This CVPR 2020 paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

GP-NAS: Gaussian Process based Neural Architecture Search

Zhihang Li[2]*, Teng Xi[3,5]*, Jiankang Deng[4], Gang Zhang[3], Shengzhao Wen[3], Ran He[1]†
[1]NLPR & CEBSIT, CAS [2]NLPR & AIR, CAS
[3]Department of Computer Vision Technology (VIS), Baidu Inc. [4]Imperial College London
[5]Department of Computer Science and Technology, Tsinghua University.
Email: {zhihang.li, rhe}@nlpr.ia.ac.cn, j.deng16@imperial.ac.uk
{xiteng01, zhanggang03, wenshengzhao}@baidu.com, xiteng@mail.tsinghua.edu.cn

Abstract

Neural architecture search (NAS) advances beyond the state-of-the-art in various computer vision tasks by automating the designs of deep neural networks. In this paper, we aim to address three important questions in NAS: (1) How to measure the correlation between architectures and their performances? (2) How to evaluate the correlation between different architectures? (3) How to learn these correlations with a small number of samples? To this end, we first model these correlations from a Bayesian perspective. Specifically, by introducing a novel Gaussian Process based NAS (GP-NAS) method, the correlations are modeled by the kernel function and mean function. The kernel function is also learnable to enable adaptive modeling for complex correlations in different search spaces. Furthermore, by incorporating a mutual information based sampling method, we can theoretically ensure the high-performance architecture with only a small set of samples. After addressing these problems, training GP-NAS once enables direct performance prediction of any architecture in different scenarios and may obtain efficient networks for different deployment platforms. Extensive experiments on both image classification and face recognition tasks verify the effectiveness of our algorithm.

1. Introduction

NAS targets on automating the design of deep neural networks. It has promoted remarkable progress in various tasks, including image classification [55, 31, 38, 53, 16, 45], object detection [19] and semantic segmentation [29, 34, 51]. However, conventional early NAS methods that are based on reinforcement learning (RL) [55] and evolutionary algorithm (EA) [38] are often computationally intensive and just search a specified architecture each time for a target task. Many recent works focus on accelerating NAS by weight sharing [35, 5, 52, 13] and achieve great improvements. But the relationship between the performance and the neural architecture is not clear enough and they may lead to the well known performance inconsistency problem between the supernet model and the fully trained stand alone model. In addition, they require repeated sampling to obtain efficient networks for different tasks and hardware platforms.

We aim to learn a performance predictor with high efficacy and identify efficient model with high performance for different platforms without repeated search process. This performance predictor would have the following properties: (1) It may model the correlation between architectures and the corresponding performances. As such, it can infer performance of any deep model (without retraining) in a given search space. (2) The predictor may be learned with a small set of samples. The training process will thus be computationally efficient. (3) It may measure the correlation between different architectures such that architectures with similar performances can be easily found. It can also provide some insights about what micro- or macro-architectures would be beneficial for learning.

To this end, we propose a theoretical modeling framework for NAS. Specifically, we model the distribution of performances conditioned on given architectures by a Gaussian Process (GP). Under this framework, the correlation between performances and architectures can be modeled by a mean function in the GP. The correlation between different architectures is measured by the kernel functions as well. Moreover, the kernel function is learnable to measure complex correlations in different search spaces. In order to speed up the search process in our proposed method , we further propose an efficient mutual information based sampling method. This sampling method intuitively enlarges the distance between obtained samples. We theoretically

*Equal contribution. This work was done when the first author was an intern at Department of Computer Vision Technology (VIS), Baidu Inc.
†corresponding author

prove that only with a small set of samples, we can obtain a universal performance predictor for given search space with high accuracy.

The GP-NAS can be solved in an alternating estimation manner which recursively updates the posterior distribution of learnable hyperparameters and enables us to approach the optimal distribution of performances conditioned on given architectures. This further ensures a good transferable property of our GP-NAS framework. For example, we experimentally validate that after updating the hyperparameters on CIFAR-10 dataset, we can use them as a prior knowledge of the learning problem on ImageNet dataset. The prior knowledge enables the distribution of hyperparameters to be quickly adapt to the new problem with fewer samples.

In addition, GP-NAS framework is orthogonal to current NAS methods. Except directly sampling architectures from a search space and training them to obtain the performances, our GP-NAS method can work with existing differentiable architecture search methods and one-shot methods. GP-NAS disentangles the training and search process in these scenarios. Therefore, GP-NAS can ensure efficient deployment of effective deep models for different tasks and different platforms without retraining and re-search process.

In conclusion, our contributions are four-fold:

- We propose a theoretical framework that can infer performance of any architectures in a given search space and yield efficient deep models for different tasks and platforms without retraining.
- As far as we know, this is the first time that distribution of performances conditioned on architectures is modeled by Gaussian Process, in which the correlation between performances and architectures and the correlation between different architectures are explicitly modeled.
- We suggest an efficient sampling method that provably ensures that our GP-NAS can be learned with only a small set of samples.
- Experiments demonstrate that the network architecture searched by GP-NAS achieves competitive results on CIFAR-10 and ImageNet with a high efficiency. The proposed sampling strategy may accelerate the learning process by 25 times than random sampling. We also obtain a competitive performance on face recognition.

2. Related Work

Existing NAS methods can be roughly classified into three categories: RL based methods [54, 2, 1, 53, 43], EA based methods [31] and GD based methods [33, 32, 46, 10, 9, 15]. The core ideas of these methods are that a sampled child network is trained on the training set and evaluated on the validation set, while an agent receives performance from the child network as a reward and learns to generate network architectures that are more likely to achieve higher performance.

Despite the success of the above NAS approaches, most existing methods need to train a large number of child networks from scratch, which is time-consuming and expensive on a large-scale dataset. To accelerate the search process, [3, 36] depend on sharing weights strategy where a single large network is trained with all candidate architectures included. When evaluating each sampling child network, just zero-out some operations is enough without the need of retraining. Another intuitive method is to reduce the search space by imposing some constraints. [32, 48] search for the best cells or blocks rather than the entire network and stack them to construct a full network. In addition, recent approaches [4, 48] have proposed alternatives to generate weights of network or predict the performance directly by a HyperNet [20]. However, training a weight or performance predictor requires a large number of ground truth. It is inevitable to spend a lot of time on sampling a large number of child networks. Thus, an efficient sampling strategy is greatly needed. Mutual information is a measure of discrepancy between two distributions, which has been widely-used in representation learning, e.g. conversational responses [50], disentangled feature [18, 8, 21], RL [22] and etc. We introduce mutual information to measure the information gain of sampling and try to estimate the hyperparameters of model with the minimal number of sampled network architectures.

