MONAS: Multi-Objective Neural Architecture Search

Chi-Hung Hsu,[1] Shu-Huan Chang,[1] Jhao-Hong Liang,[1] Hsin-Ping Chou,[1]

Chun-Hao Liu,[1] Shih-Chieh Chang,[1] Jia-Yu Pan,[2] Yu-Ting Chen,[2]

Wei Wei,[2] Da-Cheng Juan[2]

[1]National Tsing-Hua University, Hsinchu, Taiwan [2]Google, Mountain View, CA, USA
{charles1994608, tommy610240, jayveeliang, alan.durant.chou, newgod1992}@gmail.com,
scchang@cs.nthu.edu.tw, {jypan, yutingchen, wewei, dacheng}@google.com

Abstract

Recent studies on neural architecture search have shown that automatically designed neural networks perform as good as expert-crafted architectures. While most existing works aim at finding architectures that optimize the prediction accuracy, these architectures may have complexity and is therefore not suitable being deployed on certain computing environment (e.g., with limited power budgets). We propose MONAS, a framework for Multi-Objective Neural Architectural Search that employs reward functions considering both prediction accuracy and other important objectives (e.g., power consumption) when searching for neural network architectures. Experimental results showed that, compared to the state-of-the-arts, models found by MONAS achieve comparable or better classification accuracy on computer vision applications, while satisfying the additional objectives such as peak power.

Introduction

Convolutional Neural Networks (CNN) have shown impressive successes in computer-vision applications; however, designing effective neural networks heavily relies on experience and expertise, and can be very time-consuming and labor-intensive. To address this issue, the concept of neural architecture search (NAS) has been proposed (Zoph and Le 2017a), and several literatures have shown that architectures discovered from NAS outperforms state-of-the-art CNN models on prediction accuracy (Baker et al. 2017; Zoph and Le 2017a). However, models solely optimized for prediction accuracy generally have high complexity and therefore may not be suitable to be deployed on computing environment with limited resources (e.g., battery-powered cellphones with low amount of DRAM).

In this paper, we propose MONAS, a framework for Multi-Objective Neural Architectural Search that has the following properties:

- **[Multi-Objective]** MONAS considers both prediction accuracy and additional objectives (e.g., energy consump-

tion) when exploring and searching for neural architectures.

- **[Adaptability]** MONAS allows users to incorporate customized constraints introduced by applications or platforms; these constraints are converted into objectives and used in MONAS to guide the search process.

- **[Generality]** MONAS is a general framework that can be used to search a wide-spectrum of neural architectures. In this paper, we demonstrate this property by applying MONAS to search over several families of CNN models from AlexNet-like (Krizhevsky, Sutskever, and Hinton 2012), CondenseNet-like (Huang et al. 2017a), to ResNet-like (He et al. 2015) models.

- **[Effectiveness]** Models found by MONAS achieves higher accuracy and lower energy consumption, which outperforms the state-of-the-art CondenseNet (Huang et al. 2017a) in both aspects. Experimental results also confirm that MONAS effective guides the search process to find models satisfying the predefined constraints.

- **[Scalability]** To make MONAS more scalable, we further extended MONAS into MONAS-S to accelerate the search process by adopting the weight-sharing searching technique[1]. Compared to MONAS, MONAS-S is able to search for a design space up to 10^{22} larger, and at the same time, MONAS-S is 3X faster.

The remainder of this paper is organized as follows. Section 2 reviews the previous work of NAS. Section 3 details the proposed MONAS and MONAS-S. Section 4 describes the experimental setup. Section 5 provides the experimental results and Section 6 concludes this paper.

Background

Recently, automating neural network design and hyperparameter optimization has been proposed as a reinforcement learning problem. Neural Architecture Search (Zoph and Le 2017a; 2017b) used a recurrent neural network (RNN)

---

[1]The weight-sharing searching technique is first proposed by Pham et al. (Pham et al. 2018)

to generate the model descriptions of neural networks and trained this RNN with reinforcement learning to maximize the validation accuracy of the generated models. Most of these works focus on discovering models optimized for high classification accuracy without penalizing excessive resource consumption.

Recent work has explored the problem of architecture search with multiple objectives. 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.

