Chapter 7
Designing Convolutional Neural Network Architectures Using Cartesian Genetic Programming

Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao

Abstract Convolutional neural networks (CNNs), among the deep learning models, are making remarkable progress in a variety of computer vision tasks, such as image recognition, restoration, and generation. The network architecture in CNNs should be manually designed in advance. Researchers and practitioners have developed various neural network structures to improve performance. Despite the fact that the network architecture considerably affects the performance, the selection and design of architectures are tedious and require trial-and-error because the best architecture depends on the target task and amount of data. Evolutionary algorithms have been successfully applied to automate the design process of CNN architectures. This chapter aims to explain how evolutionary algorithms can support the automatic design of CNN architectures. We introduce a method based on Cartesian genetic programming (CGP) for the design of CNN architectures. CGP is a form of genetic programming and searches the network-structured program. We represent the CNN architecture via a combination of pre-defined modules and search for the high-performing architecture based on CGP. The method attempts to find better architectures by repeating the architecture generation, training, and evaluation. The effectiveness of the CGP-based CNN architecture search is demonstrated through two types of computer vision tasks: image classification and image restoration. The experimental result for image classification shows that the method can find a well-performing CNN architecture. For the experiment on image restoration tasks, we show that the method can find a simple yet high-performing architecture of a convolutional autoencoder that is a type of CNN.

M. Suganuma
Tohoku University, RIKEN Center for AIP, Sendai, Miyagi, Japan
e-mail: suganuma@vision.is.tohoku.ac.jp

S. Shirakawa (✉) · T. Nagao
Yokohama National University, Yokohama, Kanagawa, Japan
e-mail: shirakawa-shinichi-bg@ynu.ac.jp; nagao@ynu.ac.jp

© Springer Nature Singapore Pte Ltd. 2020

H. Iba, N. Noman (eds.), *Deep Neural Evolution*, Natural Computing Series,
https://doi.org/10.1007/978-981-15-3685-4_7

7.1 Introduction

Many types of CNN architecture have been developed by researchers during the last few years aiming at achieving good scores on computer vision tasks. Despite the success of CNNs, a question remains given recent developments: *what CNN architectures are good and how can we design such architectures?* One possible direction to address this question is neural architecture search (NAS) [5], in which CNN architectures are automatically designed by an algorithm such as evolutionary computation and reinforcement learning to maximize performance on targeted tasks. NAS can automate the design process of neural networks and aids in reducing the trial-and-error of developers.

This chapter is based on the works of [34–36] and explains a genetic programming-based approach to automatically design CNN architectures. In the next section, we briefly review NAS methods by categorizing them into three approaches: evolutionary computation, reinforcement learning, and gradient-descent-based approaches. Then, we describe the Cartesian genetic programming (CGP)-based NAS method for a CNN, which is categorized as an evolutionary-computation-based approach. In Sect. 7.3, the CGP-based architecture search method for image classification, termed CGP-CNN, is explained. In Sect. 7.4, the CGP-based architecture search method is extended to the convolutional autoencoder (CAE), a type of CNN, for image restoration.

7.2 Progress of Neural Architecture Search

Automatic design of neural network structures is an active topic initially presented several decades ago, e.g., [30, 33, 45]. These methods optimize the connection weights and/or network structure of low-level neurons using an evolutionary algorithm, and are also known as evolutionary neural networks. These traditional structure optimization methods target relatively small neural networks whereas recent deep neural networks, including CNNs, have greater than one million parameters though the architectures are still designed by human experts. Aiming at the automatic design of deep neural network architectures, various architecture search methods have been developed since 2017. Nowadays, the automatic design method of deep neural network architectures is termed a neural architecture search (NAS) [5].

To address large-scale architectures, neural network architectures are designed using a certain search method but the network weights are optimized by a stochastic gradient descent method through back-propagation. Evolutionary algorithms are often used to search the architectures. Real et al. [28] optimized large-scale neural networks using an evolutionary algorithm and achieved better performance than that of modern CNNs in image classification tasks. In this method, they represent the CNN architecture as a graph structure and optimize it via the evolutionary algorithm.

The connection weights of the reproduced architecture are optimized by stochastic gradient descent as typical neural network training; the accuracy for the architecture evaluation dataset is assigned as the fitness. Miikkulainen et al. [20] proposed a method termed CoDeepNEAT that is an extended version of NeuroEvolution of Augmenting Topologies (NEAT). This method designs the network architectures using blueprints and modules. The blueprint chromosome is a graph in which each node has a pointer to a particular module species. Each module chromosome is a graph that represents a small DNN. Specifically, each node in the blueprint is replaced with a module selected from a particular species to which that node points. During the evaluation phase, the modules and blueprints are combined to generate assembled networks and the networks are evaluated. Xie and Yuille [42] designed CNN architectures using the genetic algorithm with a binary string representation. They proposed a method for encoding a network structure in which the connectivity of each layer is defined by a binary string representation. The type of each layer, number of channels, and size of a receptive field are not evolved in this method. The method explained in this chapter is also an evolutionary-algorithm-based NAS. Different from the aforementioned methods, it optimizes the architecture based on genetic programming and adopts well-designed modules as the node function.

Another approach is to use reinforcement learning to search the neural architectures. In [49], a recurrent neural network (RNN) was used to generate neural network architectures. The RNN was trained with policy-gradient-based reinforcement learning to maximize the expected accuracy on a learning task. Baker et al. [2] proposed a meta-modeling approach based on reinforcement learning to produce CNN architectures. A Q-learning agent explores and exploits a space of model architectures with an ϵ-greedy strategy and experience replay.

As these methods need neural network training to evaluate the candidate architectures, they often require a considerable computational cost. For instance, the work of [49] used 800 graphics processing units (GPUs). To reduce the computational cost of NAS is an active topic. A promising approach is jointly optimizing the architecture parameter and connection weights. This approach, termed one-shot NAS (aka weight sharing), finds better architecture during single training. In one-shot NAS, the non-differentiable objective function consisting of discrete architecture parameters is transformed into a differentiable objective by continuous [17, 43] or stochastic relaxation [1, 27, 31]; both the architecture parameters and connection weights are optimized by gradient-based optimizers.

7.3 Designing CNN Architecture for Image Classification

In this section, we introduce the architecture search method based on CGP for image classification. We term the method CGP-CNN. In CGP-CNN, we directly encode the CNN architectures based on CGP and use highly functional modules as node functions. The CNN architecture defined by CGP is trained by a stochastic gradient descent using a model training dataset and assigns the fitness value based on the

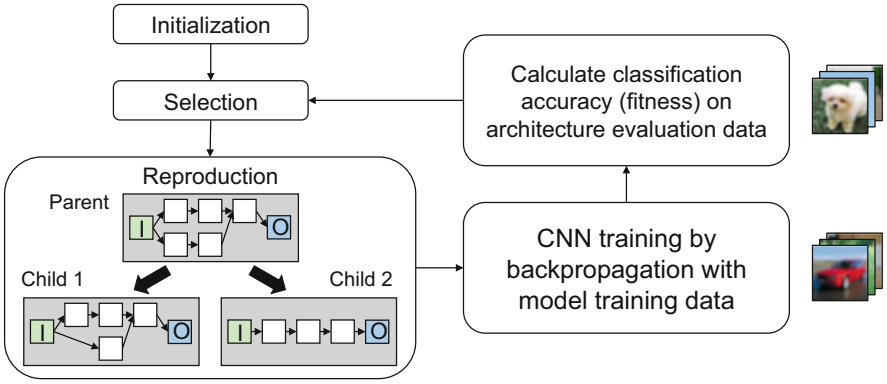

Fig. 7.1 Overview of CGP-CNN. The method represents the CNN architectures based on CGP. The CNN architecture is trained on a learning task and assigned a fitness based on the accuracies of the trained model for the architecture evaluation dataset. The evolutionary algorithm searches for better architectures

accuracies of another training dataset (i.e. the architecture evaluation dataset). Then, the architecture is optimized to maximize the accuracy of the architecture evaluation dataset using the evolutionary algorithm. Figure 7.1 shows an overview of CGP-CNN. In the following, we describe the network representation and the evolutionary algorithm used in CGP-CNN.

