An Efficient Convolutional Neural Network Architecture Search Using Multi-Layered Population Structure

Kazuki Yabuuchi [1,*] and Naoki Mori [2]

[1]Graduate School of Informatics, Osaka Metropolitan University, Osaka, Japan
[2]School of Engineering, Graduate School of Informatics, Osaka Metropolitan University, Osaka, Japan
Email: yabuuchi.kazu0105@gmail.com (K.Y.); mnao@omu.ac.jp (N.M.)
[*]Corresponding author
Manuscript received December 23, 2024; revised April 29, 2025; accepted October 14, 2025; published January 9, 2026

***Abstract*—The design of optimal Convolutional Neural Network (CNN) architectures has become increasingly complex as networks have grown deeper and more diverse. Neural Architecture Search (NAS) has emerged as a powerful technique to automate the discovery of high-performance architectures, thereby reducing reliance on manual expertise. In this work, we propose a novel approach that integrates NAS with a Multi-Layered Population Structure (MLPS), an effective evolutionary scheme originally developed for Genetic Programming (GP). We term this method Multi-Layered Population Structure Neural Architecture Search (MLPS-NAS). Our approach leverages the hierarchical, pyramid-like population management of MLPS to maintain a diverse set of partial solutions, while systematically exploring the vast search space of CNN architectures. A key feature of MLPS-NAS is its ability to construct complex architectures by repeatedly incorporating effective building blocks. To significantly accelerate the search process, we introduce a weight inheritance mechanism between generations, which drastically reduces the computational cost of training new candidate architectures. Experimental results on a real-world image classification task demonstrate that MLPS-NAS achieves accuracy comparable to an established method, CNN Architecture Design Using Cartesian Genetic Programming (CGP-CNN), while substantially reducing the required search time. This synergy between NAS and MLPS offers a promising direction for the efficient, automated design of high-performance CNNs.**

***Keywords*—neural architecture search, convolutional neural network, evolutionary computation, multi-layered population structure**

I. Introduction

The Convolutional Neural Network (CNN) has revolutionized computer image recognition [1] and is widely applied to tasks such as object detection and medical image analysis [2, 3]. Early architectures, such as Neocognitron [4] and LeNet [5], were relatively simple. In 2012, AlexNet [1] with a deeper architecture demonstrated high performance, leading to increased attention to deep learning. Since then, GoogLeNet [6], which incorporates effective substructures, and ResNet [7], which stacks residual blocks to facilitate training, have been developed. While these models have demonstrated superior performance, designing optimal architectures manually requires significant expertise and trial-and-error.

Therefore, there has been a growing interest in the field of Neural Architecture Search (NAS), which seeks to automate the design of CNN architectures, reducing human effort and improving the likelihood of discovering high-performance models. However, NAS methods often require significant computational resources due to the training and evaluation of many candidate architectures.

Various approaches have been proposed as NAS methods for CNN architectures, including those utilizing reinforcement learning [8, 9], gradient descent [10], and evolutionary computation [11].

NAS methods based on evolutionary computation have been studied since before 2000 [12]. Recent studies have proposed methods that reduce the number of evaluation epochs to shorten search time, based on the assumption that high-performing architectures achieve relatively high performance even in early epochs [13, 14]. However, there has been insufficient analysis of the correlation between performance in early epochs and performance after sufficient training.

In parallel with the development of NAS, one of the authors previously introduced a Multi-Layered Population Structure (MLPS) within the context of Genetic Programming (GP) [15], termed MLPS-GP [16]. MLPS-GP employs a hierarchical, pyramid-like population structure inspired by the Parameter-less Population Pyramid (P3) [17], making it possible to maintain diverse and high-quality partial solutions. By organizing individuals across multiple layers, MLPS-GP efficiently integrates exploration and exploitation, improving the evolutionary search process.

In this paper, we combine these two research fields, NAS and MLPS, into a unified framework: MLPS-NAS. By adapting the MLPS approach from GP to NAS, we introduce a method that not only inherits the benefits of MLPS in population management and diversity maintenance but also synergistically enhances NAS through the simplified reuse of effective structures. MLPS-NAS enables repetitive architectural patterns to emerge naturally, improving efficiency and performance. Additionally, the method reduces computational overhead via weight inheritance, significantly shortening the search time.

To evaluate the effectiveness of our proposed method, we conducted experiments for the exploration of the optimal CNN architecture for predicting the works of four-panel manga, a realistic task using the Manga109 dataset [18, 19]. As a result, our method demonstrated high performance comparable to an existing genetic programming method.

II. Related Work

A. Genetic Programming with Multi-Layered Population Structure (MLPS-GP)

MLPS-GP [16] introduced a hierarchical population inspired by P3 [17]. Initially proposed for GP, MLPS organizes individuals into multiple levels, each acting like a distinct population. This method retains and utilizes both

diverse solutions and high-performance partial solutions, facilitating efficient exploration of the search space. The MLPS approach is crucial for our current work, as we adapt its management advantages to NAS to form MLPS-NAS. Combining NAS with MLPS offers a new dynamic search, potentially outperforming existing evolutionary NAS approaches.

B. CNN Architecture Design Using Cartesian Genetic Programming (CGP-CNN)

CGP-CNN [20], based on Cartesian Genetic Programming (CGP) [21], optimizes CNN architectures represented as directed acyclic graphs composed of nodes arranged in a two-dimensional matrix. CGP-CNN focuses on evolving a single individual through gradual changes, while MLPS-GP utilizes the hierarchical population structure to combine multiple individuals. CGP-CNN has shown that evolutionary methods can automatically discover architectures comparable to established high-performance CNN models and existing NAS methods. However, reducing search time and enhancing the independent evaluation and utilization of substructures remain challenges.

III. Proposed Method: MLPS-NAS

We propose MLPS-NAS, a novel NAS approach that integrates the Multi-Layered Population Structure (MLPS) into evolutionary CNN architecture search.

MLPS-NAS preserves diversity, enabling the retention of partial solutions. This naturally supports the re-emergence and repetition of effective building blocks within CNN architectures. It is well known that CNN performance improves by repeatedly using or combining specific substructures [6, 7]. In addition, NAS methods typically require significant training time as architecture depth and dataset size grow. Our method addresses this by reusing previously trained weights, significantly reducing evaluation time.

