13 May 2020

Efficient Architecture Search for Deep Neural Networks

Ram Deepak Gottapu

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

Recommended Citation

R. D. Gottapu and C. H. Dagli, "Efficient Architecture Search for Deep Neural Networks," *Procedia Computer Science*, vol. 168, pp. 19-25, Elsevier B.V., May 2020.
The definitive version is available at https://doi.org/10.1016/j.procs.2020.02.246

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 168 (2020) 19–25

Procedia Computer Science

www.elsevier.com/locate/procedia

Complex Adaptive Systems Conference with Theme:
Leveraging AI and Machine Learning for Societal Challenges, CAS 2019

Efficient Architecture Search for Deep Neural Networks

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

Missouri University of Science and Technology

Abstract

This paper addresses the scalability challenge of automatic deep neural architecture search by implementing a parameter sharing approach with regularized genetic algorithm (RGE). The key idea is to use a regularized genetic algorithm (RGE) on a pre-determined template and discover a high-performance architecture by searching for the optimal chromosome. During evolution, each model corresponding to a discovered chromosome is trained for a fixed number of epochs to minimize a canonical cross-entropy loss on a given training dataset. Meanwhile, the performance of the trained model on validation dataset is used as a fitness value to perform the evolutions. Because of parameter sharing the trained weights in each generation are carried to the next, thereby reducing the GPU hours required for maximizing the validation accuracy. On the CIFAR-10 dataset, the approach finds a novel architecture that outperforms the best human-invented deep architecture (DenseNet). The CIFAR-10 model achieved a test error of 4.22% with only 0.96M parameters which is better than DenseNet of 4.51% with 0.8M parameters. On CIFAR-100 dataset, the approach was able to compose a novel architecture that achieved 20.53% test error with 3.7M parameters which is on par with 20.50% test error of wide ResNet with 36.5M parameters.

Keywords: Architecture search; Deep neural networks

1. Introduction

Automatic neural architecture design has shown the capability to discover high-performance neural networks, which are significantly better than human-designed models. Existing methods, irrespective of whether they use reinforcement learning (RL) [8,10], evolutionary algorithms (EA) [9] or gradient descent (GD) [11,12] rely on Neural Architecture Search (NAS) space, first

* Corresponding author. Tel.: +0-000-000-0000 ; fax: +0-000-000-0000 .
E-mail address: dagli@mst.edu

1877-0509
10.1016/j.procs.2020.02.246

published in [8]. The NAS space took its inspiration from inception [25], a wide-architecture that has better performance than previous simple linear architectures. [26]. In later stages, very deep architectures with skip connections such as ResNet [1-4] and DenseNet [5] achieved even higher performance that inception [25]. However, in the case of automatic architecture design, the inception style (wide-architecture) is preferred because of its transfer capability: use cells designed on a small dataset on a larger dataset as the search process on large datasets is computationally expensive. For example, initial approaches which did not use parameter sharing required as many as 800 GPUs and took approximately 24 days to find a single architecture using NAS space on the CIFAR-10 dataset. Therefore, implementing a similar approach upon a larger dataset is not feasible but by using the cells designed on CIFAR-10 inside deep architectures gave very high accuracies. [8-12]. With the use of parameter sharing in [10], the computation time on CIFAR-10 was reduced to approximately 18 hours, and the overall process used only 1 GPU to find an optimal architecture. Thus, the parameter sharing technique opened the possibility to work on architecture search for deep designs without consuming a lot of GPU hours.

The main contribution of this work is to use the search space introduced in [27] for deep architecture design and compose a novel architecture using parameter sharing, which is significantly better than the human-designed deep architectures. There are three main components to achieve this: (1) Define the encoding scheme for the search space such that its combinations correspond to different possible architectures. (2) Train the generated architectures with training data and test the performance with validation data. The fitness value for the RGE will be the accuracy of the trained model on validation data. (3) For each new architecture generated during evolution, assign its parameters with most recent trained values. If a particular parameter configuration is new, then perform initialization. Even though the architecture design in this paper is implemented using a genetic algorithm, it can also be implemented using reinforcement learning or gradient descent. This paper does not include the performance comparison between different optimization techniques as its main emphasis is to show an efficient way to conduct deep architecture search.

