Chapter 11
Differential Evolution for Architecture Design

11.1 Introduction

The general goal of this chapter is to explore the capacity of DE, named DECNN, to evolve deep CNN architectures and parameters automatically. Designing new crossover and mutation operators of DE, as well as an encoding scheme, and a second crossover operator will help to achieve the goal. DECNN will be evaluated on six datasets of various complexity that are widely used and compared to 12 state-of-the-art methods. The objectives are as follows:

- Breaking the limitation of predefining CNN maximal depth by refining the existing effective encoding scheme which is employed by IPPSO;
- Developing and designing novel crossover and mutation operators for DECNN, which may be used on vectors with variable length to overcome the fixed-length constraint of standard DE method;
- Designing and incorporating a second crossover operator for DECNN to generate offspring in the following generations that represent the CNN architectures which differ from their parents in length.

11.2 Algorithm Details

To meet the need of evolving variable-length CNN architectures, DECNN employs the DE as the core EA and designs a second crossover operator to produce offspring with lengths different from their parents..

Y. Sun et al., *Evolutionary Deep Neural Architecture Search: Fundamentals, Methods, and Recent Advances*, Studies in Computational Intelligence 1070,
https://doi.org/10.1007/978-3-031-16868-0_11

11.2.1 DECNN Algorithm Overview

The framework of DECNN is stated in Algorithm 1. First of all, the population P is initialized by the designed population initialization method, which will be discussed in Sect. 11.2.3, and then the a set P_best storing the individuals with best fitness values is initialized. After that, the algorithm starts the evolution, until the termination criterion is met. In each evolutionary generation, the refined crossover and mutation of DE are applied first, after that, the designed second crossover is utilized to generate two offspring, and the best on between the two offspring and their parents is chosen. next, the fitness values of the individuals newly generated are evaluated, and the individuals having best fitness in the population are placed into P_best. In the following, the core components in DECNN are discussed.

Algorithm 1: Framework of DECNN

```
1 P ← Initializing the population detailed in Sect. 11.2.3;
2 P_best ← empty;
3 while the termination criterion has not been met do
4 |   Applying the refined crossover and mutation of DE detailed in Sect. 11.2.5;
5 |   Applying the designed second crossover to generate two offspring, and the best one
  |   between the two offspring and their parents is selected as detailed in Sect. 11.2.6;
6 |   Evaluating each individual to get the fitness value;
7 |   P_best ← Retrieving the individual with best fitness value in the population;
8 end
```

11.2.2 Adjusted IP-Based Encoding Strategy

The layer of DNNs is represented by a IP address in the designed IP-based encoding strategy, and then the IP address is pushed into a series of interfaces. Each of them contains an IP address in addition to the subnet that corresponds to it, in the same way that the layers in DNNs are ordered. CNNs, for example, are made up of three types of layers: the convolutional layer, the pooling layer, as well as the fully connected layer. The initial stage in the encoding process is to determine the range that can be used to indicate each attribute of each CNN layer type. Although the attributes of CNN layers have no specified boundaries, for the purpose of applying optimization algorithms to the task, it is necessary to provide a range with sufficient capacity so the optimal accuracy on the classification tasks can be achieved. The restrictions for each attribute in this chapter are designed with the goal of achieving a low mistake rate. To be more specific, there are three attributes for the convolutional layer: the number of feature maps that is ranging from 1 to 128, the size of stride that is ranging from 1 to 4, and the size of filter that is ranging from 1 to 8. Because these three

Table 11.1 The three attributes of CNN layers with their corresponded range

Layer type	Range	# of bits	Parameter
Convolutional layer	[1,8]	3	Filter size
	[1,128]	7	# of feature maps
	[1,4]	2	Stride size
		12	Total
Pooling layer	[1,4	2]	Kernel size
	[1,4]	2	Stride size
	[1,2]	1	Type: 1(MAX), 2(AVERAGE)
		5	Total
Fully connected layer	[1,2048]	11	# of Neurons
		11	Total

attributes must be concatenated into a single value, a bits binary string can hold all three convolutional layer attributes: 7 bits to hold the number of feature maps, 2 bits to hold the stride size, and 3 bits to hold the filter size. The fully connected layer and the pooling layer can be hold in binary strings with 11 bits and 5 bits, respectively, in a similar way. Table 11.1 shows the range of each feature in detail.