The major difference between the previous works and ours is that we consider the model computation cost, such as power consumption or Multiply-ACcumulate (MAC) operations, as another constraint for architecture search. In this paper, our optimization objectives include both CNN model accuracy and its computation cost. We explore the model space by taking the peak power or accuracy to be a prerequisite, or we can directly set our objectives weights when considering the trade-off between model accuracy and computation cost.

Proposed Framework: MONAS

In this section, we describe our MONAS framework for neural architecture search with *multiple objectives*, which is built on top of the framework of reinforcement learning.

Framework Overview

Our method MONAS adopts a two-stage 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 an existing CNN model as a *target network* (*TN*) with 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 reinforcement learning. Algorithm 1 gives the pseudo code of the overall procedure of our MONAS framework.

Implementation Details

Robot Network Fig. 1 illustrates the workflow of our robot network, an RNN model with one-layer Long Short-Term Memory (LSTM) structure. At each time step, we predict the hyperparameters in Table 2 one at a time. The input sequence of the LSTM is a vector whose length is the number of all hyperparameter candidates (for example, 4 possible values for the number of filters, filter height and filter width will give a vector of length 4*3 = 12), and it is initialized to zeros. The output of the LSTM is fed into a softmax layer, and a prediction is then completed by sampling through the probabilities representing for each hyperparameter selection. For the input sequence of the next time step, we make a one-hot encoding at the position of the current selection we take.

Algorithm 1: MONAS Search Algorithm for Neural Architectures.

Input: $SearchSpace$, Reward Function(R), $n_{iterations}$, n_{epochs}
Output: Current Best Target Network(TN^*)

1   $Reward_{max} \leftarrow 0$;
2   Initialize the robot network RN;
3   **for** $i \leftarrow 1$ **to** $n_{iterations}$ **do**
4      $TN_i \leftarrow RN.generateTN(SearchSpace)$;
5   Train TN_i for n_{epochs};
6      $Reward_i \leftarrow R(TN_i)$;
7   Update RN with $Reward_i$ with the policy gradient method;
8   **if** $Reward_i > Reward_{max}$ **then**
9         $Reward_{max} \leftarrow Reward_i$;
10        $TN^* \leftarrow TN_i$;
11   **return** TN^*;

Target Networks & Search Space We demonstrate the generality of MONAS by applying to two families of CNN models. The first one, which we called *AlexNet*, is the simplified version of the AlexNet (Krizhevsky, Sutskever, and Hinton 2012) as described in the TensorFlow tutorial[2]. For the two convolutional layers in our target AlexNet, our robot network selects a value in [8, 16, 32, 48, 64] for the number of filters; [3, 5, 7, 9] for filer height and filter width. The other one is the CondenseNet (Huang et al. 2017a), an efficient version of DenseNet (Huang et al. 2017b). We predict the **stage** and **growth** for the 3 dense blocks in the CondenseNet. Our robot network selects a stage value in [6, 8, 10, 12, 14] and a growth value in [4, 8, 16, 24, 32].

Reinforcement Learning Process

In this subsection, we describe the details of our implementation of the policy gradient method to optimize the robot network. Then we describe the design of the reward functions.

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

$$\theta_{i+1} = \theta_i + \eta \nabla \bar{R}_{\theta_i} \tag{1}$$

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 $\nabla \bar{R}_\theta$ with the method of (Zoph and Le 2017a):

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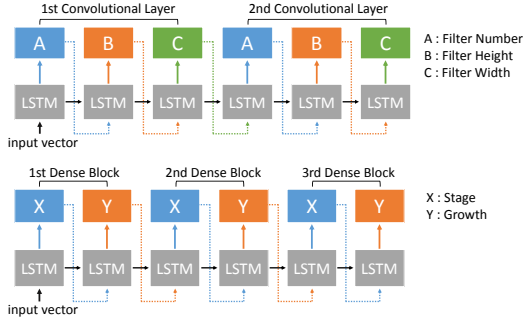
---

[2]www.tensorflow.org/tutorials/deep_cnn

Figure 1: RNN workflow for AlexNetand CondenseNet

Model	Hyperparameters
AlexNet	Number of Filters
	Filter Height
	Filter Width
CondenseNet	Block Stage
	Block Growth

