Learning Transferable Architectures for Scalable Image Recognition

Barret Zoph
Google Brain
barretzoph@google.com

Vijay Vasudevan
Google Brain
vrv@google.com

Jonathon Shlens
Google Brain
shlens@google.com

Quoc V. Le
Google Brain
qvl@google.com

Abstract

Developing state-of-the-art image classification models often requires significant architecture engineering and tuning. In this paper, we attempt to reduce the amount of architecture engineering by using Neural Architecture Search to learn an architectural building block on a small dataset that can be transferred to a large dataset. This approach is similar to learning the structure of a recurrent cell within a recurrent network. In our experiments, we search for the best convolutional cell on the CIFAR-10 dataset and then apply this learned cell to the ImageNet dataset by stacking together more of this cell. Although the cell is not learned directly on ImageNet, an architecture constructed from the best learned cell achieves state-of-the-art accuracy of 82.3% top-1 and 96.0% top-5 on ImageNet, which is 0.8% better in top-1 accuracy than the best human-invented architectures while having 9 billion fewer FLOPS. This cell can also be scaled down two orders of magnitude: a smaller network constructed from the best cell also achieves 74% top-1 accuracy, which is 3.1% better than the equivalently-sized, state-of-the-art models for mobile platforms.

1 Introduction

ImageNet classification [11] is a historically important benchmark in computer vision. The seminal work of Krizhevsky et al. [30] on using convolutional architectures [16, 31] for ImageNet classification represents one of the most important breakthroughs in deep learning. Successive advancements on this benchmark based on convolutional neural networks (CNNs) have achieved impressive results through significant architecture engineering [19, 44, 49, 50, 51, 58].

In this paper, we consider learning the convolutional architectures directly from data with application to ImageNet classification. We focus on ImageNet classification because the features derived from this network are of great importance in computer vision. For example, features from networks that perform well on ImageNet classification provide state-of-the-art performance when transferred to other computer vision tasks where labeled data is limited [12].

Our approach derives from the recently proposed Neural Architecture Search (NAS) framework [61], which uses a policy gradient algorithm to optimize architecture configurations. Running NAS directly on the ImageNet dataset is computationally expensive given the size of the dataset. We therefore use NAS to search for a good architecture on the far smaller CIFAR-10 dataset, and transfer the architecture to ImageNet. We achieve this transferrability by designing the search space so that the complexity of the architecture is independent of the depth of the network and the size of input images. More concretely, all convolutional networks in our search space are composed of convolutional cells with identical structures but different weights. Searching for the best convolutional architectures is therefore reduced to searching for the best cell structures. Searching for convolutional cells in this manner is much faster and the architecture itself is more likely to generalize to other problems. In particular, this approach significantly accelerates the search for the best architectures using CIFAR-10 (e.g., 4 weeks to 4 days) and learns architectures that successfully transfer to ImageNet.

Our primary result is that the best architecture found on CIFAR-10 achieves state-of-the-art accuracy on ImageNet classification without much modification. On ImageNet, an architecture constructed from the best cell achieves state-of-the-art accuracy of 82.3% top-1 and 96.0% top-5, which is 0.8% better in top-1 accuracy than the best human-invented architectures while having 9 billion fewer FLOPS. On CIFAR-10 itself, the architecture achieves 96.59% accuracy, while having fewer parameters than models with comparable performance. A small version of the state-of-the-art Shake-Shake model [17] with 2.9M parameters achieves 96.45% accuracy, while our 3.3M parameters model achieves a 96.59% accuracy. Not only our model has a better accuracy, it also needs only 600 epochs to train, which is one third of number of epochs for the Shake-Shake model.

Finally, by simply varying the number of the convolutional cells and number of filters in the convolutional cells, we can create convolutional architectures with different computational demands. In particular, we can generate a family of models that achieve accuracies superior to all human-invented models at equivalent or smaller computational budgets [27, 51]. Notably, the smallest version of the learned model achieves 74.0% top-1 accuracy on ImageNet, which is 3.1% better than previously engineered architectures targeted towards mobile and embedded vision tasks [22, 59].

2 Method

2.1 Neural Architecture Search for Large Scale Image Classification

Our work extends the Neural Architecture Search (NAS) framework proposed by Zoph and Le [61]. To briefly summarize the training procedure of NAS, a controller recurrent neural network (RNN) samples child networks with different architectures. The child networks are trained to convergence to obtain some accuracy on a held-out validation set. The resulting accuracies are used to update the controller so that the controller will generate better architectures over time. The controller weights are updated using a policy gradient method (Figure 1).

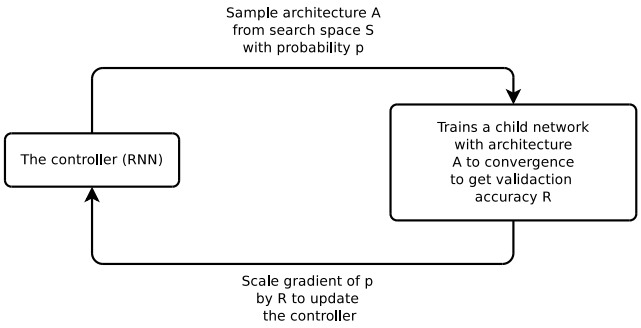

Figure 1: Overview of Neural Architecture Search [61]. A controller RNN predicts architecture A from search space S with probability p. A child network with architecture A is trained to convergence achieving accuracy R. Scale the gradients of p by R to update the RNN controller.