After determining the total number of bits included in a binary string, a particular CNN layer is then easily capable of being converted into a binary string. If a convolutional layer with the feature maps number of 32, the stride size of 2, and the filter size of 2, it may calculate the binary strings [000 1111], [01], and [001] through transforming the decimal numbers. This is because the binary string begins with 0, and the decimal attribute value of CNN layers begins with 1, 1 is deducted from the decimal number before conversion. By combining the binary strings of each attributes, the final binary string [001 000 1110 01] that represents the supplied convolutional layer is formed. Figure 11.1 shows the details of the example.

In the same way that a subnet must be defined before assigning an IP address to the interface, such as a desktop or laptop, a subnet need to be designed by the IP-based encoding strategy for each CNN layer type. Because the size of the search space is determined by the amount of bits used by each layer type, and the pooling layer uses far less bits compared with the other two, the probabilities of choosing a pooling layer are significantly lower than the other two. To equalize the probabilities of choosing each layer type, the binary string corresponding to the pooling layer occupies 11 bits after having a place-holder occupies 6 bits appended to it, bringing the probabilities of choosing a pooling layer to the identical level as choosing a fully connected layer. Because there are three layer types, each with a maximum bit count of 12, hence a binary string of 2 bytes can carry the encoded CNN layers. The IP address of convolutional layer will be represented by the 2-byte binary starting with [0000 0000 0000 0000], and the binary representation of the finishing IP address is going to be [0000 1111 1111 1111]. By adding 1 to the last IP address of the

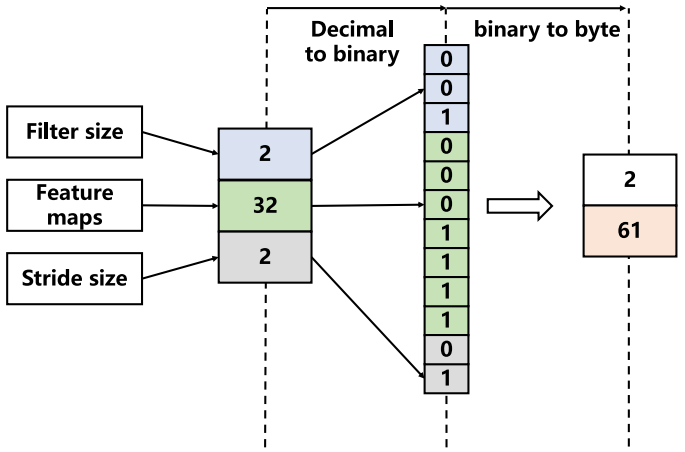

Fig. 11.1 An illustration of how a convolutional layer is encoded by a byte array

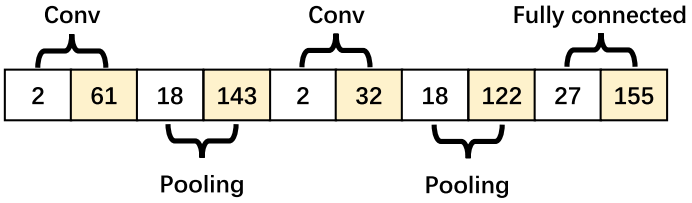

Fig. 11.2 An example of how a vector is encoded from a CNN architecture

convolutional layer, the starting IP of the 11 bits fully connected layer is [0001 0000 0000 0000], and the finishing IP address is [0001 0111 1111 1111]; likewise, the IP address of the pooling layer starts from [0001 1000 0000 0000] and ends with [0001 1111 1111 1111]. In conclusion, the 2-byte style IP ranges for every subset can be summarized as follows: the IP address of the convolutional layer ranges from 0.0 to 15.255, the IP address of the fully connected layer ranges from 16.0 to 23.255, and the IP address of the pooling layer ranges from 24.0 to 31.255, which are calculated by translating the binary strings mentioned previously to binary strings of 2 bytes. Now a CNN layer is ready to be encoded into an IP address, the convolutional layer depicted in Fig. 11.1 is used as an illustration here. The IP address is represented by the binary string [0000 0010 0011 1001], which may be transformed to a 2-byte form IP address [2.61] by adding the starting IP of the convolutional layer subnet and the binary string of the convolutional layer together. An example vector can be seen in Fig. 11.2, which is encoded from a CNN architecture consist of two convolutional layers, two pooling layers, and one fully connected layer.