Bayesian estimation is a global optimization algorithm to evaluate a black-box function, which has been applied in machine learning for model selection [25, 42, 26]. Gaussian processes (GP) is a flexible non-parametric framework for reasoning over functions, which is one of the most extensively-used methods in Bayesian estimation due to its flexibility and tractability. To model the distribution of performance under network architecture, we attempt to extend the GP and design a kernel function specially for NAS.

3. Method

In this section, we first present how to model NAS with GP in which specially designed kernel function for NAS is presented. Then, we describe how to embed mutual information into GP framework in details. Finally, we derive a closed form solution for hyperparameters and propose an alternating estimation algorithm to recursively update the hyperparameters.

3.1. Modeling of GP-NAS

Gaussian process [37] is an non-parametric framework to reason over function f defined on a domain $\mathcal{X}$, which

has been widely used as a nonlinear regression technique for data reconstruction. GP can be completely characterised by its mean $\mu : \mathcal{X} \to \mathbb{R}$ and covariance (kernel) function $k : \mathcal{X}^2 \to \mathbb{R}$. Given t observation samples $D_t = \{(x_i, y_i)\}_{i=1}^t$, the posterior process is still a GP with mean μ_t and covariance k_t.

Since GP can use observations to predict unobserved data, we utilize it to infer the performance of any network architectures based on sampled networks. In the context of NAS, the domain is defined as the space of network architecture $\mathcal{S}$ and the function f is the performance of the network on the validation set after it is trained on the training set. At time $t \in \mathcal{T}$, the observations $\mathbf{s}_t$ are the sampled architectures before time t, and $\mathbf{u}_t$ are the unobserved samples, where $\mathbf{n} = \mathbf{s}_t \cup \mathbf{u}_t$ denotes the universal set of architecture space. Let $x_t(s)$ be the performance of a network architecture s at time t, $\forall s \in \mathbf{n}, \forall t \in \mathcal{T}$. Following [28], we encode each neural architecture into a vector $s = [s^1, s^2, \ldots, s^k]^T$, where $s^i \in \mathcal{S}$ is the selected operation in i-th layer. $\mathcal{S}$ denotes the universal set of all operations. Thus, $|\mathbf{s}|$ neural architectures have been sampled at time t and their performances form a vector as follows:

$$\begin{aligned} &\mathbf{x}_t(\mathbf{s}) = \left[x_t(s_1), x_t(s_2), \ldots, x_t(s_{|\mathbf{s}|})\right]^T, \\ &\forall t \in \mathcal{T}, s_i \in \mathbf{s}, \forall i \in \{1, 2, \cdots, |\mathbf{s}|\}, \forall \mathbf{s} \subseteq \mathbf{n}. \end{aligned} \tag{1}$$

In the GP, the performance of different network architectures are assumed to be jointly Gaussian distributed, therefore

$$\begin{aligned} &\begin{bmatrix} \mathbf{x}_t(\mathbf{s}_t) \\ \mathbf{x}_t(\mathbf{u}_t) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{m}_t(\mathbf{s}_t) \\ \mathbf{m}_t(\mathbf{u}_t) \end{bmatrix}, \begin{bmatrix} \mathbf{K}_t(\mathbf{s}_t, \mathbf{s}_t) & \mathbf{K}_t(\mathbf{s}_t, \mathbf{u}_t) \\ \mathbf{K}_t(\mathbf{u}_t, \mathbf{s}_t) & \mathbf{K}_t(\mathbf{u}_t, \mathbf{u}_t) \end{bmatrix}\right), \\ &\forall \mathbf{s}_t \subset \mathbf{n}, \mathbf{u}_t = \mathbf{n} \setminus \mathbf{s}_t, \forall t \in \mathcal{T}, \end{aligned} \tag{2}$$

where $\mathcal{N}(.)$ is the probability density function of Gaussian distribution and the covariance matrix $\mathbf{K}_t(\mathbf{s}_t, \mathbf{u}_t)$ is denoted by:

$$\begin{aligned} &\mathbf{K}_t(\mathbf{s}_t, \mathbf{u}_t) = \\ &\begin{bmatrix} k_t(s_1, u_1) & k_t(s_2, u_1) & \ldots & k_t(s_{|\mathbf{s}_t|}, u_1) \\ k_t(s_1, u_2) & k_t(s_2, u_2) & \ldots & k_t(s_{|\mathbf{s}_t|}, u_2) \\ \vdots & \vdots & \ddots & \vdots \\ k_t(s_1, u_{|\mathbf{u}_t|}) & k_t(s_2, u_{|\mathbf{u}_t|}) & \ldots & k_t(s_{|\mathbf{s}_t|}, u_{|\mathbf{u}_t|}) \end{bmatrix}, \\ &s_i \in \mathbf{s}_t, \forall i \in \{1, 2, \cdots, |\mathbf{s}_t|\}, \forall \mathbf{s}_t \subset \mathbf{n}, \mathbf{u}_t = \mathbf{n} \setminus \mathbf{s}_t, \\ &u_j \in \mathbf{u}_t, \forall j \in \{1, 2, \cdots, |\mathbf{u}_t|\}, \forall t \in \mathcal{T}, \end{aligned} \tag{3}$$

where the kernel function $k_t(s, u)$ describes the covariance between s and u at t, $\forall s, u \in \mathbf{n}$. The mean vectors $\mathbf{m}_t(\mathbf{s}_t)$ and $\mathbf{m}_t(\mathbf{u}_t)$ in Equation 2 are denoted by:

$$\begin{aligned} &\mathbf{m}_t(\mathbf{s}) = \left[m_t(s_1), m_t(s_2), \ldots, m_t(s_{|\mathbf{s}|})\right]^T, \\ &\forall \mathbf{s} \subset \mathbf{n}, s_i \in \mathbf{s}, \forall i \in \{1, 2, \cdots, |\mathbf{s}|\}, \end{aligned} \tag{4}$$

where $m_t(s)$ is the mean performance of the network architecture s at t. Because the architecture of a network has great effect on performance, the mean function $m_t(s)$ should depend on the network architecture s. Note that s could be a naively encoded network or the embedding of network by some transforms. Here we take a linear mean function as an example for concision:

$$m_t(s) = \mathbf{w}^T s, \forall s \subset \mathbf{n}, \forall t \in \mathcal{T}. \tag{5}$$

where $\mathbf{w} = [w_1, w_2, \ldots, w_{|s|}]^T$ is the hyperparameter to be estimated.

