SMASH: One-Shot Model Architecture Search through HyperNetworks

Andrew Brock, Theodore Lim, & J.M. Ritchie
School of Engineering and Physical Sciences
Heriot-Watt University
Edinburgh, UK
`{ajb5, t.lim, j.m.ritchie}@hw.ac.uk`

Nick Weston
Renishaw plc
Research Ave, North
Edinburgh, UK
`Nick.Weston@renishaw.com`

Abstract

Designing architectures for deep neural networks requires expert knowledge and substantial computation time. We propose a technique to accelerate architecture selection by learning an auxiliary HyperNet that generates the weights of a main model conditioned on that model's architecture. By comparing the relative validation performance of networks with HyperNet-generated weights, we can effectively search over a wide range of architectures at the cost of a single training run. To facilitate this search, we develop a flexible mechanism based on memory read-writes that allows us to define a wide range of network connectivity patterns, with ResNet, DenseNet, and FractalNet blocks as special cases. We validate our method (SMASH) on several benchmarks, achieving competitive performance with similarly-sized hand-designed networks.

1 Introduction

The high performance of deep neural nets is tempered by the cost of extensive engineering and validation to find the best architecture for a given problem. In this work, we propose to bypass the expensive procedure of fully training candidate models by instead training an auxiliary model, a HyperNet [8], to dynamically generate the weights of a main model with variable architecture. Though these generated weights are worse than freely learned weights for a fixed architecture, we leverage the observation [13] that the relative performance of different networks early in training (i.e. some distance from the eventual optimum) often provides a meaningful indication of performance at optimality. By comparing validation performance for a set of architectures using generated weights, we can approximately rank numerous architectures at the cost of a single training run.

To facilitate this search, we develop a flexible scheme based on memory read-writes that allows us to define a diverse range of architectures, with ResNets [9], FractalNets [12] and DenseNets [10] as special cases. We validate our one-Shot Model Architecture Search through Hypernetworks (SMASH) for Convolutional Neural Networks (CNN) on several benchmarks, achieving competitive performance with similarly-sized hand-designed networks.

2 Related Work

Modern practical methods for optimizing hyperparameters rely on random search [2] or Bayesian Optimization (BO) [17, 18], treating the model performance as a black box. While successful, these methods require multiple training runs for evaluation (even when starting with a good initial model) and, in the case of BO, are not typically used to specify variable-length settings such as the connectivity and structure of the model under consideration. Relatedly, bandit-based methods [13]

NIPS 2017 Meta-Learning workshop, Long Beach, CA, USA.

provide a framework for efficiently exploring the hyperparameter space by employing an adaptive early-stopping strategy, allocating more resources to models which show promise early in training.

Evolutionary techniques [6, 19–21] offer a flexible approach for discovering variegated models from trivial initial conditions, but often struggle to scale to deep neural nets where the search space is vast, even with enormous compute power [15]. Reinforcement learning methods [1, 23] have been used to train an agent to generate network definitions using policy gradients. These methods start from trivial architectures and discover models that achieve very high performance, but can require twelve to fifteen thousand full training runs to arrive at a solution.

3 One-Shot Model Architecture Search through HyperNetworks

In SMASH, our goal is to rank a set of neural network configurations relative to one another based on each configuration's validation performance, which we accomplish using weights generated by an auxiliary network. At each training step, we randomly sample a network architecture, generate the weights for that architecture using a HyperNet, and train the entire system end-to-end through backpropagation. When the model is finished training, we sample a number of random architectures and evaluate their performance on a validation set, using weights generated by the HyperNet. We then select the architecture with the best estimated validation performance and train its weights normally.

3.1 Defining Variable Network Configurations

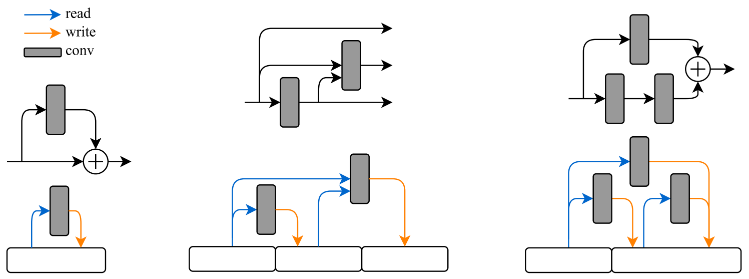

Figure 1: Memory-Bank representations of ResNet, DenseNet, and FractalNet blocks.

In order to explore a broad range of architectures with variable depth, connectivity patterns, layer sizes and beyond, we require a flexible mechanism for defining such architectures, which we can also easily encode into a conditioning vector for the HyperNet. To this end, we espouse a "memory-bank" view of feed-forward networks.

Rather than viewing a network as a series of operations applied to a forward-propagating signal, we view a network as having a set of memory banks (initially tensors filled with zeros) which it can read and write. Each layer is thus an operation that reads data from a subset of memory, modifies the data, and writes the result to another subset of memory. For a single-branch architecture, the network has one large memory bank it reads and overwrites (or, for a ResNet, adds to) at each op, while branching architectures have more complex patterns.

