Chapter 12
Evolving Deep Neural Networks for X-ray Based Detection of Dangerous Objects

Ryotaro Tsukada, Lekang Zou, and Hitoshi Iba

Abstract In recent years, neural networks with an additional convolutional layer, referred to as convolutional neural networks (CNN), have widely been recognized as being effective in the field of image recognition. In the majority of these previous researches, the structures of networks were designed by hand, and were based on experience. However, there is no established theory explaining how to build networks with higher learning abilities. In this chapter, we propose a framework on automatically obtaining network structures with the highest learning ability for image recognition, through the combination of the various core technologies. We employ EC (evolutionary computation) for the automatic extraction and synthesis of network structures. Additionally, we attempt to perform an effective search in a larger parameter space by gradually increasing the number of training epochs during the generation change process. In order to show the effectiveness of our approach, we apply the proposed method to the task of detecting dangerous objects in an X-ray image data set. Compared with the previous results, we have achieved an improvement in the mAP value. We can also find several by-passes in the structures that were actually obtained.

12.1 Introduction

In recent years, machine learning methods using neural networks have significantly outperformed traditional methods in areas such as image recognition [5], speech recognition [2], and natural language processing [1].

Neural networks with an additional convolutional layer, referred to as convolutional neural networks (CNN), are widely recognized as being effective in the field of image recognition. In ILSVRC 2012, a worldwide image recognition contest conducted in 2012, Hinton et al. [5] used a method of image recognition

R. Tsukada · L. Zou · H. Iba (✉)
Graduate School of Information Science and Technology, University of Tokyo, Tokyo, Japan
e-mail: tsukada@iba.t.u-tokyo.ac.jp; zou@iba.t.u-tokyo.ac.jp; iba@iba.t.u-tokyo.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_12

based on convolutional neural networks and achieved more than 10% significant improvement (compared with previous methods) in terms of recognition accuracy. Subsequently, this achievement has triggered a lot of proactive research on image recognition based on convolutional neural networks; this has resulted in the numerous proposals and performance enhancement of neural networks with more complex and varied structures such as the GoogLeNet [18], ResNet [4], and YOLO [12].

Image recognition is the process of extracting features from images obtained in the real world in order to recognize objects such as characters, symbols, people's faces, and animals that may appear in an image, and it has a wide range of applications. Hence, building systems with increasingly higher recognition accuracies is necessary.

A characteristic of convolutional neural networks is that networks can be constructed by combining layers that perform specific functions, just like blocks that are put together. In the majority of the previous research on image recognition using convolutional neural networks, the structures of networks were designed by hand, and were based on experience. In this context, the improvement in the learning ability of various networks from various core technologies such as dropout, batch normalization, GoogLeNet's inception module [18], and residual learning introduced in ResNet [4] has been empirically verified. However, there is no established theory explaining how to combine these technologies to build networks with higher learning abilities. In fact, networks that currently exhibit the highest learning levels contain a huge number of parameters and are deep and complex. Therefore, specialists must perform a lot of trial-and-error and craftwork in order to yield the highest learning ability on a specific data set. Consequently, there is ongoing research on the automatic design of network structures using genetic programming (GP) [9, 17], and network structure search methods using neural networks [11].

The present research focuses on automatically obtaining network structures with the highest learning ability for image recognition, through the combination of the various core technologies itemized above. We used genetic algorithms (GA) for the automatic extraction and synthesis of network structures. The advantage of genetic algorithms in the present research is that a simple network can gradually evolve into a complex network during the search process with very little prior input from a human. Additionally, we attempt to perform a search in a larger parameter space by gradually increasing the number of training epochs that evaluate each individual during the generation change process.

12.2 Related Research

12.2.1 Neuro-Evolution

Neuro-evolution in the broad sense is an attempt to generate neural networks by the use of evolutionary computation methods [9]. In the present research, our goal is to optimize the structure of the convolutional neural network by using GA. This process can be likened to an evolutionary computation; thus, this approach can be considered as a form of neuro-evolution.

The NEAT (NeuroEvolution of Augmented Topologies [16]) method is an example of neuro-evolution. The method is characterized by the growth of small structures into larger structures as they get optimized. Using GA, the network undergoes evolution by crossover and mutation, which, respectively, results in better structures and changes in the connectional relationship between nodes. An example of structural change resulting from a mutation in NEAT is illustrated in Fig. 12.1.

12.2.2 Genetic CNN

Genetic convolutional neural network (Genetic CNN) [19] is a proposed example of a convolutional neural network structure search using GA. In Genetic CNN, a stage composed of multiple convolutional layers and subsequent pooling layers is repeated multiple times. The convolutional layers in each stage are connected in the form of directed acyclic graphs. As shown in Fig. 12.2, the binary values 0 or 1 are

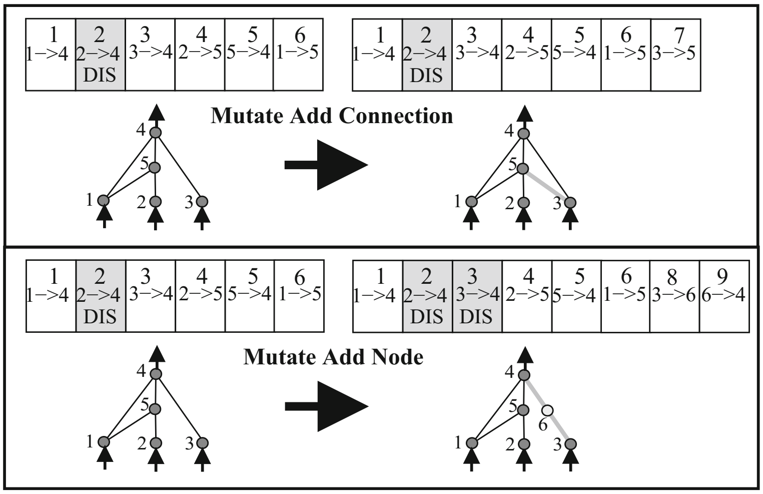


Fig. 12.1 Example of structure change due to mutation in NEAT [16]

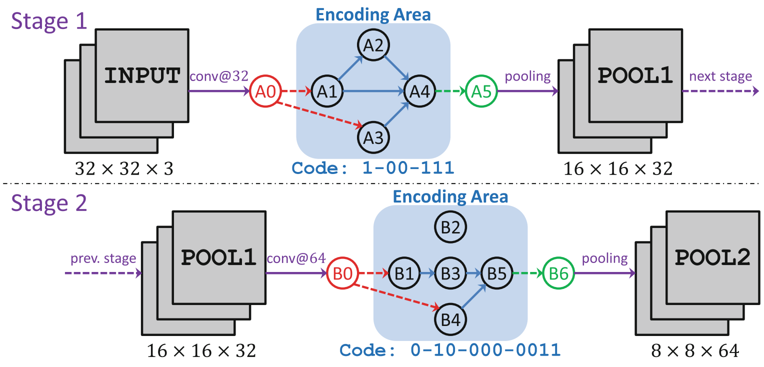


Fig. 12.2 Example of network structure encoding in Genetic CNN [19]

encoded as a sequence, specifying which previous convolutional layer is connected to itself. These codes are viewed as "genes," and the process of searching for the network structures with higher learning abilities using GA is referred to as Genetic CNN. The number of stages and the number of convolutional layers in each stage are preset; therefore, the length of the sequence that makes up the gene is also fixed.

To represent the fitness of GA, we use the recognition accuracy yielded by the network pertaining to the individual based on a specific data set used for learning. Crossover is achieved by exchanging bits between sequences, and mutations are performed by bit reversal. For the selection, a roulette wheel selection process is adopted where the probability of selection is proportional to the difference in fitness with respect to the individual with the lowest fitness. Therefore, individuals having networks with a structure that yields higher accuracy are more likely to survive into the next generation. This way, trial-and-error involving the combinations of network structures and mutations eventually results in the output of individuals having networks with the highest accuracy. The Genetic CNN algorithm is described in Algorithm 1:

12.2.3 Aggressive Selection and Mutation

In Genetic CNN, training is performed from scratch to evaluate the individuals. This process is repeated multiple times for all individuals of the generation during the generation change process. Hence, if the final number of generations is T and the number of individuals in each generation is N, training is repeated $T \times N$ times until the end of the generation change process. However, the training of a convolutional neural network normally takes a considerable amount of time for just one round. Consequently, a drawback of these repetitions is the considerable amount of time required.

Algorithm 1 Genetic CNN

```
1: Input: data set $\mathcal{D}$, final generation number $T$, number of individuals in each generation $N$,
   probabilities of crossover and mutation $p_C$ and $p_M$, parameters $q_C, q_M$ related to crossover
   and mutation
2: Initialization: the initial generation is formed, containing $N$ individuals. Each individual
   consists of a structure where bits 0 and 1 are selected at random. The individual fitness is
   evaluated (more on this later).
3: for $t = 1, 2, \ldots, T$ do
4:     Selection: considering individuals from generation $t - 1$, $N$ individuals are selected in a
   roulette wheel scheme where the selection probability is proportional to the fitness difference
   with regard to the individual with the lowest (worst) fitness; selection of the same individual
   multiple times is allowed.
5:     Crossover: for each pair of selected neighboring individuals in the same generation $t$, the
   crossover is performed with probability $p_C$ (each bit is exchanged with probability $q_C$).
6:     Mutation: for individuals that did not undergo crossover as above, mutation is carried out
   with probability $p_M$ (each bit is reversed with a probability $q_M$).
7:     Evaluation: the network of an individual is trained on data set $\mathcal{D}$ and tested using test
   images to obtain fitness, which is the accuracy obtained.
8: end for
9: Output: individuals and their recognition accuracies in the final generation.
```

Additionally, due to the utilization of a roulette wheel selection scheme where the selection probability is proportional to the difference with respect to the fitness of the individual with the lowest fitness, it is possible that individuals having networks with weak structures that are not expected to yield further improvements in accuracy could avoid elimination from selection and survive.

