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 neural network image classification models often requires significant architecture engineering. In this paper, we study a method to learn the model architectures directly on the dataset of interest. As this approach is expensive when the dataset is large, we propose to search for an architectural building block on a small dataset and then transfer the block to a larger dataset. The key contribution of this work is the design of a new search space (which we call the "NASNet search space") which enables transferability. In our experiments, we search for the best convolutional layer (or "cell") on the CIFAR-10 dataset and then apply this cell to the ImageNet dataset by stacking together more copies of this cell, each with their own parameters to design a convolutional architecture, which we name a "NASNet architecture". We also introduce a new regularization technique called ScheduledDropPath that significantly improves generalization in the NASNet models. On CIFAR-10 itself, a NASNet found by our method achieves 2.4% error rate, which is state-of-the-art. Although the cell is not searched for directly on ImageNet, a NASNet constructed from the best cell achieves, among the published works, state-of-the-art accuracy of 82.7% top-1 and 96.2% top-5 on ImageNet. Our model is 1.2% better in top-1 accuracy than the best human-invented architectures while having 9 billion fewer FLOPS – a reduction of 28% in computational demand from the previous state-of-the-art model. When evaluated at different levels of computational cost, accuracies of NASNets exceed those of the state-of-the-art human-designed models. For instance, a small version of NASNet also achieves 74% top-1 accuracy, which is 3.1% better than equivalently-sized, state-of-the-art models for mobile platforms. Finally, the image features learned from image classification are generically useful and can be transferred to other computer vision problems. On the task of object detection, the learned features by NASNet used with the Faster-RCNN framework surpass state-of-the-art by 4.0% achieving 43.1% mAP on the COCO dataset.

1. Introduction

Developing neural network image classification models often requires significant *architecture engineering*. Starting from the seminal work of [32] on using convolutional architectures [17, 34] for ImageNet [11] classification, successive advancements through architecture engineering have achieved impressive results [53, 59, 20, 60, 58, 68].

In this paper, we study a new paradigm of designing convolutional architectures and describe a scalable method to optimize convolutional architectures on a dataset of interest, for instance the ImageNet classification dataset. Our approach is inspired by the recently proposed Neural Architecture Search (NAS) framework [71], which uses a reinforcement learning search method to optimize architecture configurations. Applying NAS, or any other search methods, directly to a large dataset, such as the ImageNet dataset, is however computationally expensive. We therefore propose to search for a good architecture on a proxy dataset, for example the smaller CIFAR-10 dataset, and then transfer the learned architecture to ImageNet. We achieve this transferrability by designing a search space (which we call "the NASNet 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 layers (or "cells") with identical structure but different weights. Searching for the best convolutional architectures is therefore reduced to searching for the best cell structure. Searching for the best cell structure has two main benefits: it is much faster than searching for an entire network architecture and the cell itself is more likely to generalize to other problems. In our experiments, this approach significantly accelerates the search for the best architectures using CIFAR-10 by a factor of 7× and learns architectures that successfully transfer to ImageNet.

Our main result is that the best architecture found on CIFAR-10, called NASNet, achieves state-of-the-art accuracy when transferred to ImageNet classification without much modification. On ImageNet, NASNet achieves, among the published works, state-of-the-art accuracy of 82.7% top-1 and 96.2% top-5. This result amounts to a

1.2% improvement in top-1 accuracy than the best human-invented architectures while having 9 billion fewer FLOPS. On CIFAR-10 itself, NASNet achieves 2.4% error rate, which is also state-of-the-art.

Additionally, by simply varying the number of the convolutional cells and number of filters in the convolutional cells, we can create different versions of NASNets with different computational demands. Thanks to this property of the cells, we can generate a family of models that achieve accuracies superior to all human-invented models at equivalent or smaller computational budgets [60, 29]. Notably, the smallest version of NASNet achieves 74.0% top-1 accuracy on ImageNet, which is 3.1% better than previously engineered architectures targeted towards mobile and embedded vision tasks [24, 70].

Finally, we show that the image features learned by NASNets are generically useful and transfer to other computer vision problems. In our experiments, the features learned by NASNets from ImageNet classification can be combined with the Faster-RCNN framework [47] to achieve state-of-the-art on COCO object detection task for both the largest as well as mobile-optimized models. Our largest NASNet model achieves 43.1% mAP, which is 4% better than previous state-of-the-art.

2. Related Work

The proposed method is related to previous work in hyperparameter optimization [44, 4, 5, 54, 55, 6, 40] – especially recent approaches in designing architectures such as Neural Fabrics [48], DiffRNN [41], MetaQNN [3] and DeepArchitect [43]. A more flexible class of methods for designing architecture is evolutionary algorithms [65, 16, 57, 30, 46, 42, 67], yet they have not had as much success at large scale. Xie and Yuille [67] 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 [23, 49] has attracted much attention in recent years [1, 62, 14, 19, 35, 45, 15]. Most of these approaches have not been scaled to large problems like ImageNet. An exception is the recent work focused on learning an optimizer for ImageNet classification that achieved notable improvements [64].

The design of our search space took much inspiration from LSTMs [22], and Neural Architecture Search Cell [71]. The modular structure of the convolutional cell is also related to previous methods on ImageNet such as VGG [53], Inception [59, 60, 58], ResNet/ResNext [20, 68], and Xception/MobileNet [9, 24].

3. Method

Our work makes use of search methods to find good convolutional architectures on a dataset of interest. The main search method we use in this work is the Neural Architecture Search (NAS) framework proposed by [71]. In 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 with policy gradient (see Figure 1).

