A Genetic Programming Approach to Design Convolutional Neural Network Architectures

1st Autor
Institution
Street Adress
City, Country Post Code
emailadress

2nd Autor
Institution
Street Adress
City, Country Post Code
emailadress

3rd Autor
Institution
Street Adress
City, Country Post Code
emailadress

ABSTRACT

Convolutional neural network (CNN), which is one of the deep learning models, has seen much success in a variety of computer vision tasks. However, designing CNN architectures still requires expert knowledge and much trial and errors. In this paper, we attempt to automatically construct the CNN architectures for an image classification task based on Cartesian genetic programming (CGP). In our method, we adopt the highly-functional modules such as convolutional blocks and tensor concatenation as the node functions in CGP. The CNN structure and connectivity represented by the CGP encoding method are optimized to maximize the validation accuracy. To evaluate the proposed method, we constructed the CNN architecture for the image classification task with the CIFAR-10 dataset. The experimental result shows that the proposed method can automatically find the competitive CNN architecture compared with state-of-the-art models.

KEYWORDS

genetic programming, convolutional neural network, designing neural network architectures, deep learning

ACM Reference format:
1st Autor, 2nd Autor, and 3rd Autor. 2017. A Genetic Programming Approach to Design Convolutional Neural Network Architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference 2017, Berlin, Germany, July 15–19, 2017 (GECCO '17),* 4 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Deep learning, which uses deep neural networks as a model, has shown a surprising performance on many challenging artificial intelligence and machine learning tasks such as image recognition [15, 16], speech recognition [10], and reinforcement learning tasks ++TODO: ref. of DQN ()++. In particular, convolutional neural networks (CNNs) [16] have seen huge success in image recognition tasks in the last few years and is applied to various computer vision tasks ++TODO: ref. e.g. GAN, colorization, image to text anotation... ()++. A commonly used CNN architecture mainly consists of several convolutions, pooling, and fully connected layers.

GECCO '17, Berlin, Germany

DOI: 10.1145/nnnnnnn.nnnnnnn

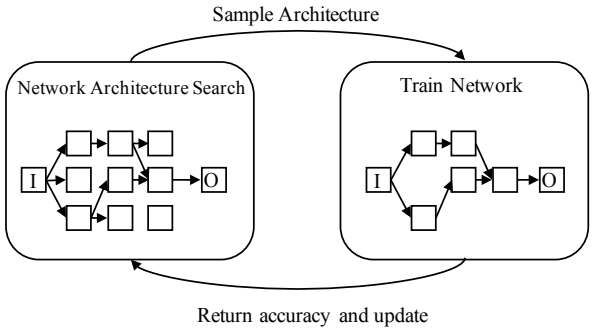

Figure 1: Overview of our method. Our method searches a CNN architecture using genetic programming. That CNN architecture is then trained on a learning task, and returns an accuracy. The network search is performed to maximize the accuracy by the evolutionary algorithm.

Several recent studies focus on developing the novel CNN architecture that achieves higher classification accuracy, e. g., GoogleNet [24], ResNet [9], and DensNet ++TODO: ref. ()++. Despite their success, designing CNN architectures is still a difficult task since there exist many design parameters such as the depth of a network, the type and parameters of each layer, and the connectivity of the layers. The state-of-the-art CNN architectures have become deep and complex, which suggests that a significant number of design parameters should be tuned to realize the best performance for a specific dataset. Therefore, the trial and error or expert knowledge are required when the users construct the suitable architecture for their target dataset. Because of this situation, the automatic design methods for the CNN architectures is highly beneficial.
[Note (Shinichi): Check up here.]

