GA-based Filter Selection for Representation in Convolutional Neural Networks

Junbong Kim[1], Minki Lee[1], Jongeun Choi[2], and Kisung Seo[1,*],
*Corresponding author

[1] Department of Electronic Engineering, Seokyeong University, Seoul, Korea
[2] School of Mechanical Engineering, Yonsei University, Seoul, Korea
tpeprrwq@skuniv.ac.kr, perpetmon@skuniv.ac.kr, jongeunchoi@yonsei.ac.kr, ksseo@skuniv.ac.kr

Abstract. One of the deep learning models, a convolutional neural network (CNN) has been very successful in a variety of computer vision tasks. Features of a CNN are automatically generated, however, they can be further optimized since they often require large scale parallel operations and there exist the possibility of overlapping redundant features. The aim of this paper is to use feature selection via evolutionary algorithms to remove the irrelevant deep features. This will minimize the computational complexity and the amount of overfitting while maintaining a good quality of representation. We demonstrate the improvement of the filter representation by performing experiments on three data sets of CIFAR10, metal surface defects, and variation of MNIST and by analyzing the classification performance as well as the variance of the filter.

Keywords: CNN · Feature Representation · Filter Optimization.

1 Introduction

Recent years, the deep learning technique such as convolutional neural networks (CNNs) is widely utilized for image recognition [8]. In general, a CNN [9] is known to be powerful in automatic feature extraction and it does not require manual feature engineering. The manual design of the feature representation is not adaptive to the data and is dependent on the specific design choices. Deep convolutional neural networks (DCNN) [7, 14, 16, 4] has a hierarchical structure that attempts to learn representations of input data with multiple levels of abstraction automatically. As the deep learning network grows deeper, the importance of feature representation in the computer vision is increasing [1]. CNN based features often rely on computationally expensive deep models, which requires high computational complexity for numerous applications. There is a strong need for new scalable and efficient approaches that can cope with this explosion of dimensionality [17]. Therefore, there is a growing need for feature descriptors that are fast to compute, memory efficient, and yet exhibiting good discriminability and robustness with respect to the input feature space.

In this paper, we propose an approach to refine the filter expression as well as computational expense. In particular, an evolutionary algorithm (e.g., a genetic algorithm (GA) [2]) will make a selection of the filters that are automatically designed by a CNN. Because a CNN has a large number of filters, instead of expressing the selected filters as genes, we try to improve the efficiency of genetic computation by expressing the filters as genes to be deleted in order to reduce the size of the genes. We represent genotypes and phenotypes differently and implement them through transformations. We use enough data to train the CNN network that consists of six layers (three convolutional layers and three pooling layers), and reconstruct the filters of each layer of the learned network through evolutionary computation. We then optimize the reconstruction by calculating the fitness using the test data. The reduction rate of the filters is set to 15.6% and 31.3% of the total filters, and the classification accuracy performance, (ACC) is used for performing evolutionary calculation of a generation. In addition, we observe the improvement of the performance by performing additional learning on the reduced CNN obtained from the final generation. We demonstrate the quality of the filter representation by performing experiments on CIFAR10, MNIST variations, and surface defect data and by analyzing the variance of the filters as well as the classification accuracy performance.

2 Literature survey

Convolutional Feature representations and/or selections are closely related to CNN network quantization and efficient CNN architectures. A number of efforts have been made in this direction, such as feature selection [1], compact binary features [11], CNN network compressions [3, 6], and optimization of CNN architectures [13, 18, 15]. Following works adopt fine-tuned strategies to further improve the discriminative power of the descriptors. First, various masking schemes are applied to select a representative subset of local convolutional features and remove a large number of redundant features from a feature map to achieve competitive retrieval performance [1]. Quantized CNN is proposed to simultaneously speed-up the computation and reduce the storage and memory overhead of CNN models. Both filter kernels in convolutional layers and weighting matrices in fully-connected layers are quantized, aiming at minimizing the estimation error of each layer's response [17].

Compression techniques to reduce the network size without affecting their accuracy are another popular approaches. Deep compression, a three stage pipeline: pruning, trained quantization and Huffman coding that work together to reduce the storage requirement of neural networks by 35 to 49x without affecting their accuracy, is introduced [3]. In a similar way, SqueezeNet [6] achieves AlexNet-level accuracy on ImageNet with 50x fewer parameters. Additionally, with model compression techniques, they are able to compress SqueezeNet to less than 0.5MB.

