Neural Computing and Applications (2023) 35:1569–1580
https://doi.org/10.1007/s00521-022-07700-9

ORIGINAL ARTICLE

Fully automatic CNN design with inception and ResNet blocks

Togzhan Barakbayeva[1] · Fatih M. Demirci[1,2]

Received: 22 December 2021 / Accepted: 2 August 2022 / Published online: 30 September 2022

Abstract
Although convolutional neural networks (CNNs) are widely used in image classification tasks and have demonstrated promising classification accuracy results, designing a CNN architecture requires a manual adjustment of parameters through a series of experiments as well as sufficient knowledge both in the problem domain and CNN architecture design. Therefore, it is difficult for users without prior experience to design a CNN for specific purposes. In this paper, we propose a framework for the automatic construction of CNN architectures based on ResNet, DenseNet, and Inception blocks and the roulette wheel selection method with a dynamic learning rate. Compared with the state of the art, the proposed approach has a significant improvement in the domain of image classification. Experimental evaluation of our approach including a comparison with the previous works on three benchmark datasets demonstrates the effectiveness of the overall method. The proposed algorithm not only improves the previous algorithm but also keeps the advantages of automatic CNN construction without requiring manual interventions. The source code of the our framework can be found at https://github.com/btogzhan2000/ea-cnn-complab.

Keywords Convolutional neural networks · Genetic algorithms · Automatic CNN construction

1 Introduction

Convolutional Neural Networks (CNNs) have demonstrated promising performance in a wide variety of problems. A CNN is firstly trained on training data and then validated on validation data, where its performance is further evaluated and its hyper parameters are checked [1]. In recent years, a fair amount of CNN architectures, e.g., GoogLeNet [2], ResNet [3], and DenseNet [4], ranging from a few to hundreds of layers have been introduced. While such models have been successfully used in different domains, it is well-known that the construction of a CNN is a time-consuming process, which requires a manual choice of hyper parameters and performing a trial-and-error approach by conducting experiments dozens of times for various depths and combinations of different parameters to reach optimal settings. This process also introduces an extra overhead when the network depth gets bigger. Furthermore, handcrafting a CNN requires expertise both in the architecture design and the problem domain, shrinking the application range of CNNs. Besides, manually designed CNNs for a specific task may not be directly applicable in different tasks since the parameters are particularly designed for one specific task and are achieved through the trial-and-error process. Therefore, utilizing a CNN model in a different context usually requires performing the time-consuming trial-error process once again to find the optimal settings. Despite the promising performance of manually designed CNN architectures, the drawbacks mentioned above pose the need for an automatic search of good CNN architectures. In this paper, we propose a completely automatic framework for finding a suitable CNN model applied to image classification. The experimental evaluation demonstrates the superior performance of the proposed framework over an existing state-of-the-art approach.

✉ Fatih M. Demirci
muhammed.demirci@nu.edu.kz

Togzhan Barakbayeva
togzhan.barakbayeva@nu.edu.kz

[1] Department of Computer Science, School of Engineering and Digital Sciences, Nazarbayev University, Nur-Sultan, Kazakhstan

[2] Department of Computer Engineering, Ankara Yildirim Beyazit University, Ankara, Turkey

To address the issue of handcrafting CNNs, a series of automatic algorithms for finding a good CNN have been proposed. Among them, the algorithms based on evolutionary algorithms (EA) and reinforcement learning (RL) such as AE-CNN [1], neural architecture search (NAS) [5], Genetic CNN [6], Block-QNN-S [7], Large-scale evolution [8], and meta-modelling (MetaQNN) [9] have shown outstanding performance exceeding manually designed counterparts. However, they have some restrictive drawbacks. Specifically, methods employing reinforcement learning require vast computational resources to perform training. For example, NAS [5], which is based on RL, took 800 Graphics Processing Units (GPUs) to find the best CNN. Since such computational power is not available to many end-users, this considerably limits the applicability of this method in practice. Consequently, one objective of the proposed framework is to yield promising results with limited computational resources. Evolution-based approaches, such as Genetic CNN [6] also show competitive results. However, the genetic algorithm is only used in the network generation process and training is performed separately, which requires human expertise, reducing the effectiveness of automatic CNN design. Another semi-automatic approach, Block-QNN-S [7] creates several small networks and combines them into a larger CNN model. This approach achieves competitive state-of-the-art accuracy. However, properly designing some layer types, e.g., pooling layers, still require human expertise to be added to the CNN framework. Large-scale evolution [8] adaptively changes the depths of CNN models by using variable-length encoding. Although effective, it only employs the mutation operator without crossover. However, these two operators together play an important role in both local and global searches. MetaQNN [9] is an RL-based algorithm, which sequentially picks CNN layers using the ϵ greedy strategy initially proposed by Mnih et al. [10] in a different domain. The approach achieves promising results. However, the same set of hyperparameters is used to train the network limiting the advantages of the completely automatic construction process. It is important to note the high classification accuracy obtained by an evolutionary algorithm AE-CNN [1], where the network is automatically constructed based on ResNet and DenseNet blocks. As a result, CNN architectures designed by AE-CNN outperformed some handcrafted and automatically designed CNNs.

Looking into AE-CNN more closely, one notices that it has improved existing approaches by removing some of the limitations to fully utilize the advantages of the evolution-based algorithms. In particular, Genetic CNN [6] was further developed by using variable-length encoding strategy instead of fixed-length, and Large-scale evolution [8] was improved by the addition of crossover along with the mutation operator. However, there are only two types of blocks that participate in the construction of the network, ResNet and DenseNet blocks [1]. Although the network constructed using ResNet and DenseNet blocks demonstrates a promising performance by utilizing shortcut connections, it can be noted that the final model does not heavily consider the image at different spatial sizes. For example, if the training dataset includes one picture of a dog standing on grass and occupying only the central part of the image and another picture only covering the face of the dog, the network needs to deal with the extraction of features both at local and global scales. The proposed framework, on the other hand effectively deals with this issue by extending AE-CNN to include additional types of deep learning layers along with ResNet and DenseNet to solve the issue. Namely, the proposed algorithm utilizes inception blocks introduced by GoogLeNet to realize multi-level feature extraction process.