Table 1: Hyperparameters used in experiments.

where $\tau = \{a_1, a_2..., a_T, r_T\}$ is the output from one MONAS iteration. $R(\tau)$ 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_T, which corresponds to the classification accuracy and power consumption of the target network defined by a sequence of actions $a_{1:T}$. N is the number of target networks in a mini-batch. By sampling N times for multiple τs, we can estimate the expected reward $\bar{R}_\theta$ of the current θ. In our experiments, we found that using $N = 1$, which means the θ is updated every time for each target network we sampled, improves the convergence rate of the learning process.

MONAS for Scalability To improve the scalability and the training time for MONAS, we adopt the techniques (such as weight sharing) in ENAS (Pham et al. 2018) with our MONAS framework and propose the MONAS-S method. Similar to ENAS, the training process of MONAS-S is a two-step process: First, the shared weights of a directed acyclic graph (DAG) that encompasses all possible networks are pre-trained. Then, the controller uses the REINFORCE policy learning method (Williams and J. 1992) to update the controller's weights – models are sampled from the DAG, and their validation accuracy as evaluated on a mini-batch data is used as the reward of each model. Since MONAS-S considers multiple objectives besides validation accuracy, other objective measures such as energy consumption are also computed for each of the sampled model, and together with the validation accuracy forms the reward to the controller.

Reward Function The reward signal for MONAS consists of multiple performance indexes: the validation accuracy, the peak power, and average energy cost during the in-

ference of the target CNN model. To demonstrate different optimization goals, our reward functions are as follows:

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

$$R = \alpha * Accuracy - (1 - \alpha) * Energy \qquad (3)$$

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

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

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

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

- **MAC Operations Constraint:** The amount of MAC operations is a proxy measure of the power used by a neural network. We use MAC in our experiments of MONAS-S. By setting a hard constraint on MAC, the controller is able to sample model with less MAC.

$$R = Accuracy, \text{if MAC} < \text{threshold} \qquad (6)$$

In the equations (3)(4)(5), the accuracy and energy cost are both normalized to [0, 1]. For Eq (4) and Eq (5), if current target network doesn't satisfy the constraint, the reward will be given zero. For Eq (6), a negative reward is given to the controller when the target network has a MAC that does not satisfy the given constraint.

Experimental Setup
Hardware: All the experiments are conducted via Python (version 3.4) with TensorFlow library (version 1.3), running on Intel XEON E5-2620v4 processor equipped with GeForce GTX1080Ti GPU cards.

GPU Profiler: To measure the peak power and energy consumption of the target network for the reward to update our robot network, we use the NVIDIA profiling tool - **nvprof**, to obtain necessary GPU informations including peak power, average power, and the runtime of CUDA kernel functions.

Dataset: In this paper, we use the CIFAR-10 dataset to validate the target network. We randomly select 5000 images from training set as the validation set and take the validation accuracy to be an objective of the reward function.

MAC Operations: We calculate the MAC operations of a sampled target network based on the approach described in MnasNet (Tan et al. 2018). We calculate MAC operations as follows:

- **Convolutional layer:**

$$K * K * C_{in} * H * W * C_{out} \qquad (7)$$

- **Depthwise-separable convolutional layer:**

$$C_{in} * H * W * (K * K + C_{out}) \qquad (8)$$

In Eq (7) (8), K is the kernel size of the filter, H/W is the height/width of the output feature map and C_{in}, C_{out} is the number of the input and output channels. We ignore the MAC operations of fully connected layer and skip connections since every target network has the same fully connected layer as the output layer and there is very few MAC operations in skip connections.

Results and Discussions

In this section, we show some experimental results to illustrate the output of our proposed method. We are in particular interested in 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 Front change under different reward functions?
- Does MONAS discover noteworthy architectures compared to the state-of-the-art?
- Can MONAS guide the search process while the search space is large?

Adaptability. MONAS allows incorporating application-specific constraints on the search objectives. In Fig. 2(b) and Fig. 2(c), we show that our method 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 search space, it can be clearly seen that MONAS directs its attention on 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. For comparison, we show the exploration process of the Random Search in Fig. 2(a), which explored the search space uniformly with no particular emphasis.

