Efficient Neural Network Architecture Search

Minghao Yang 4742702

Master of Science Thesis

TU Delft Delft University of Technology

Cognitive Robotics

Efficient Neural Network Architecture Search

Master of Science Thesis

For the degree of Master of Science in Vehicle Engineering at Delft University of Technology

Minghao Yang 4742702

July 5, 2019

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Delft University of Technology
Department of
Cognitive Robotics (CoR)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Efficient Neural Network
Architecture Search

by

Minghao Yang 4742702

in partial fulfillment of the requirements for the degree of

Master of Science Vehicle Engineering

Dated: July 5, 2019

Supervisor(s):

Dr. Wei Pan

Ir. H. Zhou

Reader(s):

Prof. dr. D.M. Gavrila

Dr. Wei Pan

Dr.ing. R. Van de Plas

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 overparameterized 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 thesis, classic Bayesian learning approach is applied to alleviate these two issues. Unlike other NAS methods, we train the over-parameterized network for only one epoch before update network architecture. Impressively, this enabled us to find the optimal architecture in both proxy and proxyless tasks on CIFAR-10 within only 0.2 GPU days using a single GPU. As a byproduct, our approach can be transferred directly to convolutional neural networks compression by enforcing structural sparsity that is able to achieve extremely sparse networks without accuracy deterioration.

Table of Contents

List of Figures

List of Tables

Preface

We came up with the idea of this project after I joined my supervisor Dr. Wei Pan's team and provided assistance with respect to engineering and experiments. After a careful and thorough discussion with my daily supervisor Hongpeng and Dr. Wei Pan, we decided that we should not constrain ourselves to a master thesis only but release the potential of this idea and fight for a top computer science conference. Luckily we made it and now my master thesis is not just about my graduation project but also a part of our published ICML paper.

Acknowledgements

I would like to thank my supervisor Prof. dr. D.M. Gavrila for his assistance and guidance during the writing of this thesis. I also want to appreciate my daily supervisor Hongpeng who was very helpful and had been working very closely with me as we had a great time dealing with challenges together.

Delft, University of Technology
July 5, 2019

Minghao Yang 4742702

"As you will it, so it shall be."
— *Dave Ramsey*

Chapter 1

Introduction

1-1 Background

Deep Neural Networks (DNN), especially Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), has drawn much attention since the publish of AlexNet[3]. Being different from many other machine learning approaches which usually gain their strength mainly from extraordinary algorithms, Deep Neural Networks' power derives from their architecture and ample amount of available data. So far many powerful architectures have been found by human expertise, such as ResNet[4] and DenseNet[5] for CNN and LSTM[6] for RNN. However, designing these deep nets requires great effort from experienced experts. This raises two questions:

- Can we automate the network architecture designing procedure?
- Can we find a neural network architecture by automation that can trade-off between performance and hardware demands (in the sense of neural network complexity), or even better, achieve both aspects simultaneously?

1-2 Deep Neural Networks

Deep Neural Networks is a Deep Learning technique whose embedded operations separate it from conventional neural networks in Machine Learning. It uses a collection of multiple layers of nonlinear processing units for feature extraction and transformation. DNN can learn multiple levels of representation that correspond to different levels of abstraction. Hierarchy of concepts can be formed in terms of passing information from the previous layers to the successive layers[7].

For computer vision tasks, convolutional operations in a DNN play an essential role as it enables DNN to integrate information from the entire image with a relatively small kernel.

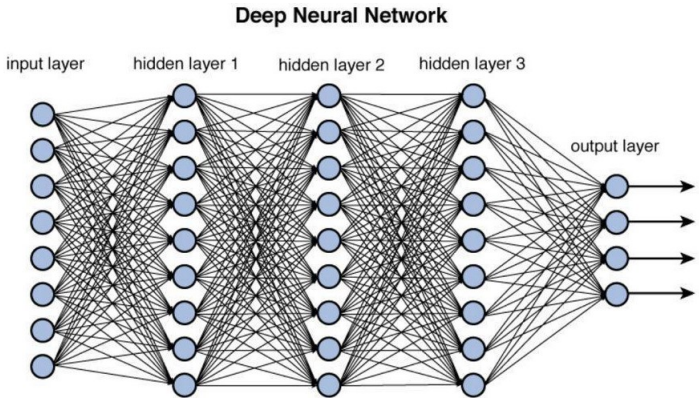

Figure 1-1: DNN with multiple layers.

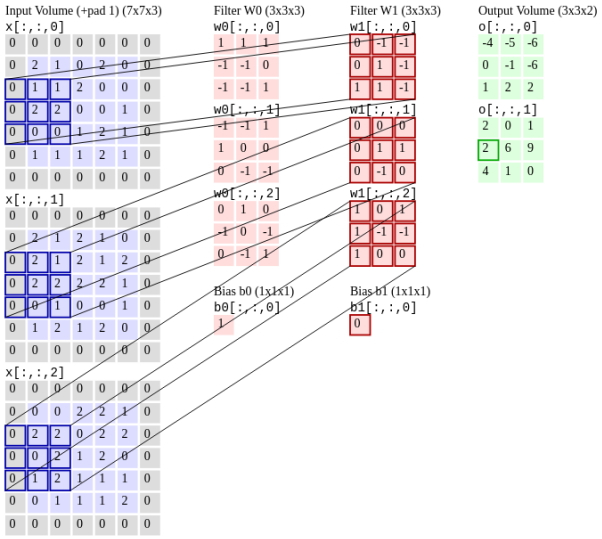

Figure 1-2: Convolutional operations within a DNN.

An example is shown in Figure 1-2.

Nonlinearity is achieved by nonlinear activation functions following linear operations, for instance convolution and matrix multiplication. Common activation functions used include softmax function, hyperbolic tangent function (Tanh) and rectified linear unit (ReLU).

1-3 Neural Architecture Search

Neural architecture search (NAS), whose name explains itself, denotes the process of designing neural networks in an automatic manner. NAS consists of three key components: search space, search strategy, and performance evaluation strategy. Search space defines what type of neural network can be constructed. Search strategy defines which method is applied to find the optimal architecture within the search space. Since the evaluation of architecture can be very costly as it requests a complete training process in principle, a performance evaluation

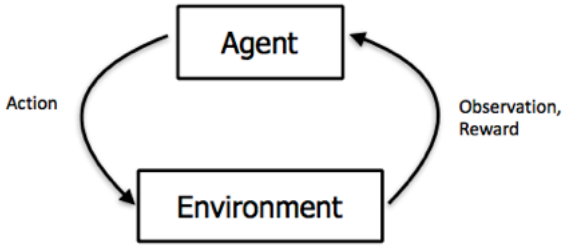

Figure 1-3: Reinforcement learning.

strategy needs to be determined to evaluate the candidates for optimal architecture within an acceptable period of time while retaining the correlation between the metric of this strategy and candidate's actual performance.

1-4 Reinforcement Learning

Reinforcement learning is a category of machine learning in which an agent learns to take actions in an environment so as to maximize expected cumulative reward. Reinforcement learning is different from both supervised learning and unsupervised learning in the sense that wrong or sub-optimal actions the agent takes do not need to be corrected and the agent learns to behave by the reward returned by the environment. Typically the environment in reinforcement learning is a Markov Decision Process (MDP) and no assumption about the underlying mathematical model of MDP is what distinguishes reinforcement learning from other MDPs.

1-5 Evolutionary Algorithms

Evolutionary algorithm are a family of algorithms that uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Each Candidate solutions to the optimization problem play the role of an individual in a population, and a fitness function provided by the environment determines the quality of this candidate. Evolution of the population takes place in the form of repeated application of combinations of the above operators.

General scheme of evolutionary algorithms includes three steps:

- Generate the first generation of the population (an initial group of candidates).
- Repeat:
  - Evaluate their fitness given the fitness function.
  - Keep only the individuals of the best fitness and generate the next generation by applying operators to them.

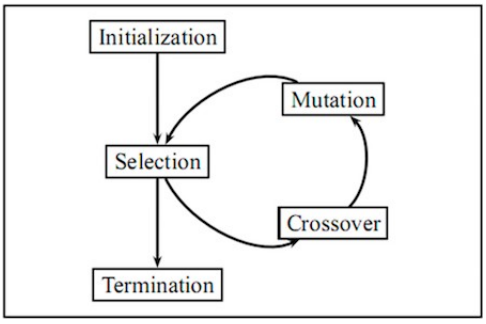

Figure 1-4: General scheme of evolutionary algorithms.

