Adaptive Stochastic Natural Gradient Method for One-Shot Neural Architecture Search

Youhei Akimoto [*1] Shinichi Shirakawa [*2] Nozomu Yoshinari [2] Kento Uchida [2] Shota Saito [2 3] Kouhei Nishida [4]

Abstract

High sensitivity of neural architecture search (NAS) methods against their input such as step-size (i.e., learning rate) and search space prevents practitioners from applying them out-of-the-box to their own problems, albeit its purpose is to automate a part of tuning process. Aiming at a fast, robust, and widely-applicable NAS, we develop a generic optimization framework for NAS. We turn a coupled optimization of connection weights and neural architecture into a differentiable optimization by means of stochastic relaxation. It accepts arbitrary search space (widely-applicable) and enables to employ a gradient-based simultaneous optimization of weights and architecture (fast). We propose a stochastic natural gradient method with an adaptive step-size mechanism built upon our theoretical investigation (robust). Despite its simplicity and no problem-dependent parameter tuning, our method exhibited near state-of-the-art performances with low computational budgets both on image classification and inpainting tasks.

1. Introduction

Neural architecture search (NAS) is a promising way to automatically find a reasonable neural network architecture and one of the most popular research topics in deep learning. The success of deep learning impresses people from outside machine learning communities and attracts practitioners to apply deep learning to their own tasks. However, they face different difficulties when applying deep learning. One difficulty is to determine the neural architecture for their previously-unseen problem. NAS is a possible solution to this difficulty.

Work published before 2017 often frames NAS as a hyper-parameter optimization, where an architecture's performance is measured by the validation error obtained after the training of the weights under a fixed architecture (Real et al., 2017; Suganuma et al., 2017; Zoph & Le, 2017). More recent studies (Brock et al., 2018; Shirakawa et al., 2018; Pham et al., 2018; Liu et al., 2019; Xie et al., 2019; Cai et al., 2019), on the other hand, optimize the weights and the architecture simultaneously within a single training by treating all possible architectures as subgraphs of a supergraph. These approaches are called *one-shot architecture search* or *one-shot NAS*. They break through the bottleneck of the hyper-parameter optimization approaches, namely, high computational cost for each architecture evaluation, and enable to perform NAS on a standard personal computer, leading to gathering more potential applications.

Research directions of NAS fall into three categories (Elsken et al., 2019): performance estimation (how to estimate the performance of architectures), search space definition (how to define the possible architectures), and search strategy (how to optimize the architecture). In the last direction, promising approaches transform a coupled optimization of weights and architectures into optimization of a differentiable objective by means of *continuous relaxation* (Liu et al., 2019; Xie et al., 2019) or *stochastic relaxation* (Shirakawa et al., 2018; Pham et al., 2018). A gradient descent or a natural gradient descent strategy with an existing adaptive step-size mechanism or a constant step-size is then employed to optimize weights and architecture simultaneously. However, optimization performance is sensitive against its inputs such as step-size (i.e., learning rate) and search space, limiting its application to unseen tasks.

To achieve a robust NAS, we develop a generic optimization framework for one-shot NAS. Our strategy is based on stochastic relaxation. We generalize the work by Shirakawa et al. (2018) to enable arbitrary types of architecture variables including categorical variables, ordinal (such as real or integer) variables, and their mixture. We develop a unified optimization framework for our stochastic relaxation based on the so-called stochastic natural gradient (Amari, 1998). Our theoretical investigation derives a condition on the step-size for our objective value to improve monotonically every iteration. We propose a step-size adaptation mechanism to

[*]Equal contribution [1]University of Tsukuba & RIKEN AIP [2]Yokohama National University [3]SkillUp AI Co., Ltd. [4]Shinshu University. Correspondence to: Youhei Akimoto <akimoto@cs.tsukuba.ac.jp>, Shinichi Shirakawa <shirakawa-shinichi-bg@ynu.ac.jp>.

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

approximately satisfy the condition. It significantly relaxes the performance sensitivity on the inputs and makes the overall framework rather flexible.

Our contributions are summarized as follows: (i) Our framework can treat virtually arbitrary types of architecture variables as long as one can define a parametric family of probability distributions on it; (ii) We propose a step-size adaptation mechanism for the stochastic natural gradient ascent, improving the optimization speed as well as its robustness against the hyper-parameter tuning. The default values are prepared for all introduced hyper-parameters, and they need not be touched even when the architecture search space changes; (iii) The proposed approach can enjoy parallel computer architecture, while it is comparable or even faster than existing approaches even on serial implementation; and (iv) Our strategy is rather simple, allowing us theoretical investigation, based on which we develop the step-size adaptation mechanism.

2. Our Approach

In this paper we address the following optimization problem

$$\max_{\boldsymbol{x}\in\mathcal{X},\ \boldsymbol{c}\in\mathcal{C}} f(\boldsymbol{x},\boldsymbol{c}) \ , \tag{1}$$

where $f : \mathcal{X} \times \mathcal{C} \to \mathbb{R}$ is the objective function that is differentiable with respect to (w.r.t.) $\boldsymbol{x} \in \mathcal{X}$ and is black-box w.r.t. $\boldsymbol{c} \in \mathcal{C}$. The domain $\mathcal{X}$ of $\boldsymbol{x}$ is a subset of n_x dimensional real space $\mathbb{R}^{n_x}$, whereas $\mathcal{C}$ can be either categorical, continuous, or their product space. Our objective is to simultaneously optimize $\boldsymbol{x}$ and $\boldsymbol{c}$ by possibly utilizing the gradient $\nabla_{\boldsymbol{x}} f$. In the context of NAS, $\boldsymbol{x}$, $\boldsymbol{c}$ and f represent connection weights, architecture parameters, a criterion to be maximized such as negative loss.

2.1. Stochastic Relaxation

We turn the original optimization problem into an optimization of differentiable objective J by means of *stochastic relaxation*. For this purpose, we introduce a family of probability distributions $\mathcal{P} = \{P_\theta : \theta \in \Theta \subseteq \mathbb{R}^{n_\theta}\}$ defined on $\mathcal{C}$. We suppose that for any $\boldsymbol{c} \in \mathcal{C}$ the family of probability distribution contains a sequence of the distributions that approaches the Dirac-Delta distribution $\delta_{\boldsymbol{c}}$ concentrated at $\boldsymbol{c}$. Moreover, we suppose that any $P_\theta \in \mathcal{P}$ admits the density function p_θ w.r.t. the reference measure $d\boldsymbol{c}$ on $\mathcal{C}$, and the log-density is differentiable w.r.t. $\theta \in \Theta$. The stochastic relaxation of f given $\mathcal{P}$ is defined as follows

$$J(\boldsymbol{x},\theta) := \int_{\boldsymbol{c}\in\mathcal{C}} f(\boldsymbol{x},\boldsymbol{c}) p_\theta(\boldsymbol{c}) d\boldsymbol{c} = \mathbb{E}_{p_\theta}[f(\boldsymbol{x},\boldsymbol{c})] \ . \tag{2}$$

Maximization of J coincides with maximization of f, as $\sup_{\theta\in\Theta} J(\boldsymbol{x},\theta) = \sup_{\boldsymbol{c}\in\mathcal{C}} f(\boldsymbol{x},\boldsymbol{c}) = f(\boldsymbol{x},\boldsymbol{c}^*)$, where the supremum of $J(\boldsymbol{x},\theta)$ is attained by the limit of the sequence $\{\theta\}$ where P_θ converges to $\delta_{\boldsymbol{c}^*}$.

The stochastic relaxation J inherits nice properties of f. For example, if $f(\boldsymbol{x},\boldsymbol{c})$ is convex and/or Lipschitz continuous w.r.t. $\boldsymbol{x}$, then so is $J(\boldsymbol{x},\theta)$ w.r.t. $\boldsymbol{x}$, respectively. Moreover, the stochastic relaxation J is differentiable w.r.t. both $\boldsymbol{x}$ and θ under mild conditions as follows

$$\nabla_{\boldsymbol{x}} J(\boldsymbol{x},\theta) = \mathbb{E}_{p_\theta}[\nabla_{\boldsymbol{x}} f(\boldsymbol{x},\boldsymbol{c})] \tag{3}$$

$$\nabla_\theta J(\boldsymbol{x},\theta) = \mathbb{E}_{p_\theta}[f(\boldsymbol{x},\boldsymbol{c})\nabla_\theta \ln(p_\theta(\boldsymbol{c}))] \ . \tag{4}$$

Stochastic Relaxation with Exponential Family: An exponential family consists of probability distributions whose density is expressed as $h(\boldsymbol{c})\cdot\exp(\eta(\theta)^{\mathrm{T}}T(\boldsymbol{c})-\varphi(\theta))$, where $T : \mathcal{C} \to \mathbb{R}^{n_\theta}$ is the sufficient statistics, $\eta : \Theta \to \mathbb{R}^{n_\theta}$ is the natural parameter of this family, and $\varphi(\theta)$ is the normalization factor. For the sake of simplicity, we limit our focus on the case $h(\boldsymbol{c}) = 1$. If we choose the parameter θ so that $\theta = \mathbb{E}_{p_\theta}[T(\boldsymbol{c})]$, it is called the *expectation parameters* of this family. Under the expectation parameters, the natural gradient of the log-likelihood reduces to $\tilde{\nabla} \ln(p_\theta(\boldsymbol{c})) = T(\boldsymbol{c}) - \theta$. The inverse of Fisher information matrix is $\mathbf{F}^{-1}(\theta) = \mathbb{E}[(T(\boldsymbol{c}) - \theta)(T(\boldsymbol{c}) - \theta)^{\mathrm{T}}]$, and is typically expressed as an analytical function of θ.

2.2. Alternating Gradient Ascent

Let $\boldsymbol{x}^t$ and θ^t represent the parameter values at iteration t. We maximize (2) by alternating optimization, namely,

$$\boldsymbol{x}^{t+1} = \mathrm{argmax}_{x\in\mathcal{X}} \ J(\boldsymbol{x},\theta^t) \tag{5}$$

$$\theta^{t+1} = \mathrm{argmax}_{\theta\in\Theta} \ J(\boldsymbol{x}^{t+1},\theta) \ . \tag{6}$$

Alternating steepest ascent is a way to avoid repeatedly solving computationally heavy optimization (5) and (6).