7.3.1 Representation of CNN Architectures

For CNN architecture representation, we use the CGP encoding scheme that represents an architecture of CNNs as directed acyclic graphs with a two-dimensional grid. CGP was proposed as a general form of genetic programming in [22]. The graph corresponding to a phenotype is encoded to a string termed a genotype and optimized using the evolutionary algorithm.

Let us assume that the grid has N_r rows by N_c columns; then, the number of intermediate nodes is $N_r \times N_c$ and the number of inputs and outputs depends on the task. The genotype consists of a string of integers of a fixed length and each gene determines the function type of the node and the connection between nodes. The c-th column's node is only allowed to be connected from the $(c-1)$ to $(c-l)$-th column's nodes, in which l is termed a level-back parameter. Figure 7.2 shows an example of the genotype, phenotype, and corresponding CNN architecture. As seen in Fig. 7.2, the CGP encoding scheme has a possibility that not all of the nodes are connected to the output nodes (e.g., node No. 5 in Fig. 7.2). We term these nodes *inactive nodes*. Whereas the genotype in CGP is a fixed-length representation, the number of nodes in the phenotypic network varies because of the inactive nodes.

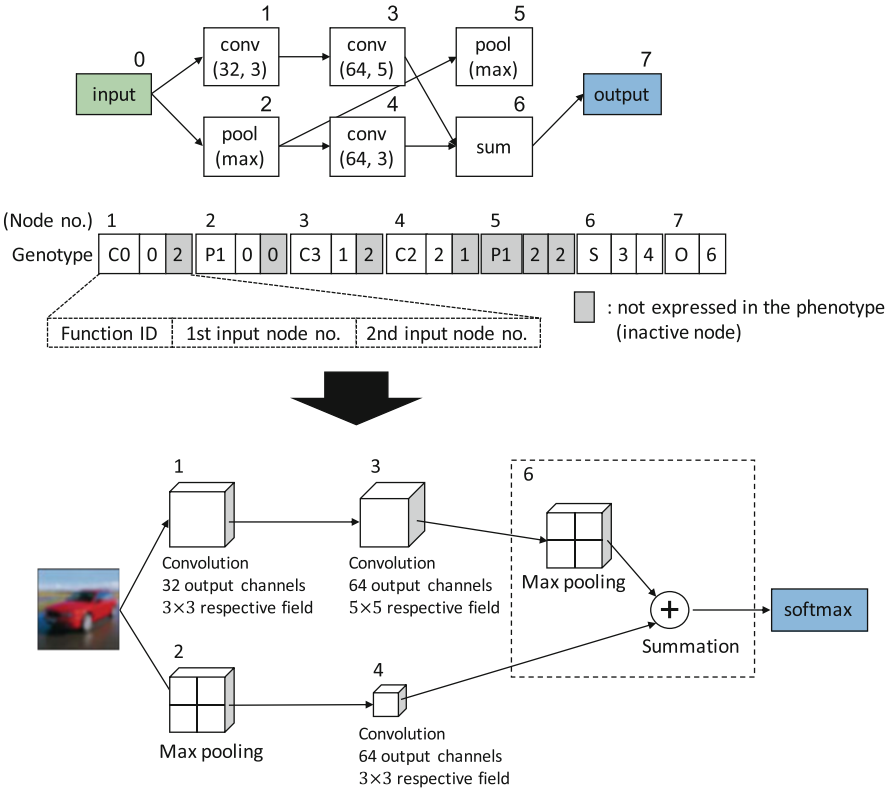

Fig. 7.2 Examples of a genotype and phenotype. The genotype (left) defines the CNN architecture (right). Node No. 5 on the left is inactive and does not appear in the path from the inputs to the outputs. The summation node applies max pooling to downsample the first input to the same size as the second input

This is a desirable feature because the number of layers can be determined using the evolutionary algorithm.

Referring to modern CNN architectures, we select the highly functional modules as the node function. The frequently used processes in the CNN are convolution and pooling; the convolution processing uses local connectivity and spatially shares the learnable weights and the pooling is nonlinear downsampling. We prepare the six types of node functions, termed ConvBlock, ResBlock, max pooling, average pooling, concatenation, and summation. These nodes operate on the three-dimensional (3-D) tensor (also known as the feature map) defined by the dimensions of the row, column, and channel.

The ConvBlock consists of a convolutional layer with a stride of one followed by the batch normalization [10] and the rectified linear unit (ReLU) [23]. To maintain the size of the input, we pad the input with zero values around the border before the convolutional operation. Therefore, the ConvBlock takes the $M \times N \times C$ tensor

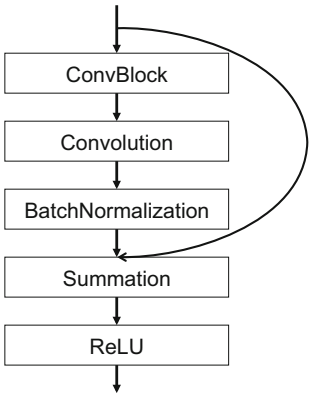

Fig. 7.3 The ResBlock architecture

as an input and produces the $M \times N \times C'$ tensor, where M, N, C, and C' are the number of rows, columns, input channels, and output channels, respectively. We prepare several ConvBlocks with different output channels and receptive field sizes (kernel sizes) in the function set of CGP.

As shown in Fig. 7.3, the ResBlock is composed of the ConvBlock, batch normalization, ReLU, and tensor summation. The ResBlock is a building block of the modern successful CNN architectures, e.g., [8, 47] and [13]. Following this recent trend of human architecture design, we decided to use ResBlock as the building block in CGP-CNN. The ResBlock performs identity mapping via the shortcut connection as described in [8]. The row and column sizes of the input are preserved in the same manner as those of the ConvBlock after convolution. As shown in Fig. 7.3, the output feature maps of the ResBlock are calculated via the ReLU activation and the summation with the input. The ResBlock takes the $M \times N \times C$ tensor as an input and produces the $M \times N \times C'$ tensor. We prepare several ResBlocks with different output channels and receptive field sizes (kernel sizes) in the function set of CGP.

The max and average poolings perform the maximum and average operations, respectively, over the local neighbors of the feature maps. We use the pooling with a 2×2 receptive field size and a stride of two. The pooling layer takes the $M \times N \times C$ tensor and produces the $M' \times N' \times C$ tensor, where $M' = \lfloor M/2 \rfloor$ and $N' = \lfloor N/2 \rfloor$.

The concatenation function takes two feature maps and concatenates them in the channel dimension. When concatenating the feature maps with different numbers of rows and columns, we downsample the larger feature map by max pooling to make them the same sizes as the inputs. Let us assume that we have two inputs of size $M_1 \times N_1 \times C_1$ and $M_2 \times N_2 \times C_2$, then the size of the output feature maps is $\min(M_1, M_2) \times \min(N_1, N_2) \times (C_1 + C_2)$.