Genetic algorithms (GA) based approaches have been used to optimize architectures of neural networks [19]. Fernando et al. [6] have proposed the differentiable pattern producing networks (DPPN) to optimize weights of a denoising autoencoder. The DPPN is a differentiable version of the compositional pattern producing networks (CPPN) [22]. While the DPPN showed a good performance on image reconstruction tasks, this method is still limited in fixed-length models defined by hand. Verbancsics et al. [25] [26] have designed artificial neural networks and CNN architectures with the hypercube-based neuroevolution of augmenting topologies (HyperNEAT) [23]. However, to the best of our knowledge, these networks designed with HyperNEAT have failed to match the performance of state-of-the-art methods. Also, these methods

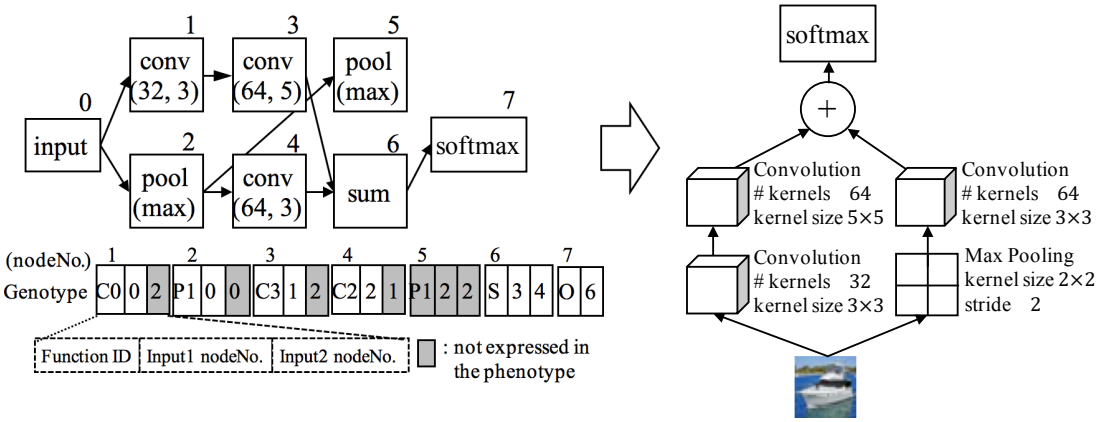

Figure 2: Example of a genotype and a phenotype. The genotype (Left) defines a CNN architecture (Right).

deal with the architectures defined by human experts, thus it is hard to design neural network architectures from scratch.

In this paper, we propose a genetic programming based approach for designing CNN architectures. Our method represents CNN architectures as a feedforward network structure that is similar to cartesian genetic programming (CGP) [17] [7], which allows us to design variable-length architectures. Intermediate nodes of the CGP have layers used in CNNs such as convolution and pooling layers. Once the CGP defines the CNN architecture, the CNN is trained on a learning task as shown in Figure 1. The types and connections of nodes in the network are then optimized to maximize the validation accuracy using the evolutionary algorithm. We call this CGP-based designing method the CGP2CNN. To evaluate the performance of the CGP2CNN, we carry out experiments with the image classification task using CIFAR-10 dataset [14]. Our experiment shows that the CGP2CNN can design good CNN architectures that is competitive with state-of-the-art methods.

The rest of this paper is organized as follows. Section 2 presents related work on automated neural network design. Section 3 describes the proposed method. Section 4 presents the experimental setup and results. In section 5, we concludes this paper and proposes future works.

2 RELATED WORK

Automating neural network design and hyperparameter optimization are important topic in machine learning. This section briefly reviews these research areas.

2.1 Hyperparameter optimization

For hyperparameter optimization, there are many methods such as grid search, gradient search [2], random search [3], and Bayesian optimization based methods [11], [20]. Recently, Bayesian optimization methods have shown a good performance in several datasets. Bergstra et al. [5] have proposed the Tree-structured parzen estimator (TPE) and shown better results than manual search and random search [3] in the hyperparameter optimization task. Also, Bergstra et al. [4] have proposed a meta-modeling approach based on the TPE for supporting automated hyperparamter optimization. While models based on the gaussian process are generally used for Bayesian optimization, gaussian process based approaches have a problem of increasing inference time cubically in the number of observations. To solve this problem, Snoek et al. [21] have replaced the gaussian process with a deep neural network, which improved its scalability.

