INTERSPEECH 2021
30 August – 3 September, 2021, Brno, Czechia

Partially-Connected Differentiable Architecture Search for Deepfake and Spoofing Detection

Wanying Ge, Michele Panariello, Jose Patino, Massimiliano Todisco and Nicholas Evans

EURECOM, Sophia Antipolis, France
firstname.lastname@eurecom.fr

Abstract

This paper reports the first successful application of a differentiable architecture search (DARTS) approach to the deepfake and spoofing detection problems. An example of neural architecture search, DARTS operates upon a continuous, differentiable search space which enables both the architecture and parameters to be optimised via gradient descent. Solutions based on partially-connected DARTS use random channel masking in the search space to reduce GPU time and automatically learn and optimise complex neural architectures composed of convolutional operations and residual blocks. Despite being learned quickly with little human effort, the resulting networks are competitive with the best performing systems reported in the literature. Some are also far less complex, containing 85% fewer parameters than a Res2Net competitor.

Index Terms: neural architecture search, differentiable architecture search, deepfakes, anti-spoofing, automatic speaker verification

1. Introduction

Compared to automatic speaker verification for which the research history is decades long, research in deepfake or spoofing detection is relatively embryonic. While recent years have seen rapid progress, front-end feature extraction as well as back-end classification approaches are still evolving [1]. Early work is characterised by a focus on front-end feature engineering, namely the design of parameters or representations which capture the tell-tale signs of manipulated or synthesized speech signals and which help to distinguish these from bona fide speech [2, 3]. More recently, greater attention has been paid to the back-end classifier design. Like all fields of speech processing, deep neural network architectures are the classifier of choice [4, 5].

The use of end-to-end (E2E) processing, whereby handcrafted and manually optimised components are replaced with automatically designed and optimised substitutes, has attracted growing attention. Thus far, E2E developments extend mostly to the front-end components [6, 7]. While back-end components can be similarly optimised, this usually extends only to the network *parameters*; the network *architecture* itself is almost always still hand-crafted. Inspired by original work in [8, 9], our first attempt to harness the potential of fully E2E processing [10] explored the use of neuro-evolution for augmenting topologies (NEAT). While NEAT is successful in learning network architectures automatically, performance was found to be far from the state of the art, while computational complexity was found to be prohibitive. Whereas more efficient NEAT implementations are reported in the literature [11, 12], we have instead turned to powerful and efficient alternatives with proven potential in speech-related tasks.

We have explored the use of neural architecture search (NAS), originally proposed in [13]. NAS solutions are based upon an architecture *search space*, a *search strategy* and an *evaluation strategy* [14]. A search space contains a set of candidate *operations*. Using some performance criteria, an architecture is selected from this space and further optimised. The particular variant of NAS known as differentiable architecture search (DARTS) [15], enables the selection of candidate operations, and hence the architecture, from a search space with continuous and learnable weights. DARTS models can be optimised with backpropagation in the usual manner with hardware acceleration. The network is designed automatically by optimising the operations contained within architecture building blocks referred to as *cells*. Candidate operations, including convolutional operations, pooling layers, and residual connections among others, are selected during an initial search phase, before the resulting cells are stacked together to build a deeper architecture which is then further optimised. The resulting networks resemble the current state of the art in anti-spoofing, hence our adoption of DARTS in this work.

This paper reports our use of a particular variant of DARTS known as partial channel connections (PC-DARTS) [16] for anti-spoofing. We show how partial channel connections, which deliver substantial savings in both computational complexity and memory, enable the automatic learning of a neural network based solution to anti-spoofing. Both the network architecture and parameters are learned automatically with only minimal human input. To the best of our knowledge, our work is both the first reported application of DARTS to anti-spoofing and the first reported application of PC-DARTS in *any* field of speech processing. The remainder of the paper is organised as follows. Section 2 introduces the related work and objectives. The proposed system is reported in Section 3. Experiments and results are reported in Sections 4 and 5. Our findings and conclusions are reported in Section 6.