The key idea of the Inception module is to use several filter sizes (1×1, 3×3, and 5×5) instead of only one, concatenate and pass them to the next layer along with the max-pooling layer [2]. This way, the features of the image at different spatial sizes are preserved. Also, to reduce the computational cost, additional 1×1 filters are added before 3×3 and 5×5 convolutional layers, which makes the Inception block not only effective for feature extraction but also computationally inexpensive. In our approach, different types of inception blocks are used to fully utilize the advantages of the module.

Although genetic algorithms generally follow a similar structure, i.e., population initialization, fitness evaluation, offspring generation, and selection as main steps, there are differences in chosen methods and implementations. For example, different methods can be used for selection, which participates in parent and next-generation selection processes. One example of such methods is tournament selection, which is used in a number of previous models, e.g., [8, 11–13]. However, the selection process could be extended to include other approaches in order to improve its effectiveness. A good example can be found in Genetic CNN, which shows promising performance and employs the Russian roulette selection method [6]. While binary tournament selection in AE-CNN chooses the individual with the highest fitness among two random individuals, the roulette method in Genetic CNN gives the advantage to individuals with higher fitness by assigning higher probabilities to them [6]. To have a variety of available selection approaches, we include both methods in our framework. As an additional point, we should add that AE-CNN utilizes the bottleneck block structure of ResNet, which is only used to reduce the dimensions during the formation of the network. On the other hand, the proposed framework

employs the basic ResNet blocs along with existing bottleneck blocks to construct better-performing individuals.

Taking into consideration all points mentioned above, one may observe that there is a need for an improved algorithm for the automatic development of CNN architectures with good performance subject to computational constraints. Furthermore, this automatic construction should require no additional expertise from the users. To fill this gap, we present an improved AE-CNN-based framework that does not suffer from its shortcomings as highlighted above and finds a completely automatic CNN architecture with good performance.

Our contributions are summarized as follows:

- Inception module is introduced to the network construction and various types of inception blocks are added. This way, we ensure that key features of the images are extracted at different scales.
- Roulette wheel selection method is added to participate in the environmental selection and parent selection processes.
- Basic ResNet block structure is included along with existing bottleneck block structure to construct better-performing individuals as it gains accuracy at the expense of memory.
- During the training process, the learning rate is dynamically altered depending on epoch size, resulting in a significant increase in the final classification accuracy.

The preliminary version of our framework that only has the addition of one type of inception block appeared in [14]. The remaining part of the paper is organized as follows. Section 2 elaborates on the proposed framework in detail. Then, the design of the experiment and obtained results are described in Sect. 3. Lastly, conclusions and directions for future work are outlined in Sect. 4. The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

2 The proposed framework

2.1 Algorithm overview

In the beginning, the population of size N is initialized by the implemented encoding method and the fitness of each individual is evaluated. Then the evolutionary process starts and continues until it reaches the maximal generation number. During each generation, an empty population is created to further include offspring generated by parent individuals, which are chosen through the binary tournament selection or roulette wheel method [15]. Afterward, the offspring are generated by crossover and altered by mutation operations with some probabilities. The offspring generation continues until the population size reaches N. The process is followed by the fitness evaluation of individuals in this population. As there are currently two populations (initial individuals and offspring), the environmental selection is employed to choose N individuals from both populations, which will be further used as both parents and initial individuals in the next generation. The process is repeated until the maximal generation is reached. Finally, the individual with the highest fitness is chosen from the final population and decoded to a CNN architecture. The proposed framework is summarized in Algorithm 1. The structure of CNN and more detailed descriptions of each evolution stage, i.e., population initialization, fitness evaluation, offspring generation, and environmental selection are presented below.

Algorithm 1: Proposed Framework

Input: Population size N, maximal generation number $maxGen$
Output: CNN architecture with the best classification accuracy

$P_0 \leftarrow$ population initialization of size N;
fitness evaluation of individuals in P_0;
$gen \leftarrow 0$;
while $gen < maxGen$ **do**
  $Q_{gen} \leftarrow$ initialize empty set;
  while $|Q_{gen}| < N$ **do**
    $parent_1, parent_2 \leftarrow$ choose by binary tournament or roulette wheel selection;
    $offspring_1, offspring_2 \leftarrow$ generate by crossover and alter by mutation with some probabilities;
    $Q_{gen} \leftarrow$ include $offspring_1$, $offspring_2$;
  end
  fitness evaluation of individuals in Q_{gen};
  $P_{gen+1} \leftarrow$environmental selection of N individuals from P_{gen} and Q_{gen};
  $gen \leftarrow gen + 1$
end
choose the individual with the best classification accuracy from P_{gen}

2.2 Network structure

In the proposed framework, three types of blocks along with pooling layers form the base of the network: ResNet, DenseNet, and Inception. While ResNet and DenseNet introduce skip connections that improve classification results by coping with the gradient vanishing problem and learning features better [1], Inception deals with multi-level feature extraction by the usage of different sized filters [2].

2.2.1 ResNet

There are two types of ResNet blocks: bottleneck and newly-added basic block. Both of them consist of convolutional layers followed by one skip connection where the input is added to the output. The difference is that the bottleneck block has three convolutions with filter sizes 1 $\times$ 1, 3 $\times$ 3, and 1 $\times$ 1, respectively, whereas the basic block consists of two convolutions both with 3 $\times$ 3 filters (see Fig. 1) Initially, the bottleneck structure was developed for practical reasons, as there are dimension reductions through 1 $\times$ 1 filters, which could be useful for deeper networks [2]. Although one may expect that adding both types of ResNet blocks may increase overall memory requirements, time complexity is shown to be the same [2].

2.2.2 DenseNet

DenseNet is composed of four convolutional layers where each convolutional layer receives inputs from all previous layers and feeds all consequent layers.

