of that candidate, retraining them from scratch with different shuffling of the data and different initialization parameters.

Algorithm 3 Ensembling Algorithm $\mathcal{A}$

1: **Input:** S_{train}, $S_{validation}$ (examples), l (int), A (all candidates)
2: **while** $i \in [p]$ (in parallel) **do**
3: $\quad \mathbf{a} \leftarrow \text{BESTCANDIDATE}(A)$
4: $\quad h_i \leftarrow \text{TRAIN}(B^{(\mathbf{a})}, S_{train})$
5: **end while**
6: $f \leftarrow \sum_{i=1}^{p} w_i h_i$
7: $\mathcal{L}(f) \leftarrow \text{EVAL}(f, S_{validation})$
8: **return** $\mathcal{L}(f)$, f

Algorithm 3 generates p towers in parallel. For each of them, BESTCANDIDATE selects the candidate with minimum loss from A and that is trained to produce a tower h_i.

Table 1: *Final results of accuracy and number of parameters.*

TASK	SYSTEM	ACCURACY	# PARAMS
KEYWORD SPOTTING	OURS	**97.04**	**184k**
	PREVIOUS	96.7	315K
LANGUAGE ID	OURS	59	**2.7M**
	OURS	**62.77**	5.4M
	OURS	**64.02**	8M
	PREVIOUS	60.3	5M

3. Experiments

In this section we present the results of applying our system on two well studied voice control problems. We have chosen what we believe are two challenging tasks where a comparison with human models designed over the years proves the good performance of our approach.

For both sets of experiments, each candidate architecture is trained for 10 million steps during the search phase. When ensembling is invoked then the best architectures are then retrained for 50 million steps.

3.1. Language identification

Language identification aims to identify a language given a spoken sentence by the user. The input $\mathcal{X}$ takes values in $\mathbb{R}^{400\times 40}$, that is, a maximum of 400 audio frames where each frame has 40 dimensional log-mel filterbanks features that are computed every 25ms with a 10ms shift. The total number of output languages to detect is 79. The dataset and the feature settings are identical to the ones used in [19].

Our system runs 15 trainers in parallel and each trainer uses 350 workers. We run the search for one week. Note that in order to speed up the search and favor exploration of different architectures, the model is not trained until full convergence.

Our search algorithm used the building blocks $\mathcal{B}_1$ defined in Table 2 where RNN_k and $LSTM_k$ define a Recurrent Neural Network and a Long Short-Term Memory [20] cell with k units, respectively; $PROJ_k$ is a dense projection (1d-convolution) with k outputs. Finally $SVDF_{k-d}$ denotes a SVDF cell [21] with k units, memory d and rank 1.

Table 2: *Types of blocks used in language identification ($\mathcal{B}_1$) and keyword spotting ($\mathcal{B}_2$).*

TYPE	$\mathcal{B}_1$ (DIMENSIONS)	$\mathcal{B}_2$ (DIMENSIONS)
RNN_k	64, 128, 256, 512	64, 128, 256
$PROJ_k$	64, 128, 256, 512	64, 128, 256
$SVDF_{k-d}$	64-16, 128-16 256-16, 512-16 1024-16	64-4, 128-4, 256-4 512-4, 64-8, 128-8 256-8, 64-16 128-16, 256-16
$LSTM_k$	128, 256, 512, 1024	–

Table 3 lists the best 5 architectures found and Figure 3 shows the accuracy progression of those architectures.

It took our system 83 iterations [2] (first architecture in Table 3) and approximately 59 hours to get to its best architecture. As can be seen, the search clearly favors LSTM cells over any

[2]Each iteration is the number of mutations required to generate the current architecture.

Table 3: *Loss at 10 million steps (search phase). Top 5 architectures found for language identification (right) and keyword spotting (left). Note that for keyword spotting the architecture refers to the encoder. The decoder is fixed to three stacked layers of* $SVDF_{32-32}$.