11.2.3 Population Initialization

Since lengths of individuals must be diverse, the population initialization begin by generating individuals with random length. The length is picked at random from a Gaussian distribution whose standard deviation ρ equals 1 and center μ is set to a predetermined length according to how complex the classification task are, as indicated in Eq. (11.1). After determining the length of candidate, the attribute values and layer type for each layer in the candidate can be produced at random. To complete the population initialization, the process is repeated until the population size is reached.

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{11.1}$$

11.2.4 Fitness Evaluation

Algorithm 2 illustrates the process of evaluating fitness. First, the fitness evaluation process receives four arguments, the candidate solution for representing an encoded architecture of CNN, the training set for training the CNN architecture, the number of training epoch to train the CNN architecture which is decoded from the candidate solution, and the dataset for fitness evaluation, which is employed to get the fitness value by testing accuracy of the trained model on the dataset. Second, the process of fitness evaluation is straightforward, it employs the back propagation as the training algorithm and a predetermined training epoch number to train the CNN architectures on the training set, and then tests the trained architectures on the testing set to get the accuracy as their fitness value. Only partial dataset is used to train the candidate CNN with a restricted epoch number to reduce computational cost, which are regulated by the arguments of fitness function—k, D_train and $D\_fitness$.

Algorithm 2: Fitness Evaluation

Input: The training set D_train, the training epoch number k, the candidate solution c, the dataset of fitness evaluation $D\_fitness$.

Output: The evaluated fitness value $fitness$.

1 Training the weights of the CNN which is represented by the candidate c for k epochs on the training set D_train;

2 $acc \leftarrow$ Evaluating the trained CNN on the dataset $D\_fitness$ of fitness evaluation;

3 $fitness \leftarrow acc$;

4 **Return** $fitness$.

11.2.5 DECNN DE Mutation and Crossover

The mutation and crossover operations of the DECNN are analogous to the crossover and mutation operations of standard DE, but an extra step is added for trimming the vectors with longer length before carrying out any operation. This is due to the fact that the DECNN candidates are of varying lengths and the traditional operations of DE in Eqs. (1.4) and (1.3) are only applicable to vectors of a fixed length. To be more specific, the three random vectors that are used for the mutation have their lengths trimmed down to the shortest length of the three. Moreover, if the length of the trial vector that is produced by the mutation has longer length compared to the length of its parent, it will be shortened until it is equal to the length of its parent.

11.2.6 DECNN Second Crossover

By slicing the vector at the cutting points, each individual of the two parents is separated into two parts, and then one of those parts is exchanged with the other, similar to the crossover of GAs. To regulate population variation, the cutting point is chosen at random using a Gaussian distribution in which a hyperparameter ρ serves as the standard deviation and the middle point serves as the center. Figure 11.3 depicts how the second crossover works.

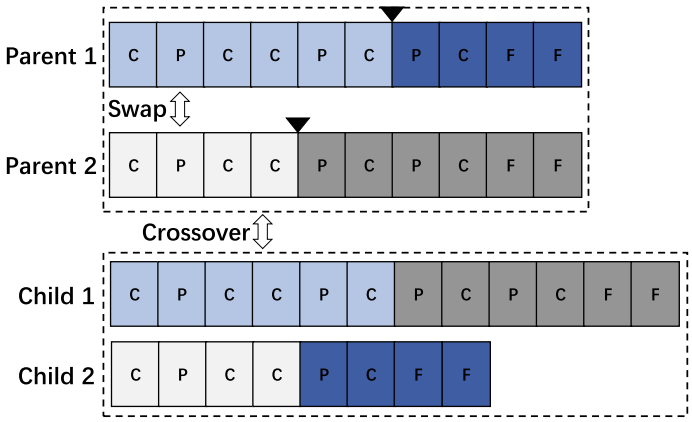

Fig. 11.3 An example of the second crossover in DECNN

11.3 Experimental Design

11.3.1 State-of-the-Art Competitors

On the benchmark datasets that are mentioned before in the literature [1], six state-of-the-art algorithms have been reported to have obtained promising performances. As a result, they are chosen as the peer competitors of DECNN. They are RandNet-2 [1], LDANet-2 [1], SAA-3 [2], ScatNet-2 [3], CAE-2 [4], PCANet-2 (Softmax) [1], TIRBM [5], SVM+Poly [2], SVM+RBF [2], DBN-3 [2], PGBM+DN1 [6], and NNet [2].