A key element of NAS is to design the search space S to generalize across problems of varying complexity and spatial scales. We observed that applying NAS directly on the ImageNet dataset would be very expensive and require months to complete an experiment. However, if the search space is properly constructed, architectural elements can transfer across datasets [61].

The focus of this work is to design a search space, such that the best architecture found on the CIFAR-10 dataset would scale to larger, higher-resolution image datasets across a range of computational settings. One inspiration for this search space is the recognition that architecture engineering with CNNs often identifies repeated motifs consisting of combinations of convolutional filter banks, nonlinearities and a prudent selection of connections to achieve state-of-the-art results [19, 49, 50, 51]. These observations suggest that it may be possible for the controller RNN to predict a generic *convolutional cell* expressed in terms of these motifs. This cell can then be stacked in series to handle inputs of arbitrary spatial dimensions and filter depth.

To construct a complete model for image classification, we take an architecture for a convolutional cell and simply repeat it many times. Each convolutional cell would have the same architecture, but

have different weights. To easily build scalable architectures for images of any size, we need two types of convolutional cells to serve two main functions when taking in a feature map as input: (1) convolutional cells that return a feature map of the same dimension, and (2) convolutional cells that return a feature map where the feature map height and width is reduced by a factor of two. We name the first type and second type of convolutional cells *Normal Cell* and *Reduction Cell* respectively. For the Reduction Cell, to reduce the height and width by a factor of two, we make the initial operation applied to cell's inputs have a stride of two. All of our operations that we consider for building our convolutional cells have an option of striding.

Figure 2 shows our placement of Normal and Reduction Cells for CIFAR and ImageNet. Note on ImageNet we have more Reduction Cells, since the incoming image size is 299x299 compared to 32x32 for CIFAR. The Reduction and Normal Cell could be the same architecture, but we empirically found it was beneficial to learn two separate architectures. We employ a common heuristic to double the number of filters in the output whenever the spatial activation size is reduced in order to maintain roughly constant hidden state dimension [30, 44]. Importantly, we consider the number of motif repetitions N and the number of initial convolutional filters as free parameters that we tailor to the scale of an image classification problem.

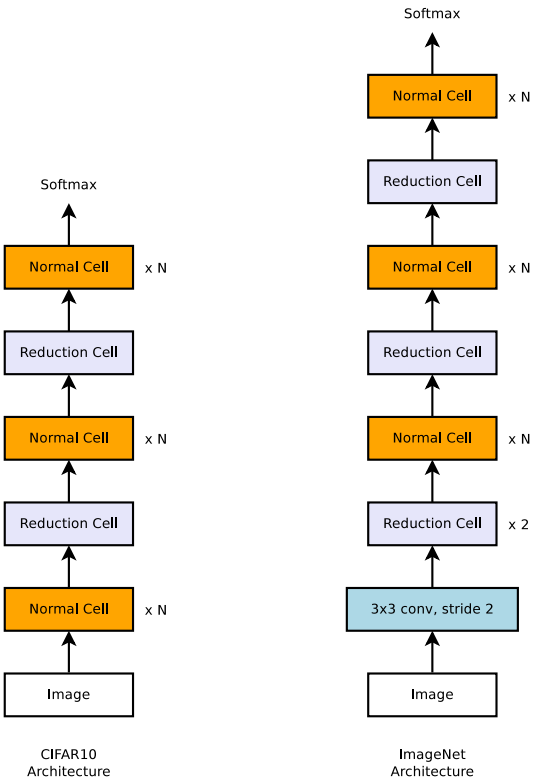

Figure 2: Scalable architecture for image classification consists of two repeated motifs termed *Normal Cell* and *Reduction Cell*. This diagram highlights the model architecture for CIFAR-10 and ImageNet. The choice for the number of times the Normal Cells that gets stacked between reduction cells, N, can vary in our experiments.

2.2 Search Space for Convolutional Cells

Our search space differs from [61] where the controller needs to predict the entire architecture of the neural networks. In our method, the controller needs to predict the structures of the two convolutional cells (Normal Cell and Reduction Cell), which can be then stacked many times to create the eventual architecture shown in Figure 2-Left. The convolutional cell is inspired by the concept of recurrent cells, where the structure of the cells is independent of the number of time steps in the recurrent

network. This is an effective way to decouple the complexity of the cells from the depth of the neural network so that the controller RNN only needs to focus on predicting the structure of the cell.

