Fast Neural Architecture Construction using EnvelopeNets

Purushotham Kamath
Cisco Systems
pukamath@cisco.com

Abhishek Singh
Cisco Systems
abhishs8@cisco.com

Debo Dutta
Cisco Systems
dedutta@cisco.com

Abstract

Fast Neural Architecture Construction is a method to construct deep network architectures by pruning and expansion of a base network. The construction method is analogous to proposed theories of human brain ontogenesis. Instead of constructing a network architecture based on validation accuracy, a single scalar, our method directly compares utility of different network blocks and constructs networks with close to state of the art accuracy, in < 1 GPU day, faster than most of the current neural architecture search methods.

1 Introduction

Neural Architecture Construction (NAC) is a method to design neural network architectures that mirrors theories on the ontogenesis of neurons in the brain. Brain development is believed to consist of neurogenesis (29), where the neural structure initially develops, gradually followed by apoptosis (9), where neural cells are eliminated, introduction of more neurons by hippocampal neurogenesis (11) and synaptic pruning (17), where synapses are eliminated. The NAC algorithm consists of analogous steps run in iterations: model initialization with a prior (neurogenesis), a truncated training cycle, pruning filters (apoptosis), adding new cells (hippocampal neurogenesis) and pruning of skip connections (synaptic pruning). Artificial neurogenesis (21) has been previously studied as, among others, a method for continuous learning (7) in neural networks.

In recent years, several architecture search methods have been proposed using techniques such as evolutionary algorithms and reinforcement learning. These methods have found neural network architectures that outperform bespoke architectures. However, most of these algorithms iterate through a number of intermediate networks, comparing their accuracy, before discovering a final network. The time needed to discover the final network is limited by the need to run a full training and evaluation cycle on each intermediate network generated, resulting in large search times. Recent work (16) includes network design methods (Neuroevolution (12), network transformation (5) AmoebaNet (31; 32), Bayesian Optimization (20), MetaQNN (3), Genetic CNN (34; 10)), cell design methods (NasNet (38; 37; 23), BlockQNN (36), Dutta et. al. (8)) and optimization based methods (DARTs (25), NAO (26)) Some of the search algorithms address the long search times in different ways: ENAS (30) and network transformation (5) use parameter sharing across iterations of generation. Others (BlockQNN (36), SMASH (4), Hypernets (14)) avoid full training by using early stop methods or predictive weight generation. Other methods reduce the search space or use more efficient search methods (DeepArchitect (27), hierarchical representations (24)).

In contrast, NAC speeds up the construction process because the pruning and expansion can be done without needing to wait for a full training cycle to complete. A filter's utility is calculated based on statistics obtained from their featuremaps obtained during the training. These statistics reach a state where the utility of different filters within a network can be evaluated. Experimentally, we find that the time needed for filters to reach this state, is much less than the time needed for the accuracy of

2nd Workshop on Meta-Learning at NeurIPS 2018, Montréal, Canada.

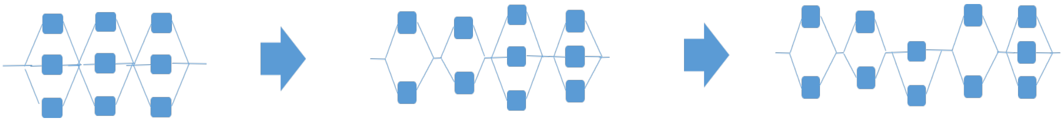

Figure 1: Neural Architecture Construction (NAC) using EnvelopeNets

a network to converge *i.e.* the time needed for an accurate comparison of the performance of two networks, resulting in a speedup in the construction time.

The pruning and expansion algorithm fits in well with intuition from previous work that indicates different layers of the network play different roles in the overall classification task (35; 28). The layers closer to the head of the network extract gross features (edges, boundaries, shapes) while deeper layers compose these into more abstract features (such as facial features). Further, (13) indicates that each stage of a network iteratively refines its estimates of the same features. In InceptionNet (6), the 3 parallel paths with different filters were shown to extract features at different levels. After training, it was found that for most layers, one of the paths dominated the others, indicating that one path was primarily activated at each layer. *EnvelopeNet* construction exploits these properties in the iterative construction of a network.