Stochastic Gradient Ascent on $\mathcal{X}$: Update step (5) is replaced with the gradient ascent w.r.t. a metric $\mathbf{A}$ on $\mathcal{X}$ with possibly time-dependent step-size $\epsilon_{\boldsymbol{x}}$,

$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t + \epsilon_{\boldsymbol{x}} \mathbf{A} \nabla_{\boldsymbol{x}} J(\boldsymbol{x}^t,\theta^t) \ . \tag{7}$$

Here $\mathbf{A}$ may change over t, leading to (quasi-) second order update. For fixed θ^t, it has been widely investigated in literature, and convergence of $\boldsymbol{x}$ to a stationary point $\|\nabla_{\boldsymbol{x}} J(\boldsymbol{x},\theta^t)\| = 0$ is guaranteed under different conditions.

Monotone improvement of J is easily derived under different conditions. An example result is as follows.

Proposition 1. *Assume that $J(\boldsymbol{x},\theta^t)$ is $\|\cdot\|_{\mathbf{A}}$-Lipschitz smooth w.r.t. $\boldsymbol{x}$: $\|\nabla_{\boldsymbol{x}} J(\boldsymbol{x}',\theta^t) - \nabla_{\boldsymbol{x}} J(\boldsymbol{x},\theta^t)\|_{\mathbf{A}} \leq L\|\boldsymbol{x}' - \boldsymbol{x}\|_{\mathbf{A}}$. (This is satisfied if $f(\boldsymbol{x},\boldsymbol{c})$ is so for all $\boldsymbol{c}$.) Then, for $\epsilon_{\boldsymbol{x}} < 2/L$, we have the monotone improvement*

$$\begin{aligned} &J(\boldsymbol{x}^{t+1},\theta^t) - J(\boldsymbol{x}^t,\theta^t) \\ &\qquad \geq (\epsilon_{\boldsymbol{x}} - (L/2)\epsilon_{\boldsymbol{x}}^2)\|\nabla_{\boldsymbol{x}} J(\boldsymbol{x}^t,\theta^t)\|_{\mathbf{A}}^2 > 0 \ . \end{aligned} \tag{8}$$

In our situation, the gradient $\nabla_{\boldsymbol{x}} J(\boldsymbol{x}^t, \theta^t)$ is not tractable. Instead, we estimate it by Monte-Carlo (MC) using $\nabla_{\boldsymbol{x}} J(\boldsymbol{x}^t, \boldsymbol{c}_i)$ with independent and identically distributed (i.i.d.) samples $\boldsymbol{c}_i \sim P_{\theta^t}$ $(i = 1, \dots, \lambda_{\boldsymbol{x}})$, namely,

$$G_{\boldsymbol{x}}(\boldsymbol{x}^t, \theta^t) = \frac{1}{\lambda_{\boldsymbol{x}}} \sum_{i=1}^{\lambda_{\boldsymbol{x}}} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^t, \boldsymbol{c}_i) \enspace . \tag{9}$$

The strong law of large numbers shows $\lim_{\lambda_{\boldsymbol{x}} \to \infty} G_{\boldsymbol{x}}(\boldsymbol{x}^t, \theta^t) = \nabla_{\boldsymbol{x}} J(\boldsymbol{x}^t, \theta^t)$ almost surely under mild conditions. The number $\lambda_{\boldsymbol{x}}$ of MC samples determines the trade-off between the accuracy and the computational cost.

We replace $\nabla_{\boldsymbol{x}} J(\boldsymbol{x}^t, \theta^t)$ with $G_{\boldsymbol{x}}(\boldsymbol{x}^t, \theta^t)$, leading to a stochastic gradient ascent, for which adaptation mechanisms for the step-size $\epsilon_{\boldsymbol{x}}$ are developed.

Stochastic Natural Gradient Ascent on Θ: Update step (6) is replaced with the natural gradient ascent with gradient normalization and step-size ϵ_θ,

$$\theta^{t+1} = \theta^t + \epsilon_\theta \tilde{\nabla}_\theta J(\boldsymbol{x}^t, \theta^t) \tag{10}$$

$$\epsilon_\theta = \delta_\theta / \| \tilde{\nabla}_\theta J(\boldsymbol{x}^t, \theta^t) \|_{\mathbf{F}(\theta^t)} \enspace , \tag{11}$$

where $\tilde{\nabla}_\theta = \mathbf{F}(\theta^t)^{-1} \nabla_\theta$. It can be approximately understood as the trust region method under the Kullback-Leibler (KL-) divergence with trust region radius δ_θ.

As the natural gradient is not analytically obtained, we use MC to obtain its approximation

$$G_\theta(\boldsymbol{x}^{t+1}, \theta^t) = \frac{1}{\lambda_\theta} \sum_{i=1}^{\lambda_\theta} f(\boldsymbol{x}^{t+1}, \boldsymbol{c}_i)(T(\boldsymbol{c}_i) - \theta^t) \enspace , \tag{12}$$

where $\boldsymbol{c}_i$ are i.i.d. from P_{θ^t}. The parameter update follows

$$\theta^{t+1} = \theta^t + \epsilon_\theta G_\theta(\boldsymbol{x}^{t+1}, \theta^t) \tag{13}$$

$$\epsilon_\theta = \delta_\theta / \| G_\theta(\boldsymbol{x}^{t+1}, \theta^t) \|_{\mathbf{F}(\theta^t)} \enspace . \tag{14}$$

2.3. Adaptive Stochastic Natural Gradient

In general, the step-size of a stochastic gradient algorithm plays one of the most important roles in performance and optimization time. Different adaptive step-size mechanisms have been proposed such as Adam (Kingma & Ba, 2015). However, our preliminary empirical study shows a specific adaptation mechanism for ϵ_θ is required to have robust performance. In the following, we first investigate the theoretical properties of the stochastic natural gradient ascent introduced above, then we introduce an adaptation mechanism for the trust-region radius δ_θ.

Theoretical Background: For problems without $\boldsymbol{x}$, i.e., fully black-box optimization of $f(\boldsymbol{c})$, the natural gradient ascent (10) of the stochastic relaxation (2) of function $f(\boldsymbol{c})$ is known as the *information geometric optimization* (IGO) (Ollivier et al., 2017) algorithm. For the case of exponential family with expectation parameters, Akimoto & Ollivier (2013) have shown that (10) leads to a monotone increase of $J(\theta)$, summarized as follows.[1]

Proposition 2 (Theorem 12 of (Akimoto & Ollivier, 2013))**.** *Assume that* $\min_{\boldsymbol{c} \in \mathcal{C}} f(\boldsymbol{c}) > 0$*. Then,* (10) *satisfies*

$$\begin{aligned} &\ln J(\theta^t + \epsilon_\theta \tilde{\nabla}_\theta J(\theta^t)) - \ln J(\theta^t) \\ &\quad \geq ((\epsilon_\theta J(\theta^t))^{-1} - 1) D_\theta(\theta^t + \epsilon_\theta \tilde{\nabla}_\theta J(\theta^t), \theta^t) \enspace , \end{aligned} \tag{15}$$

where D_θ is KL-divergence on Θ.

Proposition 2 gives us a very useful insight into the step-size ϵ_θ. It says that $\epsilon_\theta < 1/J(\theta^t)$ leads to improvement in J value as long as the parameter follows the *exact* natural gradient. Together with Proposition 1, it implies the monotone improvement of alternating update of $\boldsymbol{x}$ and θ when the exact gradients are given. However, in our situation, the natural gradient in (10) is not tractable and one needs to approximate it with MC. Then, the monotone improvement is not guaranteed. A promising feature of our framework is that the MC approximate $G_\theta(\boldsymbol{x}^{t+1}, \theta^t)$ of the natural gradient $\tilde{\nabla}_\theta J(\theta^t)$ can be made arbitrarily accurate by taking the number of MC samples λ_θ to ∞.

The following proposition shows that the loss in J is bounded if the divergence is bounded. Its proof can be found in supplementary material.

Proposition 3. *Assume that* $\min_{\boldsymbol{c} \in \mathcal{C}} f(\boldsymbol{c}) > 0$ *and let* $f^* = \max_{\boldsymbol{c} \in \mathcal{C}} f(\boldsymbol{c})$*. Then,* $\ln J(\theta') - \ln J(\theta) \geq -\frac{f^*}{J(\theta)} D_\theta(\theta', \theta)$ *for any* θ *and* θ'*.*

As a straight-forward consequence of the above two propositions, we obtain a sufficient conditions for the stochastic natural gradient ascent to improve J monotonically. This is the baseline of our proposal.

Theorem 4. *Assume that* $\min_{\boldsymbol{c} \in \mathcal{C}} f(\boldsymbol{c}) > 0$ *and let* $f^* = \max_{\boldsymbol{c} \in \mathcal{C}} f(\boldsymbol{c})$*. For any* $\epsilon > 0$*, if* $D_\theta(\theta, \theta^t + \epsilon \tilde{\nabla}_\theta J(\theta^t)) \leq \zeta D_\theta(\theta^t + \epsilon \tilde{\nabla}_\theta J(\theta^t), \theta^t)$ *holds for some* $\zeta > 0$*, we have*

$$\begin{aligned} &\ln J(\theta) - \ln J(\theta^t) \\ &\quad \geq \frac{1 - \zeta \epsilon f^* - \epsilon J(\theta^t)}{\epsilon J(\theta^t)} D_\theta(\theta^t + \epsilon \tilde{\nabla}_\theta J(\theta^t), \theta^t) \enspace . \end{aligned} \tag{16}$$

In particular, if $\epsilon < (\zeta f^* + J(\theta^t))^{-1}$ *holds,* $J(\theta) > J(\theta^t)$*.*

If we replace θ with θ^{t+1} defined in (13), we obtain a sufficient condition for the stochastic natural gradient update

[1] The statement is simplified so as not to introduce additional notation. Note that if $f(\boldsymbol{c})$ is lower bounded, considering $f(\boldsymbol{c}) - \min_{\boldsymbol{c} \in \mathcal{C}} f(\boldsymbol{c})$ in (2) instead of f is sufficient to meet the condition of Proposition 2. This modification only adds an offset to the J value without affecting the gradient.