Each cell receives as input two initial hidden states h_i and h_{i-1} which are the outputs of two cells in previous two lower layers or the input image. The job of the controller RNN is to recursively predict the rest of the structure of the convolutional cell (Figure 3). The predictions of the controller for each cell are grouped into B blocks, where each block has 5 prediction steps made by 5 distinct softmax classifiers corresponding to discrete choices of the elements of a block:

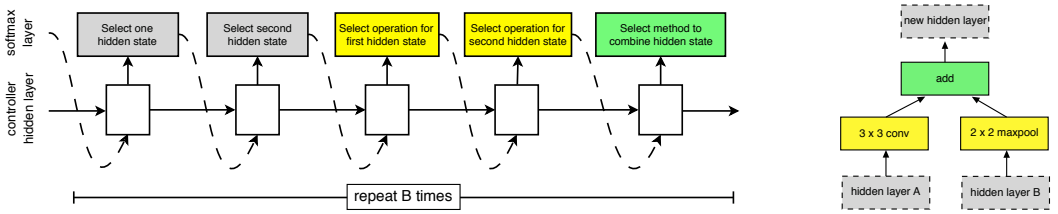

Figure 3: Controller model architecture for recursively constructing one block of a convolutional cell. Each block requires selecting 5 discrete parameters, each of which corresponds to the output of a softmax layer. Example constructed block shown on right. A convolutional cell contains B blocks, hence the controller contains $5B$ softmax layers for predicting the architecture of a convolutional cell. In our experiments, the number of blocks B is 5.

Step 1. Select a hidden state from h or from the set of hidden states created in previous blocks.

Step 2. Select a second hidden state from the same options as in Step 1.

Step 3. Select an operation to apply to the hidden state selected in Step 1.

Step 4. Select an operation to apply to the hidden state selected in Step 2.

Step 5. Select a method to combine the outputs of Step 3 and 4 to create a new hidden state.

The algorithm appends the newly-created hidden state to the set of existing hidden states as a potential input in subsequent blocks. The controller RNN repeats the above 5 prediction steps B times corresponding to the B blocks in a convolutional cell. In our experiments, selecting $B = 5$ provides good results, although we have not exhaustively searched this space due to computational limitations.

In steps 3 and 4, the controller RNN selects an operation to apply to the hidden states. We collected the following set of operations based on their prevalence in the CNN literature:

- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 separable convolution
- 7x7 separable convolution
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 separable convolution

In our experiments, we apply each separable operation twice during the execution of the child model, once that operation is selected by the controller.

In step 5 the controller RNN selects a method to combine the two hidden states, either (1) elementwise addition between two hidden states and (2) concatenation between two hidden states along the filter dimension. Finally, all of the unused hidden states generated in the convolutional cell are concatenated together in depth to provide the final cell output.

To have the controller RNN predict both the Normal and Reduction cell we simply make the controller have $2 \times 5B$ predictions in total, where the first $5B$ predictions are for the Normal Cell and the second $5B$ predictions are for the Reduction Cell.

3 Experiments and Results

In this section, we describe our experiments with Neural Architecture Search using the search space described above to learn a convolutional cell. In summary, all architecture searches are performed using the CIFAR-10 classification task [29]. The controller RNN was trained using Proximal Policy Optimization (PPO) [43] by employing a global workqueue system for generating a pool of child networks controlled by the RNN. In our experiments, the pool of workers in the workqueue consisted of 450 GPUs. Please see Appendix A for complete details of the architecture learning and controller system.

The result of this search process yields several candidate convolutional cells. Figure 4 shows a diagram of the top performing cell. Note the prevalence of separable convolutions and the number of branches as compared with competing architectures [19, 44, 49, 50, 51]. Subsequent experiments focus on this convolutional cell architecture, although we examine the efficacy of other, top-ranked convolutional cells in ImageNet experiments (described in Appendix B) and report their results as well. We call the three networks constructed from the best three cells *NASNet-A*, *NASNet-B* and *NASNet-C*.

