4-1-2019

System Architecting Approach for Designing Deep Learning Models

Ram Deepak Gottapu

Cihan H. Dagli
Missouri University of Science and Technology, dagli@mst.edu

Recommended Citation

R. D. Gottapu and C. H. Dagli, "System Architecting Approach for Designing Deep Learning Models," *Procedia Computer Science*, vol. 153, pp. 37-44, Elsevier B.V., Apr 2019.
The definitive version is available at https://doi.org/10.1016/j.procs.2019.05.053

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 153 (2019) 37–44

Procedia Computer Science

www.elsevier.com/locate/procedia

17th Annual Conference on Systems Engineering Research (CSER)

System Architecting Approach for Designing Deep Learning Models

Ram Deepak Gottapu[a]*, Cihan H Dagli[b]

[a]*Missouri University of Science & Technology, Rolla, MO, 65409, USA*

Abstract

Deep Learning (DL) models have proven to be very effective in solving many challenging problems, especially, those related to computer vision, text, and speech. However, the design of such models is challenging because of the vast search space and computational complexity that needs to be explored. Our goal in this paper is to reduce the human effort required to design architectures by using a system architecture development process that allows the exploration of large design space by automating certain model construction, alternative generation, and assessment. The proposed framework is generic and targeted at all deep learning architectures that can be expressed by logical models with certain numeric properties. The implementation of the proposed approach is presented, along with the test results achieved on CIFAR-10 dataset using a convolutional neural network (CNN). We show that the architecture generated by our approach achieves 5.23% error rate with only 1.2M parameters, which shows the capability to design high performing architectures.

Keywords: Deep learning (DL); convolutional neural network (CNN); system architecting

* Corresponding author. Tel.: 937-768-6542.
E-mail address: rgrk6@mst.edu

10.1016/j.procs.2019.05.053

1. Introduction

The exploration of neural network architecture has been an important part of research in the field of deep learning. For example, in case of computer vision architectures, starting with LeNet[1] in 2012, many architectures [2,3,4,5,6,7,8] were designed to improve the accuracy which resulted in gradual increase of the number of trainable parameters. This became significant especially after the architectures started crossing 100-layer barrier in Inception[9], ResNet[10] and highway networks[11] These architectures use hundreds of millions of trainable parameters and are computationally very expensive. As a result, the focus of research in designing CNN's became not just improving the accuracy but also to reduce the number of trainable parameters. In DenseNet[3], the authors presented an architecture which outperformed all the previous architectures while utilizing only 1M parameters which is significantly lower than any other architectures producing similar accuracy. Following that, the authors in NasNet[12,13], designed a search space and used evolutionary approaches to find architectures that further improved the accuracy with only 3M parameters. This shows that, by using innovative search spaces, it is possible to come up with efficient architectures that have high accuracy with minimum trainable parameters. It holds true for other deep learning models like long short-term memory (LSTMs) and recurrent neural networks (RNNs). However, all these approaches use a search space that only resembles the inception model (increasing width of architecture) [9] while models like ResNet[10] show that deep architectures are also capable of providing high accuracies.

Our approach is explicitly designed to explore architectures based on a search space that creates deep architectures rather than wide architectures. In addition to that, it also provides a framework that allows the user to modify the search space based on the available computation. The framework also allows the user to choose hyper-parameters and layers that can be used within the search space. The implementation starts with the user choosing an overall objective function. Once the objective function is defined, categorical information is required to define the architecture with:

- Capabilities: Functionality provided by the systems/layers in the architecture (Table 1).
- Feasible interfaces: Stipulates which layer may interface with one another and are also represented as binary values.

Following the modeling of the objective function, we use a genetic algorithm to optimize the objective. Applying genetic algorithms or any other search approaches to design complete architectures is computationally expensive. Therefore, we use our modeling approach to design a block instead of complete architecture. Since it is already established that stacking up convolutional layers improves the accuracies [13], we can now stack the optimized blocks to generate architectures that can give state-of-the-art accuracies. This approach of learning only parts of architecture was proved to be successful in [12,13].

