A Genetic Programming Approach to Integrate Multilayer CNN Features for Image Classification

Wei-Ta Chu and Hao-An Chu

National Chung Cheng University, Taiwan

Abstract. Fusing information extracted from multiple layers of a convolutional neural network has been proven effective in several domains. Common fusion techniques include feature concatenation and Fisher embedding. In this work, we propose to fuse multilayer information by genetic programming (GP). With the evolutionary strategy, we iteratively fuse multilayer information in a systematic manner. In the evaluation, we verify the effectiveness of discovered GP-based representations on three image classification datasets, and discuss characteristics of the GP process. This study is one of the few works to fuse multilayer information based on an evolutionary strategy. The reported preliminary results not only demonstrate the potential of the GP fusion scheme, but also inspire future study in several aspects.

Keywords: Genetic programming · Convolutional neural networks · Multilayer features · Image classification.

1 Introduction

Convolutional neural networks have been widely adopted in visual analysis, such as image classification and object detection. A common network structure includes a sequence of convolutional blocks followed by several fully-connected layers. Each convolutional block usually consists of one or more convolutional layers followed by a pooling layer. With vary-sized convolutional kernels and pooling, different convolutional layers extract visual features at various levels. A sequence of convolutional blocks thus can be viewed as a powerful feature extractor, and the extracted features are then fed to a classification network or a regression network to accomplish the targeted task.

Features extracted from the first few layers more likely describe basic geometric patterns, while features extracted from the last few layers more likely describe higher level object parts. Though high-level features may be preferable in recognizing visual semantics, in some domains low-level features and high-level features are better jointly considered to achieve better performance. For example, Li et al. [10] integrated features extracted from multiple CNN layers based on the Fisher encoding scheme, and demonstrated very promising performance in remote sensing scene classification. In [16], multilayer features from

CNNs were also jointly considered by the Fisher encoding scheme, and multiple CNNs were employed to extract features from multiple modalities to facilitate video classification.

To improve image classification performance, we introduce a novel way to integrate features extracted from multiple CNN layers based on *genetic programming* [8]. Both genetic programming (GP) and genetic algorithm (GA) [6] are evolutionary strategies motivated by biological inheritance and evolution. They both work by iteratively applying genetic transformations, such as crossover and mutation, to a population of individuals, in order to create better performing individuals in subsequent generations. Different from GA, individuals in GP iterations are not limited to fixed-length chromosomes. More complex data structures like tree or linked lists can be processed by GP. In our work, we take features extracted from CNN layers as the population of individuals, and attempt to find better representation by integrating individuals with GP.

Contributions of this paper are twofold:

- We introduce a multilayer feature fusion method based on genetic programming. To our knowledge, this would be the first work adopting genetic programming in integrating deep features extracted from different CNN layers.
- We demonstrate that this approach yields better image classification performance on three image benchmarks. Extensive discussions are provided to inspire future studies in several aspects.

2 Related Works

2.1 Multilayer Features

Many studies have been proposed to integrate features derived from multiple models. In this subsection, we simply review those integrating features derived from multiple layers of a single model, especially for image/video classification. To classify remote sensing scene images, i.e., aerial images, Li et al. [10] extracted visual features based on pre-trained deep convolutional neural networks. The models they used include AlexNet [9], CaffeNet [7], and variants of VGG [2][13]. From an input image, a series of images at different scales is produced by the Gaussian pyramid method. These images are fed to a CNN to get convolutional features, which are then concatenated and encoded as a Fisher vector. Basically, the idea of combining multilayer features in [10] is concatenation of convolutional features encoded by the Fisher kernel.

Yang et al. [16] extracted features from multiple layers and from multiple modalities to do video classification. Given a sequence of video frames, each frame is first fed to a CNN separately. The filter responses over time but corresponding to the same (pre-defined) spatial neighborhood are then encoded into Fisher vectors. In [16], Fisher vectors corresponding to different spatial locations are further weighted differently to improve the effectiveness. Basically, jointly considering feature maps from multiple layers in the representation of Fisher vectors is the idea of combining multilayer features.

