MNET: FRUIT LESION CLASSIFICATION CONVNET DESIGN VIA EFFICIENT NEURAL ARCHITECTURE SEARCH

[1] *Xuan-Jun Chen* (陳炫均), [2] *Chiou-Shann Fuh* (傅楸善)

[1] Department of Electronics Engineering,
[2] Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan
E-mail: r08943115@ntu.edu.tw fuh@csie.ntu.edu.tw

ABSTRACT

Deep neural networks have achieved great success in many domains. However, designing accurate and efficient networks is challenging because the design space is combinatorially large, and different tasks need very different architectures to obtain satisfactory results. The group of methods called the Neural Architecture Search (NAS) helps to find effective architecture in an automated manner. In this paper, we propose an Efficient Neural Architecture Search (ENAS) framework that uses Reinforcement Learning (RL) methods combined with weight-sharing strategy to optimize ConvNet architectures, avoiding enumerating and training individual architectures separately as in previous methods. In addition to many other methods tested on benchmark datasets, we tested it on practical agriculture task of mango lesion classification, which differs greatly in terms of difficulty in distinguishing between classes, resolution of images, data balance within the classes, and the number of data available. We name the family of models discovered by our method MNets (**M**ango-**Nets**). MNets surpass state-of-the-art models both designed manually and generated automatically. MNet-120-160 achieves 77.06% accuracy on mango lesion classification task with only 3.1M number of parameters, 12.56% higher and 44.6x smaller than VGG16. Despite higher accuracy and smaller parameter size than AutoKeras-120-160, MNet-120-160's search cost is 20.6x smaller than AutoKeras-120-160's, at only 14.25 GPU-hours.

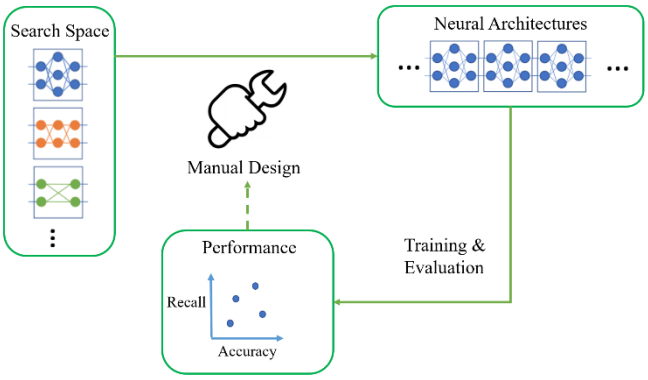
Keywords: *Image Classification, Defect Detection, Neural Architecture Search.*

1. INTRODUCTION

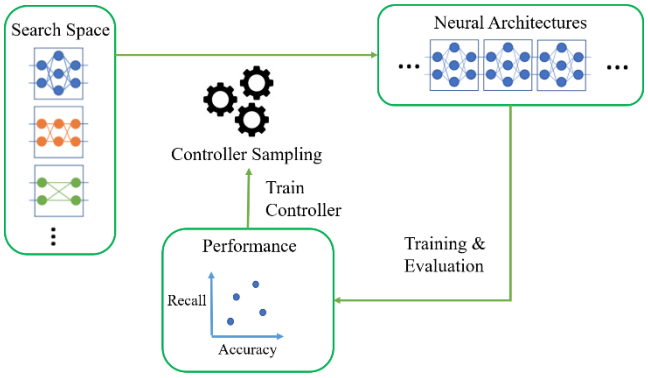
Recently, with the rise of artificial intelligence, the introduction of precise technology into the concept of smart value-added technology in agriculture will not only help increase productivity, but also improve labor shortages and income from agricultural production. It is expected that high automation and standardized support would improve output value and effectively carry out industrial transformation. In recent years, the sales volume of Ai-Wen mango has continued to increase, not only jumping into one of the three major fresh fruits for export, but also expanding the exporting countries to Japan, China, the United States, and Hong Kong. Although Taiwan mangoes have increased their popularity and market shares compared with the past, they are still encountering price-cut competition from other mangoes exporting countries, e.g., Philippines, Thailand. Therefore, many processes certainly still need to be improved.