2.2.3 Inception

The Inception block has 1 $\times$ 1, 3 $\times$ 3, and 5 $\times$ 5-sized filters at each layer followed by a max-pooling layer that helps capture different scales. Prior to 3 $\times$ 3 and 5 $\times$ 5 filters, a 1 $\times$ 1 filter is added to reduce the computational complexity [2]. There are different types of inception blocks, among which one is chosen randomly during the network construction process. Since the parameters are taken directly from the inception blocks of GoogLeNet, some types might have the same structure. Table summarizes the inception types and their parameters utilized in the proposed framework.

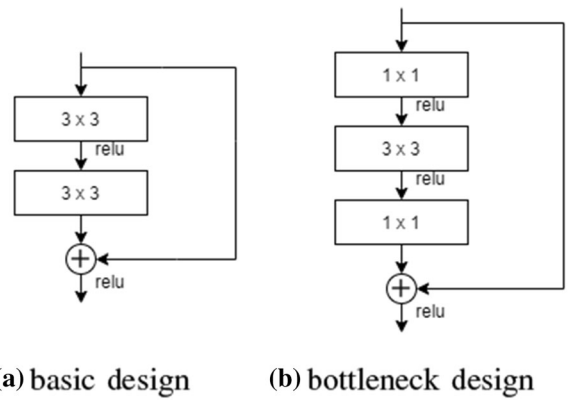


Fig. 1 ResNet block structure

2.3 Population initialization

In the first stage of evolution, the population of CNN architectures is generated. The initialized population consists of N individuals, and each individual represents a CNN model encoded by the proposed strategy. Specifically, we form a CNN by utilizing ResNet, DenseNet, Inception blocks, and pooling layers, therefore four types of units participate in the construction: ResNet block unit (RBU), DenseNet block unit (DBU), Inception block unit (IBU), and Pooling layer unit (PU). RBU and DBU consist of one or more ResNet and DenseNet blocks, respectively, which can be stacked to form one unit. On the other hand, IBU contains one Inception block, and PU is composed of either max or mean pooling layers. Initial individuals are constructed uniformly at random based on the above-mentioned units. Since pooling layers reduce the input size by half, we limit its usage in the framework in order not to shrink the input dimensions very fast.

2.4 Fitness evaluation

Fitness evaluation is a major step in assessing individuals in the population and selecting the most promising ones. It is performed after the population initialization and each time the new population is generated. As each individual represents an encoded CNN architecture design, the classification accuracy is used as a fitness evaluation measure. In this process, an individual is subject to both training and validation.

In the training step, weights are initialized through Xavier initialization method [16] and training is performed through stochastic gradient descent (SGD) [17]. The best classification accuracy on validation data represents the fitness of an individual.

2.5 Offspring generation

At each generation, offspring population is created by performing crossover between selected parents and mutation performed on the individuals. To choose parents, the binary tournament selection method or roulette wheel selection are employed. While binary tournament selection ensures population diversity, roulette wheel method gives preference to high-fitness individuals. According to the binary tournament selection method, two individuals are selected at random from the population and the individual with the highest fitness value is selected as the first parent for offspring generation. The second parent is chosen in a similar way. During roulette wheel selection, there is more priority to individuals with higher fitness, i.e., the probability that a high-fitness individual will become a parent is

bigger. The method is implemented by finding the probability for each individual through the division of fitness value by the total fitness, then by choosing randomly according to the probability distribution.

Then one-point crossover operation is performed between parents with a certain probability. Precisely, crossover is performed by randomly selecting a position from the first and second parents, separating each parent into two different parts from this position, and then combining the first part of the first parent and the second part of the second parent to generate the first offspring. The second offspring is constructed similarly, i.e., by combining the first part of the second parent and the second part of the first parent. To guarantee a valid CNN after crossover, some adjustments are made automatically to the resulting architecture. For example, we ensure that the input size is changed to match the output size of the previous unit at selected positions. One notable advantage of this algorithm is that crossover can be performed on variable-length networks.

We should also add that the generated offspring goes through mutation with a certain probability, ensuring global search. The mutation method is selected among three approaches at the selected position: addition of RBU, DBU, IBU, or PU, removal of chosen unit, and modification of RBU, DBU, or PU. If the selected unit is IBU and the modification method is chosen, then the position is re-selected. Proper changes in input and output sizes are done in order to yield a valid CNN.

2.6 Environmental selection

After the offspring generation, the total population consists of both parent and offspring individuals and there is a need to choose N individuals from the total population in order to proceed with the evolution process. To guarantee diversity and convergence in the population after the selection, an individual with the best fitness is included, and the remaining individuals are chosen by either the binary tournament selection or roulette wheel selection method. To idea of including the best CNN in the next generation is to improve the overall performance of the next generation over the current one. Furthermore, the addition of roulette wheel selection method gives more options that potentially yield better models at this stage of evolution.

3 Experiment design and results

3.1 Benchmark dataset

We evaluate the proposed method and compare its performance to the previous framework (AE-CNN) on two datasets: CINIC-10 and SVHN (The Street View House Numbers).

CINIC-10 contains images both from CIFAR-10 [18] and ImageNet [19] datasets, which are widely used in image classification tasks. Therefore, CINIC-10 can be considered as a fair choice to evaluate both methods. This dataset consists of a total of 270,000 images of which 60,000 are from CIFAR-10 and 210,000 from ImageNet. The whole dataset is divided equally into the train, validation, and test subsets. Figure 2 illustrates sample images taken from CINIC-10.

SVHN dataset consists of real-world digit images cropped from house number pictures and has all digits from 0 to 9 [20]. The number of images in this dataset exceeds 600,000. Figure 3 represents some of the sample images. For the experiments, we used individually cropped digits.

3.2 Parameter settings

In the training process of the individual, SGD has a batch size of 64. The maximum number of IBUs in each CNN is 4, while each IBU has 1 Inception block. The parameters of the Inception block are taken to match the parameters of the inception blocks of GoogleNet [2]. The momentum parameter is set in the same way as in AE-CNN settings [1]. While in some experiments, the learning rate is fixed at

Fig. 2 Sample views of the classes from CINIC-10

Fig. 3 Sample views of the classes from SVHN

the rate of 0.1, it is dynamically adjusted in others. In case a dynamic learning rate is adopted, the idea is to speed up learning when suitable and to slow down learning near local minima [21, 22]. More precisely, when a dynamic learning rate is utilized in our framework, its value is determined based on the current epoch number. In this context, for epoch numbers 0-10, the learning rate is 0.1, and for epoch number greater than 10 and 20, it is taken as 0.01 and 0.001, respectively. The adjustment was made to have higher learning rates in initial epochs and smaller towards the end of training to increase overall accuracy. These specific values are empirically selected using our training sets, whose sample views are depicted in Figs. 2 and 3.