Intuitively, the kernel function $k_t(s, u)$ measures similarity between two networks s, u. When $k_t(s, u)$ is large, the performance of two networks are close. However, it is non-trivial to define a reasonable distance between two networks, because each item in a network encoding vector denotes different types of operations and they are usually not comparable, such as the number of channels and kernel size. Therefore, we divide operations $\mathcal{S}$ into multiple groups by types:

$$\mathcal{S} = \bigcup_{i=1}^{h} \mathcal{S}_i, \tag{6}$$

where $\mathcal{S}_i$ is a type of operations, such as number of channels or kernel sizes. h is the number of types in the search space. Then, the customized kernel function $k_t(s, u)$ is formulated as:

$$\begin{aligned} &k_t(s, u) = \sum_{i=1}^{n} (\sigma_t^i)^2 exp(-(l_i \cdot (s - u))^T ((s - u) \cdot l_i)), \\ &\forall s, u \in \mathbf{n}, \forall t \in \mathcal{T}, \end{aligned} \tag{7}$$

Where l_i is the mask for the i-th type of operations, which is defined as follows:

$$\begin{aligned} &l_i = \left[l_i^1, l_i^2, \ldots, l_i^k\right]^T, \\ &k = |s|, l_i^{j_1} = 1, \forall s^{j_1} \in \mathcal{S}_i, l_i^{j_2} = 0, \forall s^{j_2} \notin \mathcal{S}_i, \\ &j_1, j_2 \in \{1, 2, \cdots, n\}. \end{aligned} \tag{8}$$

σ_t^i in Equation 7 is the hyperparameter to be estimated.

Following the results in [37], the conditional probability distribution (or, predictive posterior distribution) of the unsampled network architectures $\mathbf{x}_t(\mathbf{u}_t)$ is given by

$$\begin{aligned} &p_t(\mathbf{x}_t(\mathbf{u}_t)|\mathbf{x}_t(\mathbf{s}_t)) = \mathcal{N}(\mu_t(\mathbf{u}_t|\mathbf{s}_t), \mathbf{K}_t(\mathbf{u}_t|\mathbf{s}_t)), \\ &\forall \mathbf{s}_t \subset \mathbf{n}, \mathbf{u}_t = \mathbf{n} \setminus \mathbf{s}_t, \forall t \in \mathcal{T}, \end{aligned} \tag{9}$$

where the conditional mean vector $\mu_t(\mathbf{u}_t|\mathbf{s}_t)$ and the conditional covariance matrix $\mathbf{K}_t(\mathbf{u}_t|\mathbf{s}_t)$ are as follows:

$$\begin{aligned} &\mu_t(\mathbf{u}_t|\mathbf{s}_t) = \\ &\mathbf{m}_t(\mathbf{u}_t) + \mathbf{K}_t(\mathbf{u}_t, \mathbf{s}_t)\mathbf{K}_t(\mathbf{s}_t, \mathbf{s}_t)^{-1}(\mathbf{x}_t(\mathbf{s}_t) - \mathbf{m}_t(\mathbf{s}_t)), \\ &\forall \mathbf{s}_t \subset \mathbf{n}, \mathbf{u}_t = \mathbf{n} \setminus \mathbf{s}_t, \forall t \in \mathcal{T}, \end{aligned} \tag{10}$$

$$\mathbf{K}_t(\mathbf{u}_t|\mathbf{s}_t) = \mathbf{K}_t(\mathbf{u}_t,\mathbf{u}_t) - \mathbf{K}_t(\mathbf{u}_t,\mathbf{s}_t)\mathbf{K}_t(\mathbf{s}_t,\mathbf{s}_t)^{-1}\mathbf{K}_t(\mathbf{s}_t,\mathbf{u}_t), \forall \mathbf{s}_t \subset \mathbf{n}, \mathbf{u}_t = \mathbf{n} \setminus \mathbf{s}_t, \forall t \in \mathcal{T}. \tag{11}$$

However, training numerous sampled networks and obtaining their performances is very time-consuming. Most current works [55, 3] focus on reducing the training time of a single sampled network, such as training for fewer epochs, searching on a small dataset, learning with fewer blocks or weight sharing. From a novel perspective, we aim to minimize the sample times. To realize it, we attend to optimize the sampling strategy from an information theory perspective. Mutual information is used to measure the mutual dependence between two distributions[18, 8, 21]. At time t, the sampled network architectures $\mathbf{x}_t(\mathbf{s}_t)$ and the unsampled architectures $\mathbf{x}_t(\mathbf{u}_t)$ are formed as two distributions. Here we try to make $M(\mathbf{x}_t(\mathbf{s}_t), \mathbf{x}_t(\mathbf{u}_t))$ approach $\mathbf{x}_t(\mathbf{s}_t)$ by maximizing their mutual information. In this way, we can select a neural architecture which carries high information gain for each sampling.

Based on Equation 11, $M(\mathbf{x}_t(\mathbf{s}_t), \mathbf{x}_t(\mathbf{u}_t))$ is obtained by:

$$M(\mathbf{x}_t(\mathbf{s}_t), \mathbf{x}_t(\mathbf{u}_t)) = \frac{1}{2}\log(\frac{|\mathbf{K}_t(\mathbf{u}_t,\mathbf{u}_t)|}{|\mathbf{K}_t(\mathbf{u}_t|\mathbf{s}_t)|}), \forall \mathbf{s}_t \subset \mathbf{n}, \mathbf{u}_t = \mathbf{n} \setminus \mathbf{s}_t, \forall t \in \mathcal{T}, \tag{12}$$

where $|\mathbf{K}_t(\mathbf{u}_t,\mathbf{u}_t)|$ and $|\mathbf{K}_t(\mathbf{u}_t|\mathbf{s}_t)|$ denote the determinants of $\mathbf{K}_t(\mathbf{u}_t,\mathbf{u}_t)$ and $\mathbf{K}_t(\mathbf{u}_t|\mathbf{s}_t)$. Due to the limitation of space, more details refer to [12].

In summary, we utilize the performance of sampled networks to estimate the hyperparameters. Furthermore, we employ an efficient sampling strategy by using mutual information. The target hyperparameters to be estimated are σ_t^i and $\mathbf{w}$, $\forall i \in \{1, 2, \cdots, n\}$. In the following section, we will discuss in detail how to estimate these hyperparameters.

3.2. Hyperparameter Estimation of GP-NAS

In this section, we present how to estimate the hyperparameters of GP-NAS using a minimal number of sampled network architectures. From the definition of GP-NAS in Equation 2, we have:

$$x_t(s) = \mathbf{w}^T s + \varepsilon_t(s), \forall s \in \mathbf{n}, \forall t \in \mathcal{T}, \tag{13}$$

where $\varepsilon_t(s)$ is subject to a Gaussian distribution, and

$$p_t(\varepsilon_t(s)) = \mathcal{N}(\varepsilon_t(s)|0, k_t(s,s)) = \mathcal{N}(\varepsilon_t(s)|0, \sum_{i=1}^{n}(\sigma_t^i)^2)). \tag{14}$$

