Pareto-aware Neural Architecture Generation for Diverse Computational Budgets

Yong Guo, Yaofo Chen, Yin Zheng, Qi Chen, Peilin Zhao, Junzhou Huang, Jian Chen, Mingkui Tan

South China University of Technology

May 23, 2023

Outline

Outline

Background

- Designing effective architectures often relies heavily on human expertise and great effort.
- Neural architecture search (NAS) methods have been proposed to automatically design effective architectures.
- In practice, we have to carefully design architectures to fulfill a specific computational budget (*e.g.*, a feasible model should have a latency lower than 100ms on a specified mobile device).
- Furthermore, we may have to consider different computational budgets in the real world (*e.g.*, a company may simultaneously develop/maintain multiple applications).

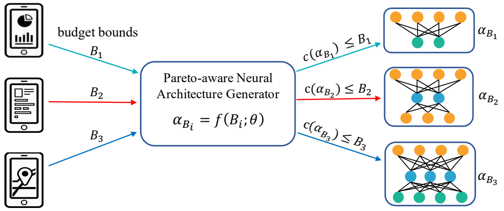
Challenge

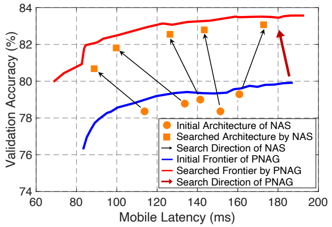
- Most NAS methods only consider a single budget and incorporate architecture's computational cost into the objective function.
- When we consider diverse budgets, these NAS methods have to conduct an independent search process for each budget, which is very inefficient yet unnecessary.

How to design effective architectures under diverse budgets in an efficient and flexible way still remains an open question.

Movivation

- We seek to devise an architecture generator that only needs to be trained once and then dynamically produces Pareto optimal architectures for diverse budgets via *inference* (as shown in the left figure).
- We seek to jointly learn the whole Pareto frontier (*i.e.*, improving the blue curve to the red curve in the right figure) instead of finding a single Pareto optimal architecture.

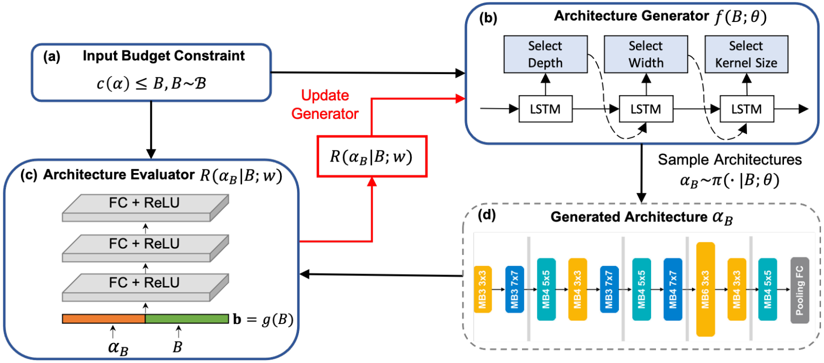

budget bounds
B_1
B_2
B_3
Pareto-aware Neural Architecture Generator
$\alpha_{B_i} = f(B_i;\theta)$
$c(\alpha_{B_1}) \leq B_1$
$c(\alpha_{B_2}) \leq B_2$
$c(\alpha_{B_3}) \leq B_3$
α_{B_1}
α_{B_2}
α_{B_3}



Outline

Pareto-aware Architecture Generation

Our Pareto-aware Neural Architecture Generator (PNAG) consists of two modules: **an architecture generator** $f(\cdot;\theta)$ that produces promising architectures and **an architecture evaluator** $R(\cdot|\cdot;w)$ that estimates the performance of the generated architectures.



Learning the Architecture Generator $f(B;\theta)$

NAS under a single budget. One can first learn a policy $\pi(\cdot;\theta)$ and then conduct sampling from it to find promising architectures, *i.e.* $\alpha \sim \pi(\cdot;\theta)$. Given a budget B, the optimization problem becomes

$$\max_{\theta} \ \mathbb{E}_{\alpha\sim\pi(\cdot;\theta)} \left[R\left(\alpha|B;w\right)\right], \ \text{s.t. } c(\alpha) \leq B. \tag{1}$$

- B is a budget (latency or MAdds) which can be considered as a random variable drawn from some distribution $\mathcal{B}$, namely $B{\sim}\mathcal{B}$.
- $\pi(\cdot;\theta)$ is the learned policy parameterized by θ.
- $R(\alpha|B;w)$ is the reward function that measures the joint performance of both the accuracy and latency of the architecture α.

Learning the Architecture Generator $f(B;\theta)$

NAS under diverse budgets. We learn an approximated Pareto frontier by finding a set of uniformly distributed Pareto optimal points. We evenly sample K budgets from the range of latency and maximize the expected reward over them.

$$\begin{aligned} \max_{\theta}\ &\mathbb{E}_{B\sim\mathcal{B}}\left[\mathbb{E}_{\alpha_B\sim\pi(\cdot|B;\theta)}\ \left[R\left(\alpha_B|B;w\right)\right]\right], \\ \text{s.t.}\ \ & c(\alpha_B)\leq B,\ \ B\sim\mathcal{B}, \end{aligned} \tag{2}$$

- $\mathbb{E}_{B\sim\mathcal{B}}\left[\cdot\right]$ denotes the expectation over the distribution of budget.
- $\pi(\cdot|B;\theta)$ is the learned policy conditioned on the budget of B.

