BayesNAS: A Bayesian Approach for Neural Architecture Search

Hongpeng Zhou [1 *] **Minghao Yang** [1 *] **Jun Wang** [2] **Wei Pan** [1]

Abstract

One-Shot Neural Architecture Search (NAS) is a promising method to significantly reduce search time without any separate training. It can be treated as a Network Compression problem on the architecture parameters from an over-parameterized network. However, there are two issues associated with most one-shot NAS methods. First, dependencies between a node and its predecessors and successors are often disregarded which result in improper treatment over *zero* operations. Second, architecture parameters pruning based on their magnitude is questionable. In this paper, we employ the classic Bayesian learning approach to alleviate these two issues by modeling architecture parameters using *hierarchical automatic relevance determination* (HARD) priors. Unlike other NAS methods, we train the over-parameterized network for only *one* epoch then update the architecture. Impressively, this enabled us to find the architecture on CIFAR-10 within only 0.2 GPU days using a single GPU. Competitive performance can be also achieved by transferring to ImageNet. As a byproduct, our approach can be applied directly to compress convolutional neural networks by enforcing structural sparsity which achieves extremely sparse networks without accuracy deterioration.

1. Introduction

Neural Architecture Search (NAS), the process of automating architecture engineering, is thus a logical next step in automating machine learning since (Zoph & Le, 2017). There are basically three existing frameworks for neural architecture search. Reinforcement learning based NAS (Baker et al., 2017; Zoph & Le, 2017; Zhong et al., 2018; Zoph et al., 2018; Cai et al., 2018) methods take the generation of a neural architecture as an agent's action with the action space identical to the search space. More recent neuro-evolutionary approaches (Real et al., 2017; Liu et al., 2018b; Real et al., 2019; Miikkulainen et al., 2019; Xie & Yuille, 2017; Elsken et al., 2019a) use gradient-based methods for optimizing weights and solely use evolutionary algorithms for optimizing the neural architecture itself. However, these two frameworks take enormous computational power when compared to a search using a single GPU. One-Shot based NAS is a promising approach to significantly reduce search time without any separate training, which treats all architectures as different subgraphs of a supergraph (the one-shot model) and shares weights between architectures that have edges of this super-graph in common (Saxena & Verbeek, 2016; Brock et al., 2018; Pham et al., 2018; Bender et al., 2018; Liu et al., 2019b; Cai et al., 2019; Xie et al., 2019; Zhang et al., 2019a;b). A comprehensive survey on Neural Architecture Search can be found in (Elsken et al., 2019b).

Our approach is a one-shot based NAS solution which treats NAS as a Network Compression/pruning problem on the architecture parameters from an over-parameterized network. However, despite it's remarkable less searching time compared to reinforcement learning and neuro-evolutionary approaches, we can identify a number of significant and practical disadvantages of the current one-shot based NAS. First, dependencies between a node and its predecessors and successors are disregarded in the process of identifying the redundant connections. This is mainly motivated by the improper treatment of *zero* operations. On one hand, the logit of *zero* may dominate some of the edges while the child network still has other non-zero edges to keep it connected (Liu et al., 2019b; Xie et al., 2019; Cai et al., 2019; Zhang et al., 2019b), for example, node 2 in Figure1a. Similarly, as shown in Figure 1 of (Xie et al., 2019), the probability of invalid/disconnected graph sampled will be $\frac{511}{1024}$ when there are three non-zero plus one *zero* operation. Though post-processing to safely remove isolated nodes is possible, *e.g.*, for chain-like structure, it demands extensive extra computations to reconstruct the graph for complex search space with additional layer types and multiple branches and skip connections. This may prevent the use of modern network structure as the backbone such as DenseNet (Huang et al., 2017), newly designed motifs (Liu et al., 2018b) and complex computer vision tasks such as semantic segmentation

[*]Equal contribution [1]Department of Cognitive Robotics, Delft University of Technology, Netherlands [2]Department of Computer Science, University College London, UK. Correspondence to: Wei Pan <wei.pan@tudelft.nl>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

(Liu et al., 2019a). On the other hand, *zero* operations should have higher priority to rule out other possible operations, since *zero* operations equal to all *non-zero* operations not being selected. Second, most one-shot NAS methods (Liu et al., 2019b; Cai et al., 2019; Xie et al., 2019; Zhang et al., 2019b; Gordon et al., 2018) rely on the magnitude of architecture parameters to prune redundant parts and this is not necessarily true. From the perspective of Network Compression (Lee et al., 2019), magnitude-based metric depends on the scale of weights thus requiring pre-training and is very sensitive to the architectural choices. Also the magnitude does not necessarily imply the optimal edge. Unfortunately, these drawbacks exist not only in Network Compression but also in one-shot NAS.

In this work, we propose a novel, efficient and highly automated framework based on the classic Bayesian learning approach to alleviate these two issues simultaneously. We model architecture parameters by a *hierarchical automatic relevance determination* (HARD) prior. The dependency can be translated by multiplication and addition of some independent Gaussian distributions. The classic Bayesian learning framework (MacKay, 1992a; Neal, 1995; Tipping, 2001) prevents overfitting and promotes sparsity by specifying sparse priors. The uncertainty of the parameter distribution can be used as a new metric to prune the redundant parts if its associated entropy $\frac{1}{2}\ln(2\pi e\gamma_{jk}^{o'})$ is nonpositive. The majority of parameters are automatically zeroed out during the learning process.

Our Contributions

- **Bayesian approach:** BayesNAS is the first Bayesian approach for one-shot NAS. Therefore, our approach shares the advantages of Bayesian learning, which prevents overfitting and does not require tuning a lot of hyperparameters. Hierarchical sparse priors are used to model the architecture parameters. Priors can not only promote sparsity, but model the dependency between a node and its predecessors and successors ensuring a connected derived graph after pruning. Furthermore, it provides a principled way to prioritize *zero* operations over other *non-zero* operations. In our experiment on CIFAR-10, we found that the variance of the prior, as well as that of posterior, is several magnitudes smaller than posterior mean which renders a good metric for architecture parameters pruning.

- **Simple and fast search:** Our algorithm is formulated simply as an iteratively re-weighted ℓ_1 type algorithm (Candes et al., 2008) where the re-weighting coefficients used for the next iteration are computed not only from the value of the current solution but also from its posterior variance. The update of posterior variance is based on Laplace approximation in Bayesian learning which requires computation of the inverse Hessian of log likelihood. To make the computation for large networks feasible, a fast Hessian calculation method is proposed. In our experiment, we train the model for only *one* epoch before calculating the Hessian to update the posterior variance. Therefore, the search time for very deep neural networks can be kept within 0.2 GPU days.