ConvNets are the *de facto* method for computer vision. In many computer vision tasks, a better ConvNet design usually leads to significant accuracy improvement. In previous works, accuracy improvement comes at the cost of higher computational complexity, making it more challenging to deploy ConvNets to general devices, on which computing capacity is limited. Instead of solely focusing on accuracy, recent work also aims to optimize for efficiency. However, designing efficient and accurate ConvNets is difficult due to several challenges described as the following.

First is the intractable design space. The design space of a ConvNet is combinatorial. Using VGG16 [1] as a motivating example: VGG16 contains 16 layers. Assume for each layer of the network, we can choose a different kernel size from {1, 3, 5} and a different filter number from {32, 64, 128, 256, 512}. Even with such simplified design choices and shallow layers, the design space contains $(3 \times 5)^{16} \approx 6 \times 10^{18}$ possible architectures. However, training a ConvNet is very time-consuming, typically taking days or even weeks. As a result, previous ConvNet design rarely explores the design space. A typical flow of manual ConvNet design is illustrated in Figure 1(a). Designers propose initial architectures and train them



(a) A typical flow of manual ConvNet design.



(b) A typical flow of reinforcement learning based neural architecture search.

Figure 1. Illustration of manual ConvNet design and reinforcement learning based neural architecture search.

on the target dataset. Based on the performance, designers evolve the architectures accordingly. Limited by the time cost of training ConvNets, the design flow has to stop after a few iterations, which is far too few to sufficiently explore the design space. Starting from [2], recent works adopt Neural Architecture Search (NAS) to explore the design space automatically. Many previous works [2, 3, 4] use Reinforcement Learning (RL) to guide the search and a typical flow is illustrated in Figure 1(b). A controller samples architecture from the search space to be trained. The performance of the trained networks is then used to train and improve the controller. Previous works [2, 3, 4] has demonstrated the effectiveness of such methods in finding accurate and efficient ConvNet models.

Second is the non-transferable optimality. The optimality of ConvNet architectures is conditioned on many factors such as input resolutions, number of channels. Once these factors change, the optimal architecture is likely to be different. For example, a comm-

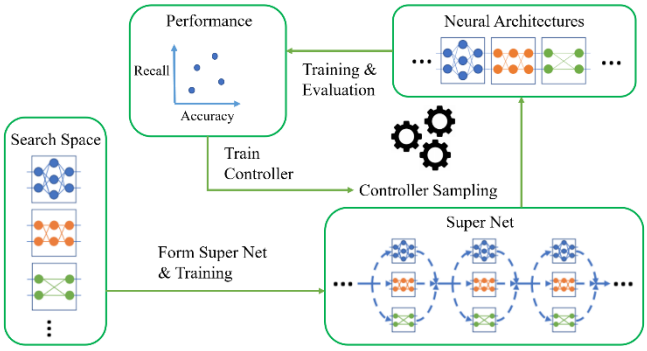


Figure 2. Proposed Efficient Neural Architecture Search (ENAS) for ConvNet design. ENAS explores a layer-wise space that each layer of a ConvNet can choose a different block. The search space is represented by a super net. The search process trains the super net using Stochastic Gradient Descent (SGD) to optimize the weights. Optimal architectures are sampled from the trained weights with Recurrent Neural Network (RNN) controller.

on practice to reduce the FLOPs count of a network is to shrink the input resolution. A smaller input resolution may require a smaller receptive field of the network and hence shallower layers. However, a smaller input resolution may lead to a lower accuracy of the network. Therefore, we should design different ConvNet architectures case-by-case to achieve the best accuracy-efficiency trade-off. In practice, however, limited by the computational cost of previous manual and automated approaches, we can only realistically design one ConvNet and use it for all conditions.

In recent years, Neural Architecture Search (NAS) has attracted attention as an automatic model design algorithm and has achieved many state-of-the-art results in many tasks. However, the main issue in NAS is to efficiently search for the optimal architecture in a large search space, which usually has at least 10^{10} architectures. To address the problems for designing ConvNet for specific agricultural mango lesion classification, we propose an Efficient Neural Architecture Search (ENAS) framework that uses Reinforcement Learning (RL) methods combined with weight-sharing strategy to discover efficient ConvNets. The flow of our algorithm is illustrated in Figure 2. ENAS allows us to explore a layer-wise search space where we can choose a different block for each layer of the network. For reinforcement learning, the framework constructs Recurrent Neural Network (RNN) controller to sample models from the search space. The training of the controller uses Reinforcement Learning (RL) which takes the performance of the sampled models as reward, and finally the controller learns to sample the best model.