Our base network structure consists of a sequence of "blocks," each with its own set of memory banks at successively halved spatial resolutions. When sampling an architecture, the number of banks and the number of channels per bank are randomly sampled at each block. When defining each layer within a block, we randomly select the read-write pattern and the definition of the op to be performed on the read data. When reading from multiple banks we concatenate the read tensors along the channel axis, and when writing to banks we add to the tensors currently in each bank.

Each op comprises a 1x1 convolution (reducing the number of incoming channels), followed by a variable number of convolutions interleaved with nonlinearities following the 2-branch pattern of [7]. We randomly select which of the four convolutions are active, along with their filter size, dilation factor, number of groups, and the number of output units (i.e. the layer size). We employ a simplified version of WeightNorm [16] where we divide the entirety of each generated 1x1 filter by its Euclidean norm, and only use BatchNorm [11] in the transition and output layers. The weights for the 1x1 convolution are generated by the HyperNet as described in Section 3.2, while the other convolutions are normally learned parameters, shared across all ops within a block.

3.2 Learning to map architectures to weights

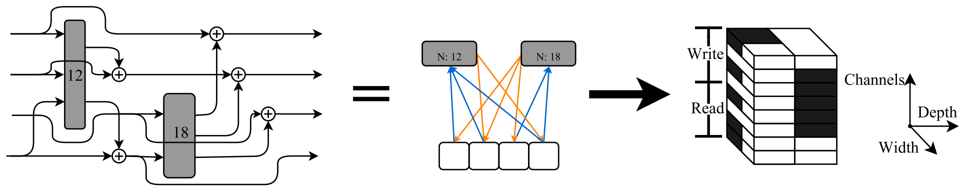

Figure 2: An unrolled graph, its equivalent memory-bank representation, and its encoded embedding.

A HyperNet [8] is a neural net used to parameterize the weights of another network, the main network. For a Static HyperNet with parameters H, the main network weights W are some function (e.g. a multilayer perceptron) of a learned embedding z, such that the number of learned weights is typically smaller than the full number of weights for the main network. For a Dynamic HyperNet, the weights W are generated conditioned on the network input x, or, for recurrent networks, on the current input x_t and the previous hidden state h_{t-1}.

We propose a variant of a Dynamic Hypernet which generates the weights W based on a tensor encoding of the main network architecture c. Our goal is to learn a mapping $W = H(c)$ that is reasonably close to the optimal W for any given c, such that we can rank each architecture based on the validation error using HyperNet-generated weights. We thus adopt a scheme for the layout of c to enable sampling of architectures with wildly variable topologies, compatibility with the toolbox available in standard libraries, and to make c's dimensions as interpretable as possible.

Our HyperNet is fully convolutional, such that the dimensionality of the output tensor W varies with the dimensionality of the input c, which we make a 4D tensor of the standard format BCHW, with a batch size of 1 so that no output elements are wholly independent. This allows us to vary the depth and width of the main network by increasing the height or width of c. Under this scheme, every slice of the spatial dimensions of W corresponds to a specific subset of c. Information describing the op that uses that W subset is embedded in the channel dimension of the corresponding c slice.

4 Experiments

We apply SMASH to several datasets, both for the purposes of benchmarking against other techniques, and to investigate the behavior of SMASH networks. Principally, we are interested in determining whether the validation error of a network using SMASH-generated weights (the "SMASH score") correlates with the validation of a normally trained network, and if so, the conditions under which the correlation holds. We are also interested in the transferability of the learned architectures to new datasets and domains, and how this relates to normal (weight-wise) transfer learning. Our code[1] is written in PyTorch [14] to leverage dynamic graphs.

4.1 Testing the SMASH correlation

First, we train a SMASH network for 300 epochs on CIFAR-100, using a standard annealing schedule [10], then sample 250 random architectures and evaluate their SMASH score on a held-out validation set formed of 5,000 random examples from the original training set. We then sort the architectures by their SMASH score and select every 5th architecture for full training and evaluation, using an accelerated training schedule of 30 epochs.

Under these conditions, we observe a correlation (Figure 3) between the SMASH score and the true validation performance, suggesting that SMASH-generated weights can be used to rapidly compare architectures. It is critical not to overstate this claim; this test is arguably a single datapoint indicating that the correlation holds in this scenario, but neither guarantees the correlation's generality nor

[1]https://github.com/ajbrock/SMASH

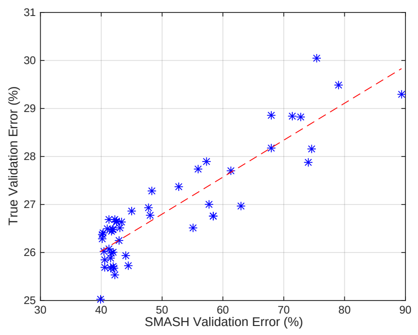

Figure 3: True error and SMASH validation error for 50 random architectures on CIFAR-100.

implies the range of conditions for which it will hold. The expense of running this experiment prohibits a satisfactory number of repeat trials, so we instead construct different experiments.

