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),* 7 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Deep learning, which uses deep neural networks as a model, has shown a good performance on many challenging artificial intelligence and machine learning tasks such as image recognition [18, 19], speech recognition [12], and reinforcement learning tasks [22, 23]. In particular, convolutional neural networks (CNNs) [19] have seen huge success in image recognition tasks in the last few years and is applied to various computer vision application [36, 37]. [removed: "e.g. GAN [7], colorization [37], image to text anotation [36]."[]] A commonly used CNN architecture mainly consists of several convolutions, pooling, and fully connected layers. Several recent studies focus on developing the novel CNN architecture that achieves higher classification accuracy, e.g., GoogleNet [32], ResNet [11], and DensNet [13]. 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.

The neural network architecture design can be viewed as the model selection problem in the machine learning. The straightforward approach is to deal with the architecture design as the hyperparameter optimization problem, optimizing the hyperparameters such as the number of layers and neurons using black-box optimization techniques [27?].

Evolutionary computation has been traditionally applied to design the neural network architectures [25, 31]. There are two types of encoding schemes for the network representation: direct and indirect coding. The direct coding represents the number and connectivity of neurons directly as the genotype, whereas the indirect coding represents a generation rule of the network architectures. While almost traditional approaches optimize the number and connectivity of low-level neurons, the modern neural network architectures for deep learning have many units and various types of units, e.g., convolution, pooling, and normalization. Optimizing those many parameters in reasonable computational time may be difficult. Therefore, the use of the highly-functional modules as a minimum unit is promising.

In this paper, we attempt to design CNN architectures based on a genetic programming. We use the Cartesian genetic programming (CGP) [9, 21?] encoding scheme, one of the direct encoding, to represent the CNN structure and connectivity. The advantage of this representation is its flexibility; it can represent variable-length network structure and the skip connections. Moreover, we adopt the relatively highly-functional modules such as convolutional blocks and tensor concatenation as the node functions in CGP to reduce the search space. To evaluate the architecture represented by the CGP, we train the network using training dataset in an ordinary way. Then the performance for another validation dataset is assigned as the fitness of the architecture. Based on this fitness evaluation, an evolutionary algorithm optimizes the CNN architectures. To check the performance of the proposed method, we conducted the experiment constructing the CNN architecture for the image classification task with the CIFAR-10 dataset [17]. The experimental result shows that the proposed method can automatically find the competitive CNN architecture compared with state-of-the-art models. [Note (Shinichi): It would be nice if we can describe the research question clearly.]

The rest of this paper is organized as follows. The next section presents related work on the neural network architecture design. In Section 3, we describe our genetic programming approach to design the CNN architectures. We test the performance of the proposed approach through the experiment. Finally, in Section 5, we describe our conclusion and future work.

2 RELATED WORK

This section briefly reviews the related work on the automatic neural network architecture design: hyperparameter optimization, evolutionary neural networks, and reinforcement learning approach.

2.1 Hyperparameter Optimization

We can consider the neural network architecture design as the model selection or hyperparameter optimization problem from machine learning perspective. There are many hyperparameter tuning methods for the machine learning algorithm such as grid search, gradient search [2], random search [3], and Bayesian optimization based methods [14, 27]. Naturally, evolutionary algorithms have also been applied to the hyperparameter optimization problems [?]. In the machine learning community, Bayesian optimization is often used and have shown good performance in several datasets. Bayesian optimization is a global optimization method of black-box and noisy objective functions, which maintains a surrogate model learned by using previously evaluated solutions. A Gaussian process is usually adopted as the surrogate model [27], which can easily handle the uncertainty and noise of the objective function. Bergstra et al. [5] have proposed the Tree-structured Parzen estimator (TPE) and shown better results than manual search and random search. They have also proposed a meta-modeling approach [4] based on the TPE for supporting automatic hyperparameter optimization. Snoek et al. [28] used a deep neural network instead of the Gaussian process to reduce the computational cost for the surrogate model building and succeeded to improve the scalability.

The hyperparameter optimization approach often tunes the predefined hyperparameters such as the numbers of layers and neurons, and the type of activation functions. While this method has seen success, it is hard to design more flexible architectures from scratch.