(13) to lead to monotone improvement, namely,

$$D_\theta(\theta^{t+1}, \theta^t + \epsilon_\theta \tilde{\nabla}_\theta J(\boldsymbol{x}^{t+1}, \theta^t)) \le \zeta D_\theta(\theta^t + \epsilon_\theta \tilde{\nabla}_\theta J(\boldsymbol{x}^{t+1}, \theta^t), \theta^t) \quad . \tag{17}$$

This can be satisfied for any $\zeta > 0$ by taking a sufficiently large λ_θ as $G_\theta(\boldsymbol{x}^{t+1}, \theta^t)$ is a consistent estimator of $\tilde{\nabla}_\theta J(\boldsymbol{x}^{t+1}, \theta^t)$ and the left hand side (LHS) is $O(\lambda_\theta^{-1})$.

However, if ϵ_θ (or δ_θ) is sufficiently small, monotone improvement at each iteration is too strict and one might only need to guarantee the improvement over $\tau > 0$ iterations, where $\tau \propto 1/\delta_\theta$. To derive an insightful formula, we put aside the mathematical rigor in the following. Let $\tilde{\nabla}^t_{\lambda_\theta} = G_\theta(\boldsymbol{x}^{t+1}, \theta^t)$ for short. We continue to consider a problem without $\boldsymbol{x}$ (or $\boldsymbol{x}$ is fixed). A common argument borrowed from stochastic approximation (e.g., Borkar (2008)) states that if ϵ_θ is so small that the parameter vector stays near θ^t and $\tilde{\nabla}^{t+i}_{\lambda_\theta}$ are considered i.i.d. for $i = 0, \dots, \tau - 1$, the parameter vector after τ steps will be approximated as

$$\theta^{t+\tau} - \theta^t \approx \epsilon_\theta \tau \mathbb{E}[\tilde{\nabla}^t_{\lambda_\theta}] + \epsilon_\theta \tau \sum_{i=0}^{\tau-1} \frac{1}{\tau} (\tilde{\nabla}^{t+i}_{\lambda_\theta} - \mathbb{E}[\tilde{\nabla}^t_{\lambda_\theta}]) \quad .$$

If we replace θ^{t+1} with $\theta^{t+\tau}$ and ϵ with $\tau\epsilon_\theta$ in (17) and apply the approximation of the KL-divergence by the Fisher information matrix, we obtain

$$\underbrace{\left\| \sum_{i=0}^{\tau-1} \frac{\tilde{\nabla}^{t+i}_{\lambda_\theta} - \mathbb{E}[\tilde{\nabla}^t_{\lambda_\theta}]}{\sqrt{\tau}} \right\|^2_{\mathbf{F}(\theta^t)}}_{\to \mathrm{Tr}(\mathrm{Cov}(\tilde{\nabla}^t_{\lambda_\theta})\mathbf{F}(\theta^t)) \text{ as } \tau \to \infty} \le \zeta \tau \|\mathbb{E}[\tilde{\nabla}^t_{\lambda_\theta}]\|^2_{\mathbf{F}(\theta^t)} \quad ,$$

The LHS tends to the variance of $\tilde{\nabla}^t_{\lambda_\theta}$ measured w.r.t. the Fisher metric and is upper bounded by $(f^*)^2 n_\theta / \lambda_\theta$. That is, λ_θ and/or δ_θ should be adapted so that

$$\frac{\|\mathbb{E}[\tilde{\nabla}^t_{\lambda_\theta}]\|^2_{\mathbf{F}(\theta^t)}}{\mathrm{Tr}(\mathrm{Cov}(\tilde{\nabla}^t_{\lambda_\theta})\mathbf{F}(\theta^t))} \ge \frac{1}{\zeta\tau} \in \Omega(\delta_\theta) \quad . \tag{18}$$

In words, the signal-to-noise ratio (LHS of (18)) must be greater than a constant proportional to δ_θ.

Adaptive Stochastic Natural Gradient: We develop an algorithm that approximately satisfies the above-mentioned condition by adapting the trust region δ_θ. The above condition can be satisfied by increasing λ_θ while δ_θ is fixed, and the same idea as described below can be used to adapt the number λ_θ of MC samples. The reason we adapt δ_θ rather than λ_θ is to update connection weights $\boldsymbol{x}$ more frequently ($\boldsymbol{x}$ is updated after every λ_θ forward network processes). If multiple GPUs are available, one can set $\lambda_\theta = \lambda_{\boldsymbol{x}} = \#$GPUs and enjoy parallel computation, allowing to keep $\epsilon_{\boldsymbol{x}}$ and δ_θ (hence ϵ_θ as well) higher as the stochastic gradient becomes more reliable.

Algorithm 1 ASNG-NAS

```
Require: $\boldsymbol{x}^0$, $\theta^0$ {initial search points}
Require: $\alpha = 1.5$, $\delta^0_\theta = 1$, $\lambda_{\boldsymbol{x}} = \lambda_\theta = 2$
$\Delta = 1, \gamma = 0, \boldsymbol{s} = \boldsymbol{0}, t = 0$
repeat
    $\delta_\theta = \delta^0_\theta / \Delta$, $\beta = \delta_\theta / n_\theta^{1/2}$
    compute $G_{\boldsymbol{x}}(\boldsymbol{x}^t, \theta^t)$ by (9) and update $\boldsymbol{x}^{t+1}$ using $G_{\boldsymbol{x}}(\boldsymbol{x}^t, \theta^t)$
    compute $G_\theta(\boldsymbol{x}^{t+1}, \theta^t)$ by (12), update $\theta^{t+1}$ with (13), then force $\theta^{t+1} \in \Theta$ by projection
    $\boldsymbol{s} \leftarrow (1-\beta)\boldsymbol{s} + \sqrt{\beta(2-\beta)} \frac{\mathbf{F}(\theta^t)^{\frac{1}{2}} G_\theta(\boldsymbol{x}^{t+1}, \theta^t)}{\|G_\theta(\boldsymbol{x}^{t+1}, \theta^t)\|_{\mathbf{F}(\theta^t)}}$
    $\gamma \leftarrow (1-\beta)^2 \gamma + \beta(2-\beta)$
    $\Delta \leftarrow \min(\Delta_{\max}, \Delta \exp(\beta(\gamma - \|\boldsymbol{s}\|^2/\alpha)))$
until termination conditions are met
```

We introduce the accumulation of the stochastic natural gradient as follows

$$\boldsymbol{s}^{(t+1)} = (1-\beta)\boldsymbol{s}^{(t)} + \sqrt{\beta(2-\beta)} \mathbf{F}(\theta^t)^{\frac{1}{2}} \tilde{\nabla}^t_{\lambda_\theta} \quad , \tag{19}$$

$$\gamma^{(t+1)} = (1-\beta)^2 \gamma^{(t)} + \beta(2-\beta) \|\tilde{\nabla}^t_{\lambda_\theta}\|^2_{\mathbf{F}(\theta^t)} \quad , \tag{20}$$

where $\boldsymbol{s}^{(0)} = \boldsymbol{0}$ and $\gamma^{(0)} = 0$. To understand the effect of $\boldsymbol{s}$ and γ, we consider the situation that $\epsilon_{\boldsymbol{x}}$ and δ_θ are small enough that $\boldsymbol{x}^t$ and θ^t stay at $(\boldsymbol{x}, \theta)$. Then, $\boldsymbol{s}^t$ approaches $\sqrt{(2-\beta)/\beta}\mathbb{E}[\mathbf{F}(\theta^t)^{\frac{1}{2}} \tilde{\nabla}_{\lambda_\theta}] + \xi$, where ξ is a random vector with $\mathbb{E}[\xi] = \boldsymbol{0}$ and $\mathrm{Cov}(\xi) = \mathbf{F}(\theta^t)^{\frac{1}{2}} \mathrm{Cov}(\tilde{\nabla}_{\lambda_\theta}) \mathbf{F}(\theta^t)^{\frac{1}{2}}$, and γ^t approximates $\mathbb{E}[\|\tilde{\nabla}_{\lambda_\theta}\|^2_{\mathbf{F}(\theta^t)}] = \|\mathbb{E}[\tilde{\nabla}_{\lambda_\theta}]\|^2_{\mathbf{F}(\theta^t)} + \mathrm{Tr}(\mathrm{Cov}(\tilde{\nabla}^t_{\lambda_\theta})\mathbf{F}(\theta^t))$. If we set $\beta \propto \delta_\theta$ and adapt λ_θ or δ_θ to keep $\|\boldsymbol{s}^{(t+1)}\|^2 / \gamma^{(t+1)} \ge \alpha$ for some $\alpha > 1$, it approximately achieves

$$\frac{\|\mathbb{E}[\tilde{\nabla}_{\lambda_\theta}]\|^2_{\mathbf{F}(\theta^t)}}{\mathrm{Tr}(\mathrm{Cov}(\tilde{\nabla}^t_{\lambda_\theta})\mathbf{F}(\theta^t))} \ge \frac{\|\mathbb{E}[\tilde{\nabla}_{\lambda_\theta}]\|^2_{\mathbf{F}(\theta^t)}}{\mathbb{E}[\|\tilde{\nabla}_{\lambda_\theta}\|^2_{\mathbf{F}(\theta^t)}]}$$

$$\approx \frac{\beta}{2-2\beta}\left(\frac{\|\boldsymbol{s}^{(t+1)}\|^2}{\gamma^{(t+1)}} - 1\right) \ge \frac{\beta(\alpha-1)}{2-2\beta} \in \Theta(\delta_\theta) \quad .$$

It results in satisfying (18).