2 Construction using EnvelopeNets

ALGORITHM 1: Neural Architecture Construction

```
// Neurogenesis:  Set network prior
$network \leftarrow hyperparams \rightarrow envelopenet$
while $iterations < hyperparams \rightarrow max\_iterations$ do
    // Learning:  Truncated training
    $filter\_stats \leftarrow short\_train(network)$
    for $stage$ in $network$ do
        $sorted\_filters \leftarrow sort(filter\_stats, stage)$
        // Apoptosis:  Prune sorted filters per stage, subject to constraints:
           Do not prune a cell if it is the last cell in a layer and limit
           number of pruned filters per stage
        $prune\_filters(stage, sorted\_filters)$
        // Synaptic pruning:  Prune skip connections
        $prune\_skip\_connections(network)$
        // Hippocampal neurogenesis:  Add envelopecell to the tail of stage
        $network \leftarrow add\_cell(network, stage, hyperparams \rightarrow envelopecell)$
    end
    $iterations \leftarrow iterations + 1$
end
return $network$
```

The construction algorithm starts from a base network called an *EnvelopeNet*, composed of several *EnvelopeCells* and evolves it to generate the final network as shown in Figure 1. An EnvelopeCell is a set of M convolution blocks connected in parallel with their output concatenated. *E.g.* one of the EnvelopeCells used in this work has 6 convolution blocks connected in parallel: 1x1 , 3x3 , 3x3 separable , 5x5 , 5x5 separable and 7x7 separable convolutions. This is a search space commonly used in neural architecture search methods. Each block consists of a convolution filter, a Relu unit and a batch normalization.

The EnvelopeNet consists of several *stages*, each with several *layers*. Each layer has a single EnvelopeCell. The stages are separated by *wideners*. A widener (also called reduction cell) is a maxpool unit and 3x3 convolution filter connected in parallel which downsamples the image dimensions by a factor of two and doubles the number of channel. The EnvelopeNet has DenseNet (15) style skip connections with depthwise concatenation on all inputs followed by a 1x1 convolution filter to control the number of output channels. In addition the network has two other types of cells: a stem (initial cell) for the network that increases the input channel width C for the EnvelopeNet and a

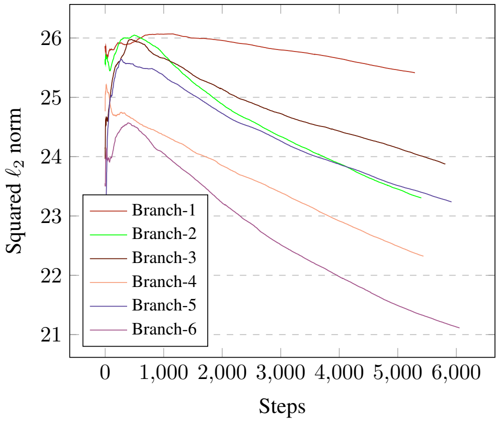

Figure 2: Squared ℓ_2 norm of individual filters normalized by the featuremap size at different branches in a network vs. training steps for a 128/10-1-1-1/6 EnvelopeNet.

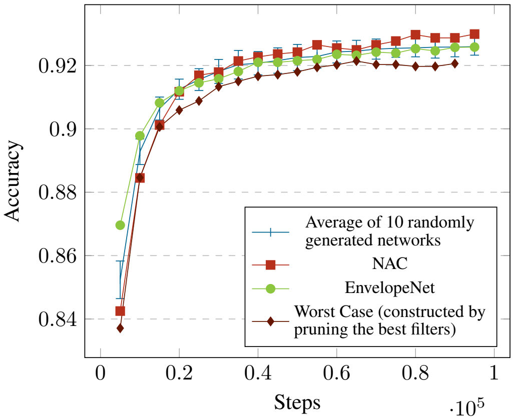

Figure 3: Accuracy vs. training iterations for the EnvelopeNet, the Constructed Net (NAC) and random networks on the CIFAR-10 data set (100 epochs (100K steps))

classification cell consisting of an average pooling block, a fully connected layer with dropout and softmax. The stem and classification blocks are placed at the head and tail of the network respectively. We refer to an Envelope network using the notation, C/n_1-n_2-n_3.../M where C is the number of input channels to the EnvelopeNet (output of the stem cell), n_i is the number of layers in stage i and M is the number of filters.