2.2 Evolutionary Neural Networks

Evolutionary algorithms have been used to optimize the neural network architectures so far [25, 31]. The traditional approaches are not suitable for the model deep neural network architecture design since they usually optimize the number and connectivity of low-level neurons.

Recently, Fernando et al. [6] have proposed the differentiable pattern-producing networks (DPPNs) to optimize weights of a denoising autoencoder. The DPPN is a differentiable version of the compositional pattern-producing networks (CPPNs) [29]. This method focuses on the effectiveness of the indirect coding for weight optimization. That is, the general structure of network should be predefined.

Verbancsics et al. [34, 35] have designed the artificial neural networks and CNN architectures with the hypercube-based neuroevolution of augmenting topologies (HyperNEAT) [30]. 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 deal with the architectures defined by human experts. Thus it is hard to design neural network architectures from scratch. [Note (Shinichi): I will modify the explanation of the Verbancsics's papers.]

2.3 Reinforcement Learning Approach

The interesting approaches, automatic designing the deep neural network architecture using reinforcement learning, have been attempted recently [1, 38]. These studies showed that a reinforcement learning based method could construct the competitive CNN architectures for image classification tasks. In [38], a recurrent neural network (RNN) is used to generate the neural network architectures, and the RNN is trained with reinforcement learning to maximize the expected accuracy on a learning task. This method uses distributed training and asynchronous parameter updates with 800 GPUs to accelerate the reinforcement learning process. Baker et al. [1] have proposed a meta-modeling approach based on reinforcement learning to produce the CNN architectures. A Q-learning agent explores and exploits a space of model architectures with an $\epsilon-$greedy strategy and experience replay.

These approaches adopt the indirect coding scheme for the network representation, which optimizes generative rules for the network architectures such as the RNN. Unlike these approaches, our approach uses the direct coding based on Cartesian genetic programming to design the CNN architectures. Besides, we introduce the relatively highly-functional modules such as convolutional blocks and tensor concatenations to find better CNN architectures efficiently.

3 CNN ARCHITECTURE DESIGN USING CARTESIAN GENETIC PROGRAMMING

Our method directly encodes the CNN architectures based on CGP [9, 21?] and uses the highly-functional modules as the node functions. The CNN architecture defined by the CGP is trained using training dataset, followed by the validation accuracy is assigned as the fitness of the architecture. Then the architecture is optimized to maximize the validation accuracy by the evolutionary algorithm. Figure 1 illustrates the overview of our method.

In this section, we describe the network representation and the evolutionary algorithm used in the proposed method in detailed.

3.1 Representation of CNN Architectures

We use the CGP encoding scheme, representing the program as the form of directed acyclic graphs with a two-dimensional grid defined on computational nodes, for the CNN architecture representation. Let assume the grid has N_r rows by N_c columns, then the number of intermediate nodes is $N_r \times N_c$, and the numbers

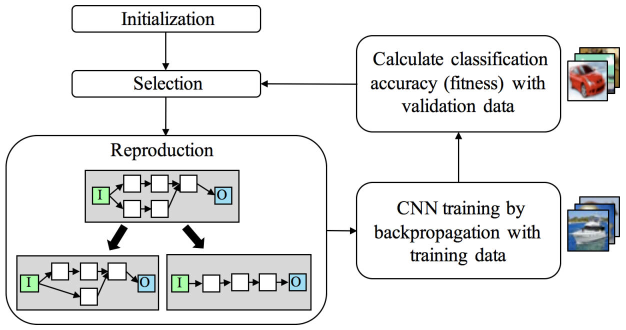

Initialization
Selection
Reproduction
I
O
Calculate classification accuracy (fitness) with validation data
CNN training by backpropagation with training data

Figure 1: Overview of our method. Our method represents the CNN architectures based on Cartegian genetic programming. The CNN architecture is trained on a learning task and assigned the validation accuracy of the trained model as the fitness. The evolutionary algorithm searches the better architectures.