The algorithm for the proposed method is shown in Algorithm 1. Sections A to D explain the individual representation and evolutionary operations.

Algorithm 1: MLPS-NAS begins with population initialization and continues until the maximum number of generations. Crossover and population update are counted as one generation

Require: An empty MLPS, $g_{\max}$: the maximum number of generations, $n_{\text{module search}}$: the number of Module Search

Ensure: An optimized CNN architecture with its model weights

1: $g \leftarrow 0$; (g: generation)
2: Initialize MLPS; (Population Initialization, Section *A*)
3: **for** $i = 1$ to $n_{\text{module search}}$ **do**
4: Perform Module Search; (Module Search, Section *B*)
5: **end for**
6: **while** $g < g_{\max}$ **do**
7: $g \leftarrow g + 1$;
8: Apply Crossover to MLPS; (Crossover, Section *C*)
9: Update individuals in MLPS; (Population Update, Section *D*)
10: **end while**

A. Individual Representation and Evolutionary Operations

In MLPS-NAS, CNN architectures are treated as individuals, with their genotypes encoded by directed acyclic graphs of layers and blocks (e.g., convolution, pooling), excluding invalid nodes such as pooling on 1 × 1 feature maps. Each node represents a layer or block, and each edge represents data flow. Two example genotypes are shown in Figs. 1 and 2. The fitness is measured by the highest validation accuracy up to the n_{f}-th epoch of training. If a genotype is not a valid CNN, the fitness is set to 0.0. Individuals retain their model weights after n_{f} epochs.

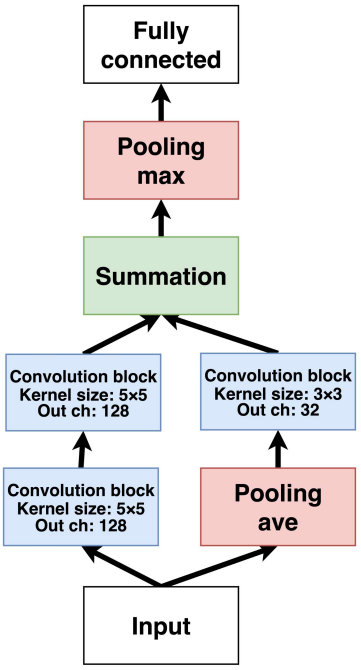

Fig. 1. An example genotype (1). The individual size s is 4.

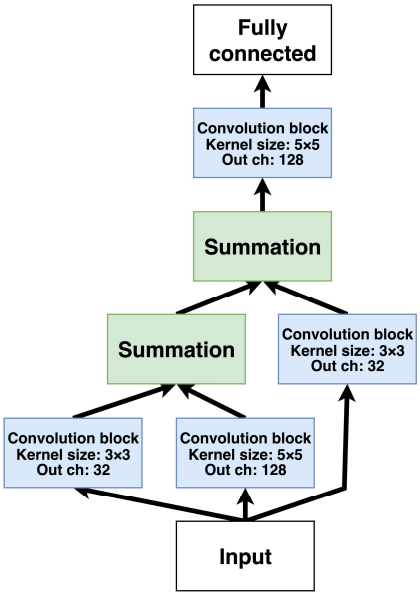

Fig. 2. An example genotype (2). The individual size s is 4.

Fig. 3 illustrates an overview of the MLPS. MLPS-NAS ensures that promising partial solutions are propagated upward while weaker candidates are filtered out. Individual size s is defined as the maximum number of nodes through which data flows from the input layer to the fully connected layer in the genotype. Individual size is an indicator used for diversity evaluation. Importantly, specific nodes created by crossover are not counted. The individual is added to the s-th level of the MLPS. At the start of the algorithm, some individuals are added to the MLPS as population initialization. In population initialization, we assume the addition of small individuals and well-known substructures. The aim is to incorporate them as substructures into potential optimal architectures.

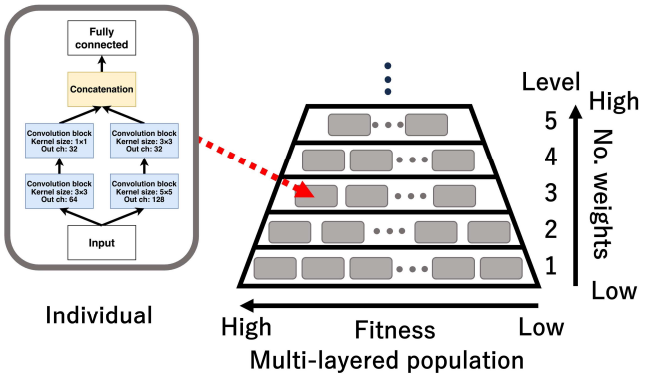

Fig. 3. MLPS organizes individuals into a pyramid-shaped hierarchy of populations. Within each level, individuals are sorted in descending order of fitness. Higher levels tend to include individuals with more model weights.

B. Module Search

In module search, CNNs with small architectures explored by a NAS method are converted into MLPS-NAS individuals and added to the MLPS. These modules serve as building blocks, enabling the more rapid formation of complex and better architectures. If the same architectures appear, only the one with the highest fitness is retained, maintaining diversity and preventing stagnation of exploration. In the experiment, CGP-CNN was employed as the module search method.

C. Crossover

Crossover is applied to two individuals selected from different levels or the same level of the population based on specific rules. The key innovation is weight inheritance: MLPS-NAS reuses trained weights (e.g., convolution weights, batch normalization weights) from parents, significantly reducing training time for offspring. This operation directly addresses the computational burden of NAS, as the model weights do not require re-training from scratch. In addition, this approach enables a new individual to naturally incorporate repetitive, effective substructures.

As shown in Fig. 4, a new individual is generated by joining one individual to another individual as the main path of a residual connection. The number of channels and feature map size of the main path features are adapted to those of the skip path features using pointwise convolution and bilinear interpolation. The weights in the convolution block are initially set to small values to preserve the performance inherited from parent A. The added features are the inputs to the fully connected block. The node where the main and skip paths meet, and the crossover point node are not added to the individual size. The individual providing features for the skip path is referred to as parent A. The individual treated as the main path is referred to as parent B.