In these figures, we also depict the *Pareto Front* of the multi-objective search space (red curves that trace the lower-right boundary of the points). We note that the models that lie on the Pareto Front exhibit the trade-offs between the different objectives and are considered equally good.

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 70Watt 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. Similarly, when given a constraint on

(a) Random Search

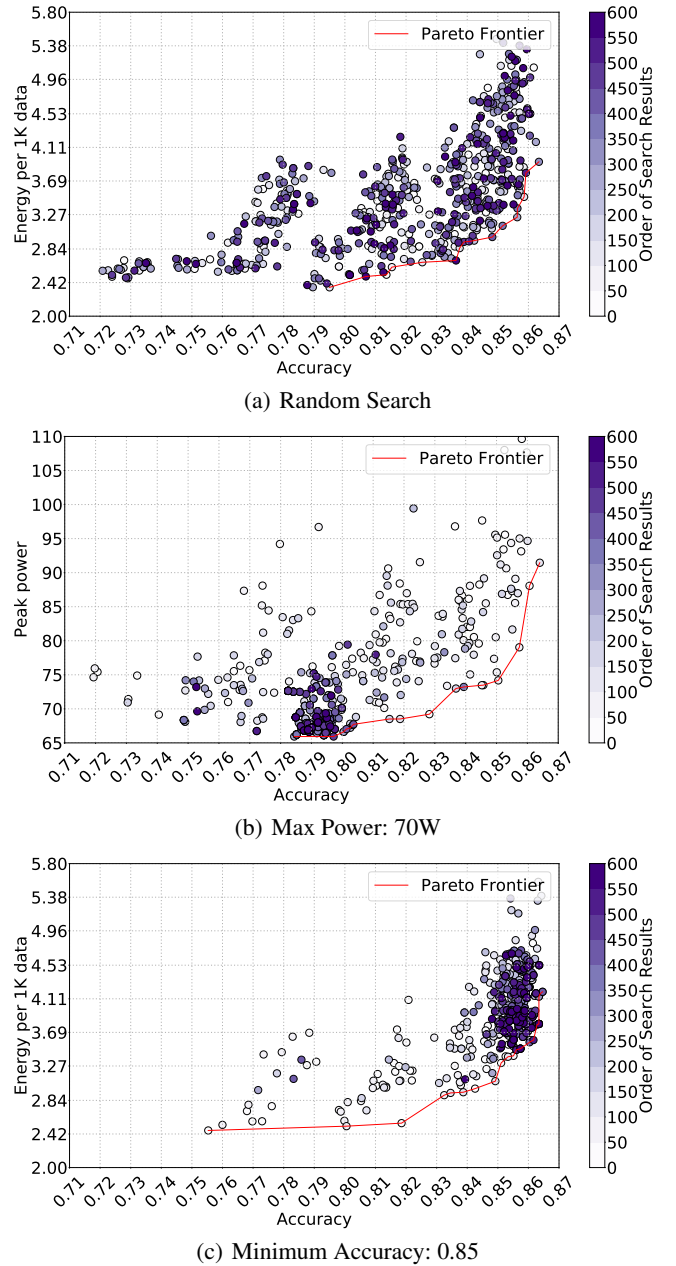(b) Max Power: 70W

(c) Minimum Accuracy: 0.85

Figure 2: AlexNet Random Search versus Power or Accuracy Constraint

classification accuracy, MONAS can also guide the search efficiently, as demonstrated in Fig. 3(b). When the overall search time is limited, MONAS is more efficient and has a higher probability to find better architectures and thus outperforms random search.

