Chapter 8
Fast Evolution of CNN Architecture for Image Classification

Ali Bakhshi, Stephan Chalup, and Nasimul Noman

Abstract The performance improvement of Convolutional Neural Network (CNN) in image classification and other applications has become a yearly event. Generally, two factors are contributing to achieving this envious success: stacking of more layers resulting in gigantic networks and use of more sophisticated network architectures, e.g. modules, skip connections, etc. Since these state-of-the-art CNN models are manually designed, finding the most optimized model is not easy. In recent years, evolutionary and other nature-inspired algorithms have become human competitors in designing CNN and other deep networks automatically. However, one challenge for these methods is their very high computational cost. In this chapter, we investigate if we can find an optimized CNN model in the classic CNN architecture and if we can do that automatically at a lower cost. Towards this aim, we present a genetic algorithm for optimizing the number of blocks and layers and some other network hyperparameters in classic CNN architecture. Experimenting with CIFAR10, CIFAR100, and SVHN datasets, it was found that the proposed GA evolved CNN models which are competitive with the other best models available.

8.1 Introduction

In recent years many pieces of research have been directed towards designing deep neural networks (DNNs). The performance of DNNs is very depended on its architecture and its hyperparameters' setting. The state-of-the-art DNN models are designed by qualified experts in various areas of machine learning. Moreover, all of these networks are designed for specific problems or data. For example, convolutional neural networks (CNNs) are most widely used in various image related applications in computer vision. Although the state-of-the-art DNNs proposed in the literature can be used for solving similar problems using some

A. Bakhshi · S. Chalup · N. Noman (✉)
The University of Newcastle, Newcastle, NSW, Australia
e-mail: ali.bakhshi@uon.edu.au; stephan.chalup@newcastle.edu.au; nasimul.noman@newcastle.edu.au

H. Iba, N. Noman (eds.), *Deep Neural Evolution*, Natural Computing Series,
https://doi.org/10.1007/978-981-15-3685-4_8

techniques like transfer learning, the same network model is not suitable for a diverse class of problems. For best performance, we need to design a DNN tailored to the problem under consideration. Consequently, many researchers are working towards automatic methods that can identify suitable DNN architecture as well as hyperparameters for a certain problem.

Evolutionary algorithms (EAs) are a class of the generic population-based meta-heuristic optimization algorithms that can be used to identify the suitable network architecture and hyperparameters [1]. There are remarkable efforts in the literature that used the variants of evolutionary algorithms such as the genetic algorithm (GA) and particle swarm optimization (PSO) to solve a variety of optimization problems [2]. Considering the promising success of the artificial neural networks (ANNs) and evolutionary algorithm in solving different machine learning problems, finding efficient ways to combine these two methods has been an active research area for the past two decades. There exists a good survey that classifies different approaches for combining ANNs and GAs into two categories: the supportive combination and collaborative combination [3]. In supportive combination, either GA or ANN is the main problem solver, and the other assists it in accomplishing that, whereas, in collaborative combination, both GA and ANN work in synergy to solve the problem. An example of a supporting combination is using GA for selecting features for a neural network, and an example of a collaborative combination is designing the ANN topology using GA.

With the emergence of DNNs as a powerful machine learning method for solving different problems, there has been a growing interest in designing and training these networks using evolutionary algorithms. Considering the success of gradient base algorithms in training the DNNs, and due to other considerations such as very large search space, there has been limited interest in training the DNNs using evolutionary algorithms. Although there exist examples of outstanding efforts in training deep neural networks for reinforcement learning using GA [4], the majority of researches concentrated on evolving DNN architectures and finding the best combination of hyperparameters for a range of classification and regression tasks [1].

Convolutional Neural Network (CNN) is one of the most successful deep architectures as manifested by its remarkable achievements in many real-world applications. The state-of-the-art CNN architectures such as VGGNet [5], ResNet [6], GoogLeNet [7], designed by experienced researchers, exhibited performance competitive to humans. However, crafting such powerful and well-designed networks requires extensive domain knowledge and expertise in neural network design. These requirements often make it a difficult task for inexperienced researchers and application engineers to design a suitable architecture according to the problem and available data. Hence, in recent years, we have seen several attempts to automatically designing CNN architectures as well as network hyperparameters using evolutionary algorithms.

In this work, we used a conventional GA to evolve optimized CNN architectures and to find the best combination of hyperparameters for the image classification task on multiple datasets. We considered the optimization of the classical CNN architecture (i.e., VGG-like networks) consisting of blocks of convolutional layers.

The proposed GA was used to optimize the number of convolutional blocks, as well as the number of layers in each block. Using a fixed-sized chromosome, we explored the search space of CNN architectures with the variable number of layers. The algorithm also searched for the optimal set of hyperparameters for the network from the selected ranges.

One particular challenge in the evolution of all kinds of DNNs is the high computational cost. The computational burden originates from the fitness evaluation of each individual in the evolutionary algorithm, which requires training of many deep neural networks. Recent research has shown that partial training is sufficient for estimating the quality of CNN architecture [8, 9]. In this work, we adopted this strategy and trained the CNN architectures for a few epochs in the evolution phase. Later, the best evolved CNN model was trained completely for evaluating its performance. The proposed GA was applied to three well-known datasets, and the evolved CNN models were compared with many existing models designed by human and automatically.

The rest of the chapter is organized as follows. A brief overview of CNN is presented in Sect. 8.2. Section 8.3 reviews the related work. The proposed GA is described in Sect. 8.4. Section 8.5 details the experimental setup and the experimental results are presented in Sect. 8.6. Section 8.7 contains a brief discussion on the results and Sect. 8.8 concludes the chapter.

8.2 A Brief Overview of CNNs

Convolutional neural networks (CNNs) that were inspired by the organization of the animal cortex [10] are mostly used for two-dimensional data like images. CNNs consist of three major types of network layers, namely: convolutional, pooling, and fully connected. The learning in a convolutional layer depends on three concepts: sparse interaction, equivariant representation, and parameter sharing [11]. Unlike the feed-forward neural network layers that utilize layer-wide matrix multiplication for relating the inputs with the outputs, convolutional layers implement sparse interactions by using filters smaller than the inputs. By sharing the same filter across the input surface, convolutional layers can achieve spatial equivariance as well as reduce the computational volume considerably. Using multiple learnable filters the convolutional layer can learn different features from the input. Pooling layers are usually placed after one or more convolutional layers to reduce the dimensionality of the data. Multiple blocks of convolutional and pooling layers are used to extract hierarchical features from the data. Also, depending on the nature of the problem, the final convolutional or pooling layer is followed by one or more fully connected or recurrent layers.