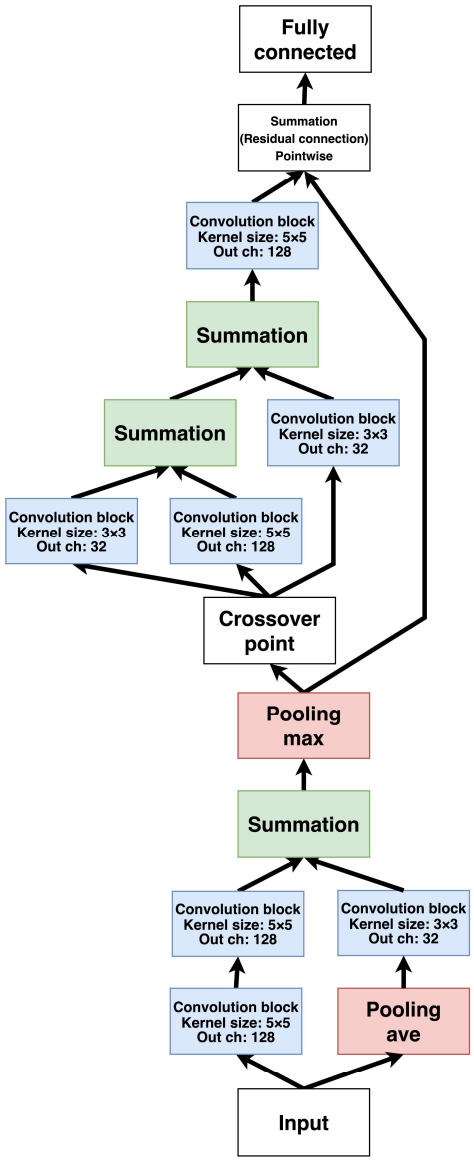

Fig. 4. The genotype generated by crossover between the two individuals shown in Figs. 1 and 2. The individual size s is 8.

D. Adjustment of Weights

In weight inheritance, certain layers of parent B may have mismatches between the number of weights and the size of input features (e.g., convolution and batch normalization).

In this study, we propose a method for adjusting weights for basic convolutional layers. As shown in Fig. 5, basic convolutional layers contain

$$C_{\text{out}} \times C_{\text{in}} \tag{1}$$

kernels, where C_{out} is the number of output channels, and C_{in} is the number of input channels. Therefore, when C_{in} changes, the kernels can be insufficient or too many. If the kernels are insufficient, new kernels are added using random values, constants, or other methods as shown in Fig. 6. If there are too many kernels, the kernels with the larger Frobenius norms, calculated as the sum of the squares of the elements of each

kernel matrix, are selected based on their higher contribution to performance as shown in Fig. 7.

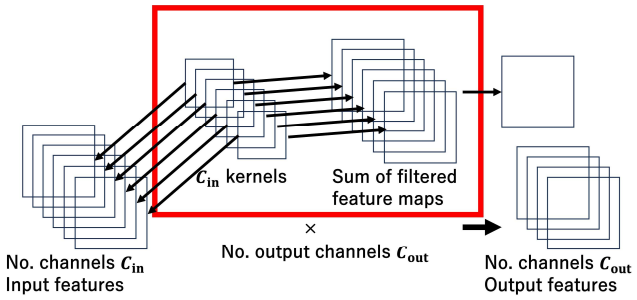

Fig. 5. An illustration of a convolutional layer operation.

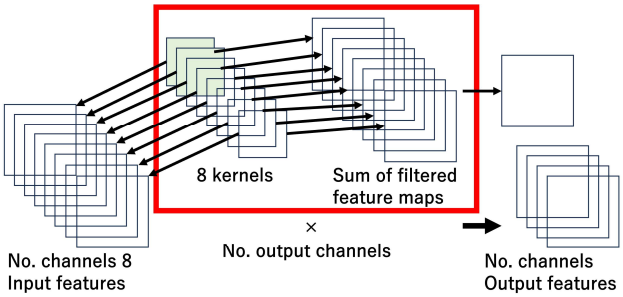

Fig. 6. When the kernels are fewer than the input channels, the missing kernels are supplemented.

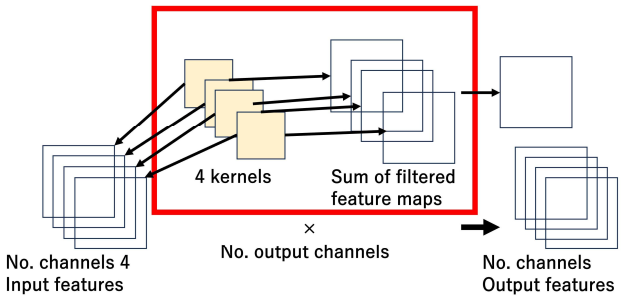

Fig. 7. When the kernels are more than the input channels, the same number of kernels as the input channels is selected.

E. Crossover Rules

First, we explain the rules for selecting crossover targets. Algorithm 2 shows the process of crossover.

Algorithm 2: Crossover is applied to individuals in MLPS with crossover rate $P_{\text{crossover}}$, unless the child size exceeds the maximum size

Require: g : generation, $s_{\max}$: the maximum size of individuals, MLPS_g: g-th generation MLPS
Ensure: MLPS_{g+1}
$\text{MLPS}_{g+1} \leftarrow \text{copy}(\text{MLPS}_g)$;
for each $(l_A, \text{population}_A)$ in MLPS_g **do**
  for each individual_A in population_A **do**
    for each $(l_B, \text{population}_B)$ in MLPS_g **do**
      if $l_B > l_A$ **then**
        break;
      end if
      for each individual_B in population_B **do**
        if (individual_A.size + individual_B.size) > $s_{\max}$ **then**
          break;
        end if
        condition ← (individual_A.get_generation == $g-1$) **or**
                    (individual_B.get_generation == $g-1$);
        if condition **then**
          $A_{\text{parent}} \leftarrow \text{copy}(\text{individual}_A)$;
          $B_{\text{parent}} \leftarrow \text{copy}(\text{individual}_B)$;
          child ← crossover(A_{parent}, B_{parent}) with crossover rate;
          Evaluate the fitness of the child;
          if the child should be added to MLPS_{g+1} then
            Add the child to MLPS_{g+1};
            break;
          end if
        end if
      end for
    end for
  end for
end for

To restrict the targets for the crossover to superior individuals in the g-th generation, we assign individuals added to the population in the $(g-1)$-th generation to either parent A or parent B. For a particular parent A, parent B is selected from levels lower than or equal to the level to which the parent A belongs in the MLPS. A parent B is selected from each level in descending order of fitness (in ascending order of the number of weights if the fitness values are the same), and crossover is applied sequentially. A child is added to the population if its fitness is higher than or equal to the fitness of any individual in levels lower than the s_{child}-th level (s_{child} is the child size). If the child is added, the search is terminated for other parent B candidates at the current level, and the process continues to the next level (l_B) for the same parent A. These rules aim to improve the efficiency of the search by excluding individuals with lower fitness and more weights, and by adding only superior individuals to the population.