The experiment was run on a computer equipped with an Nvidia GeForce GTX 1080 Ti GPU card. The experimental settings were fully implemented in Python 3.7 and PyTorch deep learning frameworks. For comparison purposes, the experiment was conducted for each new addition proposed in this paper. Namely, in each experiment, two algorithms were evaluated and compared: the previous algorithm, AE-CNN, and the proposed framework with each new addition. The maximum epoch number was set to 30 for both frameworks.

3.3 Experimental results on CINIC-10

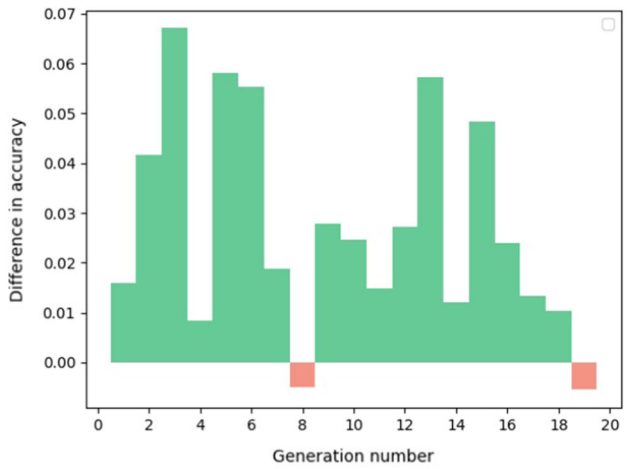
In the following experiments, the average classification accuracy over 20 generations is demonstrated for both AE-CNN and the proposed framework. In particular, we subtract the average classification accuracy of AE-CNN from that of the proposed algorithm in each generation. A positive difference demonstrates the amount by which the proposed framework outperforms AE-CNN with a new modification. This difference is colored green in the following figures. Similarly, the negative differences, which are shown in red, present that AE-CNN performs better than the proposed framework. In the follow-up experiment, we increased the number of epochs to 50. We should note that the additions are incremental, i.e, if in one experiment we have added different types of the inception blocks, then it implies that the next experiment already has this addition. We will now present the effects of these additions below.

3.3.1 Addition of inception (3a) block

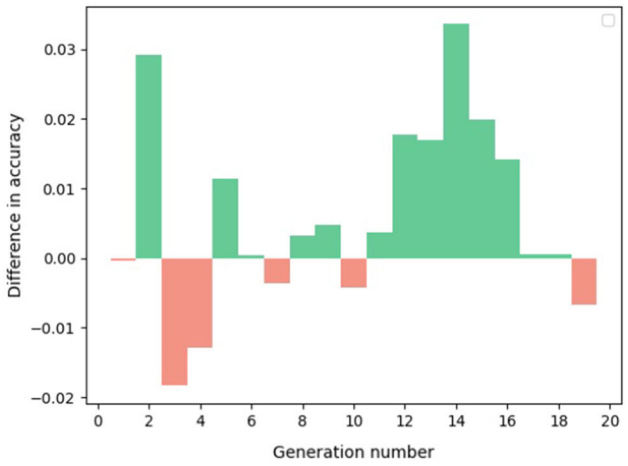
Experimental results for epoch size 30 reveal that the difference in average classification accuracy between the proposed and AE-CNN methods is mostly positive (17 times out of 19), showing that the introduction of inception blocks dramatically increases the average accuracy in most generations. The outcome supports our initial hypothesis that the addition of inception blocks improves the overall image classification accuracy. A noticeable decline in error rates over generations shows that inception blocks improve the algorithm performance through feature extraction in different spatial sizes. It also reveals that inception blocks have a huge potential in automatic algorithms for CNN architecture design and that they could be used for better feature learning.

The next experiment with epoch size 50 still shows a better classification performance than the previous work in most generations (13 times out of 19). Although fluctuations might be observed in initial generations, they are followed by a substantial increase in classification accuracies starting from the 11th generation. Overall, the results summarized in Fig. 4 demonstrate the improved classification accuracy of the proposed method over the previous work with the addition of inception blocks and present its potential in different problems.

We should note that while it took 5 GPU days to run AE-CNN with epoch size 30, the proposed framework with inception blocks needed 7 GPU days. As for epoch size 50, the computation time is 7 GPU days for AE-CNN and 10 days for our algorithm. As a result, the proposed framework outperformed the previous model with the cost of some extra computational complexity.



(a) epoch size 30



(b) epoch size 50

Fig. 4 Difference in classification accuracy between the proposed and previous approaches (AE-CNN) obtained in each generation with the addition of inception (3a) block. The green bars demonstrate the amount by which the proposed framework outperforms AE-CNN (color figure online)

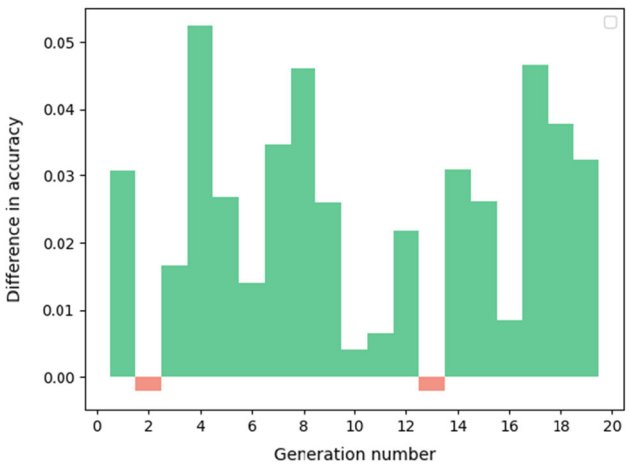


Fig. 5 The classification accuracy difference obtained in each generation with the addition of different types of inception blocks

3.3.2 Addition of various inception blocks

In this experiment, our goal was to see how other types of inception blocks as opposed to inception 3(a) affect the overall performance. Thus, we included all types of inception blocks as shown in Table 1 in the network generation. The results shown in Fig. 5 reveals that the usage of different inception block types has a significant positive effect on the overall performance. As one may observe from the graph, there are positive differences in 17 generations out of 19. This presents the potential of using different inception block types in automatically finding a suitable deep learning model.