Our experiments were mainly focused on CIFAR-10 dataset where we achieved multiple top performing blocks. The architectures are then generated from these blocks by stacking them on top of each other using a transition block. Our typical model achieved an error rate of 5.23 on CIFAR-10 dataset. Throughout our experiments, the evolution happens on the hyper-parameters of convolutional layers such as filter height, filter width, number of filters, length of the block, and connections. Since these values are limited to only a few choices, the goal is to find the best combination of choices that give the best accuracy on the block.

2. Literature Review

System architecting approaches, applied in design, analysis, and optimization have flourished in various domain-specific disciplines [20,21,22]. Our approach aims to use these capabilities in the field of deep learning to reduce the human effort required to design the architectures by considering each layer as a system and finding optimal connectivity between input and output.

The idea of using evolution to learn neural networks dates back to 2001[23]. Attempts were made to design or optimize neural networks using evolutionary algorithms, reinforcement learning and other machine learning approaches [24,25]. In recent years, these approaches resurfaced as the deep learning architectures became deeper thereby increasing the search space significantly. Initial approaches include the design architectures that can overcome the man-made models [26,27,28]. The prominent feature of these attempts is to design complete architectures and optimize them based on the validation accuracies. Since it included the design of complete architecture, they are computationally expensive and also, they did not have much success when compared to the state-of-the-art models except for [29,28].

In order to reduce the computational complexity and be able to search for optimized architectures in the design space, it is more advantageous to build a novel search space that has the capability to do both. In [13], the authors showed that by using a small and efficient search space, we can build highly efficient architectures. Our approach takes inspiration from [13] and [29] to develop a search space that is capable of designing very deep architectures. By using a system architecting approach, we can have the desired search space for multiple objectives [18,19] which is more generic.

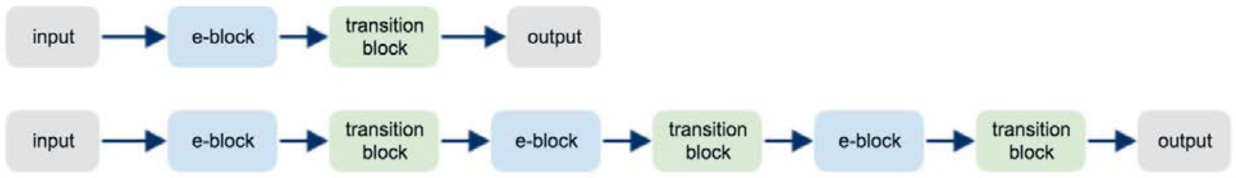

Figure 1: Pre-Determined architecture designs. The top model was used for evolution and the bottom model was used to train complete architecture.

3. Method

In this section, we first discuss the process of defining the problem and then the process of optimizing the defined problem.

3.1. Modelling the architecture:

The first step of our approach involves defining the architecture in a structured format using system engineering principles which can later be optimized using a genetic algorithm. It follows the same modeling approach defined in [18]. Any deep neural network consists of different layers which have their own role in the architecture. Therefore, we first need to identify the capabilities provided by each type of layer. Table 1 shows the layers and their corresponding capabilities.

Table 1. Table showing layers and their capabilities.

System/Layer	Linear Transformation	Non-linear Transformation	Regularization	Parameter Scaling
Convolutional layer	X	-	-	-
Pooling layer	X	-	-	-
Activation layer	-	X	-	-
Dropout	-	-	X	-
Batch-Normalization	-	-	X	X

Each architecture generated during evolution will be assessed using accuracy. However, based on the platform where the architecture is running, an additional objective can also be added such as computational cost and time. In this paper, we only used accuracy as the performance metric to focus on architecture design instead of computation. By fixing the number of systems (say 12), we can now optimize the objectives by finding optimal connectivity using different systems and their capabilities.

3.2. Evolution of designed model

After defining the properties of the model in the framework, our method makes use of evolutionary search to find an optimized architecture for a given dataset of interest. The approach starts with a random population whose fitness (objective) values are calculated. The top-n accuracies are then used to generate offspring using cross-over and mutation operations. The approach is repeated for n iterations until the objectives are stabilized.

