A Genetic Programming Approach to Designing Convolutional Neural Network Architectures

Masanori Suganuma\textsuperscript{1,2}, Shinichi Shirakawa\textsuperscript{3} and Tomoharu Nagao\textsuperscript{3}
\textsuperscript{1} RIKEN Center for AIP
\textsuperscript{2} Tohoku University
\textsuperscript{3} Yokohama National University
suganuma@vision.is.tohoku.ac.jp, shirakawa-shinichi-bg@ynu.ac.jp, nagao@ynu.ac.jp

Abstract

We propose a method for designing convolutional neural network (CNN) architectures based on Cartesian genetic programming (CGP). In the proposed method, the architectures of CNNs are represented by directed acyclic graphs, in which each node represents highly-functional modules such as convolutional blocks and tensor operations, and each edge represents the connectivity of layers. The architecture is optimized to maximize the classification accuracy for a validation dataset by an evolutionary algorithm. We show that the proposed method can find competitive CNN architectures compared with state-of-the-art methods on the image classification task using CIFAR-10 and CIFAR-100 datasets.

1 Introduction

Deep neural networks (DNNs) have shown excellent performance on many challenging machine learning tasks, such as image recognition, speech recognition, and reinforcement learning tasks. Convolutional neural networks (CNNs) [Le-Cun et al., 1998], the DNN model often used for computer vision tasks, have seen huge success, particularly in image recognition tasks in the past few years. A standard CNN architecture consists of several convolutions, pooling, and fully connected layers. Recent studies have proposed a new CNN architecture that achieves higher performance, e.g., GoogLeNet [Szegedy et al., 2015] and ResNet [He et al., 2016]. These widely-used networks are designed by humans, but designing such architectures requires expert knowledge and trial-and-error. Hence, the automatic design of CNN architectures has much attention.

Recent studies for the automatic design of CNN require many weight-parameter optimizations; the weights of candidate CNN architectures are trained by a stochastic gradient descent (SGD), and the architectures are evaluated with trained weights on a validation dataset\textsuperscript{1}. Evolutionary computation (EC) or reinforcement learning (RL) is employed as an architecture search method. The EC based architecture search methods [Miikkulainen et al., 2017; Real et al., 2017; Xie and Yuille, 2017] generate a new architecture by recombination and mutation operators, and evaluate it on a validation dataset after the weight training. The EC algorithms are expected to find the architecture that maximizes the performance (e.g., classification accuracy). The RL based architecture search methods [Baker et al., 2017; Zoph and Le, 2017] optimize the policy that generates the CNN architectures by such as the policy gradient and Q-learning. Also, the reward used in the RL algorithms is the performance on a validation dataset. Reducing the computational cost for the architecture search is crucial topic because these methods require much computational cost to optimize neural network architectures.

In this work, we attempt to design CNN architectures based on genetic programming. We use the Cartesian genetic programming (CGP) [Miller and Thomson, 2000] encoding scheme to represent the CNN architecture, where the architecture is represented by a directed acyclic graph. The advantage of this representation is its flexibility; it can represent variable-length network architectures and skip connections. In addition, we adopt relatively highly-functional modules such as convolutional blocks and tensor concatenation as the node functions in the graph to reduce the search space of architectures. To evaluate the architecture represented by CGP, we train the network by SGD on a training dataset. The performance on a validation dataset is then assigned as the fitness of the architecture. Based on this fitness evaluation, the architecture is optimized to maximize the fitness (i.e., the validation accuracy) by evolutionary algorithms. To verify the performance of the proposed approach, we experimented involving constructing a CNN architecture for the image classification task with the CIFAR-10 and CIFAR-100 datasets. The experimental result shows that the proposed method can be used to find competitive CNN architectures compared with

\textsuperscript{1}In contrast, the traditional evolutionary neural networks, so-called neuroevolution [Stanley and Miikkulainen, 2002], typically optimizes both of the weights and architecture at the same time by an evolutionary computation method.
2 CNN Architecture Design Using Cartesian Genetic Programming

2.1 Representation of CNN Architectures

We represent the architecture of CNNs by a directed acyclic graph defined on a two-dimensional grid of \(M\) rows and \(N\) columns. This graph is optimized by an evolutionary algorithm, where the graph called phenotype is encoded by a list of integers called genotype. Figure 1 shows an example of a genotype (top), a phenotype (middle) and the corresponding CNN architecture (bottom).

A genotype consists of \(MN + 1\) genes, and each gene has information regarding the type \(T\) and connections \(C\) of a node. The node corresponds to a highly-functional module in CNN, e.g., the type \(T\) specifies one of the convolution or residual blocks, pooling layers, and tensor operations described later in detail. The connections \(C\), specifying the node number in the anterior columns than the target node, represent which nodes are the input to the target node. This connection restriction ensures the feed-forward network structure. The last \((MN + 1)\)-st gene, having only connection information \(C\), represents the output layer, and its type is fixed based on the task.

