Fast Automatic Optimisation of CNN Architectures for Image Classification Using Genetic Algorithm

Ali Bakhshi
School of Elect Engg & Computing
The University of Newcastle
Newcastle, Australia
ali.bakhshi@uon.edu.au

Nasimul Noman
School of Elect Engg & Computing
The University of Newcastle
Newcastle, Australia
nasimul.noman@newcastle.edu.au

Zhiyong Chen
School of Elect Engg & Computing
The University of Newcastle
Newcastle, Australia
zhiyong.chen@newcastle.edu.au

Mohsen Zamani
School of Elect Engg & Computing
The University of Newcastle
Newcastle, Australia
mohsen.zamani@newcastle.edu.au

Stephan Chalup
School of Elect Engg & Computing
The University of Newcastle
Newcastle, Australia
stephan.chalup@newcastle.edu.au

***Abstract*—Convolutional Neural Networks (CNNs) are currently the most prominent deep neural network models and have been used with great success for image classification and other applications. The performance of CNNs depends on their architecture and hyperparameter settings. Early CNN models like LeNet and AlexNet were manually designed by experienced researchers. The empirical design and optimisation of a new CNN architecture require a lot of expertise and can be very time-consuming. In this paper, we propose a genetic algorithm that can, for a given image processing task, efficiently explore a defined space of potentially suitable CNN architectures and simultaneously optimise their hyperparameters. We named this fast automatic optimisation model fast-CNN and employed it to find competitive CNN architectures for image classification on CIFAR10. In a series of comparative simulation experiments we could demonstrate that the network designed by fast-CNN achieved nearly as good accuracy as some of the other best network models available but fast-CNN took significantly less time to evolve. The trained fast-CNN network model also generalised well to CIFAR100.**

***Index Terms*—Convolutional neural network, Genetic algorithm, Image classification, Deep learning**

I. Introduction

The performance of a neural network is highly dependent on its architecture and hyperparameter settings. Traditionally, neural networks have been designed manually by researchers with a lot of experience in this area. Since the architecture of a well-performing neural network can depend on the problem characteristics, the automatic identification of suitable network architectures has been investigated by many researchers over the past decades. Evolutionary algorithms, a class of generic population-based metaheuristic optimisation algorithms, have proven useful in identifying suitable network models [1]. For this purpose, different evolutionary algorithms such as genetic algorithm (GA) and particle swarm optimisation (PSO) have been used by researchers in different ways. Finding efficient ways to combine neural networks and genetic algorithms has become an attractive research area that has been active for more than twenty years. There is a very good survey in the literature on two basic combination techniques: supportive combinations and collaborative combinations [2]. Supportive combinations use genetic algorithms to assist the design of neural networks. This can be achieved by evolving the parameters and learning rules of neural networks. On the other hand, collaborative combinations aim to determine the neural network weights or the topology or both using genetic algorithms.

With the advent of deep neural architectures and their establishment as powerful machine learning methods, the interest in evolving and training them using evolutionary algorithms has increased. Due to the success of gradient-based algorithms in training the deep neural networks, and because it is very challenging and time-consuming to optimise the topology and weights of neural networks simultaneously, research on the latter topic was very limited. Although there are new promising efforts on training deep neural networks using genetic algorithms for reinforcement learning tasks [3], most of the efforts concentrated on evolving deep architectures using evolutionary algorithms for classification and object detection.

Convolutional Neural Networks (CNNs) are among the most important concepts in deep learning and demonstrated remarkable superiority in many different real-world applications. State-of-the-art CNN architectures, such as AlexNet [4], VGGNet [5], and ResNet [6] were designed by experienced neural network researchers. The requirement of extensive domain knowledge and expertise in neural network design often makes it difficult for inexperienced researchers and application engineers to design successful CNN architectures for a challenging applications or data.

Consequently, there has been an increasing recent efforts to evolve deep neural architectures and their hyperparameters.

Ali Bakhshi was supported by UNIPRS scholarship from University of Newcastle, Australia.