In [15], a directed acyclic graph structure was proposed to integrate features extracted from different CNN layers. Feature maps from each layer are processed with pooling, normalization, and embedding, and the processed features from all layers are element-wisely added to form the final representation. This representation is then fed to a softmax layer to achieve scene image classification.

In most of these works, the operations to combine multiple layers are concatenation or addition. We will study how to automatically find more complex operations to fuse multilayer information based on GP.

2.2 Genetic Programming

Genetic programming is a branch of evolutionary computation that iteratively manipulates the population using crossover and mutation according to some fitness function, and attempts to find the individual that achieves the best performance. This strategy has been adopted in various domains. In this subsection, we simply review studies on the ones related to visual analysis.

Shao et al. [12] proposed to use GP in feature learning for image classification. Simple features like RGB colors and intensity values are extracted from images, which are viewed as the basic primitives. To generate better representations, a set of functions is defined to process these primitives, like Gaussian smooth, addition/subtraction, and max pooling. Basic primitives are processed by a sequence of pre-defined functions, and then integrated representations can be generated. Taking classification error rate as the fitness function, each integrated representation is evaluated, and better representations are selected to generate the next-generation individuals at the next iteration.

Liang et al. [11] formulated foreground-background image segmentation as a pixel classification problem. From each pixel, the Gabor features representing gradient information at a specific scale and a specific orientation are extracted. A binary classifier categorizing pixels as foreground or background is then constructed by the GP process.

Al-Sahaf et al. [1] learnt rotation-invariant texture image descriptors by GP. The statistical values like mean and max of a window centered by a pixel are viewed as the basic primitives, and a code is generated by a series of operations on the primitives to represent information derived from a pixel. The codes of pixels over the entire image are then quantized to be the image descriptor.

Inspired by the requests of designing a CNN structure for a specific task, Suganuma et al. [14] adopted the GP search strategy to find better CNN structures. Taking common components in CNNs, like convolutional block and max pooling, as the basic primitives, the proposed method automatically learns a series of CNN structures. This work would be one of the first studies linking CNN structure design with GP.

In our work, the basic idea is more like feature learning presented in [12]. However, the primitives are feature maps derived from pre-trained CNNs, and we want to verify that the automatically learnt representation can yield better performance in image classification. Different from [10] and [16], the operations to combine multilayer features are automatically learnt by GP.

3 Overview of Genetic Programming

The three components in GP are the terminal set, the function set, and the fitness function. The terminal set includes basic primitives to be manipulated. Each single primitive itself usually is a simple solution, but we want to learn to manipulate primitives to generate a better solution. For example, the terminal set of [12] simply contains RGB colors and intensity values of pixels, and in our work we take feature maps derived from a pre-defined CNN as the terminals.

The GP algorithm dynamically selects parts of the terminals, and sequentially processes or combines them to form an integrated presentation. A sequence of processing can be illustrated as a tree, as shown in Fig. 1(a) and Fig. 1(b). In Fig. 1(a), the terminal T_1 is first processed with F_1, and then is combined with T_2 by the function F_2. Then it is combined with T_3 by the function F_3 to form the final integrated representation. Notice that the terminal nodes and the function nodes are automatically selected by the GP algorithm, given the constraint of tree height. The same terminal nodes or function nodes may be selected multiple times in the same tree, as shown in Fig. 1(b).

At each iteration of the GP algorithm, a set of S integrated representations are generated. Each integrated representation can be described by a tree and can be evaluated by the fitness function. Parts of the representations that yield higher fitness values would be selected in the mating pool. The representations in the mating pool are potential *parents* that generate children representations by genetic operations like *crossover* and *mutation* at the next iteration.