The algorithm restructures the EnvelopeNet in a series of iterations of pruning and expansion shown in Algorithm 1. As a result, EnvelopeNet network narrows and deepens and the dense skip connections get pruned (this also helps reduce the parameter count as the network gets deeper). Pruning of filters and expansion is done by stage *i.e.* only filters within a stage are compared, and each stage is expanded separately. This is motivated by studies that indicate successive layers iteratively refine their estimates of the same features (13). Skip connections are pruned by assigning an initial random weight to each skip connection. These weights are trained along with the network. During the pruning phase, s half the skip connections are pruned (the ones with the lowest weights). Although the algorithm searches across different stages in parallel, the overall time to find the network is lower or comparable with the existing methods which only search for a single cell. The EnvelopeNet provides a strong prior that helps the algorithm restructure the network rather than use resources for the discovery of a base network structure.

It can be shown (19) that for a simple single stage network, with some assumptions, the relative MSE of a pruned network relative to its generating EnvelopeNet is minimized by pruning the filter with the lowest sum of the squared ℓ_2-norm of a filter featuremap. The algorithm uses the squared ℓ_2 norm to compare filters within a stage (between wideners, where the featuremap dimensions and number of channels is the same for all filters) and identifies the filters to prune. Note that all filter's statistics are compared to design the next candidate network, while other neural architecture search methods use a single scalar, accuracy, to decide the fitness of a candidate model. This internal view of the candidate architecture makes the construction faster and more efficient than methods where a controller has to try different combinations and use a single scalar, accuracy, to select the best.

The improvement in construction time compared to conventional neural architecture search depends on the ratio of the time needed to extract featuremap statistics to a full training cycle f and the number of iterations N needed. In our experiments f was 10 epochs/100 epochs = 0.1 and N was between 5 to 10.

3 Results

We evaluate the proposed method on the task of image classification using CIFAR-10 (22) and ImageNet (33) datasets. The experiments were run using AMLA (18) and all code/results/hyperparameters are open sourced (1).

Network	Dataset	Params	Test error (%)	Search time (GPU days)
NAS-v3	CIFAR10	37.4M	3.65	1800
Block-QNN	CIFAR10	39.8M	3.54	96
AmoebaNet-B	CIFAR-10	34.9M	2.13	3150
PNAS	CIFAR10	**3.2M**	3.41	225
ENAS	CIFAR-10	4.6M	3.54	0.45
DARTS	CIFAR-10	4.6M	2.76	4
NAO	CIFAR-10	128M	**2.07**	200
NAC (128/7-6-2/6)	CIFAR10	10M	3.33	**0.25**
NASNet-A	Imagenet	4.9M	8.4	1800
AmoebaNet-A	ImageNet	6.4M	**7.6**	3150
DARTS	ImageNet	**4.7M**	8.7	4
NAC (64/7-6-2/6)	Imagenet	9.9M	11.77	**0.25**

Table 1: Accuracy, search time and number of parameters for NAC construction using EnvelopeNets vs. other methods. State of the art numbers are indicated in bold. The NAC experiments were run using single Nvidia V100 GPU. The search time reported here is the sum of the time required for the algorithm to output the final architecture, but does not include the time for the training of the final network. Please note that the ImageNet accuracy for NAC is preliminary. Current number is reported.

Figure 2 shows the running squared ℓ_2 norm of a single filter in the EnvelopeNet (other filters show similar though not identical behavior). After roughly 10 epochs, the squared ℓ_2 norms show a reasonable separation between each other, although they have not yet converged. In comparison typical training of a network takes > 100 epochs. Note that we do not use parameter sharing across iterations, which could further reduce generation time, though it could suffer neural brainwashing (2).

Figure 3 shows performance for a 128/15-6-4-4/6 constructed network generated from a 128/10-1-1-1/16 EnvelopeNet after 5 iterations. 10 equivalent random networks were generated by fixing the depth of the stages equal to the stages in constructed network and adding the same number of blocks, chosen randomly at each stage, subject to a minimum of one block per layer.

The worst case network was constructed using the construction algorithm, except it pruned the best performing filters and did not use skip pruning. The constructed network consistently outperforms the original EnvelopeNet, the worst case network and the sample average (with standard deviations) of 10 randomly generated networks. Roughly half of the randomly generated networks had lower performance than the EnvelopeNet, indicating that structure of the generated network is responsible for some of the gain, and that the entire gains do not come from deepening or more parameters.