- **Network compression:** As a byproduct, our approach can be extended directly to Network Compression by enforcing various structural sparsity over network parameters. Extremely sparse models can be obtained at the cost of minimal or no loss in accuracy across all tested architectures. This can be effortlessly integrated into BayesNAS to find sparse architecture along with sparse kernels for resource-limited hardware.

2. Related Work

Network Compression. The de facto standard criteria to prune redundant weights depends on their magnitude and is designed to be incorporated with the learning process. These methods are prohibitively slow as they require many iterations of pruning and learning steps. One category is based on the magnitude of weights. The conventional approach to achieve sparsity is by enforcing penalty terms (Chauvin, 1989; Weigend et al., 1991; Ishikawa, 1996). Weights below a certain threshold could be removed. In recent years, impressive results have been achieved using the magnitude of weight as the criterion (Han et al., 2016) as well as other variations (Guo et al., 2016). The other category is based on the magnitude of Hessian of loss with respect to weights, *i.e.*, higher the value of Hessian, greater the importance of the parameters (LeCun et al., 1990; Hassibi et al., 1993). Despite being popular, both of these categories require pre-training and are very sensitive to architectural choices. For instance, different normalization layers affect the magnitude of weights in different ways. This issue has been elaborated in (Lee et al., 2019) where the gradient information at the beginning of training is utilized for ranking the relative importance of weights' contribution to the training loss.

One-shot Neural Architecture Search. In one-shot NAS, redundant architecture parameters are pruned based on the magnitude of weights similar to that used in Network Compression. In DARTS, Liu et al. (2019b) applied a softmax function to the magnitude of w to rank the relative importance for each operation. Similar to DARTS, there are two related works: ProxylessNAS (Cai et al., 2019) and SNAS (Xie et al., 2019). ProxylessNAS binarizes w using $\texttt{clip}((1+w)/2, 0, 1)$ (Courbariaux et al., 2015) where -1 plays the role of threshold and edge with the highest weight will be selected in the end. While SNAS applies a softened one-hot random variable to rank the architecture

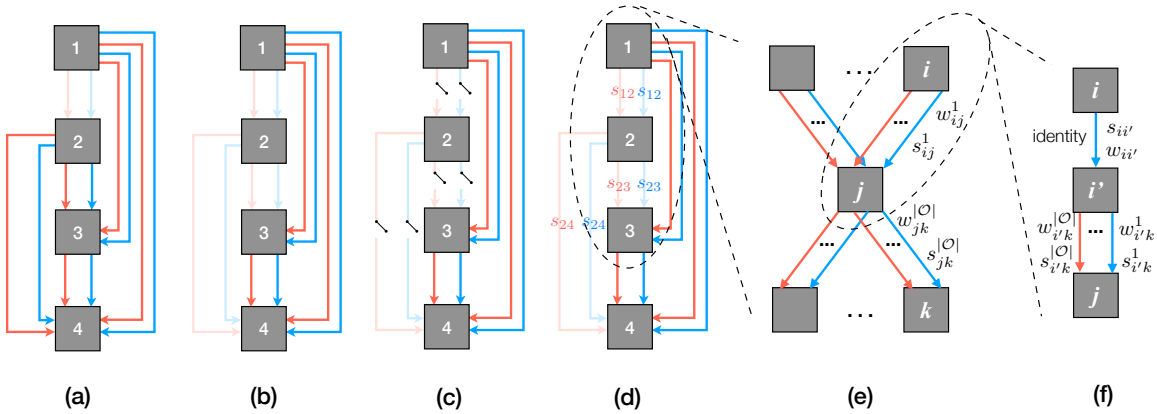

Figure 1. An illustration of BayesNAS: (a) disconnected graph with isolated node 2 caused by disregard for dependency; (b) expected connected graph with no connection from node 2 to 3 and from node 2 to 4; (c) illustration about dependency with predecessor's (e_{12}) superior control over its successors (e_{23} and e_{24}) (d) designed switches realizing the dependency and determining "on or off" of the edge; (e) elementary multi-input-multi-output motif for a graph; (f) prioritized *zero* operation over other *non-zero* operations.

parameter, Gordon et al. (2018) treats the scaling factor of Batch Normalization as an edge and normalization as its associated operation. Zhang et al. (2019b) proposed DSO-NAS which relaxes ℓ_0 norm by replacing it with ℓ_1 norm and prunes the edges by a threshold, *e.g.*, the learning rate is multiplied by a predefined regularization parameter to prune edges gradually over the course of training.

Bayesian Learning and Compression. Our approach is based on Bayesian learning. In principle, the Bayesian approach to learn neural networks does not have problems of tuning a large amount of hyperparameters or overfitting the training data (MacKay, 1992b;a; Neal, 1995; Hernández-Lobato & Adams, 2015). Recently, Bayesian learning is applied to estimate layer size and network depth in NAS problem (Dikov et al., 2019). By employing sparsity-inducing priors, the obtained model depends only on a subset of kernel functions for linear models (Tipping, 2001) and deep neural networks where the neurons can be pruned as well as all their ingoing and outgoing weights (Louizos et al., 2017). Other Bayesian methods have also been applied to network pruning (Ullrich et al., 2017; Molchanov et al., 2017a) where the former extends the soft weight-sharing to obtain a sparse and compressed network and the latter uses variational inference to learn the dropout rate that can then be used for network pruning.

3. Search Space Design

The search space defines which neural architectures a NAS approach might discover in principle. Designing a good search space is a challenging problem for NAS. Some works (Zoph & Le, 2017; Zoph et al., 2018; Pham et al., 2018; Cai et al., 2018; Zhang et al., 2019b; Liu et al., 2019b; Cai et al., 2019) have proposed that the search space could be represented by a Directed Acyclic Graph (DAG). We denote e_{ij} as the edge from node i to node j and o_{ij} stands for the operation that is associated with edge e_{ij}.

Similar to other one-shot based NAS approaches (Bender et al., 2018; Zhang et al., 2019b; Liu et al., 2019b; Cai et al., 2019; Gordon et al., 2018), we also include (different or same) scaling scalars over all operations of all edges to control the information flow, denoted as w_{ij}^o which also represent architecture parameters. The output of a mixed operation $o_{ij}, i < j$ is defined based on the outputs of its edge

$$o_j(z_i) = \sum_{o \in \mathcal{O}} w_{ij}^o o_{ij}(z_i). \tag{1}$$

Then z_j can be obtained as $\sum_{i<j} o_j(z_i)$.

