

MONAS: Multi-Objective Neural Architecture Search using Reinforcement Learning

Chi-Hung Hsu¹

Shu-Huan Chang¹

Da-Cheng Juan²

Jia-Yu Pan²

Yu-Ting Chen²

Wei Wei²

Shih-Chieh Chang¹

CHARLES1994608@GMAIL.COM

TOMMY610240@GMAIL.COM

X@DACHENG.INFO

JYPAN@GOOGLE.COM

YUTINGCHEN@GOOGLE.COM

WEIWEI@CS.CMU.EDU

SCCHANG@CS.NTHU.EDU.TW

¹National Tsing Hua University, ²Google Inc.

Abstract

Most existing works on neural architecture search aim at finding architectures that optimize for prediction accuracy. These methods may generate complex architectures consuming excessively high energy consumption, which is not suitable for computing environment with limited power budgets. We propose MONAS, a Multi-Objective Neural Architecture Search with novel reward functions that consider both prediction accuracy and power consumption when exploring neural architectures. MONAS effectively explores the design space and searches for architectures satisfying the given requirements. The experimental results demonstrate that the architectures found by MONAS achieve accuracy comparable to or better than the state-of-the-art models, while having better energy efficiency.

1. Introduction

Designing an effective convolutional neural networks (CNNs) heavily relies on experience and expertises, and therefore is treated more like an art than science. Neural architecture search (NAS) has been proposed to address this issue. Previous works have shown that CNNs found by NAS outperform state-of-the-art CNN models in terms of accuracy on image classification tasks (Baker et al., 2017; Zoph and Le, 2017a). However, these models are optimized only for classification accuracy, ignoring other important factors such as model complexity or energy consumption. As a result, these “highly accurate models” may not be suitable for certain (e.g., battery-driven) computing environment, such as cell phones.

In this paper, we propose MONAS, a multi-objective neural architectural search methodology. MONAS brings in the following contributions:

- **[Multi-Objective]** MONAS considers both prediction accuracy and energy efficiency (or power consumption) when searching through neural architecture space.
- **[Adaptability]** MONAS allows users to impose customized constraints when searching for neural architectures, for example, a constraint of the maximum power budget for mobile applications.

- **[Generality]** MONAS is general and can be applied for the architecture search of various kinds of CNNs. We demonstrate that MONAS can be effectively applied to both AlexNet (Krizhevsky et al., 2012) and CondenseNet (Huang et al., 2017).
- **[Effectiveness]** MONAS discovers novel architectures that outperform the current state-of-the-art designed by domain experts in terms of classification accuracy.

Related Works There are several reinforcement-learning-based NAS frameworks (Zoph and Le, 2017a,b; Cai et al., 2017; Baker et al., 2017). These works consider only accuracy and ignore other important objectives, such as energy consumption. Pareto-NASH (Thomas Elsken and Hutter, 2018) and Nemo (Ye-Hoon Kim and Seo, 2017) used evolutionary algorithms to search under multiple objectives. (Steven Tartakovsky and McCourt, 2017) used Bayesian optimization to optimize the trade-off between model accuracy and model inference time.

2. Proposed Framework: MONAS

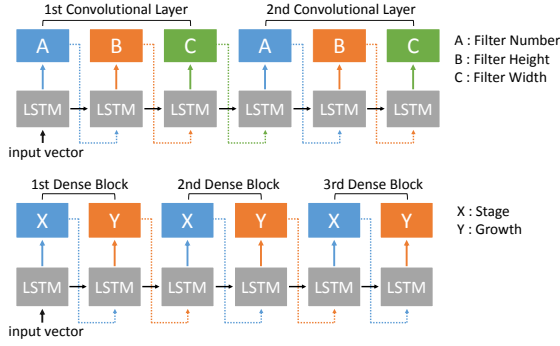
Framework Overview Our MONAS framework is built on top of a two-stage reinforcement-learning framework similar to NAS (Zoph and Le, 2017a). In the generation stage, we use a RNN as a *robot network* (RN), which generates a hyperparameter sequence for a CNN. In the evaluation stage, we train a *target network* (TN) according to the hyperparameters output by the RNN. The accuracy and energy consumption of the target network are the **rewards** to the robot network. The robot network updates itself based on this reward with the *policy gradient*. Algorithm 1 gives the pseudo code of the overall procedure.

Algorithm 1: MONAS Search Algorithm for Neural Architectures.