2. Related work and objectives

DARTS has already been applied successfully to speech and language tasks [17–19]. Its use for architecture search in a keyword spotting task is reported in [17]. Competitive results were obtained with a search space containing the regular operations used in ResNet. A successful application to automatic speech recognition reported in [18] showed promising results even when architecture search and training stages are performed using different language datasets. The first application of DARTS to speaker verification is reported in [19] which shows that smaller, automatically learned solutions compare favourably to hand-crafted architectures. While results comparable to the state of the art are reported in both [17] and [19], both also report the necessary use of small batch sizes so that architecture search can be performed upon a single GPU.

http://dx.doi.org/10.21437/Interspeech.2021-1187

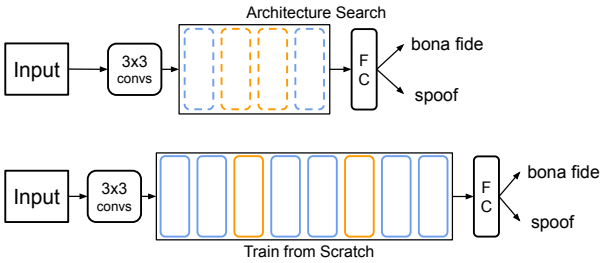


Figure 1: *An illustration of architecture search and train from scratch. Architecture search optimises a stack of 2 normal cells (dashed blue) and reduction cells (dashed yellow). The train from scratch stage optimises a deeper network of stacked cells (solid blue and yellow). Only network parameters are optimised in the second stage; the cell architectures are those fixed during architecture search.*

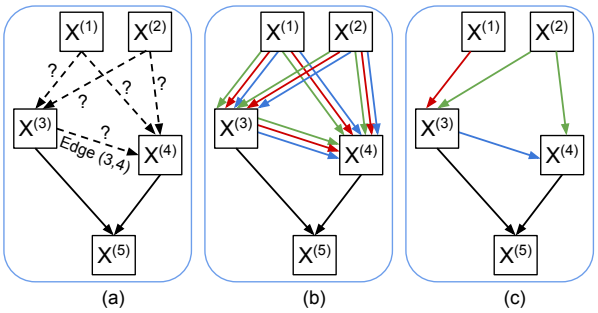


Figure 2: *An illustration of architecture search: (a) a neural cell with $N = 5$ nodes; (b) an illustration of the candidate operations performed on each edge that are optimised during architecture search; (c) resulting optimised cell with $K = 2$ inputs to each intermediate node.*

The first objective of our work is hence to determine whether neural architectures learned automatically with PC-DARTS can compete with hand-crafted networks. Second, we seek to determine the longer term scope for such networks to even outperform the current state of the art. Third, we are interested to learn whether automatically learned and optimised solutions are more efficient. While not an objective of the current work, our hypothesis is also that PC-DARTS may yield less complex networks whose behaviour may be more easily *explained*.

3. PC-DARTS

As illustrated in Figure 1, DARTS encompasses a pair of learning stages referred to as *architecture search* (top half) and *train from scratch* (bottom half). A key idea is to construct a complex network architecture from a pair of building blocks, referred to as *cells* (blue and yellow blocks in Figure 1), whose internal architecture and parameters are learned automatically. In contrast to other NAS approaches which search over a discrete set of candidate network operations, DARTS operates upon a relaxed, continuous search space. This makes the architecture representation itself *differentiable*, meaning that it can be optimised in the usual manner via gradient descent and backpropagation with hardware acceleration. In the architecture search stage, the cell *architecture parameters* are learned and fixed. The train from scratch stage operates upon a deeper network formed from the stacking of a greater number of cells, thereby forming a deeper residual network. The *network parameters* are then re-optimised. The initial architecture search stage is computationally demanding. The use of partial connections (PC-DARTS) provides a more efficient solution. Since neither DARTS, much less PC-DARTS are mainstream within the speech community, a brief overview of both is provided in the following.

3.1. Searching for the Optimal Architecture

DARTS networks are constructed from the concatenation of multiple *cells*. An example is illustrated in Figure 2. Their internal architectures are learned automatically and dictate the sequence of operations performed upon input data in generating their output.