Taking trees in Fig. 1(a) and Fig. 1(b) as the parents, Fig. 1(c) and Fig. 1(d) show two generated children by the crossover operation. Fig. 1(c) is generated from Fig. 1(a) by replacing the leaf node T_2 with a subtree from Fig. 1(b). Conversely, Fig. 1(d) is generated from Fig. 1(b) by replacing a subtree by the leaf node T_2 of Fig. 1(a). We intentionally draw subtrees from Fig. 1(a) and Fig. 1(b) in blue and green, respectively, to clarify the idea. To conduct mutation, a tree is randomly generated as the basic element, as shown in Fig. 1(e). A mutation result from Fig. 1(a) is generated by replacing a subtree rooted at F_2 by the one shown in Fig. 1(e), yielding the illustration in Fig. 1(f).

We keep combining selected parents to generate children representation until the number of children is the same as that of the previous population, i.e., the number S mentioned above. Fitness values of the representations in the newly-generated population are then evaluated, and then better ones are selected in the mating pool for generating the next population. The same process keeps iterating until some stop criterion meets. Finally, the best-so-far representation is picked as the final representation.

4 GP-based Combination

This section provides details of how we employ GP in integrating features extracted from different CNN layers. Although the proposed integration method is not limited to any specific CNN models, we take the VGG-S model [2] as the main example in this section.

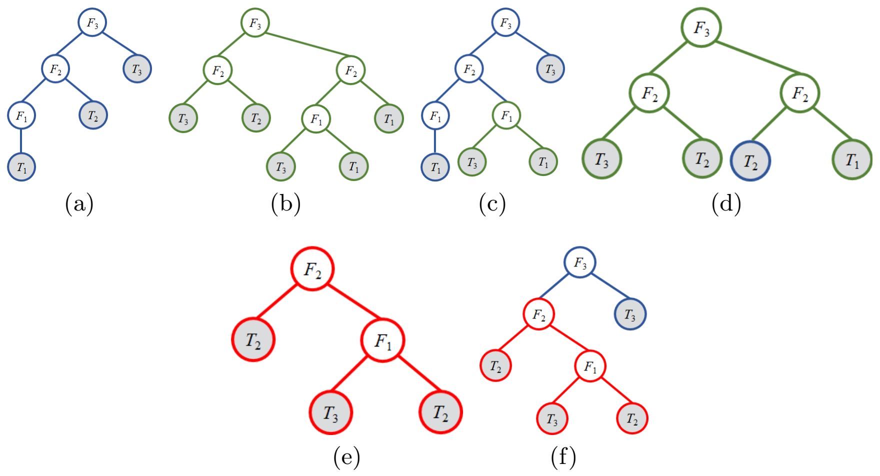

F_3 F_2 T_3 F_1 T_2 T_1
(a)
F_3 F_2 F_2 T_3 T_2 F_1 T_1 T_3 T_1
(b)
F_3 F_2 T_3 F_1 F_1 T_1 T_3 T_1
(c)
F_3 F_2 F_2 T_3 T_2 T_2 T_1
(d)
F_2 T_2 F_1 T_3 T_2
(e)
F_3 F_2 T_3 T_2 F_1 T_3 T_2
(f)

Fig. 1. Illustrations of the GP processes. (a)(b) Trees describing sequences of processes to generate integrated representations. (c)(d) Two children representations generated by applying the crossover operation on trees in (a) and (b). (e) The randomly-generated tree to conduct the mutation operation. (f) A tree generated from (a) with the mutation operation shown in (e).

Assume that there are N_i feature maps of size $M_i \times M_i$ from the ith CNN layer, and there are N_j feature maps of size $M_j \times M_j$ from the jth CNN layer. For each layer, we flatten the feature maps into vectors, and thus the ith terminal T_i is represented as N_i vectors of dimensionality $M_i \times M_i$.