11.3.2 Parameter Settings

All of the parameters are set up in accordance with community rules of DE [7], with a small population factored in to reduce time for computation and search space complexity. The population size is set to 30 and the number of generations is set to 20 for the evolutionary process. Only 10% of the training dataset is used as evaluation dataset in the fitness evaluation, and the training epochs is set to 5. The differential rate and crossover rate are set to 0.6 and 0.45, respectively, in the DE parameters. DECNN performs 30 independent runs on each of the benchmark datasets with the hyperparameters ρ for second crossover and μ for population initialization set to 2 and 10 respectively.

11.4 Experimental Results and Analysis

Due to the stochastic nature of DE, a test of statistical significance is necessary in order to produce a compelling result for the comparison. When DECNN is compared with the state-of-the-art methods, One sample T-Test is employed to test if DECNN achieves better results; when comparing error rates between the peer EC competitor dubbed IPPSO [8] and DECNN, and two sample T-Test is employed for the purpose of determining if the difference is statistically significant. Table 11.2 displays the classification results of DECNN against state-of-the-art algorithms, whereas Table 11.3 compares IPPSO with DECNN.

11.4.1 DECNN Versus State-of-the-Art Methods

Table 11.2 shows the experimental results and a comparison of DECNN with state-of-the-art methods. The terms (+) and (–) are used to represent whether the result

Table 11.2 The classification errors of DECNN in comparison to the chosen peer competitors

classifier	MB	CONVEX	MDRBI	MBI	MRD	MRB
TIRBM	–	–	–	35.5 (–)	4.2 (–)	–
CAE-2	2.48 (+)	–	45.23 (+)	15.5 (+)	9.66 (+)	10.9 (+)
ScatNet-2	1.27 (–)	6.5 (–)	50.48 (+)	18.4 (+)	7.48 (+)	12.3 (+)
PGBM+DN-1	–	–	36.76 (=)	12.15 (+)	–	6.08 (=)
PCANet-2 (Softmax)	1.4 (–)	4.19 (–)	35.86 (–)	11.55 (+)	8.52 (+)	6.85 (+)
RandNet-2	1.25 (–)	5.45 (–)	43.69 (+)	11.65 (+)	8.47 (+)	13.47 (+)
SVM+RBF	30.03 (+)	19.13 (+)	55.18 (+)	22.61 (+)	11.11 (+)	14.58 (+)
LDANet-2	1.05 (–)	7.22 (–)	38.54 (+)	12.42 (+)	7.52 (+)	6.81 (+)
NNet	4.69 (+)	32.25 (+)	62.16 (+)	27.41 (+)	18.11 (+)	20.04 (+)
SVM+Poly	3.69 (+)	19.82 (+)	54.41 (+)	24.01 (+)	15.42 (+)	16.62 (+)
DBN-3	3.11 (+)	18.63 (+)	47.39 (+)	16.31 (+)	10.3 (+)	6.73 (+)
SAA-3	3.46 (+)	18.41 (+)	51.93 (+)	23 (+)	10.3 (+)	11.28 (+)
DECNN (best)	1.032	7.992	32.852	5.666	4.066	3.458
DECNN (mean)	1.457	11.192	37.551	8.685	5.533	5.558
DECNN (standard deviation)	0.113	1.943	2.447	0.447	1.405	1.711

Table 11.3 Classification rates of IPPSO and DECNN

	MB	CONVEX	MDRBI	MBI	MRD	MRB
IPPSO (standard deviation)	0.170	2.135	5.380	1.835	0.712	1.543
IPPSO (mean)	1.558	12.645	38.791	9.857	6.072	6.255
DECNN (standard deviation)	0.113	1.943	2.447	1.405	0.447	1.711
DECNN (mean)	1.457	11.192	37.551	8.685	5.533	5.558
P-value	**0.01**	**0.01**	0.2554	**0.01**	**0.001**	0.1027

achieved by DECNN is superior to or inferior to the best result achieved by the comparable peer rival, in order to clearly present the comparison results. The term (=) represent the mean error rate achieved by DECNN is slightly lower or higher compared to the competitor, but the significance difference is not supported from a statistical standpoint. The term (–) indicates that the results of provider are not available or cannot be counted.