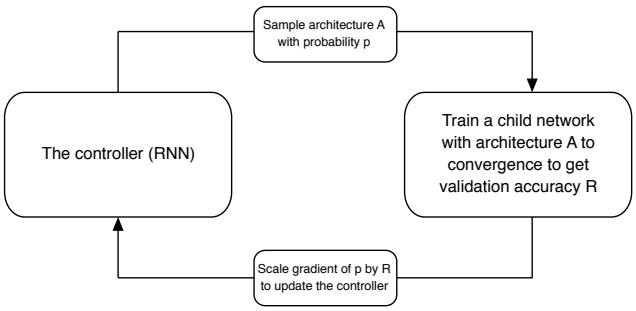

Figure 1. Overview of Neural Architecture Search [71]. A controller RNN predicts architecture A from a search space with probability p. A child network with architecture A is trained to convergence achieving accuracy R. Scale the gradients of p by R to update the RNN controller.

The main contribution of this work is the design of a novel 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. We name this search space *the NASNet search space* as it gives rise to NASNet, the best architecture found in our experiments. One inspiration for the NASNet search space is the realization 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 (such as the repeated modules present in the Inception and ResNet models [59, 20, 60, 58]). 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.

In our approach, the overall architectures of the convolutional nets are manually predetermined. They are composed of convolutional cells repeated many times where each convolutional cell has the same architecture, but 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

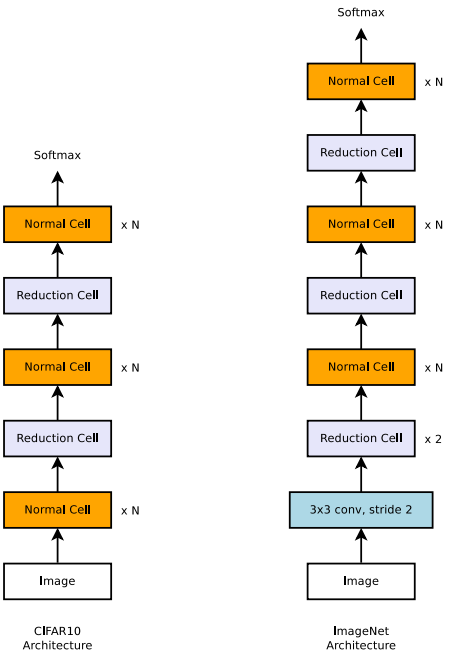

Figure 2. Scalable architectures for image classification consist of two repeated motifs termed *Normal Cell* and *Reduction Cell*. This diagram highlights the model architecture for CIFAR-10 and ImageNet. The choice for the number of times the Normal Cells that gets stacked between reduction cells,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

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, we make the initial operation applied to the cell's inputs have a stride of two to reduce the height and width. 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-10 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 have the same architecture, but we empirically found it beneficial to learn two separate architectures. We use 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 [32, 53]. Importantly, much like Inception and ResNet models [59, 20, 60, 58], 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.

What varies in the convolutional nets is the structures of the Normal and Reduction Cells, which are searched by the controller RNN. The structures of the cells can be searched within a search space defined as follows (see Appendix, Figure 7 for schematic). In our search space, 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 controller RNN recursively predicts the rest of the structure of the convolutional cell, given these two initial hidden states (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:

Step 1. Select a hidden state from h_i, h_{i-1} 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 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv

In step 5 the controller RNN selects a method to combine the two hidden states, either (1) element-wise addition between two hidden states or (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 allow the controller RNN to predict both Normal Cell 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.

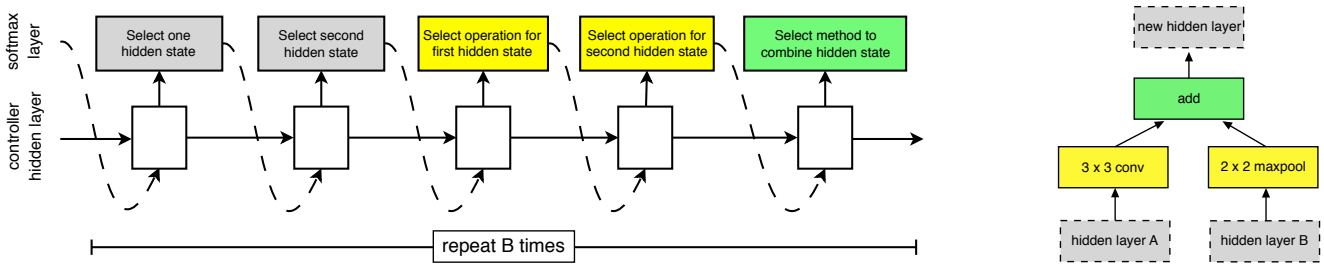

Figure 3. Controller model architecture for recursively constructing one block of a convolutional cell. Each block requires selecting 5 discrete parameters, each of which corresponds to the output of a softmax layer. Example constructed block shown on right. A convolutional cell contains B blocks, hence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

Finally, our work makes use of the reinforcement learning proposal in NAS [71]; however, it is also possible to use random search to search for architectures in the NASNet search space. In random search, instead of sampling the decisions from the softmax classifiers in the controller RNN, we can sample the decisions from the uniform distribution. In our experiments, we find that random search is slightly worse than reinforcement learning on the CIFAR-10 dataset. Although there is value in using reinforcement learning, the gap is smaller than what is found in the original work of [71]. This result suggests that 1) the NASNet search space is well-constructed such that random search can perform reasonably well and 2) random search is a difficult baseline to beat. We will compare reinforcement learning against random search in Section 4.4.