Table 1: Parameter details for each layer

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

of inputs and outputs depend on the task. The genotype consists of the integers with fixed length, and each gene has the information of the type and connections of the node. The r-th column's nodes should be connected from the $(r - l)$ to $(r - 1)$-th column's nodes, where l is called levels-back parameter. Figure 2 shows an example of the genotype, the corresponding network and CNN architecture in the case of two rows by three columns. While the genotype in CGP is a fixed length representation, the number of nodes in the phenotypic network varies since not all the nodes are not connected to the output nodes. The node No. 5 in the left of Figure 2 is inactive nodes.

Referring the modern CNN architectures, we select the highly-functional modules as the node function. The frequently-used processings in the CNN are convolution and pooling; convolution processing uses a local connectivity and spatially shares the learnable weights, and pooling is nonlinear down-sampling. We prepare the six types of node functions called ConvBlock, ResBlock, max-pooling, average-pooling, tensor concatenation, and tensor summation.

The ConvBlock consists of a standard convolution processing with stride 1 followed by batch normalization [15] and rectified linear units (ReLU) [24]. In the ConvBlock, we pad the outside of an input feature map with zero values before the convolution operation so as to keep the row and column sizes of the output. As a result, the $M \times N \times C$ input feature map is transformed into the $M \times N \times C'$ output, where M, N, C, and C' are the numbers of rows, columns, input channels, and output channels, [removed: "of a convolution layer"[]] respectively.

The ResBlock is composed of a convolution processing, batch normalization, ReLU, and tensor summation. A ResBlock architecture is shown in Figure 3. The ResBlock performs identity mapping by shortcut connections as described in [11]. The row and column sizes of the input are preserved in the same way as ConvBlock after convolution. The output feature map of the ResBlock is calculated by the ReLU activation and the tensor summation with the input feature map and the processed feature map as shown in Figure 3. In the ResBlock, the $M \times N \times C$ input feature map is transformed into the $M \times N \times C'$ output.

The max- and average-poolings use the maximum and average values as the values of the down-sampled images, respectively. In the pooling operation, the $M \times N \times C$ input feature map is transformed into the $M' \times N' \times C$ output, where $M' = \lfloor M/2 \rfloor$ and $N' = \lfloor N/2 \rfloor$.

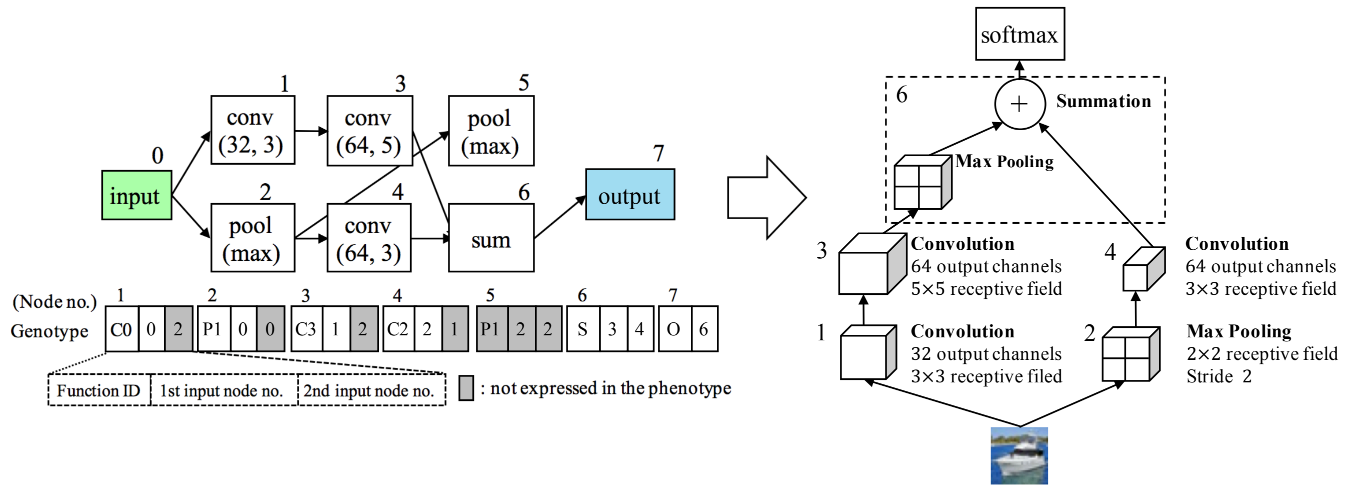
The tensor concatenation concatenates two feature maps in the channel dimensions. If the input feature maps to be concatenated have the different number of rows or columns, we down-sample the larger feature map by max-pooling so as to become the same sizes of inputs. In the concatenation operation, the size of the output feature map is $\min(M_1, M_2) \times \min(N_1, N_2) \times (C_1 + C_2)$, where the sizes of the inputs are $M_1 \times N_1 \times C_1$ and $M_2 \times N_2 \times C_2$.