Designing CNN architectures automatically is also coupled with feature representations. Recently, some meta-heuristic algorithms, such as genetic algorithm (GA) and genetic programming (GP), have been used to optimize CNNs.

GA based optimization of CNN structures with the number and size of filters, connection between consecutive layers, and activation functions of each layer is introduced [13]. Long length of chromosome is adopted to cover above variables and showed not much improved performance results. The other GA based approach is a new encoding method to represent complex convolutional layers in a fixed-length binary string. Many popular network structures can be represented using the proposed encoding scheme. Examples include VGGNet [14], ResNet [4], and a modified variant of DenseNet [5]. However, a large fraction of network structures are still unexplored and the currently available main results are at the averaged levels and so they still require to be improved.

A Cartesian genetic programming (CGP) [12] based method is adopted for a CNN structure optimization with highly functional modules, such as convolutional blocks and tensor concatenation, as the node functions in CGP [15]. The CNN structure and connectivity represented by the CGP encoding method are optimized to maximize the validation accuracy. The proposed method can be used to automatically find the competitive CNN architecture compared with state-of-the-art models.

Evolutionary algorithm based approaches on CNN design have the emphasis that is on optimizing the external structure of the network, which is far from the expression of the filter.

3 Optimization of CNN representation

3.1 CNN and Representation Problems

One of the most related variables to the CNN representation are convolution filter weights. Filters that are automatically generated through learning include various types of filters and are responsible for feature extraction across multiple stages through the hierarchical forward network. Nevertheless, the filters used in previous studies may contain redundant filters. In this study, we select the filters that significantly contribute to the image recognition through the evolutionary computation of the fully learned CNN filters. This reduces the overall number of filters and refines the filter group. The methodology differs from the existing pruning technique [10] in that it chooses the appropriate filters for the situation (or environment) using evolutionary optimization, rather than simply choosing the importance of the filter based on weight values. In other words, our evolutionary approach is more focused on relational combination set of filters among convolutional layers rather than separated operations in each layer.

3.2 Evolutionary Algorithm

Evolutionary computation consists of gene expression, gene generation, fitness evaluation, selection, and genetic computation. The process of GA is as follows.

Firstly, each chromosome composed of strings is randomly generated. Then, the candidate solution obtained by interpreting each object is evaluated by a fitness function. Then, the entities to participate in the genetic operation are selected by the given selection method. Perform genetic calculations (mating, mutation) on selected entities. The entire process is repeated until the termination condition is satisfied. The mating operator in GA is performed by replacing a portion of the parent's string at an arbitrary point.

Unlike optimization by general evolutionary computation, the optimization of CNN requires a long computation time for repetition of generations because the evaluation process of an object includes the CNN test. That is, several hundred epochs of test process should be performed to evaluate one entity. In this study, we choose a method to reduce the computation volume by applying only the test procedure to the evaluation of the changed object first by separating the learning and the test. The overall GA structure for removing redundant filters in a CNN is summarized as shown in Algorithm 1.

Algorithm 1 Evolutionary optimization algorithm for CNN filters

```
 1: CNN training for train dataset
 2: $t \leftarrow 0$
 3: initialize $P(t)$                              ▷ individuals for filter selection
 4: procedure EVALUATE $P(t)$
 5:     CNN test for test dataset
 6:     calculate accuracy
 7: while not termination-condition do
 8:     $t \leftarrow t + 1$
 9:     select $P(t)$ from $P(t-1)$
10:     crossover and mutation $P(t)$
11:     procedure EVALUATE $P(t)$
12:         CNN test for test dataset
13:         calculate accuracy
```

3.3 Evolutionary Selection of CNN filters

Including all filters in the genes for selection is very inefficient for genetic calculations because small CNNs consist of dozens of filters per layer. Therefore, we set a certain percentage (15.6% and 31.3%) to be deleted from the whole filter, and then represent filters to be deleted as genes since their size is relatively small. This is illustrated in Figure 1. The gene number in genotype means the filter number to be deleted from the layer and the gray filters are shown to be deleted. Duplicate numbers are treated as a single one, and 0 represents a filter not to be deleted. The phenotype is remaining(or selected) filters which are used for evaluations.