1-6 Bayesian Inference

Bayesian inference is a type of statistical inference in which Bayes' theorem is used to update the probability for a hypothesis as more evidence or information becomes available. It derives the posterior probability from a prior probability, a sampling probability and a marginal likelihood probability provided by a statistical model for the observed data. Bayesian inference computes the posterior probability according to Bayes' theorem:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)} \tag{1-1}$$

- H: Hypothesis to be validated
- E: Evidence
- $P(H)$: Prior probability
- $P(E|H)$: Sampling probability
- $P(E)$: Marginal likelihood probability

Chapter 2

Related Works

2-1 Search Space Design

Search space design is the very first step of formulating a neural network architecture searching problem. Not only the size of search space has a strong influence on the potential time cost for optimization but the coverage of search space implicitly control how optimal the found final solution can be. There are mainly two groups of methods for designing a search space. One is searching an architecture from scratch and the other searching with a start point based on an existing or given structure. Methods of both of these two groups then start searching within a confined search space.

2-1-1 Searching from Scratch

[8] proposed a stage-wise search space design based in combination with heuristics. The author fixed the number of stages and the number of nodes in each stage separately. Each node corresponds to one convolutional operation with a fixed kernel size and number of channels. The nodes within a stage are sorted and each node can decide from which node(s) whose sorting index is smaller than itself to receive information. And essentially these shortcut connection's formulation is all that the algorithm searched for. [9] as an earlier paper, adopted a very raw and brutal search space. In this paper, the author also fixed the number of convolutional layers for a fixed number of searching steps but it can grow as the searching steps keep increasing. However, no concept similar to that of "stage" was introduced. In other words, every convolutional layer can choose whatever previous layers to connect to. Furthermore, the kernel size was not strictly defined. Instead, it was sampled from a predefined pool of convolution operations which the author chose based on the knowledge of previous human-crafted structures. A simple comparison between the descriptions of these two design strategies indicates that the search space in [9] is much larger than that in [8] provided that the fixed total numbers of convolutional layers in these two designs are the same. The time cost and result of experiments conducted by the authors of these two papers validate the intuitive predictions mentioned at the beginning of this chapter. In [8], the search for an optimal structure on

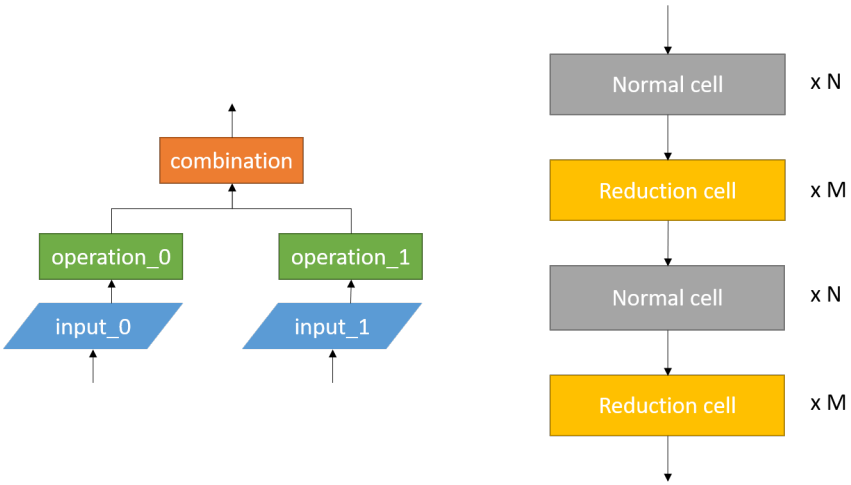

Figure 2-1: block structure and the building of the complete architecture in [1]. Left: Each block is indexed and receives two input. It applies two operations to these two inputs respectively before combining them using either element-wise addition or concatenation along channel dimension (later the combination method is fixed to element-wise addition by the author). If the resulting dimension of the output of these two operations is different the smaller one will be padded to the same size as the larger one. The two inputs can be chosen from the current cell's input, previous cell's input and the outputs of all previous blocks within the same cell based on the indices. Right: The cells are stacked to form the complete neural networks architecture.

CIFAR10[10] dataset took 17 GPU hours using a Nvidia Titan X while in [9] it took more than 4×10^5 GPU hours (though GPU model information was not mentioned). What comes with the much greater time cost measured in GPU hours is the better performance of the optimal architecture found by the algorithm: 22.94% in [8] and 5.50% in [9]measured in the form of error rate. It has to be stressed that the setting of the experiment conducted in [9] is to some degree deviant from a practical perspective. Such a great demand for computational power can rarely be satisfied. Thus its validity is questionable.

[1] introduced a novel design of search space. It adopted a cell-wise idea. In the paper's setting, there are two types of cells: normal cells and reduction cells. Each cell consists of a fixed number of blocks (in this article the author found that 5 blocks for each cell are optimal by manual trial). Each block is indexed and receives two input. It applies two operations to these two inputs respectively before combining them using either element-wise addition or concatenation along channel dimension. If the resulting dimension of the output of these two operations is different the smaller one will be padded to the same size as the larger one. The two inputs can be chosen from the current cell's input, previous cell's input and the outputs of all previous blocks within the same cell based on the indices. The structure of a normal cell and a reduction cell can be different. After the structure of these two cells are defined, the cells are stacked following a $N + M$ rule: N normal cells followed by M reduction cells and the process is repeated several times. The value of N and M, as well as the repeating times, as additional hyperparameters, are dependent on the characteristics of the dataset to evaluate on and is determined by human experts. Visualization of this search space design is shown in Figure 2-1.

The error rate obtained in [1] reached 2.40% and the computation cost is around 4.8×10^4 GPU hours (still GPU model information is not mentioned). The significant decrease of error rate and GPU hours for searching when compared with [9], based on my analysis, is caused by the heuristics provided by human experts. In [1], the frame of the entire neural network architecture is fixed and designed by human experts and the searching agent only needs to determine the two operations on the two inputs and within short connections for 5 blocks of two cells respectively. But in [9], the agent needs to determine almost everything essential for neural network architecture. Besides this there is no difference between the algorithms applied in [1] and [9]. The contrast is already sharp enough to suggest that heuristics from human expert plays an extremely important role in neural network architecture search, both in computation consumption and the final error rate of the found structure.

Many more recent papers like [11] and [12] adopted the same search space design strategy as in [1] and the similar low error rate of their found architecture further validates the importance of search space design and human heuristics.

2-1-2 Searching Based on Existing Architectures

Some papers restrict their search space based on existing human-crafted architectures that have been proved to be a success. In [8], for the searching on MNIST dataset [13], the search space is strictly restricted to be very close to LeNet 5 [14]: two stages, exactly the same first convolutional layer in each stage as in LeNet 5, only two possible convolutional kernel size 3 and 5 with the exactly same number of channels. The baseline accuracy of LeNet 5 is 99.34% and the maximum accuracy found in [8] is 99.66%. The accuracy gap is not very significant. One cause can be the similarity to the human-designed architecture and the slight improvement is possibly the result of its search space's flexibility with respect to the existing human-design architecture.

Another type of existing architecture based search space design is the adoption of the essential idea behind their success. For example, the block-wise structure is a characteristic feature of ResNet in [4]. [15] adopted the block-wise design in order to achieve not only high performance but also powerful generalization ability to different tasks and datasets. The error rate of the best architecture found in [15] is 3.54% showing the effectiveness of this approach.

2-1-3 Summary

How much human heuristics involved is the root of search space design regardless of whether it is finished by searching from scratch or based on existing architectures. Human heuristics play an important role in influencing not only the accuracy but also the searching time. The more heuristics are introduced in search space design, generally the higher accuracy the final found architecture can achieve. The search space can influence the upper bound of the accuracy that a search agent can obtain.

But introducing heuristics blindly is not wise. The critical idea of automated machine learning is to get rid of the dependency on human experts when solving a problem using machine learning. Introducing excessive heuristics may raise the question of whether a whole search

algorithm is actually "automated" and make it less practical in real production.

From another perspective, heuristics are necessary for the effectiveness of a search algorithm. Without any heuristics from human experts, an agent can learn only from abundant "trial and errors". For other conventional interaction between the environment and the agent, it is practical since each interaction and sampling costs very little time. But for neural network architecture search, the troublesome problems can appear as the cost of sampling is extremely high: each sampling equals to training a new neural network. Moreover, training time is usually positively related to a network architecture's potential. Without being overtrained, a deeper network has the stronger expressive ability but also more parameters to learn thus causing more time to train. Proper heuristic from human experts helps reduce the search space meanwhile maintain the existence of possible strong neural network architecture in the search space. Based on the analysis, confining search space to one that is to some degree similar to existing human-crafted architecture in the sense of maintaining its structural characteristic that grants its good performance is a wise choice to balance practicality and automation level.