There are many hyperparameters involved in designing a CNN architecture, such as the number of layers in a convolutional block, the kernel size, stride size, and channel size (number of feature maps) of a convolution layer, the stride size of the pooling layer, type of pooling operation, number of fully connected

layers, the number of nodes in a fully connected layer, etc. In human-designed CNNs, these hyperparameters are selected on a trial and error basis with the help of prior knowledge about the functionality of these layers. As discussed in Sect. 8.3, meta-heuristic algorithms can help to automatically select an optimal set of hyperparameters for a CNN in a given task.

8.3 Related Works

In recent years, we have seen a increasing interest in the evolutionary computation community in evolving DNN architectures and their hyperparameters. David and Greental [12] proposed a simple GA-assisted method to improve the performance of the deep autoencoder on the MNIST dataset. They stored the sets of weights of an autoencoder in the chromosomes of individuals in their GA population. Then, by calculating the root mean square error (RMSE) of each chromosome for the training sample, they set the fitness score of each individual as the inverse of RMSE. After sorting all chromosomes from the fittest to the least fit, they tuned the weights of the high ranking chromosomes using backpropagation and replaced the low-ranking members with the offspring of high ranking ones. However, they just used the fitness score as a criterion for removing the low ranking members, and selection is implemented uniformly and applied to the outstanding chromosomes with equal likelihood. The authors showed that compared to the traditional backpropagation, the GA-assisted method gives better reconstruction error and network sparsity.

Suganuma et al. [8] used Cartesian Genetic Programming (CGP) to construct the CNN structure and network connectivities. To reduce the search space, high-level functional modules, such as convolutional block and tensor concatenation, were used as the node functions of CGP. Following the training of the network using the training data, they utilized the validation accuracy as the fitness score. By evaluating the performance of the evolved CNN models on the CIFAR10 dataset, they achieved the error rates of 6.34 and 6.05% for the CGP-CNN (ConvSet) and the CGP-CNN (ResSet), respectively. Loshchilov and Hutter [13] introduced the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) as an optimization method for selecting the hyperparameters of the deep neural networks (DNNs). In their experiments, the performance of CMA-ES and other state-of-the-art algorithms were evaluated for tuning 19 hyperparameters of a DNN on the MNIST dataset. They pointed out that the CMA-ES algorithm shows competitive performance, especially in parallel evaluations.

In another study, Sun et al. [14] used a GA to design the CNN architectures automatically for image classification. They used skip layers, composed of two convolutional layers and one skip connection borrowed from ResNet [6], to increase the depth of the network. Moreover, they used the same filter size and stride for all convolutional layers, and the number of feature maps was selected by their method. The fully connected layers were omitted in their model, but the pooling layers were used. They evaluated the performance of their model on several popular

benchmarks, such as CIFAR10 and CIFAR100. The other work conducted by Sun et al. [15] utilized the ResNet and DenseNet blocks [16] for automatically evolving the CNN architectures. In their approach, a combination of three different units, ResNet block units, DenseNet block units, and pooling layer units, have been used to generate the CNN architecture. In their encoding strategy, to increase the depth of the network as well as the speed of heuristic search by changing the depth of the network, each ResNet or DenseNet unit composed of multiple ResNet and DenseNet blocks. They showed the superiority of their model by comparing their model generated results with 18 state-of-the-art algorithms on CIFAR10 and CIFAR100 datasets. Ali et al. [17] proposed a GA model to evolve a CNN architecture and other network hyperparameters. They used a generic GA to find the best combination of network hyperparameters, such as the number of layers, learning rate, and weight decay factor. Using some design rules and constraints for genotype to phenotype mapping, they evolved a CNN architecture on CIFAR10 dataset. They compared the performance of the best CNN architecture evolved by their method with 13 other models in terms of classification accuracy and GPU days. Sun et al. [9] introduced a GA model for evolving CNN architecture as well as connection weight initialization values in image classification tasks. They used an efficient variable-length gene encoding method, representing various building blocks, to find the optimal depth of the CNN. Furthermore, to avoid trapping in local minima, a major problem in gradient-based training, they introduced a new representation scheme for initializing the connection weights of DNN. They showed the effectiveness of their proposed method by comparing their results with 22 existing algorithms involving state-of-the-art models on nine popular image classification tasks.

8.4 The Proposed Genetic Algorithm for Designing CNNs

In this work, using a GA, we evolved CNN architectures with the best combination of hyperparameters for the image classification task. Our GA operates in the search space of VGG-like architectures, i.e., we assumed that the CNN architecture consists of a sequence of convolutional blocks, each followed by a pooling layer, and a fully connected layer at the end. The GA is used to optimize the number of convolutional blocks, the number of convolutional layers in each block as well as some other hyperparameters of the CNN architecture. The assumption about the organization of the CNN architecture confines the GA to discover only the classical CNN models, i.e., it does not allow to design CNN architectures with more sophisticated modules like residual blocks [6] or inception modules [7]. However, by exploring the limited search space, the proposed GA was able to design CNNs, which exhibited competitive performance with the other state-of-the-art CNN models.