4. Experiments and Results

In this section, we describe our experiments with the method described above to learn convolutional cells. In summary, all architecture searches are performed using the CIFAR-10 classification task [31]. The controller RNN was trained using Proximal Policy Optimization (PPO) [51] 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 500 GPUs.

The result of this search process over 4 days yields several candidate convolutional cells. We note that this search procedure is almost $7\times$ faster than previous approaches [71] that took 28 days.[1] Additionally, we demonstrate below that the resulting architecture is superior in accuracy.

Figure 4 shows a diagram of the top performing Normal Cell and Reduction Cell. Note the prevalence of separable convolutions and the number of branches compared with competing architectures [53, 59, 20, 60, 58]. 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 searches *NASNet-A*, *NASNet-B* and *NASNet-C*.

We demonstrate the utility of the convolutional cells 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 cells, 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. After selecting the number of initial filters, we use a common heuristic to double the number of filters whenever the stride is 2. Finally, we define a simple notation, e.g., 4 @ 64, to indicate these two parameters in all networks, where 4 and 64 indicate the number of cell repeats and the number of filters in the penultimate layer of the network, respectively.

For complete details of of the architecture learning algorithm and the controller system, please refer to Appendix A. Importantly, when training NASNets, we discovered *ScheduledDropPath*, a modified version of DropPath [33], to be an effective regularization method for NASNet. In DropPath [33], each path in the cell is stochastically dropped with some fixed probability during training. In our modified version, ScheduledDropPath, each path in the cell is dropped out with a probability that is *linearly increased over the course of training*. We find that DropPath does not work well for NASNets, while ScheduledDropPath significantly improves the final performance of NASNets in both CIFAR and ImageNet experiments.

[1] In particular, we note that previous architecture search [71] used 800 GPUs for 28 days resulting in 22,400 GPU-hours. The method in this paper uses 500 GPUs across 4 days resulting in 2,000 GPU-hours. The former effort used Nvidia K40 GPUs, whereas the current efforts used faster NVidia P100s. Discounting the fact that the we use faster hardware, we estimate that the current procedure is roughly about $7\times$ more efficient.

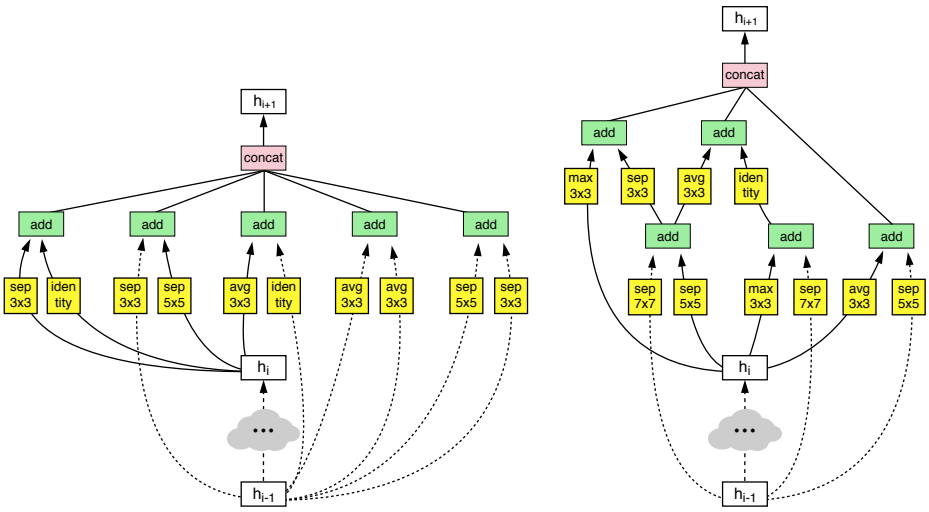

Figure 4. Architecture of the best convolutional cells (NASNet-A) with $B = 5$ blocks identified with CIFAR-10 . The input (white) is the hidden state from previous activations (or input image). The output (pink) is the result of a concatenation operation across all resulting branches. Each convolutional 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.

4.1. Results on CIFAR-10 Image Classification

For the task of image classification with CIFAR-10, we set $N = 4$ or 6 (Figure 2). The test accuracies of the best architectures are reported in Table 1 along with other state-of-the-art models. As can be seen from the Table, a large NASNet-A model with cutout data augmentation [12] achieves a state-of-the-art error rate of 2.40% (averaged across 5 runs), which is slightly better than the previous best record of 2.56% by [12]. The best single run from our model achieves 2.19% error rate.

4.2. Results on ImageNet Image Classification

We performed several sets of experiments on ImageNet with the best convolutional cells learned from CIFAR-10. We emphasize that we merely transfer the architectures from CIFAR-10 but train all ImageNet models weights from scratch.

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 with different experiments scaled in computational demand to create models that are roughly on par in computational cost with Inception-v2 [29], Inception-v3 [60] and PolyNet [69]. We show 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 [24, 70].