When the two terminals T_i and T_j are to be combined with element-wise addition, for example, we would first ensure that vectors from two terminals are comparable. Assume that $M_i > M_j$ and $N_i > N_j$, we first reduce dimensionality (M_i) of vectors in T_i into M_j by the principal component analysis method, and then concatenate all of them to form a $(N_i \times M_j \times M_j)$-dimensional vector $\boldsymbol{t}_i$ to represent T_i. For the terminal T_2, we first concatenate all vectors and get a $(N_j \times M_j \times M_j)$-dimensional vector $\boldsymbol{t}_j$. Two different addition operations are designed to make $\boldsymbol{t}_j$ compatible with $\boldsymbol{t}_i$, which are denoted as AddPad and AddTrim, respectively. If $\boldsymbol{t}_j$ and $\boldsymbol{t}_i$ are combined with the AddPad operation, we pad zeros at the end of $\boldsymbol{t}_j$ such that the dimensionality of $\boldsymbol{t}_j$ is increased to $(N_i \times M_j \times M_j)$. This dimensionality transformation strategy follows the setting mentioned in [14]. If $\boldsymbol{t}_j$ and $\boldsymbol{t}_i$ are combined with the AddTrim operation, appropriate numbers of items at the end of $\boldsymbol{t}_i$ are trimmed, such that the dimensionality of of $\boldsymbol{t}_i$ is decreased to $(N_j \times M_j \times M_j)$. Similarly, the element-wise subtraction, multiplication, and division operations all have the padded version and trimmed version. The GP process determines whether the padded version or the trimmed version is more effective automatically.

Pre-defined operations to combine two nodes are element-wise addition, subtraction, multiplication, division, and taking maximum/minimum/absolute values. Another operation is concatenation, which is also one of the most common operations used in previous works [10][3]. Taking the terminals T_i and T_j as the example, we concatenate the N_i $(M_i \times M_i)$-dim vectors with the N_j $(M_j \times M_j)$-dim vectors, and finally form a $(N_i M_i M_i N_j M_j M_j)$-dimensional vector.

A sequence of processes on terminal nodes and internal nodes can be described as a tree like Fig. 1(a), and the root node represents the final integrated descriptor. To evaluate goodness of the final representation, we define the classification error rate as the fitness value. For an evaluation dataset like the Caltech-101 image collection [4], we divide it into three parts: training set, validation set, and test set. From the training set, we feed images into the VGG-S model and extract feature maps from CNN layers. Integrated representations $\{\boldsymbol{f}_1, \boldsymbol{f}_2, ..., \boldsymbol{f}_N\}$ are generated by N sequences of processes, each of which is described by a tree. Based on the ith integrated representations $\{\boldsymbol{f}_i\}$ extracted from the training images, we construct a multi-class classifier based on a support vector machine. The ith integrated representations extracted from the validation set are then used to test the classifier, and then the classification error rate can be calculated. The training set and the validation set are in fact shuffled based on the five-fold cross validation scheme. Overall, the average classification error rate after five runs of training and validation is viewed as the fitness value, which is the clue for selecting better integrated representations into the mating pool.

One important parameter to generate an integrated representation is the height of a tree. A tree of larger height means more processes are involved in generating the integrated representation. Conceptually, if higher trees are allowed, larger search space is allowed to find better representations, but the GP algorithm is more computationally expensive. We thus dynamically increase the heights of trees in the GP algorithm. Let H_{max} denote the maximum height allowed to generate a tree, and H_{cur} denote the height of the highest tree at the current iteration. Starting from the parents selected from the tth iteration, if a children tree $\mathcal{T}_c$ is generated by crossover or mutation for the $(t+1)$th iteration, and its height is H_c, then it is filtered by the following process.