For example, David et al. [7] introduced a GA-assisted method for improving the performance of a deep autoencoder on the MNIST dataset. They considered one set of autoencoder weights as one chromosome in the GA population. They used the inverse of the root mean squared error (RMSE) as the fitness score. After sorting all chromosomes based on their fitness score, they updated the weights of high-ranking chromosomes using backpropagation and replaced the low-ranking chromosomes in the population by the offspring of high-ranking chromosomes. In this work, the fitness score was only used for removing the low-ranked chromosomes from the population. Selection was performed uniformly and applied to the remaining chromosomes with equal probability. When compared with the traditional backpropagation, the GA-assisted method, consistently exhibited better performance in terms of reconstruction error and network sparsity.

Another study, conducted by Suganuma et al. [8], proposed to create the CNN structure and its connectivity by using Cartesian Genetic Programming (CGP). In order to reduce the search space, the convolutional blocks, tensor concatenation and other high-level functional modules were adopted as the node functions of CGP. After training the network using the training data in an ordinary way, they considered the accuracy on the validation dataset as fitness score. They evaluated the performance of the evolved CNN models on the CIFAR-10 dataset and obtained error rates of 6.34% and 6.05% for CGP-CNN(ConvSet) and CGP-CNN(ResSet), respectively. Loshchilov and Hutter [9] used Covariance Matrix Adaptation Evolution Strategy (CMA-ES) for optimising the hyperparameters of deep neural networks (DNNs). They compared the performance of CMA-ES and other state-of-the-art algorithms for tuning the hyperparameters of a CNN on the MNIST dataset. In another work, Sun et al. [10] used a GA for automatically designing CNN architectures for image classification. They borrowed the concept of skip connections used in ResNet [6] for creating deeper networks. Their automatically created architectures are the combination of skip layers and pooling layers. The performance of their model was evaluated on well-known benchmarks such as CIFAR10 and CIFAR100. Sun et al. [11] introduced a method for automatically evolving CNN architecture based on ResNet and DenseNet blocks [12]. They used the combination of three different units; ResNet block units, DenseNet block units, and pooling layer units for generating the CNN architecture. In their encoding strategy, each ResNet or DenseNet unit contained multiple ResNet blocks and DenseNet blocks that helped to increase the depth of network and increased the speed of heuristic search by changing the depth of the network. They compared the performance of their model with 18 state-of-the-art algorithms on the CIFAR10 and CIFAR100 benchmarks.

In the present paper we evolved CNN architectures and tuned the CNN hyperparameters using a genetic algorithm for image classification on the CIFAR10 dataset. There are many hyperparameters that affect the network design and performance. These include learning rate, weight decay factor, and the number of layers determining the network depth. Our proposed fast-CNN method can tune these hyperparameters automatically and quickly. The performance of the resulting CNN models was evaluated on CIFAR10 and also on the CIFAR100 dataset. The rest of the paper is organized as follows: Section II presents a brief review of CNN. Section III describes the proposed algorithm. Then, the experimental setup and experimental results are explained in section IV and V, respectively. Finally, section VI concludes the paper.