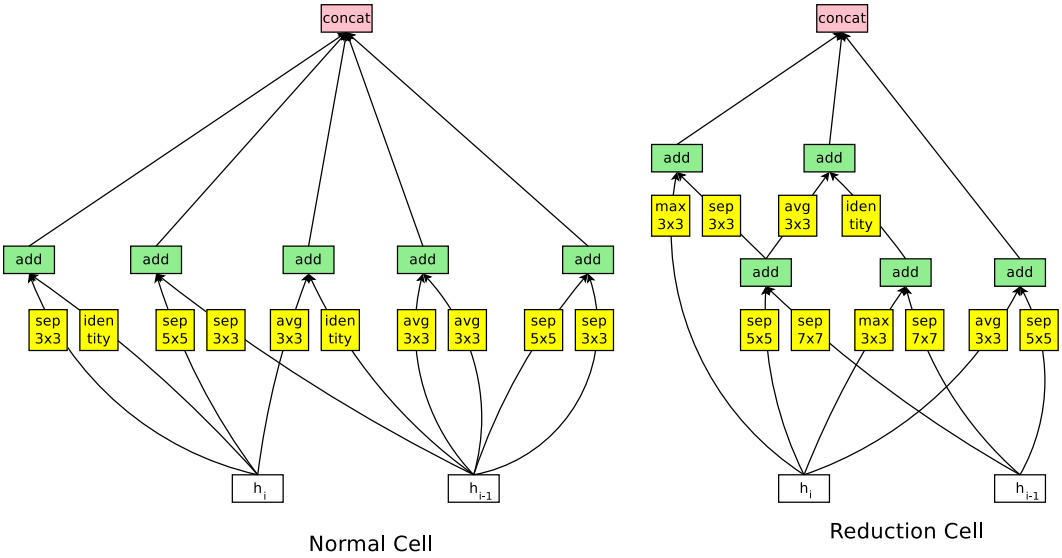

Figure 4: Architecture of the best convolutional cells (NASNet-A) with $B = 5$ blocks identified with CIFAR-10 . The input (white) is the hidden state from previous activations (or input image). The output (pink) is the result of a concatenation operation across all resulting branches. Each convolutional cell is the result of B blocks. A single block is corresponds to two primitive operations (yellow) and a combination operation (green). Note that colors correspond to operations in Figure 3.

We demonstrate the utility of the convolutional cell by employing this learned architecture on CIFAR-10 and a family of ImageNet classification tasks. The latter family of tasks is explored across a few orders of magnitude in computational budget. After having learned the convolutional cell, several hyper-parameters may be explored to build a final network for a given task: (1) the number of cell repeats N and (2) the number of filters in the initial convolutional cell. We employ a common heuristic to double the number of filters whenever the stride is 2.

3.1 General Training Strategies

We found that adding Batch Normalization and/or a ReLU between the depthwise and pointwise operations in the separable convolution operations to not help performance. L1 regularization was tried with the NASNet models, but this was found to hurt performance. We also tried ELU [10] instead of ReLUs and found that performance was about the same. Dropout [47] was also tried on the convolutional filters, but this was found to degrade performance.

Operation Ordering: All convolution operations that could be predicted are applied within the convolutional cells in the following order: ReLU, convolution operation, Batch Normalization. We found this order to improve performance over a different ordering: convolution operation, Batch Normalization and ReLU. This result is inline with findings from other papers where using the pre-ReLU activation works better. In order to be sure shapes always match in the convolutional cells, 1x1 convolutions are inserted as necessary.

Cell Path Dropout: When training our NASNet models, we found stochastically dropping out each path (edge with a yellow box) in the cell with some fixed probability to be an extremely good regularizer. This is similar to [25] and [60] where they dropout full parts of their model during training and then at test time scale the path by the probability of keeping that path during training. Interestingly we found that linearly increasing the probability of dropping out a path over the course of training to significantly improve the final performance for both CIFAR and ImageNet experiments.

3.2 Results on CIFAR-10 Image Classification

For the task of image classification with CIFAR-10, we set N=4 or 6 (Figure 2). The networks are trained on CIFAR-10 for 600 epochs. The test accuracies of the best architectures are reported in Table 1 along with other state-of-the-art models; the best architectures found by the controller RNN are better than the previous state-of-the-art Shake-Shake model when comparing with the same number of parameters. Additionally, we only train for a third of the time where Shake-Shake trains for 1800 epochs. See appendix A for more details on CIFAR training.

Model	Depth	# parameters	Error rate (%)
DenseNet ($L = 40, k = 12$) [23]	40	1.0M	5.24
DenseNet($L = 100, k = 12$) [23]	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) [23]	100	27.2M	3.74
DenseNet-BC ($L = 100, k = 40$) [24]	190	25.6M	3.46
Shake-Shake 26 2x32d [17]	26	2.9M	3.55
Shake-Shake 26 2x96d [17]	26	26.2M	2.86
NAS v1 no stride or pooling [61]	15	4.2M	5.50
NAS v2 predicting strides [61]	20	2.5M	6.01
NAS v3 max pooling [61]	39	7.1M	4.47
NAS v3 max pooling + more filters [61]	39	37.4M	3.65
NASNet-A	N=6	3.3M	3.41
NASNet-B	N=4	2.6M	3.73
NASNet-C	N=4	3.1M	3.59

Table 1: Performance of Neural Architecture Search and other state-of-the-art models on CIFAR-10.

3.3 Results on ImageNet Image Classification

We performed several sets of experiments on ImageNet with the best convolutional cells learned from CIFAR-10. Results are summarized in Table 2 and 3 and Figure 5. In the first set of experiments, we train several image classification systems operating on 299x299 or 331x331 resolution images scaled in computational demand on par with Inception-v2 [27], Inception-v3 [51] and PolyNet [60]. We demonstrate that this family of models achieve state-of-the-art performance with fewer floating point operations and parameters than comparable architectures. Second, we demonstrate that by adjusting the scale of the model we can achieve state-of-the-art performance at smaller computational budgets, exceeding streamlined CNNs hand-designed for this operating regime [22, 59].

Note we do not have residual connections around convolutional cells as the models learn skip connections on their own. We empirically found manually inserting residual connections between cells to not help performance. Our training setup on ImageNet is similar to [51], but please see Appendix A for details.