Each cell contains N nodes, where each node $\mathbf{x}^{(i)}$ represents a feature map in tensor form. The first pair of nodes, $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$, are the cell inputs and are connected to the outputs of the previous two cells. Nodes $\mathbf{x}^{(3)}$ to $\mathbf{x}^{(N-1)}$, referred to as intermediate nodes, are computed from previous nodes with operation o selected from the search space $\mathcal{O}$ according to:

$$\mathbf{x}^{(j)} = \sum_{i<j} o^{(i,j)} \left(\mathbf{x}^{(i)}\right) \tag{1}$$

where $o^{(i,j)}$ is the operation performed on edge (i, j) that connects $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$. Node $\mathbf{x}^{(N)}$ is the cell output: its feature map is constructed from the concatenation of the feature maps corresponding to the full set of intermediate nodes.

In the architecture search stage, a linear combination of operations, denoted as $\bar{o}$, is performed on edge (i, j) according to some weight $\alpha_o^{(i,j)}$. The weights form a continuous search space through a softmax function:

$$\bar{o}^{(i,j)}\left(\mathbf{x}^{(i)}\right) = \sum_{o\in\mathcal{O}} \frac{\exp\left(\alpha_o^{(i,j)}\right)}{\sum_{o'\in\mathcal{O}} \exp\left(\alpha_{o'}^{(i,j)}\right)} o\left(\mathbf{x}^{(i)}\right) \tag{2}$$

Architecture search is hence reduced to the learning of a set of continuous variables $\boldsymbol{\alpha} = \{\alpha^{(i,j)}\}$, where $\alpha^{(i,j)}$ is a vector of dimension $|\mathcal{O}|$. Both the *architecture parameters* $\boldsymbol{\alpha}$ and the *network parameters* $\boldsymbol{\omega}$ (e.g. the convolutional filter weights) can be jointly optimised through backpropagation. The goal is to determine the $\boldsymbol{\alpha}$ which minimises the validation loss L_{val}, where the optimal $\boldsymbol{\omega}$ is determined by minimising the training loss $L_{train}(\boldsymbol{\omega}, \boldsymbol{\alpha})$:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} L_{val}(\boldsymbol{\omega}^*, \boldsymbol{\alpha}) \\ &\text{s.t. } \boldsymbol{\omega}^* = \operatorname*{argmin}_{\boldsymbol{\omega}} L_{train}(\boldsymbol{\omega}, \boldsymbol{\alpha}) \end{aligned} \tag{3}$$

When the search stage is complete, $\bar{o}^{(i,j)}$ is replaced with the single operation with the highest $\alpha_o^{(i,j)}$. The final cell architecture is obtained by retaining the set of K edges entering each intermediate node which have the highest weights $\alpha_o^{(i,j)}$, where K is a hyperparameter. The remainder are discarded.

The search space $\mathcal{O}$ proposed in [19] comprises: a 3×3 separable convolution; a 5×5 separable convolution; a 3×3 dilated convolution; a 5×5 dilated convolution; a skip connection; a 3×3 average pooling; a 3×3 max pooling; none (no connection). The set of operations are used in defining two types of neural cells, namely *normal* cells and *reduction* cells. As illustrated to the base of Figure 1, cells are stacked together

to form the full, deeper residual network, with reduction cells being placed at the $\frac{1}{3}$ and $\frac{2}{3}$ depth positions of the total network depth (number of stacked cells). Feature map dimensions are fixed for the input and output of each normal cell. Reduction cells act to reduce the feature map dimensions by 50% while doubling the number of channels.

3.2. Partial Channel Connections and Edge Normalisation

DARTS remains computationally demanding, especially in the architecture search stage. To improve efficiency, we used partial channel connections and edge normalisation [16]. Partially-connected DARTS (PC-DARTS) delivers substantial savings in computation and memory. For a given edge (i, j), partial channel connections are formed from the element-wise multiplication of $\mathbf{x}^{(i)}$ by a masking operator $\mathbf{S}^{(i,j)}$ of the same dimension. The masking operator either *selects* (multiplication by 1) or *masks* (multiplication by 0) each channel in $\mathbf{x}^{(i)}$:

$$\bar{o}^{(i,j)}\left(\mathbf{x}^{(i)}\right) = \sum_{o\in\mathcal{O}} \frac{\exp\left(\alpha_o^{(i,j)}\right)}{\sum_{o'\in\mathcal{O}}\exp\left(\alpha_{o'}^{(i,j)}\right)} o\left(\mathbf{S}^{(i,j)}\odot\mathbf{x}^{(i)}\right) + \left(1-\mathbf{S}^{(i,j)}\right)\odot\mathbf{x}^{(i)} \quad (4)$$

