InstaNAS: Instance-Aware Neural Architecture Search

An-Chieh Cheng,[1*] Chieh Hubert Lin,[1*] Da-Cheng Juan,[2] Wei Wei,[2] Min Sun[1,3,4]
[1]National Tsing-Hua University, Hsinchu, Taiwan
[2]Google Research, Mountain View, USA, [3]Appier Inc., Taiwan
[4]MOST Joint Research Center for AI Technology and All Vista Healthcare, Taiwan
{accheng.tw, hubert052702}@gmail.com, sunmin@ee.nthu.edu.tw,
{dacheng, wewei}@google.com

Abstract

Conventional Neural Architecture Search (NAS) aims at finding a *single* architecture that achieves the best performance, which usually optimizes task related learning objectives such as accuracy. However, a single architecture may not be representative enough for the whole dataset with high diversity and variety. Intuitively, electing domain-expert architectures that are proficient in domain-specific features can further benefit architecture related objectives such as latency. In this paper, we propose InstaNAS—an instance-aware NAS framework—that employs a controller trained to search for a “distribution of architectures” instead of a single architecture; This allows the model to use sophisticated architectures for the difficult samples, which usually comes with large architecture related cost, and shallow architectures for those easy samples. During the inference phase, the controller assigns each of the unseen input samples with a domain expert architecture that can achieve high accuracy with customized inference costs. Experiments within a search space inspired by MobileNetV2 show InstaNAS can achieve up to 48.8% latency reduction without compromising accuracy on a series of datasets against MobileNetV2.

1 Introduction

Neural Architecture Search (NAS) has become an effective and promising approach to automate the design of deep learning models. It aims at finding the optimal model architectures based on their performances on evaluation metrics such as accuracy (Zoph and Le 2017). One popular way to implement NAS is to employ reinforcement learning (RL) that trains an RNN controller (or “agent”) to learn a search policy within a pre-defined search space. In each iteration of the search process, a set of child architectures are sampled from the policy, and evaluate performance on the target task. The performance is then used as the reward to encourage the agent to prioritize child architectures that can achieve a higher expected reward. In the end, a single architecture with a maximum reward will be selected and trained to be the final solution of the task.

[*]indicates equal contribution.

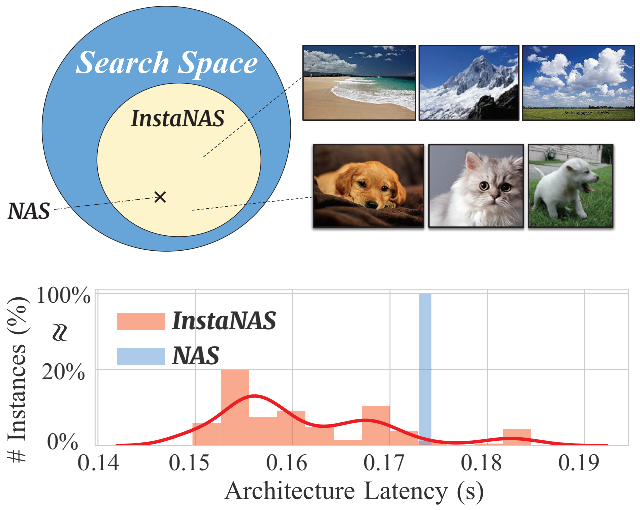

Figure 1: InstaNAS searches for a ***distribution*** of architectures instead of a single one from conventional NAS. We showcase a distribution of architecture latencies found by InstaNAS for CIFAR-10. The InstaNAS controller assigns each input instance to a domain expert architecture, which provides customized latency for different domains of data.

Although a single architecture searched using NAS seems to be sufficient to optimize task related metrics such as accuracy, its performance is largely constrained in architecture related metrics such as latency and energy. For example in a multi-objective setting where both accuracy and latency are concerned, NAS is constrained to come up with a single model to explore the trade-off between accuracy and latency for all samples. In practice, however, difficult samples require complicated and usually high latency architectures whereas easy samples work well with shallow and fast architectures. This inspires us to develop InstaNAS, a NAS framework which searches for a ***distribution*** of architectures instead of a single one. Each architecture within the final distribution is an expert of one or multiple specific domains, such as different difficulty, texture, content style and speedy inference. For each sample, the controller is trained