2-2 Search Strategy

2-2-1 Search by Reinforcement Learning

[16, 12, 17, 18, 11, 19, 20, 21] all use reinforcement learning, particularly, policy-based reinforcement learning as the search algorithm to find an optimal architecture. The main target of policy-based reinforcement learning is to learn a direct mapping from a state to an accordingly optimal action. This mapping should be first parameterized to be able to be fit into a learning process-usually gradient ascent which aims at maximizing the expected overall reward. For the general case of reinforcement learning where the optimal actions given the current state is deterministic, the mathematical expression of this process is shown by the following equations.

$$V(s_0;\theta) = P(a) \tag{2-1}$$

$$a_0 = argmax_a P(a) \tag{2-2}$$

$$s_0, a_0 \rightarrow r \tag{2-3}$$

$$\mathcal{L} = f(P(a_0), r) \tag{2-4}$$

$$\theta' = \theta + \alpha \frac{d\mathcal{L}}{d\theta} \tag{2-5}$$

Here s_0 and a_0 stand for the current state the optimal action taken based on the current state. V and P stand for the policy function (state value function) and probability distribution function of all the available actions at the current state respectively. L stands for the loss calculated based on the probability of taking this action and the corresponding reward. θ stands for the parameters of the parameterized mapping from state to the optimal action. Just like gradient descent which is the foundation of the success of deep neural networks, the gradient of the loss with respect to the parameters of mapping is used to update these parameters, but in positive relation to the gradient.

Policy-based reinforcement learning has several merits. First, it learns a direct mapping from state to the optimal action which enables faster inference during application. Second, it is a gradient-based method that can fit seamlessly into supervised learning scenarios and is equally suitable for both discrete and continuous state expression.

For its application in neural network architecture search, the general scheme remains almost unchanged among the literature that adopted this method. An agent is chosen to be an LSTM of several layers and is trained with reinforcement learning. For single-objective searching problems, as in most research papers, the loss of the training process is defined as the product of cross-entropy of the probability (output of LSTM at a step) of choosing a particular action and the corresponding reward. For multiple-objective searching problems, as [22] in which the goal is to archive both higher accuracy and lower power consumption, the cross-entropy part remains the same and only the form of reward is modified. Still taking [22] as an example, the author provided three different forms of reward, namely mixed reward, energy constraint, and accuracy constraint, for different application scenarios: unlimited accuracy or power supply, limited power supply and minimum accuracy requirement. Here Energy stands for the normalized power consumption.

$$R = \alpha * Accuracy - (1 - \alpha) * Energy, \alpha \in [0, 1] \tag{2-6}$$

$$R = \begin{cases} Accuracy, & Energy < threshold \\ 0, & Energy \geq threshold \end{cases} \tag{2-7}$$

$$R = \begin{cases} 1 - Energy, & Accuracy > threshold \\ 0, & Accuracy \leq threshold \end{cases} \tag{2-8}$$

Regardless of whether it is single-objective oriented or multiple-objective oriented, to reduce the variance of gradient estimation, the reward is commonly replaced by the difference between the actual reward and a baseline. One choice for this baseline can be an exponential moving average of the reward of several previous steps[9].

The reason for this loss function is that the cross-entropy is negatively related to the likelihood of choosing the action and getting the corresponding reward, which in another word can be interpreted as negatively related to the expectation of getting the reward. In [9], the step for LSTM increases as the search progressing and so does the depth of neural network architecture search space. For each convolution layer, it takes six steps to determine: anchor point, which defines the short connections to previous layers, and the rest five for filter height, filter width, stride height, stride width and number of filters respectively. At each step, the agent outputs a vector representing the probability of choosing an action from the predefined action pool. Then a neural network model is built based on these decisions and evaluated to calculate the reward for LSTM's training.

For cell-wise search spaces, like [1, 12, 11], most of the agent design remains the same except for the steps which will not increase as training proceeds and the number of agents since there are two different types of cells to be determined.

As contradictory to the prevalence of policy-based reinforcement learning, value-based reinforcement learning is completely ignored by the researchers. Value-based reinforcement

learning aims at finding the optimal action at a certain state based on the Q value learned from interaction with the environment. Q value can be interpreted as a quantified measurement of the expected overall reward of taking an action at this certain state.

By a detailed comparison between a policy-based and value-based approach, the value-based approach's several features are revealed. The first is being easily understandable. The value-based can provide quantified and intuitive evaluation for actions at different states. Second is being more suitable for problems with both discrete states and actions. This does not mean it cannot be applied to problems having continuous states and/or actions. For example, a radial basis function can be used as a tool to achieve continuity with a discrete expression. The third is being particularly suitable for complicated problems. One drawback of the policy-based method is the requirement for some basic knowledge about the mapping function if a good result is desired. For neural networks that mimic the mapping, it is the hyper-parameters of these neural networks, for instance, architecture and learning rate. For mapping functions in the conventional sense, it is the choice of which form or what parameters a parameterized mapping function should have.

The last two traits of value-based reinforcement learning cater to the setting of neural network architecture search. If we simply focus on searching for network architecture and ignore other continuous hyperparameters such as learning rate, the problem becomes searching in a discrete state and action space. Furthermore, neural network architecture search is much of a black box problem. By mentioning the black box, it means that we usually have no prior knowledge to provide a good heuristic for searching and this can potentially jeopardize policy-based methods whose performance partially depends on the chosen parameterized expression.
Besides the value-based method, a special variation of the policy-based method named Actor-Critic[23] is also overlooked by the research field. The basic structure of Actor-Critic is demonstrated in Figure 2-2. The essential innovation of Actor-Critic is to divide the whole learning and interaction with the environment into two parts. The actor decides which action to choose given the state and the critic evaluates the state-action pair and assigns a value indicating the expected overall reward for taking this action at this state.

Compared to the traditional policy-based method, Actor-critic has several merits. First, it tends to find a global maximum instead of converging to a local maximum. Second, it helps reduce the variance of finding the optimal policy. Third, most importantly, it can improve sample efficiency by its nature of splitting the sampling and evaluation into two parts[23]. Also because the process is divided, one can apply different methods for actor and critic separately or different form of parameterization. Specifically, for actor-critic designed to be off-policy, it can break up the correlation of recent states and policies and prevents the network from getting stuck to a certain behaviour mode and makes a technique named "experience replay", which makes effective use of experience from past exploration and helps accelerate convergence of the reinforcement learning algorithms as well as reducing required sampling times, possible[24].

In 2016, [25] from Deepmind advanced a huge step on the improvement of actor-critic algorithms as Deep Deterministic Policy Gradient (DDPG) came to view. It is an off-policy algorithm and can be trained with samples from a replay buffer to minimize the correla-

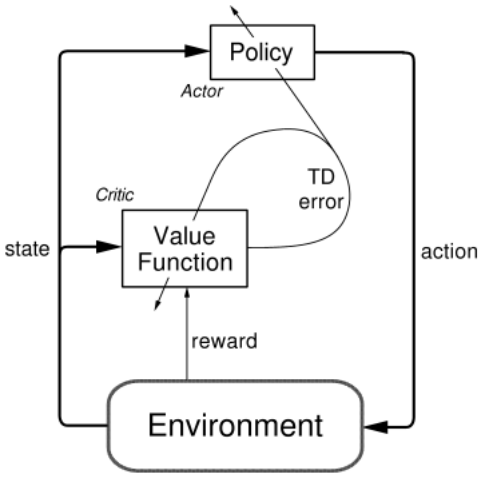

Figure 2-2: basic structure of Actor-Critic

tion between past experience. The most important contribution of this research work is the proposal of adding two separate networks alongside the original actor and critic networks provided neural networks are used to approximate the mapping and evaluation functions. The network is trained with a target critic and target actor-network to give consistent training targets during temporal difference backups[25]. From adaptability point of view, DDPG is applicable under continuous settings.

$$L(\theta^Q) = E_{s_t\sim\rho^\beta, a_t\sim\beta, r_t\sim E}[(Q(s_t, a_t|\theta^Q) - y_t)^2] \tag{2-9}$$

The loss derived based on Bellman Equation, is defined as Eq.2-9, where $y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q)$. μ represents the target policy which is deterministic and greedy and β represents the current policy. ρ is the distribution of states given a policy. The expression of loss suggests that the critic network, whose parameters are annotated as θ^Q, will be updated by TD error as in many other Q learning algorithms. Accordingly, the derivative of loss with respect to actor and critic network parameters is

$$\begin{aligned}\nabla_{\theta^\mu} J &\approx E_{s_t\sim\rho^\beta}[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}] \\ &= E_{s_t\sim\rho^\beta}[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_t}]\end{aligned} \tag{2-10}$$

If the network representing the current policy is also used as the target network, the update of Q values is likely to diverge. This the reason why two additional networks are added and are updated using a soft update rule: the update of policy network is also applied on target network but with a significant shrink (multiplied by a ratio far smaller than 1). The delayed update of target network forces the whole learning to "slow down" and stabilizes target Q value in TD error measurement in order to consistently train the critic without divergence[25].

In this paper, the author also adopted two techniques regarding the exploration of the agent and scaling state representations. For exploration, the strategy is as simple as adding noise the action generated by actor-network. For state representation scaling, a famous and very practical technique in deep learning called Batch Normalization[26] is used to automatically

re-scale information automatically without explicitly and manually re-scaling. In the paper, the author pointed out that noise added to action should be customized to be suitable for the environment.

Though DDPG has been proved to be powerful in continuous settings, exploration on its use under discrete settings is empty. DDPG, which roots in Q-learning just as value-based Q-learning, is worth research when it comes to neural network architecture search problems.