where $\odot$ indicates element wise multiplication. A hyperparameter K_C is set to conserve a random fraction $1/K_C$ of the available channels. Partial connections hence reduce the computational load by a factor K_C while acting to regularise the choice of weight-free candidate operations (e.g., max pooling) in $\mathcal{O}$ for a given edge [16]. There is hence a trade off between performance (smaller K_C) and efficiency (larger K_C). As a result of random channel sampling, the linear combination of operations $\bar{o}^{(i,j)}$ for each node can become unstable under training. This issue is addressed by introducing a set of *edge normalisation* parameters $\boldsymbol{\beta}$ which smooth node inputs according to:

$$\mathbf{x}^{(j)} = \sum_{i<j} \frac{\exp\left(\beta^{(i,j)}\right)}{\sum_{i'<j}\exp\left(\beta^{(i',j)}\right)} \bar{o}^{(i,j)}\left(\mathbf{x}^{(i)}\right) \quad (5)$$

where $\beta^{(i,j)}$ is a learnable smoothing factor. The set of architecture parameters optimised by minimizing L_{val} now comprises both $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

4. Experiments

In this section we describe the experimental setup, the choice of front-end and our specific PC-DARTS configuration.

4.1. Database, protocols and metrics

All work reported in this paper was performed using the ASVspoof 2019 Logical Access (LA) database [20] which comprises the usual train, development and evaluation partitions. In the architecture search stage, a random selection of half the number of utterances for each class in the training partition, including bona fide and spoofed (A01-A06), is used to learn network parameters. The other half is used to learn architectures, namely one normal cell and one reduction cell. The cell architectures which produce the highest classification accuracy are then used in the train from scratch stage.

After the train from scratch stage, the performance of the resulting model is assessed using the full evaluation partition. Performance is reported in terms of the pooled minimum normalised tandem detection cost function (min-tDCF) [21] in addition to the pooled equal error rate (EER).

4.2. Front-end

Initial experiments showed that the application of neural architecture search to raw audio waveforms places excessive demands upon GPU memory, implying lower batch sizes and greater training time [22]. We hence used linear frequency cepstral coefficients (LFCCs) of 60 dimensions encompassing static, delta and delta-delta coefficients. Features are extracted using 64 ms Hamming windows with a 16 ms shift and a 1024-point FFT. In order to improve generalisation, frequency masking is applied according to the procedure described in [23] with a maximum of 12 masked frequency channels per mini-batch.

4.3. PC-DARTS

As is customary [16], we applied three convolutional layers of stride 2 to the input features in order to reduce resolution. Architecture search is performed using 4 neural cells (2 normal cells and 2 reduction cells) with 16 initial channels. Each cell has $N = 7$ nodes, and each intermediate node retains $K = 2$ inputs after search.

Training for the architecture search stage is performed for 50 epochs with a batch size of 64 using an Adam optimiser to learn both architecture and network parameters. Both are optimised by minimising the weighted cross-entropy loss between spoofed and bona fide data with a ratio $1 : 9$. According to [16, 24], architecture parameters are not updated in the first 10 epochs. For the learning of architecture parameters we used a learning rate of 6e-4 and a weight decay of 0.001. For network parameters, we used an initial learning rate of 0.01 which is annealed down to 0.001 according to a cosine schedule. Partial channel connections use a value of $K_C = 2$. When the architecture search stage is complete, network parameters $\boldsymbol{\omega}$ are forgotten. Only the normal and reduction cell architectures are then retained.

During the train from scratch stage, models are trained for 100 epochs with a batch size of 128 and an initial learning rate of 0.001. The drop-path rate [16] is set to 0.2. We experimented with different numbers of stacked layers (L) and initial channels (C). The models are optimized with the same loss function as in the architecture search stage. The final scores are taken from the output for the bona fide class.

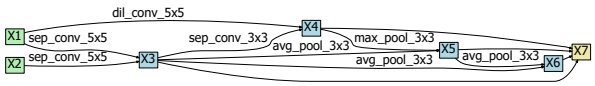
All experiments reported in this paper were performed on a single NVIDIA GeForce RTX 2080 Ti GPU. Using the implementation available online[1], all results are reproducible with the same random seed and GPU environment.