While these methods have seen success, these methods are limited in fixed-length architectures, i.e., it is difficult to design variable-length architectures from scratch.

2.2 Designing neural network architectures

Recently studies on automating neural network design using reinforcement learning have been made [27] [1]. These studies showed that a reinforcement learning based method is able to generate good CNN architectures for image classification tasks. In [27], a recurrent neural network (RNN) is used to generate the neural network architecture, and the RNN is trained with reinforcement learning to maximize the expected accuracy on a learning task. To accelerate training, this method uses distributed training and asynchronous parameter updates with 800 GPUs. Baker et al. [1] have proposed a meta-modeling approach based on reinforcement learning to compose CNN architectures. A Q-learning agent explores and exploits a space of model architectures with an $\epsilon-$greedy strategy and experience replay.

Unlike these approaches, our method adopts a genetic programming based approach to design CNN architectures. In addition, we introduce a highly-functional modules such as convolutional blocks and tensor concatenations in order to effectively find better CNN architectures.

3 DESIGNING CNN ARCHITECTURES USING GENETIC PROGRAMMING

Our method represents a CNN architecture as a feedforward network structure that is defined by the CGP. The CNN architecture defined by the CGP is trained on a learning task, and the network is optimized to maximize the validation accuracy by the evolutionary algorithm. In the following section, we will describe how CNN

Table 1: Parameter details for each layer

Node type	Parameters	Values or description
Convolution	Kernel size	$\in \{3 \times 3, 5 \times 5\}$
	# kernels	$\in \{32, 64, 128\}$
	Stride	1
	# inputs	1
Pooling	Kernel size	2×2
	Stride	2
	# inputs	1
Summation	# inputs	2 Element-wise addition.
Concatenation	# inputs	2 Concatenate in the depth dimension.

architectures can be designed and optimized with the CGP in detail.

3.1 Designing CNN architectures with the CGP

We use a feedforward network that consists of R rows by C columns, i.e., the number of intermediate nodes of the network is $R \times C$. Figure 2 shows an example of the network of two rows by three columns and its genotype and phenotype. A type of each node is represented as four different types of layers: convolution, pooling, summation, and concatenation. The detailed parameters for each layer are shown in Table 1. The convolution node consists of a standard convolution processing with batch normalization [12] and rectified linear units [18]. The pooling node consists of the standard max pooling and average pooling processing. The summation node performs element-wise addition of two feature maps, channel by channel. If input feature maps to be added have different sizes, we pad the small feature map with zeros for increasing dimensions. The concatenation node concatenates two feature maps in the depth dimensions. If input feature maps to be concatenated are different sizes, we pad the small feature map with zeros. Adding these summation and concatenation nodes to the network architecture allow our method to generate shortcut connections or branching layers such as GoogleNet [24] and Residual Net [9].

The CGP inter-connectivity is determined by the levels back parameter l, which decides nodes of how many previous columns to connect to nodes in the current column, i.e., each node can have one or two connections to nodes in the l previous columns. Note that nodes in the same column are prohibited to be connected to each other. The types and connections of nodes are optimized by the evolutionary algorithm.

3.2 Optimization of the CGP

We use mutation as a genetic operator. The mutation operator is applied to one individual as follows:

- Select several nodes with probability ϵ for each node.
- Change the type and connectivity of the selected nodes randomly.

Note that we apply the mutation operator to nodes until one or more active nodes that are associated with the output path are changed. As training a CNN can take hours, we apply the above mutation operator to save time.

Table 2: Parameter settings for the CGP

Parameters	Values or description
# Generations	300
Mutation rate	0.05
# Rows	5
# Columns	30
Levels back	10
Generation alternation model	(1 + 2) Evolution Strategy

We use a simple form of $(1 + \lambda)$ evolutionary strategy (with $\lambda = 2$) to design CNN architectures. The algorithm is as follows:

1. Generate an initial individual at random as a parent M, and train the CNN defined by M.
2. Generate a set of two offsprings C by applying the mutation operation to M.
3. Train these two CNNs defined by offsprings C.
4. Select an elite individual from the set of M and C, and replace M with the elite individual.
5. Return to step 2 until a stopping criterion is satisfied.