The general structure for any deep convolutional neural networks (CNN) is a sequential repetition of identical motifs. The accuracy of the overall architecture depends not only on just the depth but also the configuration of these motifs and the connectivity among them. This has been proven in the architectures generated so far [13]. In addition to that, the inclusion of transition layers at regular intervals provides a necessary hierarchy that is useful for learning in the architecture [3]. These observations suggest that it is possible for an evolutionary algorithm to design a generic motif which can outperform the humanly designed motifs.

In our approach, the overall architecture of the deep CNN is pre-determined (see fig1) where the evolution model uses only a part of the final model. This follows the same argument of general CNN i.e. the accuracy of the model improves by replicating the basic structure. In this case, instead of replicating a simple convolution layer we do it with a more complex section of complete architecture. The complete architecture consists of an input layer followed a block repeated multiple times. Since these blocks are designed using evolutionary algorithms, we name these blocks as evolved blocks (e-blocks). Each e-block has d layers and any two e-blocks in the architecture are separated by a transition layer. The transition layers provide the necessary hierarchy required in the architecture. The transition layer, initial layer, and the final layer are manually designed thereby making the total length of architecture: $kd + (k - 1) + 2$ where k is the number of e-blocks used. Each hidden layer inside the e-block has the option of choosing its own input size, filter size, number of filers and strides. During the course of evolution, the algorithm tries to find the best choices for each layer in order to maximize the accuracy of the overall architecture.

Table 2. Table showing the measurement type for each KPP

System	Choices	Encoding
Conv	(1,3,5,7)	[(0,0),(0,1),(1,0),(1,1)]
Number of filters	(12,24,36,48)	[(0,0),(0,1),(1,0),(1,1)]
Compression	(0.5,0.65,0.75,1	[(0,0),(0,1),(1,0),(1,1)]
stride	(1,2)	(0,1)

To be able to evolve the e-block using the genetic algorithm, we encoded the complete e-block into a binary chromosome. The chromosome for an e-block has the structure shown in fig 2. With each e-block having d layers, the total length of the chromosome is given as $n_{\text{total}} d + (d(d + 1)/2)$ where n_{total} is the number of bits required to encode each choice. The first half of the equation defines the number of bits required to choose the layer configuration details while the second half of the equation defines the inputs. The inputs to each layer can be any number of inputs from previous layers concatenated along the dimensional axis. Therefore, all the skip connections will be covered in this layer. During the course of evolution, the following steps are performed for each layer in the e-block:

- Select number of filters: number of filters used for the conv layer.
- Select filter size: dimensions of each filter.
- Select stride: number of jumps performed while sliding the filters over the input.

- Select compression: The residual connections increase the number of features after each layer. Compression defines the percentage of features to be kept for the next layer.
- Select inputs between first layer h_0 to h_{t-1} which will be concatenated to give input

The algorithm finds the best performing architectures during each iteration and tries to improve them using crossover and mutation. Table 2 shows the choices for each layer (chosen based on popular literature). with their corresponding encoding. The first three choices require 2 bits to represent their 4 unique choices while the last choice requires only one bit (0/1). Therefore n_{total}=2+2+2+1 = 7. For d layers (say d =12) the total length of chromosome will be $n\_total\ d + (d(d+1)/2)$ = 162. If the first 7 bits have an encoding sequence are: [0,1,0,0,1,1,0] then the first layer will have the following configuration: 3x3 conv, 12 filters, compression of 1 from previous filters and stride of 1. More details on the concept of compression can be found in [3].

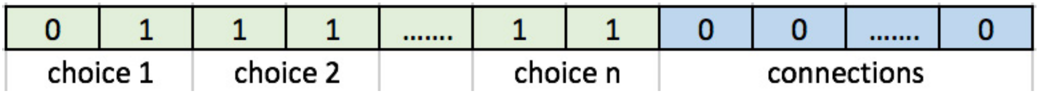

Figure 2: Chromosome for a sample architecture

The choices for each layer are chosen based on the popular literature: conv: (1,3,5,7), number of filters: (12,24,48,36), compression: (0.5,0.65,0.75,1) and stride: (1,2). Finally, in order to prevent training repeated architectures, we kept a log of all the chromosomes generated from the beginning and assigned previous fitness values to repeated architectures. We also added some randomness in the crossover to generate a few random chromosomes in each generation to prevent the algorithm being stuck in local minima. Also, in order to show the efficiency of our approach, we also used random search and compared the results with those generated using evolutions. Even though the difference is not much, we observed that the results from using evolution are slightly better. The implementation details can be found in section 4.