In order to verify the proposed scheme, a filter selection procedure is performed on a basic network with a relatively simple CNN structure as shown in

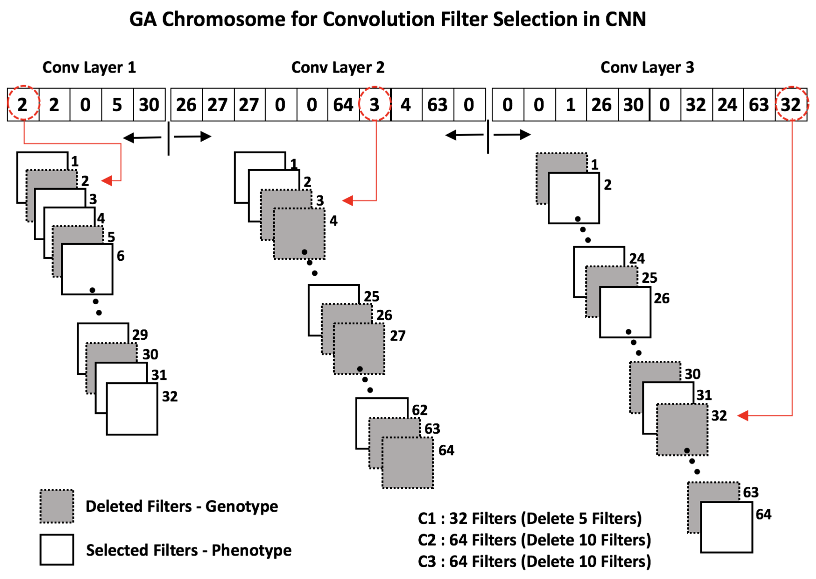
GA Chromosome for Convolution Filter Selection in CNN
Conv Layer 1
Conv Layer 2
Conv Layer 3
2 2 0 5 30 26 27 27 0 0 64 3 4 63 0 0 0 1 26 30 0 32 24 63 32
Deleted Filters - Genotype
Selected Filters - Phenotype
C1 : 32 Filters (Delete 5 Filters)
C2 : 64 Filters (Delete 10 Filters)
C3 : 64 Filters (Delete 10 Filters)

Fig. 1. GA for deleting redundant filters in a CNN a case of 15.6% deletion.

Table 1. Parameters of CNN model.

Layer	Filters	Filter size	Stride	Pad
Conv1	32	2×2×32	1	0
Pool1		2×2	2	0
Conv2	64	2×2×64	1	0
Pool2		2×2	2	0
Conv3	64	2×2×64	1	0
Pool3		2×2	2	0
Fc	128	1×1×128	1	0

Figure 2. We have trained enough a network consisting of 6 layers (3 convolutional layers, 3 pooling layers) by using CIFAR10, surface defect data, MNIST Variations. We then apply the GA evolutionary selection to the automatically generated filters of each convolution layer.

4 Experiments and Analysis

4.1 Experimental Environments

The experimental method is as follows. We use a CNN to express the filters that we learned about the data as genes. We then evolve the filters with GA to reduce the network size and improve the filters and performance similarly prior to reducing the network. Experiments were performed on the GTX-1080Ti

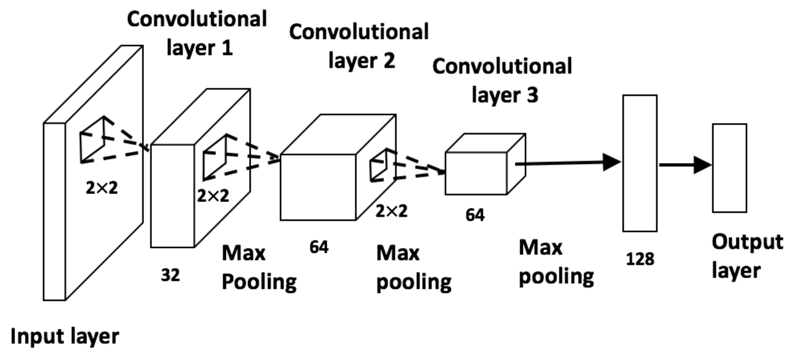

Fig. 2. CNN model structure

GPU for CIFAR10, metal exterior defect data, and modified MNIST data. The length of the binary string gene was 25 and 50, and the number of individuals was 50, and a total of 20 generations were performed for GA. The probability of crossing was set to 0.85, and the probability of mutation was set to 0.2. This is to facilitate generation of a new structure because of its long gene length. For CIFAR10, learning time per a candidate(or individual) solution took 3-4 minutes, and the total GA evolution process took about 48 hours. The learning time per a candidate for the metal exterior defects data took about 2-3 minutes, and the whole GA evolution process took about 36 hours. For MNIST variations, the learning time per a candidate took 3-4 minutes, and the entire GA evolution process took about 48 hours. The specific gene expression is as follows. The first group is the convolution C1, the second group is the three convolutions in C3, and the third group is the collection of filter numbers to be removed in C3. Gene length consists of gene 25 (5 + 10 + 10) and gene 50 (10 + 20 + 20) as shownbelow. The specific procedure is shown in Algorithm 2. The GA parameters are listed in Table 2.