Moreover, the weight-sharing strategy constructs a super net as one-shot model and sampling sub-models directly from it. Our method has three search spaces, including number of filters, dilation factors, and the connectivity between different layers.

We name the models discovered by our method as MNets (Mango-Nets). MNets surpass the state-of-the-art models both designed manually and automatically. MNet-120-160 achieves 77.06% accuracy on mango lesion classification task with only 3.1M number of parameters, 12.56% higher and 44.6x smaller than VGG16. Being better than AutoKeras-120-160, MNet-120-160's search cost is 14.25 GPU-hours, 20.6x lower than the cost for AutoKeras-120-160. Such low search cost enables us to re-design ConvNets case-by-case. In summary, our method can return the best parameter set and network structure with high performance. Taking advantage of our ENAS framework, we can search for good models and surpass state-of-the-art results in a short time.

2. REVIEW OF NAS AND RELATED WORK

Without loss of generality, the architecture search space A is represented by a directed acyclic graph (DAG). A network architecture is a subgraph $a \in A$, denoted as $N(a, w)$ with weights w.

Neural architecture search aims to solve two related problems. The first is *weight optimization* of a given network architecture as in standard deep learning,

$$w_a = \underset{w}{argmin}\, L_{train}\big(N(a, w)\big), \qquad (1)$$

where $L_{train}(\cdot)$ is the loss function on the training set.

The second is *architecture optimization*. In a general sense, it finds the architecture that is trained on the training set and has best accuracy on the validation set, as

$$a^* = \underset{a \in A}{argmax}\, ACC_{val}\big(N(a, w_a)\big), \qquad (2)$$

where $ACC_{val}(\cdot)$ is the accuracy on the validation set.

Real world tasks usually have additional requirements on a network's memory consumption, FLOPs, latency, energy consumption, etc. These requirements are up to the architecture a, software and hardware platforms, but irrelevant to the weights w_a. Thus, they are called architecture constraints. A typical constraint is that the network's latency is no more than a preset budget, as

$$Latency(a^*) \leq Lat_{max}\,. \qquad (3)$$

Note that it is challenging to satisfy Eq. (2) and Eq. (3) simultaneously for most previous approaches.

Early NAS approaches perform weight optimization and architecture optimization in a nested manner [2, 3, 5, 6]. Numerous architectures are sampled from A and trained from scratch as in Eq. (1). Efficient architecture search algorithms are critical to making Eq. (1) affordable. Previous works use reinforcement learning [2, 3, 4, 7, 8] and evolution [9, 10, 11, 12, 13]. The computation cost is still high. Only the small dataset and small search space (e.g, a single block) are affordable.

Recent NAS approaches adopt a common *weight sharing* strategy [7, 14, 15, 16, 17, 18, 19, 20]. The architecture search space A is encoded in a *super net*, denoted as $N(A, W)$, where W is the weights in the super net. The super net is trained once. All architectures inherit their weights directly from W. Thus, they share the weights in their common graph nodes. Fine tuning of an architecture is performed in need, but no training from scratch is incurred. Therefore, architecture search is fast and suitable for large datasets. After optimization, the best architecture a^* could be sampled from $A(\theta^*)$,

$$(\theta^*, W_{\theta^*}) = \underset{\theta, W}{argmin}\, L_{train}\big(N(A(\theta), W)\big). \qquad (4)$$

where θ denotes the parameters that represent the architectures in the space.

For efficient ConvNet models, designing efficient ConvNet has attracted many research attentions in recent years. SqueezeNet [21] is one of the early works focusing on reducing the parameter size of ConvNet models. It is originally designed for classification, but later extended to object detection [22] and LiDAR point-cloud segmentation [23, 24]. Following SqueezeNet, SqueezeNext [25] and ShiftNet [26] achieve further parameter size reduction.

Auto-Keras:

The Auto-Keras [27] package, developed by the DATA Lab team at Texas A&M University, is an open source framework which has yet a total of 7.1K stars on its GitHub's repository. Auto-Keras enables Bayesian Optimization (BO) to guide the network morphism, which keeps the functionality of a neural network while changing its neural architecture, for more efficient neural architecture search.