II. A BRIEF REVIEW ON CNNS

Convolutional neural networks (CNNs), a class of deep neural network architectures [13], are mostly applied to two-dimensional data such as images and videos. CNNs were inspired by the organization of the animal visual cortex [14]. CNNs can automatically learn hand-engineered filters as they were used in traditional computer vision algorithms. This independence from prior knowledge and human intervention in feature design is the main advantage of CNNs. Sparse interaction, equivariant representation, and parameter sharing are three factors that play an important role in the learning process of a CNN [15]. In traditional artificial neural networks (NNs), the relationship between input and output components was derived from matrix multiplication. But in CNNs, by using sparse interaction and creating kernels smaller than the inputs and applying that to the whole image, this computational burden was reduced considerably. Because of parameter sharing, the network only needs to learn one set of parameters at each location which improves the performance of CNNs over traditional NNs. Moreover, parameter sharing results in a deceptive property called equivariance in which by changing input, the output changes in the same way [16]. In fact, a CNN is a multi-layer neural network that basically consists of two types of layers—convolutional layers and pooling layers. The convolutional layers are the core building block of a convolutional network. In order to produce the feature maps, the input is convolved with trainable filters of a specific size called the receptive field. The pooling layers that are located between the convolutional layers progressively reduce the spatial size of the layers and thereby decrease the computational volume and the number of parameters. The pooling layers down-sample each depth slice of the input independently, according to the defined filter size. The commonly used pooling layer between a sequence of the convolutional layer has a filter size of 2×2 with a stride of 2 [17]. CNN architectures also includes other types of layers such as fully connected layer, normalisation layer etc. Therefore, there are many parameters in a CNN structure such as the number of layers, number of feature maps, receptive field or filter size, and stride size that can be evolved by a genetic algorithm based on the nature of the problem and the available data.

III. THE PROPOSED GENETIC ALGORITHM

A. Overview

In this work, we evolve the best CNN model for the task by searching for the optimal combination of hyperparameters for the network. This section presents the proposed genetic

algorithm, called fast-CNN, for evolving the optimal CNN model for the image classification task on CIFAR10 dataset. The flowchart of proposed algorithm is shown in Figure 1.

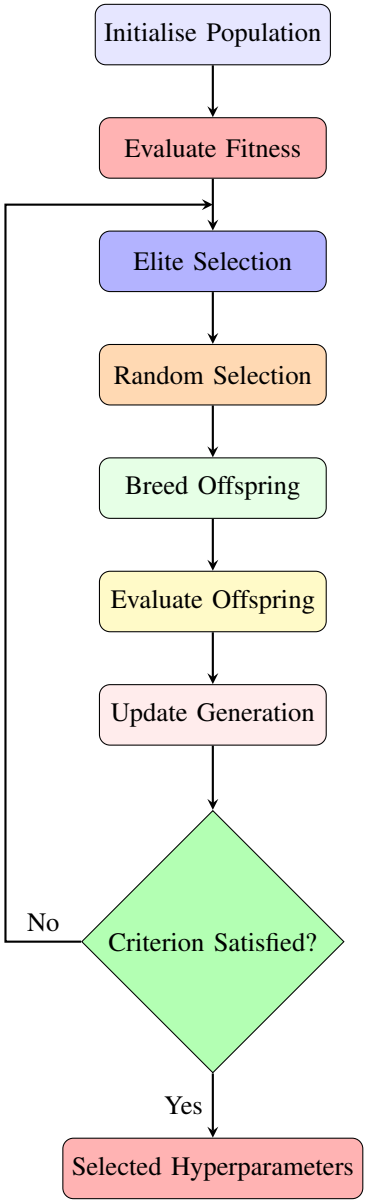

Fig. 1: The flowchart of the genetic algorithm for evolving CNN model

As shown in Fig. 1, the initial population is created by randomly selecting different genes of each individual from the search space. Each individual encodes a CNN model with a specific architecture and hyperparameters in its chromosome. The CNN model is trained independently with the training part of CIFAR10 dataset and the average classification accuracy of the network in the validation phase is considered as the fitness score of the individual. After calculating the fitness score of each network, the population of individuals is sorted in descending order of their fitness scores. Then the next generation is created by a sequence of genetic operations such as elite selection, random selection, and breeding of the new members. The process of generation alteration continues until the termination criterion is satisfied. The fast-CNN framework for evolving CNN models by GA is presented in Algorithm 1.