- If $H_c \leq H_{cur}$, take the tree $\mathcal{T}_c$ into consideration at the $(t+1)$th iteration. The representation described by $\mathcal{T}_c$ will be evaluated.
- If $H_c > H_{cur}$ and $H_c \leq H_{max}$, the representation described by $\mathcal{T}_c$ is evaluated. If the solution described by $\mathcal{T}_c$ is better than all existing solutions, we set H_{cur} as H_c. Otherwise, we discard the tree $\mathcal{T}_c$.
- If $H_c > H_{max}$, discard the tree $\mathcal{T}_c$.

The idea of the aforementioned filtering process is that we increase the search space only when better solutions can be obtained by higher trees. This guarantees a reasonable computational cost when we conduct the GP process.

5 Experiments

5.1 Evaluation Settings and Datasets

We conduct experiments on the Caltech-101 dataset [4], the Caltech-256 dataset [5], and the Stanford-40 action dataset [17]. The Caltech-101 dataset consists of

Table 1. Configurations of the baseline models [2].

Arch.	conv1	conv2	conv3	conv4	conv5	full6	full7	full8
VGG-S	$96 \times 7 \times 7$	$256 \times 5 \times 5$	$512 \times 3 \times 3$	$512 \times 3 \times 3$	$512 \times 3 \times 3$	4096	4096	1000
	st. 2, pad 0	st. 1, pad 1	st. 1, pad 1	st. 1, pad 1	st. 1, pad 1	dropout	dropout	softmax
	LRN, x3 pool	x2 pool	–	–	x3 pool			

101 widely varied object categories, and each category contains between 45 to 400 images. The Caltech-256 dataset consists of 256 object categories, and each category contains between 80 to 827 images. The Stanford-40 dataset contains 9,532 images in total with 180-300 images per action class. For the Caltech-101 dataset, we follow the experimental protocol mentioned in [12]. The first 20 images from each category are selected as the training data, the following 15 images from each category are taken as the validation data, and the remaining is taken as the testing data. For the Caltech-256 dataset, the first 45 images from each category are selected as the training data, the following 15 images from each category are taken as the validation data, and the remaining is taken as the testing data. For the Stanford-40 dataset, the first 80 images from each action class are selected as the training data, the following 20 images from each class are the validation data, and the remaining is the testing data.

To generate GP-representations, we search for good GP-representations based on information extracted by the VGG-S model [2]. Table 1 shows the configurations of the VGG-S model. The first sub-row of each cell denotes the number of convolution filters and their receptive field as "number $\times$ size $\times$ size". In the second sub-row, the notation "st" stands for the convolution stride, and "pad" stands for spatial padding. In the third sub-row, LRN stands for Local Response Normalization [9], followed by the max-pooling downsampling factor. For the fully-connected layers, we specify the number of nodes. The layers full6 and full7 are regularized using dropout, and the output of the full8 layer is activated by a softmax function. Activation function of all layers is ReLU.

In the GP process, we take responses of all layers (including convolutional layers and fully-connected layers) as the input, and iteratively combine them with predefined operations. The GP process runs for 20 generations, and at each generation, 100 trees are built and evaluated. After 20 generations, the best-so-far representation is picked as the final GP representation, which is then used to construct an SVM classifier to do image classification. The mean class accuracy is reported in the following experiments.

5.2 Performance on the Caltech-101 Dataset

We first evaluate the GP representation determined based on the VGG-S model on the Caltech-101 dataset. Table 2 shows mean class accuracies obtained based on various image representations. The first sub-table lists performance yielded by handcrafted features, including HOG, SIFT, LBP, Texton histogram, and Centrist. At most 75% accuracy can be achieved by handcrafted features.

The second sub-table shows performance obtained based on three types of learning features. The DBN item stands for a deep brief network consisting of

Table 2. The mean class accuracies obtained by handcrafted features, learned features, and the proposed GP representation, based on the Caltech-101 dataset.