Algorithm 2

```
1: Store network parameters after training a CNN
2: Input learned network parameters to GA
3: Select the filter on each candidate according to the GA performance
4: Fitness calculation (i.e., ACC by net CNN propagation using only reduced filters)
5: Re-train the CNN by lowering the learning rate of reduced networks by GA
```

4.2 Experimental Results

Results for Reduction Rate of 15.6%. Table 3 shows the results of experiments with a reduction ratio of 15.6% ((5 + 10 + 10) / (32+64+64)). For the

Table 2. GA parameters

Number of generations	20
Population size	50
Selection	Tournament (size=7)
Crossover	0.85
Mutation	0.2

Table 3. GA-based filtering reduction (15.6% reduction rate)

ACC	Original	GA Filter Selection (initial gen)	GA Filter Selection (final gen)	Retrain after GA Filter Selection (epoch 100)	Retrain after GA Filter Selection (epoch 500)
CIFAR-10	64.60%	54.57%	57.94%	62.90%	63.55% (98.3%)
Surface Defect	86.91%	84.00%	85.19%	86.40%	86.85% (99.9%)
MNIST Variations	94.13%	92.75%	92.78%	93.45%	93.48% (99.3%)

CIFAR-10, the ACC performance that was learned and tested with CNN shown in Figure 2 was 64.6%, and the result after randomly selecting and removing the filters of each layer within 15.6% was down to 54.57%.

Through the GA evolution calculation, the best performance after 20th generation is 57.94%. The performance of the re-learning of the evolved network with this reduced number of filters was very close to 98.3% of the original network ACC performance, from 500 epochs to ACC of 63.55%, while the network filter was reduced to 15.6% of the reduction rate. As for the surface defect data, the re-learning result after filter reduction is 86.85% ACC, which is equivalent to 99.9% of the original network performance of 86.91%. For the MNIST variation data, the ACC performance of the reduced network is 93.48%, 99.3% of the original ACC performance 94.13%. Our successful results show that the number of network filters is reduced by 15.6% while the performance is maintained at 98%-99% of the original deep learning network. In addition, the proposed approach is meaningful in the sense that it is a process of selecting filters that are efficient for representation rather than simple compression.

Results for Reduction Rate of 31.3%. The results of the second experiment with a reduction of 31.3%((10 + 20 + 20)/(32+64+64)). are shown in Table 4. For the CIFAR-10, the ACC performance that we learned and tested with CNN shown in Figure 2 is 64.6%, where the results are reduced to 47.33% after randomly selecting and removing the filters of each layer by 31.3%.

Through the GA evolution computation, the performance of the best after 20th generation is 53.74%. The ACC performance of the re-learning of the

Table 4. GA-based filtering reduction (31.3% reduction rate)

ACC	Original	GA Filter Selection (initial gen)	GA Filter Selection (final gen)	Retrain after GA Filter Selection (epoch 100)	Retrain after GA Filter Selection (epoch 500)
CIFAR-10	64.60%	47.33%	53.74%	59.65%	60.33% (93.4%)
Surface Defect	86.91%	73.02%	78.37%	85.73%	85.74% (98.7%)
MNIST Variations	94.13%	87.94%	90.49%	92.12%	92.55% (98.3%)

evolved network with the reduced number of filters was 60.33% at 500 epochs, which was 93.4% of the original network performance, which was somewhat lower than 15.6% at the reduction rate. Instead, the number of filters in the network has been reduced to 31.3%. As for the surface defect data, 85.74% of the re-learning result after filter reduction is almost equal to 98.7% of the original network performance 86.91%. For MNIST variation data, the performance of the reduced network is 92.55%, which is 98.3% of the original 94.13%. The successful results show that the number of network filters is reduced by 31.3% while the performance is maintained at 93%-98% of the original network performance.