3. METHOD

In this paper, we use Efficient Neural Architecture Search (ENAS) to solve the problem of ConvNet design. We formulate the neural architecture search problem as

$$\min_{a \in A} \min_{w_a} L(a, w_a). \quad (5)$$

Given an architecture space A, we seek to find an optimal architecture $a \in A$ such that after training its weights w_a, it can achieve the minimal loss $L(a, w_a)$.

Central to the idea of ENAS is the observation that all the graphs which NAS ends up iterating over can be viewed as sub-graphs of a larger graph. In other words, we can represent NAS's search space using a single Directed Acyclic Graph (DAG). Intuitively, ENAS's DAG is the superposition of all possible child models in a search space of NAS, where the nodes represent the local computations and the edges represent the flow of information. The local computations at each node have their own parameters, which are used only when the particular computation is activated. Therefore, ENAS's design allows parameters to be shared among all child models, i.e. architectures, in the search space.

In the following, we will explain how to train ENAS and how to derive architectures from ENAS's controller (Section 3.1). We will then explain our search space and our practical process for designing convolutional architectures (Section 3.2).

3.1 Training ENAS and Deriving Architectures

Our controller network is an (Long Short-Term Memory) LSTM with 100 hidden units. This LSTM samples decisions via softmax classifiers, in an autoregressive fashion: the decision in the previous step is fed as input embedding into the next step. At the first step, the controller network receives an empty embedding as input. In ENAS, there are two sets of learnable parameters: the parameters of the controller LSTM, denoted by θ, and the shared parameters of the child models, denoted by ω. The training procedure of ENAS consists of two interleaving phases. The first phase trains ω, the shared parameters of the child models, on a whole pass through the training data set. For the mango lesion classification task, ω is trained on 5,600 training mango images, separated into minibatches of size 128, where $\nabla\omega$ is computed using standard back-propagation. The second phase trains θ, the parameters of the controller LSTM, for a fixed number of steps, set to 5000 in our experiments. These two phases are alternated during the training of ENAS. More details are described as follows.

Train the shared parameters ω of the child models:

In this step, we fix the controller's policy $\pi(\boldsymbol{m};\ \theta)$ and perform Stochastic Gradient Descent (SGD) on ω to minimize the expected loss function $E_{\boldsymbol{m}\sim\pi}[L(\boldsymbol{m};\ \omega)]$. Here, $L(\boldsymbol{m};\ \omega)$ is the standard cross-entropy loss, computed on a minibatch of training data, with a model $\boldsymbol{m}$ sampled from $\pi(\boldsymbol{m};\ \theta)$. The gradient is computed using the Monte Carlo estimate

$$\nabla_\omega E_{\boldsymbol{m}\sim\pi(\boldsymbol{m};\ \theta)}[L(\boldsymbol{m};\ \omega)] \approx \frac{1}{M}\sum_{i=1}^{M} \nabla_\omega L(\boldsymbol{m}_i, \omega), \quad (6)$$

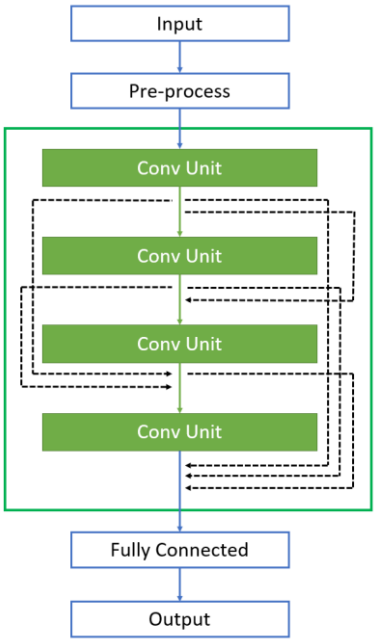
where $\boldsymbol{m}_i$'s are sampled from $\pi(\boldsymbol{m};\ \theta)$ as described above. Eq. (6) provides an unbiased estimate of the gradient $\nabla_\omega E_{\boldsymbol{m}\sim\pi(\boldsymbol{m};\ \theta)}[L(\boldsymbol{m};\ \omega)]$. However, this estimate has a higher variance than the standard SGD gradient, where $\boldsymbol{m}$ is fixed. Nevertheless, we can update ω using the gradient from any single model $\boldsymbol{m}$ sampled from $\pi(\boldsymbol{m};\ \theta)$, which $M = 1$. As mentioned, we train ω during an entire pass through the training data.