Pareto Frontier. In Fig. 4(a) and Fig. 4(b), we show the search results of 600 architectures when applying different α to the reward function in Eq (3). MONAS demonstrates different search tendency while applying different α. To better
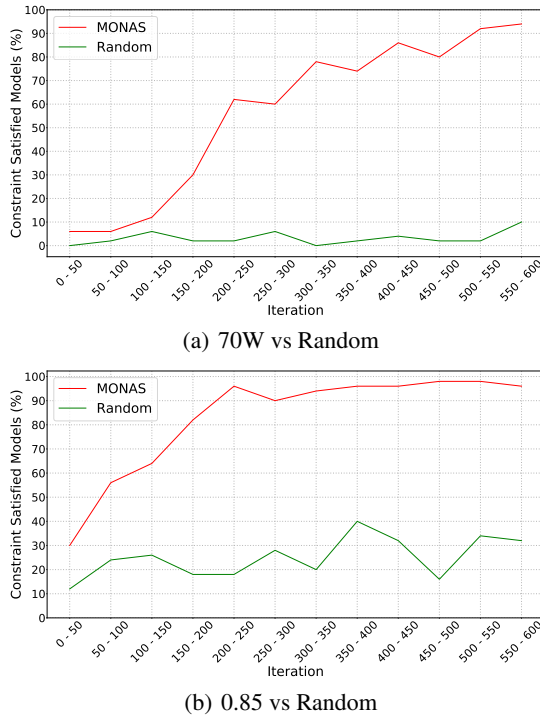
(a) 70W vs Random

(b) 0.85 vs Random

Figure 3: MONAS efficiently guides the search toward models satisfying constraints, while the random search has no particular focus.

(a) $\alpha = 0.25$

(b) $\alpha = 0.75$

Figure 4: Applying different α when searching AlexNet

understand the search tendency, we plot three Pareto Frontiers of $\alpha = 0.25$, $\alpha = 0.75$, and random search, together in Fig 5(a). When α is set to 0.25, MONAS tends to explore the architectures with lower energy consumption. Similarly, MONAS tends to find architectures 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.

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

Scalability

To further justify that the multi-objective reward function works well under a larger search space, we apply our scalable MONAS-S method to find target networks in a 12-layered architecture, similar to the search approach in ENAS (Pham et al. 2018). Among the 12 layers, the first 4 layers have the largest size, and the size is reduced in half at the next 4 layers, and then further reduced in half at the last 4 layers. The filters/operations being considered include
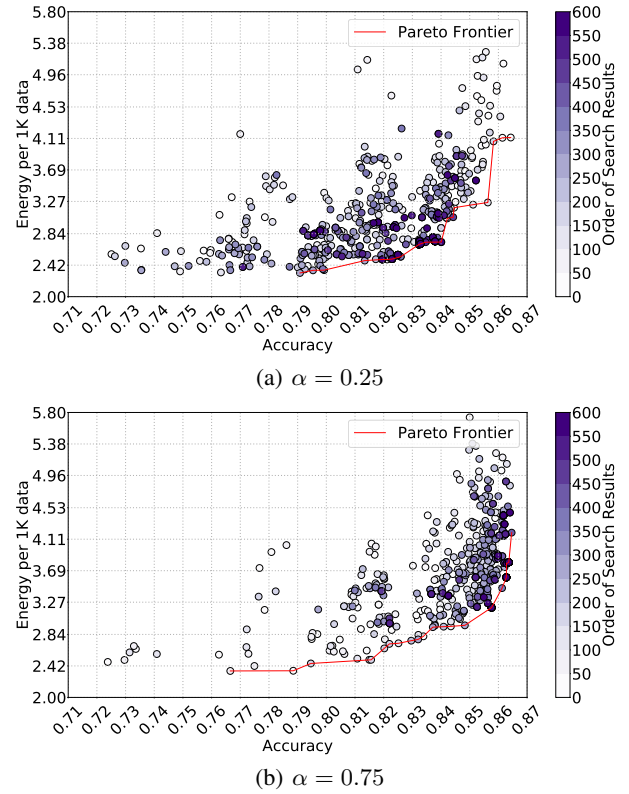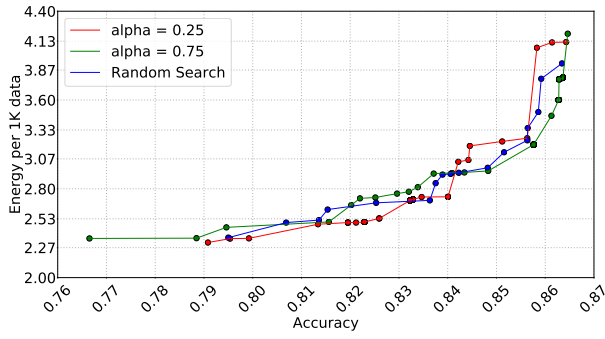
the convolution 3x3, convolution 5x5, separable convolution 3x3, separable convolution 5x5, average pooling and max pooling. This creates a search space of 1.6×10^{29} possible networks.