Input: $SearchSpace$, Reward Function(R), $n_{iterations}$, n_{epochs}
Output: Current Best Target Network(TN^*)
Initialize RN and $Reward_{max}$;
for $i \leftarrow 1$ **to** $n_{iterations}$ **do**
 $TN_i \leftarrow RN.generateTN(SearchSpace)$;
 Train TN_i for n_{epochs} ;
 $Reward_i \leftarrow R(TN_i)$;
 Update RN with $Reward_i$ with the policy gradient method;
 if $Reward_i > Reward_{max}$ **then**
 $Reward_{max} \leftarrow Reward_i$;
 $TN^* \leftarrow TN_i$;
 end
end
return TN^* ;

2.1. Implementation Details

Robot Network Fig. 1 illustrates the workflow of our RNN robot network with one-layer Long Short-Term Memory (LSTM). We predict one hyperparameter in Table 1 at each time step. The input vector to the LSTM has a length that is the number of all hyperparameter candidates, and it is initialized to zeros. The output of the LSTM is fed into a softmax layer, and a prediction is made by sampling the probabilities over the values of the hyperparameter



CNN model	Hyperparam	Search Space
AlexNet	# of Filters	8, 16, 32, 48, 64
	Filter Height	3, 5, 7, 9
	Filter Width	3, 5, 7, 9
CondenseNet	Block Stage	6, 8, 10, 12, 14
	Block Growth	4, 8, 16, 24, 32

Table 1: Hyperparameters and the search space considered in the experiments.

Figure 1: RNN workflow for AlexNet (up) and CondenseNet (down).

of that time step. The prediction value is encoded as a one-hot vector and is fed as the input of the next time step.

Target Networks & Search Space We demonstrate the generality of MONAS on two CNN models: (a) a simplified version of the *AlexNet* (Krizhevsky et al., 2012) as described in the TensorFlow tutorial (TensorFlow, 2018), and (b) the CondenseNet (Huang et al., 2017). Table 1 shows the search space of these two models.

2.2. Reinforcement Learning Process

Policy Gradient We take the RNN model as an actor with parameter θ , and the generated hyperparameters can be regarded as a series of actions in the policy-based reinforcement learning (Peters and Schaal, 2006). θ is updated by the policy gradient method to maximize the expected reward \bar{R}_θ :

$$\theta_{i+1} = \theta_i + \eta \nabla \bar{R}_{\theta_i} \quad (1)$$

where η denotes the learning rate and $\nabla \bar{R}_{\theta_i}$ is the gradient of the expected reward with parameter θ_i . We approximate the $\nabla \bar{R}_\theta$ as the following (Zoph and Le, 2017a):

$$\nabla \bar{R}_\theta = \sum_{\tau} \nabla P(\tau|\theta) R(\tau) \approx \frac{1}{N} \sum_{n=1}^N \sum_{t=1}^T \nabla \log P(a_t^n | a_{t-1:1}^n, \theta) R(\tau^n) \quad (2)$$

where $\tau = \{a_1, a_2, \dots, a_T\}$ is the output model from one MONAS iteration, and $R(\tau)$ is the reward of τ . $P(\tau|\theta)$ is the conditional probability of outputting a τ under θ . In our case, only the last action will generate a reward value $R(\tau)$, which corresponds to the classification accuracy and power consumption of the target network defined by the actions $a_{1:T}$. N is the number of target networks in a mini-batch. By sampling N times for multiple τ s, we can estimate the expected reward \bar{R}_θ of the current θ . In our experiments, we found that by using $N = 1$, which means that θ is updated every time for each target network we sampled, the convergence rate of the learning process can be improved.

Reward Function In this work, we consider three rewards signals in the MONAS framework: the validation accuracy, the peak power, and average energy cost during the inference of the target CNN model. The average energy cost is the product of the average power and execution time. We combine these reward signals into three types of reward functions:

- **Mixed Reward:** To find target network models having high accuracy and low energy cost, we directly trade off these two factors.

$$R = \alpha * Accuracy - (1 - \alpha) * Energy, \quad \alpha \in [0, 1] \quad (3)$$

- **Power Constraint:** For the purpose of searching configurations that fit specific power budgets, the peak power during testing phase is a hard constraint here.

$$R = Accuracy, \quad \text{if power} < \text{power threshold} \quad (4)$$

- **Accuracy Constraint:** By taking accuracy as a hard constraint, we can also force our RNN model to find high accuracy configurations.

$$R = 1 - Energy, \quad \text{if accuracy} > \text{accuracy threshold} \quad (5)$$

In the above equations, accuracy and energy cost are all normalized to $[0, 1]$. For Eq (4) and Eq (5), if the constraint is not satisfied, the reward will be set as zero.