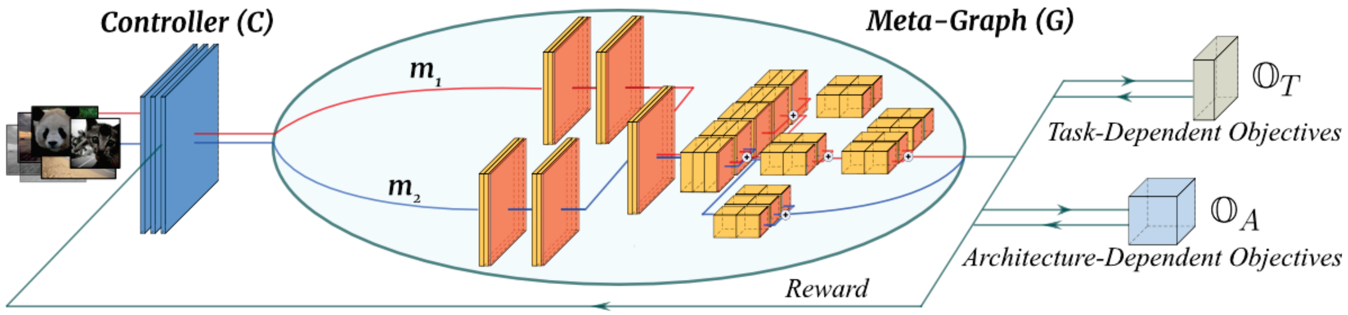

Figure 2: InstaNAS controller (C) selects an expert child architecture (m) from the meta-graph (G) for each input instance while considering task-dependent objectives ($\mathbb{O}_T$) (*e.g.*, accuracy) and architecture-dependent objectives ($\mathbb{O}_A$) (*e.g.*, latency).

to select a suitable architecture from its distribution. With basic components being shared across architectures, weights can be re-used toward architectures that have never been selected before. The InstaNAS framework allows samples to have their own architectures, making it flexible to optimize architecture related objectives.

InstaNAS is critical in many of the recently proposed settings such as multi-objective NAS (Dong et al. 2018; Tan et al. 2018), which optimizes not only task-dependent metrics such as accuracy but also those metrics that are architecture-dependent such as latency. In particular, the controller of InstaNAS has the capability of selecting the architectures by considering the variations among instances. To enable effective training, we introduce a dynamic reward function to gradually increase the difficulty of the environment, a technique commonly found in curriculum learning.

In the meanwhile, the reward interval slowly decreases its upper bound through epochs. Note that InstaNAS also aligns with the concept of conditional computing (i.e., mixture of experts) since the instance-level architecture depends on the given input sample. Most importantly, InstaNAS elegantly combines the ideas of NAS and conditional computing which learns a distribution of architectures and a controller to generate instance-level architectures.

In conclusion, the main contributions of this paper are as the following: We propose InstaNAS , the first instance-aware neural architecture search framework that generates architectures for each individual sample. Instance-awareness allows us to incorporate the variability of samples into account by designing architectures that specifically optimize each sample. To the best of our knowledge, we are the first work toward building NAS with instance-awareness. We show that InstaNAS is able to out-perform MobileNetV2 dramatically in terms of latency while keeping the same performance. Experimental results illustrate an average of 48.9%, 40.2%, 35.2% and 14.5% latency reduction with comparable accuracy on CIFAR-10, CIFAR-100, TinyImageNet and ImageNet, respectively. Further latency reduction of 26.5% can be achieved on ImageNet if a moderate accuracy drop ($\simeq$ 0.7%) is allowed.

2 Related Work

Neural Architecture Search. Neural Architecture Search (NAS) has emerged growing interest in the field of AutoML and meta-learning in recent years. Seminal work by (Zoph and Le 2017) first proposed "Neural Architecture Search (NAS)" using reinforcement learning algorithm. They introduce a learnable RNN controller that generates a sequence of actions representing a child network within a predefined search space, while the validation performance is taken as the reward to train the controller. Since the process of NAS can also be framed as a natural selection problem, some works (Real et al. 2017; 2018) propose to use evolutionary algorithms to optimize the architecture. However, all these works focus on optimizing model accuracy as their only objective. In real-world, these models may not be suitable for being deployed on certain (*e.g.*, latency-driven) applications, such as mobile applications and autonomous car.

Multi-objective Neural Architecture Search. For better flexibility and usability in real-world applications, several works are dedicated to extending NAS into multiple-objective neural architecture search, which attempts to optimize multiple objectives while searching for architectures. (Elsken, Metzen, and Hutter 2018) and (Zhou et al. 2018) use FLOPs and the number of parameters as the proxies of computational costs; (Kim et al. 2017) and (Tan et al. 2018) directly minimized the actual inference time; (Dong et al. 2018) proposed to consider both device-agnostic objectives (*e.g.*, FLOPs) and device-related objectives (*e.g.*, inference latency) using Pareto Optimization. However, all these algorithms only consider searching for a single final architecture achieving the best average accuracy for the given task. In contrast, InstaNAS is an MO-NAS approach that searches for a distribution of architectures aiming to speed up the average inference time with *instance-awareness*.

One-shot Architecture Search. Computational expensive is another fundamental challenge in NAS, conventional NAS algorithms require thousands of different child architectures to be trained from scratch and evaluated, which is often time costly. One-shot architecture search is an approach using share-weight across child architectures to amortize the search cost. The concept of weight sharing has been widely adopted by different NAS approaches with various kinds of search strategies: with evolutionary algorithm (Real et al. 2018; 2017), reinforcement learning (Pham et al. 2018), gradient descent (Liu, Simonyan, and Yang 2018), and random search (Bender et al. 2018). Instead of training each child architecture from scratch, they allow child architectures to share weights whenever possible. We also adopt the similar design principle of the one-shot architecture search to not only accelerate InstaNAS but also to reduce the total number of parameters in InstaNAS . We will explain further detail of how we leverage the one-shot architecture search to build our meta-graph in Section 3.2.