Algorithm 1: Fast-CNN framework for evolving CNN models

Input: Population size (N_p), Maximum number of generation (G_{max}), the RGB images of training dataset

Output: The best CNN model with its architecture and selected hyperparameters

```
Initialize the population with random combination of hyperparameters
Train the CNN model represented by each individual in the population and calculate its fitness score
N_G ← 0
while N_G < G_max do
    Sort individuals according to their fitness scores
    Select the elite and random individuals from the sorted list as part of the new generation
    Generate offspring by GA operators from selected parents and add to the new generation
    Evaluate offspring
    N_G ← N_G + 1
end
Return the best CNN architecture along with other hyperparameters
```

B. Population initialization

The algorithm starts with an initial population of individuals created by the combination of hyperparameters randomly chosen from the search space. The hyperparameters that our algorithm optimises for finding the best CNN model are the number of network layers, number of feature maps, learning rate, weight decay factor, and momentum. These hyperparameters are encoded in each individual's chromosome. An example of a chromosome with selected values for each hyperparameter is presented in Fig. 2.

L.R	W.D	M	N.L	N.F
1e-2	1e-3	0.7	15	4

Fig. 2: Example of a chromosome representing different hyperparameters of a CNN model.

In Fig. 2, L.R, W.D, M, N.L, and N.F denote the learning rate, weight decay factor, momentum, number of layers, and number of feature maps, respectively. Each of these variables can take different values and our proposed algorithm searches for the optimal combination of these values from a selected set or range of values shown in Table I. The set/range of possible values for each hyperparameters in Table I were selected

according to the past experiences on images classification using neural networks. After random selection of these hyperparameters, the CNN architecture will be created, according to some rules (described below), based on the number of layers (N.L) and the number of feature maps (N.F).

The CNN architecture is created by an alternative combination of the convolutional and pooling layers that are followed by a linear fully connected layer on the top. Moreover, the number, type, and ordering of layers and size of feature maps in each layer are selected randomly for each individual in the initial population. For example, as shown in Fig. 2, for N.L=15 and N.F=4, a 15 layer network will be created by a random combination of convolution and Max-pooling layers that will be followed by an average pooling layer and a linear fully connected layer on top. The rules for creating CNN architecture are listed below:

- The minimum and maximum number of consecutive convolutional layers is 2 and 4, respectively.
- Two pooling layers never occur directly in sequence.
- We keep adding a sequence of convolutional layers and pooling layer until we reach the number of layers (N.L)
- A subsequence of size N.F is selected from the sequence $\{32, 64, 128, 256, 512\}$. The elements of the selected subsequence are randomly used as the feature map value of convolutional block.
- If the number of feature maps (N.F) were less than the number of different convolutional blocks in the generated network, then one/more of the selected feature maps will be repeated randomly.

Besides, for each convolutional layer the same kernel size and stride size is used. Also, each convolutional layer is followed by batch normalization [18] and rectifier activation function [19]. Algorithm 2 shows the summary of population initialisation.

TABLE I: The range/set of values for different hyperparameters to be searched by GA

Hyperparameter	***Values***
Learning Rate (L.R)	0.1, 0.01, 0.001, 0.0001
Weight Decay (W.D)	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
No. of Layers (N.L)	10 – 25
No. of Feature maps (N.F)	3 – 5

C. Fitness Evaluation

In this work, the average classification accuracy of the CNN-model is used as the fitness score of the individual. From each chromosome, the CNN model is constructed and stored in the individual. The 90% split of the training dataset is used for training the network and the 10% split is used for validation. The constructed network is trained by the stochastic gradient descent (SGD) algorithm [20] for a fixed number of epoch ($N_{epoch} = 50$), and then the average classification accuracy in the validation phase is considered as the fitness score. For training each network, the cross-entropy loss is considered as the loss function and the learning rate is reduced by a factor of 10 in every 20 epoch. Algorithm 3 illustrates the details of fitness evaluation.