Note we do not have residual connections between 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 [60], 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 cells exceeds the predictive performance of the corresponding hand-designed model. Importantly, the largest model achieves a new state-of-the-art performance for ImageNet (82.7%) based on single, non-ensembled predictions, surpassing previous best published result by $\sim$1.2% [8]. Among the unpublished works, our model is on par with the best reported result of 82.7% [25], while having significantly fewer floating point operations. Figure 5 shows a complete summary of our results in comparison with other published 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 [24] and ShuffleNet [70] provide state-of-the-art results obtaining 70.6% and 70.9% accuracy, respectively on

model	depth	# params	error rate (%)
DenseNet ($L = 40, k = 12$) [26]	40	1.0M	5.24
DenseNet($L = 100, k = 12$) [26]	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) [26]	100	27.2M	3.74
DenseNet-BC ($L = 100, k = 40$) [26]	190	25.6M	3.46
Shake-Shake 26 2x32d [18]	26	2.9M	3.55
Shake-Shake 26 2x96d [18]	26	26.2M	2.86
Shake-Shake 26 2x96d + cutout [12]	26	26.2M	2.56
NAS v3 [71]	39	7.1M	4.47
NAS v3 [71]	39	37.4M	3.65
NASNet-A (6 @ 768)	-	3.3M	3.41
NASNet-A (6 @ 768) + cutout	-	3.3M	2.65
NASNet-A (7 @ 2304)	-	27.6M	2.97
NASNet-A (7 @ 2304) + cutout	-	27.6M	2.40
NASNet-B (4 @ 1152)	-	2.6M	3.73
NASNet-C (4 @ 640)	-	3.1M	3.59

Table 1. Performance of Neural Architecture Search and other state-of-the-art models on CIFAR-10. All results for NASNet are the mean accuracy across 5 runs.

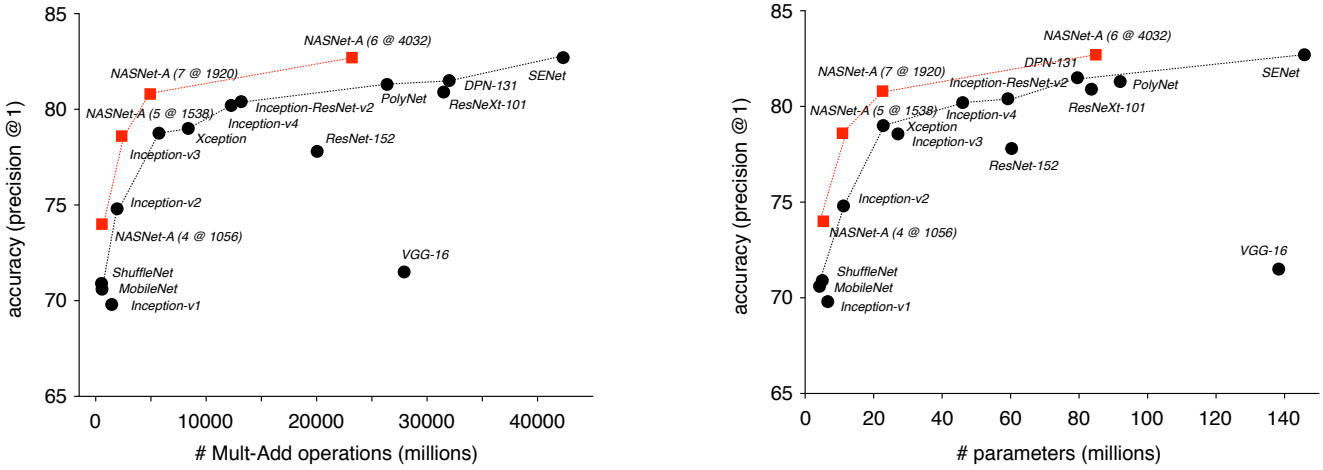

Figure 5. Accuracy versus computational demand (left) and number of parameters (right) across top performing published CNN architectures on ImageNet 2012 ILSVRC challenge prediction task. Computational demand is measured in the number of floating-point multiply-add operations to process a single image. Black circles indicate previously published results and red squares highlight our proposed models.

224x224 images using ∼550M multliply-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.

4.3. Improved features for object detection

Image classification networks provide generic image features that may be transferred to other computer vision problems [13]. One of the most important problems is the spatial localization of objects within an image. To further validate the performance of the family of NASNet-A networks, we test whether object detection systems derived from NASNet-A lead to improvements in object detection [28].

To address this question, we plug in the family of NASNet-A networks pretrained on ImageNet into the Faster-RCNN object detection pipeline [47] using an open-source software platform [28]. We retrain the resulting object detection pipeline on the combined COCO training plus validation dataset excluding 8,000 mini-validation images.

Model	image size	# parameters	Mult-Adds	Top 1 Acc. (%)	Top 5 Acc. (%)
Inception V2 [29]	224×224	11.2 M	1.94 B	74.8	92.2
NASNet-A (5 @ 1538)	**299×299**	**10.9 M**	**2.35 B**	**78.6**	**94.2**
Inception V3 [60]	299×299	23.8 M	5.72 B	78.8	94.4
Xception [9]	299×299	22.8 M	8.38 B	79.0	94.5
Inception ResNet V2 [58]	299×299	55.8 M	13.2 B	80.1	95.1
NASNet-A (7 @ 1920)	**299×299**	**22.6 M**	**4.93 B**	**80.8**	**95.3**
ResNeXt-101 (64 x 4d) [68]	320×320	83.6 M	31.5 B	80.9	95.6
PolyNet [69]	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
SENet [25]	**320×320**	**145.8 M**	**42.3 B**	**82.7**	**96.2**
NASNet-A (6 @ 4032)	**331×331**	**88.9 M**	**23.8 B**	**82.7**	**96.2**

Table 2. Performance of architecture search and other published state-of-the-art models on ImageNet classification. Mult-Adds indicate the number of composite multiply-accumulate operations for a single image. Note that the composite multiple-accumulate operations are calculated for the image size reported in the table. Model size for [25] calculated from open-source implementation.