Furthermore, elements not related to connections in the convolutional layer (hyperparameters such as the layout of the pooling layer, number of channels in each layer, filter size, and stride.) must be previously determined, resulting in a small search space for the parameters subject to search.

In [7], the above problems are addressed in the following ways:

(i) evaluation of individuals is accelerated by roughly assessing the fitness of an individual by training using a small number of epochs;
(ii) by introducing a selection and mutation scheme called "aggressive selection and mutation," weak individuals are eliminated early so that new individuals based on strong ones can be born more easily;
(iii) the space of parameters to be searched is enlarged by increasing the elements subject to mutation.

The training of convolutional neural networks is performed by repeatedly feeding the same training data to the networks. The number of such repetitions is called "number of epochs." Usually, a sufficient level of recognition accuracy is not achieved if the network is trained with a small number of epochs. However, to estimate fitness as a reference for generational change in GA, a rough estimation based on a small number of epochs is expected to be enough.

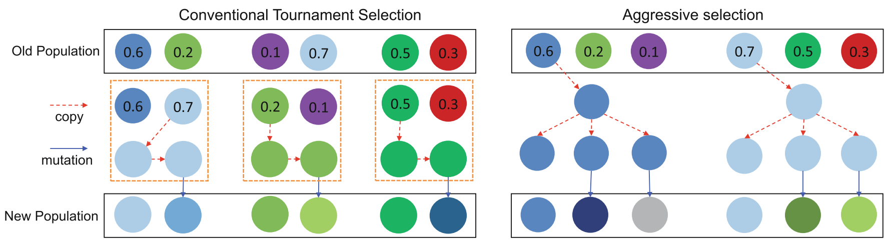


Fig. 12.3 Different strategies. Aggressive selection and mutation for $N = 6$ and $k = 2$ are shown in the right [7]

The aggressive selection and mutation method involve selecting only $k(\ll N)$ individuals with high fitness from the parents' generation, complementing the lacking part with clones, and applying mutation to those clones. The method is similar to the random mutation hill-climbing method because it searches for a solution based only on mutations. An example of aggressive selection and mutation for $N = 6$ and $k = 2$ is illustrated in Fig. 12.3.

In Genetic CNN, only connectional relations in the convolutional layer are set as a target for optimization. However, in aggressive selection and mutation, various types of mutation operations are available, such as adding or deleting layers other than convolutional layers or changing preset values of the convolutional layer itself (hyperparameters). With this method, the parameter search space gets significantly expanded.

Consequently, the time required to find the best individual is significantly reduced and the recognition accuracy of the best individual is improved dramatically.

The aggressive selection and mutation algorithm is described in Algorithm 2.

Algorithm 2 Aggressive selection and mutation

```
1: Input: data set $\mathcal{D}$, final generation number $T$, number of individuals per generation $N$, number
   of elite individuals $k$ to be added to the next generation, threshold $d$ for the distance between
   individuals
2: Initialization: the 0-th generation is formed by $N$ individuals having a fixed initial structure.
   The fitness of each individual is evaluated (to be explained later).
3: for $t = 1, 2, \ldots, T$ do
4:     Selection: $k$ individuals with high fitness are selected sequentially from generation $t - 1$.
   The individual is added to generation $t$ unless the distance with respect to the individuals
   already added to generation $t$ is less than $d$. A total of $N - k$ clones of the $k$ added individuals
   are added.
5:     Mutation: a mutation operation is selected and applied to each of the $N - k$ cloned
   individuals. Nothing is done to the remaining $k$ individuals.
6:     Evaluation: the network of each individual is trained on data set $\mathcal{D}$, and the recognition
   accuracy on test images is stored as the fitness value.
7: end for
8: Output: individuals in the final generation and the recognition accuracy for these individuals.
```

Details on the initial structure and methods of mutation are explained in Sect. 12.4.

12.2.4 YOLO

YOLO is one of single-shot object detection CNN models. YOLO first resizes the input image into a square and divides it into equal-sized regions using $S \times S$ grids. Each grid will predict B bounding boxes and probability values for C categories. An example for $S = 7$, $B = 2$, and $C = 20$ is illustrated in Fig. 12.4 [13]. Each bounding box needs to be represented by five parameters: the coordinate of the center point on the x and y axes, the height h and width w of the bounding box, and the confidence value c. Finally, YOLO selects the prediction box with the highest confidence value as the detection result. In order to avoid multiple detections of the same object, YOLO uses the non-maximum suppression method to ensure that each object is detected only once.

As shown in Fig. 12.4, the entire YOLO network is composed of convolutional layers and fully connected layers without any sub-network structure. Here, the output dimension of YOLO is $7 \times 7 \times 30$. This is because when $S = 7$, $B = 2$, and $C = 20$, each grid predicts two bounding boxes, probability values for 20 categories, and each bounding box needs five parameters $\{x, y, h, w, c\}$ in total requiring $7 \times 7 \times (5 \times 2 + 20) = 7 \times 7 \times 30$ parameters.

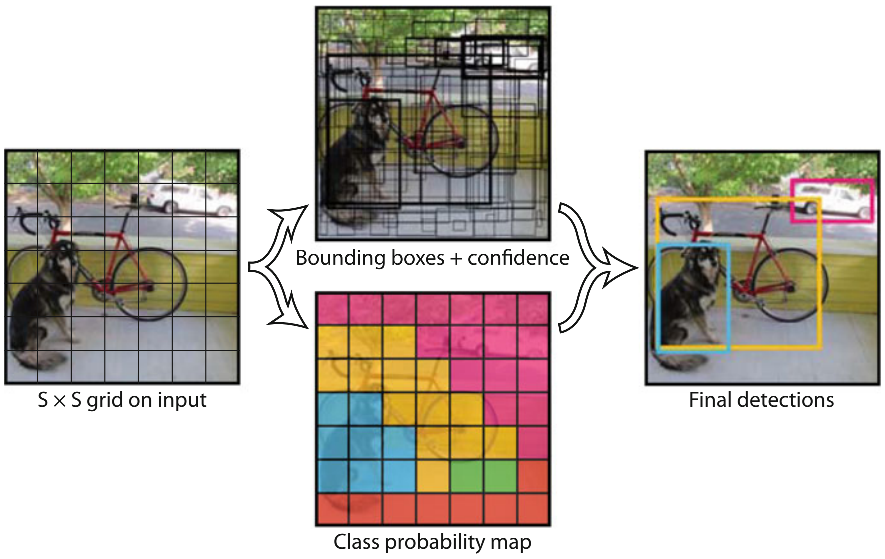


Fig. 12.4 The system model of YOLO [13]

12.2.5 Transfer Learning

Transfer learning [10] is a framework in machine learning where data on related problems and derived knowledge are used to effectively and efficiently solve the target problem. The sender and the receiver of knowledge to be transferred are called the source domain and target domain, respectively.

Humans learn various things from transfer learning. For instance, when someone who can play the piano starts to learn the electronic organ at the same time as someone who cannot play the piano, the former can learn to play the electronic organ better and in a shorter amount time than the latter. In the framework of transfer learning, the piano is the source domain, the electronic organ is the target domain, and proficiency in playing the piano assists in the learning of the electronic organ. Transfer learning is applied in various fields, including natural language processing, voice recognition, and image processing.

Figure 12.5 shows a rough flow of transfer learning. In this example, the source task concerns training on female speech and the target task is to recognize speech from males. Learning is carried out using data and knowledge associated with problems in the source and target domains to ultimately answer problems in the target domain efficiently and with high precision. Usually it is assumed that the source and the target domains have some structural relationship.

Transfer learning is very effective when there is little training data in the target domain, but substantial data in the source domain. Moreover, transferring knowledge from a domain that is highly similar to the target domain results in more efficient learning. In contrast, the transfer of knowledge from a source domain with low similarity results in a decrease in learning performance, which is called negative transfer.

In transfer learning, the maximum limit of learning performance in the training domain is normally limited by the learning performance in the source domain. In other words, a higher learning precision in the source domain results in a better chance of improving learning efficiency in the target domain.

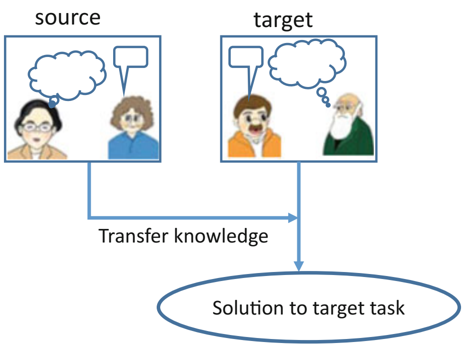


Fig. 12.5 Image of transfer learning

An important problem in transfer learning is determining what knowledge to transfer. If the data from the source domain is necessary, the most obvious approach is to appropriately map the data and use it when learning in the target domain. On the other hand, transferring feature values or parameters that exist both in the source and target domains is also possible. What knowledge can be transferred and what knowledge successfully works depends on each domain, and determining which to apply is difficult.

12.3 Proposed Method

In Sect. 12.2, we have explained the aggressive selection and mutation scheme [7]. In this section, we further introduce an extended method, which we call ASM+, where generation alternation is performed by increasing the number of epochs according to the generation number during the evaluation training of the individuals. Let $n_{\min}$, $n_{\max}$, and T be the minimum number of epochs, the maximum number of epochs, and the number of total generations, respectively. In our method ASM+, the number of epochs $n(t)$ related to generation t is defined by the following equation:

$$n(t) = \frac{(T - t) \cdot n_{\min} + t \cdot n_{\max}}{T}$$

Thus, the interval between $n(0) = n_{\min}$ and $n(T) = n_{\max}$ is uniform and depends on the final generation number T.