The summation performs element-wise summation of two feature maps, channel-by-channel. Similar to the concatenation, when summing the two feature maps with different numbers of rows and columns, we downsample the larger feature map by max pooling. In addition, if the inputs have different numbers of channels, we

Table 7.1 Node functions and abbreviated symbols used in the experiments

Node type	Symbol	Variation
ConvBlock	CB (C', k)	$C' \in \{32, 64, 128\}$
		$k \in \{3 \times 3, 5 \times 5\}$
ResBlock	RB (C', k)	$C' \in \{32, 64, 128\}$
		$k \in \{3 \times 3, 5 \times 5\}$
Max pooling	MP	–
Average pooling	AP	–
Concatenation	Concat	–
Summation	Sum	–

C': Number of output channels
k: Receptive field size (kernel size)

expand the channels of the feature maps with a smaller channel size by filling with zero. Let us assume that we have two inputs of size $M_1 \times N_1 \times C_1$ and $M_2 \times N_2 \times C_2$, then the sizes of the output feature maps are $\min(M_1, M_2) \times \min(N_1, N_2) \times \max(C_1, C_2)$. In Fig. 7.2, the summation node applies the max pooling to downsample the first input to the same size as the second input. By using the summation and concatenation operations, our method can express the shortcut connection or branch layers, such as those used in GoogLeNet [37] and residual network (ResNet) [8].

The output node represents the softmax function to produce a distribution over the target classes. The outputs fully connect to all elements of the input. The node functions used in the experiments are listed in Table 7.1.

7.3.2 Evolutionary Algorithm

Following the standard CGP, we use a point mutation as the genetic operator. The function and the connection of each node randomly change to valid values according to the mutation rate. The fitness evaluation of the CNN architecture involves CNN training and requires approximately 0.5 to 1 h in our setting. Therefore, we need to efficiently evaluate some candidate solutions in parallel at each generation. To efficiently use the computational resource, we repeatedly apply the mutation operator while an active node does not change and obtain the candidate solutions to be evaluated. We term this mutation forced mutation. Moreover, to maintain a neutral drift, which is effective for CGP evolution [21, 22], we modify a parent by neutral mutation if the fitness of the offspring do not improve. The neutral mutation operates only on the genes of inactive nodes without modification of the phenotype. We use the modified $(1+\lambda)$ evolution strategy (with $\lambda = 2$ in the experiment) using the aforementioned artifice. The procedure of our evolutionary algorithm is listed in Algorithm 1.

The $(1 + \lambda)$ evolution strategy, the default evolutionary algorithm in CGP, is an algorithm with fewer strategy parameters: the mutation rate and offspring size. We

Algorithm 1 Evolutionary algorithm for CGP-CNN and CGP-CAE

Input: G (number of generations), r (mutation probability), λ (children size), S (Training set), V (architecture evaluation set).
Initialization: (i) Generate a $parent$, (ii) train the model on the S, and (iii) assign the fitness F_p using the set V.
for $g = 1$ **to** G **do**
  for $i = 1$ **to** λ **do**
    $children_i \leftarrow$ Mutation($parent, r$) # forced mutation
    $model_i \leftarrow$ Train($children_i, S$)
    $fitness_i \leftarrow$ Evaluate($model_i, V$)
  end for
  $best \leftarrow \text{argmax}_{i=1,2,\ldots,\lambda} \{fitness_i\}$
  if $fitness_{best} \geq F_p$ **then**
    $parent \leftarrow children_{best}$
    $F_p \leftarrow fitness_{best}$
  else
    $parent \leftarrow$ Modify($parent, r$) # neutral mutation
  end if
end for
Output: $parent$ (the best architecture found by the evolutionary search).

do not need to expend considerable effort to tune such strategy parameters. Thus, we use the $(1 + \lambda)$ evolution strategy in CGP-CNN.

7.3.3 Experiment on Image Classification Tasks

7.3.3.1 Experimental Setting

We apply CGP-CNN to the CIFAR-10 and CIFAR-100 datasets consisting of 60,000 color images (32×32 pixels) in 10 and 100 classes, respectively. Each dataset is split into a training set of 50,000 images and a test set of 10,000 images. We randomly sample 45,000 examples from the training set to train the CNN and the remaining 5000 examples are used for architecture evaluation (i.e. fitness evaluation of CGP).

To assign the fitness value to the candidate CNN architecture, we train the CNN by stochastic gradient descent (SGD) with a mini-batch size of 128. The softmax cross-entropy loss is used as the loss function. We initialize the weights using the method described in [7] and use the Adam optimizer [11] with an initial learning rate $\alpha = 0.01$ and momentum $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We train each CNN for 50 epochs and use the maximum accuracy of the last 10 epochs as the fitness value. We reduce the learning rate by a factor of 10 at the 30th epoch.

We preprocess the data with pixel-mean subtraction. To prevent overfitting, we use a weight decay with the coefficient 1.0×10^{-4}. We also use data augmentation based on [8]: padding 4 pixels on each size and randomly cropping a 32×32 patch from the padded image or its horizontally flipped image.

Table 7.2 Parameter setting for the CGP-CNN on image classification tasks

Parameters	Values
Mutation rate	0.05
# Offspring (λ)	2
# Rows (N_r)	5
# Columns (N_c)	30
Minimum number of active nodes	10
Maximum number of active nodes	50
Levels-back (l)	10

The parameter setting for CGP is shown in Table 7.2. We use a relatively large number of columns to generate deep architectures. The number of active nodes in the individual of CGP is restricted. Therefore, we apply the mutation operator until the CNN architecture that satisfies the restriction of the number of active nodes is generated. The offspring size λ is two, the same number of GPUs in our experimental machines. We test two node function sets termed ConvSet and ResSet for CGP-CNN. The ConvSet contains ConvBlock, max pooling, average pooling, summation, and concatenation in Table 7.1 and the ResSet contains ResBlock, max pooling, average pooling, summation, and concatenation. The difference between these two function sets is whether the set contains ConvBlock or ResBlock. The number of generations is 500 for ConvSet and 300 for ResSet.

The best CNN architecture from the CGP process is retrained using all 50,000 images in the training set. Then, we compute the test accuracy. We optimize the weights of the obtained architecture for 500 epochs using a different training procedure; we use SGD with a momentum of 0.9, a mini-batch size of 128, and a weight decay of 5.0×10^{-4}. Following the learning rate schedule in [8], we start with a learning rate of 0.01 and set it to 0.1 at the 5th epoch. We reduce it by a factor of 10 at the 250th and 370th epochs. We report the test accuracy at the 500th epoch as the final performance.

We implement CGP-CNN using the Chainer framework [40] (version 1.16.0) and run it on a machine with two NVIDIA GeForce GTX 1080 or two GTX 1080 Ti GPUs. We use a GTX 1080 and 1080 Ti for the experiments on the CIFAR-10 and 100 datasets, respectively. Because of the memory limitation, the candidate CNNs occasionally take up the GPU memory, and the network training process fails because of an out-of-memory error. In this case, we assign a zero fitness to the candidate architecture.

