On Learning the Width of Neural Networks

Federico Errica[1], Henrik Christiansen[2], Viktor Zaverkin[2], Mathias Niepert[2], Francesco Alesiani[2]
1. NEC Italia, 2. NEC Laboratories Europe
name.surname@neclab.eu

***Abstract*—We introduce an easy-to-use technique to learn an *unbounded* width of a neural network's layer *during training*. The technique does not rely on alternate optimization nor hand-crafted gradient heuristics; rather, it jointly optimizes the width and the parameters of each layer via simple backpropagation. We apply the technique to a broad range of data domains such as tables, images, texts, and graphs, showing how the width adapts to the task's difficulty. By imposing a soft ordering of importance among neurons, it is also possible to dynamically compress the network with no performance degradation.**

***Index Terms*—Neural Networks, Learning the Number of Neurons, Adaptive Width Learning, Dynamic Architectures, Information Compression, Variational Inference**

I. Introduction

Since the construction of the Mark I Perceptron machine the effective training of neural networks has remained an open research problem of great academic and practical value. The Mark I solved image recognition tasks by exploiting a layer of 512 *fixed* "association units" that in modern language correspond to the hidden units of a (possibly) Multi-Layer Perceptron (MLP). MLPs possess universal approximation capabilities when assuming *arbitrary width* [1] and sigmoidal activations, and their convergence to good solutions was studied, for instance, in [2] where the backpropagation algorithm was described as "simple, easy to implement on parallel hardware", and improvable by other techniques such as momentum that preserve locality of weight updates.

Yet, after almost 70 years of progress, the vast majority of neural networks, be they shallow or deep, still rely on a fixed choice of the number of neurons in their hidden layers. This property is typically treated as an architectural design choice, one of the many hyper-parameters that have to be carefully tuned whenever we approach a new task. The tuning process has many names, such as model selection, hyper-parameter tuning, and cross-validation, and it is associated with non-negligible costs: Different architectural configurations are to be trained until one that performs best on a validation set is selected. The configurations' space grows exponentially in the number of layers, so practitioners often resort to shortcuts such as picking a specific number of hidden units for *all* layers, which greatly reduces the search space together with the chances of selecting a better architecture for the task.

This short version of a submitted work introduces a *simple* and *easy to use* technique to learn the width of each neural network's layer without imposing upper bounds (we refer to it as **unbounded width**). The width of each layer is *dynamically* adjusted during backpropagation, and it only requires a slight modification to the neural activations that does not alter the ability to parallelize computation.

II. Related Work

Constructive methods dynamically adjust neural network width. Cascade correlation [3] and firefly network descent [4] grow networks during training. Lifelong learning algorithms [5], [6] split/duplicate neurons for new tasks. [7] use heuristics to expand/shrink MLPs and CNNs, while [8] optimize depth. Our approach differs by directly computing loss gradients to adjust width, avoiding heuristics. Bayesian nonparametrics [9], instead, explore infinite cluster learning. Orthogonal methods like Neural Architecture Search (NAS) automate network design [10], [11] using reinforcement learning, evolution, or gradients [12]–[14]. Despite advances [15], [16], NAS remains costly and assumes a bounded search space, making it complementary to our method. Pruning [17], [18] and distillation [19] reduce model size, often with performance trade-offs. Unlike pruning, we remove connections while reducing memory; unlike distillation, we avoid retraining. These techniques can however be combined with our approach.

III. Adaptive Width Learning

We introduce a general probabilistic framework, called Adaptive Width Neural Networks (AWNN) for convenience, showing how (ultimately simple) design choices arise from a variational inference treatment of a graphical model.
We are given a dataset of N *i.i.d.* samples (x_i, y_i), with input $x_i \in \mathbb{R}^F, F \in \mathbb{N}^+$ and target y_i whose domain depends on whether the task is regression or classification. For samples X and targets Y, the objective is to maximize

$$\log p(Y|X) = \log \prod_{i=1}^{N} p(y_i|x_i) = \sum_{i=1}^{N} \log p(y_i|x_i) \qquad (1)$$