Due to the slope with respect to the epoch number, the generation change cycle is faster at the beginning of evolution, thereby enabling an evaluation of individuals with various structures over a wide range. Subsequently, individuals exhibiting good structures at the end of evolution are evaluated locally with higher accuracy.

12.4 Experiments on Evolutionary Synthesis of Convolutional Neural Networks

Experiments were carried out using two types of combinations of data sets and tasks. First, the effectiveness of the proposed method ASM+ is evaluated through a relatively simple handwritten number classification task using MNIST [6] as the data set. Second, we also apply ASM+ to the task of detecting dangerous objects in an X-ray image data set simulating luggage inspection.

12.4.1 MNIST Handwritten Number Classification Experiment

For the first experiment, the MNIST [6] data set is used. MNIST is a widely used data set designed for the classification of handwritten numbers. MNIST was selected because of a possible comparison with the previous researches [19] and [7] mentioned in Sect. 12.2. Moreover, the task is simple, and it permits the evaluation of the effectiveness of the method with a small number of computations.

The MNIST data set consists of 60,000 images for training and 10,000 images for testing. Each image is formed by a 28×28-sized gray scale corresponding to an Arabic number from 0 to 9 which are uniformly drawn on the image.

12.4.1.1 Initial Generation

As shown in Fig. 12.6, an individual in the initial generation has a network formed by 3 layers: an input layer, a global max pooling layer, and a fully connected layer. Global max pooling is the process of selecting the maximum value from all channels of the input feature map, with an output to a 1×1-sized feature map with the same number of channels. For the structure of the initial generation, the accuracy on the MNIST data set is about 11%, which is approximately the same as a network that just outputs random results. This network was selected as the initial structure to confirm that it is unnecessary to introduce restrictions in the search range by including human intervention.

12.4.1.2 Mutation Operations

In [7], the 15 types of operations described below were utilized for mutation operations. In the following, "random" refers to the selection of candidates with a uniform probability.

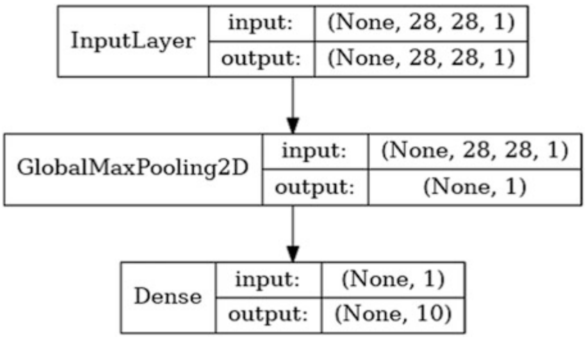


Fig. 12.6 Network structure of an individual in the initial generation. For details on the notation of the figure, refer to Fig. 12.8

- `add_convolution`: inserts a convolutional layer with 32 channels, one stride, 3 × 3 filter size, and one-pixel padding in a random position. The number of dimensions of the feature maps for the input and output before and after this operation does not change. The activation function used is ReLU.
- `remove_convolution`: selects a single convolutional layer at random and deletes it.
- `alter_channel_number`: selects a single convolutional layer at random, and selects a channel number out of {8, 16, 32, 48, 64, 96, 128} at random for the replacement.
- `alter_filter_size`: selects a convolutional layer at random and selects a filter size out of {1 × 1, 3 × 3 or 5 × 5} for the replacement.
- `alter_stride`: selects a convolutional layer at random and randomly selects a stride out of {1 or 2} for the replacement.
- `add_dropout`: selects a convolutional layer at random and inserts a dropout soon after. The dropout ratio is fixed at 0.5.
- `remove_dropout`: selects a dropout at random for removal.
- `add_pooling`: selects a convolutional layer at random and inserts a pooling layer soon after. Max pooling is adapted and the kernel size is fixed as 2 × 2.
- `remove_pooling`: selects a pooling layer at random and deletes it.
- `add_skip`: inserts a residual network, which was introduced in ResNet [4]. It precisely selects a random pair of layers where the feature maps of the outputs have the same dimension and inserts a layer that has an output formed by the sum of these outputs.
- `remove_skip`: selects at random one of the skip layers above and deletes it.
- `add_concatenate`: like `add_skip`, it selects a random pair of layers where the feature maps of the outputs have the same dimension (however, the numbers of channels may not be the same), and inserts a layer that has an output formed by concatenating the previous outputs.
- `remove_concatenate`: selects one of the concatenate layers above at random and deletes it.
- `add_fully_connected`: selects a random position just after other fully connected layers or the last layer and inserts a fully connected layer. The dimension of the output is selected at random from {50, 100, 150, 200}.
- `remove_fully_connected`: a single fully connected layer is selected at random and deleted.

Selecting and applying one out of the 15 types of operations above is considered a mutation. However, to facilitate the evolution to more complex structures, in our method ASM+, the probability of selecting `add_convolution`, `add_skip`, `add_concatenate`, `alter_stride`, `alter_filter_size`, and `alter_channel_number` is two times higher than other operations.

In some cases, it may be impossible to apply a certain operation. For instance, it is impossible to apply `remove_convolution` to an individual that does not have a convolutional layer. In such cases, the operation is selected again. In other cases, the network structure of a given individual may be considered invalid due to the dimensional mismatch after beginning the actual evaluation training. In such cases, the fitness of such individual is set to 0.

12.4.1.3 Experimental Results

Experiments were conducted on a single GPU on Google Colaboratory. Two patterns were defined as follows: with and without batch normalization process inserted just after the convolutional layer.

Other settings were: $T = 30$ (final generation number), $N = 10$ (number of individuals in each generation), $k = 1$ (number of elite individuals to be added to the next generation), $n_{\min} = 3$ (minimum number of epochs), and $n_{\max} = 12$ (maximum number of epochs).

Figure 12.7 shows how the fitness for the best individual (or image recognition accuracy) evolved along the generations for each pattern. The recognition accuracy obtained for the best individual in the final generation re-evaluated with the maximum number of epochs $n_{\max}$ and the time required for the entire evolution process are shown in Table 12.1.

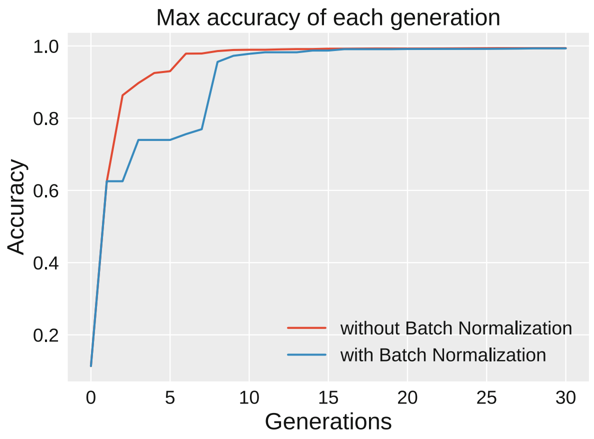


Fig. 12.7 Changes in fitness value for the best individual in each generation

Table 12.1 Comparison of the recognition accuracy for the best individual in the final generation across different methods

Method	Recognition accuracy	Computation time
Genetic CNN [19]	0.9966	48 GPUH
Aggressive selection and mutation [7]	0.9969	35 GPUH
ASM + (without batch normalization)	0.9932	N.A.
ASM + (with batch normalization)	0.9913	9 GPUH

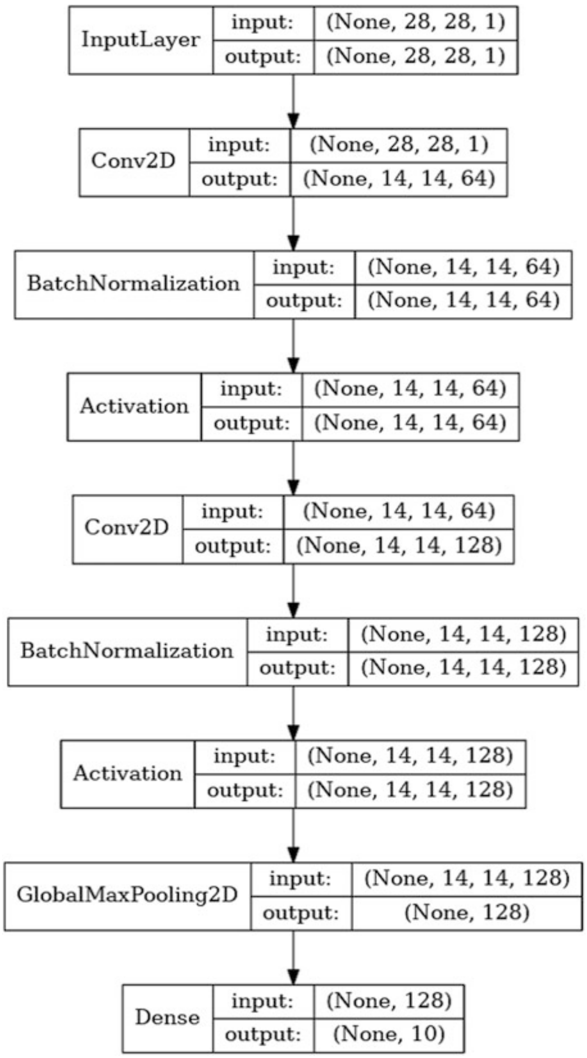

InputLayer
input: (None, 28, 28, 1)
output: (None, 28, 28, 1)
Conv2D
input: (None, 28, 28, 1)
output: (None, 14, 14, 64)
BatchNormalization
input: (None, 14, 14, 64)
output: (None, 14, 14, 64)
Activation
input: (None, 14, 14, 64)
output: (None, 14, 14, 64)
Conv2D
input: (None, 14, 14, 64)
output: (None, 14, 14, 128)
BatchNormalization
input: (None, 14, 14, 128)
output: (None, 14, 14, 128)
Activation
input: (None, 14, 14, 128)
output: (None, 14, 14, 128)
GlobalMaxPooling2D
input: (None, 14, 14, 128)
output: (None, 128)
Dense
input: (None, 128)
output: (None, 10)

Fig. 12.8 Network structure for the best individual in the final generation (with batch normalization). InputLayer represents the input layer; Conv2D is a 2-dimensional convolutional layer. BatchNormalization represents batch normalization, and Activation denotes the application of the activation function (ReLU). GlobalMaxPooling2D represents the maximum pooling operation in 2 dimensions across all channels, and Dense represents a fully connected layer. The values in the input and output fields are the dimensions of the input and output, respectively, and individually represent batch size, height, width, and number of channels. "None" indicates that the batch size is arbitrary. Note that fully connected layers have neither height nor width, thus the height and width notations are not shown before and after Dense layer

Figure 12.8 shows the network structure for the best individual in the final generation obtained by inserting batch normalization. A list of all the mutation operations that were selected in each generation resulting in that individual is given in Table 12.2. The evolution of the best individual in each generation is illustrated in Fig. 12.9 as a tree diagram.