Then, in matrix form:

$$\mathbf{x}_t(\mathbf{s}) = \Phi(\mathbf{s})\mathbf{w} + \mathbf{e}_t(\mathbf{s}), \forall \mathbf{s} \subseteq \mathbf{n}, \forall t \in \mathcal{T}, \tag{15}$$

where $\Phi(\mathbf{s})$ and $\mathbf{e}_t(\mathbf{s})$ are denoted by:

$$\Phi(\mathbf{s}) = \left[s, s \in \mathbf{s}\right]^T, \forall \mathbf{s} \subseteq \mathbf{n}, \tag{16}$$

$$\mathbf{e}_t(\mathbf{s}) = \left[\varepsilon_t(s), s \in \mathbf{s}\right]^T, \forall \mathbf{s} \subseteq \mathbf{n}, \forall t \in \mathcal{T}. \tag{17}$$

Let $p_t(\mathbf{w})$ be the prior distribution [41] of $\mathbf{w}$, where

$$p_t(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mu_{\mathbf{w}}^t, \Sigma_{\mathbf{w}}^t). \tag{18}$$

In Equation 18, $\mu_{\mathbf{w}}^t$ is the prior mean vector of $\mathbf{w}$, $\Sigma_{\mathbf{w}}^t$ is the prior covariance matrix of $\mathbf{w}$ before sampling at t.

Then, the probability density function of $x_t(s)$ is given by:

$$p_t(x_t(s)|\mathbf{w}) = \mathcal{N}(x_t(s)|\mathbf{w}^T s, \sum_{i=1}^{n}(\sigma_t^i)^2), \forall s \in \mathbf{n}, \forall t \in \mathcal{T}, \tag{19}$$

where σ_t^i can be recursively estimated $\forall i \in \{1, 2, \cdots, n\}$. Furthermore, the probability density function of $\mathbf{x}_t(\mathbf{s})$ is given by:

$$p_t(\mathbf{x}_t(\mathbf{s})|\mathbf{w}) = \mathcal{N}(\mathbf{x}_t(\mathbf{s})|\mathbf{w}^T\mathbf{s}, \mathbf{K}_t(\mathbf{s},\mathbf{s})), \forall \mathbf{s} \subseteq \mathbf{n}, \forall t \in \mathcal{T}. \tag{20}$$

Let $\hat{\sigma}_t^i$ be the estimated hyperparameter, where:

$$\hat{\sigma}_t^1, \hat{\sigma}_t^2, \cdots, \hat{\sigma}_t^n = \underset{\sigma_t^1, \sigma_t^2, \cdots, \sigma_t^n}{\arg\max}\ p_t(\mathbf{x}_t(\mathbf{s})|\mathbf{w}), \forall \mathbf{s} \subseteq \mathbf{n}, \forall t \in \mathcal{T}. \tag{21}$$

Let $\hat{\mu}_{\mathbf{w}}^t$ and $\hat{\Sigma}_{\mathbf{w}}^t$ be the posterior mean vector and covariance matrix of $\mathbf{w}$ to be updated after sampling at t. Then, the predicted performance of any network architecture(s) $\mathbf{s}$, $\hat{\mathbf{x}}_t(\mathbf{s})$, can be estimated by:

$$\hat{\mathbf{x}}_t(\mathbf{s}) = \Phi(\mathbf{s})\hat{\mu}_{\mathbf{w}}^t, \forall \mathbf{s} \subseteq \mathbf{n}, \forall t \in \mathcal{T}. \tag{22}$$

Notably, the posterior mean vector and covariance matrix of $\mathbf{w}$ at t_p turn into the prior mean vector and covariance matrix of $\mathbf{w}$, where t_p is the previous sampling time of t. Following the results by theorem 1, we can recursively update the mean vector of $\mathbf{w}$.

Theorem 1. *The posterior mean vector of* $\mathbf{w}$ *after sampling at t,* $\hat{\mu}_{\mathbf{w}}^t$*, can be updated as follows:*

$$\hat{\mu}_{\mathbf{w}}^t = \mu_{\mathbf{w}}^t + \Phi(\mathbf{s})^T\Sigma_{\mathbf{w}}^t(\mathbf{K}_t(\mathbf{s},\mathbf{s}) + \Phi(\mathbf{s})^T\Sigma_{\mathbf{w}}^t\Phi(\mathbf{s}))^{-1} \times (\mathbf{x}_t(\mathbf{s}) - \Phi(\mathbf{s})\mu_{\mathbf{w}}^t), \forall \mathbf{s} \subseteq \mathbf{n}, \forall t \in \mathcal{T}. \tag{23}$$

Proof. Let $p_t(\mathbf{w}|\mathbf{x}_t(\mathbf{s}))$ be the posterior distribution [41] of $\mathbf{w}$,

$$p_t(\mathbf{w}|\mathbf{x}_t(\mathbf{s})) \propto p_t(\mathbf{x}_t(\mathbf{s})|\mathbf{w})p_t(\mathbf{w}), \forall \mathbf{s} \subseteq \mathbf{n}, \forall t \in \mathcal{T}. \tag{24}$$

This results in the following log-likelihood function:

$$
\begin{aligned}
&L(\mathbf{w},\mathbf{s}) \triangleq \ln p_t(\mathbf{x}_t(\mathbf{s})|\mathbf{w})p_t(\mathbf{w}) \\
&= -\frac{|\mathbf{s}|+|\mathbf{s}|}{2}\ln 2\pi - \frac{1}{2}\ln|\mathbf{K}_t(\mathbf{s},\mathbf{s})| - \frac{1}{2}\ln|\Sigma^t_{\mathbf{w}}| \\
&- \frac{1}{2}(\mathbf{w}-\mu^t_{\mathbf{w}})^T(\Sigma^t_{\mathbf{w}})^{-1}(\mathbf{w}-\mu^t_{\mathbf{w}}) \\
&- \frac{1}{2}(\mathbf{x}_t(\mathbf{s})-\Phi(\mathbf{s})\mathbf{w})^T(\mathbf{K}_t(\mathbf{s},\mathbf{s}))^{-1}(\mathbf{x}_t(\mathbf{s})-\Phi(\mathbf{s})\mathbf{w}).
\end{aligned} \tag{25}
$$

The posterior mean vector of $\mathbf{w}$, $\hat{\mu}^t_{\mathbf{w}}$, can be calculated by:

$$
\hat{\mu}^t_{\mathbf{w}} = \arg\max_{\mathbf{w}} L(\mathbf{w},\mathbf{s}), \forall \mathbf{s} \subseteq \mathbf{n}. \tag{26}
$$

Taking the derivative of log-likelihood function in respect to $\mathbf{w}$ leads to:

$$
\begin{aligned}
&\frac{\partial L(\mathbf{w},\mathbf{s})}{\partial \mathbf{w}} = \\
&\Phi(\mathbf{s})^T(\mathbf{K}_t(\mathbf{s},\mathbf{s}))^{-1}(\mathbf{x}_t(\mathbf{s})-\Phi(\mathbf{s})\mathbf{w}) - (\Sigma^t_{\mathbf{w}})^{-1}(\mathbf{w}-\mu^t_{\mathbf{w}}), \\
&\forall \mathbf{s} \subseteq \mathbf{n}, \forall t \in \mathcal{T}.
\end{aligned} \tag{27}
$$