Furthermore, children in the g-th generation are added to the $(g+1)$-th generation population, which is replicated prior to crossover, so that children do not affect the selection of crossover targets from the g-th generation population.

Parent A is selected in descending order of fitness (in ascending order of the number of weights if the fitness values are the same) at each level, from lower to higher levels in the MLPS.

F. Population Update

After crossover, the population is updated by additional weight training for n_{f} epochs and re-evaluating each individual's fitness after n_{f} epochs, except for the individuals added in the current generation. Algorithm 3 shows the

process of population update. If the recalculated fitness is not higher than the previous one, neither the weights nor the fitness is updated to prevent fitness decline. This dynamic process mitigates the impact of differences in the number of weight updates on fitness, balancing new individuals with old ones. Moreover, if the number of individuals in each level of the MLPS exceeds its capacity, individuals with relatively lower fitness are excluded to reduce search time.

Algorithm 3: Population update includes additional training, fitness recalculation, and limiting the number of individuals. Individuals whose fitness does not improve compared to the previous generation (overtrained) are excluded from training and evaluation

Require: g: generation, MLPS_{g+1}: $(g+1)$-th generation MLPS
Ensure: updated MLPS_{g+1}

for each $\mathrm{population}_l$ in MLPS_{g+1} **do**
  for each individual in $\mathrm{population}_l$ **do**
    if (individual is overtrained) **or**
      (individual.get_generation $== g$) **then**
      continue;
    end if
    $\mathrm{individual}_{\mathrm{copy}} \leftarrow$ copy(individual);
    Evaluate the fitness of $\mathrm{individual}_{\mathrm{copy}}$;
    if $\mathrm{individual}_{\mathrm{copy}}$.fitness > individual.fitness **then**
      Replace individual in MLPS_{g+1} with $\mathrm{individual}_{\mathrm{copy}}$;
    end if
  end for
  Limit individuals in $\mathrm{population}_l$;
end for

IV. Experiments and Results

To validate the effectiveness of the proposed method, we conducted experiments on a real-world image classification task: predicting the title of a four-panel manga from each frame, using the Manga109 dataset [18, 19]. We compared the performance of MLPS-NAS against CGP-CNN, a strong baseline method also based on evolutionary computation.

A. Task and Dataset

The Manga109 dataset is a collection of Japanese manga, presenting a challenging classification task suitable for evaluating NAS methods. We used frame images from five four-panel manga titles: "Akuhamu", "Koukou No Hitotachi", "OL Lunch", "Tetsu-san", and "Youchien Boueigumi".

Fig. 8 shows three examples of dataset images. The dataset was split into training, validation, and test sets as detailed in Table 1. To improve model robustness, we applied the following offline data augmentation techniques, which increased the amount of training data fivefold:

1) Translation: up to 6 pixels, applied with a probability of 50%.
2) Gaussian Noise: variance randomly sampled in the range [10.0, 50.0], applied with a probability of 50%.
3) Cutout: a single 16×16 pixel region masked in black, applied with a probability of 20%.
4) Random Cropping: a 24×24 pixel region, applied with a probability of 50%.
5) Resizing: to a 32×32 pixel size.

Table 1. The names of the works and the amount of data. A relatively large portion of the data was allocated for validation to ensure reliable fitness evaluation during the evolutionary search

Works	Train	Valid	Test	Total
Akuhamu	486	122	68	676
Koukou No Hitotachi	654	163	91	908
OL Lunch	596	149	83	828
Tetsu-san	374	65	52	491
Youchien Boueigumi	259	65	36	360

Fig. 8. The input image size was $32 \times 32 \times 3$ and the dataset included both black-and-white and color images.

In addition, each channel was normalized using the mean and standard deviation calculated from the training data. The model output was a 5-dimensional vector representing the probabilities that the input image belongs to each work.

The experiments were conducted on a machine with an Intel(R) Core (TM) i9-14900 CPU, 64 GB RAM, and an NVIDIA GeForce RTX 4080 GPU.

B. Search Space

The search space of MLPS-NAS was the optimization of the architecture, which consists of convolution blocks, pooling and other operations. Table 2 and Fig. 9 show the types of layers and blocks that are assigned to the nodes in genotypes.

In some cases, the architecture of parent B required modification of the number of output channels in the pointwise convolutional layer used in the residual connection. If such an adjustment was needed, the child was regarded as an invalid individual.

Table 2. Components of the CNN. The convolution block consists of a convolutional layer, batch normalization, dropout (the dropout rate was set to 0.3) and ReLU activation

Layer or Block	Variety
Convolution block	No. output channels $C_{\mathrm{out}} \in \{32, 64, 128\}$
	Kernel size $k \times k, k \in \{1, 3, 5\}$
Pooling layer	Max, Average
Summation	-
Concatenation	-

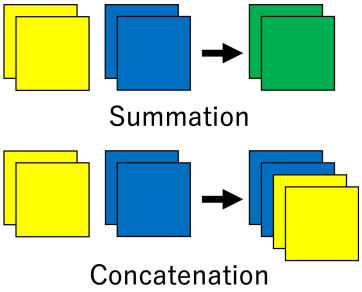

Fig. 9. Summation adds the feature maps for each channel in the image. Concatenation joins the feature maps along the channel axis. If the number of channels differs between the two input maps, the smaller one is zero-padded, and if the feature map sizes differ, the larger one is downsized using max pooling.

We introduced a convolution block to the input layer with $k = 1$ and $C_{\text{out}} = 64$. The block in the parent B was not inherited by the child in crossover because we considered layers that directly process input data to be less generic. The fully connected block consisted of global average pooling [7], a fully connected layer, and a softmax function.