In the CGP encoding scheme, there may be the nodes that do not connect to the output node; we call these nodes non-active nodes. In contrast, we define the nodes that connect to the output node as active nodes. Note that nodes depicted in the neighboring two columns are not necessarily connected. Thus, the resulting CNN architecture can have a different number of modules (layers) depending on the node connections, i.e., the number of layers is decided by evolutionary algorithms. Note that the maximum depth of a network is \(N\). To control how the number of layers will be chosen, we introduce a hyper-parameter called level-back \(L\), such that nodes given in the \(n\)-th column are allowed to be connected from nodes given in the columns ranging from \(\text{max}(0, n - L)\) to \(n - 1\), where the zeroth column indicates the inputs. If we use smaller \(L\), then the resulting CNNs will tend to be deeper.

2.2 Node Functions

Referring to the modern CNN architectures, we select the highly-functional modules as node functions. We prepare the six types of node functions called ConvBlock, ResBlock, max pooling, average pooling, concatenation, and summation.

The ConvBlock consists of standard convolution processing with a stride of 1 followed by batch normalization [Ioffe and Szegedy, 2015] and rectified linear units (ReLU) [Nair and Hinton, 2010]. The ResBlock is composed of a convolution processing, batch normalization, ReLU, and tensor summation. The ResBlock performs an identity mapping by shortcut connections as described in [He et al., 2016]. For the number \(F\) and receptive field size \(k\) of filters of ConvBlock and ResBlock, we chose them from \(F \in \{32, 64, 128\}\) and \(k \in \{3 \times 3, 5 \times 5\}\), respectively.

The max and average poolings perform a max and average operation, respectively, over the local neighbors of the feature maps. We use the pooling with the \(2 \times 2\) receptive field size and the stride of 2.

The concatenation function concatenates two feature maps in the channel dimension. If the input feature maps to be concatenated have different numbers of rows or columns, we down-sample the larger feature maps by max pooling so that they become the same sizes of the inputs. The summation performs the element-wise addition of two feature maps, channel by channel. In the same way as the concatenation, if the input feature maps to be added have different numbers of rows or columns, we down-sample the larger feature maps by max pooling. In addition, if the inputs have different numbers of channels, we pad the smaller feature maps with zeros for increasing channels. In Figure 1, the summation node performs max pooling to the first input so as to get the same input tensor sizes.

The output node represents the softmax function with the number of classes. The outputs fully connect to all elements of the input.
2.3 Evolutionary Algorithm

We use a point mutation as the genetic operator in the same way 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 the possibility of affecting only non-active nodes. In that case, the phenotype does not change by the mutation and does not require a fitness evaluation again. The fitness evaluation of the CNN architectures is so expensive because it requires the training of the CNN. To use the computational resource efficiently, we apply the mutation operator until at least one active node changes for reproducing the candidate solution. We call this mutation a forced mutation. Moreover, to maintain a neutral drift, we test our method on the image classification task using the 50,000 images for the validation set of the CGP architectures.

We run our method for three times on each dataset and report the classification performance. We compare the classification performance of our method with the state-of-the-art methods and summarize the classification error rates in Table 1. We refer to the architectures designed by our method as CGP-CNN. For instance, CGP-CNN (ConvSet) means the proposed method with the node function set of ConvSet. The models, Network in Network, VGG, ResNet, FractalNet, and Wide ResNet are hand-crafted CNN architectures, whereas the CoDeepNEAT, MetaQNN, Genetic CNN, Large-scale Evolution, and Neural Architecture Search are the models constructed by the architecture search method.

As can be seen in Table 1, the error rates of our methods are competitive with the state-of-the-art methods. In particular, CGP-CNN (ResSet) shows good performance, and the architectures constructed by using our method have a good balance between classification errors, the numbers of parameters and GPUs. The Neural Architecture Search achieved the best error rate on the CIFAR-10 dataset, but this method used 800 GPUs for the architecture search. Our method can find a competitive architecture with a reasonable machine resource.