Algorithm 2: Generating initial population

Input: The population size N_p
Output: The Initialized population P_0
Data: The range/set of values for each hyperparameter are stored in L

```
P_0 ← ∅
while |P_0| < N_p do
    Randomly select learning rate (Lr) from L[L.R]
    Randomly select weight decay factor (wd) from L[W.D]
    Randomly select momentum (M) from L[M]
    Randomly select number of layers (N_l) from L[N.L]
    Randomly select number of feature maps (N_f) from L[N.F]
    Create the network (net) with the selected hyperparameters according to the rules described in Section III-B
    P_0 ← P_0 ∪ net
```

Algorithm 3: Fitness evaluation of an individual

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

```
Create the CNN architecture (net) with the given hyperparameters
v_best ← 0
step ← 0
while step < N_epoch do
    Train the network net on D_train
    Calculate the classification accuracy (v) on D_valid
    if v > v_best then
        v_best ← v
    end
    step ← step + 1
end
Set v_best as the fitness score of the individual
Return net
```

D. Creating New Generation

After evaluating the fitness of every individual in the current generation, the individuals of the current population are sorted in descending order of their fitness scores. Then top $r_f\%$ individuals from the current population are selected as elites and added to the next generation. From the rest of the individuals in the current generation, we randomly select some individuals with probability r_s and add to the next generation. The randomly selected poorly performing individuals can

help to maintain population diversity and prevent premature convergence [21], [22]. This set of elite and randomly selected individuals form the parent pool for generating offspring for the rest of the next generation.

From the parent pool we randomly select two different individuals as parents who participate in the uniform crossover operation for creating two offspring. Since the children's genes are selected randomly from each parents, there are different possible combinations of parents' genes for each child. The offspring undergo mutation operation with a predefined mutation probability m_c. In the mutation operation, one of the network hyperparameters is randomly chosen and its value is modified randomly from the set of possible values for the chosen hyperparameter. This process of creating offspring is repeated until the total number of parents (in parent pool) and the newly created offspring reaches the size of the population. Then the parent pool and the children pool forms the new generation of GA replacing the current one. This process of creating new generation from the current one is repeated for a pre-defined number of generations (N_G) for finding the best network architecture and hyperparameters. Algorithm 4 shows how a new generation is created from the current generation.

Algorithm 4: Creating the new generation of individuals

Input: The current population of individuals with fitness score (P_o), percentage of population retained as elite (r_f), probability of retaining an individual from the non-elite part of the population (r_s), the probability of mutation (m_c), population size (N_p)

Output: The new population (P_{new})

```
Sort all individuals in current population (P_o) in descending order of their fitness scores
Add top r_f% individuals from P_o to the new population P_new
Select individuals from the bottom (1 − r_f)% of P_o with probability r_s and add to P_new
P_parents ← P_new
while size(P_new) < N_p do
    parent1 ← Randomly select an individual from P_parents
    parent2 ← Randomly select an individual from P_parents
    if parent 1 ≠ parent 2 then
        Create two children Child[1] and Child[2] from parent1 and parent2 using uniform crossover operation
        for each offspring in Child do
            r ← Randomly generate a number from (0,1)
            if m_c > r then
                Randomly replace a gene in offspring with a randomly selected value
            end
        end
        P_new ← P_new ∪ Child
    end
end
Return P_new
```

IV. Exprimental Setup

A. Datasets

In this paper, we used CIFAR10 and CIFAR100 datasets [23] as the benchmark. These datasets are widely used in image classification task for evaluating the performance of various deep neural network algorithms and many researcher reported their algorithms' performance on these datasets.

CIFAR10 dataset contains 60000 color images that are divided into 50000 and 10000 images for training and testing, respectively. This dataset, commonly used for image classification problems, consists of 10 classes with 6000 images per class, and each RGB images have the dimension of 32×32. The CIFAR100 dataset is similar to CIFAR10, except it has 100 classes that are grouped into 20 superclasses. There are 500 training images and 100 testing images per class in this dataset.

B. Experimental Environment

In this work, we used Pytorch framework (Version 0.4.0) of Python programming language (Version 3.7) in all experiments. We ran our experiments using the high-performance computing (HPC) services of the University of Newcastle with two GPUs for evolving various CNN architectures and training those during the evolutionary process and after that.