Tables 5 and 6 show the comparisons of the spatial re-learning of CNNs and GA evolution (15.6% reduction) for CIFAR-10 and surface defect data, respectively, in each activation map of the first convolution layer. The large variance of the each pixel means that the recognition of the object in the activation map is more pronounced. For the 10 classes of CIFAR-10, we can clearly see that the variance values after GA evolution and re-learning, increased by an average of 12%-13%. In CIFAR-10, there is little difference between epoch 100 and epoch 500. Surface defects are classified into four types of defects: an average increase of 6.3% in the epoch 100 and an increase of 11% in the epoch 500. In summary, we observed the mechanism of our approach that is the GA-based filter reduction for complexity and overfitting reduction, has been well compensated by re-training and resulted in the increased spatial variance of the activation map in the remaining filters.

5 Conclusions

In this paper, we have proposed a procedure to remove the existing filters in order to reduce the computational complexity while maintaining the ACC performance by utilizing a carefully designed GA. In particular, we formulate our problem as selecting filters to be removed to reduce the complexity in GA. Our results of three different data sets (CIFAR10, metal surface defects, and variation of MNIST) show that we have successfully remove filters with weak representation

Table 5. Variance of the activation map on the first convolutional layer CIFAR-10

Average Deviations CIFAR-10 Class	Original	Retrain after GA Filter Selection (epoch 100)	Retrain after GA Filter Selection (epoch 500)
Airplane	816.6	919.2 (12.6%)	918.2 (12.4%)
Automobile	1166.3	1313.6 (12.6%)	1312.0 (12.5%)
Bird	717.8	809.2 (12.7%)	808.3 (12.6%)
Cat	1042.1	1178.5 (13%)	1176.8 (13%)
Deer	626.9	707.1 (12.8%)	707.1 (12.8%)
Dog	1090.4	1234.1 (13.1%)	1232.4 (13%)
Frog	713.0	805.4 (13%)	804.4 (12.8%)
Horse	1076.6	1217.4 (13%)	1215.8 (12.9%)
Ship	965.5	1088.7 (12.8%)	1087.3 (12.6%)
Truck	1225.5	1381.1 (12.7%)	1379.4 (12.6%)

Table 6. Variance of the activation map on the first convolutional layer Surface Defect

Average Deviations Surface Defect Class	Original	Retrain after GA Filter Selection (epoch 100)	Retrain after GA Filter Selection (epoch 500)
Stain	1555.7	1653.2 (6.3%)	1733.6 (11.4%)
Normal	380.8	404.8 (6.3%)	424.1 (11.4%)
Scratch	554.4	589.3 (6.3%)	616.6 (11.2%)
Stamped	510.3	542.3 (6.3%)	568.4 (11.4%)

and retain filters with strong representation. Our approach can be applied to different deep leaning architectures in order to suppress the detrimental effect of overfitting and computational complexity. Further study will aim at refinement of feature selection for better representation using advanced evolutionary algorithm and improvement of computation costs by integrating compression techniques.

6 Acknowledgement

This work was supported by National Research Foundation of Korea Grant funded by the Korea government (NRF-2016R1D1A1A09919650). This work was also funded by the Korea Meteorological Administration Research and Development Program under Grant KMIPA(KMI2018-06710).

References

1. Do, T.T., Hoang, T., Tan, D.K.L., Cheung, N.M.: From selective deep convolutional features to compact binary representations for image retrieval. arXiv preprint

arXiv:1802.02899 (2018)
2. Goldberg, D.: Genetic algorithms in search, optimization, and machine learning, addison-wesley, reading, ma, 1989. Google Scholar (2014)
3. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149 (2015)
4. 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)
5. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: CVPR. vol. 1, p. 3 (2017)
6. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. arXiv preprint arXiv:1602.07360 (2016)
7. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
8. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436 (2015)
9. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
10. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710 (2016)
11. Lin, K., Lu, J., Chen, C.S., Zhou, J.: Learning compact binary descriptors with unsupervised deep neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1183–1192 (2016)
12. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: European Conference on Genetic Programming. pp. 121–132. Springer (2000)
13. Rikhtegar, A., Pooyan, M., Manzuri-Shalmani, M.T.: Genetic algorithm-optimised structure of convolutional neural network for face recognition applications. IET Computer Vision **10**(6), 559–566 (2016)
14. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
15. 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 (2017)
16. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1–9 (2015)
17. Wu, J., Leng, C., Wang, Y., Hu, Q., Cheng, J.: Quantized convolutional neural networks for mobile devices. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4820–4828 (2016)
18. Xie, L., Yuille, A.L.: Genetic cnn. In: ICCV. pp. 1388–1397 (2017)