Model	# parameters	Mult-Adds	Top 1 Acc. (%)	Top 5 Acc. (%)
Inception V1 [59]	6.6M	1,448 M	69.8 †	89.9
MobileNet-224 [24]	4.2 M	569 M	70.6	89.5
ShuffleNet (2x) [70]	∼ 5M	524 M	70.9	89.8
NASNet-A (4 @ 1056)	**5.3 M**	**564 M**	**74.0**	**91.6**
NASNet-B (4 @ 1536)	5.3M	488 M	72.8	91.3
NASNet-C (3 @ 960)	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 use 224x224 images. † indicates top-1 accuracy not reported in [59] but from open-source implementation.

Model	resolution	mAP (mini-val)	mAP (test-dev)
MobileNet-224 [24]	600 × 600	19.8%	-
ShuffleNet (2x) [70]	600 × 600	24.5%†	-
NASNet-A (4 @ 1056)	600 × 600	**29.6%**	-
ResNet-101-FPN [36]	800 (short side)	-	36.2%
Inception-ResNet-v2 (G-RMI) [28]	600 × 600	35.7%	35.6%
Inception-ResNet-v2 (TDM) [52]	600 × 1000	37.3%	36.8%
NASNet-A (6 @ 4032)	800 × 800	41.3%	40.7%
NASNet-A (6 @ 4032)	1200 × 1200	**43.2%**	**43.1%**
ResNet-101-FPN (RetinaNet) [37]	800 (short side)	-	39.1%

Table 4. Object detection performance on COCO on *mini-val* and *test-dev* datasets across a variety of image featurizations. All results are with the Faster-RCNN object detection framework [47] from a single crop of an image. Top rows highlight mobile-optimized image featurizations, while bottom rows indicate computationally heavy image featurizations geared towards achieving best results. All *mini-val* results employ the same 8K subset of validation images in [28].

We perform single model evaluation using 300-500 RPN proposals per image. In other words, we only pass a single image through a single network. We evaluate the model on the COCO *mini-val* [28] and *test-dev* dataset and report the mean average precision (mAP) as computed with the standard COCO metric library [38]. We perform a simple search over learning rate schedules to identify the best possible model. Finally, we examine the behavior of two object detection systems employing the best performing NASNet-A image featurization (NASNet-A, 6 @ 4032) as well as the image featurization geared towards mobile platforms (NASNet-A, 4 @ 1056).

For the mobile-optimized network, our resulting system achieves a mAP of 29.6% – exceeding previous mobile-optimized networks that employ Faster-RCNN by over 5.0% (Table 4). For the best NASNet network, our resulting

network operating on images of the same spatial resolution (800×800) achieves mAP = 40.7%, exceeding equivalent object detection systems based off lesser performing image featurization (i.e. Inception-ResNet-v2) by 4.0% [28, 52] (see Appendix for example detections on images and side-by-side comparisons). Finally, increasing the spatial resolution of the input image results in the best reported, single model result for object detection of 43.1%, surpassing the best previous best by over 4.0% [37].[2] These results provide further evidence that NASNet provides superior, generic image features that may be transferred across other computer vision tasks. Figure 10 and Figure 11 in Appendix C show four examples of object detection results produced by NASNet-A with the Faster-RCNN framework.

4.4. Efficiency of architecture search methods

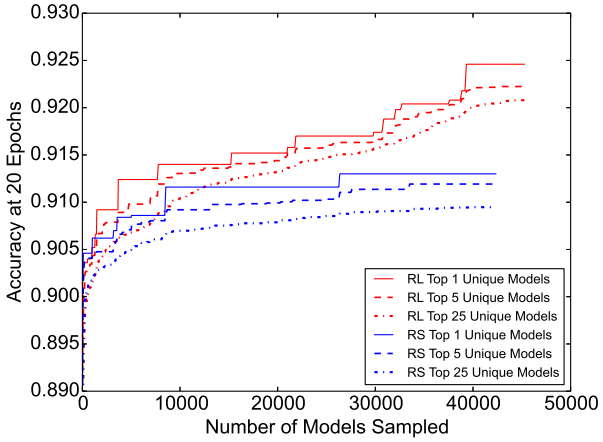

Figure 6. Comparing the efficiency of random search (RS) to reinforcement learning (RL) for learning neural architectures. The x-axis measures the total number of model architectures sampled, and the y-axis is the validation performance on CIFAR-10 after 20 epochs of training.

Though what search method to use is not the focus of the paper, an open question is how effective is the reinforcement learning search method. In this section, we study the effectiveness of reinforcement learning for architecture search on the CIFAR-10 image classification problem and compare it to brute-force random search (considered to be a very strong baseline for black-box optimization [5]) given an equivalent amount of computational resources.

Figure 6 shows the performance of reinforcement learning (RL) and random search (RS) as more model architectures are sampled. Note that the best model identified with RL is significantly better than the best model found by RS by over 1% as measured by on CIFAR-10. Additionally, RL finds an entire range of models that are of superior quality to random search. We observe this in the mean performance of the top-5 and top-25 models identified in RL versus RS. We take these results to indicate that although RS may provide a viable search strategy, RL finds better architectures in the NASNet search space.

[2] A primary advance in the best reported object detection system is the introduction of a novel loss [37]. Pairing this loss with NASNet-A image featurization may lead to even further performance gains. Additionally, performance gains are achievable through ensembling multiple inferences across multiple model instances and image crops (e.g., [28]).