The adaptation of δ_θ is then done as follows:

$$\delta_\theta \leftarrow \delta_\theta \exp\left(\beta\left(\|\boldsymbol{s}^{(t+1)}\|^2/\alpha - \gamma^{(t+1)}\right)\right) \quad . \tag{21}$$

This tries to keep $\|\boldsymbol{s}^{(t+1)}\|^2/\gamma^{(t+1)} \approx \alpha$ by adapting δ_θ.

2.4. Adaptive Stochastic Natural Gradient-based NAS

The proposed optimization method for problem (1), called Adaptive Stochastic Natural Gradient-based NAS (ASNG-NAS), is summarized in Algorithm 1. Here, we summarize some implementation remarks. One is that instead of accumulating $\mathbf{F}(\theta^t)^{\frac{1}{2}} \tilde{\nabla}^t_{\lambda_\theta}$ and $\|\tilde{\nabla}^t_{\lambda_\theta}\|^2_{\mathbf{F}(\theta^t)}$ separately in (19) and (20), we accumulate $\mathbf{F}(\theta^t)^{\frac{1}{2}}\tilde{\nabla}^t_{\lambda_\theta}/\|\tilde{\nabla}^t_{\lambda_\theta}\|_{\mathbf{F}(\theta^t)}$ in $\boldsymbol{s}$ and $\gamma \approx 1$. In our preliminary experiments, we found it more stable. The other point is that the average function value is subtracted from the function value in the stochastic natural gradient computation (12) when $\lambda_\theta = 2$. This is a well-known technique to reduce the estimation variance of gradient while the expectation is unchanged (e.g., Evans & Swartz (2000)). Since we normalize the stochastic natural gradient when the parameter is updated, it is equivalent to transform $f_1 = f(\boldsymbol{x}^{t+1}, \boldsymbol{c}_1)$ and $f_2 = f(\boldsymbol{x}^{t+1}, \boldsymbol{c}_2)$ to (1, -1) if $f_1 > f_2$, (-1, 1) if $f_1 < f_2$, and (0, 0) if $f_1 = f_2$ (in this case, we skip the update and start the next iteration). When $\lambda_\theta > 2$, we similarly transform $f_i = f(\boldsymbol{x}^{t+1}, \boldsymbol{c}_i)$ in (12) to 1 if f_i is in top $\lceil \lambda_\theta/4 \rceil$, -1 if it is in bottom $\lceil \lambda_\theta/4 \rceil$, and 0 otherwise. By doing so, we obtain invariance to a strictly increasing transformation of f, and we observed significant speedup in many cases in our preliminary study.

To instantiate ASNG-NAS, we prepare an exponential family defined on $\mathcal{C}$. If $\mathcal{C}$ is a set of categorical variables ($\mathcal{C} = [\![1, m_1]\!] \times \cdots \times [\![1, m_{n_c}]\!]$), one can simply use categorical distribution parameterized by the probability $[\theta]_{i,j} = [\theta_i]_j$ of i-th categorical variable to be j ($1 - \sum_{j=1}^{m_i - 1}[\theta]_{i,j}$ is the probability of $[\boldsymbol{c}]_i = m_i$). Then, $T(\boldsymbol{c}) = (T_1([\boldsymbol{c}]_1), \ldots, T_{n_c}([\boldsymbol{c}]_{n_c}))$, where $T_i : [\![1, m_i]\!] \to [0,1]^{m_i - 1}$ is the one-hot representation without the last element, and $\mathbf{F}(\theta) = \operatorname{diag}(\mathbf{F}_1(\theta_1), \ldots, \mathbf{F}_{n_c}(\theta_{n_c}))$, where $\mathbf{F}_i(\theta_i) = \operatorname{diag}(\theta_i)^{-1} + (1 - \sum_{j=1}^{m_i-1}[\theta_i]_j)^{-1}\mathbf{1}\mathbf{1}^{\mathrm{T}}$. If $\mathcal{C}$ is a set of ordinal variables, e.g., $\mathcal{C} \subseteq \mathbb{R}^{n_c}$, our choice will be $P_\theta = \mathcal{N}(\mu_1, \sigma_1^2) \times \cdots \times \mathcal{N}(\mu_{n_c}, \sigma_{n_c}^2)$ and $\theta = (\mu_1, \mu_1^2 + \sigma_1^2, \ldots, \mu_{n_c}, \mu_{n_c}^2 + \sigma_{n_c}^2)$. Then, we have $T(\boldsymbol{c}) = (T_1([\boldsymbol{c}]_1), \ldots, T_{n_c}([\boldsymbol{c}]_{n_c}))$ with $T_i([\boldsymbol{c}]_i) = ([\boldsymbol{c}]_i, [\boldsymbol{c}]_i^2)$, and $\mathbf{F}(\theta)$ is a block-diagonal matrix with block size 2 whose i-th block is $[\sigma_i^2, 2\mu_i\sigma_i^2; 2\mu_i\sigma_i^2, 4\mu_i^2\sigma_i^2 + 2\sigma_i^4]^{-1}$. Integer variables can be treated similarly. If $\mathcal{C}$ is a product of categorical and ordinal variable spaces, we can use their product distribution. A desired θ^0 realizes the maximal entropy in Θ unless one has a prior knowledge. Moreover, Θ should be restricted to avoid degenerated distribution. E.g., for categorical distribution, we lower bounds $[\theta]_{i,j}$ by $\theta_i^{\min} = (n_{\boldsymbol{c}}(m_i - 1))^{-1}$. See the supplementary material.

3. Experiments and Results

We investigate the robustness of ASNG on an artificial test function in §3.1. We then apply ASNG-NAS to the architecture search for image classification and inpainting in §3.2 and §3.3. To compare the quality of the obtained architecture and the computational cost, we adopt the same search spaces as in previous works. The experiments were done with a single NVIDIA GTX 1080Ti GPU, and ASNG-NAS is implemented using `PyTorch 0.4.1` (Paszke et al., 2017). The code is available at `https://github.com/shirakawas/ASNG-NAS`.

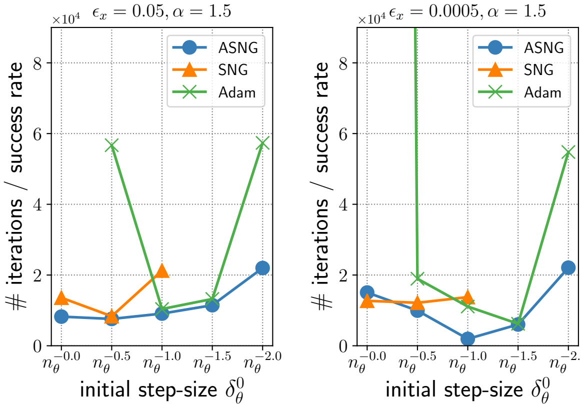

Figure 1: Results on the selective squared error function for $\epsilon_{\boldsymbol{x}} = 0.05$ and 0.0005. Median values over 100 runs are reported. Missing values indicate the setting never succeeded.

3.1. Toy Problem

We consider the following *selective squared error function* composed of continuous variable $\boldsymbol{x} \in \mathbb{R}^{D \times K}$ and categorical variables $\boldsymbol{c}$. For each i ($1 \leq i \leq D$), we denote the one-hot vector of i-th categorical variable in $\boldsymbol{c}$ by $h_i(\boldsymbol{c}) \in \{0,1\}^K$. The objective function to be minimized is

$$f(\boldsymbol{x}, \boldsymbol{c}) = \mathbb{E}_z\left[\sum_{i=1}^{D}\sum_{j=1}^{K} h_{ij}(\boldsymbol{c})\left((x_{ij} - z_i)^2 + \frac{j-1}{K}\right)\right] ,$$

where the underlying distribution of z is $\mathcal{N}(0, K^{-2}I)$. This function switches the active variables in $\boldsymbol{x}$ by the categorical variables $\boldsymbol{c}$. The global optima locate at $h_i(\boldsymbol{c}) = [1, 0, \ldots, 0]$ for $i = 1, \ldots, D$, $\boldsymbol{x}_1 = [0, \ldots, 0]$, and arbitrary variables of $\boldsymbol{x}_i$ for $i = 2, \ldots, D$. To mimic NN training, we approximate the expectation by using a sample $\boldsymbol{z}$, which is drawn from $\mathcal{N}(0, K^{-2}I)$ every parameter update.

We use the stochastic gradient descent (SGD) with a momentum of 0.9 to optimize $\boldsymbol{x}$ and anneal the step-size $\epsilon_{\boldsymbol{x}}$ by cosine scheduling (Loshchilov & Hutter, 2017), which we also use for the latter experiments. We initialize $\boldsymbol{x} \sim \mathcal{N}(0, I)$ and the distribution parameter $\theta = (1/K)\mathbf{1}$. We regard it successfully optimized if a solution with the actual objective value less than $K^{-1} + DK^{-2}$ is sampled before 10^5 iterations. We report the number of iterations to sample the target value divided by the success rate over 100 runs as the performance measure. We have tested different combinations of D and K and observed similar results. We report the results for $D = 30$ and $K = 5$ as a typical one. Figure 1 compares ASNG, SNG (stochastic natural gradient with constant step-size), and Adam (Kingma & Ba, 2015)—a standard step-size adaptation for DNNs—with different initial step-size. We replace the gradient in Adam with the normalized natural gradient as it is used in ASNG since we found in our preliminary studies that Adam does not work properly with its default. For SNG and Adam one needs to fine tune the step-size, otherwise they fail to locate the optimum. On the other hand, ASNG relaxes the sensitivity against δ_θ^0. The robustness of ASNG on the choice of α is evaluated in the supplementary material.

3.2. Architecture Search for Image Classification

Dataset: We use the CIFAR-10 dataset and adopt the standard preprocessing and data augmentation as done in the previous works, e.g., Liu et al. (2019); Pham et al. (2018). During the architecture search, we split the training dataset into halves as $\mathcal{D} = \{\mathcal{D}_{\boldsymbol{x}}, \mathcal{D}_\theta\}$ as done in Liu et al. (2019). The gradients (9) and (12) are calculated using mini-batches from $\mathcal{D}_{\boldsymbol{x}}$ and $\mathcal{D}_\theta$, respectively. We use the same mini-batch samples among the different architecture parameters in (9) and (12) to get accurate gradients. Note that we do not need the back-propagation for calculating (12). Namely, the computational cost of the θ update is less than that of $\boldsymbol{x}$.