We employ the accuracy of the CNN on a validation set as the fitness of each individual.

4 EXPERIMENTS AND RESULTS

4.1 Dataset

We test our method to the image classification task with CIFAR-10 dataset which has 10 classes with 50,000 training images and 10,000 test images. We sampled 5,000 images randomly from the training set for the validation set, the remaining 45,000 images are the training set. In this experiment, we use data preprocessing and augmentation procedure. We first preprocess the data with the per-pixel mean subtraction. We then pad 4 pixels on each side, and choose a random 32×32 crop from the padded image. At last, we perform random horizontal flips on the cropped 32×32 image.

4.2 Training details

Parameter settings for the CGP is shown in Table 2. Once the CGP samples a CNN architecture, every CNN is trained for 50 epochs with the Adam optimizer [13] with an initial learning rate of 0.01. We reduce the learning rate by a factor of 10 every 20 epochs. All weights of CNN are initialized with He [8], and the batch size is 128.

After finishing the evolution of the CGP, we select the architecture that showed the best validation accuracy. We then fine-tune the best model with a different training schedule. During this phase, we set a learning rate to 0.01 for 50 epochs, then 0.001 for 50 epochs. The other parameters are unchanged. After training the best model, we apply the model to the test set.

4.3 Results

5 CONCLUSIONS

REFERENCES

[1] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2016. Designing Neural Network Architectures using Reinforcement Learning. *arXiv preprint arXiv:1611.02167* (2016).

[2] Yoshua Bengio. 2000. Gradient-based optimization of hyperparameters. *Neural computation* 12, 8 (2000), 1889–1900.

[3] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.

[4] James Bergstra, Daniel Yamins, and David D. Cox. 2013. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. *ICML (1)* 28 (2013), 115–123.

[5] James S. Bergstra, Remi Bardenet, Yoshua Bengio, and Balazs Kegl. 2011. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*. 2546–2554.

[6] Chrisantha Fernando, Dylan Banarse, Malcolm Reynolds, Frederic Besse, David Pfau, Max Jaderberg, Marc Lanctot, and Daan Wierstra. 2016. Convolution by evolution: Differentiable pattern producing networks. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM, 109–116.

[7] Simon Harding. 2008. Evolution of image filters on graphics processor units using cartesian genetic programming. In *IEEE Congress on Evolutionary Computation*. IEEE, 1921–1928.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*. 1026–1034.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[10] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and others. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[11] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*. Springer, 507–523.

[12] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[13] Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *International Conference on Learning Representaions* (2015), 2452–2459.

[14] Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. *Technical report* (2009).

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[16] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[17] Julian F. Miller and Peter Thomson. 2000. Cartesian genetic programming. In *European Conference on Genetic Programming*. Springer, 121–132.

[18] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. 807–814.

[19] J. David Schaffer, Darrell Whitley, and Larry J. Eshelman. 1992. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *Combinations of Genetic Algorithms and Neural Networks, 1992., COGANN-92. International Workshop on*. IEEE, 1–37.

[20] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*. 2951–2959.

[21] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md Mostofa Ali Patwary, Mr Prabhat, and Ryan P. Adams. 2015. Scalable Bayesian Optimization Using Deep Neural Networks.. In *ICML*. 2171–2180.

[22] Kenneth O. Stanley. 2007. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines* 8, 2 (2007), 131–162.

[23] Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. 2009. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life* 15, 2 (2009), 185–212.

[24] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.

[25] Phillip Verbancsics and Josh Harguess. 2013. Generative neuroevolution for deep learning. *arXiv preprint arXiv:1312.5355* (2013).

[26] Phillip Verbancsics and Josh Harguess. 2015. Image Classification Using Generative Neuro Evolution for Deep Learning. In *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*. IEEE, 488–493.

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