Train the controller parameters $\boldsymbol{\theta}$:

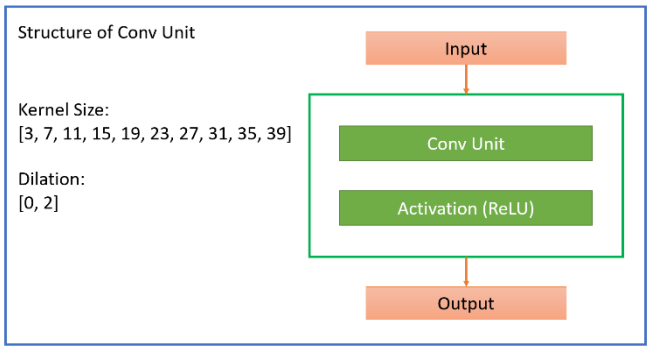
In this step, we fix ω and update the policy parameters θ, aiming to maximize the expected reward $E_{\boldsymbol{m}\sim\pi(\boldsymbol{m};\ \theta)}[R(\boldsymbol{m};\ \omega)]$. We employ the Adam optimizer, for which the gradient is computed using reinforcement learning with a moving average baseline to reduce variance. The reward $R(\boldsymbol{m};\ \omega)$ is computed on the validation set, rather than on the training set, to encourage ENAS to select models that generalize well rather than models that overfit the training set well. In our mango image classification experiments, the reward function is the accuracy on a minibatch of validation images.

Deriving Architectures:

We discuss how to derive novel architectures from a trained ENAS model. We first sample several models from the trained policy $\pi(\boldsymbol{m};\ \theta)$. For each sampled model, we compute its reward on a single minibatch sampled from the validation set. We then take only the model with the highest reward to re-train from scratch. It is possible to improve our experimental results by training all the sampled models from scratch and selecting the model with the highest performance on a separated validation set, as done by other works [2, 3]. However, our method yields surpassing performance while being much more economical.



(a) The schema of the MNet model search framework, taking 4-layer as an example.



(b) Convolution block parameter search space.

Figure 3. Illustration of the schema of the MNet model search framework, taking 4-layer as an example. And its parameter search space.

3.2 Search Space and Designing Process

The Search Space:

The model we search for is a 12-layer Convolutional Neural Network (CNN), every layer has 20 candidate operations. The filter size could be 3, 7, 11, 15, 19, 23, 27, 31, 35, or 39 and the number of dilations could be 1 or 2. Furthermore, the residual connections between layers could also be searched, that is, the output feature map in each layer can be aggregated with one of the previous outputs. As a result, the search space in our tasks is $10^{12} \times 13! \approx 6.2 \times 10^{21}$ architectures. The schema of the MNet model search framework, taking 4-layer as an example, is illustrated in Figure 3(a). And its parameter search space is shown in Figure 3(b).

The Practical Designing Process:

The designing process can be divided into three stages: 1. Training the super net; 2. Training the controller; 3. Deriving the best architecture.

1. Training the super net

First, we construct a super net which includes all the possible architectures and a sampler which performs uniform sampling to all the choices. A sub-model can be decided with a sequential sample using the sampler. In every iteration, we sample a sub-model from the super net and train this sub-model with a batch of training data. After training, the weights of the super net can be updated based on the sub-model. We repeat this process for 100 epochs.

2. Training the controller

In this stage, we aim to train a controller to help us retrieve the best model in the extremely large search space. Since we sample a sub-model layer-by-layer, the probability of all the choices in a layer is fully dependent on the previous layers. We construct the controller using LSTM cells. The main characteristic of LSTM is that it can deal with time-series data. In every time step, we decode the hidden state to get the probability of each choice and make a decision according to the sampling result. After several time steps, a sub-model can be decided and we evaluate its performance (e.g. accuracy, weighted average recall) with a batch of validation data. We then take the performance as rewards and compute the gradient of the controller by reinforcement learning.