For our second experiment, we train a low-budget SMASH network (to permit more rapid testing) with a much smaller HyperNet relative to the main network (though still the standard ratio of generated to freely learned weights). We expect the decreased capacity HyperNet to be less able to learn to generate good weights for the full range of architectures, and for the correlation between SMASH score and true performance to therefore be weak or nonexistent, and indeed observe said breakdown.

For our third experiment, we examine whether or not the HyperNet has learned to take into account the architecture definition in c, or whether it ignores c and naively generates an unconditional subspace of weights that happen to work well. We "trick" the HyperNet by sampling one architecture, but asking it to generate the weights for a different architecture by corrupting the encoding tensor c (e.g. by shuffling the dilation values). For a given architecture, we find that SMASH validation performance is consistently highest when using the correct encoding tensor, suggesting that the HyperNet has indeed learned a passable mapping from architecture to weights.

Following this, we posit that if the HyperNet learns a meaningful mapping $W = H(c)$, then the classification error $E = f(W, x) = f(H(c), x)$ can be backpropagated to find $\frac{dE}{dc}$, providing an approximate measure of the error with respect to the architecture itself. If this holds true, then perturbing the architecture according to the $\frac{dE}{dc}$ vector (within the constraints of our scheme) should allow us to guide the architecture search through a gradient descent-like procedure. Our preliminary naive tests with this idea underperformed random architectural perturbations.

4.2 Transfer Learning and Benchmarking

Models with weights initially learned on one large dataset frequently outperform models trained from scratch on a smaller dataset; it follows that architectures might display the same behavior. We evaluate the performance of the best-found architecture from CIFAR-100 (with weights trained from scratch) on STL-10 [5], ImageNet32x32 [4], and ModelNet10. Our network achieves competitive performance on CIFAR-10 and 100 (4.03% error and 20.06% error, respectively, vs a 2.86% and 15.85% error SOTA [7]), STL-10 (17.54% error vs a 15.43% Wide ResNet 28-10 [22] baseline), ImageNet32x32 (38.62% top-1 error vs 40.96% top-1 WRN28-10), and ModelNet10 (93.28% accuracy compared to a 93.61% Inception-ResNet [3] trained with additional data).

5 Conclusion

In this work, we explore a technique for accelerating architecture selection by learning a model over network parameters, conditioned on the network's parametric form. We introduce a flexible scheme for defining network connectivity patterns and generating network weights for highly variable architectures. Our results demonstrate a correlation between performance using suboptimal weights generated by the auxiliary model and performance using fully-trained weights, indicating that we can efficiently explore the architectural design space through this proxy model. Our method achieves competitive, though not state-of-the-art performance on several datasets.

References

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

[2] J. Bergstra and Y. Bengio. Random search for hyperparameter optimization. In *JMLR 2012*.

[3] A. Brock, T. Lim, J.M. Ritchie, and N. Weston. Generative and discriminative voxel modeling with convolutional neural networks. 3D Deep Learning Workshop at NIPS 2016 arXiv: 1608.04236, 2016.

[4] P. Chrabaszcz, I. Loshchilov, and F. Hutter. A downsampled variant of imagennet as an altetrnative to the cifar datasets. arXiv Preprint arXiv: 1707.08819, 2017.

[5] A. Coates, H. Lee, and A.Y. Ng. n analysis of single layer networks in unsupervised feature learning. In *AISTATS 2011*.

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

[7] X. Gastaldi. Shake-shake regularization of 3-branch residual networks. ICLR 2017 Workshop, 2017.

[8] D. Ha, A. Dai, and Q. Le. Hypernetworks. In *ICLR 2017*.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR 2016*.

[10] G. Huang, Z. Liu, K.Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *CVPR 2017*, 2017.

[11] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML 2015*.

[12] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *ICLR 2017*.

[13] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *ICLR 2017*.

[14] A. Paszke, S. Gross, and S. Chintala. Pytorch. github.com/pytorch/pytorch, 2017.

[15] E. Real, S. Moore, A. Selle, S. Saxena, Y.L. Suematsu, Q. Le, and A. Kurakin. Large-scale evolution of image classifiers. arXiv Preprint arXiv: 1703.01041, 2017.

[16] T. Salimans and D.P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *NIPS 2016*.

[17] J. Snoek, H. Larochelle, and R.P. Adams. Practical bayesian optimization of machine learning algorithms. In *NIPS 2012*, .

[18] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M.M.A. Patwary, Prabhat, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *ICML 2015*, .

[19] K.O. Stanley, D.B. D'Ambrosio, and J Gauci. A hypercube-based encoding for evolving large-scale neural networks. In *Artificial Life, 15(2):185-212, 2009*.

[20] M. Suganuma, S. Shirakawa, and T. Nagao. A genetic programming approach to designing convolutional neural network architectures. In *GECCO 2017*.

[21] D. Wierstra, F.J. Gomez, and J. Schmidhuber. Modeling systems with internal state using evolino. In *GECCO 2005*.

[22] S. Zagoruyko and N. Komodakis. Wide residual networks. arXiv Preprint arXiv: 1605.07146, 2016.

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