Conditional Computation. Several conditional computation (*i.e.*, mixture of experts) methods have been proposed to dynamically execute different modules of a model on a per-example basis (Shazeer et al. 2017; Bengio et al. 2015; Liu and Deng 2017; Teja Mullapudi et al. 2018; Wu et al. 2018; Veit and Belongie 2018). More specifically, Block-Drop(Wu et al. 2018) and ConvNet-AIG(Veit and Belongie 2018) are two state-of-the-art conditional computing methods that generate a series of decision which selectively dropped a subset of blocks in a well-known baseline hyper-network (*e.g.*, ResNet) with respect to each input. However, conditional computing is very different from NAS in many perspectives. Conditional computing assumes whole hyper-network typically achieves the best accuracy, and dropping computation implies sacrificing accuracy. For InstaNAS, the meta-graph does not achieve the best accuracy. Conditional computing requires a human-designed hyper-network to begin with the trimming process. In contrast, InstaNAS is NAS, which only requires a manually defined set of operations to form a search space. Moreover, The total number of unique computation configurations in conditional computing is much smaller than the total number of unique architectures in our search space. In practice, the experimental results demonstrate that InstaNAS discovers a more diverse and complex set of architectures.

3 InstaNAS: Instance-aware NAS

In this section, we first give the overview of InstaNAS, specifically about the meta-graph and the controller. Then, we describe how the meta-graph is constructed and pre-trained. Finally in Section 3.3, we explain how to design the multi-objective reward function for training the controller and updating the meta-graph.

3.1 Overview

InstaNAS contains two major components: a meta-graph and a controller. The meta-graph is a directed acyclic graph (DAG), with one source node (where an input image is fed) in the beginning and one sink node (where the prediction is provided) at the end; every node between the source and sink is a computing module such as a convolutional operation, and an edge connects two nodes meaning the output of one node is used as the input of the other. With this meta-graph representation, every path from source to sink can be treated as a valid child architecture as an image classifier. Therefore, the meta-graph can be treated as the set containing all possible child architectures.

The other major component of InstaNAS is the controller; it is designed and trained to be *instance-aware* and optimize for multi-objective. Particularly, for each input image, the controller selects a child architecture (*i.e.*, a valid path in the meta-graph) that accurately classifies that image, and at the same time, minimizes the inference latency (or other computational costs). Therefore, the controller is trained to achieve two objectives at the same time: maximize the classification accuracy (referred as the task-dependent objective $\mathbb{O}_T$) and minimize the inference latency (referred as the architecture-dependent objective $\mathbb{O}_A$). Note that $\mathbb{O}_A$ can also be viewed as a constraint when optimizing for $\mathbb{O}_T$.

Next, the training phase of InstaNAS consists of three stages: (a) "pre-train" the meta-graph, (b) "jointly train" both controller and the meta-graph, and (c) "fine-tune" the meta-graph. In the first stage, the meta-graph (denoted as G, parametrized by Θ) is pre-trained with $\mathbb{O}_T$. In the second stage, a controller (denoted as C, parametrized by ϕ) is trained to select a child architecture $m(x;\theta_x) = C(x, G; \phi)$ from G for each input instance x. In this stage, the controller and the meta-graph are trained in an interleaved fashion: train the controller with the meta-graph fixed in one epoch and vice versa in another epoch. This training procedure enforces the meta-graph to adapt to the distribution change of the controller. Meanwhile, the controller is trained by policy gradient with a reward function R which is aware of both $\mathbb{O}_T$ and $\mathbb{O}_A$. The training detail of the controller is described in Section 3.3. In the third stage, after the controller is trained, we fix the controller and focus on fine-tuning the meta-graph for the task-dependent objective $\mathbb{O}_T$; specifically, for each input image the controller selects a child architecture (*i.e.*, a certain path in the meta-graph), and that child architecture is trained to optimize for $\mathbb{O}_T$. After the child architecture is trained, the corresponding nodes of the meta-graph are updated accordingly.

During the inference phase, $m(x;\theta_x) = C(x, G; \phi)$ is applied to each unseen input instance x. The generated $m(x;\theta_x)$ is an architecture that tailored for each x and best trade-offs between $\mathbb{O}_T$ and $\mathbb{O}_A$. Note that the latencies we reported in Section 4 has included the controller latency, since the controller is applied for each inference.

3.2 Meta-Graph

Meta-graph is a weight-sharing mechanism designed to represent the search space of all possible child architectures with two important properties: (a) any end-to-end path (from source to sink) within the meta-graph is a valid child architecture, and (b) the performance (*e.g.*, accuracy or latency) of this child architecture, without any further training, serves as a good proxy for the final performance (*i.e.*, fully-trained performance). Without using the meta-graph, a straightforward approach of constructing instance-aware classifier might be: train many models, then introduce a controller to assign each input instance to the most suitable model. This approach is not feasible since the total number of parameters in the search space grows linearly w.r.t. the number of models considered, which is usually a very large number; for example, in this work, the search space contains 10^{25} child architectures. Therefore, InstaNAS adapts the meta-graph to reduce the total number of parameters via weight sharing; specifically, if two child architectures share any part of the meta-graph, only one set of parameters required to represent the shared sub-graph.