Learning the Architecture Evaluator $R(\cdot|B;w)$

Given any architecture β and a budget B, we predict the performance $R(\beta|B;w)$ under the budget B via a promising evaluator. We train the evaluator with the pairwise ranking loss using M architectures with accuracy and latency

$$L(w) = \frac{1}{KM(M-1)} \sum_{k=1}^{K} \sum_{i=1}^{M} \sum_{j=1, j\neq i}^{M} \phi\Big(d(\beta_i, \beta_j, B_k) \cdot \big[R(\beta_i|B_k;w) - R(\beta_j|B_k; \qquad (4)$$

- $d\big(\beta_1, \beta_2, B_k\big)$ denotes a function to indicate whether β_i is better than β_j under the budget B_k.
- $\phi(z) = \max(0, 1-z)$ is a hinge loss function and we use it to enforce the predicted ranking results $R(\beta_i|B_k;w) - R(\beta_j|B_k;w)$ to be consistent with the results of $d(\beta_i, \beta_j, B_k)$ obtained by a comparison rule based on Pareto dominance.

Pareto Dominance Rule

To compare the performance between two architectures, we need to define a reasonable function $d(\beta_1, \beta_2, B)$.

Given any two architectures β_1, β_2, we use the following Pareto dominance function $d(\beta_1, \beta_2, B)$

$$d(\beta_1, \beta_2, B) = \begin{cases} 1, & \text{if } \quad (c(\beta_1) \leq B \wedge c(\beta_2) \leq B) \\ & \qquad \wedge\ (\mathrm{Acc}(\beta_1) \geq \mathrm{Acc}(\beta_2)); \\ -1, & \text{else if } \quad (c(\beta_1) \leq B \wedge c(\beta_2) \leq B) \\ & \qquad \wedge\ (\mathrm{Acc}(\beta_1) < \mathrm{Acc}(\beta_2)); \\ 1, & \text{else if } \quad c(\beta_1) \leq \ c(\beta_2); \\ -1, & \text{otherwise.} \end{cases} \tag{5}$$

Outline

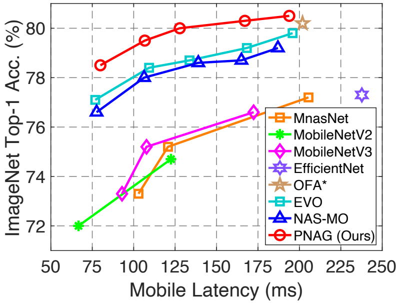
Architecture Search for Mobile Devices

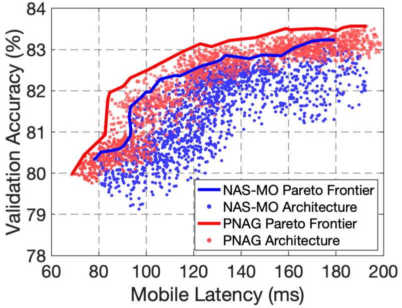
PNAG outperforms both manually designed and automatically searched architectures on mobile devices.

Architecture	Latency (ms)	Test Accuracy (%)		#Params (M)	#MAdds (M)	Search Cost (GPU Days)
		Top-1	Top-5			
MobileNetV3-Large (0.75×)	93.0	73.3	-	4.0	155	-
EVO-80	76.8	77.1	93.3	6.1	350	0.7
NAS-MO-80	77.6	76.6	93.2	7.9	340	0.7
PNAG-80	79.9	**78.3**	**94.0**	7.3	349	0.7
MobileNetV3-Large (1.0×)	107.7	75.2	-	5.4	219	-
EVO-110	109.3	78.4	94.0	10.2	482	0.7
NAS-MO-110	106.3	78.0	93.8	8.4	478	0.7
PNAG-110	106.8	**79.4**	**94.5**	9.9	451	0.7
FBNet-C	135.2	74.9	-	5.5	375	9.0
EVO-140	133.7	78.7	94.1	9.1	488	0.7
NAS-MO-140	139.0	78.6	94.0	9.5	486	0.7
PNAG-140	127.8	**79.8**	**94.7**	9.2	492	0.7
P-DARTS	168.7	75.6	92.6	4.9	577	3.8
EVO-170	168.3	79.2	94.4	10.7	661	0.7
NAS-MO-170	165.0	78.7	94.4	8.5	584	0.7
PNAG-170	167.1	**80.3**	**95.0**	10.0	606	0.7
Cream-L	-	80.0	94.7	9.7	604	12
OFA*	201.9	80.2	95.1	9.1	743	51.7
EVO-200	195.9	79.8	94.5	11.0	783	0.7
NAS-MO-200	187.4	79.2	94.4	9.1	630	0.7
PNAG-200	193.9	**80.5**	**95.2**	10.4	724	0.7

Visulizations of Generated Architectures

- PNAG (red line) consistently generates better architectures than the considered variants under diverse budgets. (left figure)
- PNAG finds a better frontier than NAS-MO due to shared knowledge across the search process under different budgets. (right figure)





Outline

Contributions

- Instead of designing architectures for a single budget, we propose a Pareto-aware Neural Architecture Generator (PNAG) which is only trained once and flexibly generates effective architectures for arbitrary budget via inference.
- To train PNAG, we explicitly learn the Pareto frontier by maximizing the expected reward of the searched architectures over diverse budgets, which shares the learned knowledge across the search processes under diverse budgets.
- To train the evaluator that is able to evaluate architectures for any given budget, we propose to learn a Pareto dominance rule which determines whether an architecture is better than the other in pairwise comparisons.