7.3.3.2 Experimental Result

We run CGP-CNN 10 times on each dataset and report the classification errors. We compare the classification performance to the hand-designed CNNs and automatically designed CNNs using the architecture search methods on the CIFAR-10 and 100 datasets. A summary of the classification performances is provided in

Table 7.3 Comparison of the error rates (%), number of learnable weight parameters, and search costs on the CIFAR-10 dataset

Model	# Params	Test error	GPU days
Maxout [6]	–	9.38	–
Network in network [15]	–	8.81	–
VGG [32]	15.2 M	7.94	–
ResNet [8]	1.7 M	6.61	–
FractalNet [14]	38.6 M	5.22	–
Wide ResNet [47]	36.5 M	4.00	–
CoDeepNEAT [20]	–	7.30	–
Genetic CNN [42]	–	7.10	17
MetaQNN [2]	3.7 M	6.92	80–100
Large-scale evolution [28]	5.4 M	5.40	2750
Neural architecture search [49]	37.4 M	**3.65**	16,800–22,400
CGP-CNN (ConvSet)	1.50 M	5.92 (6.48 ± 0.48)	31
CGP-CNN (ResSet)	2.01 M	5.01 (6.10 ± 0.89)	30

The classification error is reported in the format of "best (mean $\pm$ std)." In CGP-CNN, the number of learnable weight parameters of the best architecture is reported. The values of other models are referenced from the literature. The bold value indicates the best test error among the compared models

Tables 7.3 and 7.4. The models, Maxout, Network in Network, VGG, ResNet, FractalNet, and Wide ResNet, are the hand-designed CNN architectures whereas MetaQNN, Neural Architecture Search, Large-Scale Evolution, Genetic CNN, and CoDeepNEAT are the models obtained using the architecture search methods. The values of other models, except for VGG and ResNet on CIFAR-100, are referenced from the literature. We implement the VGG net and ResNet for CIFAR-100 because they were not applied to the dataset in [32] and [8]. The architecture of VGG is identical to that of configuration D in [32]. In Tables 7.3 and 7.4, the number of learnable weight parameters in the models is also listed. In CGP-CNN, the number of learnable weight parameters of the best architecture is reported.

On the CIFAR-10 dataset, the CGP-CNNs outperform most of the hand-designed models and show a good balance between the classification errors and the number of parameters. CGP-CNN (ResSet) shows better performance compared to that of CGP-CNN (ConvSet). Compared to other architecture search methods, CGP-CNN (ConvSet and ResSet) outperforms MetaQNN [2], Genetic CNN [42], and CoDeepNEAT [20]. The best architecture of CGP-CNN (ResSet) outperforms Large-Scale Evolution [28]. The Neural Architecture Search [49] achieved the best error rate, but this method used 800 GPUs and required considerable computational costs to search for the best architecture. Table 7.3 also lists the number of GPU days (the computational time multiplied by the number of GPUs used during the experiments) for the architecture search. As seen, CGP-CNN can find a good architecture at

Table 7.4 Comparison of the error rates (%) and number of learnable weight parameters on the CIFAR-100 dataset

Model	# Params	Test error
Maxout [6]	–	38.57
Network in network [15]	–	35.68
VGG [32]	15.2 M	33.45
ResNet [8]	1.7 M	32.40
FractalNet [14]	38.6 M	23.30
Wide ResNet [47]	36.5 M	**19.25**
CoDeepNEAT [20]	–	–
Neural architecture search [49]	37.4 M	–
Genetic CNN [42]	–	29.03
MetaQNN [2]	3.7 M	27.14
Large-scale evolution [28]	40.4 M	23.0
CGP-CNN (ConvSet)	2.01 M	26.7 (28.1 $\pm$ 0.83)
CGP-CNN (ResSet)	4.60 M	25.1 (26.8 $\pm$ 1.21)

The classification errors are reported in the format of "best (mean $\pm$ std)." In CGP-CNN, the number of learnable weight parameters of the best architecture is reported. The values of other models except for VGG and ResNet are referenced from the literature. The bold value indicates the best test error among the compared models

a reasonable computational cost. We assume that CGP-CNN, particularly with ResSet, could reduce the search space and find better architectures in an early iteration by using the highly functional modules. The CIFAR-100 dataset is a very challenging task because there are many classes. CGP-CNN finds the competitive network architectures within a reasonable computational time. Even though the obtained architecture is not at the same level as the state-of-the-art architectures, it shows a good balance between the classification errors and number of parameters.

The error rates of the architecture search methods (not only CGP-CNN) do not reach those of Wide ResNet, a human-designed architecture. However, these human-designed architectures are developed with the expenditure of tremendous human effort. An advantage of architecture search methods is that they can automatically find a good architecture for a new dataset. Another advantage of CGP-CNN is that the number of weight parameters in the discovered architectures is less than that in the human-designed architectures, which is beneficial when we want to implement CNN on a mobile device. Note that we did not introduce any criteria for the architecture complexity in the fitness function. It might be possible to find more compact architectures by introducing the penalty term into the fitness function, which is an important research direction, such as in [4, 29, 39].

Figure 7.4 shows the examples of the CNN architectures obtained by CGP-CNN (ConvSet and ResSet). Figure 7.4 shows the complex architectures that are difficult to manually design. Specifically, CGP-CNN (ConvSet) uses the summation and concatenation nodes leading to a wide network and allowing for the formation of skip connections. Therefore, the CGP-CNN (ConvSet) architecture is wider than

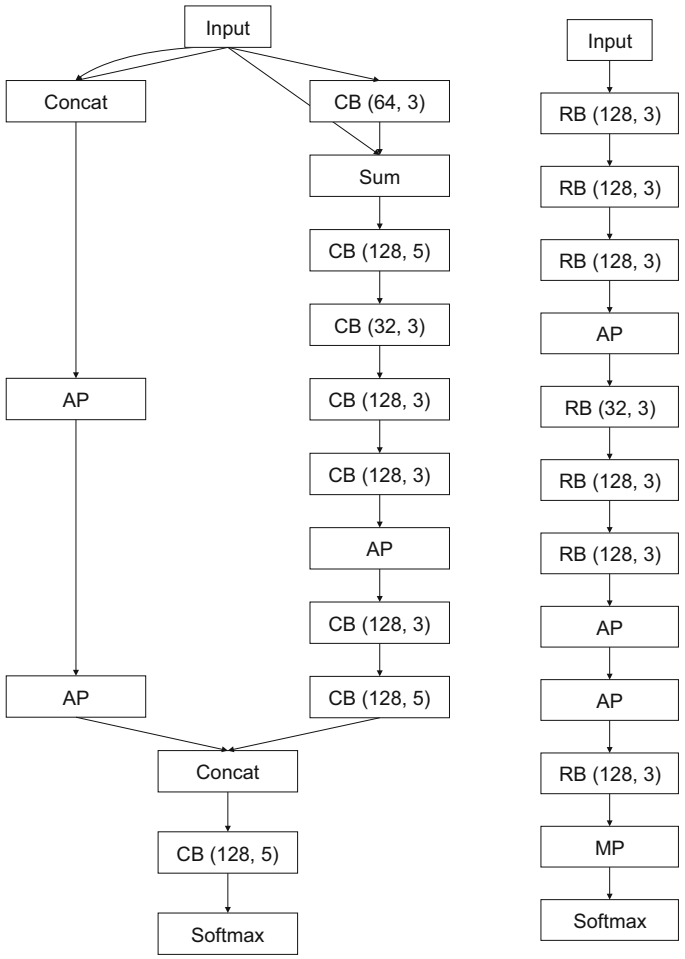

Fig. 7.4 CNN architectures obtained by CGP-CNN with ConvSet (left) and ResSet (right) on the CIFAR-10 dataset

that of CGP-CNN (ResSet). Additionally, we also observe that CGP-CNN (ResSet) has a similar structure to that of ResNet [8]. ResNet consists of a series of two types of modules: a module with several convolutions and shortcut connections without downsampling and a downsampling convolution with a stride of 2. Although CGP-CNN cannot downsample in the ConvBlock and ResBlock, we see that CGP-CNN (ResSet) uses a pooling layer as an alternative to the downsampling convolution. We can say that CGP-CNN can find an architecture similar to that designed by human experts.