Next, we explain how the meta-graph is pre-trained. At the beginning of every training iteration, part of the meta-graph is randomly zero out (also called "drop-path" in (Bender et al. 2018)), and the rest of modules within the meta-graph forms a child architecture. Then this child architecture is trained to optimize $\mathbb{O}_T$ (*e.g.*, classification accuracy) and updates the weights of the corresponding part of the meta-graph. Note that the "drop-path" rate is a hyper-parameter between [0, 1]. The drop-path rate that the meta-graph trained with will affect how the controller explores the search space in the early stage. In this work, we achieve good results by linearly increasing the drop-path rate from the middle of pre-training and eventually reach to 50%.

3.3 Controller

InstaNAS controller is different to the one in conventional NAS that aims at training for effectively exploring in the search space. Given an input image, the InstaNAS controller proposes a child architecture by $m(x; \theta_x) = C(x; \phi)$. Therefore, during the inference phase, the controller is still required, and the design principle of the controller is to be fast since its latency is included as part of the inference procedure. The controller is responsible for capturing the low-level representations (*e.g.*, the overall color, texture complexity, and sample difficulty) of each instance, then dispatches the instance to the proposed child architecture that can be treated as the domain expert to make accurate decisions. In this work, we use a three-layer convolutional network with large kernels as the implementation of a InstaNAS controller. Qualitative analysis and visualizations of how the controller categorizes samples are provided in Section 4.3 (see Figure 6 for example).

Next, we elaborate on the exploration strategy and reward function to train the controller.

Exploration Strategy. We formulate each architecture to be a set of binary options indicating whether each convolutional kernel within the meta-graph is selected. The controller takes each input image and generates a probability vector $\boldsymbol{p}$ indicating the probability of selecting a certain convolutional kernel. Then Bernoulli sampling is applied to this probability vector for exploring the architecture space. We adopt the entropy maximization mentioned in (Mnih et al. 2016), which improves exploration by encouraging a more deterministic policy (either select or not select a kernel). To further increase the diversity of sampling result during exploring the architecture space, we adopt the encouragement parameter α described in (Wu et al. 2018) which mutates the probability vector by $\boldsymbol{p}' = \alpha \cdot \boldsymbol{p} + (1 - \alpha) \cdot (1 - \boldsymbol{p})$.

The controller is trained with policy gradient. Similar to training procedure proposed in (Zoph and Le 2017; Wu et al. 2018), we introduce a "self-critical" baseline (Rennie et al. 2017) $R(\widetilde{p})$ to reduce the variance of instance-wise reward $R(p')$, where $\widetilde{p}_i = 1$ if $p > 0.5$, and $\widetilde{p}_i = 0$ otherwise. The policy gradient is estimated by:

$$\nabla_\phi J = \mathbb{E}[(R(p') - R(\widetilde{p}))\nabla_\phi \sum_i \log P(a_i)] \,, \tag{1}$$

which ϕ is the parameters of the controller and each a_i is sampled independently by a Bernoulli distribution with respect to $p_i \in \boldsymbol{p}$.

Reward Function. The reward function is designed to be multi-objective that takes both $\mathbb{O}_T$ and $\mathbb{O}_A$ into account. The reward is calculated as:

$$R = \begin{cases} R_T \cdot R_A & \text{if } R_T \text{ is positive,} \\ R_T & \text{otherwise,} \end{cases} \tag{2}$$

which R_T and R_A are obtained from $\mathbb{O}_T$ and $\mathbb{O}_A$. The design of R is based on the observation that $\mathbb{O}_T$ is generally more important and preferred than $\mathbb{O}_A$. As a result, $\mathbb{O}_A$ is only taken into account when $\mathbb{O}_T$ is secured. Otherwise, the controller is first ensured to maximize R_T. Even for the cases where R_T is positive, R_A is treated to be "preferred" (not enforced), which is done by normalizing R_A to the range $[0, 1]$ that becomes a discounting factor to R_T and never provides negative penalties to the controller through policy gradient.

Another observation is that optimizing $\mathbb{O}_A$ is generally challenging to optimize, which at times collapses the controller training. One possible reason is: take R_T to be accuracy and R_A to be latency as an example, architectures with both good latency and desirable accuracy are rare. Meanwhile, our "instance aware" setting collects reward in a "instance-wise" manner, finding architectures with extremely low latency for ***all*** samples (trivially selecting most simple kernels) is significantly easier than having generally high accuracy for any sample. Therefore, in the early stage of the controller exploration, such pattern encourages the controller to generate shallow architectures and directly ignores accuracy. Eventually, the policy collapses to a single architecture with extremely low latency with a poor accuracy close to random guessing.