5. Conclusion

In this work, we demonstrate how to learn scalable, convolutional cells from data that transfer to multiple image classification tasks. The learned architecture is quite flexible as it may be scaled in terms of computational cost and parameters to easily address a variety of problems. In all cases, the accuracy of the resulting model exceeds all human-designed models – ranging from models designed for mobile applications to computationally-heavy models designed to achieve the most accurate results.

The key insight in our 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 both CIFAR-10 and ImageNet datasets with less computational demand than human-designed architectures [60, 29, 69]. The ImageNet results are particularly important because many state-of-the-art computer vision problems (e.g., object detection [28], face detection [50], image localization [63]) derive image features or architectures from ImageNet classification models. For instance, we find that image features obtained from ImageNet used in combination with the Faster-RCNN framework achieves state-of-the-art object detection results. Finally, we demonstrate that we can use the resulting learned architecture to perform ImageNet classification with reduced computational budgets that outperform streamlined architectures targeted to mobile and embedded platforms [24, 70].

References

[1] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

[2] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[3] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*, 2016.

[4] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Neural Information Processing Systems*, 2011.

[5] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 2012.

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

[7] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz. Revisiting distributed synchronous sgd. In *International Conference on Learning Representations Workshop Track*, 2016.

[8] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng. Dual path networks. *arXiv preprint arXiv:1707.01083*, 2017.

[9] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[10] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *International Conference on Learning Representations*, 2016.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2009.

[12] T. DeVries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[13] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning*, volume 32, pages 647–655, 2014.

[14] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[15] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017.

[16] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 2008.

[17] K. Fukushima. A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, page 93202, 1980.

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

[19] D. Ha, A. Dai, and Q. V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017.

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[21] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, 2016.

[22] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

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

[24] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[25] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

[26] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[27] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, 2016.

[28] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[29] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Learning Representations*, 2015.

[30] R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Learning Representations*, 2015.

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

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing System*, 2012.

[33] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[35] K. Li and J. Malik. Learning to optimize neural nets. *arXiv preprint arXiv:1703.00441*, 2017.

[36] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[37] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017.

[38] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

[39] I. Loshchilov and F. Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

[40] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter. Towards automatically-tuned neural networks. In *Proceedings of the 2016 Workshop on Automatic Machine Learning*, pages 58–65, 2016.

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

[42] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, A. Navruzyan, N. Duffy, and B. Hodjat. Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*, 2017.

[43] R. Negrinho and G. Gordon. DeepArchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*, 2017.

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

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

[46] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, 2017.

[47] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[48] S. Saxena and J. Verbeek. Convolutional neural fabrics. In *Advances in Neural Information Processing Systems*, 2016.

[49] T. Schaul and J. Schmidhuber. Metalearning. *Scholarpedia*, 2010.

[50] F. Schroff, D. Kalenichenko, and J. 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.

[51] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[52] A. Shrivastava, R. Sukthankar, J. Malik, and A. Gupta. Beyond skip connections: Top-down modulation for object detection. *arXiv preprint arXiv:1612.06851*, 2016.

[53] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[54] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Neural Information Processing Systems*, 2012.

[55] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Ali, R. P. Adams, et al. Scalable Bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, 2015.

[56] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[57] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 2009.

[58] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. Inception-v4, Inception-Resnet and the impact of residual connections on learning. In *International Conference on Learning Representations Workshop Track*, 2016.

[59] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[60] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception architecture for computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[61] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[62] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[63] T. Weyand, I. Kostrikov, and J. Philbin. Planet-photo geolocation with convolutional neural networks. In *European Conference on Computer Vision*, 2016.

[64] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. de Freitas, and J. Sohl-Dickstein. Learned optimizers that scale and generalize. *arXiv preprint arXiv:1703.04813*, 2017.

[65] D. Wierstra, F. J. Gomez, and J. Schmidhuber. Modeling systems with internal state using evolino. In *The Genetic and Evolutionary Computation Conference*, 2005.

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

[67] L. Xie and A. Yuille. Genetic CNN. *arXiv preprint arXiv:1703.01513*, 2017.

[68] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[69] X. Zhang, Z. Li, C. C. Loy, and D. Lin. Polynet: A pursuit of structural diversity in very deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[70] X. Zhang, X. Zhou, L. Mengxiao, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.

[71] B. Zoph and Q. 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 [31] 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 [22] 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 [71], who used the REINFORCE rule [66] to update the controller, we employ Proximal Policy Optimization (PPO) [51] with learning rate 0.00035 because training with PPO is faster and more stable. To encourage exploration we also use an entropy penalty with a weight of 0.00001. In our implementation, the baseline function is 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 workqueue 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 PPO and then sample another batch of architectures that go into the global workqueue. 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. Details of architecture search space

We performed preliminary experiments to identify a flexible, expressive search space for neural architectures that learn effectively. Generally, our strategy for preliminary experiments involved small-scale explorations to identify how to run large-scale architecture search.

- All convolutions employ ReLU nonlinearity. Experiments with ELU nonlinearity [10] showed minimal benefit.
- To ensure that the shapes always match in convolutional cells, 1x1 convolutions are inserted as necessary.
- Unlike [24], all depthwise separable convolution do not employ Batch Normalization and/or a ReLU between the depthwise and pointwise operations.
- All convolutions followed an ordering of ReLU, convolution operation and Batch Normalization following [21].
- Whenever a separable convolution is selected as an operation by the model architecture, the separable convolution is applied twice to the hidden state. We found this empirically to improve overall performance.