The experimental results show that the top performing architectures found on CIFAR-10 and CIFAR-100 datasets all have a performance that is significantly higher than best human-designed deep architectures with minimal parameters. The best design on CIFAR-10 gave a test error of 4.22% with 0.96M parameters which is better than DenseNet with 4.51% test error and 0.8M parameters. Additionally, compared to the approach used in [27], which requires approximately 60 days to compose a deep architecture, the technique presented in this paper required only 2 days to design a new architecture from scratch which is a significant improvement.

2. Related Work

The exploration of neural network architectures has played a vital role since their initial discovery [20]. With the rise in popularity of neural networks, several techniques were proposed to improve the performance of the models while keeping the parameters to a minimum [1-5, 13-18]. Recently the design of architectures has shifted from purely human-designs to a combination of human-knowledge and automatic approach called neural architecture search (NAS) [6-12, 19-24].

Optimization technique: The successful optimization techniques that composed state-of-the-art neural network designs relied on (1) reinforcement learning (2) genetic algorithm and (3) gradient descent. Even though there is no concrete evidence which can prove that these techniques can guarantee convergence, their application on NAS space was able to produce high-performing architectures [8-12]. The approach presented in this paper uses a variant of genetic algorithm called regularized genetic algorithm (RGE), first published in [9] but reinforcement learning or gradient descent approach should also work.

Search Space: Initial attempts of automatic neural network design were not able to generate architectures that are better than the human-made designs [21-22]. However, the use of heavy computation for architecture search pushed the performance of automatic architectures in recent years [6,7]. Even though the performance of the architecture improved on standard datasets, it is not feasible to implement on new datasets because of the extensive computational requirements. In [8], the NAS space was introduced to address this issue, which used a pre-determined architecture design, and the search process got reduced to finding the optimal cell. The approach was not only able to generate better architectures than their predecessors but also used much fewer GPU hours. This experiment showed that innovative search spaces are capable of reducing the computation time to some extent when compared to the full architecture search.

Parameter sharing: Even though the NAS space reduced the computation time, it is still not feasible to apply to new datasets as it required approximately 800 GPUs to compose a new architecture. In [10], the architecture search is implemented using both parameter sharing and NAS space, which reduced the number of GPUs from 800 to 1, thereby making this approach feasible to any new dataset. Even though different architectures use parameters differently, the results from [6, 10-12] established that parameters

learned for a particular model can be used for other models, with little to no modifications. Therefore, the approach in this paper uses parameter sharing with its architectures to compose high-performing architectures.

3. Method

This section describes the different techniques used in composing the high-performing architecture. Sec 3.1 describes the search space in general form, and its encoding to generate the chromosomes. Sec 3.2 then introduces the regularized genetic algorithm which maximizes the validation accuracy using the search space. Finally, sec 3.3 describes the parameter sharing process, which prevents the parent architectures from discarding their trained parameters by sharing them across the generation.

3.1. Architecture representation

The central idea of this approach is the observation that all architectures that the algorithm ends up iterating over the generations are a unique combination of chromosomes. Fig. 1 illustrates a generic chromosome structure. The length of the chromosome (n) determines the length of the block, and its values denote the operations at different layers.

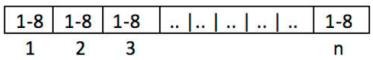

Fig. 1. Structure of the chromosome where n denotes the length of the block. Each value in the chromosome represents the operation at that layer.

The choice of operations is limited to a set of convolutions (best configurations based on previous literature) which are derived from NAS [8]. The list below shows different types of convolutions used in this paper. Since there are eight choices, the values of the chromosome will have a range between 1 and 8. An example of such representation is shown in Fig. 2 where n=3.

- 1x3 then 3x1 convolution
- 1x7 then 7x1 convolution
- 3x3 dilated convolution
- 1x1 convolution
- 3x3 convolution
- 3x3 depthwise-separable conv
- 5x5 depthwise-seperable conv
- 7x7 depthwise-separable conv

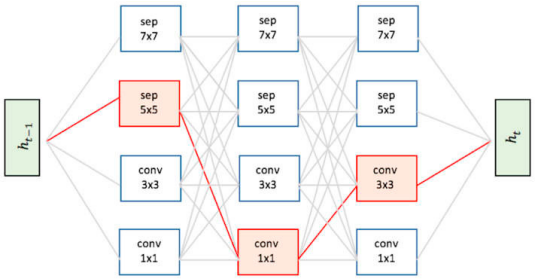

Fig. 2. A simple design chosen for a chromosome with n=3 and values [1,3,5,7]. The graph shows all the possible connections between the layers. The red lines indicate the chosen operations at each layer based on values in the chromosome.