3.3.3 Adjusting learning rate to epoch size

Our next experiment deals with evaluating the performance of the proposed framework when the learning rate is determined dynamically based on the epoch number as mentioned at the beginning of this subsection. According to our observations, as the learning rate was adjusted, the classification accuracy saw a dramatic growth (Fig. 6).

Table 1 Types of inception blocks and parameters [2]

type	# 1 × 1	Reduce	# 3 × 3	Reduce	# 5 × 5	Proj
Inception (3a)	64	96	128	16	32	32
Inception (3b)	128	128	192	32	96	64
Inception (4a)	192	96	208	16	48	64
Inception (4b)	160	112	224	24	64	64
Inception (4c)	128	128	256	24	64	64
Inception (4d)	112	144	288	32	64	64
Inception (4e)	256	160	320	32	128	128
Inception (5a)	256	160	320	32	128	128
inception (5b)	384	192	384	48	128	128

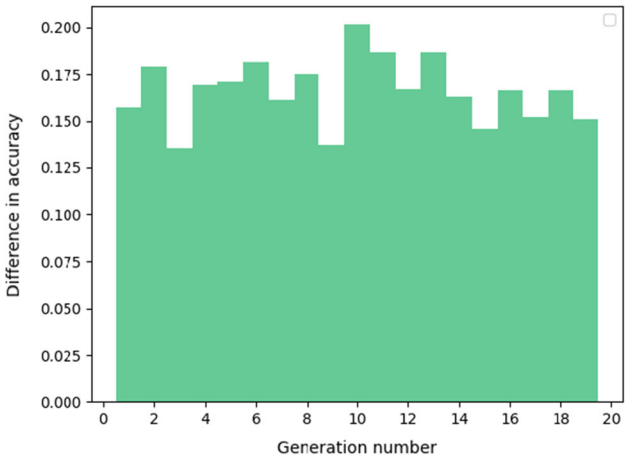
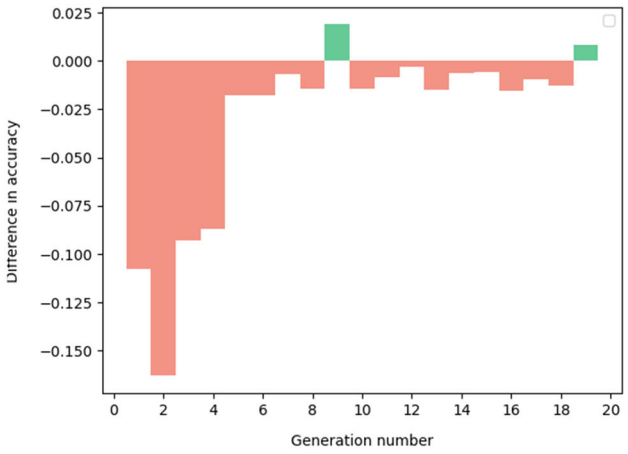
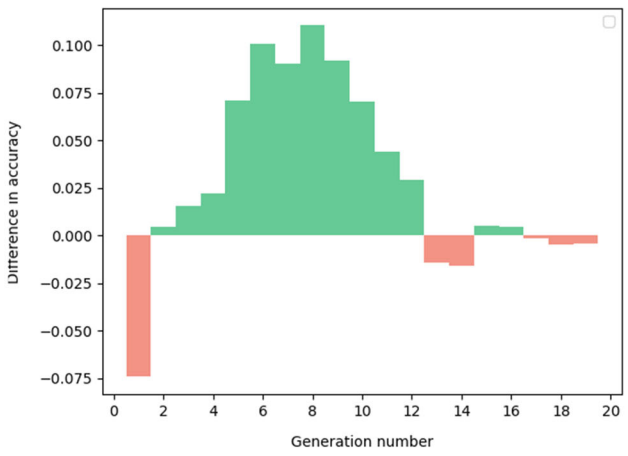


Fig. 6 The classification accuracy difference obtained in each generation with the adjusted learning rate



(a) epoch size 30



(b) epoch size 50

Fig. 7 The classification accuracy difference obtained in each generation with the addition of roulette wheel selection

This is clearly visible since the positive difference appears in all generations.

3.3.4 Addition of roulette wheel selection method

The objective of the next experiment is to evaluate the addition of the roulette wheel selection to offspring generation, which only consisted of binary tournament selection in AE-CNN. According to the initial results shown on the left of Fig. 7, there are mostly negative differences (17 times out of 19) meaning that AE-CNN performs better although the differences are small, especially after the fourth generation. Despite the fact that the addition of roulette wheel selection does not yield any improvements, having a variety of options in the algorithm to find architectures might still be beneficial. To investigate this in more detail, we repeated the same experiment after increasing the number of epochs to 50. The results shown in the right of Fig. 7 are much better, presenting 13 positive differences out of 19. We also observe that the performance of the proposed work is still very close to that of AE-CNN in remaining 5 generations giving negative differences. Overall, the results depict the benefit of adding different options to the offspring generation procedure during the CNN construction.