Table 2 shows that the convolutional cells discovered with CIFAR-10 generalize well to ImageNet problems. In particular, each model based on the convolutional cell exceeds the predictive performance of the corresponding hand-designed model. Importantly, the largest model achieves a new

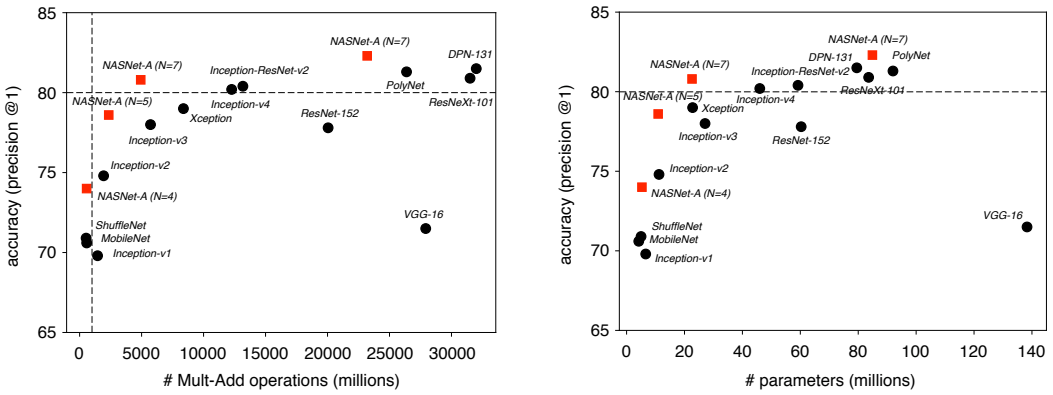

Figure 5: Accuracy versus computational demand (left) and number of parameters (right) across top performing CNN architectures on ImageNet 2012 ILSVRC challenge prediction task (compiled as of July 2017). Computational demand is measured in the number of floating-point multiply-add operations to process a single image. Black circles indicate previously published work and red squares highlight our proposed models. Vertical dashed line indicates 1 billion multiply-add operations. Horizontal dashed line indicates 80% precision@1 prediction accuracy.

Model	image size	# parameters	Mult-Adds	Top 1 Acc. (%)	Top 5 Acc. (%)
Inception V2 [27]	224×224	11.2 M	1.94 B	74.8	92.2
NASNet-A (N = 5)	**299×299**	**10.9 M**	**2.35 B**	**78.6**	**94.2**
Inception V3 [51]	299×299	23.8 M	5.72 B	78.0	93.9
Xception [9]	299×299	22.8 M	8.38 B	79.0	94.5
Inception ResNet V2 [50]	299×299	55.8 M	13.2 B	80.4	95.3
NASNet-A (N = 7)	**299×299**	**22.6 M**	**4.93 B**	**80.8**	**95.3**
ResNeXt-101 (64 x 4d) [58]	320×320	83.6 M	31.5 B	80.9	95.6
PolyNet [60]	331×331	92 M	34.7 B	81.3	95.8
DPN-131 [8]	320×320	79.5 M	32.0 B	81.5	95.8
NASNet-A (N = 7)	**331×331**	**84.9 M**	**23.2 B**	**82.3**	**96.0**

Table 2: Performance of architecture search and other state-of-the-art models on ImageNet classification. Mult-Adds indicate the number of composite multiply-accumulate operations for a single image.

Model	# parameters	Mult-Adds	Top 1 Acc. (%)	Top 5 Acc. (%)
Inception V1 [49]	6.6M	1,448 M	69.8	89.9
MobileNet-224 [22]	4.2 M	569 M	70.6	89.5
ShuffleNet (2x) [59]	∼ 5M	524 M	70.9	89.8
NASNet-A (N=4)	**5.3 M**	**564 M**	**74.0**	**91.6**
NASNet-B (N=4)	5.3M	488 M	72.8	91.3
NASNet-C (N=3)	4.9M	558 M	72.5	91.0

Table 3: Performance on ImageNet classification on a subset of models operating in a constrained computational setting, i.e., < 1.5 B multiply-accumulate operations per image. All models employ 224x224 images.

state-of-the-art performance for ImageNet (82.3%) based on single, non-ensembled predictions, surpassing previous state-of-the-art by 0.8% [8]. Figure 5 shows a complete summary of these results. Note the family of models based on convolutional cells provides an envelope over a broad class of human-invented architectures.

Finally, we test how well the best convolutional cells may perform in a resource-constrained setting, e.g., mobile devices (Table 3). In these settings, the number of floating point operations is severely

constrained and predictive performance must be weighed against latency requirements on a device with limited computational resources. MobileNet [22] and ShuffleNet [59] provide state-of-the-art results predicting 70.6% and 70.9% accuracy, respectively on 224x224 images using $\sim$ 550M multiply-add operations. An architecture constructed from the best convolutional cells achieves superior predictive performance (74.0% accuracy) surpassing previous models but with comparable computational demand. In summary, we find that the learned convolutional cells are flexible across model scales achieving state-of-the-art performance across almost 2 orders of magnitude in computational budget.