New architectures are generated during each generation of the evolution process based on the chromosome and pre-determined template, as shown in Fig. 3. As can be seen in Fig. 3c, each layer inside the e-block is based on a set of operations apart from the chosen convolutions. The 1x1 convolution control the dimensionality of the filters. The batch normalization (BN) operations prevent the explosion of gradients during training and, the ReLu operations provide the required non-linearity inside the architecture. Also, the transition layers which have the form BN -> Conv 1x1 ->ReLu -> pooling, before e-blocks provide the necessary hierarchy.

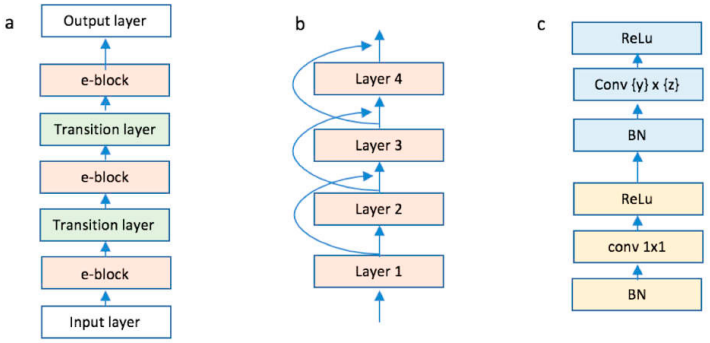

Fig. 3. a) The template used for architecture search; b) each e-block in the template has n layers which are densely connected; c) Each layer has multiple fixed operations except for one which will be chosen based on chromosome value.

3.2. Regularized genetic algorithm

Algorithm 1 gives a summary of the evolutionary algorithm used in this paper. The process starts with random initialization of population (P), which represent random initial designs. The population size (P) is always maintained and all architectures that conform to the search space described, are possible and equally likely. At each generation, the algorithm samples S random models from the population (with replacement) and selects the model with highest validation accuracy as a parent. A new architecture, called the child, is constructed from the parent by the application of a transformation called a mutation (described below). The new child architecture gets trained, evaluated, and added to the population. This process is called tournament selection, and the oldest model in the sample (S) is discarded (or killed) to maintain the population size at P.

Mutation: The mutation process changes a random value in the parent chromosome to generate a child architecture. These random modifications make the genetic algorithm traverse the search space to find an architecture that maximizes the fitness value/validation accuracy.

```
population ← empty queue                          ▷ The population.
history ← ∅                                       ▷ Will contain all models.
while |population| < P do                         ▷ Initialize population.
    model.arch ← RANDOMARCHITECTURE()
    model.accuracy ← TRAINANDEVAL(model.arch)
    add model to right of population
    add model to history
end while
while |history| < C do                            ▷ Evolve for C cycles.
    sample ← ∅                                    ▷ Parent candidates.
    while |sample| < S do
        candidate ← random element from population
                    ▷ The element stays in the population.
        add candidate to sample
    end while
    parent ← highest-accuracy model in sample
    child.arch ← MUTATE(parent.arch)
    child.accuracy ← TRAINANDEVAL(child.arch)
    add child to right of population
    add child to history
    remove dead from left of population           ▷ Oldest.
    discard dead
end while
return highest-accuracy model in history
```

Algorithm 1. Steps of Regularized Genetic Algorithm (RGE)

3.3. Parameter sharing

The architectures generated during evolutions share the parameters for training rather than initializing them for each mutation. Namely, if a layer has a matching shape to one of previously trained architecture, the weights are inherited. Therefore, the initial population will not have any inheritance, and the later generations will have it from their parents, except for the mutated layer. The mutated layer will inherit if its shape was used previously by other trained architectures. Eventually, all possible weight configurations will have an initialization, and the convergence depends on finding optimal chromosome using shared parameters. Fig. 4 shows the parameter sharing between three simple architectures. The red lines indicate the transfer of parameters whenever same convolution configurations are used. For all the new configurations, new parameters are initialized.

The evolutionary process and parameter training are alternatively implemented to maintain a balance between them. The architectures train for few epochs (say 2) and then mutated. After a few hundreds of generations (say 500), the average validation accuracy/fitness will gradually maximize. The evolution will stop either when the maximum accuracy of each generation no longer improves or when any architecture achieves the state-of-the-art accuracy. Since there is no possibility to discard any parameters, the overall search process will be able to complete using only one GPU.