To this end, the objective is to learn a simple/sparse subgraph while maintaining/improving the accuracy of the over-parameterized DAG (Bender et al., 2018). Let us formulate the search problem as an optimization problem. Given a dataset $\mathbf{D} = (\mathbf{X}, \mathbf{Y}) = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N$ and the desired sparsity level κ (*i.e.*, the number of non-zero edges), one-shot NAS problem can be written as an optimization problem with the following constraints:

$$\begin{aligned} &\min_{\boldsymbol{W}} L(\mathbf{W}; \mathbf{D}) = \min_{\boldsymbol{W}} \frac{1}{N} \sum_{n=1}^{N} \ell(\mathbf{y}_n, \text{Net}(\mathbf{x}_n, \boldsymbol{\mathcal{W}}, \mathbf{w})) \\ &\text{s.t.} \quad \boldsymbol{W} \in \mathbb{R}^{m^{\text{net}} + m^{\text{edge}}}, \quad \|\mathbf{w}\|_0 \leq \kappa^{\text{edge}} \end{aligned} \tag{2}$$

where $\mathbf{W}$ are split into two parts: network parameters $\boldsymbol{\mathcal{W}} = [\mathcal{W}_{ij}^o]$ and architecture parameters $\mathbf{w} = [w_{ij}^o]$ with dimension of m^{net} and m^{edge} respectively, and $\|\cdot\|_0$ is the standard ℓ_0 norm. The formulation in equation 2 can be substantiated by incorporating *zero* operations into $\mathcal{O}$ to allow removal of w_{ij}^o (Liu et al., 2019b; Cai et al., 2019) aiming to further reduce the size of cells and improve the design flexibility.

To alleviate the negative effect induced by the dependency

and magnitude-based metric whose issues have been discussed in Introduction, for each w_{ij}^{o}, we introduce a switch s_{ij}^{o} that is analogous to the one used in an electric circuit. There are four features associated with these switches. First, the "on-off" status is not solely determined by its magnitude. Second, dependency will be taken into account, *i.e.*, the predecessor has superior control over its successors as illustrated in Figure 1c. Third, s_{ij}^{o} is an auxiliary variable that will not be updated by gradient descent but computed directly to switch on or off the edge. Lastly, s_{ij}^{o} should work for both proxy and proxyless scenarios and can be better embedded into existing algorithmic frameworks (Liu et al., 2019b; Cai et al., 2019; Gordon et al., 2018). The calculation method will be introduced later in Section 4.

Inspired by the hierarchical representation in a DAG (Liu et al., 2019b; 2018b), we abstract a single motif as the building block of DAG, as shown in Figure 1e. Apparently, any derived motif, path, or network can be constructed by such a multi-input-multi-output motif. It shows that a successor can have multiple predecessors and each predecessor can have multiple operations over each of its successors. Since the representation is general, each directed edge can be associated with some primitive operations (*e.g.*, convolution, pooling, etc.) and a node can represent output of motifs, cells, or a network.

4. Dependency Based One-Shot Performance Estimation Strategy

4.1. Encoding the Dependency Logic

In the following, we will formally state the criterion to identify the redundant connections in Proposition 1. The idea can be illustrated by Figure 1b in which both the blue and red edges from node 2 to 3 and from node 2 to 4 might be non-zeros but should be removed as a consequence. To enable this, we have the following proposition.

Proposition 1 *There is information flow from node j to k under operation o' as shown in Figure 1e if and only if at least one operation of at least one predecessor of node j is non-zero and $w_{jk}^{o'}$ is also non-zero.*

Remark 1 *The same expression for Proposition 1 is: there is **no** information flow from node j to k under operation o' if and only if all the operation of all the predecessors of node j are zeros or $w_{jk}^{o'}$ is zero. This explains the incompleteness of the problem 2 as well as the possible phenomenon that non-zero edges become dysfunctional in Figure 1b.*

Remark 2 *The expression to encode Proposition 1 is not unique. Some examples include but not limited to, e.g., $w_{jk}^{o'}\sum_{i<j}|w_{ij}^{o}|$, $w_{jk}^{o'}\sum_{i<j}\alpha_{ij}^{o}|w_{ij}^{o}|, \forall\alpha_{ij}^{o} \in (0,1]$, $w_{jk}^{o'}\sum_{i<j}(w_{ij}^{o})^2$. Apparently, ℓ_0 norm of these quantities are difficult to be included in a constraint in the optimization problem formulation in 2.*

As can be seen in Remark 2, we will construct a probability distribution jointly over $w_{jk}^{o'}$, w_{ij}^{o}, $\forall i<j$ in the sequel, denoted as

$$p(c(w_{jk}^{o'}, w_{ij}^{o})), \forall i<j. \tag{3}$$

where c is a possible expression like in Remark 2 to encode Proposition 1.

In the following, we will show how the "switches" s can be used to implement Proposition 1. If we assume s has two states $\{\text{ON}, \text{OFF}\}$, $w_{jk}^{o'}$ is redundant when $s_{jk}^{o'}$ is OFF *or* all s_{ij}^{o} are OFF, $\forall i<j, o\in\mathcal{O}$. How to use s to encode the redundancy of $w_{jk}^{o'}$, *i.e.*, $w_{jk}^{o'}\sum_{i<j}|w_{ij}^{o}|=0$? One possible solution is

$$\bigcup_{i<j}\bigcup_{o\in\mathcal{O}} s_{ij}^{o}\cap s_{jk}^{o'} \quad \text{or} \quad \overline{\overline{\bigcup_{i<j}\bigcup_{o\in\mathcal{O}} s_{ij}^{o}} \cup \overline{s_{jk}^{o'}}} \tag{4}$$

If s is a continuous variable with $s=\infty$ for ON and 0 for OFF, set union and intersection can be arithmetically represented by addition and multiplication respectively. s does not directly determine the magnitude of w but plays the role as uncertainty or confidence for zero magnitude.