4 Related Work

The proposed method is related to previous work in hyperparameter optimization [4, 5, 6, 33, 37, 45, 46] – especially recent approaches in designing architectures such as Neural Fabrics [40], DiffRNN [34], MetaQNN [3] and DeepArchitect [36]. A more flexible class of methods for designing architecture is evolutionary algorithms [15, 28, 35, 39, 48, 55, 57], yet they have not had as much success at large scale. Xie and Yuille [57] also transferred learned architectures from CIFAR-10 to ImageNet but performance of these models (top-1 accuracy 72.1%) are notably below previous state-of-the-art (Table 2).

The concept of having one neural network interact with a second neural network to aid the learning process, or learning to learn or meta-learning [21, 41] has attracted much attention in recent years [1, 13, 14, 18, 32, 38, 52]. Most of these approaches have not been scaled to large problems like ImageNet. A notable exception is recent work focused on learning an optimizer for ImageNet classification that achieved notable improvements [54].

The design of our search space took much inspiration from LSTMs [20], and NASCell [61]. The modular structure of the convolutional cell is also related to previous methods on ImageNet such as VGG [44], Inception [49, 50, 51], ResNet/ResNext [19, 58], and Xception/MobileNet [9, 22].

5 Conclusion

In this work, we demonstrate how to learn scalable, convolutional cells from data that transfer to multiple image classification tasks. The key insight to this approach is to design a search space that decouples the complexity of an architecture from the depth of a network. This resulting search space permits identifying good architectures on a small dataset (i.e., CIFAR-10) and transferring the learned architecture to image classifications across a range of data and computational scales.

The resulting architectures approach or exceed state-of-the-art performance in terms of CIFAR-10, ImageNet classification with less computational demand than human-designed architectures [27, 51, 60]. The ImageNet results are particularly important because many state-of-the-art computer vision problems (e.g., object detection [26], face detection [42], image localization [53]) derive image features or architectures from ImageNet classification models. Finally, we demonstrate that we can employ the resulting learned architecture to perform ImageNet classification with reduced computational budgets that outperform streamlined architectures targeted to mobile and embedded platforms [22, 59].

Our results have strong implications for transfer learning and meta-learning as this is the first work to demonstrate state-of-the-art results using meta-learning on a large scale problem. This work also highlights that learned elements of network architectures, beyond model weights, can transfer across datasets.

Acknowledgements

We thank Jeff Dean, Yifeng Lu, Jonathan Shen, Vishy Tirumalashetty, Xiaoqiang Zheng, and the Google Brain team for the help with the project. We additionally thank Christian Sigg for performance improvements to depthwise separable convolutions.

References

[1] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[3] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*, 2016.

[4] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 2012.

[5] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Neural Information Processing Systems*, 2011.

[6] James Bergstra, Daniel Yamins, and David D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *International Conference on Machine Learning*, 2013.

[7] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. In *International Conference on Learning Representations Workshop Track*, 2016.

[8] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks. *arXiv preprint arXiv:1707.01083*, 2017.

[9] François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.

[10] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2009.

[12] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *Icml*, volume 32, pages 647–655, 2014.

[13] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[14] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[15] Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 2008.

[16] Kunihiko Fukushima. A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, page 93–202, 1980.

[17] Xavier Gastaldi. Shake-shake regularization of 3-branch residual networks. In *International Conference on Learning Representations Workshop Track*, 2017.

[18] David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[20] Sepp Hochreiter and Juergen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[21] Sepp Hochreiter, A Younger, and Peter Conwell. Learning to learn using gradient descent. *Artificial Neural Networks*, pages 87–94, 2001.

[22] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[23] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[24] Gao Huang, Zhuang Liu, Kilian Q. Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[25] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep networks with stochastic depth. *arXiv preprint arXiv:1603.09382*, 2016.

[26] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. *arXiv preprint arXiv:1611.10012*, 2016.

[27] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[28] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *ICML*, 2015.

[29] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing System*, 2012.

[31] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[32] Ke Li and Jitendra Malik. Learning to optimize neural nets. *arXiv preprint arXiv:1703.00441*, 2017.

[33] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Towards automatically-tuned neural networks. In *Proceedings of the 2016 Workshop on Automatic Machine Learning*, pages 58–65, 2016.

[34] Thomas Miconi. Neural networks with differentiable structure. *arXiv preprint arXiv:1606.06216*, 2016.

[35] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*, 2017.

[36] Renato Negrinho and Geoff Gordon. DeepArchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*, 2017.

[37] Nicolas Pinto, David Doukhan, James J DiCarlo, and David D Cox. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS Computational Biology*, 5(11):e1000579, 2009.

[38] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2017.

[39] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017.

[40] Shreyas Saxena and Jakob Verbeek. Convolutional neural fabrics. In *Advances in Neural Information Processing Systems*, 2016.

[41] Tom Schaul and Juergen Schmidhuber. Metalearning. *Scholarpedia*, 2010.

[42] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

[43] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, 2015.

[44] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[45] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Neural Information Processing Systems*, 2012.