Search Space: The search space is based on the one in Pham et al. (2018), which consists of models obtained by connecting two motifs (called normal cell and reduction cell) repeatedly. Each cell consists of B $(= 5)$ nodes and receives the outputs of the previous two cells as inputs. Each node receives two inputs from previous nodes, applies an operation to each of the inputs, and adds them. Our search space includes 5 operations: identity, 3×3 and 5×5 separable convolutions (Chollet, 2017), and 3×3 average and max poolings. The separable convolutions are applied twice in the order of ReLU-Conv-BatchNorm. We select a node by 4 categorical variables representing 2 outputs of the previous nodes and 2 operations applied to them. Consequently, we treat $4B$-dimensional categorical variables for each cell. After deciding B nodes, all of the unused outputs of the nodes are concatenated as the output of the cell. The number of the categorical variables is $n_{\boldsymbol{c}} = 40$, and the dimension of θ becomes $n_\theta = 140$.

Training Details: In the architecture search phase, we optimize $\boldsymbol{x}$ and θ for 100 epochs (about 40K iterations) with a mini-batch size of 64. We stack 2 normal cells $(N = 2)$ and set the number of channels at the first cell to 16. We use SGD with a momentum of 0.9 to optimize weights $\boldsymbol{x}$. The step-size $\epsilon_{\boldsymbol{x}}$ changes from 0.025 to 0 following the cosine schedule (Loshchilov & Hutter, 2017). After the architecture search phase, we *retrain* the network with the most likely architecture, $\hat{\boldsymbol{c}} = \mathrm{argmax}_{\boldsymbol{c}}\, p_\theta(\boldsymbol{c})$, from scratch, which is a commonly used technique (Brock et al., 2018; Liu et al., 2019; Pham et al., 2018) to improve final performance. In the retraining stage, we can exclude the redundant (unused) weights. Then, we optimize $\boldsymbol{x}$ for 600 epochs with a mini-batch size of 80. We stack 6 normal cells $(N = 6)$ and increase the number of channels at the first cell so that the model has the nearly equal number of weight parameters to 4 million. We report the average (avg.) and standard deviation (std.) among 3 independent experiments.

Result and Discussion: Table 1 compares the search cost and the test error of different NAS methods. The bottom 5 methods adopt similar search spaces, hence showing the performance differences due to search algorithms. The avg. and std. of ASNG-NAS are those of architecture search + retraining (whole NAS process), whereas the values for the other methods are taken from the references and have different meanings. E.g., the values for DARTS and SNAS are the avg. and std. of 10 independent *retraining* of the best found architecture among 4 NAS processes.

We clearly see the trade-off between search cost and final performance. The more accurate the performance estimation of neural architecture is (as in NASNet and NAONet), the better final performance is obtained at the risk of speed. Among relatively fast NAS methods (ENAS, DARTS, SNAS, and ASNG-NAS), ASNG-NAS is the fastest and achieves a competitive error rate. The reason of speed difference between these algorithms is discussed in §4. We observed that the probability vector θ of the categorical distribution converges to a certain category. More precisely, the average value of the $\max_j [\theta]_{i,j}$ reaches around 0.9 at the 50th epoch. The architecture of the best model obtained by ASNG-NAS is found in supplementary material.

Figure 2 compares the test error w.r.t. elapsed time in the architecture search phase. The test accuracy of the most likely architecture $\hat{\boldsymbol{c}}$ is plotted for ASNG-NAS. DARTS (mix) and DARTS (fix) are the architectures obtained by the same run of DARTS. The former is the one mixing all possible operations with real-valued structure parameters and is the one optimized during the architecture search, whereas the latter is the one that takes the operations with the highest weights and is the one used after the architecture search. We see that DARTS (fix) do not improve the test accuracy during the architecture search phase and the retraining is a must. DARTS (mix) achieves better performance than ASNG-NAS in the end, but the obtained architecture is not one-hot and is computationally expensive. ASNG-NAS shows the best performance for small time budgets.

3.3. Architecture Search for Inpainting

Dataset: We use the CelebFaces Attributes Dataset (CelebA) (Liu et al., 2015). The data preprocessing and augmentation method is the same as Suganuma et al. (2018). We use three different masks to generate images with missing regions; a central square block mask (Center); a random pixel mask where 80% of the pixels were randomly masked (Pixel), and a half image mask where either the vertical or horizontal half of the image is randomly selected (Half). Following Suganuma et al. (2018), we use two standard eval-

Table 1: Comparison of different architecture search methods on CIFAR-10. The search cost indicates GPU days for architecture search excluding the retraining cost.

Method	Search Cost (GPU days)	Params (M)	Test Error (%)
NASNet-A (Zoph et al., 2018)	1800	3.3	2.65
NAONet (Luo et al., 2018)	200	128	2.11
ProxylessNAS-G (Cai et al., 2019)	4	5.7	2.08
SMASHv2 (Brock et al., 2018)	1.5	16.0	4.03
DARTS second order (Liu et al., 2019)	4	3.3	2.76 (±0.09)
DARTS first order (Liu et al., 2019)	1.5	3.3	3.00 (±0.14)
SNAS (Xie et al., 2019)	1.5	2.8	2.85 (±0.02)
ENAS (Pham et al., 2018)	0.45	4.6	2.89
ASNG-NAS	0.11	3.9	2.83 (±0.14)

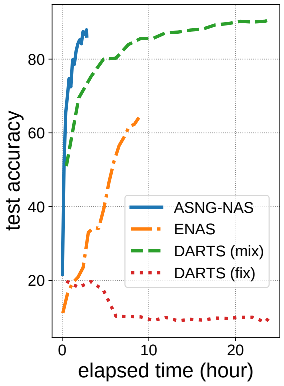

Figure 2: Transitions of test error against elapsed time in the architecture search phase.

uation measures: the peak-signal to noise ratio (PSNR) and the structural similarity index (SSIM) (Wang et al., 2004) to evaluate the restored images. Higher values of these measures indicate a better image restoration.

Search Space: The search space we use is based on Suganuma et al. (2018) for comparison. The architecture encoding is slightly different but it can represent the exact same network architectures. We employ the symmetric convolutional autoencoder (CAE) as a base architecture. A skip connection between the convolutional layer and the mirrored deconvolution layer can exist. We prepare six types of layers: the combination of the kernel sizes $\{1\times 1, 3\times 3, 5\times 5\}$ and the existence of the skip connection. The layers with different settings do not share weight parameters.

We implement two ASNG-NAS algorithms with only categorical variables (ASNG-NAS (Cat)) and with mixed categorical and ordinal (integer) variables (ASNG-NAS (Int)) to demonstrate the flexibility of the proposed approach. The former encodes the layer type, channel size, and connections for each hidden layer, and the connection for the output layer using categorical variables. We select the output channel size of each of 20 hidden layers from $\{64, 128, 256\}$. The latter encodes the kernel size and the channel size by integers in $[\![1, 3]\!]$ (corresponding to $\{1 \times 1, 3 \times 3, 5 \times 5\}$) and $[\![64, 256]\!]$. We employ the Gaussian distribution as described in §2.4. Sampled variables are clipped to $[1, 3]$ and $[64, 256]$ and rounded to integers (only for architecture evaluation). The dimension of θ amounts to $n_\theta = 214$ for ASNG-NAS (Cat) and $n_\theta = 174$ for ASNG-NAS (Int).

Training Details: We use the mean squared error (MSE) as the loss function and a mini-batch size of 16. In the architecture search phase, we use SGD with momentum with the same setting in §3.2, while we use Adam in the retraining phase. We apply gradient clipping with the norm of 5 to prevent a too long gradient step. The maximum numbers of iterations are $50K$ and $500K$ in the architecture search and retraining phases, respectively. The setting of the retraining is the same as in Suganuma et al. (2018). Differently from the previous experiment, we retrain the obtained architecture without any change in this experiment.

Result and Discussion: Table 2 shows the comparison of PSNR and SSIM. The performances of ASNG-NAS are better than CE, SII, and BASE on all mask types and comparable to E-CAE. Suganuma et al. (2018) reported that E-CAE spent approximately 12 GPU days (3 days with 4 GPUs) for the architecture search and retraining. On the other hand, the average computational times of ASNG-NAS were less than 1 GPU days. ASNG-NAS (Cat) took approximately 6 hours for the architecture search and 14 hours for the retraining on average, whereas the average retraining time of ASNG-NAS (Int) was reduced to 11 hours. This is because the architectures obtained by ASNG-NAS (Int) tended to have a small number of channels compared to ASNG-NAS (Cat) that selects from the predefined three channel sizes.

In conclusion, ASNG-NAS achieved practically significant speedup over E-CAE without compromising the final performance. The flexibility of ASNG-NAS has been shown as well. The capability of ASNG-NAS to treat mixed categorical and ordinal variables potentially decreases the number of the architecture parameters (good for speed) and enlarges the search space (good for performance).

4. Related Work and Discussion

ASNG-NAS falls into one-shot NAS. On this line of the research, three different existing approaches are reported. The 1st category is based on a meta-network. SMASH (Brock et al., 2018) employs HyperNet that takes an architecture

Table 2: Results on the inpainting tasks. CE and SII indicate the context encoder (Pathak et al., 2016) and the semantic image inpainting (Yeh et al., 2017), which are the human-designed CNN. E-CAE refers to the model obtained by the architecture search method using the evolutionary algorithm (Suganuma et al., 2018). BASE is the same depth of the best architecture obtained by E-CAE but having 64 channels and 3×3 filters in each layer, along with a skip connection. ASNG-NAS (Cat) encodes all architecture parameters into categorical variables, whereas ASNG-NAS (Int) encodes the kernel and channel sizes into integer variables. The values of CE, SII, BASE, and E-CAE are referenced from Suganuma et al. (2018).