A straightforward way to encode this logic is to assign a probability distribution, for example Gaussian distribution, over $w_{jk}^{o'}$

$$p(w_{jk}^{o'}) = \mathcal{N}(w_{jk}^{o'}|0, s_{jk}^{o'}), \quad \sum_{i<j} p(w_{ij}^{o}) = \sum_{i<j}\mathcal{N}(w_{ij}^{o}|0, s_{ij}^{o})$$

Since $w_{ij}^{o}, \forall i,j,o$ are independent with each other, we construct the following distribution to express equation 3:

$$\begin{aligned} p(c(w_{jk}^{o'}, w_{ij}^{o})) &\triangleq \mathcal{N}(w_{jk}^{o'}|0, s_{jk}^{o'})\sum_{i<j}\mathcal{N}(w_{ij}^{o}|0, s_{ij}^{o}) \\ &= \mathcal{N}(w_{jk}^{o'}|0, s_{jk}^{o'})\mathcal{N}\left(\sum_{i<j}\frac{s_{ij}^{o}w_{ij}^{o}}{\sum_{i<j}s_{ij}^{o}}|0, s_{ij}^{o}\right) \\ &= \mathcal{N}\left(w_{jk}^{o'}\sum_{i<j}\frac{s_{ij}^{o}w_{ij}^{o}}{\sum_{i<j}s_{ij}^{o}}|0, \gamma_{jk}^{o'}\right) \end{aligned} \tag{5}$$

where

$$\gamma_{jk}^{o'} \triangleq \left(\frac{1}{\sum_{i<j}\sum_{o\in\mathcal{O}} s_{ij}^{o}} + \frac{1}{s_{jk}^{o'}}\right)^{-1}. \tag{6}$$

Since $s_{ij}^{o}>0$ in equation 5 always holds, regardless of what s_{ij}^{o} is, we can use the following simpler alternative to substitute equation 5 to encode Proposition 1:

$$p(c(w_{jk}^{o'}, w_{ij}^{o})) \triangleq \mathcal{N}\left(w_{jk}^{o'}\sum_{i<j}w_{ij}^{o}|0, \gamma_{jk}^{o'}\right). \tag{7}$$

Interestingly, equation 7 and 4 are equivalent. This means that we may find an algorithm that is able to find the sparse solution in a probabilistic manner. However, Gaussian distribution, in general, does not promote sparsity. Fortunately,

some classic yet powerful techniques in Bayesian learning are applicable, *i.e.*, sparse Bayesian learning (SBL) (Tipping, 2001; Pan, 2017) and automatic relevance determination (ARD) prior (MacKay, 1996; Neal, 1995) in Bayesian neural networks.

4.2. Zero Operation Ruling All

In our paper, we do not include *zero* operation as a primitive operation. Instead, between node i and j we compulsively add one more node i' and allow only a single *identity* operation (see Figure 1f). The associated weight $w_{ii'}$ is trainable and initialized to 1 as well as its switch $s_{ii'}$. The idea is that if $s_{ii'}$ is OFF, all the operations from i' to j will be disabled as a consequence. Then $\gamma_{jk}^{o'}$ in equation 6 can be substituted by

$$\gamma_{jk}^{o'} \triangleq \left(\frac{1}{s_{ii'}} + \frac{1}{\sum\limits_{i'<j} \sum\limits_{o \in \mathcal{O}} s_{i'j}^{o}} + \frac{1}{s_{jk}^{o'}} \right)^{-1}. \tag{8}$$

5. Bayesian Learning Search Strategy

5.1. Bayesian Neural Network

The likelihood for the network weights $\boldsymbol{\mathcal{W}}$ and the noise precision σ^{-2} with data $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$ is

$$p(\mathbf{Y} \,|\, \boldsymbol{\mathcal{W}}, \mathbf{w}, \mathbf{X}, \sigma^2) = \prod_{n=1}^{N} \mathcal{N}(y_n \,|\, \text{Net}(\mathbf{x}_n; \boldsymbol{\mathcal{W}}, \mathbf{w}); \sigma^2). \tag{9}$$

To complete our probabilistic model, we specify a Gaussian prior distribution for each entry in each of the weight matrices in $\boldsymbol{\mathcal{W}}$. In particular,

$$p(\boldsymbol{\mathcal{W}} \,|\, \lambda) = \prod_{i<j} \prod_{o \in \mathcal{O}} \mathcal{N}(\mathcal{W}_{ij}^{o} \,|\, 0, \lambda^{-1}) \tag{10}$$

$$p(\mathbf{w} \,|\, \mathbf{s}) = \prod_{j<k} \prod_{o \in \mathcal{O}} \prod_{o' \in \mathcal{O}} \mathcal{N}\left(w_{jk}^{o'} \sum_{i<j} w_{ij}^{o} | 0, \gamma_{jk}^{o'} \right) \tag{11}$$

where $\gamma_{jk}^{o'}$ is defined in equation 8. σ^{-2}, λ and $\mathbf{s}$ are *hyperparameters*. Importantly, there is an individual hyperparameter associated independently with every edge weight and a single one with all network weight. Follow Mackay's evidence framework (MacKay, 1992a), 'hierarchical priors' are employed on the latent variables using Gamma priors on the inverse variances. The hyper-priors for σ^{-2}, λ and $\mathbf{s}$ are chosen to be a gamma distribution (Berger, 2013), *i.e.*, $p(\lambda) = \text{Gam}(\lambda \,|\, a^{\lambda}, b^{\lambda}), p(\beta) = \text{Gam}(\beta \,|\, a^{\beta}, b^{\beta})$ with $\beta = \sigma^{-2}$, and $p(s_{ij}^{o}) = \text{Gam}(s_{ij}^{o} \,|\, a^{s_{ij}^{o}}, b^{s_{ij}^{o}})$. Essentially, the choice of Gamma priors has the effect of making the marginal distribution of the latent variable prior the non-Gaussian StudentâĂŹs *t* therefore promoting the sparsity (Tipping, 2001, Section 2 and 5.1). To make these priors non-informative (*i.e.*, flat), we simply fix a and b to zero by assuming uniform scale priors for analysis and implementation. This formulation of prior distributions is a type of *hierarchically constructed automatic relevance determination* (HARD) prior which is built upon classic ARD prior (Neal, 1995; Tipping, 2001).