In population initialization, we added individuals representing six different CNNs as the smallest substructures. Each consisted of an input layer, a convolution block with a kernel size $k = 3$ or 5, output channels $C_{\text{out}} = 32$, 64, or 128, and a fully connected block. The fitness was set to 0.0, and these were excluded from the population update and from being selected as parent A in crossover.

C. Parameter Settings

We set the maximum number of generations of our method to 10, the top level of MLPS to 30, and the maximum number of individuals per level to 6. Table 3 lists CGP-CNN parameters for the module search and the comparison method.

Table 3. Parameters of CGP-CNN, the module search and the comparison method

Parameter name	Module search	Comparison method
No. offspring per generation	2	2
Mutation rate	0.05	0.05
No. rows in graph N_r	5	5
No. columns in graph N_c	Integers [4, 6]	30
Levels-back	No limit	10
Minimum of intermediate nodes	$N_c + 1$	9
No. epochs for fitness evaluation	5	50
Search termination condition	15 evaluations	200 generations

Six CNNs with architectures explored in module search were added to the population after 40 (n_f) training epochs to evaluate their fitness. In crossover, the fitness was evaluated in 4 (n_f) training epochs starting from inherited weights. In population update, each individual fitness and weights were updated by training for 4 (n_f) epochs. The crossover rate $P_{\text{crossover}}$ was designed by Eq. (2).

$$P_{\text{crossover}}(r_A) = e^{-0.15 r_A} \tag{2}$$

where r_A is the descending order of fitness of each parent A at its level, and e is Napier's constant. To reduce search time, the crossover rate was set based on the positive correlation between the fitness of parents and children.

The interpolation method for kernels in weight inheritance was sampling new weights from a normal distribution $N(\mu, \sigma^2)$ based on the mean μ and standard deviation σ of the weights of the target convolutional layer. This approach aimed to stabilize both the output and the training process.

Table 4 shows the training parameters of the CNNs used in the search. In the test, the CNN represented by the best individual was trained with a learning rate of 0.0005 until the accuracy on the validation data stopped increasing for 20 consecutive epochs. The MLPS-NAS started training from the weights of the explored individual, while the CGP-CNN started training from the weights with He initialization [22].

Table 4. Training parameters of the CNN model used in the search by MLPS-NAS and CGP-CNN

Parameter name	Value
Batch size	80
Optimization method	AdamW
Loss function	Cross entropy loss
Learning rate	0.0005
Initialization of weights	He initialization [22]

D. Results and Discussion

The average accuracies on the test dataset of three trials with the CNNs optimized by MLPS-NAS and CGP-CNN were 0.9051 ± 0.0062 and 0.9071 ± 0.0028, respectively. The accuracy was computed on the test dataset $D_{\text{test}} = \{(x_j, y_j)\}_{j=1}^{N_{\text{test}}}$, where each ground-truth label y_j and prediction $\hat{y}_j^{(k)}$ are represented as one-hot vectors. For the model $\hat{f}^{(k)}$ obtained in each trial ($k = 1, 2, 3$), the accuracy was given by

$$\text{Accuracy}(\hat{f}^{(k)}) = \frac{1}{N_{\text{test}}} \sum_{j=1}^{N_{\text{test}}} \hat{y}_j^{(k)} \cdot y_j$$

For each method, the best model found during the search was fully trained and then evaluated on the test dataset. The accuracy of MLPS-NAS was comparable to that of the existing method.

Table 5 shows the performance and optimization costs of MLPS-NAS and CGP-CNN in the trials with the highest test accuracy. MLPS-NAS evaluated more individuals in a shorter search time and for fewer total training epochs by weight inheritance. On the other hand, the number of parameters was approximately four times more than the CGP-CNN results. This suggests the need to remove substructures with low performance contributions during or after the search, or to improve the crossover operation. Moreover, enhancing accuracy to consistently outperform other methods remains a future challenge.

Table 5. The results are from the trial (among three independent trials) that achieved the highest accuracy on the test data ($\max_{k \in \{1,2,3\}} \text{Accuracy}(\hat{f}^{(k)})$)

Method	Accuracy	No. Parameters	Search Time [GPU hours]	No. Evaluations	No. Epochs
Our method	0.9121	5,046,245	15.324	739	3866
CGP-CNN	0.9091	1,244,229	30.551	289	14,450

Note: "Search time", "No. evaluations" and "No. epochs" indicate the total values at the point when the best individual was discovered during the search

Figs. 10 and 11 illustrate the optimal CNN architectures obtained by MLPS-NAS and CGP-CNN, respectively, in the trial corresponding to results in Table 5. Unlike CGP-CNN, MLPS-NAS generated deeper networks composed of sequentially connected substructures. Based on these observations, potential improvements include crossover using summation or concatenation operations, or mutation that replaces components with larger substructures. In the three trials, both an architecture like the one shown in Fig. 10 and another one shown in Fig. 12 were obtained. Although the latter showed high performance, it did not achieve the highest accuracy, indicating redundant exploration. Potential improvements include broader exploration by increasing the number of architectures explored in module search or introducing additional evolutionary operations.