3. Experimental Results and Discussions

Experiment Setup All experiments are implemented with TensorFlow, running on Intel XEON E5-2620v4 processor equipped with NVIDIA GTX1080Ti GPU cards. The NVIDIA profiling tool, **nvprof**, is used to measure the peak power, average power, and the runtime of CUDA kernel functions, used by the target network. We use the CIFAR-10 dataset to evaluate the target network, where 5000 images are randomly selected as the validation set.

Results and Discussions We conduct experiments to study the following questions:

- Does MONAS adapt to different reward functions?
- How efficient does MONAS guide the exploration in the solution space?
- How does the Pareto Frontier change under different reward functions?
- Does MONAS discover noteworthy architectures compared to the state-of-the-art?

Adaptability. MONAS allows incorporating application-specific constraints on the search objectives. In Fig. 2(b) and Fig. 2(c), we show that MONAS successfully guided the search process with constraints that either limit the peak power at 70 watts or require a minimum accuracy of 0.85. In fact, during the 600 iterations of exploration in the search space, it can be clearly seen that MONAS directs its attention to the region that satisfies the given constraint. In Fig. 2(b), 340 target networks out of 600 are in the region where peak power is lower than 70 watts. In Fig. 2(c), 498 target networks out of 600 are in the region where classification accuracy is at least 0.85. In comparison, the exploration process of the Random Search (Fig. 2(a)) explores the search space uniformly with no particular emphasis.

In these figures, we also depict the *Pareto Frontier* of the multi-objective search space in the red curves that trace the lower-right boundary of the points. We note that the models

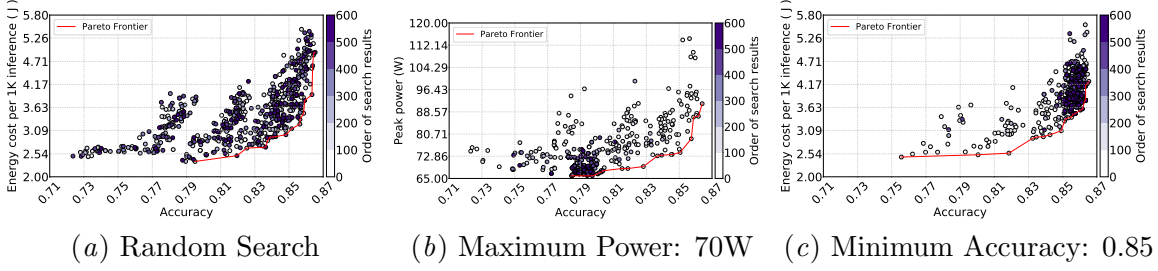


Figure 2: Random search vs. MONAS, with constraints on power or accuracy. Each point is a generated model. Deeper color indicates models generated in later iterations.

that lie on the Pareto Frontier exhibit the trade-offs between the different objectives and are all considered optimal.

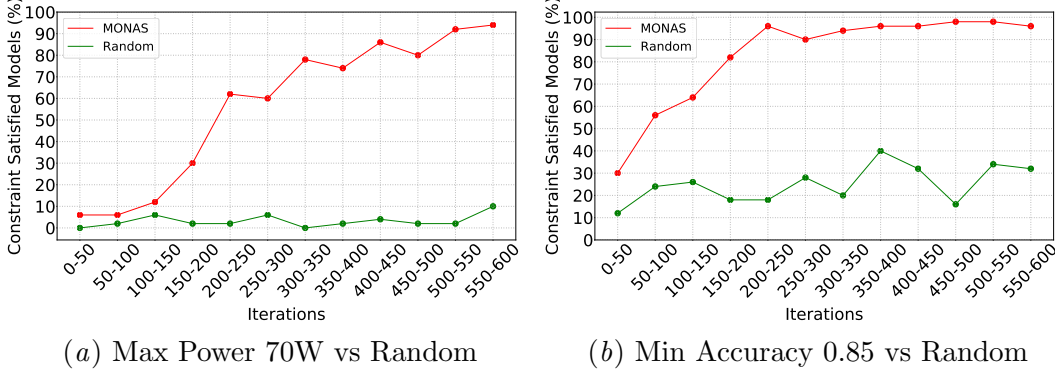
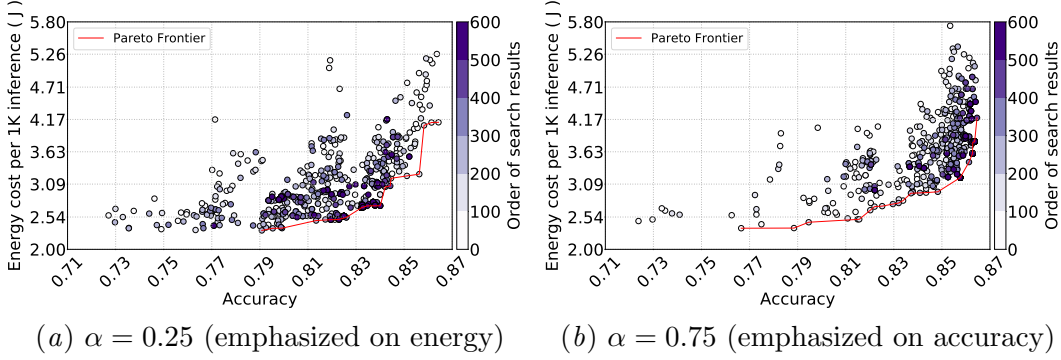