3.3.5 Addition of basic ResNet block

In the next experiment, we aimed at measuring the effect of adding the basic ResNet block to the architecture design process. After including the basic ResNet block, we noticed that the proposed framework gained a noticeable improvement in accuracy. Specifically, we observed that in 14 out of 19 generations, the proposed framework outperformed AE-CNN, showing how ResNet block could be

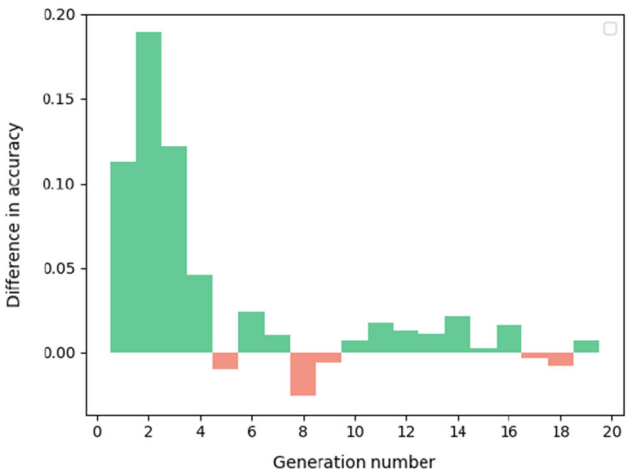
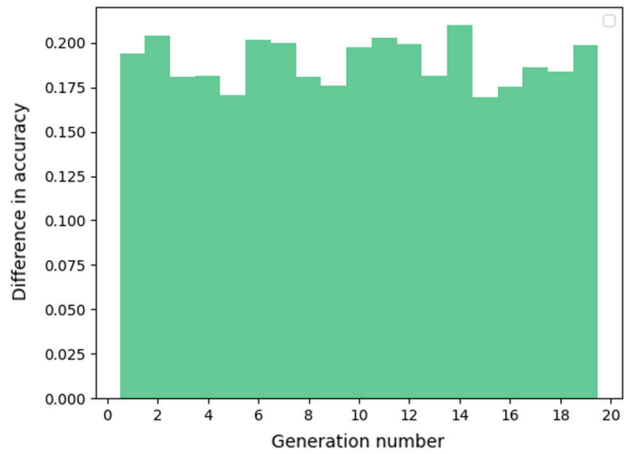


Fig. 8 The classification accuracy difference obtained in each generation with the addition of basic ResNet block

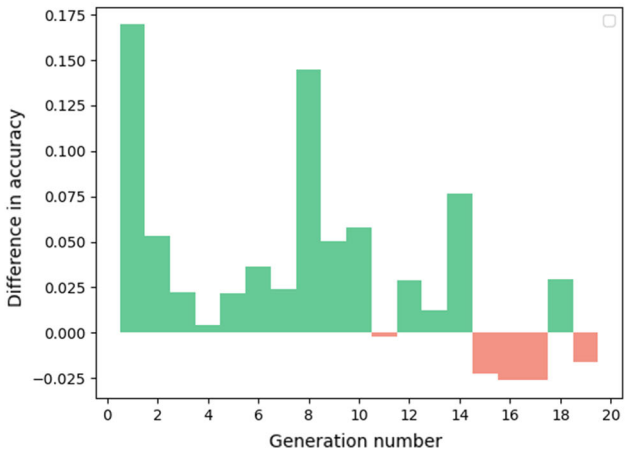
valuable in the overall construction (Fig. 8). Similar to our observations in the previous experiment, the performance of the proposed framework was close to that of the previous work in the remaining 5 generations, where AE-CNN achieved a slightly better performance.

In addition to the previous experiments, we also made an overall comparison of the proposed method with AE-CNN in an epxerimental setup where the proposed method included all of the additions presented above. For the sake of fairness, the AE-CNN algorithm was run with both fixed and varying learning rates.

In the case where AE-CNN was utilized with a fixed learning rate, the proposed framework obtained better performance across all generations. When the varying learning rate was utilized in AE-CNN, we noticed the increase in its performance although the proposed method still obtains better scores in 14 out of 19 generations. This, in fact, verifies the importance of having a varying learning rate in the final CNN model. The results summarized in Fig. 9 suggest the high potential of the proposed framework with new additions for the automatic development of CNNs.



(a) Fixed learning rate



(b) Dynamic learning rate

Fig. 9 After arming the proposed method with all new additions presented in the paper, it outperforms AE-CNN with fixed learning rate across all generations as shown in **a**. When a dynamic learning rate is utilized in AE-CNN, its performance is slightly increased as shown in **b**. However, the proposed framework still outperforms AE-CNN in 14 out of 19 generations

Table 2 The best CNN architecture found on CINIC-10 dataset

id	Type	Configuration
0	RBU	Amount = 3, in = 3, out = 128
1	PU	Max pooling
2	RBU	Amount = 7, in = 128, out = 128
3	PU	Mean pooling
4	PU	Max pooling
5	IBU	Type = 5a, in = 128, out = 832
6	PU	Mean pooling
7	IBU	Type = 4e, in = 832, out = 832

The best CNN architecture found when performing experiments on CINIC-10 dataset is given in Table 2

3.4 Experimental results on SVHN

Both AE-CNN and the proposed framework were also evaluated on the SVHN dataset. Since the previous experiments demonstrate the benefits for all proposed additions, our framework included all of them, i.e., all types of inception blocks, dynamic learning rate, the addition of roulette wheel selection and basic ResNet block in the SVHN experiments. According to the results shown in Figure 10, the proposed framework achieved a better performance in 16 out of 19 generations.

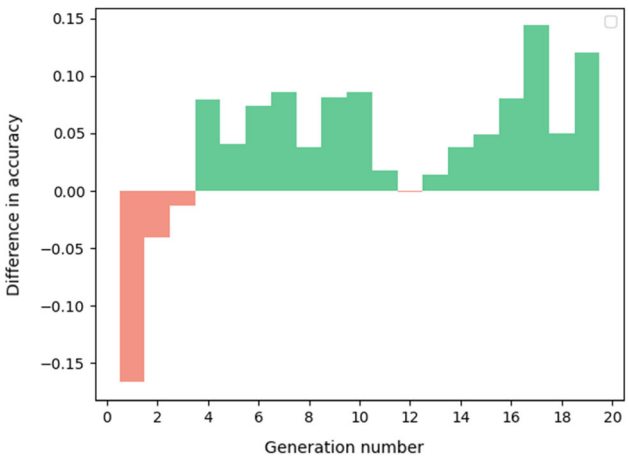


Fig. 10 Difference in classification accuracy obtained in each generation (AE-CNN vs Proposed framework) on SVHN