	Handcrafted features				
	HOG	SIFT	LBP	Texton his.	CENTRIST
Accuracy	60.7	63.3	58.3	73.4	75.1
	Learnt features				
	DBN	CNN	MOGP [12]	VGG-S (w/o fine-tuning)	VGG-S (with fine-tuning)
Accuracy	78.9	75.8	80.3	72.4	87.8
	GP representation				
Accuracy	**90.4**				

three layers with 500, 500, and 2000 nodes, respectively. Responses of the final layer are taken as the image representation, and an SVM classifier is constructed to do image classification. The CNN item stands for a convolutional neural network consisting of five convolutional layers. Responses of the final convolutional layer are taken as the image representation to construct an SVM classifier. These two learnt features yield performance better than handcrafted features. The MOGP (Multi-Objective Genetic Programming) [12] is a GP-based method that integrates simple handcrafted features, i.e., pixel's RGB values and intensity, by genetic programming. We see that over 80% accuracy can be achieved, even better than the DBN or CNN features, though only very simple features are used as the foundation for feature fusion.

Our main idea is that we would like to further improve performance by fusing multilayer learnt features by GP. In this experiment, the baseline learnt features are from the output of the last layer of the VGG-S model. The last two items in the second sub-table show performance yielded by the baseline features. Without fine-tuning, the VGG-S features do not work well. By fine-tuning with the Caltech-101 dataset, the classification accuracy is largely boosted to 87.8%, conforming to the trend shown in [2]. The third sub-table shows that the determined GP representation yields the best performance, i.e., 90.4% mean class accuracy. This verifies that the proposed GP method can effectively integrate multilayer information and boost performance.

Fig. 2(a) shows that how the classification error rate gradually decreases as the number of iteration increases. This shows appropriately fusing multilayer features by GP really yields better classification performance. Fig. 2(b) shows that, as the number of GP iteration increases, how the number of tree nodes and the number of tree height change. From the orange curve, we see that generally the number of nodes increases as the number of iteration increases. This means the GP process tends to fuse more features as the evolution proceeds. From the red curve, we see that trees grow higher as the evolution proceeds. The height of the tree yielding the best performance in Table 2 is three.

Fig. 3(a) shows the final fusion result that yields the best performance. As shown in the tree, information extracted by full7 and full6 is first combined by the MaxPad operation (taking element-wise maximum after padding) to generate the internal representation X1. Another subtree shows that full8 and full6 are also combined by the MaxPad operation to generate X2. The international representations X1 and X2 are then concatenated to form the final GP representation X3. Notice that information from the same layer may be adopted multiple times to form the GP representation, e.g., full6 in this case. The ways to combine multilayer information and the information to be fused are all determined by the genetic programming automatically.

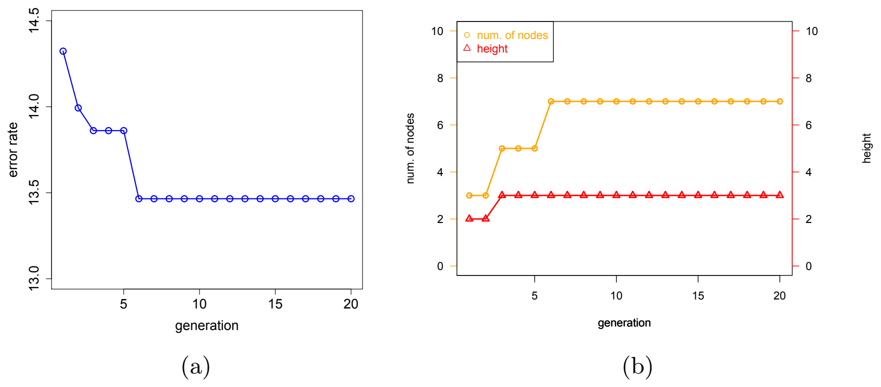


Fig. 2. (a) The evolutions of error rate as the number of generation increases. (b) The evolutions of number of tree nodes and number of tree height as the number of generation increases.