7.4 Designing CNN Architectures for Image Restoration

In this section, we apply the CGP-based architecture search method to an image restoration task of recovering a clean image from its degraded version. We term this method CGP-CAE. Recently, learning-based approaches based on CNNs have been applied to image restoration tasks and have significantly improved the state-of-the-art performance. Researchers have approached this problem mainly from three directions: designing new network architectures, loss functions, and training strategies. In this section, we focus on designing a new network architecture for image restoration and report that simple convolutional autoencoders (CAEs) designed by evolutionary algorithms can outperform existing image restoration methods which are designed manually.

7.4.1 Search Space of Network Architectures

In this work, we consider CAEs that are built only on convolutional layers with downsampling and skip connections. In addition, we use *symmetric* CAEs such that their first half (encoder part) is symmetric to the second half (decoder part). The final layer is attached to top of the decoder part to obtain images of fixed channels (i.e. single-channel grayscale or three-channel color images), for which either one or three filters of 3×3 size are used. Therefore, specifying the encoder part of a CAE solely determines its entire architecture. The encoder part can have an arbitrary number of convolutional layers up to a specified maximum, which is selected by the evolutionary algorithm. Each convolutional layer can have an arbitrary number and size of filters, and is followed by ReLU [23]. In addition, each layer can have an optional skip connection [8, 18] that connects the layer to its mirrored counterpart in the decoder part. Specifically, the output feature maps (obtained after ReLU) of the layer are passed to and are added element-wise to the output feature maps (obtained before ReLU) of the counterpart layer. We can use additional downsampling after each convolutional layer depending on the task. Whether to use downsampling is determined in advance and thus it is not selected by the architectural search, as explained later.

7.4.2 Representation of CAE Architectures

Following [34], we represent architectures of CAEs via a directed acyclic graph which is defined on a two-dimensional grid. This graph is optimized by the evolutionary algorithm, in which the graph is termed a phenotype and is encoded by a data structure termed a genotype.

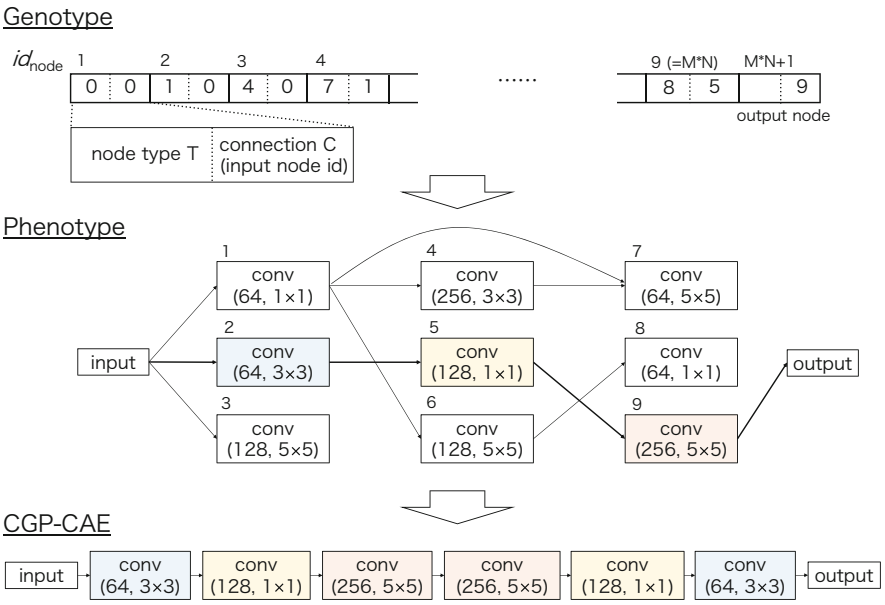

Fig. 7.5 An example of a genotype and a phenotype of CGP-CAE. A phenotype is a graph representation of a network architecture and a genotype encodes a phenotype. They encode only the encoder part of a CAE and its decoder part is automatically created such that it is symmetrical to the encoder part. In this example, the phenotype is defined on a grid of three rows and three columns

Figure 7.5 shows an example of a genotype and a phenotype of CGP-CAE. Each node of the graph represents a convolutional layer followed by a ReLU in a CAE. An edge connecting two nodes represents the connectivity of the two corresponding layers. The graph has two additional special nodes termed input and output nodes. The former represents the input layer of the CAE and the latter represents the output of the encoder part, or equivalently the input of the decoder part of the CAE. As the input of each node is connected to at most one node, there is a single unique path starting from the input node and ending at the output node. This unique path identifies the architecture of the CAE, as shown in the middle row of Fig. 7.5. Note that the nodes depicted in the neighboring two columns are not necessarily connected. Thus, the CAE can have a different number of layers depending on how the nodes are connected. Because the maximum number of layers (of the encoder part) of the CAE is $N_{\max}$, the total number of layers is $2N_{\max} + 1$ including the output layer. To control how the number of layers will be chosen, we introduce a hyper-parameter termed level-back l, such that nodes given in the c-th column are allowed to be connected from nodes given in the columns ranging from $c - l$ to $c - 1$. If we use a smaller l, then the resulting CAEs will tend to be deeper.

A genotype encodes a phenotype and is manipulated by the evolutionary algorithm. The genotype encoding a phenotype with N_r rows and N_c columns has

$N_r N_c + 1$ genes, each of which represents attributes of a node with two integers (i.e. type and connection). The type specifies the number F and size k of the filters of the node, and whether the layer has skip connections or not, by an integer encoding their combination. The connection specifies the node that is connected to the input of this node. The last $(N_r N_c + 1)$-st gene represents the output node that stores only the connection determining the node connected to the output node. An example of a genotype is shown in the top row of Fig. 7.5, where $F \in \{64, 128, 256\}$ and $k \in \{1 \times 1, 3 \times 3, 5 \times 5\}$.

We use the same evolutionary algorithm as used in the previous section to perform a search in the architecture space (see Algorithm 1).

7.4.3 Experiment on Image Restoration Tasks

We conducted experiments to test the effectiveness of CGP-CAE. We chose two tasks: image inpainting and denoising.

7.4.3.1 Experimental Settings

Inpainting

We followed the procedures suggested in [46] for experimental design. We used three benchmark datasets: the CelebFaces Attributes Dataset (CelebA) [16], the Stanford Cars Dataset (Cars) [12], and the Street View House Numbers (SVHN) [24]. The CelebA contains 202,599 images, from which we randomly selected 100,000, 1000, and 2000 images for training, architecture evaluation, and testing, respectively. All images were cropped to properly contain the entire face and resized to 64×64 pixels. For Cars and SVHN, we used the provided training and testing split. The images of Cars were cropped according to the provided bounding boxes and resized to 64×64 pixels. The images of SVHN were resized to 64×64 pixels.

We generated images with missing regions of the following three types: a central square block mask (*Center*), random pixel masks such that 80% of all the pixels were randomly masked (*Pixel*), and half-image masks such that a randomly chosen vertical or horizontal half of the image was masked (*Half*). For the latter two, a mask was randomly generated for each training mini-batch and each test image.

Considering the nature of this task, we consider CAEs endowed with downsampling. To be specific, the same counts of downsampling and upsampling with stride $= 2$ were employed such that the entire network had a symmetric hourglass shape. For simplicity, we used a skip connection and downsampling in an exclusive manner; in other words, every layer (in the encoder part) employed either a skip connection or downsampling.

Denoising
We followed the experimental procedures described in [18, 38]. We used grayscale 300 and 200 images belonging to the BSD500 dataset [19] to generate training and test images, respectively. For each image, we randomly extracted 64×64 patches, to each of which Gaussian noise with different $\sigma = 30, 50$, and 70 are added. As utilized in the previous studies, we trained a single model for all different noise levels.