Table 3 The best CNN architecture found on SVHN dataset

id	Type	Configuration
0	RBU	Amount = 5, in = 3, out = 128
1	RBU	Amount = 7, in = 128, out = 128
2	PU	Max pooling
3	IBU	Type = 4a, in = 128, out = 512
4	RBU	Amount = 6, in = 512, out = 64
5	IBU	Type = 3b, in = 64, out = 480
6	PU	Mean pooling
7	DBU	Amount = 5, k = 40, in = 480, out = 232
8	IBU	Type = 3a, in = 232, out = 256
9	RBU	Amount = 6, in = 256, out = 256
10	PU	Max pooling
11	DBU	Amount = 10, k = 20, in = 256, out = 264
12	IBU	Type = 5b, in = 264, out = 1024
13	PU	Mean pooling
14	DBU	Amount = 4, k = 40, in = 1024, out = 192
15	DBU	Amount = 5, k = 12, in = 192, out = 188

Looking at the overall results in both datasets, we observe that all types of inception blocks and dynamic learning rates have huge potentials in the automatic CNN construction as they significantly increase the performance. Although adding the roulette wheel selection and ResNet block do not yield dramatic increase in accuracy, they still improve the performance and thus should be considered in the construction process.

The best CNN architecture found when performing experiments on SVHN dataset is given in Table 3.

3.5 Comparison to other approaches

In addition to demonstrating the effectiveness of our algorithm applied to image classification and analyse its performance together with AE-CNN, we compare our results to other leading automatic and semi-automatic CNN architecture design algorithms. For the comparison, we used CIFAR-100, which is a widely used image classification benchmark dataset for recognizing nature objects. CIFAR-100 has 100 classes consisting of 50,000 training and 10,000 test images in total. Sample CIFAR-100 images are shown in Fig. 11.

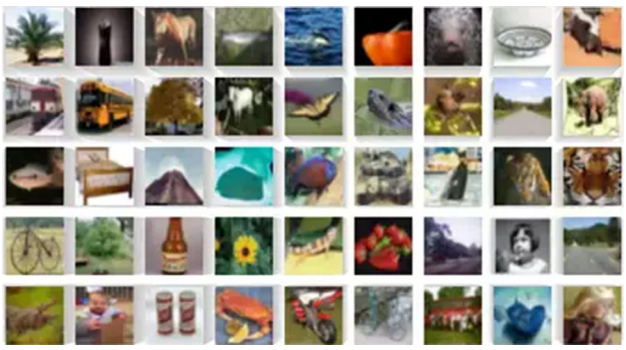


Fig. 11 Sample images from CIFAR-10

Similar to the SVHN experiments, the proposed framework consisted of all types of inception and ResNet blocks and roulette wheel selection with a dynamic learning rate. Our results along with those reported by AE-CNN [1], Large-scale Evolution [8], MetaQNN [9], Block-QNN-S [7], and Genetic CNN [6] are shown in Table 4. Moreover, we also present the number of parameters and GPU days of each method in this table. For the sake of completeness, we include the performances of two state-of-the-art hand-crafted models DenseNet [4], and FractalNet [23] in this evaluation. The '-' symbol in this table indicates that is no result publicly reported by the corresponding method.

As shown in Table 4, the proposed framework outperforms all the previous approaches utilizing completely automatic CNN construction for CIFAR-100. This sets a new state-of-the-art performance among completely automatic frameworks on CIFAR-100. Specifically, our method achieves a significantly higher classification accuracy than MetaQNN and Large-scale Evolution. Compared with AE-CNN, the proposed framework achieves a 0.11% higher performance. In additon, the proposed method has a better classification rate than the ones whose architectures are manually designed for this dataset. Specifically, our score is approximately 4% better than DenseNet and 1.5% better than FractalNet, which are the two best-performing hand-crafted architectures on CIFAR-100. Compared with semiautomatic models, our algorithm performs 8.3% better than Genetic CNN. The only approach that slightly outperforms our method is Block-QNN-S, which achieves only 0.09% better accuracy. However, one should take into consideration that domain expertise is still needed in the case of semi-automatic construction. On the other hand, the proposed framework is fully automatic precluding the need for such an expertise. Overall, the results demonstrate that the proposed approach not only performs better than AE-CNN but also achieves the best score among all methods utilizing fully automatic CNN construction and second best score among all types of image classification models including fully automatic, semi-automatic, and hand-crafted frameworks designed for CIFAR-100. Considering the number of parameters, DenseNet has 1.0 M, while the proposed method, AE-CNN and Block-QNN-S have 4.9 M, 5.4 M, 6.1 M parameters, respectively. This shows that the proposed approach has the least number of parameters in the completely automatic frameworks. Also, compared to Block-QNN-S, which currently has the best performance, the proposed framework reduces its number of parameters

Table 4 The classification performances of the state-of-the-art approaches and proposed framework on CIFAR-100, and their number of parameters

Method	Performance (%)	# of Parameters	GPU days	Type
Proposed framework	79.26	4.9 M	47	Completely automatic
AE-CNN [1]	79.15	5.4 M	36	Completely automatic
Large-scale Evolution [8]	77	40.4 M	2750	Completely automatic
MetaQNN [9]	72.86	–	100	Completely automatic
Block-QNN-S [7]	79.35	6.1 M	90	Semi-automatic
Genetic CNN [6]	70.95	–	17	Semi-automatic
DenseNet [4]	75.28	1.0 M	–	Hand-crafted
FractalNet [23]	77.7	38.6 M	–	Hand-crafted

by almost 20% with only 0.09% drop in classification accuracy. With regard to GPU days, the proposed framework consumes much less GPU days than that of Large-scale Evolution, MetaQNN, and Block-QNN-S while obtaining better classification scores. Genetic-CNN and AE-CNN have lower GPU days compared to our framework. However, Genetic-CNN is semi-automatic with only 70.95% classification performance. AE-CNN, on the other hand, requires less computational power, which is reasonable since the new additions to our framework, i.e., addition of inception and basic ResNet block structures, roulette wheel selection method that participated in the environmental selection and parent selection, and adopting dynamic learning rate, have yielded an improved performance with the cost of some extra computational complexity occured in GPU days. Overall, the proposed framework is still practical with improved scores obtained in image classification.