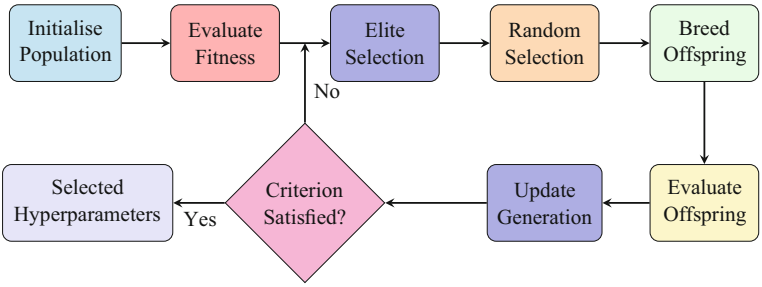

Fig. 8.1 The flowchart of the genetic algorithm for evolving CNN model

The proposed algorithm for optimizing CNN architecture works in the generic GA framework with standard GA operations (Fig. 8.1). The algorithm begins with an initial population created by the random selection of genes for each individual. The chromosome of each individual uniquely determines the architecture of a CNN as well as some of its hyperparameters. Then, each CNN model is trained and validated using the training dataset, and the average classification accuracy of the network in the validation phase is used as the individual's fitness score. Next, the population of individuals is sorted in descending order of their fitness scores. Then, by applying a chain of genetic operations such as elite selection, random selection, and breeding of the new members, the next generation of the population is created. This process is repeated until the satisfaction of the termination criterion. The pseudocode of the genetic algorithm for CNN architecture optimization is shown in Algorithm 1, and the details of each module are presented in the following subsections.

Algorithm 1: Proposed GA framework for evolving CNN models

Input: Population size (N_P), Maximum number of generation (G_{max}), the range of values for the selected hyperparameters (L_H), the RGB images of training dataset
Output: The best CNN architecture with its hyperparameters

```
Initialize the population using a random combination of hyperparameters [Algorithm 2]
Train the CNN model designated by each individual in the population, and calculate the corresponding fitness score [Algorithm 3]
Store the population of individuals and their fitness scores in a list called P
N_G ← 0
while N_G < G_max do
    Create a new generation P_new consisting of elite individuals, random individuals and offspring created from P [Algorithm 4]
    Evaluate individuals in P_new
    Set P ← P_new and N_G ← N_G + 1
end
Return the best CNN architecture in P along with its hyperparameters
```

LR	WD	M	DR	B1	B2	B3	B4	B5	F1	F2	F3	F4	F5
0.01	0.001	0.8	0.25	2	0	3	2	4	256	128	32	512	256

Fig. 8.2 The chromosome of an individual showing different hyperparameters of the CNN model

8.4.1 Population Initialization

As mentioned earlier, the proposed GA is used to seek the optimal CNN model in the search space of classical CNN architectures. We limited the maximum number of convolutional layers in a CNN to 20 divided into a maximum of 5 blocks $B1$, $B2$, $B3$, $B4$, $B5$. Each of the blocks can have any number of layers between 0 and 4. The number of feature maps for each block is also optimized by GA, which can be chosen from {32, 64, 128, 256, 512}. The other hyperparameters, optimized by our GA, are learning rate (LR), weight decay (WD), momentum (M), and dropout rate (DR). The structure of the chromosome is shown in Fig. 8.2.

From the chromosome structure, it becomes clear that the proposed GA works with a fixed size chromosome. However, by allowing a block size to be zero, the GA can actually search for variable-length CNN models having variable number of blocks with any number of layers between 0 and 20. The example in Fig. 8.2 shows a CNN architecture with 11 layers where the first block consists of 2 convolutional layers with a feature map size of 256. The second block does not exist, the third, fourth, and fifth block have 3, 2, and 4 layers and their corresponding feature map sizes are 32, 512, and 256, respectively.

In genotype to phenotype mapping, a couple of additional layers are added in each CNN model. Each convolutional block is followed by a max-pooling layer with kernel size 2 and stride size 2, an average-pooling layer with a kernel size 2 and a stride size 1 is added after the final max-pooling layer, and a linear fully connected layer is placed at the end of the network. Moreover, each convolutional layer is followed by a batch normalization layer [18] and a Rectified Linear Unit (ReLU) layer [19], and a dropout layer is added at the rear of each convolutional block.

Algorithm 2 summarizes the process of population initialization for the GA. Each gene in a chromosome can take a range of values, and the proposed GA searches for the optimal combination of these values through the evolution process. The range of possible values for each gene (shown in Table 8.1) is selected according to the previous experiences in different classification problems using CNNs. Following the random selection of the hyperparameters, the CNN architecture is created without any constraints on the number or the order of convolutional layers or feature maps. Often, human-designed CNN architectures are created following some rules, e.g., an increasing number of feature maps are used in successive convolutional blocks. However, in the proposed GA model, no such restriction was imposed, and the architecture of a CNN is completely guided by the chromosome.

Algorithm 2: For generating initial population

Input: The population size N_P
Output: The initialized population P_{init}
Data: The ranges of values for different hyperparameter are stored in a list called L_H

$P_{init} \leftarrow \emptyset$
while $|P_{init}| < N_P$ **do**
  Select the learning rate (lr) randomly from the L_H [LR]
  Select the weight decay factor (wd) randomly from the L_H [WD]
  Select the momentum (m) randomly from the L_H [M]
  Select the dropout (d) randomly from the L_H [DR]
  Select the number of convolutional layer in each block $\{B1, B2, B3, B4, B5\}$ randomly from the L_H [NL]
  Select the number of feature maps corresponding to each block $\{F1, F2, F3, F4, F5\}$ randomly from the L_H [NF]
  Create an individual (Ind) with the selected hyperparameters
  $P_{init} \leftarrow P_{init} \cup Ind$
Return P_{init}

Table 8.1 The range of possible values for different hyperparameters to be searched by the GA

Hyperparameter	Values
Learning rate (LR)	0.1, 0.01, 0.001, 0.0001
Weight decay (WD)	0.1, 0.01, 0.001, 0.0001, 0.00001
Momentum (M)	0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9
Dropout rate (DR)	0.25, 0.5, 0.75
Block size (NL) in B1-B5	0, 1, 2, 3, 4
Feature map size (NF) in F1-F5	32, 64, 128, 256, 512

Finally, it should be noted that the chromosome structure and the algorithm are flexible enough to make it more general by considering the optimization of additional hyperparameters, e.g., activation function, individual kernel size for each convolutional block, etc. Nevertheless, increasing the search space size will necessitate more extensive searching. We did some preliminary study with some other variants of the chromosome, but later fixed those hyperparameter values (e.g., fixed the kernel size of convolutional layers to 3) to reduce the computational burden.