2-2-2 Search by Evolutionary Algorithm

The essential idea of the evolutionary algorithm, which sometimes is also called genetic search, is "selection by the environment". Just like nature, each sample to be evaluated is assigned a fitness based on how much it fits the problem's target and only samples with good fitness will have the next generation. However, the next generation will appear in a non-deterministic manner in the sense of the existence of mutation and crossover. On the contrary samples with very poor fitness will be eliminated and no longer being part of the competition.

[8] used a string with a fixed length to represent network architecture. First, the algorithm sampled several random architectures (fixed-length strings). Each bit in each individual architecture is independently sampled from a Bernoulli distribution. These architectures were then trained from scratch and evaluated to get their fitness which is used to determine a non-uniform selection on which architectures shall survive and provide the next generation by selection, mutation, and crossover. Then this evaluation-selection-generation process will repeat itself for a predefined number of times.

[27] also applied a genetic algorithm as a search strategy but used a completely different design of search space. Unlike in [8] where the author directly searched for a fixed length string representing the optimal network architecture, the researcher in [27] tried to find an optimal path in a directed graph. The directed graph consists of two parts: nodes and edges. A node here, except a "source node" and a "sink node", represents one of summation, activation or pooling operation which will be applied to the output of its predecessor nodes. "Source node" is the inlet of information and "sink node" is the output of the network.

The author of [27] defined five possible mutations related to edge and node modifications such as node creation, linking and edge creation and pruning. Later the rank of fitness (performance of sampled network architecture) is used to determine the probability distribution[28] of sampling on the current population and the sampled (survived) candidates will go through random mutation to reproduce their offspring. The author adopted Boltzmann distribution as the probability distribution model which has the form

$$p(k) = \frac{(1 - e^{\lambda})e^{-\lambda k}}{1 - e^{\lambda N}} \tag{2-11}$$

Here k stands for the rank, N stands for the amount of population and λ stands for the shape parameter that balances exploitation and exploration.

[29] proposed an idea of combining both reinforcement learning and evolutionary algorithm.

Different from the conventional evolutionary algorithm, the mutation for a specific candidate is no longer randomly selected. Instead, a reinforcement-learning based mutator is trained to generate a mutation according to possible effects of different mutations given the architecture of a candidate.

An evolutionary algorithm can suffer from poor population initialization and error accumulation. The initialization of the population determines the converging speed of the algorithm and the best performance can be achieved. Since the initialization of the population is a random process, it may raise problems. The evolutionary algorithm also implicitly assumes that the offspring of an individual with good fitness is more likely to beat that of an individual with poor fitness. However, this assumption remains doubtful because parts of neural network architecture are highly correlated with respect to the network's performance, which means a low-order Markov Decision Process very close to this assumption is not appropriate.

2-2-3 Search by One-shot Algorithm

[30] proposed a novel search strategy named "one-shot". The main idea of the one-shot algorithm is to squeeze all possible operations into one network. In [30] this compact network is then trained to give these operations relative appropriate and reasonable weights. Then the network is partly frozen and validated for several times to obtain one-shot validation accuracy of sub-architectures. Sub-architectures are ranked by this accuracy and to make the algorithm robust, a few top-ranked architectures are trained from scratch to get their performance substantiated and the best one is selected.

One-shot algorithm is fast since it needs only one epoch for one-shot accuracy and a few more epochs for several prior candidates. But it has a strong demand for dedicated GPU memory as all possible operations are squeezed into one network. Furthermore, one-shot algorithm is likely to suffer from hanging edges which will be further discussed in the following text.

2-3 Performance Evaluation Strategy

2-3-1 Performance Prediction

The metrics used to evaluate an architecture is the accuracy achieved by it on a validation dataset after being trained from scratch. A group of researchers developed an interest in getting accuracy without having to go through the entire training stage.

[31] proposed an additional network to predict a network architecture's performance based on the data from networks that are shallower. Architecture-validation accuracy pairs are collected during different stages of exploration and used by the prediction network which is later utilized to predict the performance of architectures in the following stage.

This method has two weaknesses worth being pointed out. The first one is the problem

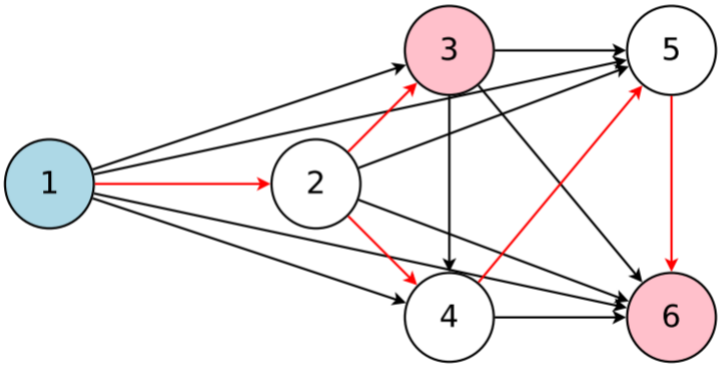

Figure 2-3: directed acyclic graph. Each node represents an operation and an architecture is built by a path going through nodes.

of error accumulation. As the number of stages gone through increases, the error of prediction produced by the prediction network also accumulates. This may hinder the prediction network's functionality and jeopardize the overall ability to find an optimal architecture by the search algorithm. The second one is the correlation between predicted performance and actual performance of architecture. In this paper, the prediction network is trained on data collected from previous stages but used to predict the performance of architecture in the following stage and this makes the correlation doubtful since in the next stage the depth of newly exploited architectures grows and it will significantly change an architecture's potential.

In [16] researchers took a less aggressive approach. A prediction network is trained on learning curve data from both the searching process and other sources. For architecture, it is trained from scratch for a predefined number of epochs. Then the prediction network will output an expected converged accuracy based on the early learning curve of these epochs. This approach avoids the risk of uncorrelation and gains enhancement from larger available datasets on learning curves. However, it still suffers from error accumulation and this "predefined number of epochs" requires careful choice to trade off between bias and improvement on speed.

2-3-2 Weight Sharing

The training of a neural network is a process of getting appropriate weights for a specific task. Many pieces of research have been carried out focusing on reusing of weights. [11] introduced directed acyclic graph into network architecture search and reformulated the problem as finding the optimal path passing through nodes that represent different operations. The weight of each node(operation) is shared between different paths. In [17] the weights obtained from evaluating architectures at the current stage are reused in evaluation at the next stage and architectures at successive stages are strongly related. The weight reuse in [11] is problematic because as path changes, the depth of a node also changes. Since the depth to some extent represents how abstract feature maps are, reusing the weight can possibly lead to a similar result after poor weight initialization. In addition, change of path can also change the shape of weights of operation and in this paper, the author simply padded with zero or cropped the

weight, which may still have the equivalent effect of wrong initialization.

By dividing the search process into stages, [17] avoided the problems mentioned above. However, the stage-wise search has its own drawbacks such as error accumulation and omission.

Chapter 3

Introduction to the Bayesian Approach

Bayesian approach is one-shot based NAS which is treated as a Network Compression/pruning problem on the architecture parameters from an over-parameterized network. Moreover, besides it's remarkable less searching time compared to reinforcement learning and neuro-evolutionary approaches, it 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, though the child network still has other non-zero edges to keep it connected [12, 32, 33, 34], for example, node 2 in Figure3-1a. Similarly, as shown in Figure 1 of [32], 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 the isolated nodes is possible, *e.g.*, for chain-like structure, it demands extensive extra computations to reconstruct the graph for more 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 [5], newly designed motifs [35] and for complex computer vision tasks such as semantic segmentation [36]. On the other hand, *zero* operations should have a higher priority to rule out other possible operations, since *zero* operations equal to all the *non-zero* operations not selected after all. Second, most one-shot NAS methods [12, 33, 32, 34, 37] rely on the magnitude of architecture parameters to prune the redundant parts, which this is not necessarily true. From the perspective of Network Compression [38], a magnitude-based metric depends on the scale of the weights and in turn require pre-training and is very sensitive to the architectural choices, and the magnitude does not necessarily imply the optimal edge. Unfortunately, these drawbacks exist not only for Network Compression but also for one-shot NAS.