The tensor summation performs element-wise addition of two feature maps, channel by channel. In the same way as concatenation, if the input feature maps to be added have the different number of rows or columns, we down-sample the larger feature map by max-pooling. Besides, if the inputs have the different number of channels, we pad the smaller feature map with zeros for increasing channels. In the summation operation, the size of the output feature map is $\min(M_1, M_2) \times \min(N_1, N_2) \times \max(C_1, C_2)$, where the sizes of the inputs are $M_1 \times N_1 \times C_1$ and $M_2 \times N_2 \times C_2$.

Adding these summation and concatenation operations allows our method to represent shortcut connections or branching layers such as used in GoogleNet [32] and Residual Net [11] without the ResBlock. The output node represents softmax function with the number of classes. The outputs fully connect to all elements of the input. The node functions used in the experiments are displayed in Table 1.

3.2 Evolutionary Algorithm

We use a point mutation as the genetic operator as same as the standard CGP. The type and connections of each node randomly change to valid values according to a mutation rate. The standard CGP mutation has a possibility only to affect the inactive node.



input
conv (32, 3)
pool (max)
conv (64, 5)
conv (64, 3)
pool (max)
sum
output
(Node no.)
Genotype
C0 0 2 P1 0 0 C3 1 2 C2 2 1 P1 2 2 S 3 4 O 6
Function ID
1st input node no.
2nd input node no.
: not expressed in the phenotype
softmax
Summation
Max Pooling
Convolution 64 output channels 5×5 receptive field
Convolution 64 output channels 3×3 receptive field
Convolution 32 output channels 3×3 receptive filed
Max Pooling 2×2 receptive field Stride 2

Figure 2: Example of a genotype and a phenotype. The genotype (Left) defines the CNN architecture (Right). The max-pooling operation is inserted before the summation processing because each input feature map to the summation node is a different size.

In that case, the phenotype (representing CNN architecture) does not change by the mutation and does not need fitness evaluation again.

The fitness evaluation of the CNN architectures is so expensive since it requires the training of CNN. To efficiently use the computational resource, we want to evaluate some candidate solutions in parallel at each generation. Therefore, we apply the mutation operator until at least one active node changes to reproduce the candidate solution. We call this mutation as forced mutation. Moreover, to keep a neutral drift, which is effective for CGP evolution [21?], we modify a parent by the neutral mutation if the fitnesses of the offsprings do not improve. Here, the neutral mutation only changes the genes of inactive nodes without modification of the phenotype.

We use the modified $(1 + \lambda)$ evolutionary strategy (with $\lambda = 2$ in our experiments) that take in the above artifice. The procedure of our modified algorithm is as follows:

(1) Generate an initial individual at random as a parent P and train the CNN represented by P followed by assigning the validation accuracy as the fitness.
(2) Generate a set of λ offsprings C by applying the forced mutation to P.
(3) Train the λ CNNs represented by offsprings C in parallel and assign the validation accuracies as the fitness.
(4) Apply the neutral mutation to the parent P.
(5) Select an elite individual from the set of P and C and then replace P with the elite individual.
(6) Return to step 2 until a stopping criterion is satisfied.

4 EXPERIMENTS AND RESULTS

4.1 Dataset

We test our method on the image classification task using the CIFAR-10 dataset in which the number of classes is 10. The numbers of training and test images are 50,000 and 10,000, respectively, and the size of images is 32×32.