Table 11.2 shows that the results achieved by DECNN in terms of error rates are encouraging. To be more specific, on both the MB and CONVEX benchmark datasets, DECNN ranks fifth. On the MBI benchmark, DECNN outperforms all state-of-the-art methods. On the MDRBI dataset, DECNN achieves the fourth best mean error rate. However the P-value from one sample T-Test between the third best and DECNN is 0.0871, indicating that the difference is not statistically significant, hence DECNN is tied for the third with PGBM+DN-1. On the MRB benchmark, DECNN achieves mean error rate that is lower compared to all other methods. However, there is not a significant difference between the second best algorithm and DECNN given the calculated P-value of 0.1053, hence DECNN is tied for first with PGBM+DN-1. On the MRD benchmark, DECNN outruns all other methods except TIRBM. Furthermore, when comparing the best results of DECNN with the 12 state-of-the-art algorithms on five out of the six datasets, DECNN obtains the lowest error rates: 32.852% on MDRBI, 4.066% on MRD, 3.458% on MRB, 1.032% on MB, and 5.666% on MBI. This demonstrates that DECNN is able to enhance the results obtained by the state of the arts.

11.4.2 DECNN Versus IPPSO

Table 11.3 shows that when the results of DECNN is compared with IPPSO, the mean error rates achieved by DECNN are lower on all six benchmark datasets, as well as the standard deviations of DECNN are lower on five of the six datasets. This indicates that DECNN outperforms IPPSO in terms of the overall performance. Since a kind of local search is performed in terms of the parameters of CNN architectures and their depth between the two parents and their offspring, the second crossover operator increases DECNN performance.

11.4.3 Evolved CNN Architecture

After the evolved architectures of CNN being evaluated, it is discovered that DECNN is capable of evolving the length of architecture. Individual lengths are approximately 10 when the evolutionary process begins, but the evolved CNN architectures lengths drop to between 3 and 5 according to how complex the datasets are, demonstrating that DECNN can effectively evolve CNN architectures of any length.

11.5 Chapter Summary

In this chapter, we introduced the DECNN method, which is a DE-based NAS algorithm. In particular, a new genetic operation including the crossover operator and the mutation operator is designed in DECNN. In addition, an encoding scheme, as well as a second crossover operator, are also developed in DECNN. These designs can collectively help DECNN to perform well on image classification tasks. DECNN is firstly compared with state-of-the-art peer competitors and then compared with IPPSO which was introduced in the previous chapter. Both quantitative comparisons demonstrated the effectiveness of DECNN. In addition, the architecture resulting from DECNN in the experiments has also been described.

All the algorithms introduced in this part concern the algorithmic designs, and the experiments are all about general image classification tasks. In the next chapter, we will introduce an architecture design algorithm focusing on the applications to the images of the hyper spectrum.

References

1. Chan, T.-H., Jia, K., Gao, S., Jiwen, L., Zeng, Z., & Ma, Y. (2015). Pcanet: A simple deep learning baseline for image classification? *IEEE Transactions on Image Processing, 24*(12), 5017–5032.
2. Larochelle, H., Erhan, D., Courville, A., Bergstra, J., & Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning* (pp. 473–480). ACM. https://doi.org/10.1145/1273496.1273556.
3. Bruna, J., & Mallat, S. (2013). Invariant scattering convolution networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 35*(8), 1872–1886. https://doi.org/10.1109/tpami.2012.230.
4. Rifai, S., Vincent, P., Muller, X., Glorot, X., & Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning* (pp. 833–840).
5. Sohn, K., Lee, H. (2012). *Learning invariant representations with local transformations*. https://arxiv.org/abs/1206.6418.
6. Sohn K., Zhou G., Lee C., & Lee, H. (2013). *Learning and selecting features jointly with point-wise gated boltzmann machines*. https://dl.acm.org/citation.cfm?id=3042918.
7. Gamperle, R., Muller, S. D., & Koumoutsakos, A. (2002). A parameter study for differential evolution. *NNA-FSFS-EC 2002, 10*, 293–298.
8. Wang, B., Sun, Y., Xue, B., & Zhang, M. (2018a). Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. In *2018 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1–8). IEEE.