Figure 3: MONAS efficiently guides the search toward models satisfying constraints.

Efficiency. As shown above, starting with zero knowledge about the search space, MONAS can direct the search to the region that satisfies the given constraint. However, how fast can MONAS lock onto the desirable region? In Fig. 3(a), we show the percentage of architectures that satisfy the constraint of peak power 70 Watt at every 50 iterations. After 200 iterations, more than 60% of architectures generated by MONAS satisfy the constraint. Compared to the random search which generates less than 10% of architectures that satisfy the constraint. Fig. 3(b) shows similar efficiency of MONAS when given a constraint on classification accuracy. MONAS is more efficient and has a higher probability to find better architectures than random search.

Pareto Frontier. In Fig. 4(a) and Fig. 4(b), we show the search results of 600 iterations when applying different α values to the reward function in Eq (3). With different α values, MONAS demonstrates different search tendency, which is illustrated through the Pareto Frontier from the models explored. In Fig 5(a), we plot three Pareto Frontiers of $\alpha = 0.25$, $\alpha = 0.75$, and the random search. When α is set to 0.25, MONAS tends to explore the architectures with lower energy consumption. Similarly, MONAS tends to find architectures


 Figure 4: Applying different α when searching AlexNet.

with higher accuracy when α is set to 0.75. Compared to random search, MONAS can find architectures with higher accuracy or lower energy by using α to guide the search tendency.

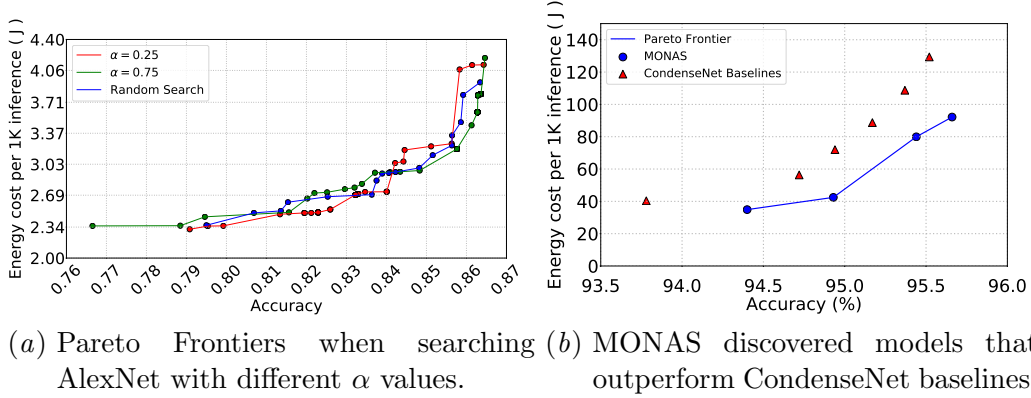


Figure 5: Pareto Frontiers on AlexNet and CondenseNet.

Discover Better Models. In Fig. 5(b), we compare the best CondenseNet models found by MONAS and the best ones selected from (Huang et al., 2017). Each point in the figure corresponds to a model. The Pareto Frontier in the figure demonstrates that MONAS has the ability to discover novel architectures with higher accuracy and lower energy than the ones designed by human. Table 2 in Appendix B gives the complete hyperparameter settings and results of the best models selected by MONAS.

4. Conclusion

We propose MONAS, a multi-objective architecture search framework based on reinforcement learning. We show that MONAS can adapt to application-specific constraints and effectively guide the search process to the models of interest. MONAS is general, and we applied it to two different kinds of CNN architectures, AlexNet and CondenseNet. For CondenseNet, MONAS discovers models that outperform the ones reported in the original paper, achieving higher accuracy and lower power consumption.