To address the aforementioned problem, we propose a training framework using "dynamic reward." Dynamic reward encourages the controller to satisfy a gradually decreasing latency reward with bounds (upper-bound U_t and lower-bound L_t, which t is the number of epochs) during search time. The idea of dynamic reward shares a similar concept with curriculum learning (Bengio et al. 2009), except that we aim at gradually increasing the task difficulty to avoid the sudden collapsing. In this work, we propose the reward R_A to be a quadratic function parametrized by U_t and L_t. For each sample, we measure architecture-related performance z, then calculate $R_A = -\frac{1}{\gamma}\,(z - U_t) \times (z - L_t)$, which γ is a normalization factor that normalizes R_A to the range $[0, 1]$ by $\gamma = (\frac{U_t - L_t}{2})^2$. Such a design (quadratic function) encourages the controller to maintain the expectation of $\mathbb{O}_A$ near the center of the reward interval, while still be aware of

$\mathbb{O}_T$. Otherwise, the child architectures may fall outside the reward interval upon the reward interval changes.

4 Experiments

In this section, we explain and analyze the building blocks of InstaNAS. We first demonstrate some quantitative results of InstaNAS against other models. Then we visualize and discuss some empirical insights of InstaNAS. Throughout the experiments, we use the same search space. For the search objectives, we choose accuracy as our task-dependent objective and latency as the architecture-dependent objective, which are the most influential factors of architecture choice in real-world applications.

4.1 Experiment Setups

Search Space. We validate InstaNAS in a search space inspired by (Tan et al. 2018), using MobileNetV2 as the backbone network. Our search space consists of a stack of 17 cells, each cell has five module choices. Specifically, we allow one basic convolution *(BasicConv)* and four mobile inverted bottleneck convolution *(MBConv)* layers with various kernel sizes $\{3, 5\}$ and filter expansion ratios $\{3, 6\}$ as choices in the cell, which has $2^5 = 32$ combinations. Different from (Dong et al. 2018; Zoph et al. 2018), we do not restrict all cells to share the same combination of choices. Therefore, across the entire search space, there are $32^{17} \simeq 10^{25}$ child architecture configurations.

Module Latency Profiling. In the instance-aware setting, evaluating the latency reward is a challenging task as each input instance is possibly assigned to different child architectures. However, measuring the latency individually for each child architecture is considerably time costly during training. Therefore, to accelerate training, we evaluate the latency reward with estimated values. Specifically, we build up module-wise look-up tables with pre-measured latency consumption of each module. For each sampled child architecture, we look up the table of each module and accumulate the layer-wise measurements to estimate the network-wise latency consumption.

4.2 Quantitative Results

Experiments on CIFAR-10/100. We validate InstaNAS on CIFAR-10/100 with the search space described in the previous section. Across all training stages, we apply random copping, random horizontal flipping, and cut-out (DeVries and Taylor 2017) as data augmentation methods. For pre-training the meta-graph, we use Stochastic Gradient Descent optimizer with initial learning rate 0.1. After the joint training ends, some controllers are picked by human preference by considering the accuracy and latency trade-off. At this point, the accuracy measured in the joint training stage can only consider as a reference value, the meta-graph needs to re-train from scratch with respect to the picked policy. We use Adam optimizer with learning rate 0.01 and decays with cosine annealing.

[2]For a fair comparison, all CPU latencies are measured in the same work station and the same framework (PyTorch v1.0.0).

Table 1: InstaNAS shows competitive latency and accuracy trade-off in CIFAR-10 (Krizhevsky and Hinton 2009) against other state-of-the-art human-designed models (first row) and NAS-found models (second row). All five InstaNAS models are all obtained within a single search, and the controller latency is already included in the reported latency. Note that we measure the model's error rates with our implementation if it is not reported in the original paper. [2]

Model	Err. (%)	Latency
ResNet50 (He et al. 2016)	6.38	0.051
ResNet101 (He et al. 2016)	6.25	0.095
ShuffleNet v2 1.0× (Ma et al. 2018)	7.40	0.035
ShuffleNet v2 1.5× (Ma et al. 2018)	6.36	0.052
IGCV3-D 1.0× (Sun et al. 2018)	5.54	0.185
IGCV3-D 0.5× (Sun et al. 2018)	5.27	0.095
NASNet-A (Zoph et al. 2018)	3.41	0.219
DARTS (Liu, Simonyan, and Yang 2018)	2.83	0.236
DPP-Net-$Mobile$ (Dong et al. 2018)	5.84	0.062
MobileNet v2 0.4× (Sandler et al. 2018)	7.44	0.038
MobileNet v2 1.0× (Sandler et al. 2018)	5.56	0.092
MobileNet v2 1.4× (Sandler et al. 2018)	4.92	0.129
InstaNAS-C10-A	**4.30**	0.085
InstaNAS-C10-B	**4.50**	0.055
InstaNAS-C10-C	5.20	**0.047**
InstaNAS-C10-D	6.00	**0.033**
InstaNAS-C10-E	8.10	**0.016**

Table 1 shows the quantitative comparison with state-of-the-art efficient classification models and NAS-found architectures. The result suggests InstaNAS is prone to find good trade-off frontier relative to both human-designed and NAS-found architectures. In comparison to MobileNetV2 (1.0×), which the search space is referenced to, InstaNAS-C10-A improves accuracy by 1.26% without latency trade-off; InstaNAS-C10-C reduces 48.9% average latency with comparable accuracy, and InstaNAS-C10-E reduces 82.6% latency with moderate accuracy drop. Note that these three variances of InstaNAS are all obtained within a single search, then re-train from scratch individually.