8.4.2 Fitness Evaluation

In order to assess the quality of a CNN model constructed from the chromosome of an individual, we need to train it and evaluate its classification performance. Training and evaluating a deep neural network is the computationally most expensive part of any deep neuroevolution algorithm. Recent studies have suggested that it is possible to roughly assess the architectural quality of a CNN model based on its

evaluation after partial training [8, 9]. Henceforth, during the evolution process, we evaluated the performance of the CNN networks after partially training them for only $N_{epoch} = 10$ epoch, which significantly accelerated the genetic algorithm.

Algorithm 3: For fitness evaluation of an individual

Input: The individual (Ind), training data (D_{train}), validation data (D_{valid}), the number of epoch in training phase (N_{epoch})
Output: The fitness score of the individual

Create the CNN model (net) from the hyperparameters of Ind augmented with pooling, fully connected, batch-normalization, ReLU and dropout layers (details in Sect. 8.4.1)
$Acc \leftarrow \emptyset$
$step \leftarrow 0$
$Acc_{avg} \leftarrow 0$
while $step < N_{epoch}$ **do**
    Train the model net using the D_{train}
    Calculate the classification accuracy (acc) using the D_{valid}
    $Acc \leftarrow Acc \cup acc$
    $step \leftarrow step + 1$
end
$Acc_{avg} \leftarrow$ Average of accuracies in Acc
Return Acc_{avg}

We used the average validation accuracy of the constructed CNN model as the fitness score of the corresponding individual. 90% of the training data is used during the training phase, and the rest 10% is utilized for validation. The constructed CNN model is trained by the stochastic gradient descent (SGD) algorithm [20], for a constant number of epochs ($N_{epoch} = 10$), and the average classification accuracy of the validation phase is used as the fitness score of the individual. In all experiments, during the training phase, the cross-entropy loss is used as the loss function, and the learning rate is reduced by a factor of 10 in every 10 epochs during complete training for the model after the evolutionary phase. The details of the fitness evaluation of an individual are summarized in Algorithm 3.

8.4.3 Creating New Generation

In the proposed GA, the next generation of individuals is created from the current generation using elite selection, random selection, and offspring generation. First, the individuals in the current generation are sorted based on their fitness scores. Top $e\%$ of the individuals, known as elites, are selected from the current population and added to the next generation. To maintain the population diversity and to prevent premature convergence, some random individuals are also added [21, 22]. Specifically, from the rest of the current population, individuals are randomly selected with a probability of p_r and added to the next generation. Finally, the selected elite and random individuals form the parent pool to breed offspring.

Algorithm 4: For creating a new generation of individuals

Input: The current population of individuals with their fitness scores (P), the percentage of population preserved as the elite (e), the probability of preserving an individual from the non-elite section of the current population (p_r), the probability of mutation (p_m), and the population size (N_p)

Output: The new population (P_{new})

```
1  P_new ← ∅
2  Sort the individuals in P in descending order of their fitness scores
3  Add top e% individuals from P to the new population P_new
4  Select the individuals from the bottom (1 − e)% of P with probability p_r and add them to
   P_new
5  P_parents ← P_new
6  while |P_new| < N_p do
7      Par_1 ← An individual randomly selected from P_parents
8      Par_2 ← An individual randomly selected from P_parents
9      if Par_1 ≠ Par_2 then
10         Create two children from the selected parents using uniform crossover operation
           and save them in Children
11         for each Child in Children do
12             r ← Randomly generate a number from the range (0,1)
13             if p_m > r then
14                 Randomly replace a gene in Child with the randomly selected value
15             end
16         end
17         P_new ← P_new ∪ Children
18     end
19 end
20 Return P_new
```

The process of generating the offspring starts with the random selection of two dissimilar individuals from the parent pool. The selected parents participate in uniform crossover operation to create two offspring. Each child may inherit various combinations of genes from the parents because its genes are randomly selected from the parents. Then, the child undergoes the mutation operation with a predefined mutation probability of p_m. If the mutation condition is met, one randomly selected gene in the offspring chromosome is randomly modified from a set of predefined values (shown in Table 8.1). Newly created offspring are added to the next generation. The process of creating new children is repeated until the number of individuals in the new generation reaches the population size. Finally, the new generation, consisting of parent pool and children pool, replaces the current generation. The process of generation alternation is repeated $G_{\max}$ times so that the GA can search for the best network architecture and hyperparameters. The process of creating a new generation from the current generation is shown in Algorithm 4.

8.5 Experimental Setup

8.5.1 Datasets

In this work, we used three popular datasets CIFAR10, CIFAR100, and SVHN as the benchmark for image classification problems. These datasets have been used in many pieces of research for evaluation of the state-of-the-art deep neural network models which makes our evolved models comparable with those models. The CIFAR10 dataset includes 60,000 color RGB images belonging to 10 classes and is mostly used for image classification tasks. These images are of dimension 32×32 and are divided into training and testing parts. The training set contains 50,000 images, and the rest of 10,000 images are used as the testing set. There is an equal number of training and testing samples for each class.

The CIFAR100 dataset is similar to the CIFAR10, except with 100 classes that are categorized into 20 superclasses each of which contains five classes. There exist only 500 training images and 100 testing images, per class, making classification more challenging in this dataset.

The SVHN (Street View House Numbers) dataset that can be considered in essence similar to the MNIST dataset but with more labeled data contains 73257 and 26032 digits for training and testing, respectively. Compared to the MNIST dataset, the SVHN dataset originates from a more difficult real-world problem. This dataset contains the original, colored, and variable resolution images from the house numbers in Google Street View images. However, all digits of the house numbers have been resized to a fixed resolution 32×32 and originate from 10 different classes [23].

8.5.2 Experimental Environment

In this work, we used the Pytorch framework (Version 1.2.0) of the python programming language (Version 3.7) in all experiments. Besides, we ran the codes using both high-performance computing (HPC) services and the DGX station machine at the University of Newcastle. All codes ran with two GPUs at all stages including the evolution of various CNN architectures, complete training of the selected models with a larger epoch, and testing the best-performing models.

8.5.3 Parameter Selection

As stated before, our proposed framework is very flexible for increasing the search space. In other words, many hyperparameters can be evolved through the GA framework, but because of the limitation imposed by computational resources, we

just evolved some of the selected hyperparameters. Also, according to our earlier experimental results, a specific value is always selected by the GA for some of these hyperparameters, namely activation function, optimizer. Hence, in all experiments, the ReLU activation function and SGD optimizer have been used. Besides, the kernel size of the convolutional layers was set to 3 and the stride size of the convolutional layer and the max-pooling layer was set to 2.