The posterior distribution for the parameters $\boldsymbol{\mathcal{W}}$, γ and λ can then be obtained by applying Bayes' rule:

$$\begin{aligned} & p(\boldsymbol{\mathcal{W}}, \mathbf{w}, \lambda, \mathbf{s}, \sigma^2 \,|\, \mathcal{D}) \\ = & \frac{p(\mathbf{Y} \,|\, \mathbf{X}, \boldsymbol{\mathcal{W}}, \mathbf{w}, \lambda, \mathbf{s}, \sigma^2) p(\boldsymbol{\mathcal{W}} \,|\, \lambda) p(\mathbf{w} \,|\, \mathbf{s}) p(\lambda) p(\gamma) p(\sigma^2)}{p(\mathbf{Y} \,|\, \mathbf{X})}, \end{aligned} \tag{12}$$

where $p(\mathbf{Y} \,|\, \mathbf{X})$ is a normalization constant. Given a new input vector $\mathbf{x}_\star$, we can make predictions for its output $\mathbf{y}_\star$ using the predictive distribution given by

$$\begin{aligned} & p(\mathbf{y}_\star \,|\, \mathbf{x}_\star, \mathcal{D}) \\ = & \int p(\mathbf{y}_\star | \mathbf{x}_\star, \boldsymbol{\mathcal{W}}, \mathbf{w}, \lambda, \mathbf{s}, \sigma^2) p(\boldsymbol{\mathcal{W}}, \mathbf{w}, \lambda, \mathbf{s}, \sigma^2 \,|\, \mathcal{D}) \\ & d\sigma^2 \, d\lambda \, d\mathbf{s} \, d\boldsymbol{\mathcal{W}} d\mathbf{w}, \end{aligned} \tag{13}$$

where $p(\mathbf{y}_\star | \mathbf{x}_\star, \boldsymbol{\mathcal{W}}, \mathbf{w}, \lambda, \mathbf{s}, \sigma^2) = \mathcal{N}(\mathbf{y}_\star \,|\, \text{Net}(\mathbf{x}_\star), \sigma^2)$. However, the exact computation of $p(\boldsymbol{\mathcal{W}}, \mathbf{w}, \lambda, \mathbf{s}, \sigma^2 \,|\, \mathcal{D})$ and $p(\mathbf{y}_\star \,|\, \mathbf{x}_\star)$ is not tractable in most cases. Therefore, in practice, we have to resort to approximate inference methods.

It should be noted that λ is the same for all network parameters. However, it can be different for $\boldsymbol{\mathcal{W}}$ or constructed to represent the structural sparsity for Convolutional kernels in NN aiming for Network Compression, which is related to Bayesian compression (Louizos et al., 2017) and structural sparsity compression (Wen et al., 2016). We give some examples in Figure 2 and more can be found in the Appendix B.2 where extremely sparse networks on MNIST and CIFAR-10 can be obtained without accuracy deterioration. Since our main focus is on architecture parameters, without breaking the flow, we will fix λ which is equivalent to the weight decay coefficient in SGD and $\sigma^2 = 0.01$ that is equivalent to the regularization coefficient for network parameters.

In case of uniform hyperpriors, we only need to maximize the term $p(\mathbf{Y} \,|\, \lambda, \mathbf{s}, \sigma^2)$ (MacKay, 1992a; Berger, 2013)

$$\int \int p(\mathbf{Y} \,|\, \boldsymbol{\mathcal{W}}, \mathbf{w}, \mathbf{X}, \sigma^2) p(\boldsymbol{\mathcal{W}} \,|\, \lambda) p(\mathbf{w} \,|\, \mathbf{s}) d\boldsymbol{\mathcal{W}} d\mathbf{w}. \tag{14}$$

We assume that the distribution of data likelihood belongs to the exponential family

$$p(\mathbf{Y} \,|\, \boldsymbol{\mathcal{W}}, \mathbf{w}, \mathbf{X}, \sigma^2) \sim \exp\left(-E_D(\mathbf{Y}; \text{Net}(\mathbf{X}; \boldsymbol{\mathcal{W}}, \mathbf{w}); \sigma^2) \right) \tag{15}$$

where $E_D(*)$ is the *energy function* over data.

5.2. Laplace Approximation and Efficient Hessian Computation

In related Bayesian models, the quantity in equation 14 is known as the marginal likelihood and its maximization is known as the type-II maximum likelihood method

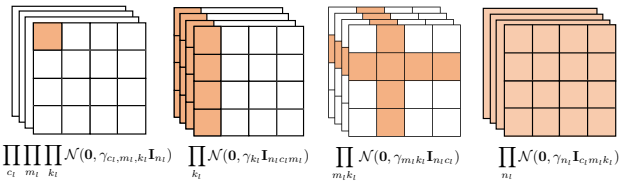

Figure 2. Structural Sparsity

(Berger, 2013). And neural networks can also be treated in a Bayesian manner known as Bayesian learning for neural networks (MacKay, 1992b; Neal, 1995). Several approaches have been proposed based on, *e.g.*, the Laplace approximation (MacKay, 1992b), Hamiltonian Monte Carlo (Neal, 1995), expectation propagation (Jylänki et al., 2014; Hernández-Lobato & Adams, 2015), and variational inference (Hinton & Van Camp, 1993; Graves, 2011). Among these methods, we adopt Laplace approximation. However, Laplace approximation requires computation of the inverse Hessian of log-likelihood, which can be infeasible to compute for large networks. Nevertheless, we are motivated by 1) its easy implementation, especially using recent popular deep learning open source software; 2) versatility for modern NN structures such as CNN and RNN as well as their modern variations; 3) close relationship between computation of Hessian and Network Compression using Hessian metric (LeCun et al., 1990; Hassibi et al., 1993); 4) acceleration effect to training convergence by second-order optimization algorithm (Botev et al., 2017) to which it is related. In this paper, we propose the efficient calculation/approximation of Hessian for convolutional layer and architecture parameter. The detailed calculation procedures are explained in Appendix C.2 and C.3 respectively.

5.3. Optimization Algorithm

As analyzed before, the optimization objective of searching architecture becomes removing redundant edges. The training algorithm is iteratively indexed by t. Each iteration may contain several epochs. The pseudo code is summarized in Algorithm 1. The cost function is simply maximum likelihood over the data D with regularization whose intensity is controlled by the re-weighted coefficient ω

$$\mathcal{L}_D = E_D(\cdot) + \lambda_w \sum_{j<k} \sum_{o' \in \mathcal{O}} \|\omega_{jk}^{o'}(t) w_{jk}^{o'}\|_1 + \lambda \|\boldsymbol{\mathcal{W}}\|_2^2 \quad (16)$$

The derivation can be found in Appendix A.1 and A.2. The algorithm mainly includes five parts. The first part is to jointly train $\mathcal{W}$ and $\mathbf{w}$. The second part is to freeze the architecture parameters and prepare to compute their Hessian. The third part is to update the variables associated with the architecture parameters. The fourth part is to prune the architecture parameters and the pruned net will be trained in a standard way in the fifth part. As discussed previously on the drawback of magnitude based pruning metric, we propose a new metric based on maximum entropy of the distribution. Since $p(w_{jk}^{o'})$ in equation 5 is Gaussian with zero mean $\gamma_{jk}^{o'}$ variance, the maximum entropy is $\frac{1}{2}\ln(2\pi e \gamma_{jk}^{o'})$. We set the threshold for $\gamma_{jk}^{o'}$ to prune related edges when $\frac{1}{2}\ln(2\pi e \gamma_{jk}^{o'}) \leq 0$, *i.e.*, $\gamma_{jk}^{o'} \leq 0.0585$.