C. Parameter Selection

As mentioned before, the main objective of this paper is to develop a genetic algorithm that can automatically design a deep architecture with optimal number of CNN layers, as well as other hyperparameters such as the learning rate, weight decay etc. Our GA works with a number of parameters which were set based on our experience with GA and using some preliminary studies on the problem under consideration. In our GA the parameters were set as follows: maximum number of generation $G_{max} = 20$, the percentage of population retained as elite $r_f = 0.4$, the probability of retaining an individual from the non-elite part of the population $r_s = 0.1$ and the probability of mutation $m_c = 0.2$.

It should be noted that we used our GA for evolving some of the CNN hyperparameters such as learning rate, weight decay, momentum etc. whereas other hyperparameters like dropout rate, activation function, and optimiser were kept constant. In fact, initially we tried to evolve all of these hyperparameters but based on our preliminary results, some of these hyperparameters were kept constants for the rest of our experiments. In all experiments we used dropout rate $p = 0.5$, the ReLU activation function, and the SGD optimiser. The filter size for convolution layers and stride size for pooling layers were also fixed to 3 and 2, respectively. Additionally, to reduce the computational burden, during the process of evolution by GA, each network is trained with a lower number

of the epoch ($N_{epoch} = 50$). After finding the best network hyperparameters, the selected network is trained with a higher number of the epoch (350 epoch) for improving the average classification accuracy of each network. In addition, to increase the possible combination of layers with various feature maps, the available numbers of feature maps are set to 32, 64, 128, 256, and 512.

V. Experimental Results

In this work we proposed fast-CNN, a conventional GA, for evolving deep CNN architecture as well as finding the hyperparameters of the network. The proposed deep-evolve model is evaluated by applying it for searching the optimal setting of these hyperparameters to increase the image classification accuracy on CIFAR10 dataset. We also evaluated the performance of the evolved CNN model by testing it on CIFAR100 dataset with the same hyperparameter setting.

TABLE II: Hyperparameters of the top five CNN models evolved by fast-CNN

Hyperparameters			**CIFAR10**	**CIFAR100**
Learning Rate	***Weight Decay Factor***	***Momentum***	***Accuracy***	***Accuracy***
0.01	0.01	0.65	94.70	75.63
0.1	0.001	0.55	94.24	74.03
0.1	0.001	0.8	93.80	74.81
0.01	0.001	0.8	93.56	72.12
0.1	0.01	0.7	92.27	74.45

At first, we compared the top 5 individuals in the final population of fast-CNN in Table II. All of these five individuals have the same CNN architecture shown in Fig. 3. We can see that the architecture has an alternation between convolution blocks and pooling layers followed by a fully connected layer. Moreover, each convolution layer is followed by a two-dimensional batch-normalization layer and a rectified linear unit (ReLU) function which are not shown in Fig. 3 for simplicity. The top five combinations of hyperparameters for the CNN-model of Fig. 3 lie in close range as shown in Table II. Their performances on CIFAR10 dataset are also very close. The performance of the top CNN-model is best in both CIFAR10 and CIFAR100 datasets. Performance of other models more or less corresponds in both datasets.

In order to establish the competitiveness of the evolved CNN-model by the proposed fast-CNN algorithm, it is compared with the other state-of-the-art CNN-architectures - created by manually, semi-automatic, and automatically. Table III summarizes the results of this comparison on the CIFAR10 and CIFAR100 datasets.

Other than the classification accuracy, Table III compares the CNN-models in terms of model design technique (manual assistance, semi-automatic, and automatic) and the required computational effort in automatic construction of those models. For the CNN-models designed automatically, Table III reports the required GPU days which can give us a rough estimate about the speed of the algorithm. It must be mentioned that some of the results reported in Table III were reproduced by us and others (indicated in table footnote) were copied from results reported in [10].