Our results on CIFAR-100 are shown in Table 2, which the average latency consistently shows reduction - with 40.2% comparing to MobileNetV2, 36.1% comparing to ShuffleNetV2. InstaNAS stably shows overall improvement in the trade-off frontier against competitive models.

Experiments on TinyImageNet and ImageNet. To validate the scalability, stability and generalization of InstaNAS, we evaluate our approach on two more fine-grained datasets, TinyImageNet and ImageNet. We ran the experiment using directly the same set of hyperparameters configuration from the CIFAR-10/100 experiment. As shown in Table 3 and Table 4, InstaNAS comparing to MobileNetV2, again, found accurate model with 35.2% latency reduction on TinyImageNet and 14.5% on ImageNet. Furthermore, if moderate accuracy drop ($\simeq$ 0.7%) is tolerable, InstaNAS can further achieve 26.5% average latency reduction on ImageNet.

Table 2: InstaNAS consistently provides significant accuracy improvement and latency reduction on CIFAR-100. Again, all the InstaNAS variants are obtained within a single search.

Model	Err. (%)	Latency
ShuffleNet v2 1.0× (Ma et al. 2018)	30.60	0.035
ShuffleNet v2 1.5× (Ma et al. 2018)	28.30	0.052
ShuffleNet v2 2.0× (Ma et al. 2018)	28.88	0.072
MobileNet v2 0.4× (Sandler et al. 2018)	30.72	0.049
MobileNet v2 1.0× (Sandler et al. 2018)	27.00	0.092
MobileNet v2 1.4× (Sandler et al. 2018)	25.66	0.129
InstaNAS-C100-*A*	**24.20**	0.089
InstaNAS-C100-*B*	**24.40**	0.086
InstaNAS-C100-*C*	**24.90**	0.065
InstaNAS-C100-*D*	26.60	**0.055**
InstaNAS-C100-*E*	27.80	**0.046**

Table 3: InstaNAS can generalize to larger scale dataset and provide decent latency on TinyImageNet.

Model	Err. (%)	Latency
MobileNetV1	56.4	-
MobileNetV2 1.0 (Sandler et al. 2018)	48.1	0.264
MobileNetV2 1.4 (Sandler et al. 2018)	42.8	0.377
InstaNAS-Tiny-*A*	**41.4**	0.223
InstaNAS-Tiny-*B*	43.9	0.179
InstaNAS-Tiny-*C*	46.1	**0.171**

Comparison with Conditional Computing methods. In this section, we compare and show that InstaNAS outperforms several state-of-the-art conditional computing (*i.e.*, mixture of experts) methods. Figure 3 shown that InstaNAS significantly outperforms the two state-of-the-art conditional computing methods, BlockDrop (Wu et al. 2018) and ConvNetAIG (Veit and Belongie 2018), by a large margin. This is caused by the fundamental differences between NAS and conditional computing as mentioned in the related work.

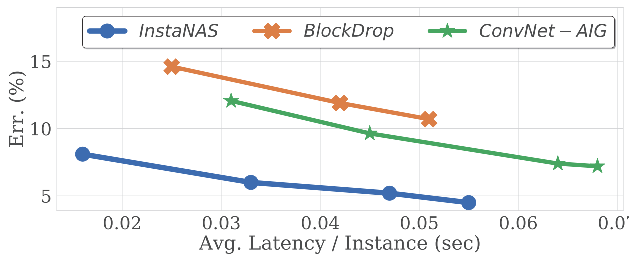

Figure 3: InstaNAS out-performs two state-of-the-arts conditional computing methods (Veit and Belongie 2018; Wu et al. 2018)) within MobileNetV2 search space.

Comparison with NAS methods. Figure 4 illustrates the best architectures found on the trade-off frontier for InstaNAS and two state-of-the-arts NAS methods: OneshotNAS (Bender et al. 2018), MNasNet (Tan et al. 2018). For OneshotNAS, we follow the one-shot search procedures (Bender et al. 2018) to sample 10,000 models from the meta-graph and train the trade-off frontier points from scratch on CIFAR-10. From Figure 4, we observe that the trade-off frontier achieved by InstaNAS is significantly better than OneshotNAS and MNasNet, which in turn confirms instance-awareness to be an effective characteristic for NAS.

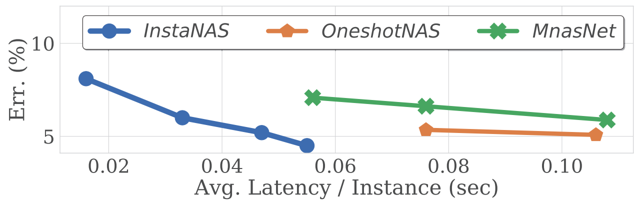

Figure 4: The InstaNAS trade-off frontier dominates both OneshotNAS and MNasNet on CIFAR-10 dataset.