Examples of CNN architectures designed by our method are shown in [Suganuma et al., 2017a; 2017b]. We can observe that these architectures are quite different; the concatenation and summation nodes are not frequently used in CGP-CNN (ResSet), whereas these nodes are frequently used in...
<table>
<thead>
<tr>
<th>Model</th>
<th># params</th>
<th># GPUs</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network in Network [Lin et al., 2014]</td>
<td>–</td>
<td>–</td>
<td>8.81</td>
<td>35.68</td>
</tr>
<tr>
<td>VGG [Simonyan and Zisserman, 2015]</td>
<td>15.2M</td>
<td>2</td>
<td>7.94</td>
<td>33.45</td>
</tr>
<tr>
<td>ResNet [He et al., 2016]</td>
<td>1.7M</td>
<td>2</td>
<td>6.61</td>
<td>32.40</td>
</tr>
<tr>
<td>FractalNet [Larsson et al., 2017]</td>
<td>38.6M</td>
<td>2</td>
<td>5.22</td>
<td>23.30</td>
</tr>
<tr>
<td>Wide ResNet [Zagoruyko and Komodakis, 2016]</td>
<td>36.5M</td>
<td>2</td>
<td>4.00</td>
<td>19.25</td>
</tr>
<tr>
<td>CoDeepNEAT [Mikkulainen et al., 2017]</td>
<td>–</td>
<td>–</td>
<td>7.3</td>
<td>–</td>
</tr>
<tr>
<td>MetaQNN [Baker et al., 2017]</td>
<td>3.7M</td>
<td>10</td>
<td>6.92</td>
<td>27.14</td>
</tr>
<tr>
<td>Genetic CNN [Xie and Yuille, 2017]</td>
<td>–</td>
<td>10</td>
<td>7.10</td>
<td>29.03</td>
</tr>
<tr>
<td>Large-Scale Evolution [Real et al., 2017]</td>
<td>5.4M</td>
<td>250</td>
<td>5.40</td>
<td>–</td>
</tr>
<tr>
<td>Large-Scale Evolution [Real et al., 2017]</td>
<td>40.4M</td>
<td>250</td>
<td>–</td>
<td>23.0</td>
</tr>
<tr>
<td>Neural Architecture Search [Zoph and Le, 2017]</td>
<td>37.4M</td>
<td>800</td>
<td>3.65</td>
<td>–</td>
</tr>
<tr>
<td>CGP-CNN (ConvSet)</td>
<td>1.5M</td>
<td>2</td>
<td>5.92 (6.70)</td>
<td>–</td>
</tr>
<tr>
<td>CGP-CNN (ConvSet)</td>
<td>2.0M</td>
<td>2</td>
<td>–</td>
<td>26.7 (27.9)</td>
</tr>
<tr>
<td>CGP-CNN (ResSet)</td>
<td>1.7M</td>
<td>2</td>
<td>5.01 (6.00)</td>
<td>–</td>
</tr>
<tr>
<td>CGP-CNN (ResSet)</td>
<td>4.6M</td>
<td>2</td>
<td>–</td>
<td>26.5 (27.6)</td>
</tr>
</tbody>
</table>

Table 1: Comparison of the error rates (%) and the number of learnable weight parameters on the CIFAR-10 and 100 datasets. We run the proposed method for three times for each dataset and report the classification errors in the format of “best (mean).” We refer to the architectures constructed by the proposed method as CGP-CNN. In CGP-CNN, the numbers of learnable weight parameters of the best architecture are reported. The values of other models except for VGG and ResNet for CIFAR-100 are referred from the literature.

4 Related Work

Some architecture search methods based on either EC [Mikkulainen et al., 2017; Real et al., 2017; Xie and Yuille, 2017] or RL [Baker et al., 2017; Zoph and Le, 2017] for DNNs have been proposed at nearly the same time with our work. Since these approaches require much computational cost in general, the recent works concentrate on the reduction of computational cost and improvement of architecture search efficiency [Cai et al., 2018; Liu et al., 2017; 2018; Pham et al., 2018; Real et al., 2018; Zhong et al., 2017; Zoph et al., 2017]. For instance, Cai et al. [Cai et al., 2018] specify an initial architecture by an existing one, and the actions, deepening the architecture by adding a layer or widening an existing layer by replacing an existing layer with a wider layer, are applied to the initial architecture to generate a new architecture. Pham et al. [Pham et al., 2018] have introduced the parameter sharing across models during the architecture search and achieved a test error of 2.89% on the CIFAR-10 dataset with one GPU.

Most methods of the CNN architecture search are applied to image classification tasks such as using CIFAR-10. They, however, can be naturally applied to other computer vision tasks and DNNs. The architecture search method explained in this paper has applied to automatically construct the auto-encoder typed CNNs for the image denoising and inpainting tasks [Suganuma et al., 2018]. The result shows that the designed architectures outperform the existing state-of-the-art hand-crafted architectures on both denoising and inpainting tasks.

5 Conclusion

In this work, we have attempted to take a GP-based approach for designing the CNN architectures and have verified its potential. The proposed method designs the CNN architectures based on CGP and adopts the highly-functional modules, such as convolutional blocks and tensor operations, for searching for the adequate architectures efficiently. We have constructed the CNN architecture for the image classification task with the CIFAR-10 and CIFAR-100 datasets. The experimental result showed that the proposed method can automatically find competitive CNN architectures compared with the state-of-the-art models.

One direction of future work is to develop the evolutionary algorithm to reduce the computational cost of the architecture design, e.g., increasing the training data for the neural network as the generation progresses. Another future work is to apply the proposed method to other image datasets and tasks.

References