with respect to the learnable parameters of $p(y|x)$. To formalize learning of a neural network that maximizes Equation (1) and **learns** an unbounded width for each hidden layer ℓ, we assume the existence of an **infinite** sequence of *i.i.d.* latent variables $\boldsymbol{\theta}_\ell = \{\theta_{\ell n}\}_{n=1}^{\infty}$, where $\theta_{\ell n}$ is a multivariate variable over the learnable weights of neuron n at layer ℓ. Since this implies modeling an infinite-width layer, we introduce a latent variable λ_ℓ that decides how many neurons to use at each layer ℓ. That is, it "truncates" an infinite width to a **finite** value so that we can perform inference. For a network of L layers, we define $\boldsymbol{\theta} = \{\boldsymbol{\theta}_\ell\}_{\ell=1}^{L}$ and $\boldsymbol{\lambda} = \{\lambda_\ell\}_{\ell=1}^{L}$, assuming independence across layers. Therefore, one can write

$p(y_i|x_i) = \int p(y_i, \boldsymbol{\lambda}, \boldsymbol{\theta}|x_i) d\boldsymbol{\lambda} d\boldsymbol{\theta}$. We assume that the joint distribution decomposes as:

$$p(Y, \boldsymbol{\lambda}, \boldsymbol{\theta}|X) = \prod_{i=1}^{N} p(y_i, \boldsymbol{\lambda}, \boldsymbol{\theta}|x_i) \tag{2}$$

$$p(y_i, \boldsymbol{\lambda}, \boldsymbol{\theta}|x_i) = p(y_i|\boldsymbol{\lambda}, \boldsymbol{\theta}, x_i) p(\boldsymbol{\lambda}) p(\boldsymbol{\theta}) \tag{3}$$

$$p(\boldsymbol{\lambda}) = \prod_{\ell=1}^{L} p(\lambda_\ell) = \prod_{\ell=1}^{L} \mathcal{N}(\lambda_\ell; \mu_\ell^\lambda, \sigma_\ell^\lambda) \tag{4}$$

$$p(\boldsymbol{\theta}) = \prod_{\ell=1}^{L} \prod_{n=1}^{\infty} p(\theta_{\ell n}) = \prod_{\ell=1}^{L} \prod_{n=1}^{\infty} \mathcal{N}(\theta_{\ell n}; \mathbf{0}, \text{diag}(\sigma_\ell^\theta)) \tag{5}$$

$$p(y_i|\boldsymbol{\lambda}, \boldsymbol{\theta}, x_i) = \text{Neural Network of Section III-A.} \tag{6}$$

Here, $\sigma_\ell^\theta, \mu_\ell^\lambda, \sigma_\ell^\lambda$ are hyper-parameters. The neural network is parametrized by realizations $\boldsymbol{\lambda}, \boldsymbol{\theta}$, so it relies on a finite number of neurons and outputs either class probabilities (classification) or the mean of a Gaussian distribution (regression) to parametrize $p(y_i|\boldsymbol{\lambda}, \boldsymbol{\theta}, x_i)$ depending on the task. Maximizing Equation (1), however, requires computing the above integral, which is intractable. Therefore, we turn to mean-field variational inference [20] to maximize an expected lower bound (ELBO) instead. This requires to define a distribution over the latent variables $q(\boldsymbol{\lambda}, \boldsymbol{\theta})$ and re-phrase the objective as:

$$\log p(Y|X) \geq \sum_{i=1}^{N} \mathbb{E}_{q(\boldsymbol{\lambda}, \boldsymbol{\theta})} \left[\log \frac{p(y_i, \boldsymbol{\lambda}, \boldsymbol{\theta}|x_i)}{q(\boldsymbol{\lambda}, \boldsymbol{\theta})} \right], \tag{7}$$

where $q(\boldsymbol{\lambda}, \boldsymbol{\theta})$ is parametrized by learnable *variational* parameters. For a distribution f_ℓ parametrized by λ_ℓ, we factorize the variational distribution into:

$$q(\boldsymbol{\lambda}, \boldsymbol{\theta}) = q(\boldsymbol{\lambda}) q(\boldsymbol{\theta}|\boldsymbol{\lambda}) \tag{8}$$

$$q(\boldsymbol{\lambda}) = \prod_{\ell=1}^{L} q(\lambda_\ell) = \prod_{\ell=1}^{L} \mathcal{N}(\lambda_\ell; \nu_\ell, 1) \tag{9}$$

$$q(\boldsymbol{\theta}|\boldsymbol{\lambda}) = \prod_{\ell=1}^{L} \prod_{n=1}^{D_\ell} q(\theta_{\ell n}) \prod_{D_\ell+1}^{\infty} p(\theta_{\ell n}) \tag{10}$$

$$q(\theta_{\ell n}) = \mathcal{N}(\theta_{\ell n}; \text{diag}(\rho_{\ell n}), \mathbf{I}). \tag{11}$$

$$D_\ell = \text{quantile function of } f_\ell(\cdot; \lambda_\ell) \text{ evaluated at } k \tag{12}$$