The GA parameters were set as follows: maximum number of generation $G_{\max} = 40$, population size $N_P = 30$, the percentage of population retained as the elite $e = 40\%$, the probability of retaining an individual from the non-elite part of the population $p_r = 0.1$, and the probability of the mutation $p_m = 0.2$. These parameters were set based on our experience with evolutionary algorithms and using some primary studies. Moreover, to decrease the computational burden and speed up the evolution process, during evolution, we trained the networks with a smaller number of the epochs $N_{epoch} = 10$. After the evolutionary phase, the best CNN model is trained completely before it is evaluated with the test dataset. Precisely, the best model evolved by the GA is trained for a higher number of epoch $N_{epoch} = 350$ using the full training set.

8.6 Experimental Results

In our experiments, we applied the proposed GA for evolving the CNN architectures for each dataset. Considering the stochastic nature of the algorithm, we repeated each experiment 5 times and the best CNN model evolved in each experiment is later trained completely and tested on the corresponding dataset. Table 8.2 shows the performance of the evolved models in different datasets in terms of their average accuracy, standard deviation, best accuracy, and the worst accuracy. The CNN models designed by the GA was able to achieve a good performance in all three datasets. Considering the average, best, and worst accuracies as well as the standard deviations shown in Table 8.2, it is evident that the evolutionary algorithm was pretty reliable in finding CNN models of similar quality over multiple experimental runs.

The average convergence graph of GA from a single representative run is shown in Figs. 8.3 and 8.4 for all three datasets. Note that the fitness score in these graphs is the average validation accuracy of the CNN models in each generation which were

Table 8.2 The average accuracy, standard deviation, best and worst accuracy of the best CNN models evolved by multiple GA runs in each dataset

Dataset	Average accuracy	STD	Best accuracy	Worst accuracy
CIFAR10	94.75	0.650	95.82	94.01
CIFAR100	75.90	0.521	76.79	75.34
SVHN	95.11	0.48	95.57	94.52

Each network was trained for 350 epochs

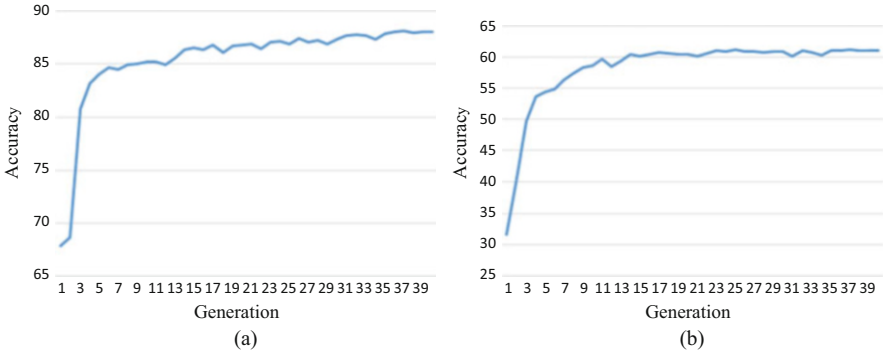

Fig. 8.3 The average convergence graph of the GA population in one representative run on (**a**) CIFAR10 dataset and (**b**) CIFAR100 dataset

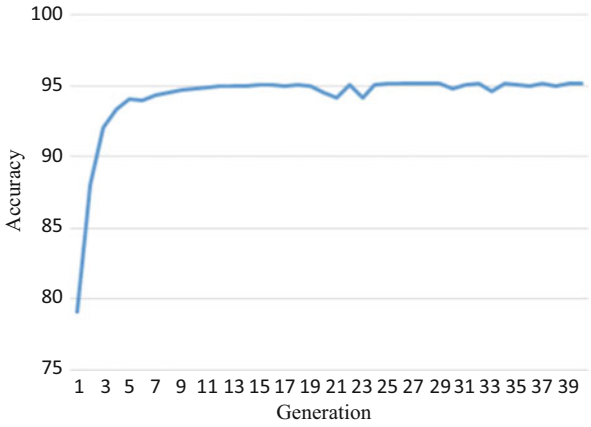

Fig. 8.4 The average convergence graph of the GA population in one representative run on SVHN dataset

trained only for 10 epochs. As shown in these graphs, the proposed algorithm was successful to improve the overall fitness of the population on all datasets. The fitness of the population improved quickly in the first 10 generations and then slowed down gradually. However, this behavior is expected because of the high selection pressure from the elitism strategy and the participation of the elite individuals in offspring generation.

Table 8.3 shows the structures of the best networks evolved in different GA runs for each dataset. Each row of Table 8.3 shows the CNN architecture in terms of the number of blocks, number of convolutional layers in each block, and the feature map size of each block. The convolutional blocks are separated by comma and in each block we show the number of layers and the feature map size (in parenthesis) for each layer in that block. For example in the first row of Table 8.3, $4 \times (128)$ represents that the first convolutional block of the CNN consists

Table 8.3 The best structures evolved in different GA runs for different datasets

Dataset	Network ID	No. parameters	Evolved architecture
CIFAR10	Net 1	14.3 M	[4 × (128), 4 × (256), 2 × (256), 3 × (512), 2 × (512)]
CIFAR10	Net 2	25.4 M	[4 × (512), 3 × (512), 4 × (512), 2 × (256)]
CIFAR10	Net 3	5.8 M	[2 × (512), 4 × (256), 2 × (128)]
CIFAR10	Net 4	6.7 M	[2 × (256), 3 × (512), 2 × (32)]
CIFAR10	Net 5	20.7 M	[3 × (512), 2 × (512), 4 × (512), 2 × (256)]
CIFAR100	Net 1	1.7 M	[4 × (256), 2 × (512), 4 × (256)]
CIFAR100	Net 2	14.2 M	[3 × (512), 4 × (512)]
CIFAR100	Net 3	11.2 M	[2 × (256), 4 × (512), 4 × (512)]
CIFAR100	Net 4	18.3 M	[3 × (512), 3 × (512), 4 × (256)]
CIFAR100	Net 5	14.8 M	[2 × (256), 2 × (512), 3 × (512)]
SVHN	Net 1	19 M	[3 × (512), 4 × (512), 4 × (256)]
SVHN	Net 2	7.8 M	[3 × (32), 3 × (512), 4 × (512), 2 × (512)]
SVHN	Net 3	17.1 M	[3 × (32), 3 × (512), 4 × (512), 4 × (512)]
SVHN	Net 4	23.8 M	[3 × (256), 3 × (256), 2 × (256), 4 × (512)]
SVHN	Net 5	11.9 M	[3 × (32), 3 × (512), 4 × (512), 2 × (512)]