References

- Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*, 2017. URL <https://arxiv.org/pdf/1611.02167>.
- Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Reinforcement learning for architecture search by network transformation. *arXiv preprint arXiv:1707.04873*, 2017.
- Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. *arXiv preprint arXiv:1711.09224*, 2017.
- Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. URL <https://arxiv.org/pdf/1412.6980>.
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- Shichen Liu. Condensenet: Light weighted cnn for mobile devices, 2017. URL <https://github.com/ShichenLiu/CondenseNet>.
- J. Peters and S. Schaal. Policy gradient methods for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2219–2225, 2006.
- Scott Clark Steven Tartakovsky and Michael McCourt. Deep learning hyperparameter optimization with competing objectives, 2017. URL <https://devblogs.nvidia.com/sigopt-deep-learning-hyperparameter-optimization/>.
- TensorFlow. Convolutional neural networks, 2018. URL https://www.tensorflow.org/tutorials/deep_cnn.
- Jan Hendrik Metzen Thomas Elsken and Frank Hutter. Multi-objective architecture search for cnns. *arXiv preprint arXiv:1804.09081*, 2018.
- Sojung Yun Ye-Hoon Kim, Bhargava Reddy and Chanwon Seo. Nemo: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy. In *ICML17 AutoML Workshop*, 2017.
- Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017a. URL <https://arxiv.org/abs/1611.01578>.
- Vasudevan Vijay Shlens Jonathon Zoph, Barret and Quoc V Le. Learning Transferable Architectures for Scalable Image Recognition. *arXiv preprint arXiv:1707.07012*, 2017b.

Appendix A. Training Details for Target Networks

AlexNet: We design our robot network as an one-layer LSTM with 24 hidden units and train with the ADAM optimizer (Kingma and Ba, 2015) with learning rate = 0.03. The target network is trained for 256 epochs after the robot network selects the hyperparameters for it. We refer to the TensorFlow tutorial (TensorFlow, 2018) for other settings.

CondenseNet: Our robot network is constructed with one-layer LSTM with 20 hidden units and trained with the ADAM optimizer with learning rate = 0.008. We train the target network for 30 epochs and set the other configurations the same as the CondenseNet (Huang et al., 2017) public released code (Liu, 2017). We train the best performing models found by MONAS for 300 epochs on the whole training set and report the test error.

Appendix B. Experimental Results - CondenseNet

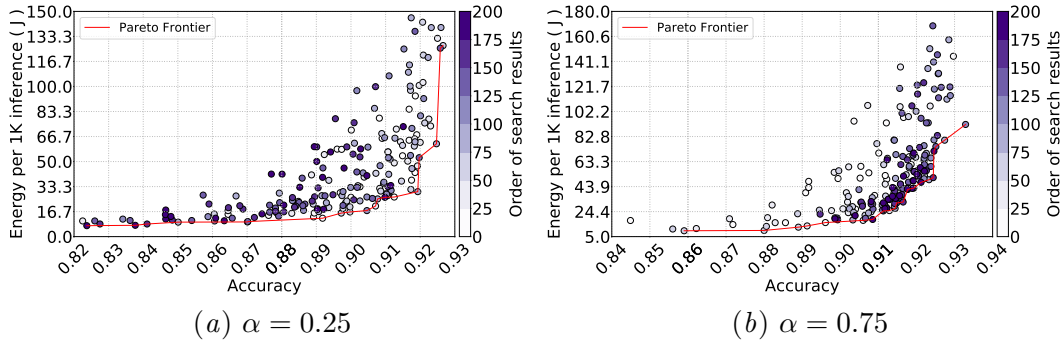


Figure 6: Applying different α when searching CondenseNet

Model	ST1	ST2	ST3	GR1	GR2	GR3	Error(%)	Energy(J)
CondenseNet 122	20	20	20	8	16	32	4.48	129.37
CondenseNet 110	18	18	18	8	16	32	4.63	108.74
CondenseNet 98	16	16	16	8	16	32	4.83	88.73
CondenseNet 86	14	14	14	8	16	32	5.06	71.98
CondenseNet 74	12	12	12	8	16	32	5.28	56.35
CondenseNet 50	8	8	8	8	16	32	6.22	40.35
CondenseNet-MONAS	6	14	14	32	32	32	4.34	92.16
	8	8	12	32	32	32	4.56	79.93
	6	12	14	8	32	32	5.07	42.46
	14	14	12	4	16	32	5.6	34.88

ST: Stage, the number of convolution layers in each block.

GR: Growth, the increasing number of filters of each layer.

Energy: Energy cost every 1000 inferences

Table 2: Models found by MONAS with $\alpha = 0.75$ and CondenseNet Baselines