[46] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mostofa Ali, Ryan P. Adams, et al. Scalable Bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, 2015.

[47] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[48] Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 2009.

[49] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[50] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

[51] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[52] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[53] Tobias Weyand, Ilya Kostrikov, and James Philbin. Planet-photo geolocation with convolutional neural networks. *arXiv preprint arXiv:1602.05314*, 2016.

[54] Olga Wichrowska, Niru Maheswaranathan, Matthew W. Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. *arXiv preprint arXiv:1703.04813*, 2017.

[55] Daan Wierstra, Faustino J. Gomez, and Jürgen Schmidhuber. Modeling systems with internal state using evolino. In *The Genetic and Evolutionary Computation Conference*, 2005.

[56] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, 1992.

[57] Lingxi Xie and Alan Yuille. Genetic CNN. *arXiv preprint arXiv:1703.01513*, 2017.

[58] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*, 2016.

[59] Xiangyu Zhang, Xinyu Zhou, Lin Mengxiao, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.

[60] Xingcheng Zhang, Zhizhong Li, Chen Change Loy, and Dahua Lin. Polynet: A pursuit of structural diversity in very deep networks. *arXiv preprint arXiv:1611.05725*, 2016.

[61] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

Appendix

A Experimental Details

A.1 Dataset for Architecture Search

The CIFAR-10 dataset [29] consists of 60,000 32x32 RGB images across 10 classes (50,000 train and 10,000 test images). We partition a random subset of 5,000 images from the training set to use as a validation set for the controller RNN. All images are whitened and then undergone several data augmentation steps: we randomly crop 32x32 patches from upsampled images of size 40x40 and apply random horizontal flips. This data augmentation procedure is common among related work.

A.2 Controller architecture

The controller RNN is a one-layer LSTM [20] with 100 hidden units at each layer and $2 \times 5B$ softmax predictions for the two convolutional cells (where B is typically 5) associated with each architecture decision. Each of the $10B$ predictions of the controller RNN is associated with a probability. The joint probability of a child network is the product of all probabilities at these $10B$ softmaxes. This joint probability is used to compute the gradient for the controller RNN. The gradient is scaled by the validation accuracy of the child network to update the controller RNN such that the controller assigns low probabilities for bad child networks and high probabilities for good child networks.

Unlike Zoph and Le [61], who used the REINFORCE rule [56] to update the controller, we employ Proximal Policy Optimization (PPO) [43] with learning rate 0.00035 because it made training more robust and increased convergence speed. To encourage exploration we also use an entropy penalty with a weight of 0.00001. We also use a baseline function, which we set to be an exponential moving average of previous rewards, with a weight of 0.95. The weights of the controller are initialized uniformly between -0.1 and 0.1.

A.3 Training of the Controller

For distributed training, we use a work queue system where all the samples generated from the controller RNN are added to a global workqueue. A free "child" worker in a distributed worker pool asks the controller for new work from the global workqueue. Once the training of the child network is complete, the accuracy on a held-out validation set is computed and reported to the controller RNN. In our experiments we use a child worker pool size of 450, which means there are 450 networks being trained on 450 GPUs concurrently at any time. Upon receiving enough child model training results, the Controller RNN will perform a gradient update on its weights using TRPO and then sample another batch of architectures that go into the global work queue. This process continues until a predetermined number of architectures have been sampled. In our experiments, this predetermined number of architectures is 20,000 which means the search process is terminated after 20,000 child models have been trained. Additionally, we update the controller RNN with minibatches of 20 architectures. Once the search is over, the top 250 architectures are then chosen to train until convergence on CIFAR-10 to determine the very best architecture.

A.4 Training of CIFAR models

All of our CIFAR models use a single period cosine decay as in [17]. All models use the momentum optimizer with momentum rate set to 0.9. All models also use L2 weight decay. Each architecture is trained for a fixed 20 epochs on CIFAR-10 during the architecture search process. Additionally, we found it beneficial to use the cosine learning rate decay during the 20 epochs the CIFAR models were trained for as this helped to further differentiate good architectures. We also found that having the CIFAR models use a small N=2 during the architecture search process allowed for models to train quite quickly, while still finding cells that work well once more were stacked.

A.5 Training of ImageNet models

We use ImageNet 2012 ILSVRC challenge data for large scale image classification. The dataset consists of $\sim$ 1.2M images labeled across 1000 classes [11]. Overall our training and testing

procedures are almost identical to [51]. ImageNet models are trained and evaluated on 299x299 or 331x331 images using the same data augmentation procedures as described previously [51]. We use distributed synchronous SGD to train the ImageNet model with 50 workers (and 3 backup workers) each with a Tesla K40 GPU [7]. We use RMSProp with a decay of 0.9 and epsilon of 1.0. Evaluations are calculated using with a running average of parameters over time with a decay rate of 0.9999. We use label smoothing with a value of 0.1 for all ImageNet models as done in [51]. Additionally, all models use an auxiliary classifier located at 2/3 of the way up the network. The loss of the auxiliary classifier is weighted by 0.4 as done in [51]. We empirically found our network to be insensitive to the number of parameters associated with this auxiliary classifier along with the weight associated with the loss. All models also use L2 regularization. The learning rate decay scheme is the exponential decay scheme used in [51]. Dropout is applied to the final softmax matrix with probability 0.5.