Table 12.2 Mutation operations performed on the best individual of each generation (with batch normalization)

Generation	Mutation operation	Maximum recognition accuracy
0	`initialize`	0.1135
1	`add_convolution`	0.6254
2	-	0.6254
3	`alter_channel_number`	0.7398
4	-	0.7398
5	-	0.7398
6	`alter_filter_size`	0.7558
7	`add_convolution`	0.7695
8	`add_convolution`	0.9559
9	`alter_filter_size`	0.9728
10	`alter_filter_size`	0.9783
11	`alter_channel_number`	0.9825
12	-	0.9825
13	-	0.9825
14	`alter_filter_size`	0.9873
15	-	0.9873
16	`alter_stride`	0.9911
17	-	0.9911
18	-	0.9911
19	-	0.9911
20	`alter_stride`	0.9916
21	-	0.9916
22	-	0.9916
23	-	0.9916
24	`alter_channel_number`	0.9922
25	-	0.9922
26	-	0.9922
27	-	0.9922
28	`alter_channel_number`	0.9933
29	-	0.9933
30	-	0.9933

Blank fields denote that no operation took place on the best individual in that generation

12.4.2 Experiment on Detecting Dangerous Objects in X-ray Images

The next stage involves experiments that are carried out on a data set of X-ray images created by Zou [20]. This data set contains 6121 X-ray images with 3 types

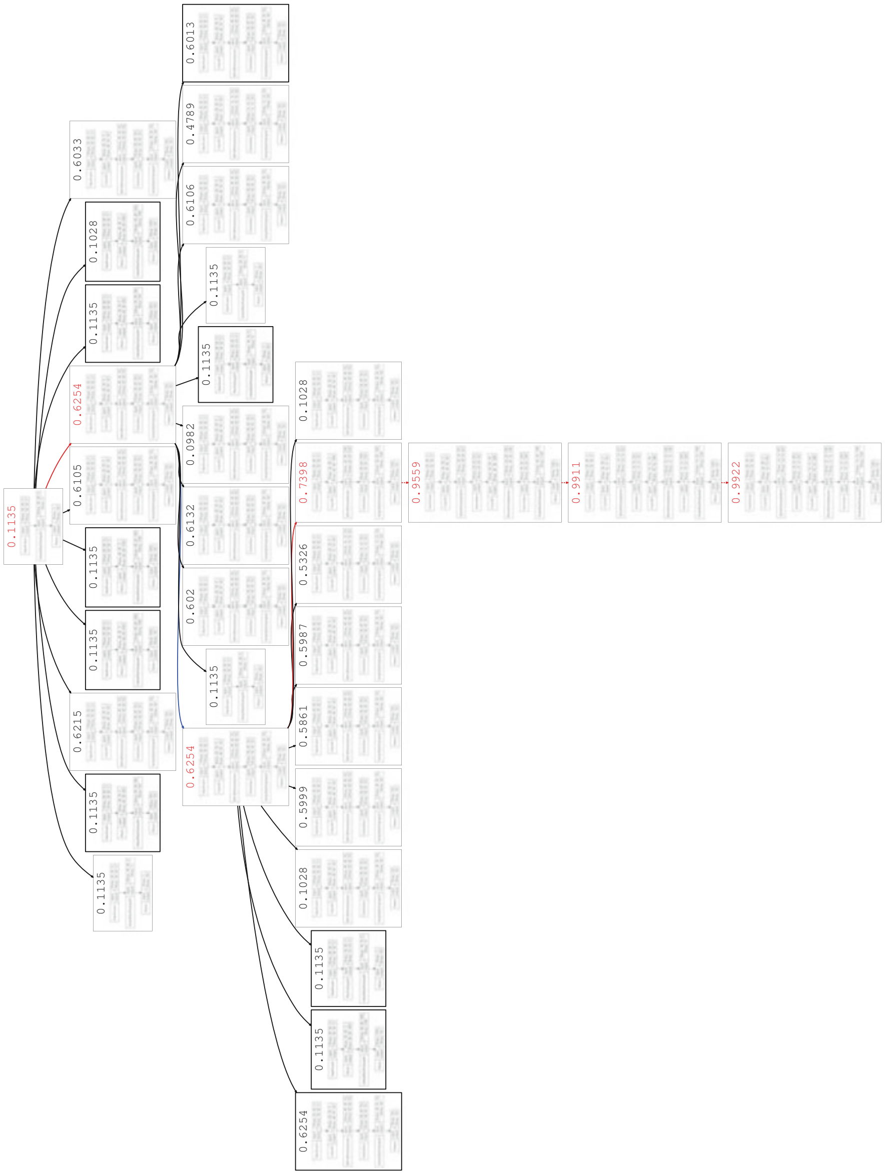


Fig. 12.9 Tree diagram showing the evolution process of the best individual in each generation (partially omitted). The value accompanying the image of the network structure of an individual represents its fitness. Individuals with a value written in red are those selected for the next generation. Colored arrows indicate the selection flow. Red arrows indicate the occurrence of mutation, and blue arrows indicate cloning

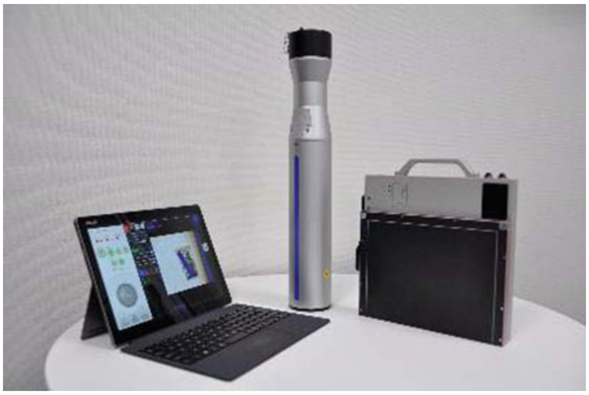


Fig. 12.10 Portable X-ray device (NS-100-L)

Table 12.3 Division of X-ray images into training and test groups

	Training images	Test images	Total
Natural images	662 (10.82%)	442 (7.22%)	1104 (18.04%)
Synthetic images	3010 (49.17%)	2007 (32.79%)	5017 (81.96%)
Total	3672 (59.99%)	2449 (40.01%)	6121 (100.00%)

of dangerous objects (scissors, knives, PET bottles), with annotations related to each image.[1]

Experiments were carried out after dividing the 6121 images into two groups: one to be used for training and the other one for testing. The proportion of images for training and for testing is approximately 6:4 (Table 12.3). This ratio is equal to the previous research [20].[2]

12.4.2.1 Initial Generation

Individuals in the initial generation are defined as those having the network structure described in Table 12.4. This network is the same as the one used in YOLOv2 [12], which specialized in the detection of objects. In the original YOLOv2, another network containing the layers indicated above the double line of the table is

[1] Instead of multi-view X-ray devices which are expensive and heavy, we use a portable X-ray device (see Fig. 12.10). While multi-view devices need to colorize images, our device can collect single-view X-ray images in real time.

[2]More precisely, the actual images to be used for training and testing are randomly changed each time and the system is run 20 times from Run 1 up to Run 20. Here, we used the set of images corresponding to Run 11, whose results are the most representative of the average.

Table 12.4 Network structure of individuals in the initial generation

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	416×416
Maxpool		$2 \times 2/2$	208×208
Convolutional	64	3×3	208×208
Maxpool		$2 \times 2/2$	104×104
Convolutional	128	3×3	104×104
Convolutional	64	1×1	104×104
Convolutional	128	3×3	104×104
Maxpool		$2 \times 2/2$	52×52
Convolutional	256	3×3	52×52
Convolutional	128	1×1	52×52
Convolutional	256	3×3	52×52
Maxpool		$2 \times 2/2$	26×26
Convolutional	512	3×3	26×26
Convolutional	256	1×1	26×26
Convolutional	512	3×3	26×26
Convolutional	256	1×1	26×26
Convolutional (*)	512	3×3	26×26
Maxpool		$2 \times 2/2$	13×13
Convolutional	1024	3×3	13×13
Convolutional	512	1×1	13×13
Convolutional	1024	3×3	13×13
Convolutional	512	1×1	13×13
Convolutional	1024	3×3	13×13
Convolutional	1024	3×3	13×13
Convolutional	1024	3×3	13×13
Concatenate (**)	$1024 + 256$	–	13×13
Convolutional	1024	3×3	13×13
Convolutional	40	1×1	13×13

The structure is the same as the network of YOLOv2 [12]. The outputs of convolutional layers marked with (*) are also connected to concatenate layers marked with (**), being combined along the direction of the channel number. Here, the outputs (*) are reduced to 64 channels by 64 filters that perform 1×1 convolution. Furthermore, to obtain a 13×13 height and width for the output, they are, respectively, reduced by $\frac{1}{2}$, and the channel number is converted to $64 \times 2^2 = 256$. The by-pass described in Sect. 12.2.4 is realized by this concatenate layer

previously trained on the ImageNet [15] data set where it acquires the ability to extract the features of object recognition. Subsequently, the layers below the double line are replaced by those shown in Table 12.4 and the training for object recognition is repeated, thereby characterizing a transfer learning method (see Fig. 12.11). Using this as a reference, only the layers below the double line becomes the target of evolution. The output is a $13 \times 13 \times 40$ tensor because $B = 5$ and $C = 3$, similar

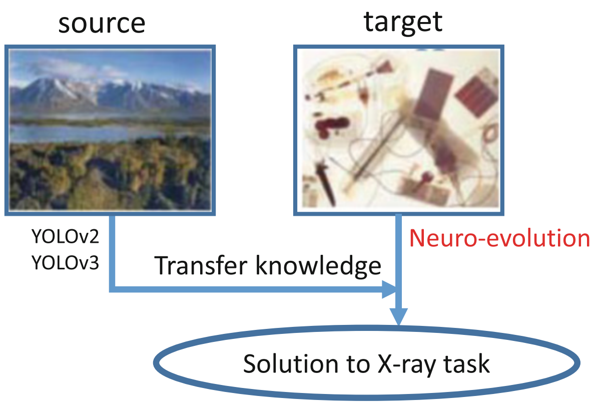


Fig. 12.11 Transfer learning for X-ray based detection of dangerous objects

to the case mentioned in Sect. 12.2.4. Here, $C = 3$ represents the three classes of scissors, knives, and PET bottles.