The value k is a hyper-parameter, $\nu_\ell, \rho_{\ell n}$ are variational parameters and, as before, we define $\boldsymbol{\rho}_\ell = \{\rho_{\ell n}\}_{n=1}^{D_\ell}$, $\boldsymbol{\rho} = \{\boldsymbol{\rho}_\ell\}_{\ell=1}^{L}$ and $\boldsymbol{\nu} = \{\nu_\ell\}_{\ell=1}^{L}$. Note that the set of variational parameters is **finite**. The **truncated width** D_ℓ, that is the finite number of neurons at layer ℓ, is computed as the quantile function evaluated at k of a distribution f_ℓ with infinite support over $\mathbb{N}^+$, parametrized by λ_ℓ. *W.l.o.g.*, we implement f_ℓ as a discretized exponential distribution, following the discretization strategy of [21]: For a natural x, the discretized distribution relies on the cumulative distribution function of the exponential:

$$f_\ell(x; \lambda_\ell) = (1 - e^{\lambda_\ell(x+1)}) - (1 - e^{\lambda_\ell(x)}). \tag{13}$$

We choose the exponential because it is a **monotonically decreasing** function and allows us to impose an ordering of importance among neurons, as detailed in Section III-A.

By expanding Equation 7 using the above definitions and approximating the expectations at the first order, *i.e.,* $\mathbb{E}_{q(\boldsymbol{\lambda})}[f(\boldsymbol{\lambda})] = f(\boldsymbol{\nu})$ and $\mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\lambda})}[f(\boldsymbol{\theta})] = f(\boldsymbol{\rho})$ as in [8], we obtain the final form of the objective:

$$\sum_{\ell}^{L} \log \frac{p(\nu_\ell; \mu_\ell^\lambda, \sigma_\ell^\lambda)}{q(\nu_\ell; \nu_\ell)} + \sum_{\ell}^{L} \sum_{n=1}^{D_\ell} \log \frac{p(\rho_{\ell n}; \sigma_\ell^\theta)}{q(\rho_{\ell n}; \rho_{\ell n})} + \\ + \sum_{i=1}^{N} \log p(y_i|\boldsymbol{\lambda}{=}\boldsymbol{\nu}, \boldsymbol{\theta}{=}\boldsymbol{\rho}, x_i), \tag{14}$$

where distributions' parameters are made explicit to distinguish them. The first two terms in the loss regularize the width of the layers and the magnitude of the parameters, respectively, whereas the third is is the predictive loss.

In practice, the finite variational parameters $\boldsymbol{\nu}, \boldsymbol{\rho}$ are those used by the neural network in place of $\boldsymbol{\lambda}, \boldsymbol{\theta}$, which enables easy optimization via backpropagation. Maximizing Equation (14) will change each variational parameter ν_ℓ, which in turn will change the value of D_ℓ **during training**. If D_ℓ increases we initialize new neurons and draw their weights from a standard normal distribution, otherwise we remove the excess ones.

A. Imposing a Soft Ordering on Neurons' Importance

Now that the learning objective has been formalized, the missing ingredient is the definition of the neural network $p(y_i|\boldsymbol{\lambda}{=}\boldsymbol{\nu}, \boldsymbol{\theta}{=}\boldsymbol{\rho}, x_i)$ of Equation 6 as a modified MLP. Compared to a standard MLP, we need to make use of the variational parameters $\boldsymbol{\nu}$ that affect the truncation width at each hidden layer, whereas $\boldsymbol{\rho}$ are the weights. We choose a monotonically decreasing function f_ℓ, so that when a new neuron is added its relative importance is low and will not drastically impact the network. In other words, we are imposing a soft ordering of importance among neurons. We modify the classical activation h_j^ℓ of a hidden neuron j at layer ℓ as

$$h_j^\ell = \sigma \left(\sum_{k=1}^{D_{\ell-1}} w_{jk}^\ell h_k^{\ell-1} \right) f_\ell(j; \nu_\ell), \tag{15}$$

where $D_{\ell-1}$ is the truncated width of the previous layer, σ is a non-linear activation function and $w_{jk}^\ell \in \rho_{\ell j}$. That is, we rescale the activation of each neuron k by its "importance" $f_\ell(j; \nu_\ell)$. Note that the bias parameter is taken into account by concatenating a dummy value 1 to $h_k^{\ell-1}$.