B Additional Experiments

We now present two additional cells that performed well on CIFAR and ImageNet. The search spaces used for these cells are slightly different than what was used for NASNet-A. For the NASNet-B model in Figure 6 we do not concatenate all of the unused hidden states generated in the convolutional cell. Instead all of the hiddenstates created within the convolutional cell, even if they are currently used, are fed into the next layer. Note that $B = 4$ and there are 4 hiddenstates as input to the cell as these numbers must match for this cell to be valid. We also allow addition followed by layer normalization [2] or instance normalization to be predicted as two of the combination operations within the cell, along with addition or concatenation.

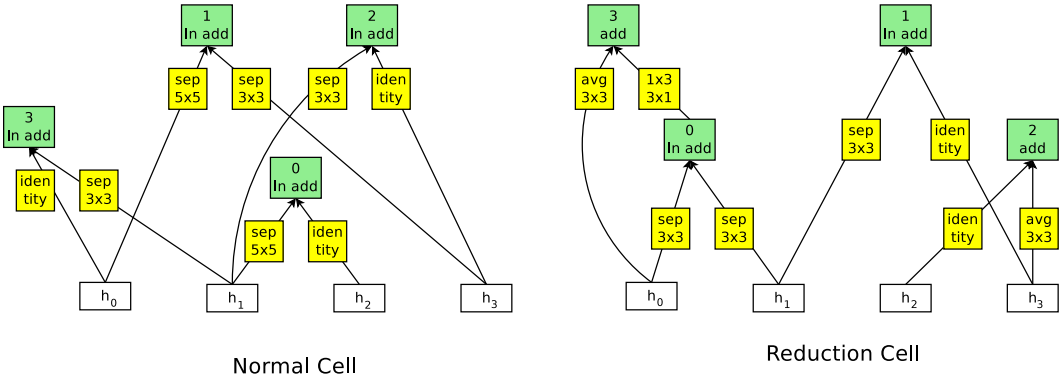

Figure 6: Architecture of NASNet-B convolutional cell with $B = 4$ blocks identified with CIFAR-10. The input (white) is the hidden state from previous activations (or input image). Each convolutional cell is the result of B blocks. A single block is corresponds to two primitive operations (yellow) and a combination operation (green). As do we not concatenate the output hidden states, each output hidden state is used as a hidden state in the future layers. Each cell takes in 4 hidden states and thus needs to also create 4 output hidden states. Each output hidden state is therefore labeled with 0, 1, 2, 3 to represent the next four layers in that order.

For NASNet-C (Figure 7), we concatenate all of the unused hidden states generated in the convolutional cell like in NASNet-A, but now we allow the prediction of addition followed by layer normalization or instance normalization like in NASNet-B.

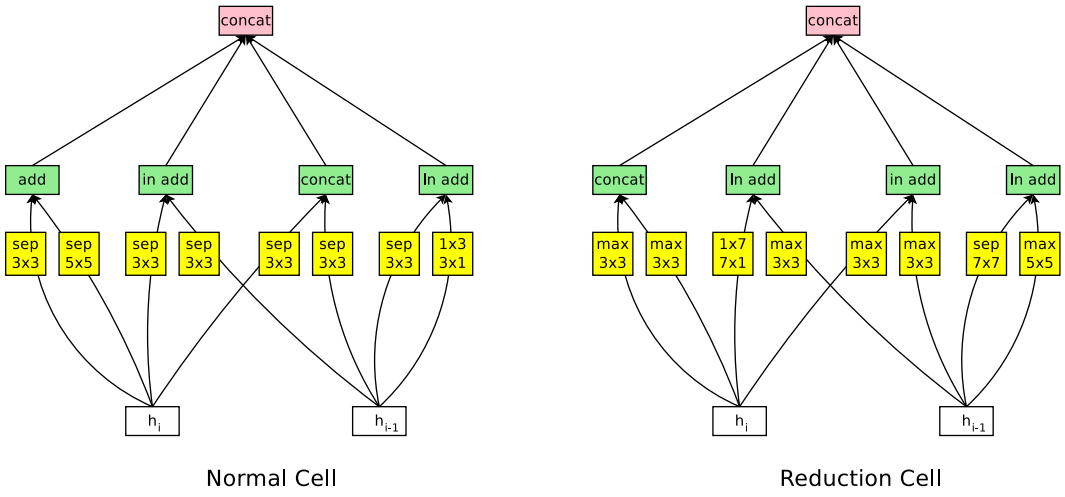

Figure 7: Architecture of NASNet-C convolutional cell with $B = 4$ blocks identified with CIFAR-10. The input (white) is the hidden state from previous activations (or input image). The output (pink) is the result of a concatenation operation across all resulting branches. Each convolutional cell is the result of B blocks. A single block corresponds to two primitive operations (yellow) and a combination operation (green).