Table 1 shows the performance of a network generated from a 128/2-2-2/6 Envelope with 23.8M parameters, run for 5 iterations to generate a 128/7-6-2/6 Constructed Network for CIFAR10. We use the same network to train on ImageNet and compare along with other methods, with state of the art performance indicated.

The primary limitation of the method is that the gains from structure appear to reduce when the network parameters increase. When the number of parameters is extremely large, *e.g.* if we use a classification block with several fully connected blocks or as the number of stages (number of output channels) increases. In this regime gain in the accuracy of network comes from a larger number of parameters rather than intelligent structure design.

4 Conclusions

Neural networks exhibit properties that allow simple heuristic based construction techniques using statistics derived during training. We have exploited one of these properties to design networks faster than a search method that evaluates the accuracy of networks. The generated networks show close to state of the art performance and outperform randomly and handcrafted networks with equivalent number of parameters and blocks.

References

[1] 2018. AMLA. (2018). https://github.com/ciscoai/amla,https://github.com/ciscoai/envelopenets

[2] Anonymous. 2019. Overcoming Neural Brainwashing. In *Submitted to International Conference on Learning Representations*. https://openreview.net/forum?id=r1fE3sAcYQ under review.

[3] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2016. Designing Neural Network Architectures using Reinforcement Learning. *CoRR* abs/1611.02167 (2016). http://arxiv.org/abs/1611.02167

[4] Andrew Brock, Theodore Lim, James M. Ritchie, and Nick Weston. 2017. SMASH: One-Shot Model Architecture Search through HyperNetworks. *CoRR* abs/1708.05344 (2017).

[5] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. 2017. Efficient Architecture Search by Network Transformation. *CoRR* abs/1707.04873 (2017). arXiv:1707.04873 http://arxiv.org/abs/1707.04873

[6] François Chollet. 2016. Xception: Deep Learning with Depthwise Separable Convolutions. *CoRR* abs/1610.02357 (2016). http://arxiv.org/abs/1610.02357

[7] Timothy J. Draelos, Nadine E. Miner, Christopher C. Lamb, Craig M. Vineyard, Kristofor D. Carlson, Conrad D. James, and James B. Aimone. 2016. Neurogenesis Deep Learning. *CoRR* abs/1612.03770 (2016). http://arxiv.org/abs/1612.03770

[8] Jayanta K Dutta, Jiayi Liu, Unmesh Kurup, and Mohak Shah. 2018. Effective Building Block Design for Deep Convolutional Neural Networks using Search. *CoRR* abs/1801.08577 (2018). arXiv:1801.08577 http://arxiv.org/abs/1801.08577

[9] Susan Elmore. 2007. Apoptosis: A Review of Programmed Cell Death. *Toxicologic Pathology* 35, 4 (2007), 495–516. https://doi.org/10.1080/01926230701320337 arXiv:https://doi.org/10.1080/01926230701320337 PMID: 17562483.

[10] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. 2017. Simple And Efficient Architecture Search for Convolutional Neural Networks. *CoRR* abs/1711.04528 (2017). arXiv:1711.04528 http://arxiv.org/abs/1711.04528

[11] Peter S. Eriksson, Ekaterina Perfilieva, Thomas Björk-Eriksson, Ann-Marie Alborn, Claes Nordborg, Daniel A. Peterson, and Fred H. Gage. 1998. Neurogenesis in the adult human hippocampus. *Nature Medicine* 4 (01 Nov 1998), 1313 EP –. http://dx.doi.org/10.1038/3305 Article.

[12] D. Floreano, P. Dürr, and C. Mattiussi. 2008. Neuroevolution: from architectures to learning. Evolutionary Intelligence (2008). https://doi.org/10.1007/s12065-007-0002-4

[13] Klaus Greff, Rupesh Kumar Srivastava, and Jürgen Schmidhuber. 2016. Highway and Residual Networks learn Unrolled Iterative Estimation. *CoRR* abs/1612.07771 (2016). http://arxiv.org/abs/1612.07771

[14] David Ha, Andrew M. Dai, and Quoc V. Le. 2016. HyperNetworks. *CoRR* abs/1609.09106 (2016). arXiv:1609.09106 http://arxiv.org/abs/1609.09106