By setting Equation (27) to zero, we have:

$$
\begin{aligned}
\hat{\mu}^t_{\mathbf{w}} =& \mu^t_{\mathbf{w}} + \Phi(\mathbf{s})^T\Sigma^t_{\mathbf{w}}(\mathbf{K}_t(\mathbf{s},\mathbf{s}) + \Phi(\mathbf{s})^T\Sigma^t_{\mathbf{w}}\Phi(\mathbf{s}))^{-1} \\
&(\mathbf{x}_t(\mathbf{s}) - \Phi(\mathbf{s})\mu^t_{\mathbf{w}}), \forall \mathbf{s} \subseteq \mathbf{n}, \forall t \in \mathcal{T},
\end{aligned} \tag{28}
$$

which proves theorem 1. □

Theorem 2 shows how we can recursively update the covariance matrix of $\mathbf{w}$.

Theorem 2. *The posterior covariance matrix of of* $\mathbf{w}$, $\hat{\Sigma}^t_{\mathbf{w}}$, *can be updated by Equation 29.*

$$
\hat{\Sigma}^t_{\mathbf{w}} = (\Phi(\mathbf{s})^T(\mathbf{K}_t(\mathbf{s},\mathbf{s}))^{-1}\Phi(\mathbf{s}) + (\Sigma^t_{\mathbf{w}})^{-1})^{-1}. \tag{29}
$$

Proof. According to the Bayesian Cramér-Rao bound [47], the mean square error matrix $\hat{\Sigma}^t_{\mathbf{w}}$ is bounded from below by the inverse of the Fisher information matrix J($\mathbf{w}$), where

$$
\hat{\Sigma}^t_{\mathbf{w}} = E[(\mathbf{w}-\hat{\mu}^t_{\mathbf{w}})(\mathbf{w}-\hat{\mu}^t_{\mathbf{w}})^T], \tag{30}
$$

and

$$
J(\mathbf{w}) = E[-\partial^2_{\mathbf{w}} \ln p_t(\mathbf{x}_t(\mathbf{s}),\mathbf{w})] \tag{31}
$$

where $\partial^2_{\mathbf{w}}$ denotes the second-order differential or Laplacian operator with respect to $\mathbf{w}$. Then, we have

$$
\hat{\Sigma}^t_{\mathbf{w}} \succeq J(\mathbf{w})^{-1}. \tag{32}
$$

Furthermore, we have:

$$
\begin{aligned}
J(\mathbf{w}) &= -E[\partial^2_{\mathbf{w}} \ln p_t(\mathbf{x}_t(\mathbf{s})|\mathbf{w})p_t(\mathbf{w})] \\
&= \Phi(\mathbf{s})^T(\mathbf{K}_t(\mathbf{s},\mathbf{s}))^{-1}\Phi(\mathbf{s}) + (\Sigma^t_{\mathbf{w}})^{-1},
\end{aligned} \tag{33}
$$

It is worth noting that, in Equation 33, the expectation is in respect to $\mathbf{w}$.

According to [6], under linear Gaussian condition, $\hat{\mu}^t_{\mathbf{w}}$ is a best linear unbiased estimator which can achieve the Cramér-Rao lower bound J($\mathbf{w}$).

Thus, we have:

$$
\begin{aligned}
\hat{\Sigma}^t_{\mathbf{w}} &= J(\mathbf{w})^{-1} \\
&= (\Phi(\mathbf{s})^T(\mathbf{K}_t(\mathbf{s},\mathbf{s}))^{-1}\Phi(\mathbf{s}) + (\Sigma^t_{\mathbf{w}})^{-1})^{-1},
\end{aligned} \tag{34}
$$

which proves theorem 2. □

It is worth noting that, $\hat{\Sigma}^t_{\mathbf{w}}$ only shows how well we can estimate $\mathbf{w}$. Theorem 3 below illustrates how well we can predict the performance of any network architectures.

Theorem 3. *The expected mean square error (MSE) matrix between the ground truth and the estimation can be obtained by Equation 35.*

$$
\begin{aligned}
&E\left[(\hat{\mathbf{x}}(\mathbf{s})-\mathbf{x}_t(\mathbf{s}))^T(\hat{\mathbf{x}}(\mathbf{s})-\mathbf{x}_t(\mathbf{s}))\right] \\
&= \Phi(\mathbf{s})\hat{\Sigma}^t_{\mathbf{w}}\Phi(\mathbf{s})^T + \mathbf{K}_t(\mathbf{s},\mathbf{s}),
\end{aligned} \tag{35}
$$

where $\forall \mathbf{s} \subseteq \mathbf{n}, \forall t \in \mathcal{T}$.

Proof. As mentioned before, $\mathbf{w}$ and $\mathbf{x}_t(\mathbf{s})$ conditioned on $\mathbf{w}$ are Gaussian distributed according to:

$$
p_t(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\hat{\mu}^t_{\mathbf{w}}, \hat{\Sigma}^t_{\mathbf{w}}), \tag{36}
$$

$$
p_t(\mathbf{x}_t(\mathbf{s})|\mathbf{w}) = \mathcal{N}(\mathbf{x}_t(\mathbf{s})|\Phi(\mathbf{s})\mathbf{w}, (\mathbf{K}_t(\mathbf{s},\mathbf{s})), \tag{37}
$$

where $\forall \mathbf{s} \subseteq \mathbf{n}, \forall t \in \mathcal{T}$.

Then, based on the affine transformation property of multivariate Gaussian distributions [44], the joint distribution of $\mathbf{w}$ and $\mathbf{x}_t(\mathbf{s})$ is given by:

$$
\begin{aligned}
p_t(\mathbf{w},\mathbf{x}_t(\mathbf{s})) &= \mathcal{N}\left(\begin{pmatrix}\mathbf{w}\\ \mathbf{x}_t(\mathbf{s})\end{pmatrix} \middle| \begin{pmatrix}\hat{\mu}^t_{\mathbf{w}}\\ \Phi(\mathbf{s})\hat{\mu}^t_{\mathbf{w}}\end{pmatrix}, \Sigma_*\right) \\
&= \mathcal{N}\left(\begin{pmatrix}\mathbf{w}\\ \mathbf{x}_t(\mathbf{s})\end{pmatrix} \middle| \begin{pmatrix}\hat{\mu}^t_{\mathbf{w}}\\ \hat{\mathbf{x}}(\mathbf{s})\end{pmatrix}, \Sigma_*\right),
\end{aligned} \tag{38}
$$