KEYWORD SPOTTING		LANGUAGE IDENTIFICATION	
ARCHITECTURE	LOSS	ARCHITECTURE	LOSS
$SVDF_{256-16}, RNN_{256}, SVDF_{64-16}$	0.0880834	$LSTM_{256}, LSTM_{512}, LSTM_{256}$	1.5922
$SVDF_{256-16}, RNN_{256}, SVDF_{64-16}, RNN_{128}$	0.0929639	$LSTM_{256}, LSTM_{512}, LSTM_{256}, RNN_{256}$	1.6562
$SVDF_{256-16}, RNN_{256}, SVDF_{64-16}, SVDF_{128-4}$	0.0935278	$LSTM_{256}, LSTM_{512}, LSTM_{256}, SVDF_{1024-16}$	1.8217
$SVDF_{256-16}, RNN_{256}, SVDF_{64-16}, RNN_{256}$	0.0938140	$LSTM_{256}, LSTM_{512}, LSTM_{256}, SVDF_{1024-16}$	1.8518
$SVDF_{256-16}, RNN_{256}$	0.0940614	$LSTM_{256}, LSTM_{512}, LSTM_{256}, RNN_{512}$	1.8614
PREVIOUS	0.0985776	PREVIOUS	2.256

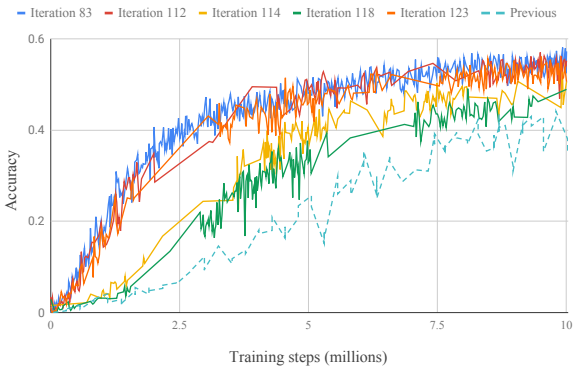

Figure 3: *Language identification accuracy while searching the top 5 architectures and the previous system.*

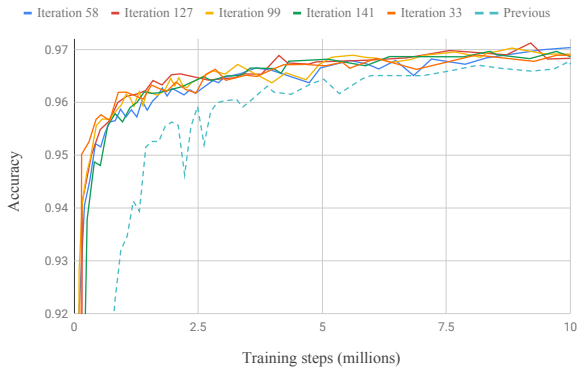

Figure 4: *Keyword spotting accuracy while searching the top 5 architectures and the previous system.*

other cells. That matches the current intuitions of the human designed model. However, unlike the human designed network that is over parameterized, the search algorithm favored a shallower smaller architecture that is well hypertuned [3].

The best architecture found during search is in iteration 83. Its accuracy climbs close to 59% when trained for evaluation (i.e., 50 million steps). Note that this model has almost half the number of parameters compared to the current best model (see Table 1). When ensembling this configuration twice ($p = 2$), we improve the accuracy to 62.77%. This model uses almost the same number of parameters as the production model, which is human designed and achieves 60.3% accuracy when trained for 50 million steps. That is, our model has a relative increase of 4.09% with the same number of parameters. Further, when ensembling with $p = 3$ and train for 50 million steps its accuracy achieves 64.02%, a relative improvement of around 6.1%.

3.2. Keyword Spotting

The problem of keyword spotting involves two parts: detecting the phonemes (i.e., encoder) spotted in an "Ok Google" sequence; and triggering when the right sequence of phonemes is heard (i.e., decoder). In this paper we focused on the former, that is, the objective of our search aims at finding the best DNN architecture for the encoder, keeping the decoder's architecture intact.

Our input consists of an audio signal of variable length. The sound is translated to a sequential input, where each element in the sequence consists of 3 time frames of 40 dimensional log-mel filterbanks. The labels are the phonemes of each input in the sequence.

[3]Batch size 20; dropout between blocks, 0.3; exponential decay, 0.86156; number of times applying step decay 4; learning rate $5.24 \cdot 10^{-5}$ and Adam optimizer [22].

Our dataset, settings and decoder are identical to the ones shown in [2]. However, unlike the previous work where a weighted loss of the encoder and decoder was used to train the whole system, now we alternate between training the encoder and decoder until convergence.