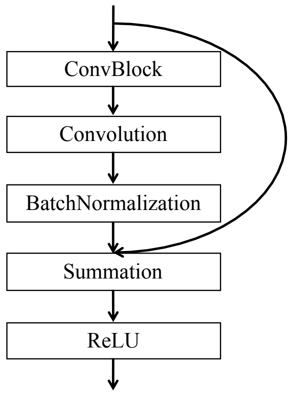


Figure 3: ResBlock architecture.

We consider two experimental scenarios: default scenario and small-data scenario. The default scenario uses the default numbers of the training images, while the small-data scenario assumes that we only use 5,000 images as the learning data.

In the default scenario, we randomly sampled 45,000 images from the training set to train the CNN and use the remaining 5,000 images for the validation set of the CGP fitness evaluation. In the small-data scenario, we randomly sampled 4,500 images for the training and 500 images for the validation.

4.2 Experimental Setting

To assign the fitness to the candidate CNN architectures, we train the CNN by stochastic gradient descent with the mini-batch size of 128. The softmax cross entropy loss is used as the loss function. We initialize the weights by the He's method [10] and use the Adam

Table 2: Parameter settings for the CGP

Parameters	Values
# Generations	300
Mutation rate	0.05
# Rows (N_r)	5
# Columns (N_c)	30
Levels-back (l)	10

optimizer [16] with an initial learning rate of 0.01. We train each CNN for 50 epochs and reduce the learning rate by a factor of 10 at 30th epoch.

We preprocess the data with the per-pixel mean subtraction. To prevent overfitting, we use weight decay with the coefficient 1.0×10^{-4} and data augmentation. We use the data augmentation method based on [11]; padding 4 pixels on each side followed by choosing a random 32×32 crop from the padded image, and random horizontal flips on the cropped 32×32 image.

We implement our methods using the Chainer [33] (version 1.16.0) framework and run the experiments on the machines with 3.2GHz CPU, 32GB RAM, and two NVIDIA GeForce GTX 1080 GPUs. The parameter setting for the CGP is shown in Table 2. We use the relatively larger number of columns than the number of rows to generate deep architectures likely. The offspring size λ is set to two that is the same number of GPUs in our experimental machines. The numbers of generations are 300 and 2,000 for the default and small-data scenario, respectively. [Note (Shinichi): It may change...]
++TODO: Describe about node function sets; ConvSet, ResSet ()++

After the CGP process, we re-train the best CNN architecture using each training images (50,000 for the default scenario and 5,000 for the small-data scenario) and calculate the classification accuracy for the 10,000 test images to evaluate the constructed CNN architectures. In this re-training phase, we optimize the weights of the obtained architecture with a different training schedule; we use SGD with a mini-batch size of 128, a weight decay of 10^{-4}, and a momentum of 0.9. We set an initial learning rate to 0.01, then divide it by a factor of 10 at 250 and 375 epochs, and train the model for 500 epochs. The other parameters are same as used in the CGP process.
[Note (Shinichi): Checked up to here.]

4.3 Results with the default scenario

We compare the classification performance of our method with the state-of-the-art-methods and summarize the results in Table 3. We refer to our architecture designed using the ConvBlock, max-pooling, average-pooling, summation, and concatenation functions as CGP-CNN (A) and the architecture using the ResBlock as an alternative to the ConvBlock as CGP-CNN (B). As can be seen in Table 3, our results are competitive with the state-of-the-art-methods. In particular, CGP-CNN (B) outperforms all hand-crafted models and our architectures have a good balance between classification accuracy and the number of parameters. In this experiment, the architectures of CGP-CNN (A) and CGP-CNN (B) are quite different as seen in Figure 4. For example, the summation and concatenation node are used frequently in CGP-CNN (A). The nodes, on the other hand, are not used in CGP-CNN (B) but each model shows a good classification performance. Added to this, we find that CGP-CNN (B) architecture has a similar features to the ResNet [11]. The architecture of the ResNet [11] has several pairs of convolution layers and convolutions that performs downsampling with a stride of 2. Although our method cannot perform downsampling by convolution functions, we can see from Figure 4 that CGP-CNN (B) uses average pooling as an alternative to the convolution layers for downsampling. Furthermore, CGP-CNN (B) has some pairs of convolution nodes like the ResNet. It follows from what has been said that our method also can design a architecture similar to one designed by human experts and its model shows a better performance. Also, while the ResNet has a deep 110-layer architecture, CGP-CNN (B) has a relatively shallow architecture and outperforms the ResNet. We can see from this result that the number of output channels of the ResBlock is more effective than the depth at improving the classification accuracy on the CIFAR-10 dataset.