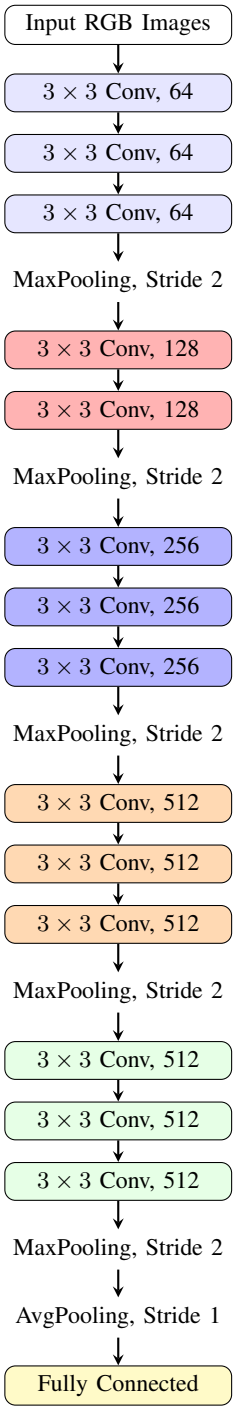

Fig. 3: Architecture of top CNN model(s) evolved by fast-CNN

Since the GPU days are not available for the hand-designed algorithms, we can not compare all algorithms with similar criteria. Hence, we compare the fast-CNN model with hand-designed models in terms of accuracy only, and with the automatically designed models in terms of both accuracy and

TABLE III: The comparisons between the fast-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	-	22400	Automatic
Meta-QNN[a]	93.08	72.86	100	Automatic
CNN-GA[a]	95.22	77.97	35	Automatic
fast-CNN	94.70	75.63	14	Automatic

[a]The values of this algorithm reported in [10]

the required GPU days by the algorithm. As can be seen from the Table III, the model evolved by fast-CNN achieved better accuracy than all manually designed models in both CIFAR10 and CIFAR100 datasets (except DenseNet performed better in CIFAR100). Specifically, in comparison with the manually designed algorithms the classification accuracy of fast-CNN model on CIFAR10 dataset improved by 1.65%, 2.11%, 0.62%, 0.18%, and 4.00% on VGG16, VGG19, ResNet101, DenseNet, and Maxout, respectively.

Compared to the semi-automatically designed models, the fast-CNN model exhibits very competitive performance in terms of classification accuracy, but requires lower GPU days. For example, in comparison with the CNN model designed by the semi-automatic algorithms 'Genetic CNN' the classification accuracy of the fast-CNN model is 1.80% higher with almost similar GPU days. Although the classification accuracy of fast-CNN model is 1.67% and 0.92% lower than 'Hierarchical Evolution', and 'Block-QNN-S' respectively, it takes only around one-twentieth of the GPU days of 'Hierarchical Evolution' and one-sixth of the GPU days of 'Block-QNN-S'.

When compared with the automatically designed models, it is evident from the Table III that the classification accuracy of fast-CNN model is 0.10% and 0.71% better than 'Large-scale Evolution' and 'NAS' models respectively but fast-CNN required much lower GPU days than those algorithms. Also, compared with 'CGP-CNN' and 'Meta-QNN', the classification accuracy of fast-CNN model improved by 0.68% and 1.62% respectively, with half of the GPU days required for 'CGP-CNN' and one-seventh of the GPU days required for 'Meta-QNN'. Finally, the classification accuracy of 'CNN-GA' model is 0.52% better than the fast-CNN model but the GPU days of this algorithm is around twice larger than fast-CNN.

In order to further assess the quality of the fast-CNN model, we used the transfer learning to evaluate the performance of the the CNN architecture evolved on CIFAR10 dataset with the same hyperparameters for CIFAR100 dataset. Therefore, considering the results achieved on CIFAR100 dataset, the fast-CNN model achieves 0.69%, 1.59%, 0.24%, and 14.23% improvement over VGG16, VGG19, ResNet101 and Maxout, respectively. DenseNet peformed 0.98% better than fast-CNN in terms of accuracy. Moreover, comparing the classification accuracy of the models by the semi-automatic algorithms, the fast-CNN model shows 4.66% higher and 3.72% lower accuracy rate than 'Genetic CNN' and 'Block-QNN-S', respectively. In addition, the classification accuracy of fast-CNN model is 2.34% and 1.37% lower than 'CNN-GA' model and 'Large-scale Evolution' model, respectively, but 2.77% higher than the 'Meta-QNN', model.