	PSNR [dB] / SSIM					
Mask	CE	SII	BASE	E-CAE (12 GPU days)	ASNG-NAS (Cat) (0.84 GPU days)	ASNG-NAS (Int) (0.75 GPU days)
Center	28.5 / 0.912	19.4 / 0.907	27.1 / 0.883	**29.9 / 0.934**	29.2 / 0.903	29.3 / 0.911
Pixel	22.9 / 0.730	22.8 / 0.710	27.5 / 0.836	27.8 / 0.887	28.4 / 0.905	**28.6 / 0.909**
Half	19.9 / 0.747	13.7 / 0.582	11.8 / 0.604	**21.1** / 0.771	20.5 / **0.779**	20.6 / **0.779**

$\boldsymbol{c}$ as its input and returns the weights for the network with architecture $\boldsymbol{c}$. The weights of HyperNet is then optimized by backprop while $\boldsymbol{c}$ is randomly chosen during architecture search. ENAS (Pham et al., 2018) employs a recurrent neural network (RNN) to generate a sequence of categorical variables $\boldsymbol{c}$ representing neural architecture. It optimizes the weights and the RNN weights alternatively. ENAS and ASNG-NAS are different in that the latter directly introduces a probability distribution behind $\boldsymbol{c}$ while the former employ RNN. The advantage of ASNG-NAS over meta-network based approaches is that we do not need to design the architecture of a meta-network, which may be a tedious task for practitioners.

The 2nd category is based on continuous relaxation. DARTS (Liu et al., 2019) extends essentially categorical architecture parameters (selection of operations and connections) to a real-valued vector by considering a linear combination of outputs of all possible operations. This enables gradient descent both on the connection weights and the weights for the linear combination. This seminal work is followed by further improvements (Xie et al., 2019; Cai et al., 2019). An advantage of ASNG-NAS is that DARTS requires to compute all possible operations and connections to perform backprop, whereas we only require to process sub-networks with sampled architectures $\boldsymbol{c}$, hence ASNG-NAS is faster. This advantage is reflected in Table 1. Another advantage is its flexibility in the sense that the continuous relaxation of DARTS requires the output of all possible operations to live in the same domain to add them.

The last category is based on stochastic relaxation, which is another approach enabling to use gradient descent. Shirakawa et al. (2018) has introduced it to model connections and types of activation functions in multi-layer perceptrons. They are encoded by a binary vector $\boldsymbol{c}$ and Bernoulli distribution is considered as the underlying distribution of $\boldsymbol{c}$. The probability parameters of Bernoulli distribution is updated by SNG. We improve their work in the following directions: generalization to arbitrary architecture parameters (categorical, ordinal, or their mixture), theoretical investigation of monotone improvement, robustness against its input parameter by introducing a step-size adaptation mechanism.

This paper focused on the optimization framework for NAS. One can easily incorporate a different search space and a different performance estimation method into our framework. The step-size adaptation mechanism eases hyper-parameter tuning when different components are introduced. The ability to treat ordinal variables such as the number and size of filters and the number of layers accepts more flexible search space. In existing studies they are modeled by categorical variables by choosing a few representative numbers beforehand. Moreover, the ordinal variables potentially decreases the dimension of architecture parameters. When multiple GPUs are available, ASNG-NAS can easily enjoy them by increasing $\lambda_{\boldsymbol{x}}$ and $\lambda_{\boldsymbol{c}}$ and distributing them. In our preliminary study, we found that the larger they are, the greater step-size are allowed and the step-size adaptation automatically increases it. Our simple formulation allows theoretical investigation, which we think is missing in the current NAS research fields. Further theoretical investigation will contribute better understanding and further improvement of NAS.

One-shot NAS including our method does not optimize parameters involved in learning process such as the step-size for weight update. It is because their effects do not appear in the one-shot loss and will not be optimized effectively. If we employ hyper-parameter optimizers such as Bayesian optimization to optimize these parameters while each training process is replaced by our method, both architectures and other hyper-parameters could be optimized. The fast and robust properties of our method will be useful to combine one-shot NAS and hyper-parameter optimizer. This is an important direction towards automation of deep learning.

Acknowledgement

This work is partially supported by the SECOM Science and Technology Foundation.

References

Akimoto, Y. and Ollivier, Y. Objective improvement in information-geometric optimization. In *FOGA XII '13: Proceedings of the twelfth workshop on Foundations of genetic algorithms XII*, pp. 1–10. ACM, jan 2013.

Amari, S. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276, 1998.

Borkar, V. S. Stochastic approximation: a dynamical systems viewpoint. Cambridge University Press, 2008.

Brock, A., Lim, T., Ritchie, J., and Weston, N. SMASH: One-Shot Model Architecture Search through HyperNetworks. In *International Conference on Learning Representations (ICLR)*, 2018.

Cai, H., Zhu, L., and Han, S. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *International Conference on Learning Representations (ICLR)*, 2019.

Chollet, F. Xception: Deep Learning with Depthwise Separable Convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1800–1807, 2017.

DeVries, T. and Taylor, G. W. Improved Regularization of Convolutional Neural Networks with Cutout. *arXiv preprint:1708.04552*, 2017.

Elsken, T., Metzen, J. H., and Hutter, F. Neural Architecture Search: A Survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

Evans, M. J. and Swartz, T. *Approximating integrals via Monte Carlo and deterministic methods*. Oxford University Press, USA, 2000. ISBN 0198502788.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations (ICLR)*, 2019.

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep Learning Face Attributes in the Wild. In *IEEE International Conference on Computer Vision (ICCV)*, pp. 3730–3738, 2015.

Loshchilov, I. and Hutter, F. SGDR: Stochastic Gradient Descent with Warm Restarts. In *International Conference on Learning Representations (ICLR)*, 2017.

Luo, R., Tian, F., Qin, T., Chen, E., and Liu, T. Neural Architecture Optimization. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 7827–7838, 2018.

Mao, X., Shen, C., and Yang, Y. Image Restoration Using Very Deep Convolutional Encoder-Decoder Networks with Symmetric Skip Connections. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2802–2810, 2016.

Ollivier, Y., Arnold, L., Auger, A., and Hansen, N. Information-Geometric Optimization Algorithms: A Unifying Picture via Invariance Principles. *Journal of Machine Learning Research*, 18(1):564–628, 2017.

Paszke, A., Chanan, G., Lin, Z., Gross, S., Yang, E., Antiga, L., and Devito, Z. Automatic differentiation in PyTorch. In *Autodiff Workshop in Thirty-first Conference on Neural Information Processing Systems (NIPS)*, 2017.

Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., and Efros, A. A. Context Encoders: Feature Learning by Inpainting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2536–2544, 2016.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient Neural Architecture Search via Parameter Sharing. In *The 35th International Conference on Machine Learning (ICML)*, volume 80, pp. 4095–4104, 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 *The 34th International Conference on Machine Learning (ICML)*, volume 70, pp. 2902–2911, 2017.

Shirakawa, S., Iwata, Y., and Akimoto, Y. Dynamic Optimization of Neural Network Structures Using Probabilistic Modeling. In *Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*, pp. 4074–4082, 2018.

Suganuma, M., Shirakawa, S., and Nagao, T. A Genetic Programming Approach to Designing Convolutional Neural Network Architectures. In *The Genetic and Evolutionary Computation Conference (GECCO)*, pp. 497–504, 2017.

Suganuma, M., Ozay, M., and Okatani, T. Exploiting the Potential of Standard Convolutional Autoencoders for Image Restoration by Evolutionary Search. In *The 35th International Conference on Machine Learning (ICML)*, volume 80, pp. 4778–4787, 2018.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the Inception Architecture for Computer Vision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826, 2016.

Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, 13 (4):600–612, 2004.

Xie, S., Zheng, H., Liu, C., and Lin, L. SNAS: Stochastic Neural Architecture Search. In *International Conference on Learning Representations (ICLR)*, 2019.

Yeh, R. A., Chen, C., Lim, T. Y., Schwing, A. G., Hasegawa-Johnson, M., and Do, M. N. Semantic Image Inpainting with Deep Generative Models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6882–6890, 2017.

Zoph, B. and Le, Q. V. Neural Architecture Search with Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning Transferable Architectures for Scalable Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8697–8710, 2018.

A. Proof of Proposition 3

Proof. It follows

$$\begin{aligned}
&\ln J(\theta') - \ln J(\theta)\\
&= \ln \int \frac{p_{\theta'}(\boldsymbol{c})}{p_{\theta}(\boldsymbol{c})} \frac{f(\boldsymbol{c})}{J(\theta)} p_{\theta}(\boldsymbol{c})\\
&\geq \int \ln\left(\frac{p_{\theta'}(\boldsymbol{c})}{p_{\theta}(\boldsymbol{c})}\right) \frac{f(\boldsymbol{c})}{J(\theta)} p_{\theta}(\boldsymbol{c})\\
&\geq \int_{\boldsymbol{c}: p_{\theta'}(\boldsymbol{c}) < p_{\theta}(\boldsymbol{c})} \ln\left(\frac{p_{\theta'}(\boldsymbol{c})}{p_{\theta}(\boldsymbol{c})}\right) \frac{f(\boldsymbol{c})}{J(\theta)} p_{\theta}(\boldsymbol{c})\\
&\geq \frac{f^*}{J(\theta)} \int_{\boldsymbol{c}: p_{\theta'}(\boldsymbol{c}) < p_{\theta}(\boldsymbol{c})} \ln\left(\frac{p_{\theta'}(\boldsymbol{c})}{p_{\theta}(\boldsymbol{c})}\right) p_{\theta}(\boldsymbol{c})\\
&= -\frac{f^*}{J(\theta)} \int_{\boldsymbol{c}: p_{\theta'}(\boldsymbol{c}) < p_{\theta}(\boldsymbol{c})} \ln\left(\frac{p_{\theta}(\boldsymbol{c})}{p_{\theta'}(\boldsymbol{c})}\right) p_{\theta}(\boldsymbol{c})\\
&\geq -\frac{f^*}{J(\theta)} D_{\theta}(\theta', \theta) \enspace .
\end{aligned}$$

□

B. Derivation for Categorical Distribution

We derive the Fisher information matrix, its inverse and square root, and the natural gradient for the categorical distribution defined on $\mathcal{C} = \llbracket 1, m_1 \rrbracket \times \cdots \times \llbracket 1, m_{n_{\boldsymbol{c}}} \rrbracket$.