BayesNAS is a novel, efficient and highly automated framework based on the classic Bayesian learning approach to alleviate these two issues simultaneously. Architecture parameters are modeled by a *hierarchical automatic relevance determination* (HARD) prior. The dependency can be translated by multiplication and accumulation of some independent Gaussian distributions. The classic Bayesian learning framework [39, 40, 41] prevents overfitting and

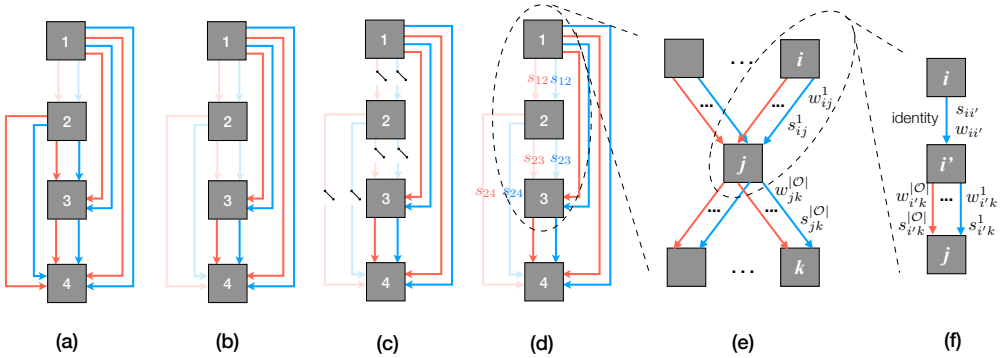

Figure 3-1: An illustration of BayesNAS: (a) disconnected graph with isolated node 2 caused by disregard for dependency; (b) expected connected graph with no connection from node 2 to 3 and from node 2 to 4; (c) illustration about dependency with predecessor's (e_{12}) superior control over its successors (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.

promotes sparsity by specifying sparse priors. The uncertainty of the parameter distribution can be used as a new metric to prune the redundant parts and the majority of parameters are automatically set to zero during the learning process.

Key Features

- **Bayesian approach:** BayesNAS is the first Bayesian approach for NAS. Therefore, it 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. Not only promote sparsity, but the priors can also 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 the experiment on CIFAR-10, it was observed 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:** This algorithm is simply formulated as an iteratively re-weighted ℓ_1 type algorithm [42] 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 the log likelihood. To make the computation for large networks feasible, a fast Hessian calculation method is adopted. In the experiment, the model is trained 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.

3-1 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 [43, 1, 11, 44, 34, 12, 33] have proposed that the search space could be represented by a Directed Acyclic Graph (DAG). Here e_{ij} represents 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 [30, 34, 12, 33, 37], BayesNAS includes (different or same) scaling scalars overall operations of all edges to control the information flow, denoted as w_{ij}^o which also represent architecture parameters. And the output of a mixed operation $o_{ij}, i < j$ is defined based on the outputs of its edge

$$o_j(z_i) = \sum_{o \in \mathcal{O}} w_{ij}^o o_{ij}(z_i). \tag{3-1}$$

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 [30]. The search problem can be formulated 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:

$$\begin{aligned} \min_{\mathbf{w}} L(\mathbf{W}; \mathbf{D}) = \min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} \ell(\mathbf{y}_n, \mathrm{Net}(\mathbf{x}_n, \boldsymbol{\mathcal{W}}, \mathbf{w})) \\ \text{s.t.} \quad \mathbf{w} \in \mathbb{R}^{m^{\text{net}} + m^{\text{edge}}}, \quad \|\mathbf{w}\|_0 \leq \kappa^{\text{edge}} \end{aligned} \tag{3-2}$$

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 (3-2) can be substantiated by incorporating *zero* operations into $\mathcal{O}$ to allow removal of w_{ij}^o [12, 33] 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, a switch s_{ij}^o is introduced 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 3-1c. 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 [12, 33, 37]. The calculation method will be introduced later in Section 3-2.

Inspired by the hierarchical representation in a DAG [12, 35], a single motif is abstracted as the building block of DAG, as shown in Figure 3-1e. 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 the output of motifs, cells, or the network.

3-2 Dependency based one-shot performance estimation strategy

3-2-1 Encoding the dependency logic

Following are the formal statements about the criterion to identify the redundant connections in Proposition 3-2-1. Before the statement, the idea can be illustrated by Figure3-1b 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. Naturally, it comes to the following proposition.

Proposition 1 There is information flow from node j to k under operation o' as shown in Figure 3-1d 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, *i.e.*, $w_{jk}^{o'} \sum_{i<j} w_{ij}^{o} \neq 0$.

Remark 1 The same expression for Proposition 3-2-1 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, *i.e.*, $w_{jk}^{o'} \sum_{i<j} w_{ij}^{o} = 0$. This explains the incompleteness of the problem 3-2 as well as the possible phenomenon that non-zero edges become dysfunctional in Figure 3-1b.

Remark 2 $w_{jk}^{o'} \sum_{i<j} w_{ij}^{o}$ in Proposition 3-2-1 and Remark 3-2-1 is not unique. Some other alternatives include but not limited to, *e.g.*, $(w_{jk}^{o'})^2 + \sum_{i<j} (w_{ij}^{o})^2$, $w_{jk}^{o'} \sum_{i<j} (w_{ij}^{o})^2$. Apparently, ℓ_0 norm of these quantities are difficult to be included in a constraint in the optimization problem formulation in 3-2.

Following describes show how the "switches" s can be used to implement Proposition 3-2-1. If 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

$$\bigcup_{i<j} \bigcup_{o \in \mathcal{O}} s_{ij}^{o} \cap s_{jk}^{o'} \quad \text{or} \quad \overline{\overline{\bigcup_{i<j} \bigcup_{o \in \mathcal{O}} s_{ij}^{o}} \cup \overline{s_{jk}^{o'}}} \tag{3-3}$$

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 can get the distribution over $w_{jk}^{o'} \sum_{i<j} w_{ij}^{o}$ as

$$\begin{aligned} p(w_{jk}^{o'} \sum_{i<j} w_{ij}^{o}) &= \mathcal{N}(w_{jk}^{o'}|0, s_{jk}^{o'}) \sum_{i<j} \mathcal{N}(w_{ij}^{o}|0, s_{ij}^{o}) \\ &= \mathcal{N}(w_{jk}^{o'} \sum_{i<j} w_{ij}^{o}|0, \gamma_{jk}^{o'}) \end{aligned} \tag{3-4}$$

where

$$\gamma_{jk}^{o'} \triangleq \left(\frac{1}{\sum_{i<j} \sum_{o \in \mathcal{O}} s_{ij}^{o}} + \frac{1}{s_{jk}^{o'}} \right)^{-1}. \tag{3-5}$$

(3-5) and 3-3 are equivalent. This indicates the possibility of finding an algorithm that is able to find a spare 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) [41] and automatic relevance determination (ARD) prior [45, 40] in Bayesian neural networks.

3-2-2 Zero operation ruling all

Between node i and j one more node i' is appended which allows only a single *identity* operation (see Figure 3-1f). The associated weight $w_{ii'}$ is trainable and initialized as 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 (3-6) can be substituted by

$$\gamma_{jk}^{o'} \triangleq \left(\frac{1}{\sum_{i<j} \sum_{o \in \mathcal{O}} \left(\frac{1}{s_{ii'}} + \frac{1}{s_{i'j}^{o}} \right)^{-1}} + \frac{1}{s_{jk}^{o'}} \right)^{-1}. \tag{3-6}$$

Chapter 4

BayesNAS and Hessian for Neural Networks

4-1 Introduction

The optimization objective of searching architecture becomes removing redundant/unimportant edges. The training algorithm is iteratively indexed by t. Each iteration may contain several epochs. The pseudo code is summarized in Algorithm 1[46]. The cost function is simply the maximum likelihood over the data D with regularization whose intensity is controlled by the re-weighted coefficient ω.

$$\mathcal{L}_D = E_D(\cdot) + \lambda_w \sum_{j<k} \sum_{o' \in \mathcal{O}} \|\omega_{jk}^{o'}(t) w_{jk}^{o'}\|_1 + \lambda \|\boldsymbol{\mathcal{W}}\|_2^2 \tag{4-1}$$

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 in the third part. The fourth part is to update the variables associated with the architecture parameters. The fifth part is to prune the architecture parameters.

As discussed previously on the drawback of the magnitude based pruning metric, *i.e.*, the posterior mean in our case or the point estimate in (4-1), we propose a new metric based on maximum entropy of the distribution. Since $p(w_{jk}^{o'})$ in (3-4) 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.*, $\gamma_{jk}^{o'} \leq 0.0585$.

The algorithm can be easily transferred to other scenarios. One scenario involves proxy tasks to find the cell. Similar to (4-1), we group the 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 the re-weighted group Lasso:

Algorithm 1 BayesNAS Algorithm.