of 4 convolutional layer each having a feature map size of 128. The complete CNN model is constructed by adding ReLU, batch normalization, dropout, average pooling layer and fully connected layers as described in Sect. 8.4.1. This table also shows the number of trainable parameters for different evolved models. The other hyperparameters of each evolved network for CIFAR10, CIFAR100, and SVHN dataset are shown in Tables 8.4, 8.5, and 8.6, respectively. Figure 8.5 visualizes the architecture of the best CNN models evolved by GA over five repeated runs in three datasets.

It can be seen from Table 8.3 that different evolutionary runs evolved quite different CNN architectures in terms of the number of blocks, number of layers, and number of trainable parameters even for the same dataset. However, with these different architecture the models achieved very similar accuracy in the respective datasets as shown in Tables 8.4, 8.5, and 8.6. One interesting observation in evolved architectures by the GA is the feature map sizes in different blocks. In human-designed architectures, usually the feature map size increases in later convolutional blocks as we have seen in case of different VGG models. In some of the evolved models we notice the same characteristics, e.g. Net 1 for CIFAR10, however, in general, this order was not maintained in the evolved models. Some architecture has it in decreasing order (Net 3 for CIFAR10) and some has it in no specific order (Net 4 for CIFAR10). From this observation we can infer that use of increased feature map size in later convolutional block is not absolutely necessary for good CNN architecture design.

In order to further assess the merit of the evolved models, we trained each evolved model with the other datasets and tested its accuracy. Specifically, Table 8.4 shows the best evolved models for the CIFAR10 datasets and then applied in CIFAR10,

Table 8.4 Hyperparameters of the top five CNN models evolved by different GA runs for CIFAR10 dataset

Hyperparameters							CIFAR10	CIFAR100	SVHN
Net name	LR	WD	M	DR	No. blocks	No. layers	accuracy	accuracy	accuracy
Net 1	0.01	0.01	0.65	0.5	5	15	95.82	79.48	96.93
Net 2	0.01	0.001	0.8	0.5	4	13	95.09	76.95	96.60
Net 3	0.1	0.001	0.7	0.25	3	8	94.64	74.41	95.64
Net 4	0.01	0.01	0.7	0.5	3	7	94.21	71.53	92.80
Net 5	0.01	0.0001	0.8	0.5	4	11	94.01	76.07	96.27

Table 8.5 Hyperparameters of the top five CNN models evolved by different GA runs for CIFAR100 dataset

Hyperparameters							CIFAR10	CIFAR100	SVHN
Net name	LR	WD	M	DR	No. blocks	No. layers	accuracy	accuracy	accuracy
Net 1	0.01	0.001	0.8	0.5	3	10	93.28	76.79	96.03
Net 2	0.1	0.0001	0.75	0.5	2	7	94.14	76.04	95.87
Net 3	0.01	0.0001	0.8	0.5	3	10	94.66	75.59	96.54
Net 4	0.01	0.001	0.8	0.25	3	10	94.56	75.52	96.35
Net 5	0.01	0.0001	0.9	0.5	3	7	94.34	75.34	96.03

Table 8.6 Hyperparameters of the top five CNN models evolved by different GA runs for SVHN dataset

Hyperparameters							CIFAR10	CIFAR100	SVHN
Net name	LR	WD	M	DR	No. blocks	No. layers	accuracy	accuracy	accuracy
Net 1	0.01	0.0001	0.9	0.25	3	11	95.10	73.95	95.57
Net 2	0.01	0.0001	0.7	0.5	4	12	93.21	73.53	95.51
Net 3	0.01	0.0001	0.7	0.5	4	14	92.58	72.89	95.43
Net 4	0.01	0.001	0.7	0.5	4	12	94.19	73.18	94.53
Net 5	0.01	0.0001	0.8	0.5	4	12	94.05	73.10	94.52

CIFAR100 and SVHN datasets. Similarly, Tables 8.5 and 8.6 show the performance of the evolved models for the CIFAR100 and SVHN datasets, respectively, in all three datasets.

It was expected that the model evolved for a particular dataset will exhibit the best performance in that dataset. However, we notice that in general the best performance was exhibited by the models evolved for CIFAR10 dataset (Table 8.4). We hypothesize that for more complex datasets like CIFAR100 and SVHN, the training of 10 epochs is not sufficient to assess the quality of the network architecture, therefore, the evolved model did not perform the best in the respective datasets. One particular point to note in Tables 8.4, 8.5, and 8.6 is that evolutionary runs selected network hyperparameters pretty consistently. For example almost in every model, dropout rate was chosen as 0.5 and learning rate was chosen as 0.01.