4. Experiments and results

In this section, we describe the experiments conducted to learn e-blocks using the method mentioned above. We used CIFAR-10 dataset to learn multiple top performing e-blocks. All our experiments were performed using an NVIDIA TITAN X GPU.

The search process took over 2 weeks using a single GPU and covered 500 architectures. We ran our approach for 50 generations with a population size of 10. The crossover and mutation operations are performed in such a way that repeated architectures are not generated.

Training conditions: Throughout the evolution process, we train each architecture using stochastic gradient descent (SGD) with a learning rate of 0.1 and batch size of 64. We also observed that running each architecture of 25 epochs gave optimal results. Data augmentation was applied on the dataset so that each architecture will learn on noisy data.

4.1. Datasets

CIFAR: The CIFAR datasets consist of colored natural images with 32×32 pixels. CIFAR-10 (C10) consists of images drawn from 10 classes. The training and test sets contain 50,000 and 10,000 images respectively, and we hold out 5,000 training images as a validation set. We adopt a standard data augmentation scheme (mirroring/shifting) that is widely used for this dataset [1,2,3,4,5,6,7,8]. For preprocessing, we normalize the data using the channel means and standard deviations. During evolution, each new architecture was trained on 45,000 images and validation accuracy was calculated on remaining 5000 images. The validation accuracy is used a fitness values to evolve the architectures. For the final run, we use all 10,000 test images and report the final test error at the end of training.

4.2. Results on CIFAR-10 dataset

For the task of image classification on CIFAR datasets, we used three e-blocks separated by a transition layer (see Fig2). 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, an architecture using e-block@1 and e-block@2 with data augmentation achieves a performance that is similar to state-of-the-art models.

Table :3 Results showing the performance of our approach with respect to state of the art models

Method	Depth	Params (M)	CIFAR-10 error percentage
Network in Network [6]	-	-	8.81
All CNN [7]	-	-	7.25
Deeply Supervised Net [5]	-	-	7.97
Fractal Net [4]	21	38.6	5.22
ResNet [10]	110	1.7	6.61
Wide ResNet [8]	16	11	4.81
ResNet (pre-activation) [2]	164	1.7	5.46
DenseNet [3]	100	0.8	4.51
e-block@1	100	1.2	5.23
e-block@2	100	2.8	4.61

5. Discussion

5.1. Feasible Architectures

During the search process (evolution), there is a possibility to end up with architectures that are not feasible. For example, the dimension of output became less than 1 due to too many pooling layers. Such architectures are allocated a fitness value of -1 by default to prevent the generation of such architectures.

5.2. Limitations

Even though our approach is capable of producing deep and efficient architectures, the size of each e-block is pre-determined. In our experiments, we used the size of 12, however, if we need to experiment with different size e-block after each transition layer, the entire search process has to be repeated.

6. Conclusion

In this paper, we showed the possibility of using system architecting methodologies for designing deep learning models. The approach allows the user to define their own objectives and find the best suitable deep learning model using a pre-determined structure. Note that we did not use the latest approaches for the architecture search as our main intention is not to improve the accuracy but to show the possibility of using system architecting for deep learning.

7. Acknowledgements

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

8. References

[1] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.

[2] K. He, X. Zhang, S. Ren, J. Sun, Identity mappings in deep residual networks, in: European conference on computer vision, Springer, 2016, pp. 630–645.

[3] G. Huang, Z. Liu, L. Van Der Maaten, K. Q. Weinberger, Densely connected convolutional networks., in: CVPR, Vol. 1, 2017, p. 3.

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

[5] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, Z. Tu, Deeply-supervised nets, in: Artificial Intelli- gence and Statistics, 2015, pp. 562–570.