Require: $\boldsymbol{\gamma}(0), \boldsymbol{\omega}(0), \mathbf{w}(0) = \mathbf{1}$; $\lambda = 0.01$; sparsity intensity $\lambda_w^o \in \mathbb{R}^+$
Ensure:
for $t = 1$ to $T_{\max}$ **do**
1. Update $\mathbf{w}$ and $\mathcal{W}$ by minimizing $\mathcal{L}_D$ in (4-1)
2. Compute Hessian for $\mathbf{w}$
3. Update variables associated with $\mathbf{w}$
while $i < j < k, o, o' \in \mathcal{O}$ **do**

$$C_{jk}^{o'}(t) = \left(\frac{1}{\gamma_{jk}^{o'}(t-1)} + \mathbf{H}_{jk}^{o'}(t) \right)^{-1} \tag{4-2}$$

$$\omega_{jk}^{o'}(t) = \frac{\sqrt{\gamma_{jk}^{o'}(t-1) - \mathrm{C}_{jk}^{o'}(t)}}{\gamma_{jk}^{o'}(t-1)} \tag{4-3}$$

$$s_{jk}^{o'}(t) = \left| \frac{w_{jk}^{o'}(t)}{\omega_{jk}^{o'}(t)} \right| \tag{4-4}$$

$$\gamma_{jk}^{o'}(t) \text{ is given by } 3-5 \text{ or } 3-6 \tag{4-5}$$

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

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

4-2 Hessian Computation

To use the algorithm, we need Hessian of architecture parameters. Although the Hessian of weight matrix has been widely used in second-order optimization techniques to speed up the training process [47, 48], it still remains infeasible to calculate explicit Hessian directly due to the intensive computation burden [49, 50]. Moreover, as most of the current deep neural networks include plenty of Convolutional (Conv) layers, it further increases the difficulty of calculation due to the indirect convolution operation. Inspired by the Hessian calculation methods for Fully Connected (FC) layers as shown in [50], we propose a recursive and efficient method to compute the Hessian of Conv layers by converting the Conv layers to FC layers [51]. Therefore the Hessian of the resulting equivalent FC layer is ready to be obtained.

4-2-1 Hessian for Fully Connected Layer

The mathematical operation in a fully connected layer could be formulated as:

$$h_j^o = W_{ij}^o a_i, \quad a_i = \sigma(h_i) \tag{4-6}$$

where h_i is the pre-activation value for node i and a_i is the activation value. $\sigma()$ is the element-wise activation function. $\mathcal{W}_{ij}^o$ stands for the weight matrix associated with operation o in edge e_{ij}^o. In [50], a recursive method is proposed to compute the Hessian $\mathbf{H}$ for $\mathcal{W}_{ij}^o$:

$$\mathbf{H}_{ij}^o = a_i \cdot (a_i)^\top \otimes H_j^o \tag{4-7}$$

where $\otimes$ stands for Kronecker product; The pre-activation Hessian H_j^o is known and could be used to compute the pre-activation Hessian recursively for the previous layer:

$$H_i = B_i(\mathcal{W}_{ij}^o)^\top H_j^o \mathcal{W}_{ij}^o B_i + D_i, \quad B_i = diag(\sigma'(h_i)), \quad D_i = diag(\sigma''(h_i) \circ \frac{\partial L}{\partial a_i}) \tag{4-8}$$

In order to reduce computation complexity, the original pre-activation Hessian H and Hessian $\mathbf{H}$ in Eq 4-7-4-8 are replaced with their diagonal values for recursive computation. Thus the matrix multiplication could be reduced to vector multiplication. The hessian calculation process could be reformulated as:

$$\mathbf{H}_{ij}^o = a_i^2 \otimes H_j^o \tag{4-9}$$

$$H_i = B_i^2 \circ (((\mathcal{W}_{ij}^o)^\top)^2 H_j^o) + D_i, \quad B_i = \sigma'(H_i), \quad D_i = \sigma''(H_i) \circ \frac{\partial L}{\partial a_i} \tag{4-10}$$

Where diag() means the operation to extract the diagonal values of input variable. If we compute Hessian with the approximate method as Eq 4-9 - 4-10, the multiply accumulate operation (MACs) for the pre-activatiion Hessian H and Hessian $\mathbf{H}$ could be significantly reduced.

4-2-2 Hessian for Convolutional Layer

Conv layers can be converted to FC layers thereafter the Hessian of the resulting equivalent FC layer is ready to be obtained. Specifically, suppose a convolution operation o is selected between node i and j ($i < j$). The corresponding input vector, weight and output vector of this edge are denoted as $B_i \in \mathbb{R}^{b \times C_i \times H_i \times W_i}$, $\mathcal{W}_{ij}^o \in \mathbb{R}^{C_j^o \times C_i \times m_{ij}^o \times k_{ij}^o}$ and $B_j^o \in \mathbb{R}^{b \times C_j^o \times H_j^o \times W_j^o}$ respectively, where b is the batch size, C_i, H_i, W_i are the size of input channel, height and width; C_j^o is the size of output channel, $m_{ij}^o \times k_{ij}^o$ is the kernel dimension; H_j^o and W_j^o are the size of output height and width.

As in [51], B_i is converted to two dimensional matrix for FC layer, with dimension $(bH_j^oW_j^o) \times (C_i m_{ij}^o k_{ij}^o)$. Similarly, the dimension of B_j^o is changed from $b \times C_j^o \times H_j^o \times W_j^o$ to $(bH_j^oW_j^o) \times C_j^o$. The dimension of $\mathcal{W}_{ij}^o$ is changed to $\mathbb{R}^{(C_i m_{ij}^o k_{ij}^o) \times C_j^o}$. The input vector, output vector and weight for the FC layer are denoted as M_i, M_j^o and $\mathcal{W}_{ij}^{oM}$.

Secondly, M_i, M_j^o and H_j^{oM} are decomposed into a total of $bH_j^oW_j^o$ row vectors with $(M_i)^n \in \mathbb{R}^{C_i m_{ij}^o k_{ij}^o}$, $(M_j^o)^n \in \mathbb{R}^{C_j^o}$ and $(H_j^{oM})^n \in \mathbb{R}^{C_j^o}$ $(n = 1, \ldots, bH_j^oW_j^o)$ respectively. It is easy to

understand that $(M_i)^n$, $(M_j^o)^n$ could be regarded as the input vector and output vector of a FC layer with weight matrix $\mathcal{W}_{ij}^{oM}$. Then we can obtain the Hessian $\mathbf{H}_{ij}^{o}$ for $\mathcal{W}_{ij}^{o}$ as follows:

$$(\mathbf{H}_{ij}^{oM})^n = (M_i)^n \cdot (M_i)^{n\top} \otimes (H_j^{oM})^n \tag{4-11}$$

$(H_j^{oM})^n$ is the pre-activation Hessian which could be computed recursively. With $(H_j^{oM})^n$ known, the pre-activation Hessian for $(M_i)^n$ could be calculated as:

$$\begin{aligned}
(H_i^M)^n &= (B_i)^n \mathcal{W}_{ij}^{oM\top} (H_j^{oM})^n \mathcal{W}_{ij}^{oM} (B_i)^n + (D_i)^n \\
(B_i)^n &= diag\left(\sigma'((h_i)^n)\right) \\
(D_i)^n &= diag\left(\sigma''((h_i)^n) \frac{\partial L}{\partial (M_i)^n}\right)
\end{aligned}$$

where $(h_i)^n$ is the pre-activation value for FC layer and L means the loss function. The pre-activation Hessian H_i^M could be obtained after concatenating all $(H_i^M)^n$ as

$$H_i^M = [diag((H_i^M)^1); \ldots; diag((H_i^M)^{bH_j^o W_j^o})] \tag{4-12}$$

the Hessian $\mathbf{H}_{ij}^{oM}$ for $\mathcal{W}_{ij}^{oM}$ can be obtained as:

$$\mathbf{H}_{ij}^{oM} = \frac{1}{bH_j^o W_j^o} \sum_{n=1}^{bH_j^o W_j^o} (\mathbf{H}_{ij}^{oM})^n \tag{4-13}$$

It should be noted that as pre-activation Hessian is a recursive variable for convolutional layer and Hessian will be used for updating hyper-parameters which will be introduced later, both H_i^M and $\mathbf{H}_{ij}^{oM}$ should be converted back to conv type before imparting to next layer with dimension $\mathbb{R}^{b\times C_i \times H_i \times W_i}$ and $\mathbb{R}^{C_j^o \times C_i \times m_{ij}^o \times k_{ij}^o}$.

Chapter 5

Experiments

The experiments focus on two scenarios in NAS: proxy NAS and proxyless NAS. For proxy NAS, we follow the pipeline in DARTS [12] and SNAS [32]. 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 on CIFAR-10 to compare the performance of BayesNAS with other state-of-the-art methods. For proxyless NAS, we follow the pipeline in ProxylessNAS [33]. First, the tree-like cell from [44] with multiple paths is integrated into the PyramidNet [52]. 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.

Dataset CIFAR-10 dataset [53] is a basic dataset for image classification, which consists of 50,000 training images and 10,000 testing images. Data transformation is achieved by the standard data pre-processing and augmentation techniques (see Appendix **??**).

5-1 Proxy Search

Motivation We apply BayesNAS to find convolutional cells on CIFAR-10 for image classification. Unlike DARTS and SNAS, which evaluate the performance of child networks during the searching stage by training their snapshots from scratch or use search validation accuracy as the performance evaluation criterion, we use γ in BayesNAS as performance evaluation criterion which enables us to achieve it in a one-shot manner.