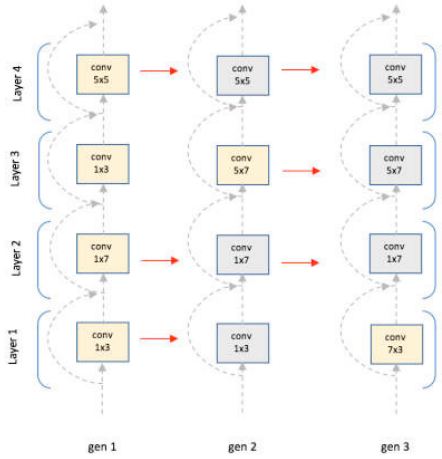

Fig. 4. The representation of parameter sharing between three different architectures. The yellow nodes show the initialization of parameters. The first architecture is all yellow as there are no previous parameters. The second and third architectures have only 1 yellow nodes as the rest of the parameters are inherited from previous architectures.

4. Experiments and Results

This section complements the methods section with the details necessary to reproduce the experiments. The first part describes the datasets and pre-processing used on the datasets. The latter part describes the parameters used for RGE and training the generated models. Finally, Table 1 shows a comparison with the previous best deep architectures along with results from the approach presented in this paper.

Datasets: The two CIFAR datasets consist of 60k colored natural images with 32×32 pixels. CIFAR-10 (C10) consists of images drawn from 10 and CIFAR-100 (C100) from 100 classes. The training and validation sets contain 50k and 5k images respectively, and the remaining 5k images are held as a test set. The experiments adopt a standard data augmentation scheme that is widely used for these two datasets [13-18] and denoted by a "+" mark at the end of the dataset name (e.g., C10+). For preprocessing, the images were normalized using the channel means and standard deviations. During evolution, the training and validation are implemented using corresponding datasets. The final results were reported on the test images using the best architecture.

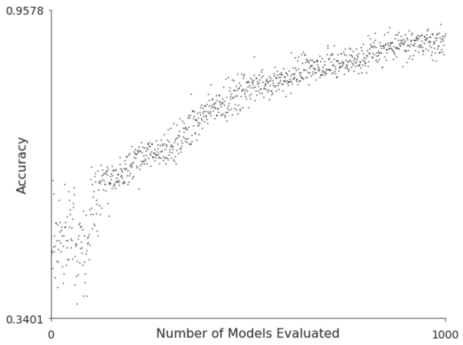

Fig. 5. The progress of validation accuracy on CIFAR-10 over 1000 generations.

Training details: The population for the RGE algorithm is set at P=100 and the sample size S is set at 10. The weights of all the architectures are initialized uniformly between -0.08 and 0.08. Once the RGE samples a population, a child model is constructed using mutation and trained for 2 epochs. The validation accuracy/fitness from trained model is used for selecting parents. The settings for training the CIFAR-10 and CIFAR-100 child models are the same as those used in [5]: momentum Optimizer with a learning rate of 0.1, weight decay of 1e-4, momentum of 0.9 and used Nesterov Momentum [28].

Table 1. Error rates % on CIFAR datasets.

Method	Params (M)	C10	C10+	C100	C100+
ResNet [1]	1.7M	-	6.61	-	-
ResNet (reported by [2])	1.7M	13.63	6.41	44.74	27.22
ResNet with Stochastic Depth [2]	1.7M	11.66	5.23	37.80	24.58
	10.2M	-	4.91	-	-
Wide ResNet [3]	11.0M	-	4.81	-	22.07
with Dropout	36.5M	-	4.17	-	20.50
ResNet (pre-activation) [4]	1.7M	11.26	5.46	35.58	24.33
	10.2M	10.56	4.62	33.47	22.71
DenseNet [5]	0.8M	5.92	4.51	27.15	22.27
e-block@C10	0.96M	4.69	4.22	-	-
e-block@2C100	3.7M	-	-	21.13	20.53

Results on CIFAR: For the task of image classification, the length of e-block is set to N = 12. The test accuracies of the best architectures are reported in Table 1 along with other state-of-the-art deep architecture models. As can be seen from the Fig 5, a deep model using e-block with data augmentation achieves an error rate of 4.22% on CIFAR-10, which is slightly better than the previous deep architecture of 4.51%. Fig 6 shows the e-block which achieved the top accuracy. Similarly, on CIFAR-100, it achieves an accuracy of 20.53% which is on par with the previous best deep model 20.50 %.