In summary, fast-CNN outperforms the state-of-the-art algorithms in terms of either classification accuracy or GPU days, or both. Although some of the semi-automatic algorithms show better classification accuracy, they need manual intervention and much longer GPU days. Moreover, despite the classification accuracy of some automatically designed model is higher than the fast-CNN model, the later is at least two times faster than all of them.

VI. Conclusion

In this paper, we present a deep evolutionary approach for automatically discovering the best architecture for a CNN model as well as finding the best combination of hyperparameters for the network. We used a conventional genetic algorithm, called fast-CNN, to find the best combination of hyperparameters for CNN such as the number of layers, number of feature maps, learning rate, weight decay factor, and momentum. The CNN model was evolved on CIFAR10 dataset but the performance of the network was evaluated on both CIFAR10 and CIFAR100 datasets. The performance of the evolved fast-CNN model was compared with 13 state-of-the-art algorithms in terms of classification accuracy, GPU days, and parameter setting method. Among these 13 algorithms, five were manually designed, three used a semi-automatic design, and the remaining were automatically designed. The CNN model designed by the proposed GA outperformed all manually designed models in terms of classification accuracy. Although the classification accuracy of the fast-CNN model was slightly lower than some semi-automatic and automatically designed models, in terms of GPU days it was the best compared to all other methods.

Acknowledgment

This work was supported by a strategic pilot grant awarded by the School of Electrical Engineering and Computing at The University of Newcastle, Australia.

References

[1] P. A. Vikhar, "Evolutionary algorithms: A critical review and its future prospects," in *International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), 2016*, pp. 261–265, IEEE, 2016.

[2] J. D. Schaffer, D. Whitley, and L. J. Eshelman, "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., COGANN-92.*, pp. 1–37, IEEE, 1992.

[3] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep Neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *arXiv preprint arXiv:1712.06567*, 2017.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing systems*, pp. 1097–1105, 2012.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[7] O. E. David and I. Greental, "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, 2014.

[8] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 497–504, ACM, 2017.

[9] I. Loshchilov and F. Hutter, "CMA-ES for hyperparameter optimization of deep neural networks," *arXiv preprint arXiv:1604.07269*, 2016.

[10] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Automatically designing CNN architectures using genetic algorithm for image classification," *arXiv preprint arXiv:1808.03818*, 2018.

[11] Y. Sun, B. Xue, and M. Zhang, "Automatically evolving CNN architectures based on blocks," *arXiv preprint arXiv:1810.11875*, 2018.

[12] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, IEEE, 2017.

[13] I. Arel, D. C. Rose, T. P. Karnowski, *et al.*, "Deep machine learning-a new frontier in artificial intelligence research," *IEEE Computational Intelligence Magazine*, vol. 5, no. 4, pp. 13–18, 2010.

[14] M. Matsugu, K. Mori, Y. Mitari, and Y. Kaneda, "Subject independent facial expression recognition with robust face detection using a convolutional neural network," *Neural Networks*, vol. 16, no. 5-6, pp. 555–559, 2003.

[15] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[16] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[17] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Artificial Neural Networks–ICANN 2010*, pp. 92–101, Springer, 2010.

[18] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[19] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.

[20] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

[21] C. M. Anderson-Cook, "Practical genetic algorithms," *Journal of the American Statistical Association*, vol. 100, no. 471, pp. 1099–1099, 2005.

[22] S. Malik and S. Wadhwa, "Preventing premature convergence in genetic algorithm using dgca and elitist technique," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, no. 6, 2014.

[23] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," tech. rep., Citeseer, 2009.