A.5. Training with ScheduledDropPath

We performed several experiments with various stochastic regularization methods. Naively applying dropout [56] across convolutional filters degraded performance. However, we discovered a new technique called *ScheduledDropPath*, a modified version of DropPath [33], that works well in regularizing NASNets. In DropPath, we stochastically drop out each path (i.e., edge with a yellow box in Figure 4) in the cell with some fixed probability. This is similar to [27] and [69] 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 also found that DropPath alone does not help NASNet training much, but DropPath with *linearly increasing the probability of dropping out a path over the course of training* significantly improves the final performance for both CIFAR and ImageNet experiments. We name this method ScheduledDropPath.

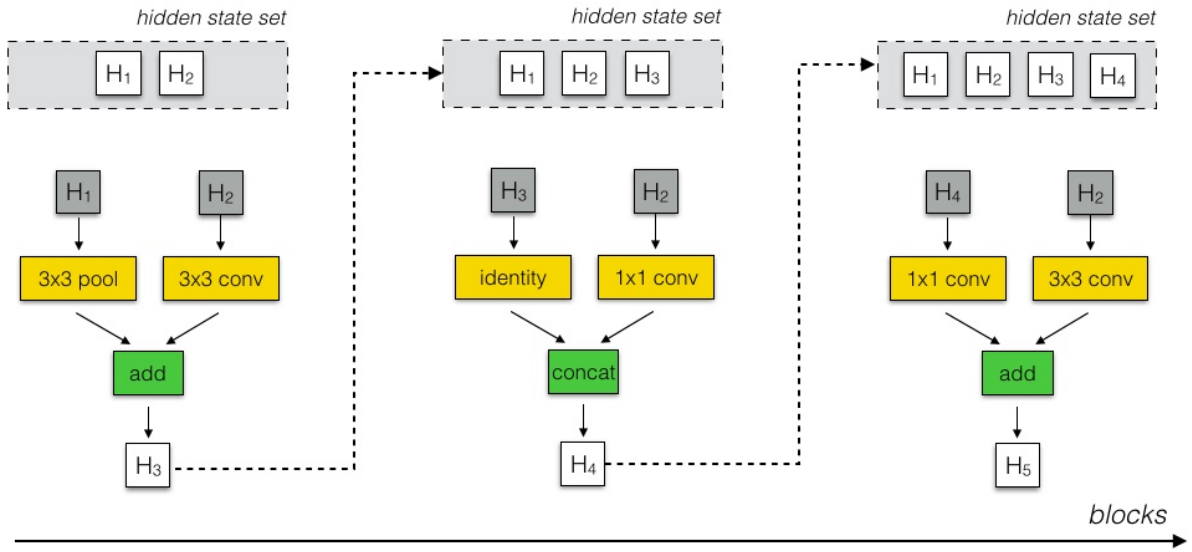

Figure 7. Schematic diagram of the NASNet search space. Network motifs are constructed recursively in stages termed blocks. Each block consists of the controller selecting a pair of hidden states (dark gray), operations to perform on those hidden states (yellow) and a combination operation (green). The resulting hidden state is retained in the set of potential hidden states to be selected on subsequent blocks.

A.6. Training of CIFAR models

All of our CIFAR models use a single period cosine decay as in [39, 18]. 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 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.7. 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 1,000 classes [11]. Overall our training and testing procedures are almost identical to [60]. ImageNet models are trained and evaluated on 299x299 or 331x331 images using the same data augmentation procedures as described previously [60]. 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 [60]. 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 [60]. 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 [60]. 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 8 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 [61] to be predicted as two of the combination operations within the cell, along with addition or concatenation.

For NASNet-C (Figure 9), 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.

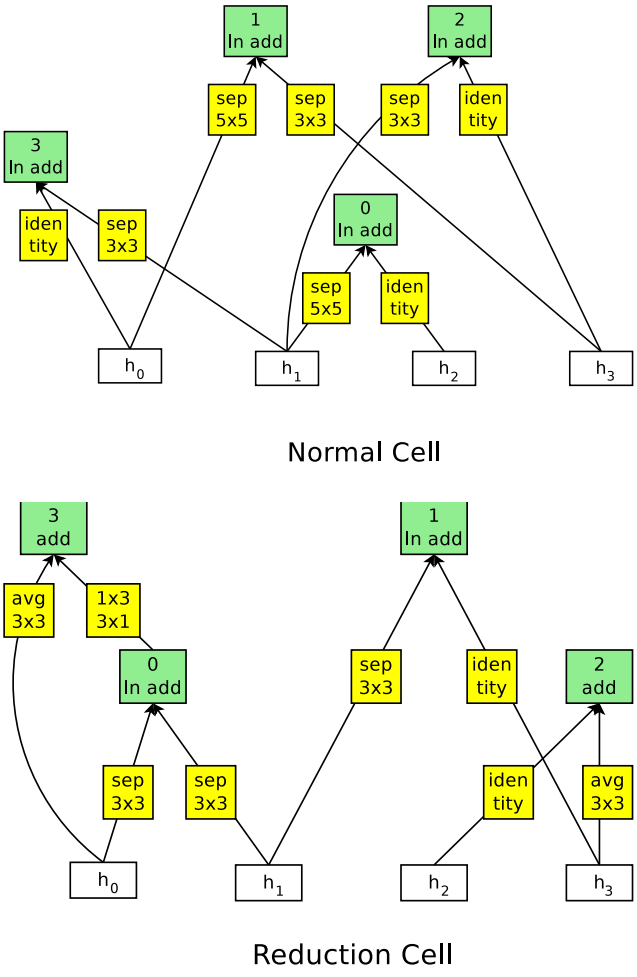

Figure 8. Architecture of NASNet-B convolutional cell with $B = 4$ blocks identified with CIFAR-10. The input (white) is the hidden state from previous activations (or input image). Each convolutional cell is the result of B blocks. A single block is corresponds to two primitive operations (yellow) and 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.