where

$$
\begin{aligned}
\Sigma_* &= \begin{pmatrix}\mathbf{K}^*_t(\mathbf{s},\mathbf{s}) + (\hat{\Sigma}^t_{\mathbf{w}})^{-1} & -\Phi(\mathbf{s})^T(\mathbf{K}_t(\mathbf{s},\mathbf{s}))^{-1} \\ -(\mathbf{K}_t(\mathbf{s},\mathbf{s}))^{-1}\Phi(\mathbf{s}) & (\mathbf{K}_t(\mathbf{s},\mathbf{s}))^{-1}\end{pmatrix}^{-1} \\
&= \begin{pmatrix}\hat{\Sigma}^t_{\mathbf{w}} & \hat{\Sigma}^t_{\mathbf{w}}\Phi(\mathbf{s})^T \\ \Phi(\mathbf{s})\hat{\Sigma}^t_{\mathbf{w}} & \Phi(\mathbf{s})\hat{\Sigma}^t_{\mathbf{w}}\Phi(\mathbf{s})^T + \mathbf{K}_t(\mathbf{s},\mathbf{s})\end{pmatrix},
\end{aligned} \tag{39}
$$

Algorithm 1: AEA

Initialize the prior mean vector and covariance matrix of $\mathbf{w}$, $\mu_{\mathbf{w}}^t$ and $\Sigma_{\mathbf{w}}^t$.
while *Algorithm not converge* **do**
  Sampling the best encoded network architecture at t according to Equation 12.
  Train the sampled network $\mathbf{s}_t \setminus \mathbf{s}_{t_p}$ and obtain the performance as a reward $\mathbf{x}_t(\mathbf{s}_t \setminus \mathbf{s}_{t_p})$.
  Estimate hyperparameters $\hat{\sigma}_t^i$, according to Equation 21 where $\mathbf{w}$ is set to $\mu_{\mathbf{w}}^t$.
  Set hyperparameters of kernel function in Equation 7 σ_t^i to $\hat{\sigma}_t^i, \forall i \in \{1, 2, \cdots, n\}$.
  Estimate the posterior mean vector of $\mathbf{w}$, $\hat{\mu}_{\mathbf{w}}^t$, according to theorem 1.
  Estimate the posterior covariance matrix of of $\mathbf{w}$, $\hat{\Sigma}_{\mathbf{w}}^t$, according theorem 2.
  Estimate the expected MSE matrix between the ground truth and the estimation, according to theorem 3.
  Updates t to next sampling time and set prior distribution at t to posterior distribution at t_p.
end

where

$$\mathbf{K}_t^*(\mathbf{s}, \mathbf{s}) = \Phi(\mathbf{s})^T (\mathbf{K}_t(\mathbf{s}, \mathbf{s}))^{-1} \Phi(\mathbf{s}). \tag{40}$$

Thus, we have:

$$\begin{aligned} &E\left[(\hat{\mathbf{x}}(\mathbf{s}) - \mathbf{x}_t(\mathbf{s}))^T (\hat{\mathbf{x}}(\mathbf{s}) - \mathbf{x}_t(\mathbf{s}))\right] \\ &= \Phi(\mathbf{s}) \hat{\Sigma}_{\mathbf{w}}^t \Phi(\mathbf{s})^T + \mathbf{K}_t(\mathbf{s}, \mathbf{s}), \\ &\forall \mathbf{s} \subseteq \mathbf{n}, \forall t \in \mathcal{T}, \end{aligned} \tag{41}$$

which proves theorem 3. □

Based on analysis above, we propose an alternating estimation algorithm (AEA) to estimate the hyperparameters of GP-NAS. Algorithm 1 shows the pseudocode of AEA. First, according to Equation 12, we can get the best encoded network architecture(s) to be estimated. Then, we train the sampled network architecture(s) and get performance(s). Based on the performance(s), we freeze $\mathbf{w}$ to estimate hyperparameters $\hat{\sigma}_t^i$ according to Equation 21, $\forall i \in \{1, 2, \cdots, n\}$. Alternatively, we freeze σ_t^i to estimate $\hat{\mu}_{\mathbf{w}}^t$ and $\hat{\Sigma}_{\mathbf{w}}^t$ according to theorem 1 and theorem 2. Finally, we estimate the expected mean square error (MSE) matrix between the ground truth and the estimation according to theorem 3. If the expected MSE matrix satisfies the setting threshold, then AEA converges and stops. Otherwise, the prior distribution at t is updated to the posterior distribution at t_p. Then, the AEA recursively estimates the hyperparameters. Therefore, the final number of sampled architectures is determined by theorem 3.

4. Experiments

In this section, we conduct experiments on CIFAR-10 [27], ImageNet [39] dataset and face recognition [14] tasks. First, the proposed GP-NAS searches for the best CNN architecture on CIFAR-10 for image classification. The search space and training details are presented. Next, we investigate the transferability of the best architecture learned on CIFAR-10 by evaluating it on ImageNet. Then, we analyse the predicted accuracy of GP-NAS. Finally, we search for the customized network for face recognition task.

4.1. Architecture Evaluation on CIFAR-10

Dataset CIFAR-10 [27] is a standard image classification dataset, which consists of 50,000 training images and 10,000 testing images of size $32 \times 32 \times 3$.

Search space Following [11], our search space is based on MobileNetV3-large[23]. Specifically, we search the kernel size $k \in \{3, 5, 7\}$, expansion rate $n \in \{3, 6\}$ of each inverted bottleneck block and whether the squeezing and excitation mechanism is enabled or not. In practice, we keep the same amount of layers and activation functions as MobileNetV3. Therefore, the search space contains 12^{14} child architectures, which is too large to enumerate.

Training During architecture searching, the child network is trained for 10 epochs on the V100 GPU. The learning rate is 0.1. Half of the training data is hold out for hyperparameters estimation. The final network is trained for 300 epochs. We set the learning rate to 0.05 with cosine decay strategy. We utilize SGD to optimize the network weights with momentum 0.9 and decay coefficient 3×10^{-4}. The batch size is 128. Data augmentations involve cutout and mixup.