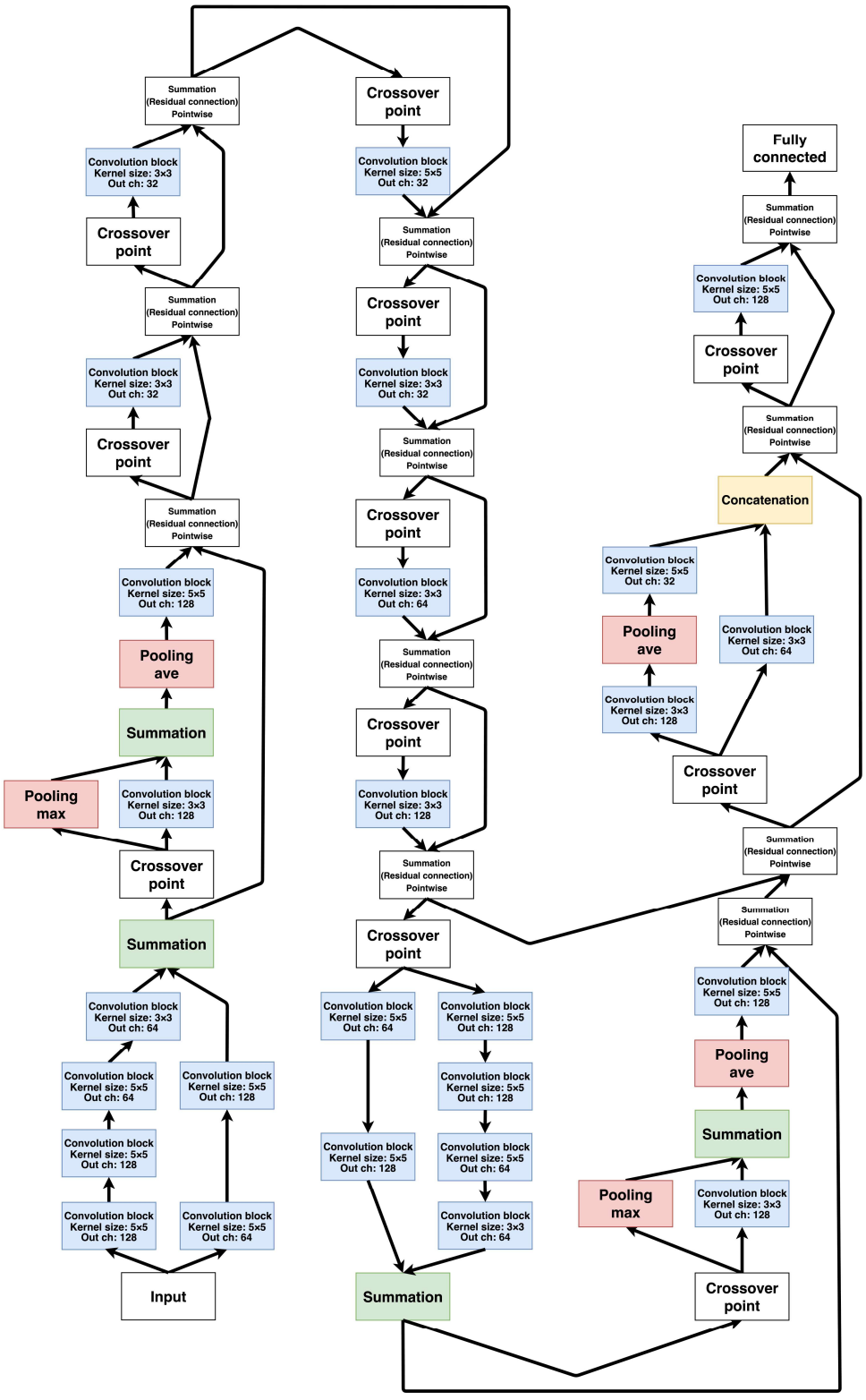

Fig. 10. The best CNN discovered by MLPS-NAS includes diverse and repeated substructures, generated through evolutionary combinations with MLPS and weight inheritance. Such patterns would be difficult to design manually.

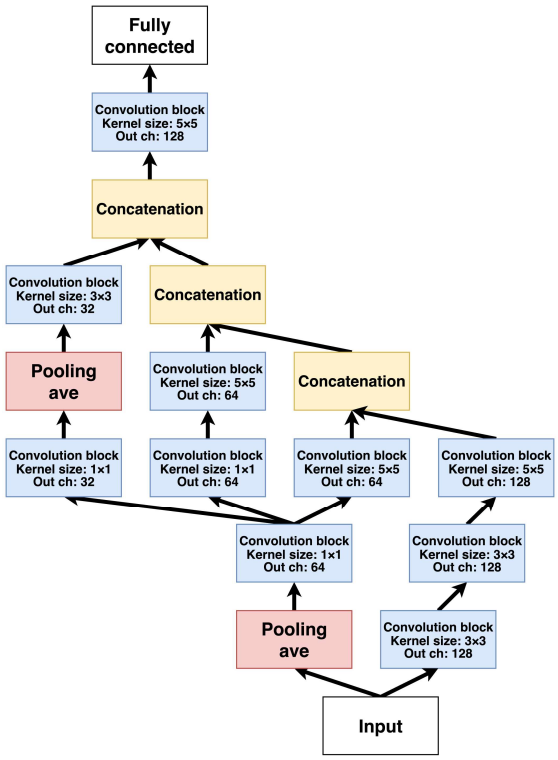

Fig. 11. The best CNN discovered by CGP-CNN features a shallow architecture with more parallel operations.

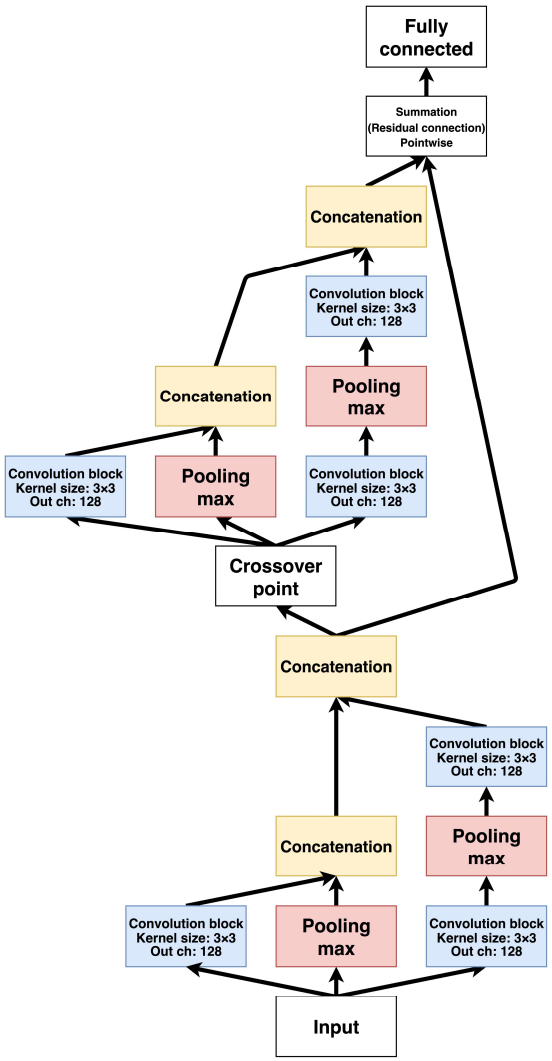

Fig. 12. The best CNN obtained in another trial was generated through just one crossover operation.