C. Example object detection results

Finally, we will present examples of object detection results on the COCO dataset in Figure 10 and Figure 11. As can be seen from the figures, NASNet-A featurization works well with Faster-RCNN and gives accurate localization of objects.

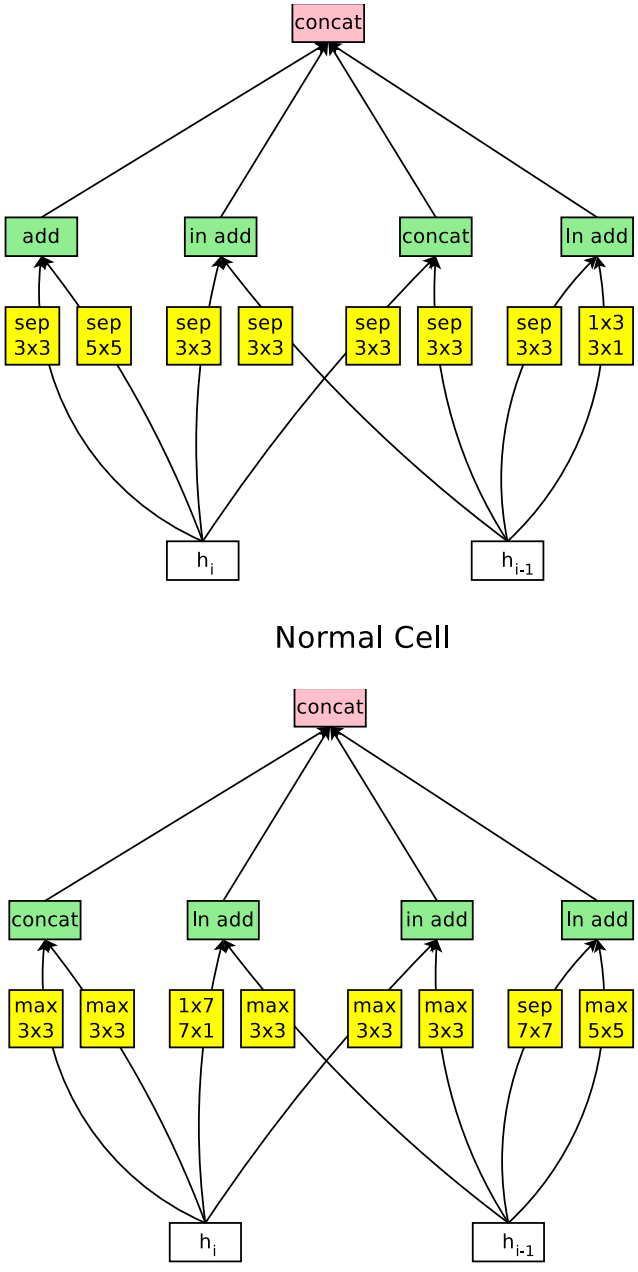

Figure 9. Architecture of NASNet-C convolutional cell with $B = 4$ blocks identified with CIFAR-10. The input (white) is the hidden state from previous activations (or input image). The output (pink) is the result of a concatenation operation across all resulting branches. Each convolutional cell is the result of B blocks. A single block corresponds to two primitive operations (yellow) and a combination operation (green).

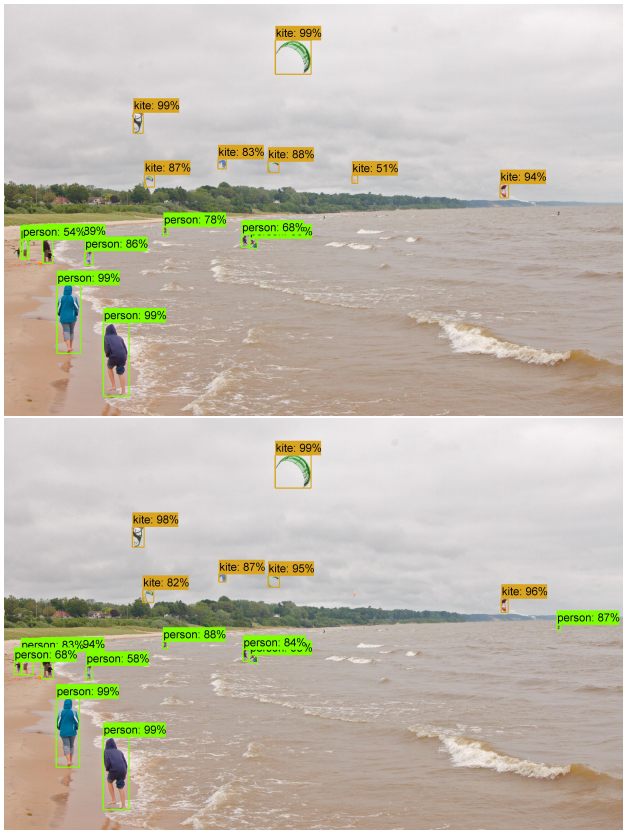

Figure 10. Example detections showing improvements of object detection over previous state-of-the-art model for Faster-RCNN with Inception-ResNet-v2 featurization [28] (top) and NASNet-A featurization (bottom).

Figure 11. Example detections of best performing NASNet-A featurization with Faster-RCNN trained on COCO dataset. Top and middle images courtesy of `http://wikipedia.org`. Bottom image courtesy of Jonathan Huang