5. Results

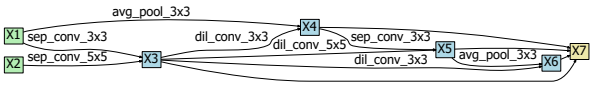
5.1. Architecture Search

The architecture search stage is the most computationally expensive. We are hence interested in both the search time and performance, both of which are illustrated in Table 1 for experiments with DARTS and PC-DARTS for models with 4 layers and 16 channels ($L = 4, C = 16$). In DARTS case, the batch size is set to the largest possible given GPU memory constraints. The use of partial connections improves on search time by approximately 50% while regularisation results in improved accuracy. Performance also translates well from the training partition to the development partition. Illustrations of the resulting normal and reduction cell operations are shown in Figure 3.

[1] https://github.com/eurecom-asp/pc-darts-anti-spoofing



(a) *Normal cell*



(b) *Reduction cell*

Figure 3: *An illustration of the (a) normal and (b) reduction cells resulting from architecture search. As illustrated in Figure 1, they form the basic building blocks used to construct the architecture used in the train from scratch stage.*

Table 1: *A comparison of DARTS and PC-DARTS models with $L = 4$ layers and $C = 16$ channels. Results in terms of processing efficiency (GPU-days) and accuracy for ASVspoof 2019 LA training and development partitions.*

Model size	Systems	Search Cost GPU-days	Best Architecture Train Acc	Best Architecture Dev Acc
$(L = 4,$	DARTS	0.29	98.80	97.21
$C = 16)$	PC-DARTS	0.15	99.97	100

Table 2: *Number of parameters and results for a selection of different PC-DARTS models. Results for the ASVspoof 2019 LA database.*

Model size	Params	Dev min-tDCF	Dev EER	Eval min-tDCF	Eval EER
$(L = 2, C = 4)$	0.007M	0.0004	0.04	0.1244	5.80
$(L = 4, C = 16)$	0.14M	0	0	0.0992	5.53
$(L = 8, C = 32)$	0.97M	0.00004	0.002	0.1177	4.87
$(L = 16, C = 64)$	7.51M	0	0	0.0914	4.96
$(L = 24, C = 64)$	10.57M	0.0001	0.039	0.1045	5.51

5.2. Train from Scratch

Table 2 shows results for a set of different PC-DARTS configurations (column 1) and number of parameters (column 2). min-tDCF and EER results are shown for both the development partition (columns 3 and 4) and evaluation partition (columns 5 and 6). According to the primary min-tDCF metric, the best performing model has 16 layers and 64 initial channels. For the evaluation partition, it delivers a min-tDCF of 0.0914 and an EER of 4.96%. The second best model with 4 layers and 16 initial channels delivers a min-tDCF of 0.0992 and an EER of 5.53%. This is achieved with 7.37M fewer parameters. Performance for the smallest model is substantially degraded in terms of min-tDCF, albeit if the EER is still respectable. The largest tested model size offers no benefit in terms of performance which is likely the result of over-fitting to training data.

5.3. Comparison to competing systems

Table 3 shows a comparison of results to top-performing systems reported in the literature and the two ASVspoof baselines [25]. The best (16,64) model achieves substantially better performance than the two ASVspoof baselines and also outperforms all but two others, both Res2Net models. Even then, the differences in terms of min-tDCF are modest (even if greater in terms of EER). Our second best model, with 85% fewer parameters than the best Res2Net model, remains competitive. These are satisfying results and are the first to show that anti-spoofing models whose *architecture and parameters* are learned automatically can compete with models designed with models designed with far greater human effort.