IV. Experiments and Setup

We first quantitatively verify that AWNN does not harm the performance compared to baseline models and compare the chosen width by means of grid-search model selection with the learned width of AWNN. Secondly, we check that AWNN chooses a higher width for harder tasks, which can be seen as increasing the hypotheses space until the neural network finds a good path to convergence. Finally, we analyze the ability to compress information during training.

We compare a baseline that undergoes proper hyper-parameter tuning against its AWNN version, where we replace any fixed MLP with an adaptive one. For space reasons, we

Table I
QUANTITATIVE RESULTS AND CHOSEN WIDTH. "LINEAR" MEANS LINEAR MODEL AS OPPOSED TO AN MLP.

	Fixed		AWNN		Width (Fixed)	Width (AWNN)	
	Mean	(Std)	Mean	(Std)		Mean	(Std)
DoubleMoon	100.0	(0.0)	100.0	(0.0)	8	8.1	(2.8)
Spiral	99.5	(0.5)	99.8	(0.1)	16	65.9	(8.7)
SpiralHard	98.0	(2.0)	100.0	(0.0)	32	227.4	(32.4)
MNIST	99.6	(0.1)	99.7	(0.0)	Linear	19.4	(4.8)
CIFAR10	91.4	(0.2)	91.4	(0.2)	Linear	80.1	(12.4)
CIFAR100	66.5	(0.4)	63.1	(4.0)	256	161.9	(57.8)
NCI1	80.0	(1.4)	80.0	(1.1)	Unknown	731.3	(128.2)
REDDIT-B	87.0	(4.4)	90.2	(1.3)	Unknown	793.6	(574.0)
Multi30k (↓)	1.43	(0.4)	1.51	(0.2)	24576	123.2	(187.9)

cannot provide a detailed hyper-parameters list, but the guiding principle is that the set of hyper-parameters tried is the same with the exception of the width, which in AWNN is learned. First, we train an MLP on 3 synthetic two-dimensional tabular tasks of increasing binary classification difficulty, namely a double moon (2500 samples), a spiral (2500 samples), and a double spiral (5000 samples) that we call SpiralHard. A stratified hold-out split of 70% training/10% validation/20% test for risk assessment is chosen at random for these datasets. Similarly, we consider a ResNet-20 [22] trained on 3 image classification tasks, namely MNIST, CIFAR10, and CIFAR100, where data splits and preprocessing are taken from the original paper and AWNN is applied to the downstream classifier. In the graph domain, we train a Graph Isomorphism Network on the NCI1 and REDDIT-B classification tasks using the same split and evaluation setup of [23] (in this case, we report published results). On all these tasks, the metric of interest is the accuracy. Finally, for the textual domain we train a Transformer architecture on the Multi30k English-German translation task [24], using a pretrained GPT-2 Tokenizer, and we evaluate the cross-entropy loss over the translated words. On tabular, image, and text-based tasks, an internal validation set (10%) for model selection is extracted from the union of outer training and validation sets, and the best configuration chosen according to the internal validation set is retrained 10 times on the outer train/validation/test splits, averaging test performances.

V. RESULTS

We begin by discussing the quantitative results of our experiments: Table I reports means and standard deviations across the 10 final training runs. In terms of performance, we observe that AWNN is more stable or accurate than a fixed MLP on DoubleMoon, Spiral, and SpiralHard; all other things being equal, it seems that using more neurons and their soft ordering are the main contributing factors to these improvements. On the image datasets, performances of AWNN are comparable to those of the fixed baseline but for CIFAR100, due to an unlucky run that did not converge. In this case, AWNN learns a smaller total width compared to grid search.