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

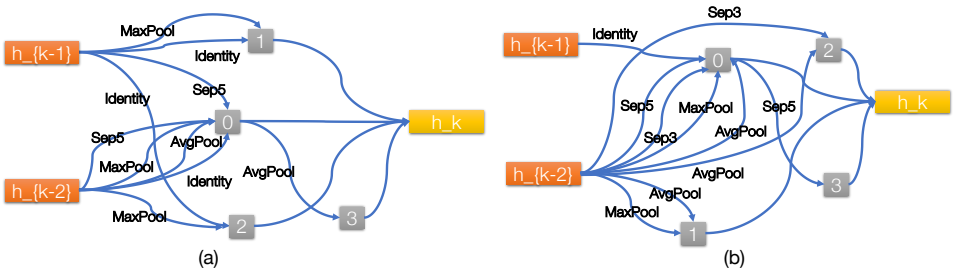

Figure 5-1: Normal and reduction cell found by BayesNAS with $\lambda_w^o = 0.01$.

network to reduce the spatial resolution of feature maps. Available operations include *identity*, *3x3 depth-separable convolution*, *5x5 depth-separable convolution*, *3x3 dilated convolution*, *5x5 dilated convolution*, *max pooling* and *average pooling*. 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. The search takes about 3 hours on a single GPU[1].

Searching Process The normal and reduction cells learned on CIFAR-10 using BayesNAS are shown in Figure 5-1a and 5-1b. Though architecture is determined after only one training epoch, to consolidate the validity of the found architecture, we examined the validation accuracy for several extra epochs after the architecture is determined and fixed and observed that it can reach beyond 80% accuracy.

Searching Results 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 5-1. The test error rate of BayesNAS is still competitive against the state-of-the-art reinforcement learning-based, evolution algorithm-based, and gradient-based NAS. For model complexity, BayesNAS is able to find convolutional cells with fewer parameters when compared to DARTS and SNAS.

5-2 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. What is shared between these two scenarios is that the process is a one-shot search crediting to the adoption of γ in BayesNAS as a performance evaluation criterion.

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

Table 5-1: 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 [5]	3.46	25.6	-	manual
NASNet-A + cutout [1]	2.65	3.3	1800	RL
AmoebaNet-B + cutout [54]	2.55 ± 0.05	2.8	3150	evolution
Hierarchical Evo [35]	3.75 ± 0.12	15.7	300	evolution
PNAS [55]	3.41 ± 0.09	3.2	225	SMBO
ENAS + cutout [11]	2.89	4.6	0.5	RL
Random search baseline + cutout [12]	3.29 ± 0.15	3.2	1	random
DARTS (2nd order bi-level) + cutout [12]	2.76 ± 0.09	3.4	1	gradient
SNAS (single-level) + moderate con + cutout [32]	2.85 ± 0.02	2.8	1.5	gradient
DSO-NAS-share+cutout [34]	2.84 ± 0.07	3.0	1	gradient
Proxyless-G + cutout [33]	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

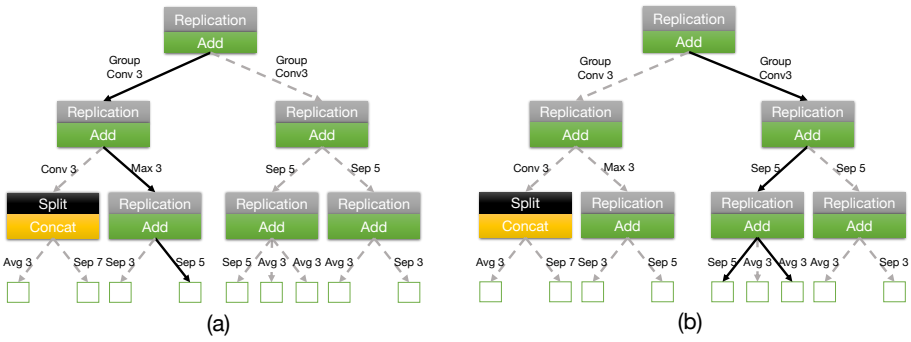

Figure 5-2: The pruned tree-cell: (a) The chain-like where only one path exists in the cell connecting the input of the cell to its output. (b) The inception structure where divergence and convergence both exist in the cell. The solid directed lines denote the path found by BayesNAS while the dashed 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 [44].

Training Settings In the searching stage, we set the 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}.

Searching Process Because in proxyless search each cell can have a different structure, we demonstrate only two typical types of cell structure among all of them in Figure 5-2a and Figure 5-2b. 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 the inception structure where divergence and convergence both exist in the cell. Regardless of which type the cells belong to, they are much simpler than the original tree-cell in [44]. In addition, our further observation reveals that some cells are dispensable with respect to the entire network.

Searching Results 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 [56]. The validation accuracy is also presented in Table 5-1. Although test error increases slightly compared to [33], there is a significant drop in the number of model parameters to be learned which is beneficial for both training and inference.

5-3 Transferability to ImageNet

For the 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 5-2 show that the cell learned on CIFAR-10 can be transferred to ImageNet and is capable of achieving competitive performance.

5-4 Conclusion

BayesNAS can directly learn a sparse neural network architecture on both proxy and proxyless tasks and 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, theoretically, Hessian computation can be done along with standard backpropagation which will potentially further reduce the searching time and scale our approach to larger search space.

Table 5-2: Comparison with state-of-the-art image classifiers on ImageNet in the mobile setting.

Architecture	**Test Error (%)**		**Params**	**Search Cost**	**Search**
	top-1	top-5	**(M)**	**(GPU days)**	**Method**
Inception-v1 [57]	30.2	10.1	6.6	–	manual
MobileNet [58]	29.4	10.5	4.2	–	manual
ShuffleNet 2× (v1) [59]	29.1	10.2	∼5	–	manual
ShuffleNet 2× (v2) [59]	26.3	–	∼5	–	manual
NASNet-A [1]	26.0	8.4	5.3	1800	RL
NASNet-B [1]	27.2	8.7	5.3	1800	RL
NASNet-C [1]	27.5	9.0	4.9	1800	RL
AmoebaNet-A [54]	25.5	8.0	5.1	3150	evolution
AmoebaNet-B [54]	26.0	8.5	5.3	3150	evolution
AmoebaNet-C [54]	24.3	7.6	6.4	3150	evolution
PNAS [55]	25.8	8.1	5.1	∼225	SMBO
DARTS [12]	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

Appendix A

Application in Network Compression

A-1 Introduction

In addition to applying the proposed Bayesian approach to address NAS problem, we also explore the possibility of our method on network structural compression problem. For weight in the l-th convolutional layer $W^l \in \mathbb{R}^{N_l \times C_l \times m_l \times k_l}$, some examples of the structured sparsity are shown in Fig.A-1.

Algorithm 2[46] for structural compression is a variation of Algorithm 1 in BayesNAS and hessian computation here follows the same rules as in 4-2.

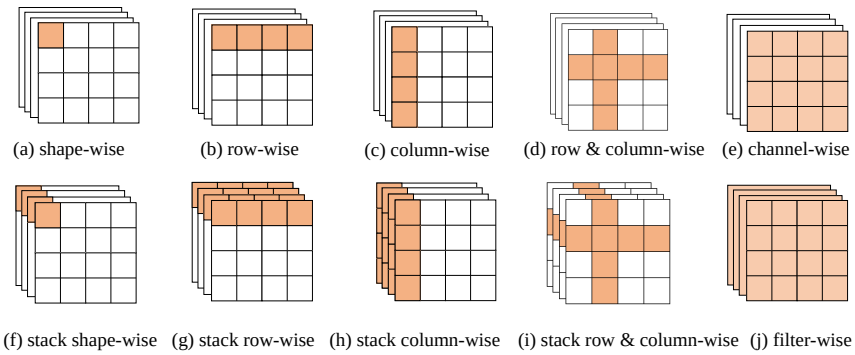

Figure A-1: some examples of structured sparsity for the 3D filters in the Conv layer with extensions of [2]. Colored squares mean the weights to be pruned. It should be noted that the FC layer can be easily enforced by (a)-(e).

Algorithm 2 Variation for network compression.