Table 3: *A performance comparison between PC-DARTS models and competing state-of-the-art systems reported in the literature. Results for the ASVspoof LA evaluation partition.*

Systems	Features	min-tDCF	EER	Params
Res2Net [26]	CQT	0.0743	2.50	0.96M
Res2Net [26]	LFCC	0.0786	2.87	0.96M
PC-DARTS $(16, 64)$	LFCC	0.0914	4.96	7.51M
PC-DARTS $(4, 16)$	LFCC	0.0992	5.53	0.14M
LCNN [27] [28]	LFCC	0.1000	5.06	10M
LCNN [27] [28]	LPS	0.1028	4.53	10M
LFCC-GMM [25]	LFCC	0.2116	8.09	-
Res2Net [26]	LPS	0.2237	8.78	0.96M
CQCC-GMM [25]	CQCC	0.2366	9.57	-
Deep Res-Net [29]	LPS	0.2741	9.68	0.31M

6. Conclusions

This paper reports what is, to the best of our knowledge, the first successful application of neural architecture search (NAS) to the spoofing detection problem. We show that partially connected differentiable architecture search (PC-DARTS) is able to learn complex neural architectures from a fixed set of candidate operations. Architectures learned with PC-DARTS can be optimised using backpropagation and with hardware acceleration, meaning that even complex convolutional and residual networks can be learned automatically.

The performance of the resulting models is competitive with the state of the art. Our best performing model achieves a min-tDCF of 0.09 for the ASVspoof 2019 Logical Access database, a result outdone only by a Res2Net system, and even then only by a modest margin. Given that our result was generated by a network whose architecture and parameters are all learned automatically, instead of from many hours of manual optimisation, this is a satisfying result. Our second-best system which achieves a min-tDCF of 0.1 has 85% fewer parameters than the best performing Res2Net system. With these results, we are encouraged to pursue PC-DARTS further. The obvious next step is to apply PC-DARTS directly to raw signal inputs. Other directions include the use of PC-DARTS as a full end-to-end solution to both spoofing detection and automatic speaker verification.

7. Acknowledgements

This work is supported by the TReSPAsS-ETN project funded from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No.860813. It is also supported by the ExTENSoR project funded by the French Agence Nationale de la Recherche (ANR).

8. References

[1] M. Sahidullah, H. Delgado, M. Todisco, T. Kinnunen, N. Evans, J. Yamagishi, and K. Lee, "Introduction to voice presentation attack detection and recent advances," in *Handbook of Biometric Anti-Spoofing - Presentation Attack Detection, Second Edition*, ser. Advances in Computer Vision and Pattern Recognition. Springer, 2019, pp. 321–361.

[2] M. Todisco, H. Delgado, and N. Evans, "A New Feature for Automatic Speaker Verification Anti-Spoofing: Constant Q Cepstral Coefficients," in *Proc. Speaker Odyssey 2016*, vol. 2016, 2016, pp. 283–290.

[3] Y. Zhang, F. Jiang, and Z. Duan, "One-class learning towards generalized voice spoofing detection," *arXiv preprint arXiv:2010.13995*, 2020.

[4] R. Li, M. Zhao, Z. Li, L. Li, and Q. Hong, "Anti-Spoofing Speaker Verification System with Multi-Feature Integration and Multi-Task Learning," in *Proc. Interspeech 2019*, 2019, pp. 1048–1052.

[5] Cheng-I Lai and Nanxin Chen and Jesús Villalba and Najim Dehak, "ASSERT: Anti-Spoofing with Squeeze-Excitation and Residual Networks," in *Proc. Interspeech 2019*, 2019, pp. 1013–1017.

[6] J.-w. Jung, H.-S. Heo, J.-h. Kim, H.-j. Shim, and H.-J. Yu, "RawNet: Advanced End-to-End Deep Neural Network Using Raw Waveforms for Text-Independent Speaker Verification," *Proc. Interspeech 2019*, pp. 1268–1272, 2019.

[7] M. Ravanelli and Y. Bengio, "Speaker recognition from raw waveform with sincnet," in *Proc. IEEE SLT*. IEEE, 2018, pp. 1021–1028.

[8] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[9] A. Daniel, "Evolving Recurrent Neural Networks That Process and Classify Raw Audio in a Streaming Fashion," in *Proc. Interspeech 2017*, 2017, pp. 2040–2041.

[10] G. Valenti, H. Delgado, M. Todisco, N. W. Evans, and L. Pilati, "An end-to-end spoofing countermeasure for automatic speaker verification using evolving recurrent neural networks," in *Proc. Speaker Odyssey 2018*, 2018, pp. 288–295.