Results on graph datasets are interesting in two respects: First, the performance on REDDIT-B is significantly improved by AWNN both in terms of average performance and stability of results; second, the total learned width is significantly higher than those tried in [23], [25], meaning that a biased choice of a good range of width has had a profound influence on the estimation of the risk for a specific family of DGN models (i.e., GIN). This result makes it evident that it is important to let the network decide how many neurons are necessary to solve the task. Finally, the results on the Multi30k show that the AWNN Transformer learns to use 200x parameters less than the fixed Transformer for the feed-forward networks, achieving a statistically comparable test loss.

A. Adaptation to Task Difficulty and Convergence

Intuitively, one would expect that AWNN learned larger widths for more difficult tasks. This is indeed what happens on the tabular datasets where some tasks are clearly harder than others. Figure 1 (left) shows that, given the same starting width per layer, the learned number of neurons grows according to the task's difficulty. It also appears that convergence is not affected by the introduction of AWNN, as investigated in Figure 1 (right), which was not obvious considering the parametrization constraints encouraged by the rescaling of neurons' activations.

B. Online Network Compression via Regularization

So far, we have used an uninformative prior $p(\lambda)$ over the neural networks' width. We demonstrate the effect of an informative prior by performing an annealing experiment on the SpiralHard dataset. We set an uninformative $p(\boldsymbol{\theta})$ and ReLU6 nonlinearity. At epoch 1000, we introduce $p(\lambda_\ell) = \mathcal{N}(\lambda_\ell; 0.05, 1)$, and gradually anneal the standard deviation up to 0.1 at epoch 2500. Figure 2 shows that the width of the network reduces from approximately 800 neurons to 300 without any test performance de gradation. We hypothesize that the least important neurons mostly carry negligible information, and therefore they can be safely removed without drastic changes in the output of the model. This technique might be useful to compress large models with billions of parameters.

VI. CONCLUSIONS

We introduced a new methodology to learn an unbounded width of neural network layers within a single training, by imposing a soft ordering of importance among neurons. Our approach requires very few changes to the architecture, adapts the width to the task's difficulty, and does not impact negatively convergence. A by-product of neurons' ordering is the ability to easily compress the network during training.

REFERENCES

[1] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[3] S. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Proceedings of the 3rd Conference on Neural Information Processing Systems (NIPS)*, 1989.

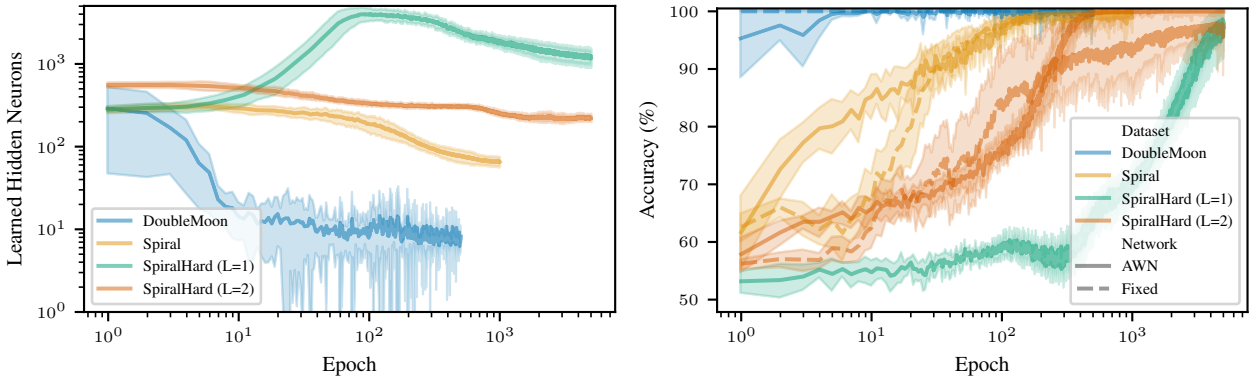

Figure 1. (Left) The learned width adapts to the increasing difficulty of the task, from the DoubleMoon to SpiralHard. (Right) AWNN reaches perfect test accuracy with a comparable amount of epochs on DoubleMoon and Spiral, while it converges faster on SpiralHard.