Algorithm 1 BayesNAS Algorithm.

Initialization: $\boldsymbol{\gamma}(0), \boldsymbol{\omega}(0), \mathbf{w}(0) = \mathbf{1}$; $\lambda = 0.01$; sparsity intensity $\lambda_w^o \in \mathbb{R}^+$

Iteration:

for $t = 1$ to $T_{\max}$ **do**

1. Update $\mathbf{w}$ and $\mathcal{W}$ by minimizing $\mathcal{L}_D$ in equation 16
2. Compute Hessian for $\mathbf{w}$ (equation C.2.2, C.3.1, C.3.2)
3. Update variables associated with $\mathbf{w}$

while $i < j < k, o, o' \in \mathcal{O}$ **do**

$$C_{jk}^{o'}(t) = \left(\frac{1}{\gamma_{jk}^{o'}(t-1)} + \mathbf{H}_{jk}^{o'}(t)\right)^{-1} \quad (17)$$

$$\omega_{jk}^{o'}(t) = \frac{\sqrt{\gamma_{jk}^{o'}(t-1) - \mathrm{C}_{jk}^{o'}(t)}}{\gamma_{jk}^{o'}(t-1)} \quad (18)$$

$$s_{jk}^{o'}(t) = \left|\frac{w_{jk}^{o'}(t)}{\omega_{jk}^{o'}(t)}\right| \quad (19)$$

$$\gamma_{jk}^{o'}(t) \text{ is given by 6 or 8} \quad (20)$$

end while

4. Prune the architecture if the entropy $\frac{\ln(2\pi e \gamma_{jk}^{o'})}{2} \leq 0$
5. Fix $\mathbf{w} = \mathbf{1}$, train the pruned net in the standard way

end for

The algorithm can be easily transferred to other scenarios. One scenario involves proxy tasks to find the cell. Similar to equation 16, we group same edge/operation in the repeated stacked cells where g is the index. The cost function for proxy tasks is then given as follows in the form of re-weighted group Lasso:

$$\mathcal{L}_D = E_D(\cdot) + \lambda_w \sum_g \sum_{j<k} \sum_{o' \in \mathcal{O}} \|\omega_{jk,g}^{o'}(t) w_{jk,g}^{o'}\|_2 + \lambda \|\mathcal{W}\|_2^2 \quad (21)$$

The details are summarized in Algorithm 2 of Appendix A.3. Another scenario is on Network Compression with structural sparsity, which is summarized in Algorithm 3 of Appendix B.

6. Experiments

The experiments focus on two scenarios in NAS: proxy NAS and proxyless NAS. For proxy NAS, we follow the pipeline in DARTS (Liu et al., 2019b) and SNAS (Xie et al., 2019). First BayesNAS is applied to search for the best convolutional cells in a complete network on CIFAR-10. Then a network constructed by stacking learned cells is retrained

Table 1. Classification errors of BayesNAS and state-of-the-art image classifiers on CIFAR-10.

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	Search Method
DenseNet-BC (Huang et al., 2017)	3.46	25.6	-	manual
NASNet-A + cutout (Zoph et al., 2018)	2.65	3.3	1800	RL
AmoebaNet-B + cutout (Real et al., 2019)	2.55 ± 0.05	2.8	3150	evolution
Hierarchical Evo (Liu et al., 2018b)	3.75 ± 0.12	15.7	300	evolution
PNAS (Liu et al., 2018a)	3.41 ± 0.09	3.2	225	SMBO
ENAS + cutout (Pham et al., 2018)	2.89	4.6	0.5	RL
Random search baseline + cutout (Liu et al., 2019b)	3.29 ± 0.15	3.2	1	random
DARTS (2nd order bi-level) + cutout (Liu et al., 2019b)	2.76 ± 0.09	3.4	1	gradient
SNAS (single-level) + moderate con + cutout (Xie et al., 2019)	2.85 ± 0.02	2.8	1.5	gradient
DSO-NAS-share+cutout (Zhang et al., 2019b)	2.84 ± 0.07	3.0	1	gradient
Proxyless-G + cutout (Cai et al., 2019)	2.08	5.7	-	gradient
BayesNAS + cutout + $\lambda_w^o = 0.01$	3.02 ± 0.04	2.59 ± 0.23	0.2	gradient
BayesNAS + cutout + $\lambda_w^o = 0.007$	2.90 ± 0.05	3.10 ± 0.15	0.2	gradient
BayesNAS + cutout + $\lambda_w^o = 0.005$	2.81 ± 0.04	3.40 ± 0.62	0.2	gradient
BayesNAS + TreeCell-A + Pyrimaid backbone + cutout	2.41	3.4	0.1	gradient

for performance comparison. For proxyless NAS, we follow the pipeline in ProxylessNAS (Cai et al., 2019). First, the tree-like cell from (Cai et al., 2018) with multiple paths is integrated into the PyramidNet (Han et al., 2017). Then we search for the optimal path(s) within each cell by BayesNAS. Finally, the network is reconstructed by retaining only the optimal path(s) and retrained on CIFAR-10 for performance comparison. Detailed experiments setting is in Appendix D.1.

6.1. Proxy Search

Motivation Unlike DARTS and SNAS that rely on validation accuracy during or after search, we use γ in BayesNAS as performance evaluation criterion which enables us to achieve it in an one-shot manner.

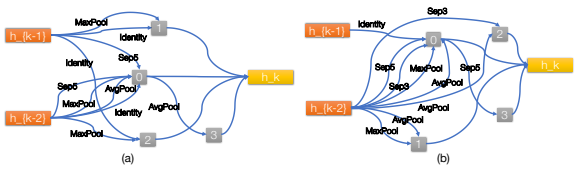

Figure 3. Normal and reduction cell found by BayesNAS with $\lambda_w^o = 0.01$.