Table 4: InstaNAS can find model with 14.5% latency reduction without compromising accuracy on ImageNet. If moderate accuracy drop (0.7%) is tolerable, InstaNAS-ImgNet-B can further achieve 26.5% average latency reduction comparing to MobileNetV2 ×1.0.

Model	Err. (%)	Latency
MNasNet ×0.5 (Tan et al. 2018)	32.4	0.121
ProxylessNAS (Cai, Zhu, and Han 2018)	31.8	0.136
MobileNetV2 ×1.0 (Sandler et al. 2018)	28.2	0.257
MobileNetV2 ×0.75 (Sandler et al. 2018)	30.2	0.200
InstaNAS-ImgNet-*A*	28.1	0.239
InstaNAS-ImgNet-*B*	28.9	**0.189**
InstaNAS-ImgNet-*C*	30.1	0.171

4.3 Qualitative Results

In this section, we provide a qualitative analysis on the child architectures selected by InstaNAS for ImageNet. Figure 5 illustrates the various images of three classes (brambling, matchstick, and dishwasher) sorted by its assigned architecture's latency (showed as the number below each image) normalized by the average latency—0% represents the average latency of all the architectures assigned to the images under a certain class. The images on the left are considered to be "simpler" (the architectures used have lower latency), and images on the right are "complex." Note that these levels are determined by the controller, which also matches humans' perception on the image complexity: *e.g.*, images with a cluttered background, high intra-class variation, illumination conditions are more complex and therefore sophisticated architectures (with higher latency) are assigned to classify these complex images.

Figure 6 illustrates architecture distribution (projected onto 2-D) with each dot representing an architecture and the color being the corresponding latency (red represents high latency and blue means low). We also randomly select three architectures and highlight the images samples assigned to them (by the controller) for making an inference. Notice that the image samples in each box share similar low-level visual patterns (*e.g.*, background color, object position/orientation,

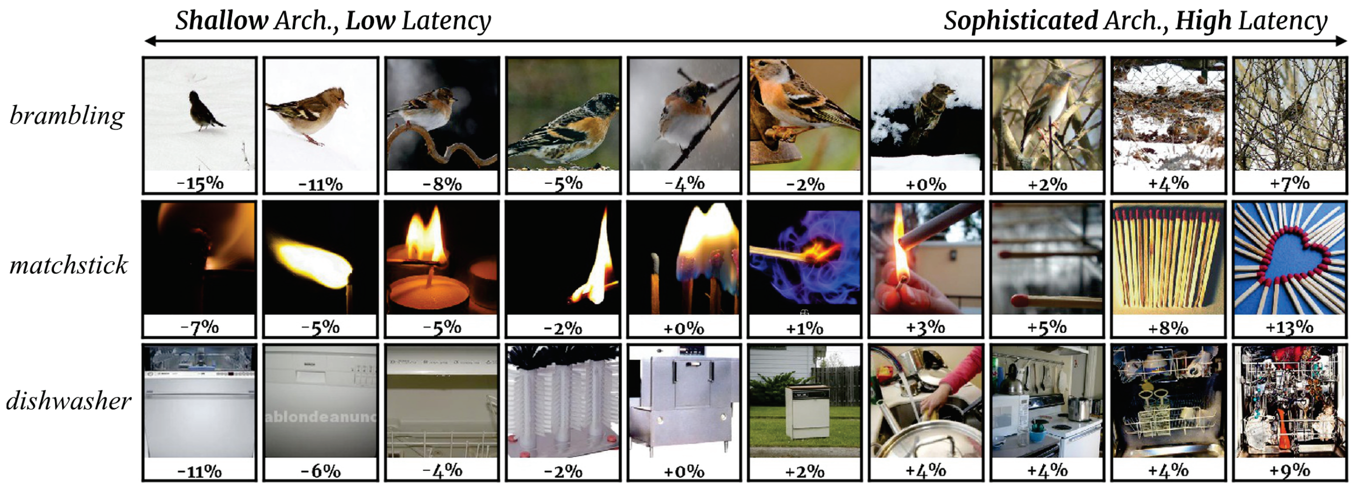

Figure 5: InstaNAS selects architectures tailored for each image. Each row represents samples from ImageNet with the same label; the images on the left are considered to be "simpler" and images on the right are "complex." These levels are determined by the controller, which also matches humans' perception: *e.g.*, cluttered background, high intra-class variation, illumination conditions. The number below each image represents the relative difference on latency. 0% means the average latency of all architectures selected for the images within certain class.