In our parameterization $\theta = (\theta_1, \dots, \theta_{n_{\boldsymbol{c}}})$, the probability of i-th ($i \in \llbracket 1, n_{\boldsymbol{c}} \rrbracket$) categorical variable $[\boldsymbol{c}]_i$ to be $j \in \llbracket 1, m_i - 1 \rrbracket$ is $[\theta]_{i,j} = [\theta_i]_j$, and $1 - \sum_{j=1}^{m_i - 1} [\theta]_{i,j}$ is the probability of $[\boldsymbol{c}]_i = m_i$. For the sake of simplicity, we denote $[\theta]_{i,m_i} = 1 - \sum_{j=1}^{m_i-1} [\theta]_{i,j}$. Then, $T(\boldsymbol{c}) = (T_1([\boldsymbol{c}]_1), \dots, T_{n_{\boldsymbol{c}}}([\boldsymbol{c}]_{n_{\boldsymbol{c}}}))$, where $T_i : \llbracket 1, m_i \rrbracket \to [0, 1]^{m_i - 1}$ is the one-hot representation without the last element. It is the exponential parameterization as $\theta = \mathbb{E}[T(\boldsymbol{c})]$.

The inverse of the Fisher information matrix simply follows from the formula for the exponential family: $\mathbf{F}(\theta)^{-1} = \mathbb{E}[(T(\boldsymbol{c}) - \theta)(T(\boldsymbol{c}) - \theta)^{\mathrm{T}}]$. It is a block diagonal matrix $\mathbf{F}(\theta)^{-1} = \operatorname{diag}(\mathbf{F}_1(\theta_1)^{-1}, \dots, \mathbf{F}_{n_{\boldsymbol{c}}}(\theta_{n_{\boldsymbol{c}}})^{-1})$, where $\mathbf{F}_i(\theta_i)^{-1} = \operatorname{diag}(\theta_i) - \theta_i \theta_i^{\mathrm{T}}$. Sherman-Morrison formula reads $\mathbf{F}_i(\theta_i) = \operatorname{diag}(\theta_i)^{-1} + (1 - \sum_{j=1}^{m_i-1} [\theta_i]_j)^{-1} \mathbf{1}\mathbf{1}^{\mathrm{T}}$ and we have $\mathbf{F}(\theta) = \operatorname{diag}(\mathbf{F}_1(\theta_1), \dots, \mathbf{F}_{n_{\boldsymbol{c}}}(\theta_{n_{\boldsymbol{c}}}))$.

As the Fisher information matrix is block-diagonal, and each block is of size $m_i - 1$, a naive computation of $\mathbf{F}(\theta)^{\frac{1}{2}}$ requires $O(\sum_{i=1}^{n_c} (m_i - 1)^3)$. This is usually not expensive as $n_{\boldsymbol{c}} \gg m_i$. An alternative way that we employ in this paper is to replace $\mathbf{F}(\theta)^{\frac{1}{2}}$ with a tractable factorization A with $\mathbf{F}(\theta) = AA^{\mathrm{T}}$. Our choice of A is the block-diagonal matrix whose i-th block is square, of size $m_i - 1$, and

$$A_i = \operatorname{diag}(\theta_i)^{-\frac{1}{2}} + \frac{1}{\sqrt{[\theta]_{i,m_i}} + [\theta]_{i,m_i}} \mathbf{1} \sqrt{\theta_i}^{\mathrm{T}} \enspace ,$$

where $\sqrt{\theta_i}$ is a vector whose j-th element is the square root of $[\theta]_{i,j}$. Then, the product of A and a vector can be computed in $O(\sum_{i=1}^{n_c} (m_i - 1))$. In our preliminary study we did not obverse any significant performance difference by this approximation.

C. Derivation for Gaussian Distribution

We derive the Fisher information matrix, its inverse and square root, and the natural gradient for the Gaussian distribution defined on $\mathcal{C} \subseteq \mathbb{R}^{n_{\boldsymbol{c}}}$.

Our choice is $P_\theta = \mathcal{N}(\mu_1, \sigma_1^2) \times \cdots \times \mathcal{N}(\mu_{n_{\boldsymbol{c}}}, \sigma_{n_{\boldsymbol{c}}}^2)$ and $\theta = (\mu_1, \mu_1^2 + \sigma_1^2, \dots, \mu_{n_{\boldsymbol{c}}}, \mu_{n_{\boldsymbol{c}}}^2 + \sigma_{n_{\boldsymbol{c}}}^2)$. Then, we have $T(\boldsymbol{c}) = (T_1([\boldsymbol{c}]_1), \dots, T_{n_{\boldsymbol{c}}}([\boldsymbol{c}]_{n_{\boldsymbol{c}}}))$ with $T_i([\boldsymbol{c}]_i) = ([\boldsymbol{c}]_i, [\boldsymbol{c}]_i^2)$. It is the exponential parameterization as $\theta_i = (\mu_i, \mu_i^2 + \sigma_i^2) = \mathbb{E}[T_i([\boldsymbol{c}]_i)]$ and $\theta = (\theta_1, \dots, \theta_{n_{\boldsymbol{c}}})$.

The inverse of the Fisher information matrix simply follows from the formula for the exponential family. $\mathbf{F}(\theta)^{-1}$ is a block-diagonal matrix with block size 2 whose i-th block is $\mathbf{F}_i(\theta_i)^{-1} = [\sigma_i^2, 2\mu_i\sigma_i^2; 2\mu_i\sigma_i^2, 4\mu_i^2\sigma_i^2 + 2\sigma_i^4]$. Since each block is a symmetric matrix of dimension 2, its eigen decomposition $\mathbf{F}_i(\theta_i)^{-1} = \mathbf{E}\Lambda\mathbf{E}^{\mathrm{T}}$ can be analytically obtained. With the decomposition, we have $\mathbf{F}_i(\theta_i) = \mathbf{E}\Lambda^{-1}\mathbf{E}^{\mathrm{T}}$ and $\mathbf{F}(\theta_i)^{\frac{1}{2}} = \mathbf{E}\Lambda^{-\frac{1}{2}}\mathbf{E}^{\mathrm{T}}$. Then, we have $\mathbf{F}(\theta) = \operatorname{diag}(\mathbf{F}_1(\theta_1), \dots, \mathbf{F}_{n_{\boldsymbol{c}}}(\theta_{n_{\boldsymbol{c}}}))$ and $\mathbf{F}(\theta)^{\frac{1}{2}} = \operatorname{diag}(\mathbf{F}_1(\theta_1)^{\frac{1}{2}}, \dots, \mathbf{F}_{n_{\boldsymbol{c}}}(\theta_{n_{\boldsymbol{c}}})^{\frac{1}{2}})$.

D. Restriction for the Range of θ

To guarantee that the Fisher information matrix is nonsingular and the natural gradient is well defined, we restrict the domain Θ of the parameter of the probability distribution.

For the categorical distribution, we set $\Theta = [\theta_1^{\min}, \theta^{\max}]^{m_1 - 1} \times \cdots \times [\theta_{n_{\boldsymbol{c}}}^{\min}, \theta^{\max}]^{m_{n_{\boldsymbol{c}}} - 1}$, where $\theta_i^{\min} = \frac{1}{n_{\boldsymbol{c}}(m_i - 1)}$ and $\theta^{\max} = 1 - \frac{1}{n_{\boldsymbol{c}}}$. Then, a small yet positive probability for all combinations of categorical variables is guaranteed and the Fisher information matrix is nonsingular at any point of Θ.

To force the parameter to live in Θ, we apply the following steps after θ update:

$$[\theta]_{i,j} \leftarrow \max\{[\theta]_{i,j}, \theta_i^{\min}\} \text{ for all } i, j \text{, then}$$

$$[\theta]_{i,j} \leftarrow [\theta]_{i,j} + \frac{1 - \sum_{k=1}^{m_i} [\theta]_{i,k}}{\sum_{k=1}^{m_i} \left([\theta]_{i,k} - \theta_i^{\min}\right)} \left([\theta]_{i,j} - \theta_i^{\min}\right) \enspace .$$

The first line guarantees $[\theta]_{i,j} \geq \theta_i^{\min}$. The second line ensures $\sum_{j=1}^{m_i} [\theta]_{i,j} = 1$, while keeping $[\theta]_{i,j} \geq \theta_i^{\min}$.

For the integer variables, the parameters of the Gaussian distributions, $[\theta]_{i,1} := \mu_i$ and $[\theta]_{i,2} := \mu_i^2 + \sigma_i^2$, are forced to be in a compact set as follows. The range of the mean value of each integer variable is $[\mu_i^{\min}, \mu_i^{\max}]$, which is the same as the range of the integer variable. The standard deviation is forced to be no smaller than $\sigma_i^{\min} = 1/4$ and

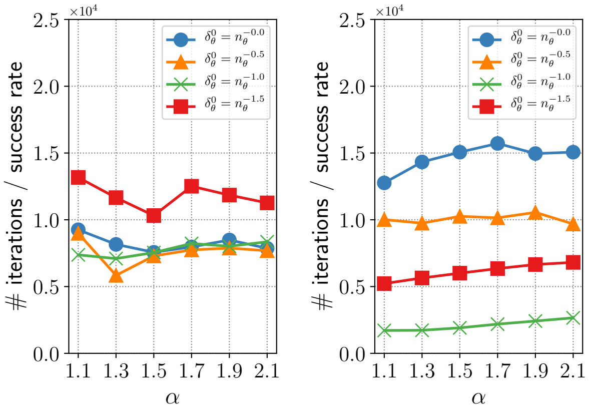

Figure 3: Performance of ASNG with the different α settings on the selective squared error function for $\epsilon_{\boldsymbol{x}} = 0.05$ (left) and 0.0005 (right). Median values over 100 runs are reported.

no greater than $\sigma_i^{\max} = (\mu_i^{\max} - \mu_i^{\min})/2$. To keep the parameters inside these ranges, after every θ update we clip $[\theta]_{i,1}$ to $[\mu_i^{\min}, \mu_i^{\max}]$ and $[\theta]_{i,2}$ to $[[\theta]_{i,1}^2 + (\sigma_i^{\min})^2, [\theta]_{i,1}^2 + (\sigma_i^{\max})^2]$. If the variables are real-value, rather than integer, then $\sigma_i^{\min}$ may be set smaller depending on the meaning of the variable.