[11] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artificial life*, vol. 15, no. 2, pp. 185–212, 2009.

[12] S. Risi and K. O. Stanley, "Enhancing es-hyperneat to evolve more complex regular neural networks," in *Proc. Genetic and evolutionary computation 2011*, 2011, pp. 1539–1546.

[13] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *Proc. ICLR 2017*, 2017.

[14] T. Elsken, J. H. Metzen, F. Hutter *et al.*, "Neural architecture search: A survey." *Machine Learnign Research*, vol. 20, no. 55, pp. 1–21, 2019.

[15] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable Architecture Search," in *Proc. ICML 2019*, 2019.

[16] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong, "PC-DARTS: Partial channel connections for memory-efficient architecture search," *arXiv preprint arXiv:1907.05737*, 2019.

[17] T. Mo, Y. Yu, M. Salameh, D. Niu, and S. Jui, "Neural Architecture Search for Keyword Spotting," in *Proc. Interspeech 2020*, 2020, pp. 1982–1986.

[18] Yi-Chen Chen and Jui-Yang Hsu and Cheng-Kuang Lee and Hung-yi Lee, "DARTS-ASR: Differentiable Architecture Search for Multilingual Speech Recognition and Adaptation," in *Proc. Interspeech 2020*, 2020, pp. 1803–1807.

[19] S. Ding, T. Chen, X. Gong, W. Zha, and Z. Wang, "AutoSpeech: Neural Architecture Search for Speaker Recognition," in *Proc. Interspeech 2020*, 2020, pp. 916–920.

[20] X. Wang, J. Yamagishi, M. Todisco, H. Delgado, A. Nautsch, N. Evans, M. Sahidullah, V. Vestman, T. Kinnunen, K. A. Lee *et al.*, "ASVspoof 2019: A large-scale public database of synthesized, converted and replayed speech," *Computer Speech & Language*, vol. 64, p. 101114, 2020.

[21] T. Kinnunen, K. A. Lee, H. Delgado, N. Evans, M. Todisco, M. Sahidullah, J. Yamagishi, and D. Reynolds, "t-DCF: a Detection Cost Function for the Tandem Assessment of Spoofing Countermeasures and Automatic Speaker Verification," in *Proc. Speaker Odyssey 2018*, 2018.

[22] H. Tak, J. Patino, M. Todisco, A. Nautsch, N. Evans, and A. Larcher, "End-to-end anti-spoofing with RawNet2," *Proc. ICASSP 2020 (to appear)*, 2020.

[23] T. Chen, A. Kumar, P. Nagarsheth, G. Sivaraman, and E. Khoury, "Generalization of audio deepfake detection," in *Proc. Speaker Odyssey 2020*, 2020, pp. 1–5.

[24] Z. Yu, C. Zhao, Z. Wang, Y. Qin, Z. Su, X. Li, F. Zhou, and G. Zhao, "Searching Central Difference Convolutional Networks for Face Anti-Spoofing," pp. 5294–5304, 2020.

[25] M. Todisco, X. Wang, V. Vestman, M. Sahidullah, H. Delgado, A. Nautsch *et al.*, "ASVspoof 2019: Future Horizons in Spoofed and Fake Audio Detection," in *Proc. Interspeech 2019*, 2019, pp. 1008–1012.

[26] X. Li, N. Li, C. Weng, X. Liu, D. Su, D. Yu, and H. Meng, "Replay and Synthetic Speech Detection with Res2net Architecture," *arXiv preprint arXiv:2010.15006*, 2020.

[27] G. Lavrentyeva, S. Novoselov, A. Tseren, M. Volkova, A. Gorlanov, and A. Kozlov, "STC Antispoofing Systems for the ASVspoof2019 Challenge," pp. 1033–1037, 2019.

[28] S. Liu, H. Wu, H.-y. Lee, and H. Meng, "Adversarial attacks on spoofing countermeasures of automatic speaker verification," in *Proc. IEEE ASRU 2019*. IEEE, 2019, pp. 312–319.

[29] M. Alzantot, Z. Wang, and M. B. Srivastava, "Deep Residual Neural Networks for Audio Spoofing Detection," *Proc. Interspeech 2019*, pp. 1078–1082, 2019.