Since evaluating the fitness (mAP, see Sect. 12.4.2.4) of individuals from this initial generation takes a certain amount of time, unlike the initial generation described in Sect. 12.4.1.1 for the MNIST experiment, this process was skipped. Therefore, the group of individuals for the initial generation was just a formality. The experiment essentially began with the group of individuals of the 1st generation, who were obtained by performing mutation operations on the initial generation.

12.4.2.2 Mutation Operations

As in Sect. 12.4.1.2, the following nine types of operations based on [7] were chosen as mutation operations. As mentioned in Sect. 12.2.4, the output size of the last layer is fixed in order to output results related to object recognition. Furthermore, to avoid overfitting in YOLO, batch normalization is introduced; thus, dropout is unnecessary [12]. Due to these restrictions, six operations, namely `add_dropout`, `remove_dropout`, `add_skip`, `remove_skip`, `add_fully_connected`, `remove_fully_connected`, were removed. In the following, "random" was defined as the selection of each candidate with uniform probability.

- `add_convolution`: inserts a convolutional layer with 1024 channels, one stride, a 3×3 filter size, and one-pixel padding in a random position. The numbers of dimensions of the feature maps for the input and output before and after this operation do not change. The activation function used is Leaky ReLU [8].
- `remove_convolution`: selects a single convolutional layer at random and deletes it.
- `alter_channel_number`: selects a single convolutional layer at random and selects a random channel number out of $\{512, 1024$ or $2048\}$ for the replacement.

- `alter_filter_size`: selects a convolutional layer at random and randomly selects a filter size out of {1×1, 3×3 or 5×5} for the replacement.
- `alter_stride`: selects a convolutional layer at random and selects a stride out of {1 or 2} at random for the replacement.
- `add_pooling`: inserts a pooling layer in a random position. Max pooling is adapted, and the kernel size is fixed as 2×2.
- `remove_pooling`: selects a pooling layer at random and deletes it.
- `add_concatenate`: selects a layer above and a layer below the double line of Table 12.4. It then inserts a layer with an output formed by concatenating those layers. The insertion point is just after the selected lower layer.
- `remove_concatenate`: selects one of the concatenate layers above at random and deletes it.

Mutation is defined here as the operation of selecting and applying one of the nine operations above. As in Sect. 12.4.1.2, the probability of selecting operations `add_convolution`, `add_concatenate`, `alter_stride`, `alter_filter_size`, and `alter_channel_number` is set as twice as large as other operations to help facilitate the evolution of the network structure of an individual to a more complex one.

12.4.2.3 Restructuring of Network Structures Due to Mutation

The results of Sect. 12.4.1.3 indicate that in some cases a mutation operation selected at random cannot be applied due to restrictions related to the number of dimensions of the input and output before and after the layer. For these cases, we introduced a method to enhance the probability that the selected operation can be applied by trying to restructure the network as much as possible to make dimensions match when `add_concatenate` is selected as a mutation operation. The restructuring algorithm is shown in Algorithm 3. This algorithm imitates the operations carried out in the concatenate layer marked with (**) in Table 12.4.

12.4.2.4 Fitness (mAP) Calculation Method

In the MNIST handwritten classification task mentioned in Sect. 12.4.1, the classification accuracy was used as a measure of fitness. However, in an object detection task, it is necessary not only to classify the object but also to estimate the position and area where the object exists. Therefore, it is not possible to introduce the concept of correct and incorrect detection results as it is. For this reason, it is necessary to redefine the concept of fitness.

Algorithm 3 Restructure algorithm for the network structure to apply `add_concatenate`

```
1: Input: input from the immediately preceding layer (height $h_{\text{in}}$, width $w_{\text{in}}$, number of channels $c_{\text{in}}$), input from the source layer (height $h_{\text{source}}$, width $w_{\text{source}}$, channel $c_{\text{source}}$)
2: if $h_{\text{source}} = m \cdot h_{\text{in}}$ or $h_{\text{source}} = \frac{1}{n} \cdot h_{\text{in}}$ $(m, n \in \mathrm{N})$ then
3:     Pointwise convolution: connects the input from the source layer to the convolutional layer having 64 channels, one stride, $1 \times 1$ filter size, and one-pixel padding. Due to this operation, the number of channels of the input from the source layer is fixed to 64. Hence, the number of parameters is reduced, and the growth in the number of channels for the next operation is suppressed.
4:     Reorganization: performs a transformation where the height and width of the input from the source layer are, respectively, reduced by $\frac{1}{m}$ (magnified by $n$ times), and the number of channels is multiplied by $m^2$ (reduction by $\frac{1}{n^2}$). As a result of this operation, we have $h_{\text{source}} = h_{\text{in}}$, $w_{\text{source}} = w_{\text{in}}$ and the height and width dimensions of the inputs from the two layers coincide. The number of parameters does not change before and after this operation.
5:     Concatenation: the inputs from two layers whose height and width dimensions coincide are concatenated along the channel number direction to form the output of this layer.
6: else
7:     Failure: since restructuring is impossible, another mutation operation is selected.
8: end if
9: Output: formed by the concatenation of inputs from two layers along the channel number direction
```

There are several performance evaluation criteria for the object detection task, but here we consider mAP (mean Average Precision) as a fitness measure. Below is an explanation of the mAP calculation method [3]. For simplicity, the subject of detection is only one type of object.

If the image of the target of object detection is input to a network, several square areas (set of numbers stating the coordinates of the center, width, and height) are obtained. The output also includes the probability that an object exists in each rectangular area (confidence). Of the above, we only consider the rectangular areas where confidence is greater than or equal to a threshold c. Three terms are defined below:

True Positive	the number of predicted areas that was correctly "detected"
False Positive	the number of predicted areas that was incorrectly not "detected"
False Negative	the number of areas that should have been "detected" but was not.

Here a predicted area "detects" a correct area if the value of the intersection over union (IoU) is equal to or greater than 0.5. Note that IoU is a measure of overlap between the two areas. Figure 12.12 shows the definition of IoU. Hence, IoU is the ratio of the intersection of two areas to the area of the union between them. If several predicted areas cover a single correct area, only the predicted area of one of them is counted as True Positive; all other predicted areas are counted as False Positive.

Fig. 12.12 Definition of IoU

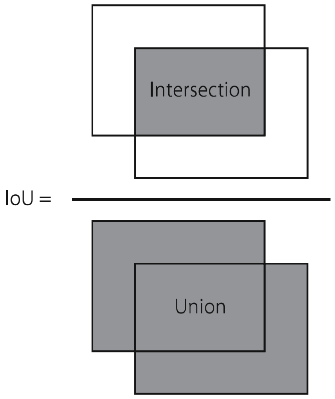
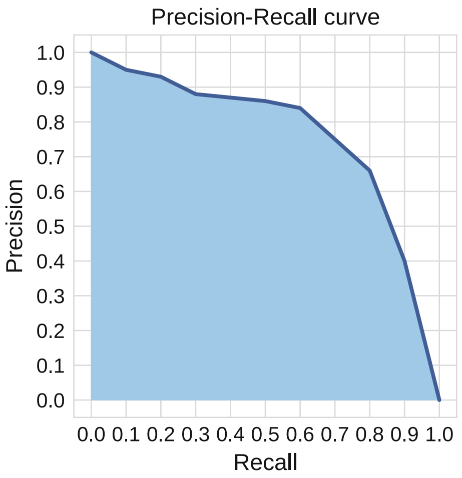


Fig. 12.13 Example of PR curve. The blue-painted region represents AP



Based on the definition above, the values of Recall and Precision are obtained by counting the number of True Positive (TP), False Positive (FP), and False Negative (FN) for all the given test images, as follows:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Recall is the ratio of "the number of actually detected objects to the number of all objects to be detected," while Precision is a ratio of "the number of areas where objects actually exist out of the areas where they are predicted to exist." Here, if the threshold value c is changed, the number of predicted areas to be considered also changes. Therefore, the values of TP, FP, and FN also change, and so do the values of Recall and Precision. Thus, a plot such as the one in Fig. 12.13 (PR curve: precision-recall curve) can be obtained by plotting Recall on the horizontal axis, Precision on the vertical axis, and connecting the points obtained when the value of threshold c is changed from 0 to 1.

The area of the region delimited by the PR curve and the axes is called AP (Average Precision) and assumes a value between 0 and 1. The higher the value of the AP, the higher is the detection accuracy.

Even if the number of classes to be detected is increased, it is possible to calculate the value of AP for each class. mAP is the average value of AP calculated for each class.