Table 3: Comparison of error rate on CIFAR-10 dataset.

Model	Error rate	# params (10^6)
Maxout [8]	9.38	-
MetaQNN [1] [1]	9.09	3.7
Network in Network [20]	8.81	-
VGG [26] [2]	7.94	15.2
CGP-CNN (A)	7.89	1.52
ResNet [11]	6.61	1.7
CGP-CNN (B)	5.98	1.68
Neural Architecture Search [38]	3.84	32.0

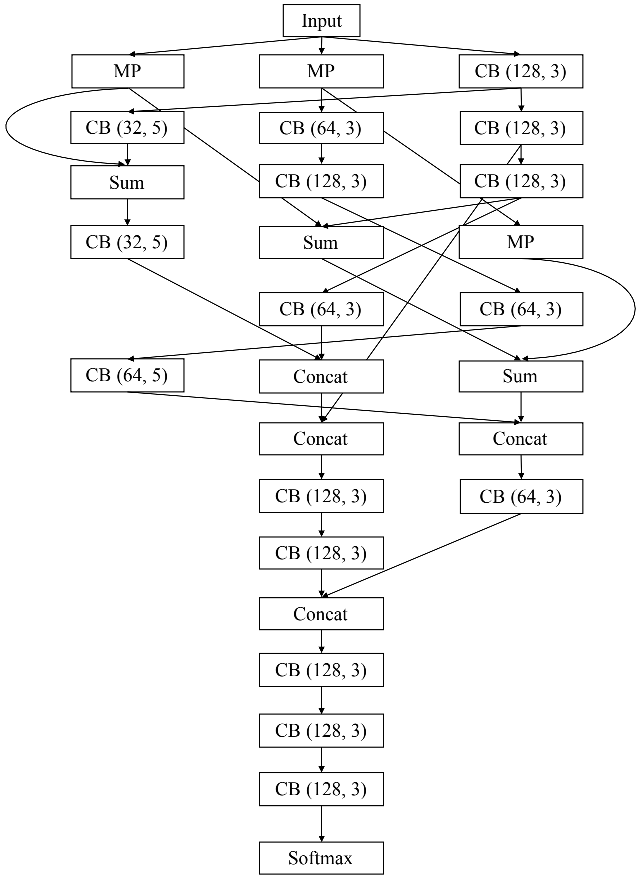
To explore the effectiveness of the number of output channels, we change the number of output channels CGP-CNN (A) can use from $\{32, 64, 128\}$ to $\{64, 128, 256, 512\}$ and refer to this model as CGP-CNN (C). The result is improved to the error rate 7.25% and the CGP-CNN (C) has a more shallow architecture and fewer nodes than that of the CGP-CNN (A).

4.4 Results with the small-data scenario

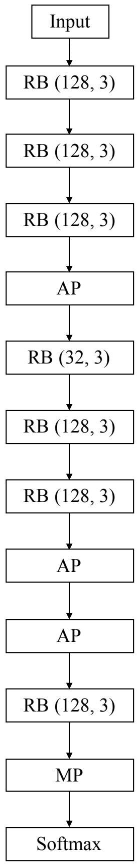
For settings of our method, we changed the number of generation to 2,000. The other experimental settings are the same in the preceding section. In this experiment, we compare our method with the VGG and ResNet. For the VGG settings, the VGG is trained for 200 epochs with SGD with an initial learning rate of 0.01. We reduce the learning rate by a factor of 10 every 50 epochs. For the ResNet, we train the model for 500 epochs. We use SGD and start with a learning rate of 0.1, then divide it by 10 at 250 and 375 epochs.