E. Experimental Details

E.1. Toy Problem

To check the robustness of ASNG for the hyper-parameter α, we ran ASNG on the selective squared error function with the varying α and initial step-size δ_θ^0 for the step-sizes of $\epsilon_{\boldsymbol{x}} = \{0.05, 0.0005\}$. Figure 3 shows the performance of ASNG with the different α settings. We observe that the hyper-parameter α is not sensitive for the performance, and ASNG reaches the target value for all settings.

E.2. Image Classification

Dataset: We use the CIFAR-10 dataset which consists of 50,000 and 10,000 RGB images of 32 × 32, for training and testing. All images are standardized in each channel by subtracting the mean and then dividing by the standard deviation. We adopt the standard data augmentation for each training mini-batch: padding 4 pixels on each side, followed by choosing randomly cropped 32 × 32 images and by performing random horizontal flips on the cropped images. We also apply the cutout (DeVries & Taylor, 2017) to the training data.

Search Space: Figure 4 shows the overall model structure for the classification task. We optimize the architecture of the normal and reduction cells by ASNG-NAS. In the retraining phase, we construct the CNN using the optimized cell architecture with an increased number of cells N and channels. In the reduction cell, all operations applied to the inputs of the cell have a stride of 2, and the number of channels is doubled to keep the dimension of the output roughly constant.

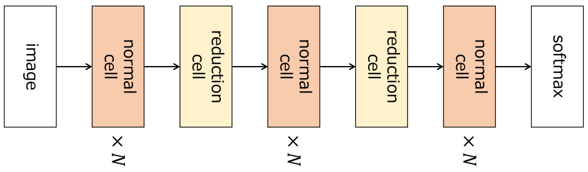

Figure 4: Overall model structure for the classification task.

Training Details: In the architecture search phase, we fix affine parameters of batch normalizations for the purpose of absorbing effect of the dynamic change in architecture. We apply weight decay of 3×10^{-4} and clip the norm of gradient at 5. In the retraining phase, we make all batch normalizations have learnable affine parameters because the architecture no longer changes. We apply the ScheduledDropPath (Zoph et al., 2018) dropping out each path between nodes, and the drop path rate linearly increases from 0 to 0.3 during the training. We also add the auxiliary classifier (Szegedy et al., 2016) with the weight of 0.4 that is connected from the second reduction cell. The total loss is a weighted sum of the losses of the auxiliary classifier and output layer. Other settings are the same as the architecture search phase.

The best cell structures: The best cell structures that achieve the error rate of 2.66% is displayed in Figure 5.

E.3. Inpainting

Dataset: The CelebA is a large-scale human face image dataset that contains 202,599 RGB images. We select 101,000 and 2,000 images for training and test, respectively, in the same way as Suganuma et al. (2018). All images were cropped to properly contain the entire face by using the provided the bounding boxes, and resized to 64 × 64 pixels. All images are normalized by dividing by 255, and we perform data augmentation of random horizontal flipping on the training images. We adopt three masks, Center, Pixel, and Half, to make corrupted images. The purpose of the task is to recover a clean image from the corrupted image as much as possible. The masks in random pixel and half image masks were randomly generated for each training mini-batch and each test image.

Evaluation Measure: The PSNR is the metric evaluating the error between the ground truth and restored images and corresponds to the mean squared error (MSE). But the

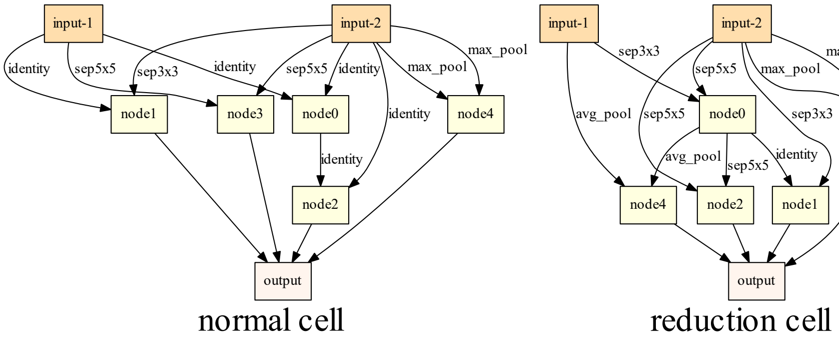

Figure 5: The best cell structures discovered by ASNG-NAS in the classification task.

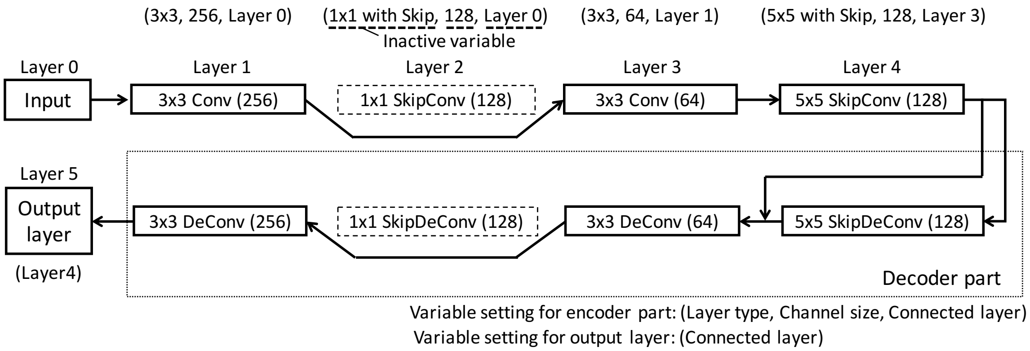

Figure 6: A conceptual example of the decoded symmetric CAE architecture and the corresponding categorical variables. The decoder part is automatically decided from the encoder structure as a symmetric manner.

PSNR are not very well matched to perceived visual quality because the PSNR can not distinguish between the large difference on local region and the small difference on overall region. For this reason, the SSIM is also often used together with the PSNR, and more clearly assesses difference in each local region. We quantize the generated image within $[0, 255]$ followed by to calculate the PSNR and SSIM value. The setting of SSIM is based on Wang et al. (2004).

Search Space: We employ the convolutional autoencoder (CAE), which is similar to RED-Net (Mao et al., 2016), as a base architecture. RED-Net consists of a chain of convolution layers and symmetric deconvolution layers as the encoder and decoder parts, respectively. The encoder and decoder parts perform the same counts of downsampling and upsampling with a stride of 2, and a skip connection between the convolutional layer and the mirrored deconvolution layer can exist. For simplicity, each layer employs either a skip connection or a downsampling, and the decoder part is employed in the same manner. In the skip connected deconvolution layer, the input feature maps from the encoder part are added to the output of the deconvolution operation, followed by ReLU. In the other layers, the ReLU activation is performed after the convolution and deconvolution operations. We prepare six types of layers: the combination of the kernel sizes $\{1 \times 1, 3 \times 3, 5 \times 5\}$ and the existence of the skip connection. The layers with different settings do not share weight parameters.

To represent a symmetric CAE, it is enough to represent the encoder part. We consider N_c hidden layers and the output layer. We encode the type, channel size, and connections of each hidden layer. The kernel size and stride of the output deconvolution layer are fixed with 3×3 and 1, respectively, but the connection is determined by a categorical variable. To ensure the feed-forward architecture and to control the network depth, the connection of the i-th layer is only allowed to be connected from $(i-1)$ to $\max(0, i-b)$-th layers, where b ($b > 0$) is called the level-back parameter. Namely, the categorical variable representing the connection of the i-th layer has $\min(i, b)$ categories. Obviously, the first hidden layer always connects with the input, and we can ignore this part. With this representation, there can exist *inactive* layers that do not connect to the output layer. Therefore, this model can represent variable length architectures by the fixed-dimensional variables. We choose $N_c = 20$ and the level-back parameter of $b = 5$.

ASNG-NAS (Cat) encodes the type and channel size of each hidden layer by categorical variables with 6 and 3 categories, respectively. We select the output channel size of each

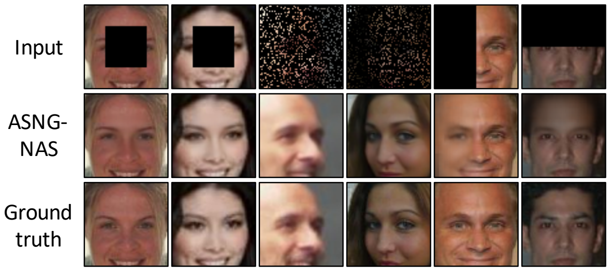

Figure 7: Example of inpainting results obtained by ASNG-NAS.

hidden layer from $\{64, 128, 256\}$. It amounts to $n_{\boldsymbol{c}} = 60$ (# categorical variables) and $n_\theta = 214$ (dimension of θ). A conceptual example of the symmetric CAE architecture and the corresponding representation by the categorical variables is shown in Figure 6. ASNG-NAS (Int) encodes the kernel size and the channel size by integers in $[\![1, 3]\!]$ (corresponding to $\{1 \times 1, 3 \times 3, 5 \times 5\}$) and $[\![64, 256]\!]$. The existence of skip connection is determined by a categorical variable with 2 categories. It amounts to $n_{\boldsymbol{c}} = 80$ (# categorical and integer variables) and $n_\theta = 174$ (dimension of θ).

Example of Inpainting Result: Figure 7 shows the example of inpainting results obtained by ASNG-NAS.