Evaluation To verify the performance of GP-NAS, we compare our methods with state-of-the-art NAS methods, including RL-based methods [55, 1], EA-based methods [38, 31, 17], GD-based methods [32, 46] and other methods [30, 4, 35, 48]. Table 1 shows the test error, parameters and search cost of different methods. Although most NAS methods achieve comparable performances with moderate parameters on the testing set, RL-based and EA-based methods spend three orders of magnitude more computation resources (1800 GPU days for NASNet and 3150 GPU days for AmoebatNet). These results suggest that training the controller in the RL-Based NAS requires lots of network candidates, which is resource-consuming. Although some strategies [31, 30] have been proposed to accelerate their search speeds, the search cost still lags far behind other methods (225 GPU days for PNAS). Alternatively, SMASH[4] and GHN[48] reduce the training time of each child network by training a hypernetwork, which can directly generate the weights of network or predict the performance based on the architecture. In this way, the training of child networks can be avoided. Therefore, SMASH[4] finishes neural search in 1.5 GPU days, but the parameters increases to $16M$. Meanwhile, weight sharing strategy in ENAS[35] has been put forward where different

Method	Test Error (%)	Params (M)	Search Cost (GPU days)
NASNet-A [55]	2.65	3.3	1800
NASNet-B [55]	3.73	2.6	1800
NASNet-C [55]	3.10	3.59	1800
AmoebaNet-A [38]	3.34 ± 0.06	3.2	3150
AmoebaNet-B [38]	2.55 ± 0.05	2.8	3150
Hierarchical Evo [31]	3.75 ± 0.12	15.7	300
PNAS [30]	3.41 ± 0.09	3.2	225
Macro NAS + Q-Learning [1]	6.92	11.2	100
SMASH [4]	4.03	16.0	1.5
ENAS [35]	2.89	4.6	0.45
DARTS (first order) [32]	2.94	2.9	1.5
DARTS (second order) [32]	2.83 ± 0.06	3.4	4
SNAS (single-level) + mild constraint [46]	2.98	2.9	1.5
GHN Top-Best, 1K (F=32) [48]	2.94 ± 0.07	5.7	0.84
LEMONADE [17]	2.58	13.1	-
ProxylessNAS [5]	2.08	-	4.0
GP-NAS-rdm	3.98	4.2	23
GP-NAS	3.79	3.90	0.9

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

Method	FLOPs $\times 10^6$	Accuracy Top 1	Top 5
NASNet-A [55]	564	74.0	91.6
NASNet-B [55]	488	72.8	91.3
NASNet-C [55]	558	72.5	91.0
AmoebaNet-A [38]	555	74.5	92.0
AmoebaNet-B [38]	555	74.0	91.5
AmoebaNet-C [38]	570	75.7	92.4
PNAS [30]	588	74.2	91.9
DARTS (second order) [32]	595	73.1	91.0
SNAS (mild constraint) [46]	522	72.7	90.8
GHN Top-Best, 1K [48]	569	73.0	91.3
ProxylessNAS (GPU) [5]	465	75.1	92.5
GP-NAS	225	73.4	91.3
GP-NAS*	225	75.8	92.8

Table 2. Comparison with the state-of-the-art NAS methods on ImageNet-Mobile.$*$ indicates the results trained using larger batch size and epochs.

child network shares weighs, which is orthogonal to our work. Differentiable architecture search [32] converts the discrete search space to continue one and makes full use of the efficiency of gradient descent. DARTS (first order) can search a good network within 1.2 GPU days. It is obvious that most of these methods try to speed neural architecture search by reducing the training time of each child network. Different from them, our GP-NAS combines mutual information and Bayes estimation to efficiently estimate hyper-parameter with minimum sampling networks. In this way, the number of training child network is greatly decreased. To verify the effectiveness of mutual information in GP-NAS, we replace the sampling strategy Eq.12 in algorithm 1 with random sampling, called GP-NAS-rdm. Although GP-NAS-rdm costs about 23 GPU days, it still achieves 3.98% test error with $4.2M$ parameters. Thus, it is reasonable to learn the distribution of performance under network architecture by the specially designed GP. When we adopt the sampling strategy based on maximizing mutual information, our GP-NAS only requires 0.9 GPU days, which is about 25 times faster than GP-NAS-rdm. Furthermore, GP-NAS also achieves a comparable performance 3.79% test error with only $3.9M$ parameters.

4.2. Transferability of learned Architectures of ImageNet

To validate the effectiveness of GP-NAS, we also evaluate it on the ImageNet datasets that contain 1.28 million training 224×224 images. Following [48, 32], the ImageNet mobile setting restricts the model size to be under $600M$ FLOPS. With the tradeoff between the performance and model size, we transfer the network architecture of GP-NAS searched on CIFAR-10 dataset to the ImageNet dataset. The final model is trained for 150 epochs with batch size 256. The initial learning rate is 0.045 with cosine decay strategy. We report the top-1 and top-5 ac-

curacy on the validation set. As shown in Table. 2, our GP-NAS achieves 73.4% and 91.3% accuracy of Top1 and Top5, which is competitive with other advanced methods. Meanwhile, the architecture searched by GP-NAS only has $225M$ FLOPs, which are half of other methods, such as DARTS and SNAS. When the batch size and epochs increase to 4096 and 360, GP-NAS obtains 75.8% and 92.8% accuracy of Top1 and Top5.

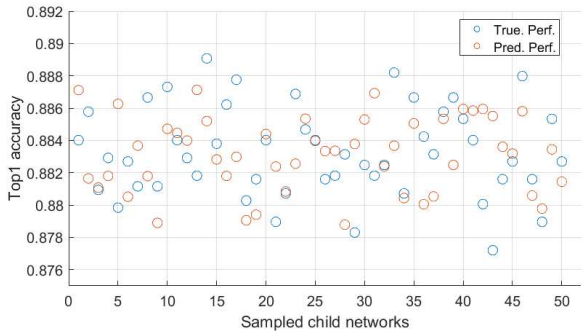

Figure 1. Predicting vs. True accuracies on CIFAR-10 of different models.

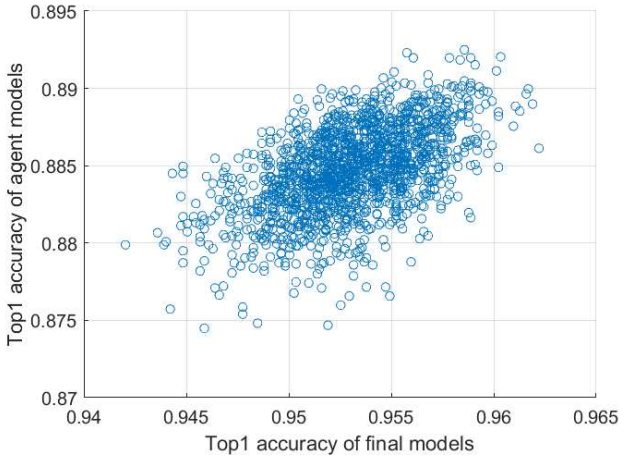

Figure 2. Final models vs. agent models accuracies on CIFAR-10.

4.3. Predicted Performance Correlation

In this section, we evaluate the predicted performance of child networks by GP-NAS on CIFAR-10 dataset. After finishing the hyperparameters estimation, we randomly sample child networks and train them following the training strategy in the search stage. Trained networks are evaluated on the validation set, which is treated as truth accuracy, while their performance is directly predicted by GP-NAS. Fig. 1 shows the predicted accuracy and truth accuracy. We show 50 networks for a clear illustration. It is obvious that GP-NAS can predict accuracy well. Moreover, we also fully train the selected models in the stand-alone way. Fig.2 shows the accuracy of stand-alone model vs. agent model. We observe that the performances of stand-alone and agent model have a high correlation.