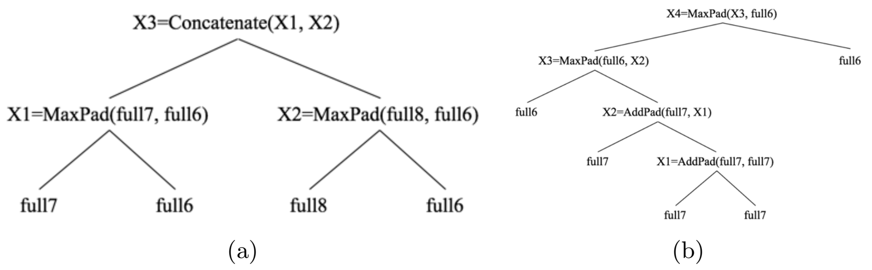


Fig. 3. (a) The tree representing the final fusion result yielding the best performance in Table 2. (b) The tree representing the final fusion result yielding the best performance in Table 3.

5.3 Performance on Other Datasets

We evaluate the GP representation determined based on the VGG-S model on the Caltech-256 dataset. Table 3 shows mean class accuracies obtained by the

Table 3. The mean class accuracies obtained by the baseline and the proposed GP representation, based on the Caltech-256 dataset.

	VGG-S (with fine-tuning)	GP representation
Accuracy	70.32	71.78

baseline model and the GP representation. Notice that the setting for fine tuning in Table 3 is different from that used in [2]. In [2], 60 images from each class were selected for fine tuning, while in our work, only 45 images are selected for fine tuning, and the remaining 15 images are used for validation in the GP process due to hardware limit of our implementation. With such setting, Table 3 again shows the superiority of the GP representation. The performance gain is around 1%. Fig. 3(b) shows how the GP representation is constructed. The internal node X2 is actually the result of multiplying full7 by three. This simple process is done by adding full7 three times. The internal node X3 is obtained by finding the maximum of X2 and full6 element-wisely. Finally, the final GP representation X4 is obtained by finding the maximum of X3 and full6 element-wisely.

We also evaluate the determined GP representation on the Stanford-40 action dataset. Table 4 shows mean class accuracies obtained by the baseline model and the GP representation. We see that, based on the GP representation, around 2% performance improvement can be obtained.

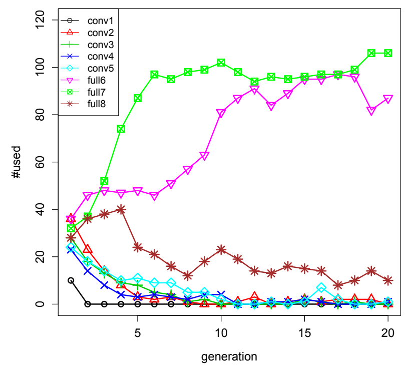
Table 4. The mean class accuracies obtained by the baseline and the proposed GP representation, based on the Stanford-40 dataset.

	VGG-S (with fine-tuning)	GP representation
Accuracy	59.76	61.80

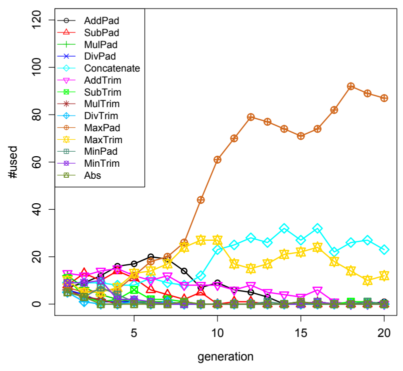
5.4 Discussion

We make discussion based on the experiments on the Caltech-101 dataset. Fig. 4(a) shows the number of times different layers' information used in fusion as the evolution proceeds. We count how many times a layer's output is used at each iteration. As can be seen in Fig. 4(a), the outputs of full6, full7, and full8 are much frequently utilized in combination. Interestingly, this trend conforms to previous studies that show deeper layers of a neural network extract high-level semantics and are usually used to do image classification and many other tasks.