12.4.2.5 Changing the Evaluation Method for Individuals in the Elite Group of Each Generation

In the experiment described in Sect. 12.4.1, when individuals from the elite group with the highest fitness in their generation are cloned into the next generation, their fitness is just inherited without being re-evaluated. This results in an issue in the proposed method ASM+ where the number of epochs for evaluation increases with each generation. Furthermore, when individuals in the elite group are in a state of overfitting where their fitness does not improve with the number of epochs, they are not updated and enter a state of local stagnation. However, this issue was solved by also re-evaluating those individuals in the elite group just like the other ones using the number of epochs corresponding to the generation. Therefore, unlike the MNIST experiment where the fitness of the best individual improves uniformly with the generation, in some cases the fitness of the best individual decreases as the generation advances.

12.4.2.6 Experimental Results

Experiments were conducted on an NVIDIA GeForce GTX 1080 Ti GPU. The parameters used are shown in Table 12.5.

Table 12.5 Parameters used in the experiment

Parameter	Value
Final generation number T	4
Number of individuals in each generation N	4
Number k of elite individuals to be added to the next generation	1
Batch size b	64
Number of training images L_{train} (Table 12.3)	3672
Number of testing images L_{test} (Table 12.3)	2449
Minimum iteration number $i_{\min}$	10,000
Maximum iteration number $i_{\max}$	25,000

For reasons related to implementation, instead of defining the minimum number of epochs $n_{\min}$ and the maximum number of epochs $n_{\max}$, the minimum and maximum number of iterations, $i_{\min}$ and $i_{\max}$, have, respectively, been defined. Referring to the number of iterations as i and the number of epochs as n, the number of epochs is converted to the number of iterations using the equation below:

$$n = \frac{b \cdot i}{L_{\text{train}}}$$

Since batch size $b = 64$ and $L_{\text{train}} = 3672$, if we set $i_{\min} = 10{,}000$ and $i_{\max} = 25{,}000$, this is almost equivalent to setting $n_{\min} = 174$, $n_{\max} = 436$.

The method proposed ASM+ in Sect. 12.3 can be applied in the same way even if the number of epochs is replaced by the number of iterations. Hence, in order to split the interval between $i_{\min}$ and $i_{\max}$ uniformly with respect to generation number $T = 4$, the number of iterations is varied from the first up to the 4th generation in the following order: 10,000, 15,000, 20,000, and 25,000.

Figure 12.14 shows how the fitness (mAP) for an individual changed from generation to generation. Table 12.6 shows the mAP for the best individual obtained by evolution; it also shows the computation time required by the entire process. Table 12.7 shows which mutation operations were selected in each generation.

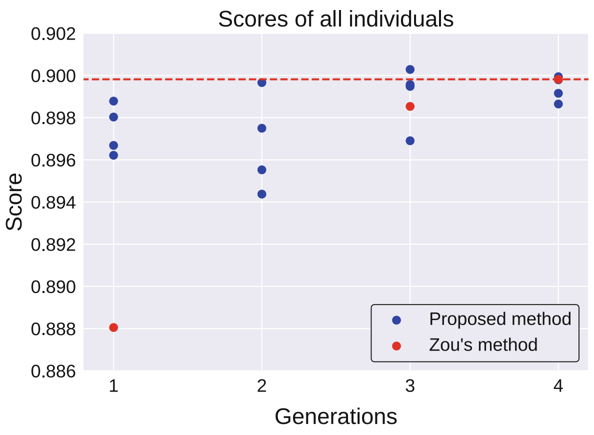


Fig. 12.14 Change of fitness of all individuals in each generation. The red dot is the result of reference [20] (Exp. 5/Run 11, including data during the training), where the number of epochs upon evaluation is plotted after conversion to the corresponding generation number

Table 12.6 Comparison of True Positive (TP), False Positive (FP), False Negative (FN), AP, mAP (converted to %), and computation times for each method

	AP							
Method	TP	FP	FN	Scissors	Knives	PET bottles	mAP	Computation time
Zou [20]	5474	517	469	90.46%	88.82%	90.67%	89.98%	7 GPUH
ASM+	5485	459	458	90.32%	89.16%	90.60%	90.03%	84 GPUH

The threshold used for the computation of TP, FP, and FN is $c = 0.25$

Table 12.7 Mutation operations performed on the best individual of each generation

Generation	Mutation operation	Maximum mAP
0	`initialize`	–
1	`add_concatenate`	0.8988
2	-	0.8997
3	`add_convolution`	0.9003
4	`add_concatenate`	0.8999

Blank fields indicate that no update was carried out on a specific individual/generation

The following Figs. 12.15 and 12.16 show the network structures of the individuals exhibiting the best and the second-best fitness during the evolution process. Figure 12.17 is a tree diagram describing the evolution process of the best individual of each generation.

However, note that, in Figs. 12.15, 12.16, and 12.17, batch normalization and Leaky ReLU are applied immediately after all Conv2D (2-dimensional convolutional layer) in all layers except the last one; however, this was abridged in the figures.

12.5 Discussion

12.5.1 Handwritten Number Classification Experiment Using MNIST

Compared with Genetic CNN [19] and aggressive selection and mutation [7], the recognition accuracy for the best individual is slightly lower; however, the level of accuracy achieved is approximately equivalent.

From the dimensional changes that occurred in the feature map of each layer of the network that was eventually obtained, we observed that the vertical and horizontal dimensions of the output feature map of the convolutional layer decreased, while the number of channels increased. This structure is also seen in classical networks such as LeNet [6] and is supposed to reflect the process of grasping the local features of the image by changing the scale.

However, the final network has an extremely simple structure formed by two convolutional layers. There is no pooling layer, dropout, or by-pass, which could have been introduced by `add_pooling`, `add_dropout`, and `add_skip`, respectively. This could be attributed to the strict conditions for the application of mutations; therefore, the operations involving the addition of such structures were not selected. Moreover, from Fig. 12.9 and Table 12.2, we observed that the addition of a convolutional layer at an early stage of evolution produces a dramatic effect

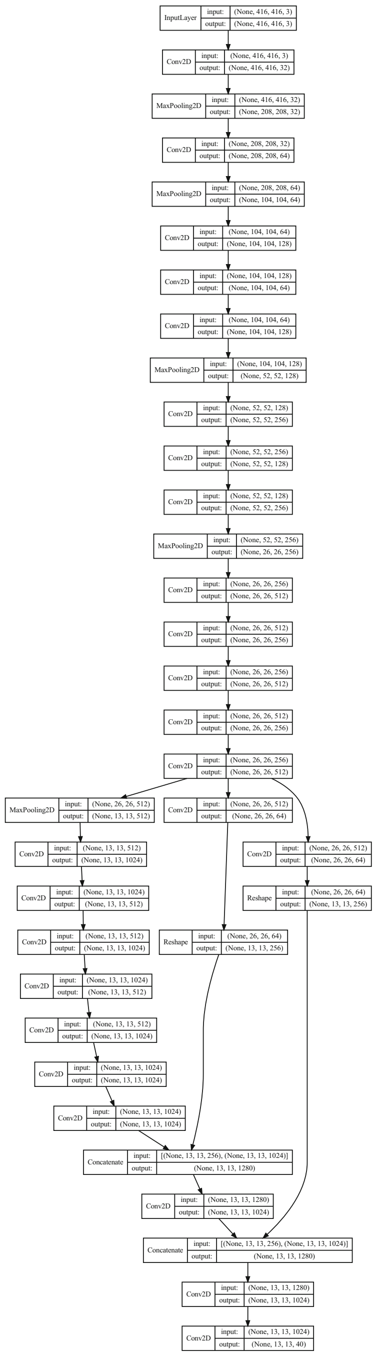

InputLayer input: (None, 416, 416, 3) output: (None, 416, 416, 3)
Conv2D input: (None, 416, 416, 3) output: (None, 416, 416, 32)
MaxPooling2D input: (None, 416, 416, 32) output: (None, 208, 208, 32)
Conv2D input: (None, 208, 208, 32) output: (None, 208, 208, 64)
MaxPooling2D input: (None, 208, 208, 64) output: (None, 104, 104, 64)
Conv2D input: (None, 104, 104, 64) output: (None, 104, 104, 128)
Conv2D input: (None, 104, 104, 128) output: (None, 104, 104, 64)
Conv2D input: (None, 104, 104, 64) output: (None, 104, 104, 128)
MaxPooling2D input: (None, 104, 104, 128) output: (None, 52, 52, 128)
Conv2D input: (None, 52, 52, 128) output: (None, 52, 52, 256)
Conv2D input: (None, 52, 52, 256) output: (None, 52, 52, 128)
Conv2D input: (None, 52, 52, 128) output: (None, 52, 52, 256)
MaxPooling2D input: (None, 52, 52, 256) output: (None, 26, 26, 256)
Conv2D input: (None, 26, 26, 256) output: (None, 26, 26, 512)
Conv2D input: (None, 26, 26, 512) output: (None, 26, 26, 256)
Conv2D input: (None, 26, 26, 256) output: (None, 26, 26, 512)
Conv2D input: (None, 26, 26, 512) output: (None, 26, 26, 256)
Conv2D input: (None, 26, 26, 256) output: (None, 26, 26, 512)
MaxPooling2D input: (None, 26, 26, 512) output: (None, 13, 13, 512)
Conv2D input: (None, 26, 26, 512) output: (None, 26, 26, 64)
Conv2D input: (None, 13, 13, 512) output: (None, 13, 13, 1024)
Conv2D input: (None, 26, 26, 512) output: (None, 26, 26, 64)
Conv2D input: (None, 13, 13, 1024) output: (None, 13, 13, 512)
Reshape input: (None, 26, 26, 64) output: (None, 13, 13, 256)
Conv2D input: (None, 13, 13, 512) output: (None, 13, 13, 1024)
Reshape input: (None, 26, 26, 64) output: (None, 13, 13, 256)
Conv2D input: (None, 13, 13, 1024) output: (None, 13, 13, 512)
Conv2D input: (None, 13, 13, 512) output: (None, 13, 13, 1024)
Conv2D input: (None, 13, 13, 1024) output: (None, 13, 13, 1024)
Conv2D input: (None, 13, 13, 1024) output: (None, 13, 13, 1024)
Concatenate input: [(None, 13, 13, 256), (None, 13, 13, 1024)] output: (None, 13, 13, 1280)
Conv2D input: (None, 13, 13, 1280) output: (None, 13, 13, 1024)
Concatenate input: [(None, 13, 13, 256), (None, 13, 13, 1024)] output: (None, 13, 13, 1280)
Conv2D input: (None, 13, 13, 1280) output: (None, 13, 13, 1024)
Conv2D input: (None, 13, 13, 1024) output: (None, 13, 13, 40)