4.4. Architecture Search on Face Recognition

Face Recognition is a fundamental task in computer vision. To evaluate the generalization of GP-NAS, we search a neural network structure for this recognition field. Following [7], we search network architectures on CASIA-Webface dataset and evaluate face verification accuracy on LFW dataset. Our search space is based on MobileFaceNet [7] with width multipliers of 0.75. We search expansion rate $n \in \{2, 4, 6\}$ and whether the squeezing and excitation mechanism is enabled or not for each block. The search space contains 6^{15} child networks. Table. 3 shows the FLOPs, parameters and accuracy of each method. Compared with MobileFaceNet, model searched by GP-NAS gains higher performance 99.17% with only $151M$ FLOPs and $0.83M$ parameters. When the width multipliers are set to 0.6, we can search a more compact network.

Method	FLOPs	Params	Acc.
MobileNetV1 [24]	-	3.2M	98.63%
MobileNetV2 [40]	-	2.1M	98.58%
ShuffleNet(1×,g=3)[49]	-	0.83M	98.70%
MobileNetV2-GDConv	-	2.1M	98.88%
MobileFaceNet [7]	223M	0.98M	99.15%
GP-NAS	151M	0.83M	99.17%
GP-NAS*	130M	0.61M	99.13%

Table 3. Performance comparison among mobile models tested on LFW. $*$ indicates width multipliers of 0.6.

5. Conclusion

In this paper, we propose the Gaussian Process based Neural Architecture Search (GP-NAS), a theoretical modeling for NAS. Jointly considering mutual information and Bayesian estimation, we can estimate the hyperparameters of GP-NAS with the minimal number of sampled networks, which significantly accelerates the search speed. We also propose an alternating estimation algorithm (AEA) to recursively update the hyperparameters. The learned GP-NAS is capable of inferring the performance of any network architectures. Finally, we show that GP-NAS achieves a competitive performance on classification and face recognition tasks.

Acknowledgement: This work is funded by Beijing Natural Science Foundation (Grants No. JQ18017) and Shandong Provincial Key Research and Development Program (Major Scientific and Technological Innovation Project)(No. 2019JZZY010119).

References

[1] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017. 2, 6, 7

[2] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V Le. Neural optimizer search with reinforcement learning. In *ICML*, 2017. 2

[3] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *ICML*, 2018. 2, 4

[4] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. In *ICLR*, 2018. 2, 6, 7

[5] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019. 1, 7

[6] James V Candy. *Bayesian signal processing: classical, modern, and particle filtering methods*. John Wiley & Sons, 2016. 5

[7] Sheng Chen, Yang Liu, Xiang Gao, and Zhen Han. Mobilefacenets: Efficient cnns for accurate real-time face verification on mobile devices. In *CCBR*, 2018. 8

[8] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NeurIPS*, 2016. 2, 4

[9] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. 2019. 2

[10] Yukang Chen, Gaofeng Meng, Qian Zhang, Shiming Xiang, Chang Huang, Lisen Mu, and Xinggang Wang. Renas: Reinforced evolutionary neural architecture search. In *CVPR*, 2019. 2

[11] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. Moga: Searching beyond mobilenetv3. *arXiv preprint arXiv:1908.01314*, 2019. 6

[12] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012. 4

[13] Jiequan Cui, Pengguang Chen, Ruiyu Li, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Fast and practical neural architecture search. In *ICCV*, 2019. 1

[14] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *CVPR*, 2019. 6

[15] Xuanyi Dong and Yi Yang. One-shot neural architecture search via self-evaluated template network. In *ICCV*, 2019. 2

[16] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *CVPR*, 2019. 1

[17] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In *ICLR*, 2019. 6, 7

[18] Shuyang Gao, Greg Ver Steeg, and Aram Galstyan. Variational information maximization for feature selection. In *NeurIPS*, 2016. 2, 4

[19] Golnaz Ghiasi, Tsung-Yi Lin, Ruoming Pang, and Quoc V Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. 2019. 1

[20] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. In *ICLR*, 2017. 2

[21] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *ICLR*, 2019. 2, 4

[22] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *NeurIPS*, 2016. 2

[23] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. In *ICCV*, 2019. 6

[24] Andrew Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. 2017. 8

[25] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *LION*, 2011. 2

[26] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *NeurIPS*, 2018. 2

[27] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 6

[28] Xin Li, Yiming Zhou, Zheng Pan, and Jiashi Feng. Partial order pruning: for best speed/accuracy trade-off in neural architecture search. In *CVPR*, 2019. 3

[29] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. 2019. 1

[30] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018. 6, 7

[31] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *ICLR*, 2018. 1, 2, 6, 7

[32] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *ICLR*, 2019. 2, 6, 7

[33] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *NeurIPS*, 2018. 2

[34] Vladimir Nekrasov, Hao Chen, Chunhua Shen, and Ian Reid. Fast neural architecture search of compact semantic segmentation models via auxiliary cells. In *CVPR*, 2019. 1

[35] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. 2018. 1, 6, 7

[36] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Faster discovery of neural architectures by searching for paths in a large model. In *ICLRW*, 2018. 2

[37] Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning*. MIT Press, Cambridge, Mass, 2006;. 2, 3

[38] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019. 1, 6, 7

[39] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 2015. 6

[40] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 8

[41] Simo Särkkä. *Bayesian filtering and smoothing*. Cambridge University Press, 2013. 4

[42] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *NeurIPS*, 2012. 2

[43] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019. 2

[44] Yung Liang Tong. *The multivariate normal distribution*. Springer Science & Business Media, 2012. 5

[45] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019. 1

[46] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. In *ICLR*, 2019. 2, 6, 7

[47] Dave Zachariah and Petre Stoica. Cramér-rao bound analog of bayes' rule [lecture notes]. *IEEE Signal Processing Magazine*, 2015. 5

[48] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. In *ICLR*, 2019. 2, 6, 7

[49] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. 07 2017. 8

[50] Yizhe Zhang, Michel Galley, Jianfeng Gao, Zhe Gan, Xiujun Li, Chris Brockett, and Bill Dolan. Generating informative and diverse conversational responses via adversarial information maximization. In *NeurIPS*, 2018. 2

[51] Yiheng Zhang, Zhaofan Qiu, Jingen Liu, Ting Yao, Dong Liu, and Tao Mei. Customizable architecture search for semantic segmentation. In *CVPR*, 2019. 1

[52] Xiawu Zheng, Rongrong Ji, Lang Tang, Baochang Zhang, Jianzhuang Liu, and Qi Tian. Multinomial distribution learning for effective neural architecture search. In *ICCV*, 2019. 1

[53] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *CVPR*, 2018. 1, 2

[54] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 2

[55] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018. 1, 4, 6, 7