Fig. 13 shows the highest fitness value at each generation. The horizontal axis represents the generation, and the vertical axis represents the fitness value. After the seventh generation, there is little change in the best fitness. This stagnation may indicate that the optimization process has largely been completed. However, one possible reason is that, in the early stages of the search, the population included individuals with weights trained for fewer epochs. This led to the simultaneous optimization of both weights and architectures, resulting in a rapid increase in fitness. To obtain better individuals and mitigate stagnation, potential solutions include mutation operations that reduce the number of parameters and individual selection that considers diversity based on the number of parameters and the generation in which they were obtained.

Fig. 14 shows statistical information on the number of weights in individuals across the different levels of the MLPS. The horizontal axis represents the level in the MLPS, while the vertical axis represents the number of weights. Interestingly, some individuals across different levels have approximately the same number of weights, suggesting that fitness is influenced not only by the representational capacity (determined by the number of parameters) but also by the combinations of operations. In addition, differences in the frequency of weight updates and the rate of learning convergence may not be fully reflected in individual evaluations. Considering these factors could lead to the discovery of more effective architectures. Furthermore, crossover operations that introduce parallel connections may enhance parameter diversity within the same level, thereby enabling broader exploration.

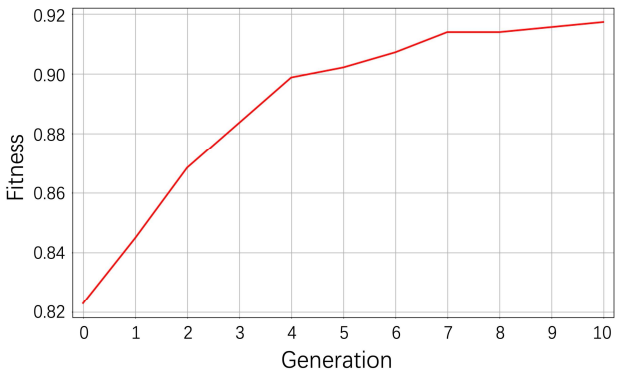

Fig. 13. MLPS-NAS steadily improved the highest fitness across generations through weight inheritance, demonstrating the effectiveness of combining NAS and MLPS.

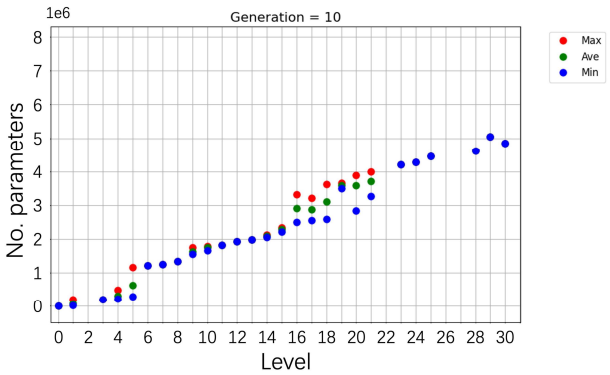

Fig. 14. Distribution of the number of weights in the MLPS after the search. Individuals with more parameters tended to be located at higher levels.

Figs. 15–20 show statistical information on fitness and the number of individuals in each level of the MLPS in the trial shown in Table 5. The horizontal axis represents the level in the MLPS. In Figs. 15–17, the vertical axis represents the fitness. In Figs. 18 to 20, the vertical axis represents the number of individuals. In the stacked bar graphs, the shades of color indicate the cumulative number of training epochs for weights inherited from the parent *A*.

In terms of the fitness distribution, the lower levels show greater variation in fitness, while the higher levels show only slight differences. This is because, as the search progresses, the difference between the converged fitness value and the threshold for addition to the MLPS becomes smaller.

The evolutionary operations generated the larger, better-performing individuals that incorporated the effective substructures. Reducing the number of individuals at lower levels through the selection operation contributes to more efficient exploration. In addition, to improve fairness in evaluating overtrained individuals, simultaneous optimization of the learning rate could be a more effective approach.

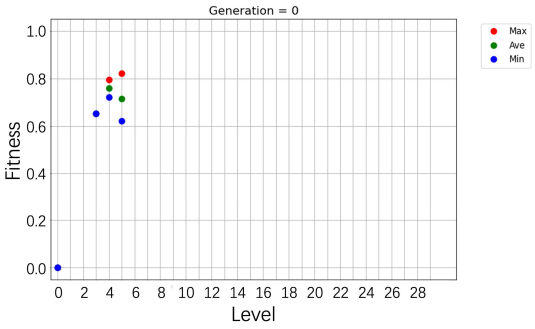

Fig. 15. Fitness distribution in each level of MLPS after module search.

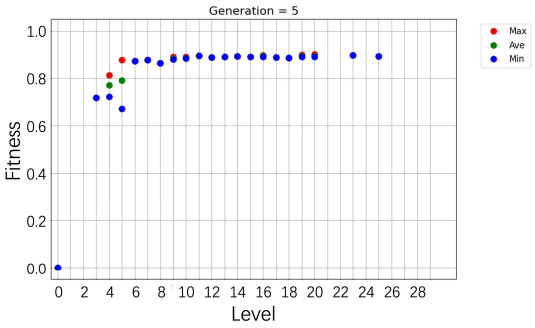

Fig. 16. Fitness distribution in each level of MLPS after the fifth generation.

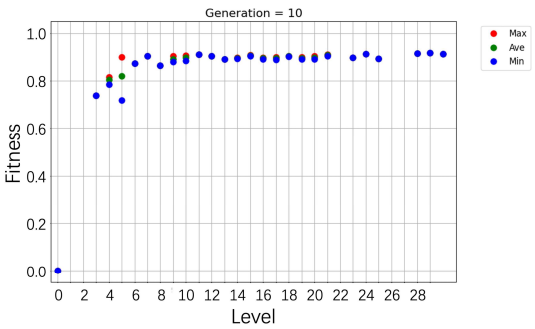

Fig. 17. Fitness distribution in each level of MLPS after the tenth generation.

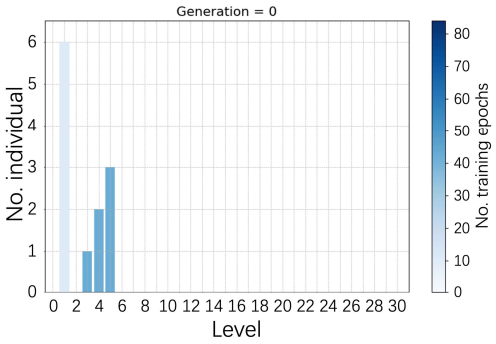

Fig. 18. Number of individuals in each level of MLPS after module search.