For this task, we used CAE models without downsampling following the previous studies [18, 38]. We zero-padded the input feature maps computed in each convolution layer not to change the size of the input and output feature space of the layer.

Configurations of the Architectural Search
For the evolutionary algorithm, we chose the mutation probability as $r = 0.1$, number of children as $\lambda = 4$, and number of generations as $G = 250$. For the phenotype, we used the graph with $N_r = 3$, $N_c = 20$, and level-back $l = 5$. For the number F and size k of the filters at each layer, we chose them from $\{64, 128, 256\}$ and $\{1 \times 1, 3 \times 3, 5 \times 5\}$, respectively. During an evolution process, we trained each CAE for $I = 20{,}000$ iterations with a mini-batch of size $b = 16$. We set the learning rate of the ADAM optimizer to be 0.001. For the training loss, we used the mean squared error (MSE) between the restored images and their ground truths:

$$L(\theta_D) = \frac{1}{|S|} \sum_{i=1}^{|S|} ||D(y_i; \theta_D) - x_i||_2^2, \tag{7.1}$$

where the CAE and its weight parameters are D and θ_D, respectively; S is the training set, x_i is a ground truth image, and y_i is a corrupted image. For the fitness function of the evolutionary algorithm, we use the peak signal-to-noise ratio (PSNR) of which the higher value indicates the better image restoration.

Following completion of the evolution process, we fine-tuned the best CAE using the training set of images for additional 500,000 iterations, in which the learning rate is reduced by a factor of 10 at the 200,000 and 400,000 iterations. We then calculated its performance using the test set of images. We implemented CGP-CAE using PyTorch [25] and performed the experiments using four P100 GPUs. Execution of the evolutionary algorithm and the fine-tuning of the best model took approximately 3 days for the inpainting tasks and 4 days for the denoising tasks.

7.4.3.2 Results of the Inpainting Tasks

We use two standard evaluation measures, the PSNR and structural similarity index (SSIM) [41], to evaluate the restored images. Higher values of these measures indicate better image restoration.

Table 7.5 Inpainting results

Dataset	Type	PSNR					SSIM				
		Rand	BASE	CE	SII	CGP-CAE	Rand	BASE	CE	SII	CGP-CAE
CelebA	Center	15.3	27.1	28.5	19.4	**29.9**	0.740	0.883	0.912	0.907	**0.934**
	Pixel	25.5	27.5	22.9	22.8	**27.8**	0.766	0.836	0.730	0.710	**0.887**
	Half	12.7	11.8	19.9	13.7	**21.1**	0.549	0.604	0.747	0.582	**0.771**
Cars	Center	17.1	19.5	19.6	13.5	**20.9**	0.704	0.767	0.767	0.721	**0.846**
	Pixel	17.0	19.2	15.6	18.9	**19.5**	0.533	0.679	0.408	0.412	**0.738**
	Half	13.0	11.6	14.8	11.1	**16.2**	0.511	0.541	0.576	0.525	**0.610**
SVHN	Center	23.5	29.9	16.4	19.0	**33.3**	0.819	0.895	0.791	0.825	**0.953**
	Pixel	29.0	40.1	30.5	33.0	**40.4**	0.687	0.899	0.888	0.786	**0.969**
	Half	11.3	12.9	21.6	14.6	**24.8**	0.574	0.617	0.756	0.702	**0.848**

Comparison of two baseline architectures (RAND and BASE), Context Autoencoder (CE) [26], Semantic Image Inpainting (SII) [46], and CAEs designed by CGP-CAE using three datasets and three masking patterns. The bold values indicate the best performance among the compared architectures

As previously mentioned, we follow the experimental procedure employed in [46]. In the paper, the authors reported the performances of their proposed method, Semantic Image Inpainting (SII), and Context Autoencoder (CE) [26]. However, we found that CE can provide considerably better results than those reported in [46] in terms of PSNR. Thus, we report here PSNR and SSIM values for CE that we obtained by running the code provided by the authors.[1] To calculate SSIM values of SII, which were not reported in [46], we run the authors' code[2] for SII.

To further validate the effectiveness of the evolutionary search, we evaluate two baseline architectures; an architecture generated by a random search (RAND) and an architecture with same depth as the best-performing architecture found by CGP-CAE but having a constant number (64) of fixed size (3×3) filters in each layer with a skip connection (BASE). In the random search, we generate 10 architectures at random in the same search space as ours and report their average PSNR and SSIM values. All other experimental setups are the same.

Table 7.5 shows the PSNR and SSIM values obtained using five methods on three datasets and three masking patterns. We run the evolutionary algorithm three times and report the average accuracy values of the three optimized CAEs. As shown, CGP-CAE outperforms the other four methods for each of the dataset-mask combinations. Notably, CE and SII use mask patterns for inference. To be specific, their networks estimate only pixel values of the missing regions specified by the provided masks, and then they are merged with the unmasked regions of clean pixels. Thus, the pixel intensities of the unmasked regions are identical to their ground truths. On the other hand, CGP-CAE does not use masks yet outputs

[1] https://github.com/pathak22/context-encoder.

[2] https://github.com/moodoki/semantic_image_inpainting.

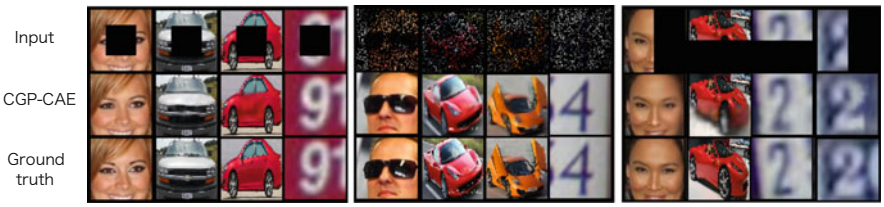

Fig. 7.6 Examples of inpainting results obtained by CGP-CAE (CAEs designed by the evolutionary algorithm)

Table 7.6 Denoising results on BSD200

	PSNR					SSIM				
Noise σ	Rand	BASE	RED	MemNet	CGP-CAE	Rand	BASE	RED	MemNet	CGP-CAE
30	27.25	27.00	27.95	28.04	**28.23**	0.7491	0.7414	0.8019	**0.8053**	0.8047
50	25.11	24.88	25.75	25.86	**26.17**	0.6468	0.6229	0.7167	0.7202	**0.7255**
70	23.50	23.22	24.37	24.53	**24.83**	0.5658	0.5349	0.6551	0.6608	**0.6636**

Comparison of results of two baseline architectures (RAND and BASE), RED [18], MemNet [38], and CGP-CAE. The bold values indicate the best performance among the compared architectures

complete images such that the missing regions are hopefully correctly inpainted. We then calculate the PSNR of the output image against the ground truth without identifying missing regions. This difference should help CE and SII to achieve high PSNR and SSIM values, but nevertheless CGP-CAE performs better.

Sample inpainted images obtained by CGP-CAE along with the masked inputs and the ground truths are shown in Fig. 7.6. It is observed that overall CGP-CAE stably performs; the output images do not have large errors for all types of masks. It performs particularly well for random pixel masks (the middle column of Fig. 7.6); the images are realistic and sharp. It is also observed that CGP-CAE tends to yield less sharp images for those with a filled region of missing pixels. However, CGP-CAE can accurately infer their contents, as shown in the examples of inpainting images of numbers (the rightmost column of Fig. 7.6).