Search Space Our setup follows DARTS and SNAS, where convolutional cells of 7 nodes are stacked for multiple times to form a network. The input nodes, *i.e.*, the first and second nodes, of cell k are set equal to the outputs of cell $k-1$ and cell $k-2$ respectively, with 1×1 convolutions inserted as necessary, and the output node is the depthwise concatenation of all the intermediate nodes. Reduction cells are located at the 1/3 and 2/3 of the total depth of the network to reduce the spatial resolution of feature maps. Details about all operations included are shown in Appendix D.1. Unlike DARTS and SNAS, we exclude *zero* operations.

Training Settings In the searching stage, we train a small network stacked by 8 cells using BayesNAS with different λ_w. This network size is determined to fit into a single GPU. Since we cache the feature maps in memory, we can only set batch size as 18. The optimizer we use is SGD optimizer with momentum 0.9 and fixed learning rate 0.1. Other training setups follow DARTS and SNAS (Appendix D.1). The search takes about 3 hours on a single GPU[1].

Search Results The normal and reduction cells learned on CIFAR-10 using BayesNAS are shown in Figure 3a and 3b. A large network of 20 cells where cells at 1/3 and 2/3 are reduction cells is trained from scratch with the batch size of 128. The validation accuracy is presented in Table 1. The test error rate of BayesNAS is competitive against state-of-the-art techniques and BayesNAS is able to find convolutional cells with fewer parameters when compared to DARTS and SNAS.

6.2. Proxyless Search

Motivation Using existing tree-like cell, we apply BayesNAS to search for the optimal path(s) within each cell. Varying from proxy search, cells do not share architecture in proxyless search.

[1]All the experiments were performed using NVIDIA TITAN V GPUs

Table 2. Comparison with state-of-the-art image classifiers on ImageNet in the mobile setting.

Architecture	**Test Error (%)**		**Params**	**Search Cost**	**Search Method**
	top-1	top-5	**(M)**	**(GPU days)**	
Inception-v1 (Szegedy et al., 2015)	30.2	10.1	6.6	–	manual
MobileNet (Howard et al., 2017)	29.4	10.5	4.2	–	manual
ShuffleNet 2× (v1) (Zhang et al., 2018)	29.1	10.2	∼5	–	manual
ShuffleNet 2× (v2) (Zhang et al., 2018)	26.3	–	∼5	–	manual
NASNet-A (Zoph et al., 2018)	26.0	8.4	5.3	1800	RL
NASNet-B (Zoph et al., 2018)	27.2	8.7	5.3	1800	RL
NASNet-C (Zoph et al., 2018)	27.5	9.0	4.9	1800	RL
AmoebaNet-A (Real et al., 2019)	25.5	8.0	5.1	3150	evolution
AmoebaNet-B (Real et al., 2019)	26.0	8.5	5.3	3150	evolution
AmoebaNet-C (Real et al., 2019)	24.3	7.6	6.4	3150	evolution
PNAS (Liu et al., 2018a)	25.8	8.1	5.1	∼225	SMBO
DARTS (Liu et al., 2019b)	26.9	9.0	4.9	4	gradient
BayesNAS ($\lambda_w^o = 0.01$)	28.1	9.4	4.0	0.2	gradient
BayesNAS ($\lambda_w^o = 0.007$)	27.3	8.4	3.3	0.2	gradient
BayesNAS ($\lambda_w^o = 0.005$)	26.5	8.9	3.9	0.2	gradient

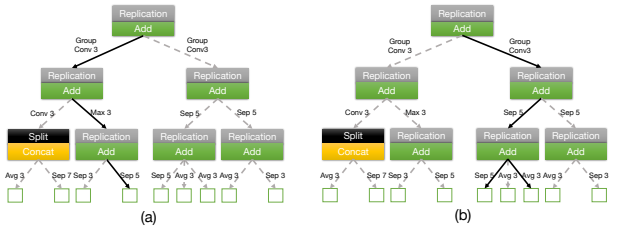

Figure 4. The pruned tree-cell: (a) The chain-like where only one path exists in the cell connecting the input of the cell to its output. (b) The inception structure where divergence and convergence both exist in the cell. The solid directed lines denote the path found by BayesNAS while the dashed ones denote the paths discarded.

Search Space The backbone used is PyramidNet with three layers each consisting of 18 bottleneck blocks and $\alpha = 84$. All 3×3 convolution in bottleneck blocks are replaced by the tree-cell that has in total 9 possible paths within. The groups for grouped convolution is set to 2. For the detailed structure of the tree-cell, we refer to (Cai et al., 2018).

Training Settings In the searching stage, we set batch size to 32 and learning rate to 0.1. We use the same optimizer as for proxy search. The λ of BayesNAS for each possible path is set to 1×10^{-2}.

Search Results Because each cell can have a different structure in proxyless setting, we demonstrate only two typical types of cell structure among all of them in Figure 4a and Figure 4b. The first type is a chain-like structure where only one path exists in the cell connecting the input of the cell to its output. The second type is an inception structure where divergence and convergence both exist in the cell. Our further observation reveals that some cells are dispensable with respect to the entire network. After the architecture is determined, the network is trained from scratch with the batch size of 64, learning rate as 0.1 and cosine annealing learning rate decay schedule (Loshchilov & Hutter, 2017). The validation accuracy is also presented in Table 1. Although test error increases slightly compared to (Cai et al., 2019), there is a significant drop in the number of model parameters to be learned which is beneficial for both training and inference.

7. Transferability to ImageNet

For ImageNet mobile setting, the input images are of size 224×224. A network of 14 cells is trained for 250 epochs with batch size 128, weight decay 3×10^{-5} and initial SGD learning rate 0.1 (decayed by a factor of 0.97 after each epoch). Results in Table 2 show that the cell learned on CIFAR-10 can be transfered to ImageNet and is capable of achieving competitive performance.

8. Conclusion and Future Work

We introduce BayesNAS that can directly learn a sparse neural network architecture. We significantly reduce the search time by using only one epoch to get the candidate architecture. Our current implementation is inefficient by caching all the feature maps in memory to compute the Hessian. However, Hessian computation can be done along with backpropagation which will potentially further reduce the searching time and scale our approach to larger search spaces.

Acknowledgements

The work of Hongpeng Zhou is sponsored by the program of China Scholarships Council (No.201706120017).