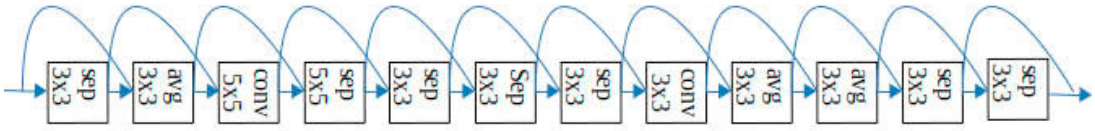

Fig. 6. The best configuration of e-block found on CIFAR-10 dataset.

5. Conclusion and Future work

This paper shows an efficient approach to search for deep architectures. It is the first attempt where a genetic algorithm and parameter sharing were used together for composing architectures. The results show that this approach is indeed capable of discovering architectures that are better than human made designs while using much less parameters than their predecessors. However, the results were not trained to beat the state-of-the-art results as its main focus is to show that deep search spaces are indeed capable of generating high performance architectures. Therefore, the future work will be targeted to generate architectures that are better than the current best architectures.

References

[1] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[2] Huang, Gao, et al. "Deep networks with stochastic depth." European conference on computer vision. Springer, Cham, 2016.

[3] Zagoruyko, Sergey, and Nikos Komodakis. "Wide residual networks." arXiv preprint arXiv:1605.07146 (2016).

[4] He, Kaiming, et al. "Identity mappings in deep residual networks." European conference on computer vision. Springer, Cham, 2016.

[5] Huang, Gao, et al. "Densely connected convolutional networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[6] Real, Esteban, et al. "Large-scale evolution of image classifiers." Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017.

[7] Zoph, Barret, and Quoc V. Le. "Neural architecture search with reinforcement learning." arXiv preprint arXiv:1611.01578(2016).

[8] Zoph, Barret, et al. "Learning transferable architectures for scalable image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

[9] Real, Esteban, et al. "Regularized evolution for image classifier architecture search." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 33. 2019.

[10] Pham, Hieu, et al. "Efficient neural architecture search via parameter sharing." arXiv preprint arXiv:1802.03268 (2018).

[11] Liu, Hanxiao, Karen Simonyan, and Yiming Yang. "Darts: Differentiable architecture search." arXiv preprint arXiv:1806.09055 (2018).

[12] Luo, Renqian, et al. "Neural architecture optimization." Advances in neural information processing systems. 2018.

[13] Larsson, Gustav, Michael Maire, and Gregory Shakhnarovich. "Fractalnet: Ultra-deep neural networks without residuals." arXiv preprint arXiv:1605.07648 (2016).

[14] Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." arXiv preprint arXiv:1312.4400 (2013).

[15] Romero, Adriana, et al. "Fitnets: Hints for thin deep nets." arXiv preprint arXiv:1412.6550 (2014).

[16] Lee, Chen-Yu, et al. "Deeply-supervised nets." Artificial intelligence and statistics. 2015.

[17] Springenberg, Jost Tobias, et al. "Striving for simplicity: The all convolutional net." arXiv preprint arXiv:1412.6806 (2014).

[18] Srivastava, Rupesh K., Klaus Greff, and Jürgen Schmidhuber. "Training very deep networks." Advances in neural information processing systems. 2015.

[19] Floreano, Dario, Peter Dürr, and Claudio Mattiussi. "Neuroevolution: from architectures to learning." Evolutionary intelligence 1.1 (2008): 47-62.

[20] Yao, Xin. "Evolving artificial neural networks." Proceedings of the IEEE 87.9 (1999): 1423-1447.

[21] Xie, Lingxi, and Alan Yuille. "Genetic cnn." Proceedings of the IEEE International Conference on Computer Vision. 2017.

[22] Baker, Bowen, et al. "Designing neural network architectures using reinforcement learning." arXiv preprint arXiv:1611.02167(2016).

[23] Cai, Han, et al. "Efficient architecture search by network transformation." Thirty-Second AAAI Conference on Artificial Intelligence. 2018.

[24] Liu, Chenxi, et al. "Progressive neural architecture search." Proceedings of the European Conference on Computer Vision (ECCV). 2018.

[25] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

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

[27] Gottapu, Ram Deepak, and Cihan H. Dagli. "System Architecting Approach for Designing Deep Learning Models." Procedia Computer Science 153 (2019): 37-44.

[28] Sutskever, Ilya, et al. "On the importance of initialization and momentum in deep learning." International conference on machine learning. 2013.