3. Deriving the best architecture

After the training of the super net and the controller, we repeat the sample process but only keep the sub-model with the highest probability as the best architecture. We then train it from scratch to obtain the best performance in our task.

4. EXPERIMENTS

For the mango image classification, dataset was created at three fruit collection facilities in Fangshan and Pingtung, containing tens of thousands of images. The images were made in well-lighted spaces using smart phones and the video recordings with HDR camera. Each of the image

contained a shot of one mango that was put on data collectors' hand or the conveyor. Furthermore, each image was labeled with a mango grade by professionals, which includes three grades A, B, or C. The dataset was divided into train, development and test sets, where the number of mango images with classified grades in each set are 5600, 800, and 1600, respectively.

For all these data, the images were preprocessed before search starts. The steps are shown as following: 1. Resize image resolution as 24 × 32 or 120 × 160 pixels; 2. Rotate the image by 90 degrees if it's in portrait orientation; 3. Augment training dataset by horizontally flipping each image with a probability of 0.5; 4. Encode each grades, A, B, or C, into one-hot vectors.

Next, we train the super net with search space described in Section 3.2. Due to the memory constraint, we set the input resolution of the network to 24-by-32 first and increase to 120-by-160 pixels later. We train the super net for 100 epochs. In each epoch, the weights are trained on 80% of training set using Stochastic Gradient Decent (SGD) with momentum of 0.9. Then, we train the Recurrent Neural Network (RNN) controller on the rest 20% of training set as the validation set with Adam optimizer. After the search finishes, we sample the best architecture that has the highest accuracy on the validation set, and finally train the best architecture from scratch. Hyper-parameters' settings are summarized in Table 1, and our architecture search framework and searched models training are all implemented in PyTorch.

We compare our searched models with state-of-the-art efficient models both designed manually and automatically. The criterion for evaluating the model is weighted average recall (i.e., accuracy). For baseline manual models, we used a variety of well-known convolutional neural networks which have been used extensively in tasks of image classification, object detection, image pixel-wise segmentation and instance segmentation. We selected the following pretrained models as our baselines to compare with: ResNeXt (ResNeXt-50 32×4d), AlexNet, VGG16, DenseNet (Densenet-121), and ShuffleNet (ShuffleNetV2 with 1.0x output channels). For automatically designed models, we use the popular Auto-Keras, which has yet over a total of 7.1K stars on its GitHub's repository, as our baselines to compare with.

Running on a single Nvidia Tesla P100 GPU, we present the performance of MNets in various aspects as well as other baselines in Table 2. For those well-known manually designed convolutional neural networks, the one that achieves the highest accuracy is the AlexNet (i.e., 69.63%) with 62.4M number of parameters, which is 7.43% lower and 20.13x larger than MNet-120-160. For automatically designed models, despite higher accuracy and smaller parameter size than AutoKeras-120-160, MNet-120-160's search cost is 20.6x smaller than AutoKeras-120-160's, at only 14.25 GPU-hours.

Table 1. Hyper-parameters' setting

Hyper-parameters' Setting	
Batch size	128
Supernet learning rate	0.01
Controller learning rate	0.1
Learning rate	0.001
Momentum	0.9
Weight decay	0.0005
Controller optimizer	Adam
Optimizer	SGD
Supernet epochs	100
Controller steps	5,000
Epochs	100
Layers	12
Feature dimension	256

The accuracy results are compared independently with whole baselines including feature-level and decision-level concatenate fusion of different convolutional neural networks and are summarized in Table 3. For feature-level fusion approach, it concatenates the feature sets to obtain the fusion-level features as input to the Support Vector Machine (SVM). For decision-level fusion approach, it first performs classification using features extraction from those single nets and further apply one-versus-rest decision function to obtain the values of distance between sample and the separating hyperplane. Among the both fusion approaches, the decision-level fusion approach achieves the highest 71.63% recognition accuracy, which is 5.43% lower than the models searched by our framework.

5. CONCLUSION

We present ENAS, an efficient neural architecture search framework that uses Reinforcement Learning (RL) methods combined with weight-sharing strategy to optimize ConvNet architectures. And test it on practical

Table 2. Mango image classification performance in various aspects compared with baselines.