4 Conclusion and future work

Finding a suitable CNN model requires expertise both in the model construction and the problem domain. Although numerous existing promising CNN models exist in the literature, they may not be directly applicable to different domains since the parameters have been selected for one specific task and are achieved through the trial-and-error process. In this paper, we have proposed an evaluation-based completely automatic framework for constructing a CNN model applied to the domain of image classification.

The proposed framework considerably improves an existing state-of-the-art algorithm by including important additions. First, we have added inception blocks to extract features from different scales. Thus, instead of being restricted to a single filter size, we use multiple filter sizes, capturing the local structure of the input through multi-scale features. Second, we have added Roulette wheel selection in the environmental selection and parent selection processes to make the overall algorithm more flexible and powerful by offering alternative options. Third, we have included the basic ResNet block to construct better-performing individuals. Moreover, we have also dynamically adjusted the learning rate in the experiments to gain better accuracy.

Experimental evaluation of our framework including a comparison with the previous state-of-the-art approaches demonstrates that these additions both individually and altogether have high potential in the automatic CNN construction. Although we have applied our approach to image classification in this paper, our construction process is quite general can be applied to other problems. Thus, effectively utilizing our framework to be used in different domains is one of our future plans. The network construction and architecture search have more options, e.g., adding other layer and connection types. We will also study them in the future.

Acknowledgements This research has been funded under the Nazarbayev University faculty development grant "Forming Reliable Feature Correspondences and Distortion-free Graph Embedding with Deep Learning". Grant# 110119FD4530.

Funding M. Fatih Demirci has received research grants from Nazarbayev University (Kazakhstan) and TUBITAK (Turkey), and has worked at Nazarbayev University (Kazakhstan), TOBB University of Economics and Technology (Turkey), Utrecht University (Netherlands), and Drexel University (USA) before.

Declarations

Conflict of interest Togzhan Barakbayeva declares no conflicts of interest.

References

1. Sun Y, Xue B, Zhang M, Yen GG (2020) Completely automated CNN architecture design based on blocks. IEEE Trans Neural Netw Learn Syst 31(4):1242–1254
2. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: IEEE conference on computer vision and pattern recognition (CVPR) 2015, pp 1–9
3. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: IEEE conference on computer vision and pattern recognition (CVPR) 2016, pp 770–778

4. Huang G, Liu Z, Van Der Maaten L, Weinberger KQ (2017) Densely connected convolutional networks. In: IEEE conference on computer vision and pattern recognition (CVPR) 2017, pp 2261–2269
5. Zoph B, Le QV (2016) Neural architecture search with reinforcement learning. CoRR, vol. abs/1611.01578 [Online]. arXiv:1611.01578
6. Xie L, Yuille A (2017) Genetic CNN. In: IEEE international conference on computer vision (ICCV) 2017, pp 1388–1397
7. Zhong JYZ, Liu C-L (2018) Practical network blocks design with q-learning. In: Proceedings of the 2018 AAAI conference on artificial intelligence
8. Real E, Moore S, Selle A, Saxena S, Suematsu YL, Le QV, Kurakin A (2017) Large-scale evolution of image classifiers. CoRR vol. abs/1703.01041 [Online]. arXiv:1703.01041
9. Baker NNB, Gupta O, Raskar R (2017) Designing neural network architectures using reinforcement learning. In: Proceedings of the 2017 international conference on learning representations
10. Mnih V, Kavukcuoglu K, Silver D, Rusu A, Veness J, Bellemare M, Graves A, Riedmiller M, Fidjeland A, Ostrovski G et al (2015) Human-level control through deep reinforcement learning. Nature 5187(540):529–533. https://doi.org/10.1038/nature14236
11. Lu Z, Whalen I, Dhebar YD, Deb K, Goodman ED, Banzhaf W, Boddeti VN (2019) Multi-criterion evolutionary design of deep convolutional neural networks. CoRR vol. abs/1912.01369 [Online]. arXiv:1912.01369
12. Londt T, Gao X, Andreae P (2020) Evolving character-level densenet architectures using genetic programming. CoRR, vol. abs/2012.02327 [Online]. arXiv:2012.02327
13. Sun Y, Xue B, Zhang M, Yen GG (2018) Automatically designing CNN architectures using genetic algorithm for image classification. CoRR, vol. abs/1808.03818 [Online]. arXiv:1808.03818
14. Barakbayeva T, Demirci MF (2021) Fully automatic CNN design with inception blocks. In: Jiang X, Fujita H (eds) Thirteenth international conference on digital image processing (ICDIP 2021), vol 11878, International Society for Optics and Photonics. SPIE, pp 275–280. https://doi.org/10.1117/12.2601117
15. Miller BL, Miller BL, Goldberg DE, Goldberg DE (1995) Genetic algorithms, tournament selection, and the effects of noise. Complex Syst 9:193–212
16. Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: Teh YW, Titterington M (eds) Proceedings of the thirteenth international conference on artificial intelligence and statistics, series proceedings of machine learning research, vol 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp 249–256. http://proceedings.mlr.press/v9/glorot10a.html
17. Lecun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
18. Krizhevsky A (2009) Learning multiple layers of features from tiny images. Technical report
19. Deng J, Dong W, Socher R, Li L-J, Li K, Fei-Fei L (2009) Imagenet: a large-scale hierarchical image database. In: IEEE conference on computer vision and pattern recognition 2009, pp 248–255
20. Netzer Y, Wang T, Coates A, Bissacco A, Wu B, Ng A (2011) Reading digits in natural images with unsupervised feature learning. NIPS, 01
21. Zeiler M (2012) Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701
22. Luo H, Hanagud S (1997) Dynamic learning rate neural network training and composite structural damage detection. AIAA J 35(9):1522–1527
23. Larsson MMG, Shakhnarovich G (2016) Fractalnet: ultradeep neural networks without residuals. In: The 5th international conference on learning representations