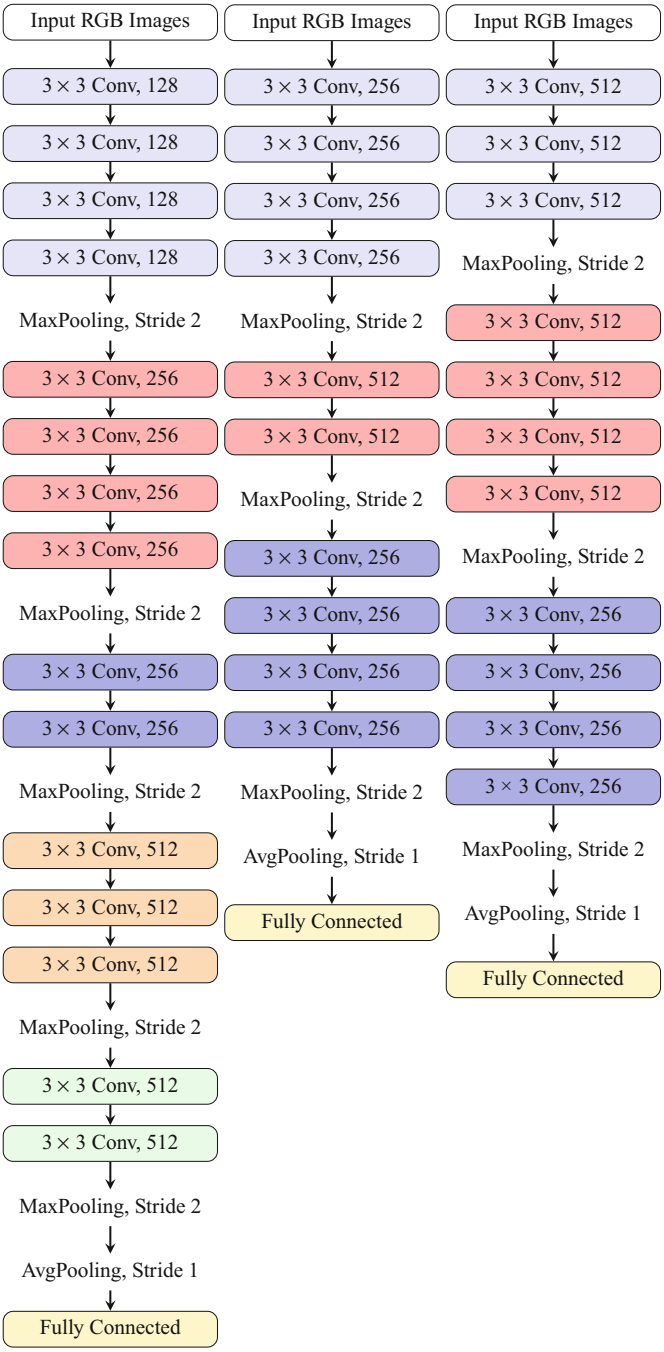

Fig. 8.5 Architectures of top CNN models evolved by GA (left to right for: CIFAR10, CIFAR100, and SVHN datasets)

Table 8.7 The comparisons between the GA-evolved CNN model and the state-of-the-art CNN algorithms in terms of the classification accuracy (%)

Algorithm name	Accuracy CIFAR10	Accuracy CIFAR100	GPU days	Parameter setting
VGG16	93.05	74.94	–	Manually
VGG19	92.59	74.04	–	Manually
ResNet101	94.08	75.39	–	Manually
DenseNet	94.52	76.61	–	Manually
Maxout[a]	90.70	61.40	–	Manually
Genetic CNN[a]	92.90	70.97	17	Semi-auto
Hierarchical evol.[a]	**96.37**	–	300	Semi-auto
Block-QNN-S[a]	95.62	79.35	90	Semi-auto
Large-scale evol.[a]	94.60	77	2750	Automatic
CGP-CNN[a]	94.02	–	27	Automatic
NAS[a]	93.99	–	22,400	Automatic
Meta-QNN[a]	93.08	72.86	100	Automatic
CNN-GA[a]	95.22	77.97	35	Automatic
Fast-CNN [17]	94.70	75.63	14	Automatic
This work (CIFAR10)	95.82	**79.48**	6	Automatic

[a]The values of this algorithm reported in [14]

The purpose of partial training (with a smaller number of epochs) of different models during evolutionary phase was to reduce the computation burden. Some other work [8, 9] and our previous experiments [17] showed that such partial training can be sufficient for assessing network architecture in image classification. In this work we aimed to investigate it further by training the network for only 10 epochs. No doubt it accelerates the evolution process greatly—the average time for evolving the networks was only 6 GPU days. However, based on the performance of the networks evolved for CIFAR100 and SVHN datasets, we infer that such minimum training will be useful for simpler dataset but will not be sufficient for assessing the network's quality in complex datasets.

Finally, to assess the competitiveness of the GA-evolved models with the other state-of-the-art CNN models, we compared their performance in Table 8.7 (best performances are shown in bold). We contrasted the performance of three categories of networks, manually designed, designed semi-automatically, and designed completely automatically [14]. Among the manually designed networks are VGG16 [5], VGG19 [5], ResNet101 [6], DenseNet [16], and Maxout [24]. Among the semi-automatically designed networks are Genetic CNN, Hierarchical Evolution, Block-QNN-S and from the fully automatically designed networks are Large-scale Evolution, CGP-CNN, NAS, Meta-QNN, CNN-GA, and Fast-CNN [17]. We compared the performance of the best model evolved by the proposed GA for CIFAR10 dataset on both CIFAR10 and CIFAR100 datasets (last row). Besides comparing these models in terms of their accuracy in CIFAR10 and CIFAR100 datasets, for the automatically designed networks, we compared them in terms of

GPU days required to design those networks. The GPU days are a rough estimation for determining the speed of the algorithm, but it is not applicable to the manually designed models. It should be noted that some of the results have been reproduced by us, while others (indicated in the table footnote) were just copied from [14].

From Table 8.7 it can be seen that the CNN model evolved by our proposed GA was obviously better than VGG models as well as other human designed CNN models in CIFAR10 dataset. The CNN model was also better than all other models designed automatically. Its performance was second best among the compared models in CIFAR10 dataset. The best performance was exhibited by the model designed by hierarchical evolution which is a semi-automatically designed network. In terms of performance on CIFA100 dataset, the evolved model in this work exhibited the best accuracy compared to all other CNN models whether designed manually, semi-automatically, or automatically. Finally, when compared in terms of required computational power, the proposed GA was really fast, requiring only 6 GPU days, in finding the optimal CNN architecture compared to other semi-automatic and automatic methods. Although the Hierarchical Evolutionary model in the semi-automatic category shows better classification accuracy on CIFAR10, its GPU days is 50 times bigger than that required by the proposed method in this work.