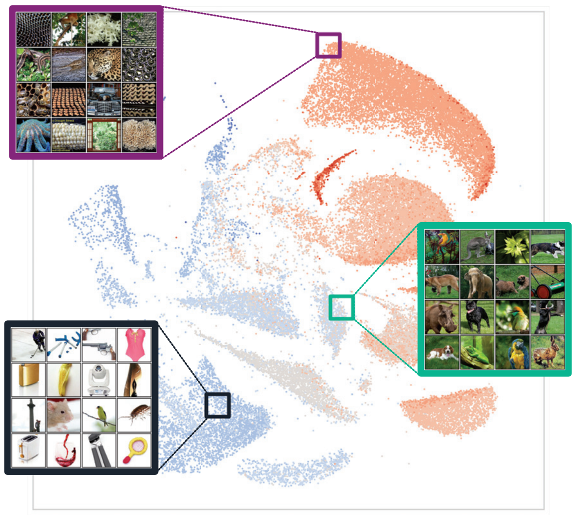

Figure 6: Distribution of InstaNAS architectures on ImageNet. Each point corresponds to an architecture probability vector p. We adopt UMAP to project high-dimensional p into 2D space, and color-code each architecture by its inference latency: *red* for high latency and *blue* for low. We also visualize three set of instances (in rectangle boxes) and instances in each box share the same architecture. The controller categorizes input instances base on their low-level visual characteristic, such as the background color (*green*), object position/orientation (*black*) and texture complexity (*purple*). Then the controller assigns each instance to a down-stream expert architecture for accurate classification.

and texture complexity) that agree with humans' perception. Both qualitative analyses confirm InstaNAS 's design intuition that the controller learns to discriminate each instance based on its low-level characteristic for best assigning that instance to the corresponding expert architecture.

5 Conclusion

In this paper, we propose InstaNAS, the first instance-aware neural architecture search framework. InstaNAS exploits instance-level variation to search for a *distribution of architectures*; during the inference phase, for each new image InstaNAS selects one corresponding architecture that best classifies the image while using less computational resource (*e.g.*, latency). The experimental results on CIFAR-10/100, TinyImageNet, and ImageNet all confirm that better accuracy/latency trade-off is achieved comparing to MobileNetV2, which we designed our search space against. Qualitative results further show that the proposed instance-aware controller learns to capture the low-level characteristic of the input image, which agrees with human perception. InstaNAS presents that instance-awareness is an important but missing piece in multiple-objective NAS and its potential in advancing the research in computational efficient models.

6 Acknowledgments

We are grateful to the National Center for Highperformance Computing for computer time and facilities, and Google Research, MediaTek, MOST 107-2634-F007-007 for their support. This research is also supported in part by the Ministry of Science and Technology of Taiwan (MOST 107-2633-E-002-001), MOST Joint Research Center for AI Technology and All Vista Healthcare, Novatek, and National Taiwan University.

References

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

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, 41–48. ACM.

Bengio, E.; Bacon, P.-L.; Pineau, J.; and Precup, D. 2015. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*.

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

DeVries, T., and Taylor, G. W. 2017. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*.

Dong, J.-D.; Cheng, A.-C.; Juan, D.-C.; Wei, W.; and Sun, M. 2018. Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XI*, 540–555.

Elsken, T.; Metzen, J. H.; and Hutter, F. 2018. Multi-objective architecture search for cnns. *arXiv preprint arXiv:1804.09081*.

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

Kim, Y.-H.; Reddy, B.; Yun, S.; and Seo, C. 2017. Nemo: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy. In *ICML 2017 AutoML Workshop*.

Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images. Technical report, Citeseer.

Liu, L., and Deng, J. 2017. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*.

Liu, H.; Simonyan, K.; and Yang, Y. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Ma, N.; Zhang, X.; Zheng, H.-T.; and Sun, J. 2018. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIV*, 122–138.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937.

Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient neural architecture search via parameter sharing. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, 4092–4101.

Real, E.; Moore, S.; Selle, A.; Saxena, S.; Suematsu, Y. L.; Tan, J.; Le, Q.; and Kurakin, A. 2017. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 2902–2911.

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

Rennie, S. J.; Marcheret, E.; Mroueh, Y.; Ross, J.; and Goel, V. 2017. Self-critical sequence training for image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 7008–7024.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4510–4520.

Shazeer, N.; Mirhoseini, A.; Maziarz, K.; Davis, A.; Le, Q.; Hinton, G.; and Dean, J. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Sun, K.; Li, M.; Liu, D.; and Wang, J. 2018. Igcv3: Interleaved low-rank group convolutions for efficient deep neural networks. In *British Machine Vision Conference 2018, BMVC 2018, Northumbria University, Newcastle, UK, September 3-6, 2018*, 101.

Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; and Le, Q. V. 2018. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*.

Teja Mullapudi, R.; Mark, W. R.; Shazeer, N.; and Fatahalian, K. 2018. Hydranets: Specialized dynamic architectures for efficient inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8080–8089.

Veit, A., and Belongie, S. 2018. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 3–18.

Wu, Z.; Nagarajan, T.; Kumar, A.; Rennie, S.; Davis, L. S.; Grauman, K.; and Feris, R. 2018. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8817–8826.

Zhou, Y.; Ebrahimi, S.; Arık, S. Ö.; Yu, H.; Liu, H.; and Diamos, G. 2018. Resource-efficient neural architect. *arXiv preprint arXiv:1806.07912*.

Zoph, B., and Le, Q. V. 2017. Neural architecture search with reinforcement learning. In *ICLR*.

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning transferable architectures for scalable image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.