Fig. 12.15 Network structure of an individual from the 3rd generation, which exhibited the best fitness among all individuals (fitness = 0.9003). For the notation, refer to Fig. 12.8

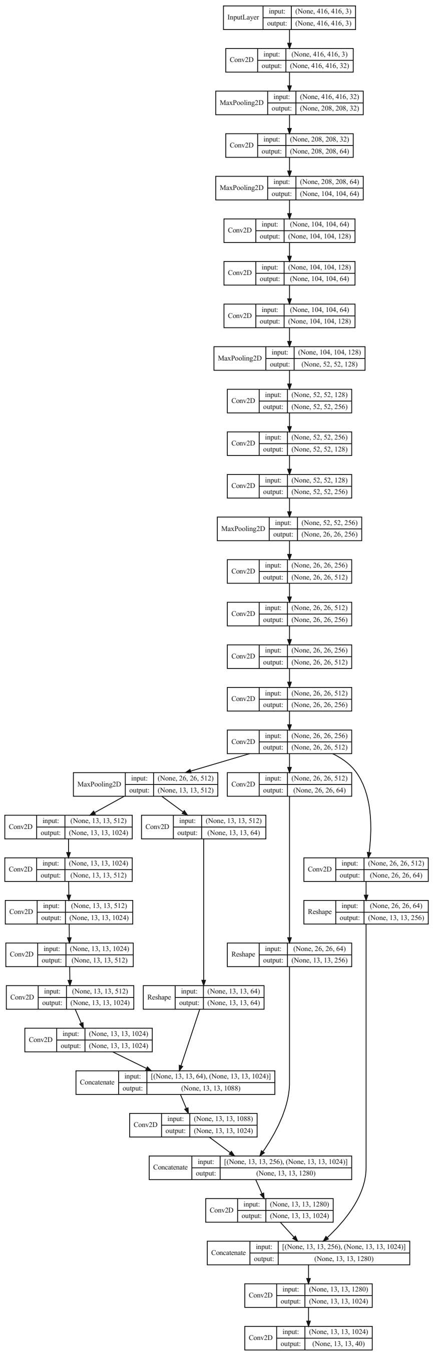

InputLayer input: (None, 416, 416, 3) output: (None, 416, 416, 3)
Conv2D input: (None, 416, 416, 3) output: (None, 416, 416, 32)
MaxPooling2D input: (None, 416, 416, 32) output: (None, 208, 208, 32)
Conv2D input: (None, 208, 208, 32) output: (None, 208, 208, 64)
MaxPooling2D input: (None, 208, 208, 64) output: (None, 104, 104, 64)
Conv2D input: (None, 104, 104, 64) output: (None, 104, 104, 128)
Conv2D input: (None, 104, 104, 128) output: (None, 104, 104, 64)
Conv2D input: (None, 104, 104, 64) output: (None, 104, 104, 128)
MaxPooling2D input: (None, 104, 104, 128) output: (None, 52, 52, 128)
Conv2D input: (None, 52, 52, 128) output: (None, 52, 52, 256)
Conv2D input: (None, 52, 52, 256) output: (None, 52, 52, 128)
Conv2D input: (None, 52, 52, 128) output: (None, 52, 52, 256)
MaxPooling2D input: (None, 52, 52, 256) output: (None, 26, 26, 256)
Conv2D input: (None, 26, 26, 256) output: (None, 26, 26, 512)
Conv2D input: (None, 26, 26, 512) output: (None, 26, 26, 256)
Conv2D input: (None, 26, 26, 256) output: (None, 26, 26, 512)
Conv2D input: (None, 26, 26, 512) output: (None, 26, 26, 256)
Conv2D input: (None, 26, 26, 256) output: (None, 26, 26, 512)
MaxPooling2D input: (None, 26, 26, 512) output: (None, 13, 13, 512)
Conv2D input: (None, 26, 26, 512) output: (None, 26, 26, 64)
Conv2D input: (None, 13, 13, 512) output: (None, 13, 13, 1024)
Conv2D input: (None, 13, 13, 512) output: (None, 13, 13, 64)
Conv2D input: (None, 13, 13, 1024) output: (None, 13, 13, 512)
Conv2D input: (None, 26, 26, 512) output: (None, 26, 26, 64)
Conv2D input: (None, 13, 13, 512) output: (None, 13, 13, 1024)
Reshape input: (None, 26, 26, 64) output: (None, 13, 13, 256)
Conv2D input: (None, 13, 13, 1024) output: (None, 13, 13, 512)
Reshape input: (None, 26, 26, 64) output: (None, 13, 13, 256)
Conv2D input: (None, 13, 13, 512) output: (None, 13, 13, 1024)
Reshape input: (None, 13, 13, 64) output: (None, 13, 13, 64)
Conv2D input: (None, 13, 13, 1024) output: (None, 13, 13, 1024)
Concatenate input: [(None, 13, 13, 64), (None, 13, 13, 1024)] output: (None, 13, 13, 1088)
Conv2D input: (None, 13, 13, 1088) output: (None, 13, 13, 1024)
Concatenate input: [(None, 13, 13, 256), (None, 13, 13, 1024)] output: (None, 13, 13, 1280)
Conv2D input: (None, 13, 13, 1280) output: (None, 13, 13, 1024)
Concatenate input: [(None, 13, 13, 256), (None, 13, 13, 1024)] output: (None, 13, 13, 1280)
Conv2D input: (None, 13, 13, 1280) output: (None, 13, 13, 1024)
Conv2D input: (None, 13, 13, 1024) output: (None, 13, 13, 40)

Fig. 12.16 Network structure of an individual from the 4th generation whose network structure yielded the second largest fitness among all individuals (fitness = 0.8999)

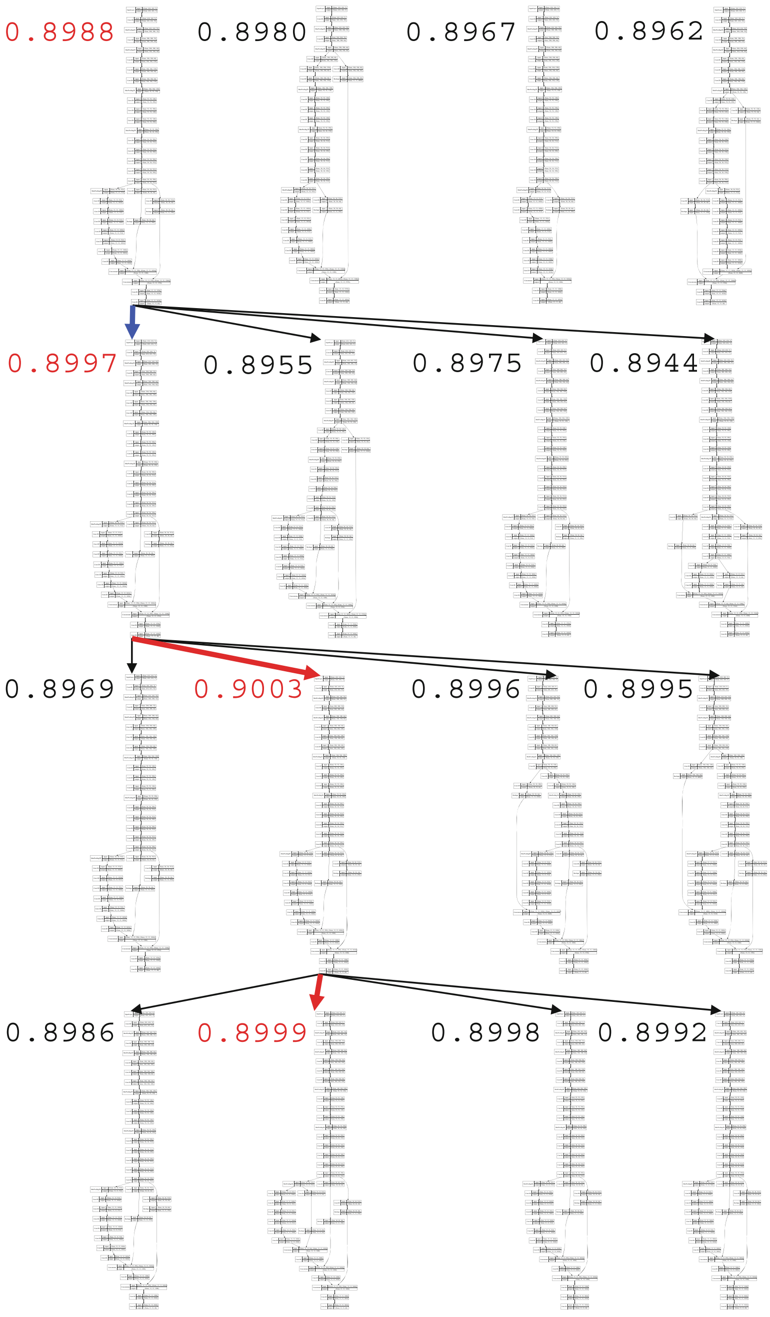


Fig. 12.17 Tree diagram showing the evolution process of the best individual of each generation. The value accompanying the image of the network structure of an individual represents its fitness (mAP). Individuals with a value written in red are those selected for the next generation. Colored arrows indicate the selection flow. Red arrows indicate the occurrence of mutation, and blue arrows indicate cloning

on recognition accuracy; subsequently, only operations that change the hyperparameters in the convolutional layer (`alter_stride`, `alter_filter_size`, `alter_channel_number`) are selected as operations that result in the birth of better individuals.