Table 4 shows that our model can find better architectures than the VGG and ResNet. The CGP-CNN (A) architecture is shown in Figure 5. As seen in Figure 5, our method discovers the structure different from that in the preceding section. In addition, we retrain this model with 50,000 training images and then compute the test

[1]The mean error rate and the number of parameters of top 5 models are shown.
[2]We implemented the VGG net [26] and applied the model to the CIFAR-10 dataset since the VGG net is not applied to the CIFAR-10 dataset in [26]. The architecture of the VGG is identical with the configuration D in [26]. We denote this model as VGG in this paper.



(a) CGP-CNN (A)



(b) CGP-CNN (B)

Figure 4: Architectures designed by our method.

Table 4: Comparison of error rate with the small CIFAR-10 dataset.

Model	Error rate	# params (10^6)
VGG [26]	27.67	15.2
CGP-CNN (B)	24.50	0.91
ResNet [11]	24.10	1.7
CGP-CNN (A)	22.82	3.9

accuracy, and achieves 9.2% error rate on the test set. We can see from these experiments that even with a small dataset, our method can design a relatively good architecture.

5 CONCLUSIONS

In this paper we propose a GP-based method for designing CNN architectures and show that our method is able to deign competitive models compared with state-of-the-art method on the image classification task. By using a highly-functional modules such as ConvBlock and ResBlock, our method achieves a high performance and efficient optimization.

In our future work, we will evaluate our method with a more challenging task. Also, we will improve optimization cost of our method because our method take about a week to design architectures in this experiment.

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] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative

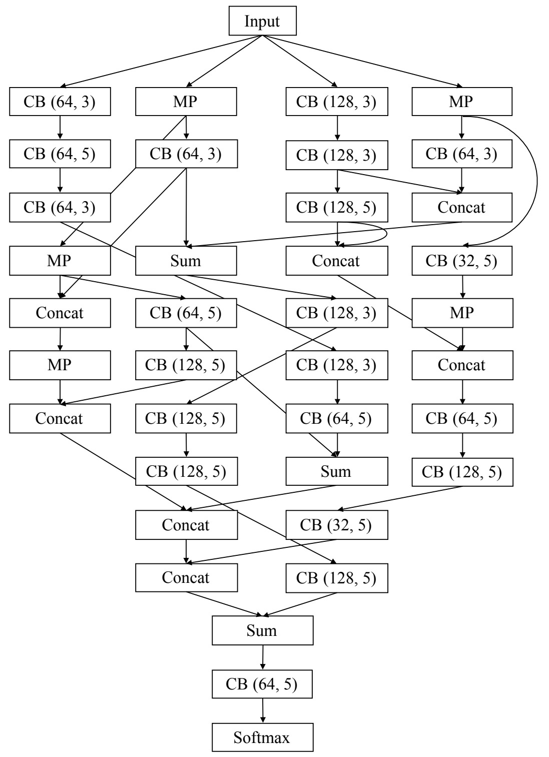


Figure 5: The architecture of CGP-CNN (A) with the small-data scenario.

adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[8] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C. Courville, and Yoshua Bengio. 2013. Maxout Networks. *ICML (3)* 28 (2013), 1319–1327.

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

[10] 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.

[11] 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.

[12] 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.

[13] Gao Huang, Zhuang Liu, Kilian Q. Weinberger, and Laurens van der Maaten. 2016. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993* (2016).

[14] 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.

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

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

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

[18] 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.

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

[20] Min Lin, Qiang Chen, and Shuicheng Yan. 2013. Network in network. *International Conference on Learning Representations* (2013).

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

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *Advances in neural information processing systems Workshop on Deep Learning* (2013).

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, and others. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533. http://www.nature.com/nature/journal/v518/n7540/abs/nature14236.html

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

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

[30] 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.

[31] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127.

[32] 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.

[33] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. 2015. Chainer: a next-generation open source framework for deep learning. In *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*.

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

[35] 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.

[36] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3156–3164.

[37] Richard Zhang, Phillip Isola, and Alexei A. Efros. 2016. Colorful image colorization. In *European Conference on Computer Vision*. Springer, 649–666.

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