7.4.3.3 Results of the Denoising Task

We compare CGP-CAE to two baseline architectures (i.e. RAND and BASE described in Sect. 7.4.3.2) and two state-of-the-art methods RED [18] and MemNet [38]. Table 7.6 shows the PSNR and SSIM values for three versions of the BSD200 test set with different noise levels $\sigma = 30, 50$, and 70, in which the performance values of RED and MemNet are obtained from [38]. CGP-CAE again achieves the best performance for all cases except for a single case (MemNet for $\sigma = 30$). It is worth noting that the networks of RED and MemNet have 30 and 80 layers, respectively, whereas our best CAE has only 15 layers (including the decoder part

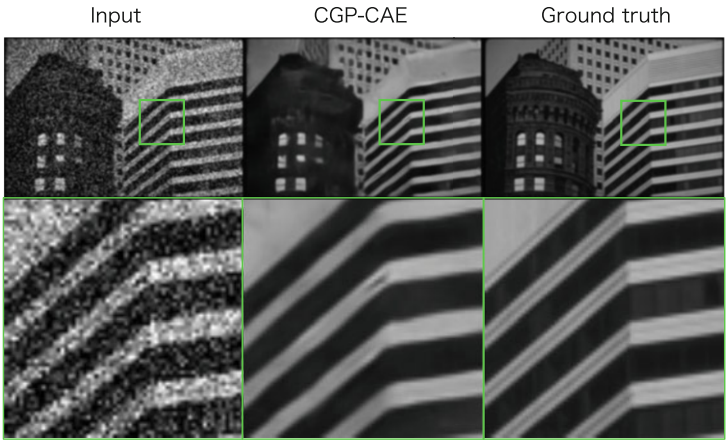

Fig. 7.7 Examples of images reconstructed by CGP-CAE for the denoising task. The first column shows the input image with noise level $\sigma = 50$

and output layer), showing that our evolutionary method was able to find simpler architectures that can provide more accurate results.

An example of an image recovered by CGP-CAE is shown in Fig. 7.7. As we can see, CGP-CAE correctly removes the noise and produces an image as sharp as the ground truth.

7.4.3.4 Analysis of Optimized Architectures

Table 7.7 shows the top five best-performing architectures designed by CGP-CAE for the image inpainting task using center masks on the CelebA dataset and the denoising task, along with their performances measured on their test datasets. One of the best-performing architectures for each task is shown in Fig. 7.8. We can see that although their overall structures do not appear unique, mostly because of the limited search space of CAEs, the number and size of filters are quite different across layers, which is difficult to manually determine. Although it is difficult to provide a general interpretation of why the parameters of each layer are selected, we can make the following observations: (1) regardless of the task, almost all networks have a skip connection in the first layer, implying that the input images contain essential information to yield accurate outputs; (2) 1×1 convolution seems to be an important ingredient for both tasks; 1×1 convolution layers dominate the denoising networks, and all the inpainting networks employ two 1×1 convolution layers; (3) when comparing the inpainting networks to the denoising networks, the following differences are apparent: the largest filters of size 5×5 tend to be employed by the former more often than the latter (2.8 vs. 0.8 layers on average), and 1×1 filters tend to be employed by the former less often than the latter (2.0 vs. 3.2 layers on average).

Table 7.7 Best-performing five architectures of CGP-CAE

Architecture (Inpainting)	PSNR	SSIM
$CS(128, 3) - C(64, 3) - CS(128, 5) - C(128, 1) - CS(256, 5) - C(256, 1) - CS(64, 5)$	29.91	0.9344
$C(256, 3) - CS(64, 1) - C(128, 3) - CS(256, 5) - CS(64, 1) - C(64, 3) - CS(128, 5)$	29.91	0.9343
$CS(128, 5) - CS(256, 3) - C(64, 1) - CS(128, 3) - CS(64, 5) - CS(64, 1) - C(128, 5) - C(256, 5)$	29.89	0.9334
$CS(128, 3) - CS(64, 3) - C(64, 5) - CS(256, 3) - C(128, 3) - CS(128, 5) - CS(64, 1) - CS(64, 1)$	29.88	0.9346
$CS(64, 1) - C(128, 5) - CS(64, 3) - C(64, 1) - CS(256, 5) - C(128, 5)$	29.63	0.9308
Architecture (Denoising)	**PSNR**	**SSIM**
$CS(64, 3) - C(64, 1) - C(128, 3) - CS(64, 1) - CS(128, 5) - C(128, 3) - C(64, 1)$	26.67	0.7313
$CS(64, 5) - CS(256, 1) - C(256, 1) - C(64, 3) - CS(128, 1) - C(64, 3) - CS(128, 1) - C(128, 3)$	26.28	0.7113
$CS(64, 3) - C(64, 1) - C(128, 3) - CS(64, 1) - CS(128, 5) - C(128, 3) - C(64, 1)$	26.28	0.7107
$CS(128, 3) - CS(64, 1) - C(64, 3) - C(64, 3) - CS(64, 1) - C(64, 3)$	26.20	0.7047
$CS(64, 5) - CS(128, 1) - CS(256, 3) - CS(128, 1) - CS(128, 1) - C(64, 1) - CS(64, 3)$	26.18	0.7037

$C(F, k)$ indicates that the layer has F filters of size $k \times k$ without a skip connection. CS indicates that the layer has a skip connection. This table shows only the encoder part of CAEs. For denoising, the average PSNR and SSIM values of three noise levels are shown

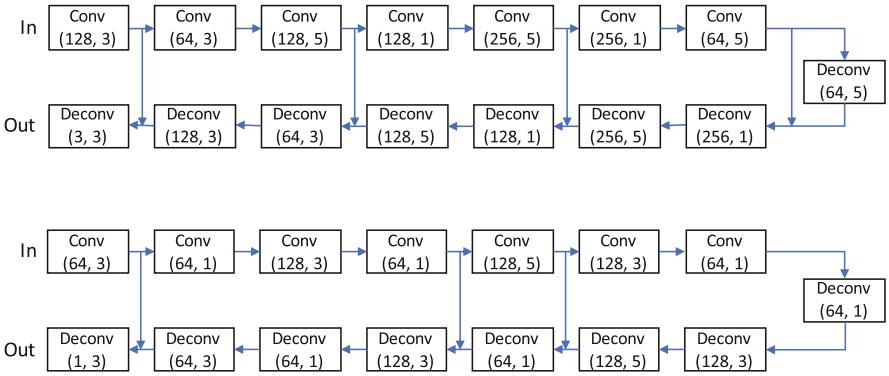

Fig. 7.8 One of the best-performing architectures given in Table 7.7 for inpainting (upper) and denoising (lower) tasks

7.5 Summary

This chapter introduced a neural architecture search for CNNs: a CGP-based approach for designing deep CNN architectures. Specifically, the methods, CGP-CNN for image classification and CGP-CAE for image restoration, were explained. The methods generate CNN architectures based on the CGP encoding scheme with highly functional modules and use the evolutionary algorithm to find good architectures. The effectiveness and potential of CGP-CNN and CGP-CAE were verified through numerical experiments. The experimental results of image classification showed that CGP-CNN can find a well-performing CNN architecture. In the experiment on image restoration tasks, we showed that CGP-CAE can find a simple yet high-performing architecture of a CAE. We believe that evolutionary computation is a promising solution for NAS.

The bottleneck of the architecture search of DNN is the computational cost. Simple yet effective acceleration techniques, termed rich initialization and early termination of network training, can be found in [36]. Another possible acceleration technique is starting with a small data size and increasing the training data for the neural networks as the generation progresses. Moreover, to simplify and compact the CNN architectures, we may introduce regularization techniques to the architecture search process. Alternatively, we may be able to manually simplify the obtained CNN architectures by removing redundant or less effective layers.

Considerable room remains for exploration of search spaces of architectures of classical convolutional networks, which may apply to other tasks such as single image colorization [48], depth estimation [3, 44], and optical flow estimation [9].