Furthermore, from Fig. 12.7, we observe that evolution saturates around the 15th generation. Similarly, the selection of operations to be applied to the individuals of that generation is restricted, resulting in the selection of locally optimized solutions. Another factor is that since the number of epochs increases as the generations advance, overfitting tends to occur in new individuals, and the best individuals end up not being updated.

Conversely, the computation time for the entire evolution process is shorter than the previous results [19] and [7]. This is possibly due to a persisting situation where only networks with simple structures that require short training times were obtained.

The presence or absence of batch normalization did not produce a significant difference in the results.

12.5.2 Experiment on Detecting Dangerous Objects in X-ray Images

Compared with the results of the method used by Zou [20], individuals showing an improvement of 0.05% in the mAP value were obtained (Table 12.6). The number of “detected” dangerous objects (True Positive) also increased. Additionally, the number of other objects wrongly detected as dangerous (False Positive) decreased, and the number of dangerous objects that were not detected (overlooked) also decreased (False Negative). Therefore, for the dangerous object detection task, both “Recall” and “Precision” improved, which denotes a definite performance improvement. Figure 12.18 shows an example of a dangerous object that was not detected by Zou’s method but was detected by the proposed method ASM+.

In the tree diagram of Fig. 12.17, we can find several by-passes generated by `add_concatenate` in the structures that were actually obtained. This can be contrasted with the case of the MNIST handwritten number classification task, where no by-pass was observed due to restrictions on the application of mutation operations (Sect. 12.5.1). Our assumption is that the network structure reconstruction algorithm for applying `add_concatenate` introduced in Sect. 12.4.2.3 has produced effective results.

The network structure of one of the two individuals that yielded a better mAP than Zou’s method, which exists in the last generation shown in Fig. 12.16, is examined closely. In that structure, three by-pass lines stretch out from a layer with a 26×26 (width, height) output towards a layer having a 13×13 input located closer to the final layer. This structure is similar to the U-type structure of U-Net [14], which produced good results in medical image segmentation. As mentioned in Sect. 12.2.4, employing a by-pass from the high-resolution output of a layer with a large feature

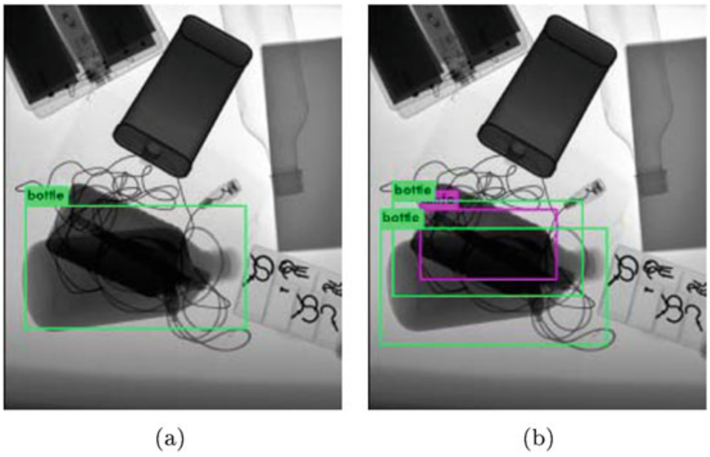


Fig. 12.18 Example of where an image that was overlooked by the Zou's method [20] was correctly detected by the proposed method ASM+. Images of two PET bottles and a knife are overlapped in the center of the image. (**a**) Detection results in Zou's method. (**b**) Detection results by ASM+

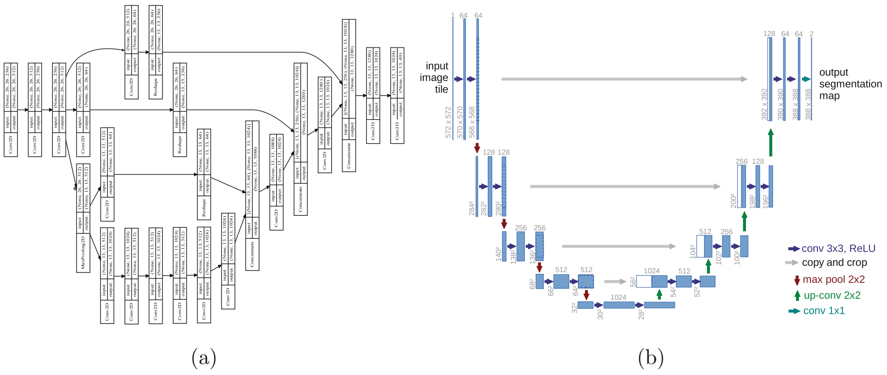


Fig. 12.19 Comparison of evolved network structures. (**a**) Aggressive selection and mutation. (**b**) U-Net [14]

map to a layer with a small feature map acts to improve the ability to detect small objects (see Fig. 12.19).

Next, from Fig. 12.14, we observe that the fitness of the best individual of the 4th generation is lower than that of the 3rd generation. This is an indication that the high number of epochs used in the evaluation results in overfitting and deterioration in the mAP value in individuals that are cloned into the next generation as an elite. As a result, the proposed method ASM+ of increasing the number of epochs as the generations advance has the advantage of finding an optimal number of epochs that does not result in overfitting.

12.6 Conclusion

Optimizing the structure of convolutional neural networks is a task that requires high-level specialized knowledge and a huge amount of trial-and-error work. To enable the automatic search of optimal network structures using GA, we proposed a method ASM+ in which the number of epochs upon evaluation is increased as the number of generations advance during the evolution process in the present research.

In the MNIST handwritten number classification experiment, we showed that ASM+ exhibited a slightly lower recognition accuracy, but is relatively fast in terms of computation time. However, this might be because the algorithm was not able to search for complex structures.

In the experiment related to detecting dangerous objects in X-ray images, we introduced a method that made it easier for individuals to undergo mutation in view of the results of the MNIST experiments where only simple structures emerged from the search. Consequently, it became possible to evaluate individuals possessing several by-pass structures, resulting in an enlarged search space. Additionally, with respect to cloned individuals belonging to the elite group, a change was introduced where fitness is re-evaluated with a higher number of epochs instead of being inherited into the next generation.

Due to this change, we avoided the problem of individuals becoming overfitted with an increasing number of epochs and stuck in local solutions. Moreover, we can consequently detect the optimal number of epochs that do not cause overfitting.

A future task is the acceleration of evolution. The proposed method ASM+ requires at least 10 times more computation than the Zou's method [20], as is shown in Table 12.6. Pham et. al. [11], as in ASM+, attempted to optimize the convolutional neural network structure by using neural networks instead of GA. A considerable reduction in computation needed to evaluate individuals may be possible through a transfer learning approach where the weight parameters are shared to some extent.

Acknowledgments We would like to thank Shin Yokoshima and Yoji Nikaido, T&S Corporation, for providing us with X-ray data collection environment.

References

1. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. J. Mach. Learn. Res. **12**, 2493–2537 (2011)
2. Dahl, G.E., Yu, D., Deng, L., Acero, A.: Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. IEEE Trans. Audio Speech Lang. Process. **20**(1), 30–42 (2012)
3. Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2012 (VOC2012). http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html (2012). Accessed 07 Feb 2019

4. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (2016)
5. Krizhevsky, A., Sutskever, I., Hinton, G.: ImageNet classification with deep convolutional neural networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS12), pp.1097–1105 (2012)
6. LeCun,Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
7. Li, Z., Xiong, X., Ren, Z., Zhang, N., Wang, X., Yang, T.: An Aggressive Genetic Programming Approach for Searching Neural Network Structure Under Computational Constraints (2018). Preprint arXiv:1806.00851
8. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: Proceedings of International Conference on Machine Learning (ICML), vol. 30, no.1 (2013)
9. Nie, Y., Iba, H.: evolving convolutional neural network architectures using self-modifying Cartesian genetic programming. Master Thesis, Graduate School of Information Science and Technology, The University of Tokyo, 2019
10. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Trans. Knowl. Data Eng. **22**(10), 1345–1359 (2010)
11. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient Neural Architecture Search via Parameter Sharing (2018). Preprint arXiv: 1802.03268
12. Redmon, J., Farhadi, A.: YOLO9000: Better, Faster, Stronger (2016). Preprint arXiv: 1612.08242
13. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You Only Look Once: Unified, Real-Time Object Detection (2015). Preprint arXiv: 1506.02640
14. Ronneberger, O., Fischer, P., Brox, T.: U-Net: Convolutional Networks for Biomedical Image Segmentation (2015). Preprint arXiv: 1505.04597
15. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, Z., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet large scale visual recognition challenge. Int. J. Comput. Vis. **115**(3), 115–211 (2015)
16. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evol. Comput. **10**(2), 99–127 (2002)
17. 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 2017 (GECCO2017), pp. 497–504 (2017)
18. 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 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.1–9 (2015)
19. Xie, L., Yuille, A.: Genetic CNN. In: Proceedings of the 2017 IEEE International Conference on Computer Vision (ICCV), pp.1388–1397 (2017)
20. Zou, L., Tanaka, Y., Iba, H.: Dangerous objects detection of X-ray images using convolution neural network. In: Yang, C.N., Peng, S.L., Jain, L. (eds.) Security with Intelligent Computing and Big-data Services (SICBS2018). Advances in Intelligent Systems and Computing, vol. 895, pp. 714–728. Springer, Cham (2019)