In Fig. 7(b), we show that MONAS-S successfully guided the search process, even the search space being explored is such larger. As a comparison, we also do a single-objective search, which is actually ENAS (Pham et al. 2018), using only the validation accuracy (Fig. 7(a)). In this experiment, both MONAS-S and ENAS train its controller for 600 epochs. To compare the two trained controllers, we sample 1000 models from each controller, and compare their validation accuracy and the MAC operations required. It is clear that the target networks found by MONAS-S are biased towards ones that require less MAC and mostly satisfy the given constraint (Fig. 7(b)). We note that, unlike previous experiments, the accuracy here is the validation accuracy of a mini-batch.

We are interested in knowing which components are preferred by the controller trained by MONAS-S when building (sampling) a model, since such knowledge may give us insights in the design of energy-efficient, high-accuracy models. In Fig. 8, we show the distribution of the operations selected by the two controllers by ENAS (Fig. 7(a)) and MONAS-S (Fig. 7(b)).

Compared to ENAS's controller, MONAS-S's controller tends to select operations with lower MAC operations, while managing to maintain a high-level accuracy at the same

(a) Pareto Fronts of searching AlexNet with different α.
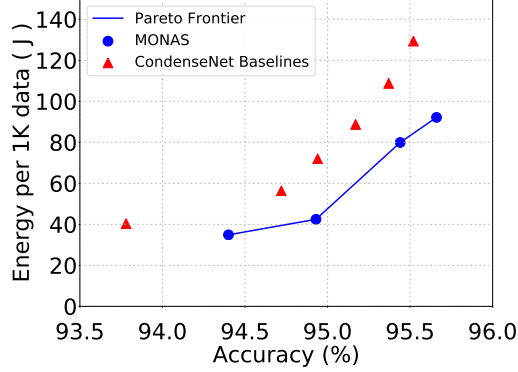
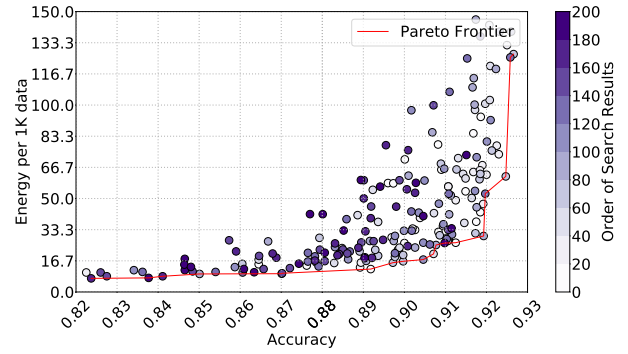(b) MONAS discovered models that outperform CondenseNet baselines

Figure 5: MONAS Pareto Fronts on AlexNet and CondenseNet

(a) $\alpha = 0.25$

(b) $\alpha = 0.75$

Figure 6: Applying different α when searching CondenseNet

time. Operations like depth-wise separable convolution and average pooling are chosen over other operations. This could explain why the target networks found by MONAS-S have lower MAC operations.
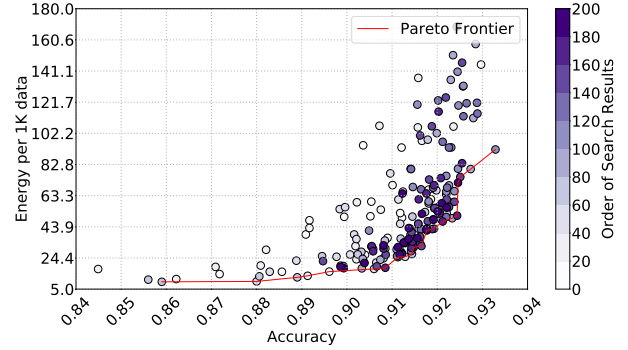
Furthermore, as an attempt to understand how these MAC-efficient components are used at each layer, we show the operation distribution at each layer (Fig. 9). Interestingly, the main differences between target networks sampled by ENAS and those sampled by MONAS-S happened in the first four layers, while the preference of operators are very similar in the later layers. In the first four layers, MONAS-S prefers MAC-efficient `sep_conv_5x5` and `avg_pool` operations, while ENAS prefers `conv_5x5` and `sep_conv_3x3`. We notice that the first four layers are also the ones that have the largest size of feature maps (the size of feature maps reduces by half after every 4 layers) and are tended to need more computation. It is a pleasant surprise that MONAS-S is able to learn a controller that focuses on reducing MAC at the first 4 layers which have larger feature map sizes and potentially computation-hungry.