References

1. Akimoto, Y., Shirakawa, S., Yoshinari, N., Uchida, K., Saito, S., Nishida, K.: Adaptive stochastic natural gradient method for one-shot neural architecture search. In: Proceedings of the 36th International Conference on Machine Learning (ICML), vol. 97, pp. 171–180 (2019)
2. Baker, B., Gupta, O., Naik, N., Raskar, R.: Designing neural network architectures using reinforcement learning. In: Proceedings of the 5th International Conference on Learning Representations (ICLR) (2017)
3. Eigen, D., Puhrsch, C., Fergus, R.: Depth map prediction from a single image using a multi-scale deep network. In: Advances in Neural Information Processing Systems 27 (NIPS '14), pp. 2366–2374 (2014)
4. Elsken, T., Metzen, J.H., Hutter, F.: Efficient multi-objective neural architecture search via Lamarckian evolution. In: Proceedings of the 7th International Conference on Learning Representations (ICLR) (2019)
5. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: a survey. Journal of Machine Learning Research **20**(55), 1–21 (2019)
6. Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout networks. In: Proceedings of the 30th International Conference on Machine Learning (ICML), pp. 1319–1327 (2013)

7. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on ImageNet classification. In: Proceedings of the International Conference on Computer Vision (ICCV), pp. 1026–1034 (2015)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016)
9. Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., Brox, T.: Flownet 2.0: Evolution of optical flow estimation with deep networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
10. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on Machine Learning (ICML), pp. 448–456 (2015)
11. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: Proceedings of the 3rd International Conference on Learning Representations (ICLR) (2015)
12. Krause, J., Stark, M., Deng, J., Fei-Fei, L.: 3D object representations for fine-grained categorization. In: Proceedings of the International Conference on Computer Vision Workshops (ICCVW), pp. 554–561 (2013)
13. Kupyn, O., Budzan, V., Mykhailych, M., Mishkin, D., Matas, J.: DeblurGAN: blind motion deblurring using conditional adversarial networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 8183–8192 (2018)
14. Larsson, G., Maire, M., Shakhnarovich, G.: FractalNet: ultra-deep neural networks without residuals. In: Proceedings of the 5th International Conference on Learning Representations (ICLR) (2017)
15. Lin, M., Chen, Q., Yan, S.: Network in network. In: Proceedings of the 2nd International Conference on Learning Representations (ICLR) (2014)
16. Liu, Z., Luo, P., Wang, X., Tang, X.: Deep learning face attributes in the wild. In: Proceedings of the International Conference on Computer Vision (ICCV), pp. 3730–3738 (2015)
17. Liu, H., Simonyan, K., Yang, Y.: Darts: differentiable architecture search. In: Proceedings of the International Conference on Learning Representations (ICLR) (2019)
18. Mao, X., Shen, C., Yang, Y.: Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. In: Advances in Neural Information Processing Systems (NIPS), pp. 2802–2810 (2016)
19. Martin, D., Fowlkes, C., Tal, D., Malik, J.: A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In: Proceedings of the International Conference on Computer Vision (ICCV), pp. 416–423 (2001)
20. Miikkulainen, R., Liang, J.Z., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., Hodjat, B.: Evolving deep neural networks. Preprint. arXiv:1703.00548 (2017)
21. Miller, J.F., Smith, S.L.: Redundancy and computational efficiency in Cartesian genetic programming. IEEE Trans. Evol. Comput. **10**(2), 167–174 (2006)
22. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Proceedings of the European Conference on Genetic Programming (EuroGP), pp. 121–132 (2000)
23. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML), pp. 807–814 (2010)
24. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning. In: Advances in Neural Information Processing Systems (NIPS) Workshop on Deep Learning and Unsupervised Feature Learning (2011)
25. Paszke, A., Chanan, G., Lin, Z., Gross, S., Yang, E., Antiga, L., Devito, Z.: Automatic differentiation in PyTorch. In: Autodiff Workshop in Thirty-first Conference on Neural Information Processing Systems (NIPS) (2017)
26. Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., Efros, A.A.: Context encoders: feature learning by inpainting. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2536–2544 (2016)

27. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. In: Proceedings of the 35th International Conference on Machine Learning (ICML), vol. 80, pp. 4095–4104 (2018)
28. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Le, Q.V., Kurakin, A.: Large-scale evolution of image classifiers. In: Proceedings of the 34th International Conference on Machine Learning (ICML), pp. 2902–2911 (2017)
29. Saito, S., Shirakawa, S.: Controlling model complexity in probabilistic model-based dynamic optimization of neural network structures. In: Proceedings of the 28th International Conference on Artificial Neural Networks (ICANN), Part II (2019)
30. Schaffer, J.D., Whitley, D., Eshelman, L.J.: Combinations of genetic algorithms and neural networks: a survey of the state of the art. In: Proceedings of International Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN '92), pp. 1–37 (1992)
31. Shirakawa, S., Iwata, Y., Akimoto, Y.: Dynamic optimization of neural network structures using probabilistic modeling. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-18), pp. 4074–4082 (2018)
32. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: Proceedings of the 3rd International Conference on Learning Representations (ICLR) (2015)
33. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002)
34. Suganuma, M., Shirakawa, S., Nagao, T.: A genetic programming approach to designing convolutional neural network architectures. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 497–504 (2017)
35. Suganuma, M., Ozay, M., Okatani, T.: Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search. In: Proceedings of the 35th International Conference on Machine Learning (ICML), vol. 80, pp. 4771–4780 (2018)
36. Suganuma, M., Kobayashi, M., Shirakawa, S., Nagao, T.: Evolution of deep convolutional neural networks using Cartesian genetic programming. Evol. Comput. (2019). https://doi.org/10.1162/evco_a_00253. Early access
37. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–9 (2015)
38. Tai, Y., Yang, J., Liu, X., Xu, C.: MemNet: A persistent memory network for image restoration. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4539–4547 (2017)
39. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: MnasNet: platform-aware neural architecture search for mobile. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
40. Tokui, S., Oono, K., Hido, S., Clayton, J.: 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) (2015)
41. Wang, Z., Bovik, A., Sheikh, H., Simoncelli, E.: Image quality assessment: from error visibility to structural similarity. IEEE Trans. Image Process. **13**(4), 600–612 (2004)
42. Xie, L., Yuille, A.: Genetic CNN. In: Proceedings of the International Conference on Computer Vision (ICCV), pp. 1388–1397 (2017)
43. Xie, S., Zheng, H., Liu, C., Lin, L.: SNAS: stochastic neural architecture search. In: Proceedings of the International Conference on Learning Representations (ICLR) (2019)
44. Xu, D., Ricci, E., Ouyang, W., Wang, X., Sebe, N.: Multi-scale continuous CRFs as sequential deep networks for monocular depth estimation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5354–5362 (2017)
45. Yao, X.: Evolving artificial neural networks. Proc. IEEE **87**(9), 1423–1447 (1999)

46. Yeh, R.A., Chen, C., Lim, T.Y., Schwing, A.G., Hasegawa-Johnson, M., Do, M.N.: Semantic image inpainting with deep generative models. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6882–6890 (2017)
47. Zagoruyko, S., Komodakis, N.: Wide residual networks. In: Proceedings of the British Machine Vision Conference (BMVC), pp. 87.1–87.12 (2016)
48. Zhang, R., Isola, P., Efros, A.A.: Colorful image colorization. In: European Conference on Computer Vision (ECCV) 2016. Lecture Notes in Computer Science, vol. 9907, pp. 649–666. Springer, Berlin (2016)
49. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: Proceedings of the 5th International Conference on Learning Representations (ICLR) (2017)