[15] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. 2016. Densely Connected Convolutional Networks. *CoRR* abs/1608.06993 (2016). arXiv:1608.06993 http://arxiv.org/abs/1608.06993

[16] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. 2018. AutoML: Methods, Systems, Challenges. *https://www.automl.org/book/* (2018). https://www.automl.org/book/

[17] Giorgio M. Innocenti. 1995. Exuberant development of connections, and its possible permissive role in cortical evolution. *Trends in Neurosciences* 18, 9 (1995), 397 – 402. https://doi.org/10.1016/0166-2236(95)93936-R

[18] Purushotham Kamath, Abhishek Singh, and Debo Dutta. 2018. AMLA: an AutoML frAmework for Neural Networks. *AutoML 2018 at ICML 2018* (2018). http://

[19] Purushotham Kamath, Abhishek Singh, and Debo Dutta. 2018. Neural Architecture Construction using EnvelopeNets. *CoRR* abs/1803.06744 (2018). arXiv:1803.06744 http://arxiv.org/abs/1803.06744

[20] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabás Póczos, and Eric Xing. 2018. Neural Architecture Search with Bayesian Optimisation and Optimal Transport. *CoRR* abs/1802.07191 (2018). http://arxiv.org/abs/1802.07191

[21] Taras Kowaliw, Nicolas Bredèche, Sylvain Chevallier, and René Doursat. 2016. Artificial Neurogenesis : An Introduction and Selective Review.

[22] A Krizhevsky and G Hinton. 2009. Learning multiple layers of features from tiny images. 1 (01 2009).

[23] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. 2017. Progressive Neural Architecture Search. *CoRR*. arXiv:cs.CV/1712.00559

[24] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2017. Hierarchical Representations for Efficient Architecture Search. *CoRR* abs/1711.00436 (2017). http://arxiv.org/abs/1711.00436

[25] H. Liu, K. Simonyan, and Y. Yang. 2018. DARTS: Differentiable Architecture Search. *ArXiv e-prints* (June 2018). arXiv:1806.09055

[26] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu. 2018. Neural Architecture Optimization. *ArXiv e-prints* (Aug. 2018). arXiv:1808.07233

[27] Renato Negrinho and Geoff Gordon. 2017. DeepArchitect: Automatically Designing and Training Deep Architectures. *CoRR* abs/1704.08792 (2017). http://arxiv.org/abs/1704.08792

[28] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. 2018. The Building Blocks of Interpretability. *Distill* (2018). https://doi.org/10.23915/distill.00010 https://distill.pub/2018/building-blocks.

[29] Judith TML Paridaen and Wieland B. Huttner. 2014. Neurogenesis during development of the vertebrate central nervous system. *EMBO Rep* 15, 4 (17 Apr 2014), 351–364. https://doi.org/10.1002/embr.201438447 24639559[pmid].

[30] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameter Sharing. *CoRR* abs/1802.03268 (2018). arXiv:1802.03268 http://arxiv.org/abs/1802.03268

[31] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2018. Regularized Evolution for Image Classifier Architecture Search. *CoRR* abs/1802.01548 (2018). http://arxiv.org/abs/1802.01548

[32] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Quoc V. Le, and Alex Kurakin. 2017. Large-Scale Evolution of Image Classifiers. *CoRR* abs/1703.01041. arXiv:1703.01041 http://arxiv.org/abs/1703.01041

[33] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[34] Lingxi Xie and Alan L. Yuille. 2017. Genetic CNN. *CoRR* abs/1703.01513 (2017). http://arxiv.org/abs/1703.01513

[35] Matthew D. Zeiler and Rob Fergus. 2013. Visualizing and Understanding Convolutional Networks. *CoRR* abs/1311.2901 (2013). arXiv:1311.2901 http://arxiv.org/abs/1311.2901

[36] Zhao Zhong, Junjie Yan2, Wei Wu2, Jing Shao2, and Cheng-Lin Liu. 2018. Practical Block-wise Neural Network Architecture Generation. *CoRR* abs/1708.05552 (2018). arXiv:1708.05552 `http://arxiv.org/abs/1708.05552`

[37] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. In *ICLR 2017*. `https://arxiv.org/abs/1611.01578`

[38] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2017. Learning Transferable Architectures for Scalable Image Recognition. *CoRR* abs/1707.07012. arXiv:1707.07012 `http://arxiv.org/abs/1707.07012`