Training Details for Target Networks

AlexNet: We design our robot network with one-layer LSTM with 24 hidden units and train with ADAM optimizer (Kingma and Ba 2015) with learning rate = 0.03. The target network is constructed and trained for 256 epochs after

the robot network selects the hyperparameters for it. Other model settings refer to the TensorFlow tutorial.[3]

CondenseNet: Our robot network is constructed with one-layer LSTM with 20 hidden units and trained with ADAM optimizer with learning rate = 0.008. We train the target network for 30 epochs and other configurations refer to the CondenseNet (Huang et al. 2017a) public released code.[4] We train the best performing models found by MONAS for 300 epochs on the whole training set and report the test error.

MONAS-S: We build MONAS-S controller by extending ENAS with our MONAS framework. We train the sharing weight and controller for 600 epochs. For the other configurations we refer to the ENAS public release code.[5]

Conclusion

In this paper, we propose MONAS, a multi-objective architecture search framework based on deep reinforcement learning. We show that MONAS can adapt to application-specific constraints and effectively guide the search process to the region of interest. In particular, when applied on CondenseNet, MONAS discovered models that outperform the

---

[3] www.tensorflow.org/tutorials/deep_cnn
[4] github.com/ShichenLiu/CondenseNet
[5] github.com/melodyguan/enas

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, GR: Growth, Energy: Energy cost every 1000 inferences

Table 2: **MONAS models and CondenseNet Baselines**

(a) Without MAC Operations Constraint

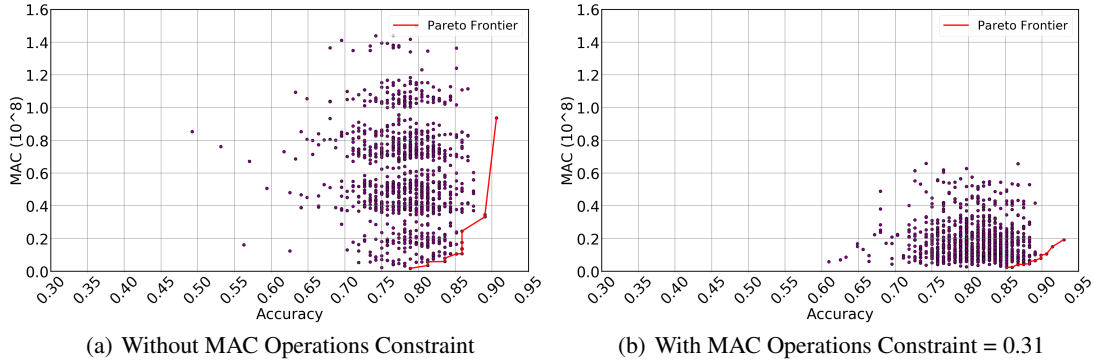(b) With MAC Operations Constraint = 0.31