References

Amari, S.-I. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Baker, B., Gupta, O., Naik, N., and Raskar, R. Designing neural network architectures using reinforcement learning. *International Conference on Learning Representations*, 2017.

Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., and Le, Q. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pp. 549–558, 2018.

Berger, J. O. *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013.

Botev, A., Ritter, H., and Barber, D. Practical gauss-newton optimisation for deep learning. *ICML*, 2017.

Boyd, S. and Vandenberghe, L. *Convex optimisation*. Cambridge university press, 2004.

Brock, A., Lim, T., Ritchie, J., and Weston, N. SMASH: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*, 2018.

Cai, H., Yang, J., Zhang, W., Han, S., and Yu, Y. Path-level network transformation for efficient architecture search. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pp. 677–686. PMLR, 2018.

Cai, H., Zhu, L., and Han, S. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/pdf/1812.00332.pdf.

Candes, E. J., Wakin, M. B., and Boyd, S. P. Enhancing sparsity by reweighted ℓ_1 minimization. *Journal of Fourier analysis and applications*, 14(5-6):877–905, 2008.

Chauvin, Y. A back-propagation algorithm with optimal use of hidden units. In *Advances in neural information processing systems*, pp. 519–526, 1989.

Courbariaux, M., Bengio, Y., and David, J.-P. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pp. 3123–3131, 2015.

DeVries, T. and Taylor, G. W. Improved regularization of convolutional neural networks with cutout, 2017.

Dikov, G., van der Smagt, P., and Bayer, J. Bayesian learning of neural network architectures. *AISTATS*, 2019.

Elsken, T., Metzen, J. H., and Hutter, F. Efficient multi-objective neural architecture search via lamarckian evolution. In *International Conference on Learning Representations*, 2019a.

Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019b.

Gordon, A., Eban, E., Nachum, O., Chen, B., Wu, H., Yang, T.-J., and Choi, E. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1586–1595, 2018.

Graves, A. Practical variational inference for neural networks. In *Advances in neural information processing systems*, pp. 2348–2356, 2011.

Guo, Y., Yao, A., and Chen, Y. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pp. 1379–1387, 2016.

Han, D., Kim, J., and Kim, J. Deep pyramidal residual networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pp. 6307–6315. IEEE, 2017.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*, 2016.

Hassibi, B., Stork, D. G., and Wolff, G. J. Optimal brain surgeon and general network pruning. In *IEEE International Conference on Neural Networks*, pp. 293–299 vol.1, March 1993. doi: 10.1109/ICNN.1993.298572.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hernández-Lobato, J. M. and Adams, R. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pp. 1861–1869, 2015.

Hinton, G. E. and Van Camp, D. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pp. 5–13. ACM, 1993.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Ishikawa, M. Structural learning with forgetting. *Neural networks*, 9(3):509–521, 1996.

Jylänki, P., Nummenmaa, A., and Vehtari, A. Expectation propagation for neural networks with sparsity-promoting priors. *The Journal of Machine Learning Research*, 15 (1):1849–1901, 2014.

LeCun, Y. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Lee, N., Ajanthan, T., and Torr, P. H. S. Snip: Single-shot network pruning based on connection sensitivity. 2019.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 19–34, 2018a.

Liu, C., Chen, L.-C., Schroff, F., Adam, H., Hua, W., Yuille, A., and Fei-Fei, L. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation, 2019a.

Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*, 2018b. URL https://openreview.net/forum?id=BJQRKzbA-.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019b. URL https://openreview.net/forum?id=S1eYHoC5FX.

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

Louizos, C., Ullrich, K., and Welling, M. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pp. 3288–3298, 2017.

Louizos, C., Welling, M., and Kingma, D. P. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1Y8hhg0b.

Ma, W. and Lu, J. An equivalence of fully connected layer and convolutional layer, 2017.

MacKay, D. J. Bayesian interpolation. *Neural computation*, 4(3):415–447, 1992a.

MacKay, D. J. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992b.

MacKay, D. J. Bayesian methods for backpropagation networks. In *Models of neural networks III*, pp. 211–254. Springer, 1996.

Martens, J. and Grosse, R. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417, 2015.

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pp. 293–312. Elsevier, 2019.

Molchanov, D., Ashukha, A., and Vetrov, D. Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 2498–2507. JMLR.org, 2017a. URL http://dl.acm.org/citation.cfm?id=3305890.3305939.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations*, 2017b.

Neal, R. M. Bayesian learning for neural networks. 1995.

Neklyudov, K., Molchanov, D., Ashukha, A., and Vetrov, D. P. Structured bayesian pruning via log-normal multiplicative noise. In *Advances in Neural Information Processing Systems*, pp. 6775–6784, 2017.

Nocedal, J. and Wright, S. J. *Numerical Optimization*. Springer, 2006.

Pan, W. *Bayesian learning for nonlinear system identification*. PhD thesis, Imperial College London, 2017.

Pham, H. Q., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. In *ICML*, 2018.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2902–2911. JMLR. org, 2017.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *AAAI*, 2019.

Saxena, S. and Verbeek, J. Convolutional neural fabrics. In *Advances in Neural Information Processing Systems*, pp. 4053–4061, 2016.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Tipping, M. E. Sparse bayesian learning and the relevance vector machine. *Journal of machine learning research*, 1 (Jun):211–244, 2001.

Ullrich, K., Meeds, E., and Welling, M. Soft weight-sharing for neural network compression. In *International Conference on Learning Representations*, 2017.

Weigend, A. S., Rumelhart, D. E., and Huberman, B. A. Generalization by weight-elimination with application to forecasting. In *Advances in neural information processing systems*, pp. 875–882, 1991.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 2074–2082, 2016.

Xie, L. and Yuille, A. Genetic cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 1388–1397. IEEE, 2017.

Xie, S., Zheng, H., Liu, C., and Lin, L. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rylqooRqK7.

Zhang, C., Ren, M., and Urtasun, R. Graph hypernetworks for neural architecture search. In *International Conference on Learning Representations*, 2019a. URL https://openreview.net/forum?id=rkgW0oA9FX.

Zhang, X., Zhou, X., Lin, M., and Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6848–6856, 2018.

Zhang, X., Huang, Z., and Wang, N. Single shot neural architecture search via direct sparse optimization, 2019b. URL https://openreview.net/forum?id=ryxjH3R5KQ.

Zhong, Z., Yan, J., Wu, W., Shao, J., and Liu, C.-L. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2423–2432, 2018.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8697–8710, 2018.