[6] M. Lin, Q. Chen, S. Yan, Network in network, arXiv preprint arXiv:1312.4400.

[7] J. T. Springenberg, A. Dosovitskiy, T. Brox, M. Riedmiller, Striving for simplicity: The all convolutional net, arXiv preprint arXiv:1412.6806.

[8] S. Zagoruyko, N. Komodakis, Wide residual networks, arXiv preprint arXiv:1605.07146.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1–9.

[10] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[11] R. K. Srivastava, K. Greff, J. Schmidhuber, Training very deep networks, in: Advances in neural information processing systems, 2015, pp. 2377–2385.

[12] B. Zoph, Q. V. Le, Neural architecture search with reinforcement learning, arXiv preprint arXiv:1611.01578.

[13] B. Zoph, V. Vasudevan, J. Shlens, Q. V. Le, Learning transferable architectures for scalable image recognition, arXiv preprint arXiv:1707.07012 2 (6).

[14] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, W. Banzhaf, Nsga- net: A multi-objective genetic algorithm for neural architecture search, arXiv preprint arXiv:1810.03522.

[15] M. Tan, B. Chen, R. Pang, V. Vasudevan, Q. V. Le, Mnasnet: Platform-aware neural archi- tecture search for mobile, arXiv preprint arXiv:1807.11626.

[16] T. Elsken, J. H. Metzen, F. Hutter, Multi-objective architecture search for cnns, arXiv preprint arXiv:1804.09081.

[17] C.-H. Hsu, S.-H. Chang, D.-C. Juan, J.-Y. Pan, Y.-T. Chen, W. Wei, S.-C. Chang, Monas: Multi-objective neural architecture search using reinforcement learning, arXiv preprint arXiv:1806.10332.

[18] S. Agarwal, L. E. Pape, C. H. Dagli, N. K. Ergin, D. Enke, A. Gosavi, R. Qin, D. Konur, R. Wang, R. D. Gottapu, Flexible and intelligent learning architectures for sos (fila-sos): Ar- chitectural evolution in systems-of-systems, Procedia Computer Science 44 (2015) 76–85.

[19] C. H. Dagli, D. L. Enke, N. Ergin, D. Konur, R. Qin, A. Gosavi, R. Wang, I. Pape, S. Agarwal, R. D. Gottapu, et al., Flexible and intelligent learning architectures for sos (fila-sos), Center for Systems and Software Engineering, University of Southern California IN 20.

[20] J. M. Parker, Applying a system of systems approach for improved transportation, SAPI EN. S. Surveys and Perspectives Integrating Environment and Society (3.2).

[21] E. Curry, System of systems information interoperability using a linked dataspace, in: System of Systems Engineering (SoSE), 2012 7th International Conference on, IEEE, 2012, pp. 101– 106.

[22] E. Kremers, P. Viejo, O. Barambones, J. G. de Durana, A complex systems modelling ap- proach for decentralised simulation of electrical microgrids, in: 2010 15th IEEE International Conference on Engineering of Complex Computer Systems, IEEE, 2010, pp. 302–311.

[23] S. Hochreiter, A. S. Younger, P. R. Conwell, Learning to learn using gradient descent, in: International Conference on Artificial Neural Networks, Springer, 2001, pp. 87–94.

[24] K. O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evolutionary computation 10 (2) (2002) 99–127.

[25] J. K. Pugh, K. O. Stanley, Evolving multimodal controllers with hyperneat, in: Proceedings of the 15th annual conference on Genetic and evolutionary computation, ACM, 2013, pp. 735–742.

[26] S. Saxena, J. Verbeek, Convolutional neural fabrics, in: Advances in Neural Information Pro- cessing Systems, 2016, pp. 4053–4061.

[27] T. Schaul, J. Schmidhuber, Metalearning, Scholarpedia 5 (6) (2010) 4650.

[28] B. Baker, O. Gupta, N. Naik, R. Raskar, Designing neural network architectures using rein- forcement learning, arXiv preprint arXiv:1611.02167.

[29] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, A. Kurakin, Large-scale evolution of image classifiers, arXiv preprint arXiv:1703.01041.