Figure 7: The 1000 target networks in (a) are sampled by ENAS and those in (b) are sampled by MONAS-S. Both controllers were trained for 600 epochs.
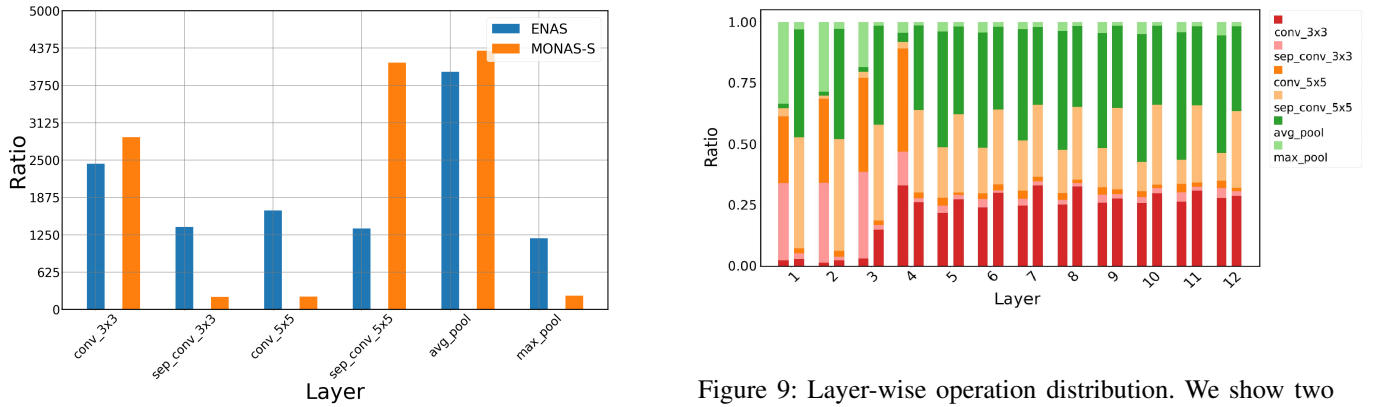

Figure 8: Model operation distribution. The blue bars are sampled by ENAS and the orange bars are sampled by MONAS-S. The Y-axis is the count of the operations being used in a layer of a sampled network. We sampled 1000 12-layered networks in this experiment, so the sum of either the blue or orange bars is 12000. Model operation distribution.

Figure 9: Layer-wise operation distribution. We show two bars at each layer: the left bar is the distribution of operations sampled by ENAS and the right one is that by MONAS-S. The main difference is at the first 4 layers, within which MONAS-S's controller is more likely to sample MAC-efficient operations such as sep_conv_5x5. We note that the first 4 layers are also the layers with the largest size of feature maps (size is reduced by half every 4 layers in our search space) and tended to need more computation. This shows that MONAS is, surprisingly, able to correctly identify the best opportunity to reduce MAC operations.

ones reported in the original paper, with higher accuracy and lower power consumption. In order to work with larger search spaces and reduce search time, we extend the concepts of MONAS and propose the MONAS-S, which is scalable and fast. MONAS-S explores larger search space in a shorter amount of time.

References

Baker, B.; Gupta, O.; Naik, N.; and Raskar, R. 2017. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep residual learning for image recognition. *CoRR*.

Huang, G.; Liu, S.; van der Maaten, L.; and Weinberger, K. Q. 2017a. Condensenet: An efficient densenet using learned group convolutions. *arXiv preprint arXiv:1711.09224*.

Huang, G.; Liu, Z.; van der Maaten, L.; and Weinberger, K. Q. 2017b. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Kingma, D. P., and Ba, J. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Peters, J., and Schaal, S. 2006. Policy gradient methods for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2219–2225.

Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient neural architecture search via parameter sharing. In *ICML*.

Steven Tartakovsky, S. C., and McCourt, M. 2017. Deep learning hyperparameter optimization with competing objectives.

Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; and Le, Q. V. 2018. MnasNet: Platform-Aware Neural Architecture Search for Mobile. *ArXiv e-prints*.

Thomas Elsken, J. H. M., and Hutter, F. 2018. Multiobjective architecture search for cnns. *arXiv preprint arXiv:1804.09081*.

Williams, and J., R. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*.

Ye-Hoon Kim, Bhargava Reddy, S. Y., and Seo, C. 2017. Nemo: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy. In *ICML17 AutoML Workshop*.

Zoph, B., and Le, Q. V. 2017a. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*.

Zoph, Barret, V. V. S. J., and Le, Q. V. 2017b. Learning Transferable Architectures for Scalable Image Recognition. *arXiv preprint arXiv:1707.07012*.