Our search algorithm used the building blocks $\mathcal{B}_2$ defined in Table 2. Our system runs 15 trainers in parallel and each trainer uses 200 workers which takes one week for a full training. Table 3 lists the best 5 architectures found by the search algorithm and Figure 4 shows the accuracy progression.

For the keyword spotting problem, we serve the model on multiple devices and different models exists for each language and locale. This introduces constraints on the model. Namely, it needs a limited number of parameters for low latency inference and it should be fast to train. Therefore, when searching for the best architecture ensembling is disabled. We also remove computationally expensive layers like LSTMs from the search.

The system found the best model at iteration 58. It took it approximately 56 hours to find it. Our found model is significantly smaller that the human designed one, almost half the size. It achieves accuracy of 97.04% on detecting phonemes on the test set compared to the human designed larger model that achieves 96.7%.

4. Conclusion

In this paper we have presented a successful algorithm for architectural search applied to two speech processing task, keyword spotting and language identification. We have shown how our algorithm can successfully find competitive architectures that have better accuracy, have fewer parameters and are designed with minimum human intervention. Our architectures are smaller and converge faster than the existing ones.

5. References

[1] F. R. rahman Chowdhury, Q. Wang, I. L. Moreno, and L. Wan, "Attention-based models for text-dependent speaker verification," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5359–5363.

[2] H. J. P. R. Alvarez, "End-to-end streaming keyword spotting," *CoRR*, vol. abs/1812.02802, 2019. [Online]. Available: https://arxiv.org/pdf/1812.02802.pdf

[3] Y. Pinsky. (2017) Tomato, tomahto. google home now supports multiple users. https://www.blog.google/products/assistant/tomato-tomahto-google-home-now-supports-multiple-users.

[4] M. Matei. (2017) Voice match will allow google home to recognize your voice. https://www.androidheadlines.com/2017/10/voice-match-will-allow-google-home-to-recognize-your-voice.html.

[5] O. Bojar, C. Federmann, M. Fishel, Y. Graham, B. Haddow, P. Koehn, and C. Monz, "Findings of the 2018 conference on machine translation (wmt18)," in *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*. Belgium, Brussels: Association for Computational Linguistics, Oct. 2018, pp. 272–303. [Online]. Available: http://www.aclweb.org/anthology/W18-6401

[6] G. Hinton, L. Deng, D. Yu, G. Dahl, A. rahman Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition," *Signal Processing Magazine*, 2012.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017. [Online]. Available: http://doi.acm.org/10.1145/3065386

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[10] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[11] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2962–2970. [Online]. Available: http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf

[12] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," *CoRR*, vol. abs/1707.07012, 2017. [Online]. Available: http://arxiv.org/abs/1707.07012

[13] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," *CoRR*, vol. abs/1802.01548, 2018. [Online]. Available: http://arxiv.org/abs/1802.01548

[14] C. Wong and A. Gesmundo, "Transfer learning to learn with multitask neural model search," *CoRR*, vol. abs/1710.10776, 2017. [Online]. Available: http://arxiv.org/abs/1710.10776

[15] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," *CoRR*, vol. abs/1806.09055, 2018. [Online]. Available: http://arxiv.org/abs/1806.09055

[16] T. Véniat, O. Schwander, and L. Denoyer, "Stochastic adaptive neural architecture search for keyword spotting," *CoRR*, vol. abs/1811.06753, 2018. [Online]. Available: http://arxiv.org/abs/1811.06753

[17] V. Macko, C. Weill, H. Mazzawi, and J. Gonzalvo, "Improving neural architecture search image classifiers via ensemble learning," 2019.

[18] Z.-H. Zhou, *Ensemble Methods: Foundations and Algorithms*, 1st ed. Chapman & Hall/CRC, 2012.

[19] L. Wan, P. Sridhar, Y. Yu, Q. Wang, and I. L. Moreno, "Tuplemax loss for language identification," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[21] J. Zhang, Q. Lei, and I. S. Dhillon, "Stabilizing gradients for deep neural networks via efficient SVD parameterization," *CoRR*, vol. abs/1803.09327, 2018. [Online]. Available: http://arxiv.org/abs/1803.09327

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980