Fig. 4(b) shows the number of times different operations used to combine multiplayer information as the GP process proceeds. Overall, the operations of MaxPad, MaxTrim, and Concatenation are more frequently utilized to fuse multilayer information. We think this characteristic may pave a way to improve the commonly-used neural networks, but the reason why these operations are utilized more frequently still needs further investigation in the future.



(a)



(b)

Fig. 4. (a) The number of times different layers' information used in fusion as the number of generation increases. (b) The number of times different operations used to combine multilayer information as the number of generation increases.

6 Conclusion

We have presented a fusion method based on genetic programming to integrate information extracted from multiple layers of a neural network. We verify the effectiveness of the proposed GP method by showing that the automatically determined representation yields better performance than the output of the best single layer. In addition, we also discuss the characteristics of the trees embodying the determined GP representation, and the trends of utilized operations and layers as the evolution proceeds. A few directions can be investigated in the future, such as conducting evaluation based on a large-scale collection, and considering more basic operations in the GP process.

Acknowledgement. This work was partially supported by the Ministry of Science and Technology under the grant 107-2221-E-194-038-MY2 and 107-2218-E-002-054, and the Advanced Institute of Manufacturing with High-tech Innovations (AIM-HI) from The Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE) in Taiwan.

References

1. Al-Sahaf, H., Al-Sahaf, A., Xue, B., Johnston, M., Zhang, M.: Automatically evolving rotation-invariant texture image descriptors by genetic programming. IEEE Transactions on Evolutionary Computation **21**(1), 83–101 (2017)
2. Chatfield, K., Simonyan, K., Vedaldi, A., Zisserman, A.: Return of the devil in the details: Delving deep into convolutional networks. In: Proceedings of British Machine Vision Conference (2014)

3. Dosovitskiy, A., Fischer, P., Ilg, E., Hausser, P., Hazirbas, C., Golkov, V., van der Smagt, P., Cremers, D., Brox, T.: Flownet: Learning optical flow with convolutional networks. In: Proceedings of IEEE International Conference on Computer Vision (2015)
4. Fei-Fei, L., Fergus, R., Perona, P.: Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In: Proceedings of CVPR Workshop of Generative Model Based Vision (2004)
5. Griffin, G., Holub, A., Perona, P.: Caltech-256 object category dataset. Tech. rep., California Institute of Technology (2007)
6. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. MIT Press (1992)
7. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. In: Proceedings of ACM International Conference on Multimedia. pp. 675–678 (2014)
8. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
9. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Proceedings of International Conference on Neural Information Processing Systems. pp. 1097–1105 (2012)
10. Li, E., Xia, J., Du, P., Lin, C., Samat, A.: Integrating multilayer features of convolutional neural networks for remote sensing scene classification. IEEE Transactions on Geoscience and Remote Sensing **55**(10), 5653–5665 (2017)
11. Liang, Y., Zhang, M., Browne, W.N.: Figure-ground image segmentation using genetic programming and feature selection. In: Proceedings of IEEE Congress on Evolutionary Computation (2016)
12. Shao, L., Liu, L., Li, X.: Feature learning for image classification via multiobjective genetic programming. IEEE Transactions on Neural Networks and Learning Systems **25**(7), 1359–1371 (2014)
13. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: Proceedings of International Conference on Learning Representation (2015)
14. 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 (2017)
15. Yang, S., Ramanan, D.: Multi-scale recognition with dag-cnns. In: Proceedings of IEEE International Conference on Computer Vision (2015)
16. Yang, X., Molchanov, P., Kautz, J.: Multilayer and multimodal fusion of deep neural networks for video classification. In: Proceedings of ACM Multimedia Conference. pp. 978–987 (2016)
17. Yao, B., Jiang, X., Khosla, A., Lin, A.L., Guibas, L., Fei-Fei, L.: Human action recognition by learning bases of action attributes and parts. In: Proceedings of IEEE International Conference on Computer Vision (2011)