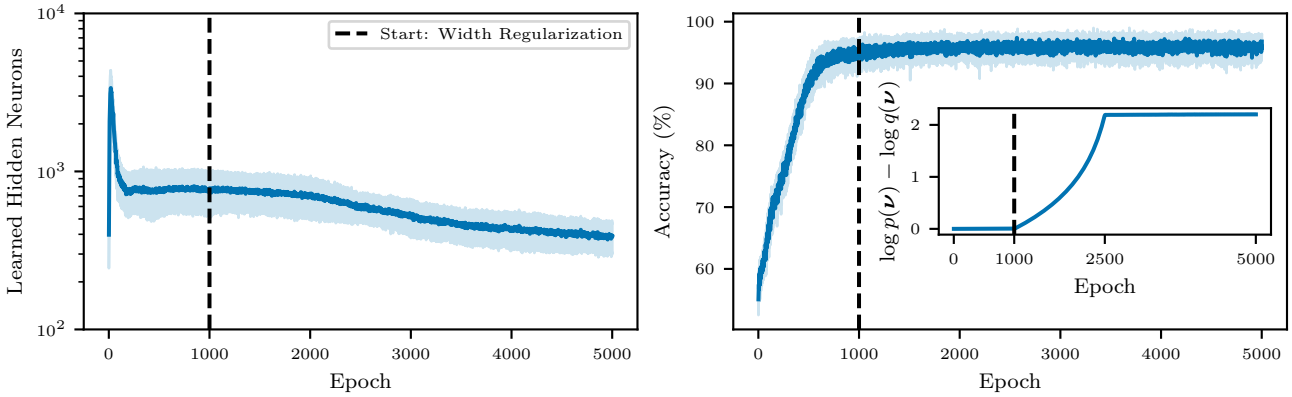

Figure 2. It is possible to regularize the width at training time by increasing the magnitude of the loss term $\log \frac{p(\nu)}{q(\nu)}$. The total width is reduced by more than 50% (left) while preserving accuracy (right). The inset plot refers to the loss term that AWNN tries to maximize.

[4] L. Wu, B. Liu, P. Stone, and Q. Liu, "Firefly neural architecture descent: a general approach for growing neural networks," in *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020.

[5] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, "Lifelong learning with dynamically expandable networks," in *6th International Conference on Learning Representations (ICLR)*, 2018.

[6] L. Wu, D. Wang, and Q. Liu, "Splitting steepest descent for growing neural architectures," in *Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[7] R. Mitchell, M. Mundt, and K. Kersting, "Self expanding neural networks," *arXiv preprint*, 2023.

[8] A. Nazaret and D. Blei, "Variational inference for infinitely deep neural networks," in *Proceedings of the 39th International Conference on Machine Learning (ICML)*, 2022.

[9] P. Orbanz and Y. W. Teh, "Bayesian nonparametric models," *Encyclopedia of machine learning*, vol. 1, 2010.

[10] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, pp. 1–21, 2019.

[11] C. White, M. Safari, R. Sukthanker, B. Ru, T. Elsken, A. Zela, D. Dey, and F. Hutter, "Neural architecture search: Insights from 1000 papers," *arXiv preprint*, 2023.

[12] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *4th International Conference on Learning Representations (ICLR)*, 2016.

[13] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[14] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *7th International Conference on Learning Representations (ICLR)*, 2019.

[15] A. Brock, T. Lim, J. Ritchie, and N. Weston, "SMASH: One-shot model architecture search through hypernetworks," in *6th International Conference on Learning Representations (ICLR)*, 2018.

[16] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[17] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?," in *Proceedings of machine learning and systems (MLSys)*, 2020.

[18] A. Mishra, J. A. Latorre, J. Pool, D. Stosic, D. Stosic, G. Venkatesh, C. Yu, and P. Micikevicius, "Accelerating sparse deep neural networks," *arXiv preprint*, 2021.

[19] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint*, 2015.

[20] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[21] D. Roy, "The discrete normal distribution," *Communications in Statistics - Theory and Methods*, vol. 32, pp. 1871–1883, 2003.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[23] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," in *8th International Conference on Learning Representations (ICLR)*, 2020.

[24] D. Elliott, S. Frank, K. Sima'an, and L. Specia, "Multi30k: Multilingual english-german image descriptions," in *Proceedings of the 5th Workshop on Vision and Language*, 2016.

[25] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *7th International Conference on Learning Representations (ICLR)*, 2019.