Input size	Model	Search method	Search cost (GPU hours / relative)	#Params (million)	Acc (%)
224 × 224	ResNeXt-50	manual	-	25.0	56.75
224 × 224	AlexNet	manual	-	62.4	69.63
224 × 224	VGG16	manual	-	138.4	64.5
224 × 224	DenseNet-121	manual	-	8.1	52.38
224 × 224	ShuffleNetV2 1.0x	manual	-	2.3	57.38
24 × 32	AutoKeras-24-32	Bayesian Optimization (BO)	10 / 12.2x	3.3	66.73
24 × 32	MNet-24-32 (ours)	Reinforcement Learning (RL)	0.82 / 1.0x	2.5	73.21
120 × 160	AutoKeras-120-160	Bayesian Optimization (BO)	294 / 20.6x	61.3	70.75
120 × 160	MNet-120-160 (ours)	Reinforcement Learning (RL)	14.25 / 1.0x	**3.1**	**77.06**

Table 3. Mango image classification accuracy (%) result compared independently with whole baselines.

Single Net				
ResNeXt-50 (R)	AlexNet (A)	VGG16 (V)	DenseNet-121 (D)	ShuffleNetV2 1.0x (S)
56.75	69.63	64.5	52.38	57.38
Feature-level Concatenate Fusion (only the combination with best accuracy is presented)				
R + A			70.88	
Decision-level Concatenate Fusion (only the combination with best accuracy is presented)				
R + A + V + D			71.63	
AutoKeras				
AutoKeras-24-32	66.73		AutoKeras-120-160	70.75
MNet (ours)				
MNet-24-32	73.21		MNet-120-160	**77.06**

agriculture task of mango lesion classification in addition to many other methods test on benchmark datasets, which differs greatly in terms of difficulty in distinguishing between classes, resolution of images, data balance within the classes, and the number of data available. MNets, a family of models discovered by ENAS surpass state-of-the-art models, both manually and automatically designed: MNet-120-160 achieves 77.06% accuracy on mango lesion classification task with only 3.1M number of parameters, 12.56% higher and 44.6x smaller than VGG16. It also achieves better accuracy and smaller parameter size than AutoKeras-120-160, a popular and state-of-the-art efficient model designed automatically, with the search cost of ENAS is 20.6x smaller. Such efficiency allows us to re-design ConvNets case-by-case.

REFERENCES

[1] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

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

[3] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.

[4] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019.

[5] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu. Practical block-wise neural network architecture generation. In *CVPR*, 2018.

[6] Z. Zhong, Z. Yang, B. Deng, J. Yan, W. Wu, J. Shao, and C.-L. Liu. Blockqnn: Efficient block-wise neural network architecture generation. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

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

[8] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017.

[9] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *ICML*, 2017.

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

[11] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, 2019.

[12] L. Xie and A. Yuille. Genetic cnn. In *ICCV*, 2017.

[13] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. In *ICLR*, 2018.

[14] H. Cai, L. Zhu, and S. Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019.

[15] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. In *ICLR*, 2019.

[16] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019.

[17] S. Xie, H. Zheng, C. Liu, and L. Lin. Snas: stochastic neural architecture search. In *ICLR*, 2019.

[18] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le. Understanding and simplifying one-shot architecture search. In *ICML*, 2018.

[19] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

[20] X. Zhang, Z. Huang, and N. Wang. You only search once: Single shot neural architecture search via direct sparse optimization. *arXiv preprint arXiv:1811.01567*, 2018.

[21] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[22] B. Wu, F. N. Iandola, P. H. Jin, and K. Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *CVPR Workshops*, 2017.

[23] B. Wu, A. Wan, X. Yue, and K. Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time roadobject segmentation from 3d lidar point cloud. In *ICRA*, 2018.

[24] B. Wu, X. Zhou, S. Zhao, X. Yue, and K. Keutzer. Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud. In *ICRA*, 2019.

[25] A. Gholami, K. Kwon, B. Wu, Z. Tai, X. Yue, P. Jin, S. Zhao, and K. Keutzer. Squeezenext: Hardware-aware neural network design. *arXiv preprint arXiv:1803.10615*, 2018.

[26] B. Wu, A. Wan, X. Yue, P. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez, and K. Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. *arXiv preprint arXiv:1711.08141*, 2017.

[27] H. Jin, Q. Song, and X. Hu. Auto-keras: An efficient neural architecture search system. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.