Require: Initialization: $\forall l = 1, \ldots, L$, $\omega^l(0), \gamma^l(0) = 1$; $\lambda^l \in \mathbb{R}^+$;
Ensure:
for $t = 1$ to $T_{\max}$ **do**
1. Maximum likelihood with regularization:

$$\min_{\mathcal{W}} E_D(\cdot) + \sum_{l=1}^{L} \lambda^l R(\omega^l(t) \circ \mathcal{W}^l) \tag{A-1}$$

2. Compute the Hessian for fully connected layer and convolutional layers.
3. Update hyper-parameters:

$$\gamma^l(t) \leftarrow \text{Update}(\omega^l(t-1), \mathcal{W}^l(t)), \Gamma^l(t) = [\gamma^l(t)] \tag{A-2}$$

$$C^l(t) \leftarrow \left((\Gamma^l(t))^{-1} + diag(\mathrm{H}^l(t))\right)^{-1}, \tag{A-3}$$

$$\alpha^l(t) \leftarrow -\frac{\mathrm{C}^l(t)}{\gamma^l(t)^2} + \frac{1}{\gamma^l(t)}\{\text{element-wise division}\} \tag{A-4}$$

$$\omega^l(t) \leftarrow \text{Update}(\alpha^l(t)) \tag{A-5}$$

4. Prune the unimportant connections.
end for

A-2 Experiments

A-2-1 LeNet-300-100 and LeNet-5 on MNIST

We first perform LeNet-300-100 and LeNet-5 on MNIST dataset [60]. For LeNet-300-100, we apply shape-wise, row-wise and column-wise regularization as shown in Fig. A-1(a), A-1(b) and A-1(c), for the 2D weight matrices. The hyper-parameters γ, ω and α are updated every ten epochs for a total of $T_{\max} = 10$ loops. The learned structure is $465 - 37 - 90$ with 1.54% test error and 0.04 FLOPS [61]. Comparison with other methods can be found in Table A-1. For LeNet-5, we apply shape-wise and filter-wise regularization for the conv layer as shown in Fig. A-1(a) and A-1(j); row-wise and column-wise regularization for fc layer as shown in Fig. A-1(b) and A-1(c). The learned structure is $5 - 10 - 65 - 25$ with 1.00% test error and 0.57 FLOPS. Comparison with other methods can be found in Table A-2.

Table A-1: Comparison of the learned architecture with other methods using LeNet-300-100 on MNIST dataset

Method	Pruned Architecture	Error Rate (%)	FLOPs (M)
Baseline	784-300-100	1.39	0.53
SBP ([62])	245-160-55	1.60	0.10
BC-GNJ ([63])	278-98-13	1.80	0.06
BC-GHS ([63])	311-86-14	1.80	0.06
Practical ℓ_0([64])	219-214-100	1.40	0.14
Practical ℓ_0 ([64])	266-88-33	1.80	0.05
Proposed method	465-37-90	1.54	0.04

A-2-2 ResNet-18 on CIFAR-10

We also evaluate our algorithm on Cifar10 dataset using ResNet-18 as initialized backbone [4]. In addition to the input conv layer and output fc layer, the other 16 conv layers are separated into 8 blocks with 2 layers each. We apply shape-wise and filter-wise regularization to the conv layer as shown in Fig. A-1(a) and A-1(j); row-wise and column-wise regularization to the fc layer as shown in Fig. A-1(b) and A-1(c). The result is given in Table A-3. It can

Table A-2: Comparison of the learned architecture with other methods using LeNet-5 on MNIST dataset

Method	Pruned Architecture	Error Rate (%)	FLOPs (M)
Baseline	20-50-800-500	0.83	8.85
SBP ([62])	3-18-284-283	0.90	0.69
BC-GNJ ([63])	8-13-88-13	1.00	1.09
BC-GHS ([63])	5-10-76-16	1.00	0.57
Practical ℓ_0([64])	20-25-45-462	0.90	4.49
Practical ℓ_0 ([64])	9-18-65-25	1.00	1.55
Proposed method	5-10-65-25	1.00	0.57

be found that two Conv layers in block 4 are pruned away which shows the potential of our method to reduce the number of layers.

Table A-3: Sparsity for each layer in ResNet-18 on Cifar10 dataset

conv1	conv2-5	conv6-9	conv10-13	con14-17	FC layer	Total	Test error
22.80%	10.06%	9.24%	20.64%	1.43%	10.33%	2.96%	6.58% (baseline)
	22.87%	5.45%	15.04%	0.19%			
	20.05%	4.94%	10.77%	0			6.23% (our method)
	12.99%	10.73%	4.61%	0			

Appendix B

Supplementary Material

Code for this project can found in *https://github.com/BayesNAS/BayesNAS*.

Bibliography

[1] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," *arXiv preprint arXiv:1707.07012*, vol. 2, no. 6, 2017.

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

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[4] 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*, pp. 770–778, 2016.

[5] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks.," in *CVPR*, vol. 1, p. 3, 2017.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] L. Deng, D. Yu, *et al.*, "Deep learning: methods and applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[8] L. Xie and A. L. Yuille, "Genetic cnn.," in *ICCV*, pp. 1388–1397, 2017.

[9] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[10] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)," *URL http://www. cs. toronto. edu/kriz/cifar. html*, 2010.

[11] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.

[12] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[13] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, 2010.

[14] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, p. 20, 2015.

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

[16] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," *arXiv preprint arXiv:1705.10823*, 2017.

[17] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," AAAI, 2018.

[18] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," *ArXiv e-prints*, 1804.

[19] H. Jin, Q. Song, and X. Hu, "Efficient neural architecture search with network morphism," *arXiv preprint arXiv:1806.10282*, 2018.

[20] C. Zhang, M. Ren, and R. Urtasun, "Graph hypernetworks for neural architecture search," *arXiv preprint arXiv:1810.05749*, 2018.

[21] Anonymous, "Learnable embedding space for efficient neural architecture compression," in *Submitted to International Conference on Learning Representations*, 2019. under review.

[22] A.-C. Cheng, J.-D. Dong, C.-H. Hsu, S.-H. Chang, M. Sun, S.-C. Chang, J.-Y. Pan, Y.-T. Chen, W. Wei, and D.-C. Juan, "Searching toward pareto-optimal device-aware neural architectures," *arXiv preprint arXiv:1808.09830*, 2018.

[23] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, pp. 1008–1014, 2000.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[25] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[26] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[27] H. Zhang, S. Kiranyaz, and M. Gabbouj, "Finding better topologies for deep convolutional neural networks by evolution," *arXiv preprint arXiv:1809.03242*, 2018.

[28] K. A. De Jong, *Evolutionary computation: a unified approach*. MIT press, 2006.

[29] Y. Chen, Q. Zhang, C. Huang, L. Mu, G. Meng, and X. Wang, "Reinforced evolutionary neural architecture search," *arXiv preprint arXiv:1808.00193*, 2018.

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

[31] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," *arXiv preprint arXiv:1712.00559*, 2017.

[32] S. Xie, H. Zheng, C. Liu, and L. Lin, "Snas: stochastic neural architecture search," *arXiv preprint arXiv:1812.09926*, 2018.

[33] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[34] X. Zhang, Z. Huang, and N. Wang, "Single shot neural architecture search via direct sparse optimization," 2018.

[35] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," *arXiv preprint arXiv:1711.00436*, 2017.

[36] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. Yuille, and L. Fei-Fei, "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," *arXiv preprint arXiv:1901.02985*, 2019.

[37] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, "Morphnet: Fast & simple resource-constrained structure learning of deep networks," *arXiv preprint arXiv:1711.06798*, 2017.

[38] N. Lee, T. Ajanthan, and P. H. Torr, "Snip: Single-shot network pruning based on connection sensitivity," *arXiv preprint arXiv:1810.02340*, 2018.

[39] D. J. MacKay, "Bayesian interpolation," *Neural computation*, vol. 4, no. 3, pp. 415–447, 1992.

[40] R. M. Neal, "Bayesian learning for neural networks," 1995.

[41] M. E. Tipping, "Sparse bayesian learning and the relevance vector machine," *Journal of machine learning research*, vol. 1, no. Jun, pp. 211–244, 2001.

[42] E. J. Candes, M. B. Wakin, and S. P. Boyd, "Enhancing sparsity by reweighted ℓ_1 minimization," *Journal of Fourier analysis and applications*, vol. 14, no. 5-6, pp. 877–905, 2008.

[43] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[44] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu, "Path-level network transformation for efficient architecture search," *arXiv preprint arXiv:1806.02639*, 2018.

[45] D. J. MacKay, "Bayesian methods for backpropagation networks," in *Models of neural networks III*, pp. 211–254, Springer, 1996.

[46] H. Zhou, M. Yang, J. Wang, and W. Pan, "Bayesnas: A bayesian approach for neural architecture search," 2019.

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

[48] S.-I. Amari, "Natural gradient works efficiently in learning," *Neural computation*, vol. 10, no. 2, pp. 251–276, 1998.

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

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

[51] W. Ma and J. Lu, "An equivalence of fully connected layer and convolutional layer," *arXiv preprint arXiv:1712.01252*, 2017.

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

[53] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," tech. rep., Citeseer, 2009.

[54] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," *arXiv preprint arXiv:1802.01548*, 2018.

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

[56] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

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

[58] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[59] X. Zhang, X. Zhou, M. Lin, and J. Sun, "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.

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

[61] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

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

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

[64] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through l_0 regularization," *arXiv preprint arXiv:1712.01312*, 2017.

Glossary

List of Acronyms

DNN	Deep Neural Networks
CNN	Convolutional Neural Networks
RNN	Recurrent Neural Networks
Tanh	Hyperbolic Tangent Function
ReLU	Rectified Linear Unit
NAS	Neural Architecture Search
MDP	Markov Decision Process