8.7 Discussion

This chapter basically investigates if a genetic algorithm can help to find an optimized VGG-like CNN model for image classification. Using a fixed length chromosome, the proposed GA explored the search space of CNN architectures consisting of variable number of layers divided into a variable number of blocks. Experiments with three widely used datasets show that the proposed GA is able to design CNN models optimizing both its structure and hyperparameters. The evolved models were better than the classic VGG models and several other human-designed CNN models. Despite having a VGG-like architecture, the GA designed models were also very competitive with other state-of-the-art CNN models designed by semi-automatic and automatic methods. Additionally, the GA designed CNN models sometimes had structural characteristics different from those designed by humans. We also evaluated the performance of the CNN model, evolved by GA for one dataset, on other datasets. The high-quality performance of the models on other datasets indicates the superiority of the architectures evolved by GA. Based on these results, we conclude that the proposed GA is capable of optimizing classic CNN models for higher performance.

For the answer to our second question, if we can reduce the high cost of evolving CNN architecture by using partial training of the models, we evaluated the performance of networks after a few epochs of training. From our experiments with the CIFAR10 dataset, we found that partial training of the models with only a few epochs was good for finding very good architectures. However, the CNN models

evolved for CIFAR10 dataset exhibited an overall better performance in CIFAR100 and SVHN datasets than those evolved for these two datasets. Based on these results, we hypothesize that perhaps partial training was not effective in evaluating network's performance in complex datasets. Nevertheless, a more detailed study by varying the training epochs on multiple datasets is required for a more general and accurate conclusion.

In our experiments, some of the structural parameters and hyperparameters of CNN were kept fixed. Although the presented framework is ready to be extended for the evolution of those parameters, the expansion of the search space will necessitate more computational power to find the optimal CNN model. On the other hand, at that expense, it might be possible to find a more efficient CNN model. Besides, the size of the evolved networks is big compared to many other architectures because no measure was taken to restrict the network size. The current framework can also be extended to incorporate that criterion either in a single or multi-objective setup. Optimizing a larger set of parameters may also help in finding smaller network models.

8.8 Conclusion and Future Work

In this chapter, we showed how a simple genetic algorithm (GA) can be used to automatically discover the optimized CNN model. Exploring the search space of the classic CNN models, the proposed GA optimized the number of convolutional blocks, number of convolutional layers in each blocks, the size of feature maps for each block and other training related hyperparameters such as dropout rate, learning rate, weight decay, and momentum. To reduce the computational burden in model training, which is a common challenge for all deep neuroevolution algorithms, we trained the models partially during evolution. The proposed GA, when evaluated in three popular datasets, CIFAR10, CIFAR100 and SVHN, designed very high quality CNN models over multiple repeated experiments. Performance of the evolved CNN model is compared with 14 state-of-the-art models, chosen from different categories, in terms of classification accuracy and GPU days. The best CNN model evolved on the CIFAR10 dataset was found very competitive with other human designed and automatically designed CNN models in terms of the classification accuracy and better in terms of GPU days to evolve them.

The proposed GA framework can be used for searching more structure and training related hyperparameters of CNN, e.g. kernel size, stride size, activation function, optimizer choice, etc., with very minimum changes. Besides, the framework can be extended for finding a smaller network model in terms of parameter numbers. Additionally, the performance of proposed GA can be tested in other applications of CNN as well as for optimizing other types of deep neural networks.

References

1. Darwish, A., Hassanien, A.E., Das, S.: A survey of swarm and evolutionary computing approaches for deep learning. Artif. Intell. Rev., 1–46 (2019)
2. Iba, H., Noman, N.: New Frontier in Evolutionary Algorithms: Theory and applications. Imperial College Press, London (2011)
3. Schaffer, J.D., Whitley, D., Eshelman, L.J.: Combinations of genetic algorithms and neural networks: a survey of the state of the art. In: International Workshop on Combinations of Genetic Algorithms and Neural Networks, 1992. OGANN-92, pp. 1–37. IEEE, Piscataway (1992)
4. Such, F.P., Madhavan, V., Conti, E., Lehman, J., Stanley, K.O., Clune, J.: Deep Neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. Preprint. arXiv:1712.06567 (2017)
5. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. Preprint. arXiv:1409.1556 (2014)
6. 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, pp. 770–778 (2016)
7. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015)
8. 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, pp. 497–504. ACM, New York (2017)
9. Sun, Y., Xue, B., Zhang, M., Yen, G.G.: Evolving deep convolutional neural networks for image classification. IEEE Trans. Evol. Comput. **24**(2), 394–407 (2020)
10. Matsugu, M., Mori, K., Mitari, Y., Kaneda, Y.: Subject independent facial expression recognition with robust face detection using a convolutional neural network. Neural Netw. **16**(5–6), 555–559 (2003)
11. Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y.: Deep Learning, vol. 1. MIT Press, Cambridge (2016)
12. David, O.E., Greental, I.: Genetic algorithms for evolving deep neural networks. In: Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 1451–1452. ACM, New York (2014)
13. Loshchilov, I., Hutter, F.: CMA-ES for hyperparameter optimization of deep neural networks. Preprint. arXiv:1604.07269 (2016)
14. Sun, Y., Xue, B., Zhang, M., Yen, G.G.: Automatically designing CNN architectures using genetic algorithm for image classification. Preprint. arXiv:1808.03818 (2018)
15. Sun, Y., Xue, B., Zhang, M.: Automatically evolving CNN architectures based on blocks. Preprint. arXiv:1810.11875 (2018)
16. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2261–2269. IEEE, Piscataway (2017)
17. Bakhshi, A., Noman, N., Chen, Z., Zamani, M., Chalup, S.: Fast automatic optimisation of CNN architectures for image classification using genetic algorithm. In: 2019 IEEE Congress on Evolutionary Computation (CEC), pp. 1283–1290. IEEE, Piscataway (2019)
18. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. Preprint. arXiv:1502.03167 (2015)
19. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 315–323 (2011)
20. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: Proceedings of COMPSTAT'2010, pp. 177–186. Springer, Berlin (2010)

21. Anderson-Cook, C.M.: Practical genetic algorithms. J. Am. Stat. Assoc. **100**(471), 1099–1099 (2005)
22. Malik, S., Wadhwa, S.: Preventing premature convergence in genetic algorithm using DGCA and elitist technique. Int. J. Adv. Res. Comput. Sci. Softw. Eng. **4**(6) (2014)
23. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning. In: NIPS Workshop on Deep Learning and Unsupervised Feature Learning (2011)
24. Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout networks. Preprint. arXiv:1302.4389 (2013)