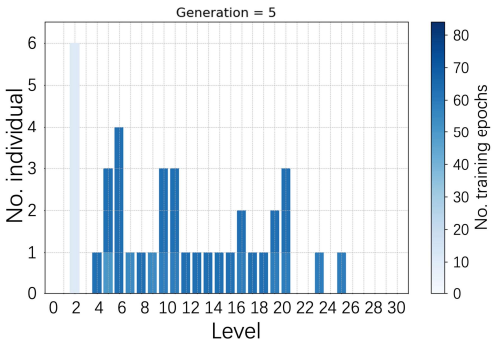

Fig. 19. Number of individuals in each level of MLPS after the fifth generation.

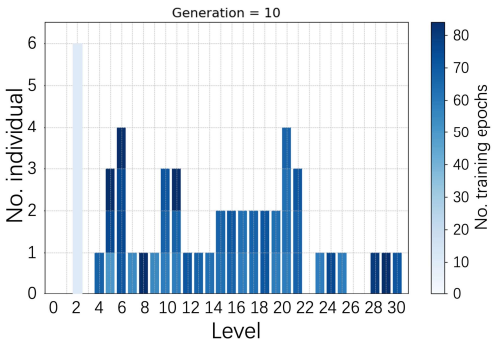

Fig. 20. Number of individuals in each level of MLPS after the tenth generation, forming a pyramid-like distribution.

V. Conclusion and Future Work

A. Conclusion

We proposed MLPS-NAS, a novel method that combines Neural Architecture Search with the Multi-Layered Population Structure originally developed for Genetic Programming. The core innovation lies in adapting the principles of hierarchical population management and partial solution reuse to the automated design of CNNs. By integrating a weight inheritance mechanism, MLPS-NAS dramatically accelerates the search process.

Our experimental results demonstrated that MLPS-NAS achieves classification accuracy comparable to the CGP-CNN baseline while significantly improving search efficiency. This work shows that the synergy between MLPS and NAS offers a powerful and promising approach to mitigating the substantial computational burden associated with automated deep learning model design.

B. Limitations and Future Work

The findings of this study also highlight several limitations and clear directions for future research.

First, to address the issue of model bloat, future work should focus on discovering more lightweight models. The current crossover operator in MLPS-NAS inherently favors the creation of large networks. This could be counteracted by integrating pruning or knowledge distillation techniques into the search process. A more direct approach would be to evolve the architecture using multi-objective optimization, where model size (e.g., number of parameters or computational cost) is treated as a second objective to be minimized alongside classification error. Furthermore, introducing mutation operators that can remove or replace substructures could provide a necessary counter-balance to the additive crossover.

Second, the generalizability of MLPS-NAS should be more comprehensively evaluated. This includes conducting comparisons against a wider range of state-of-the-art NAS methods (e.g., DARTS, ENAS) on standard academic method on other computer vision tasks, such as object detection and semantic segmentation, would also be essential.

Finally, the MLPS-NAS framework could be extended to other types of deep learning models. Applying its principles to search for optimal Transformer-based architectures [23] or Graph Neural Networks (GNNs) [24] could further demonstrate the versatility and power of this approach.

By continuing to unite concepts from evolutionary computation with the challenges of deep learning, this line of research can pave the way for more efficient and automated design of sophisticated AI systems.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Kazuki Yabuuchi developed the algorithm, prepared the experimental environment, conducted the experiments, and wrote the manuscript. Naoki Mori proposed the original concept based on his research, developed the algorithm, and provided comments and advice. All authors reviewed and approved the final version of the paper.

Funding

This work was supported by JSPS KAKENHI Grant, Grant-in-Aid for Scientific Research (C), 23K11252.

Acknowledgment

We would like to express our sincere gratitude to the editors and the anonymous reviewers for their insightful comments and valuable feedback. We also thank A. Liu for her support throughout the submission process.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, 2012.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587.

[3] H. Yu, L. T. Yang, Q. Zhang *et al.*, "Convolutional neural networks for medical image analysis: State-of-the-art, comparisons, improvement and perspectives," *Neurocomputing*, vol. 444, pp. 92–110, July 2021.

[4] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, April 1980.

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[6] C. Szegedy, W. Liu, Y. Jia *et al.*, "Going deeper with convolutions," in *Proc. the 2015 IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

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

[8] B. Zoph and Q. Le, "Neural architecture search with reinforcement learning," arXiv Preprint, arXiv:1611.01578, 2016.

[9] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proc. the 35th International Conference on Machine Learning*, 2018, pp. 4095–4104.

[10] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," arXiv Preprint, arXiv:1806.09055, 2018.

[11] E. Real, S. Moore, A. Selle *et al.*, "Large-scale evolution of image classifiers," in *Proc. the 34th International Conference on Machine Learning*, 2017, pp. 2902–2911.

[12] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[13] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 394–407, April 2020.

[14] R. de Lima Mendes, A. H. da Silva Alves, M. de Souza Gomes *et al.*, "gaCNN: Composing CNNs and GAs to build an optimized hybrid classification architecture," in *Proc. the 2021 IEEE Congress on Evolutionary Computation*, 2021, pp. 79–86.

[15] J. R. Koza and R. Poli, "Genetic Programming," in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, E. K. Burke and G. Kendall, Eds. Boston, MA: Springer US, 2005, pp. 127–164.

[16] T. Hasegawa, N. Mori, and K. Matsumoto, "Genetic programming with multi-layered population structure," in *Proc. the Genetic and Evolutionary Computation Conference Companion*, 2017, pp. 229–230.

[17] B. W. Goldman and W. F. Punch, "Parameter-less population pyramid," in *Proc. the 2014 Annual Conference on Genetic and Evolutionary Computation*, pp. 785–792, 2014.

[18] Y. Matsui, K. Ito, Y. Aramaki *et al.*, "Sketch-based manga retrieval using manga109 dataset," *Multimedia Tools and Applications*, vol. 76, no. 20, pp. 21811–21838, November 2017.

[19] K. Aizawa, A. Fujimoto, A. Otsubo *et al.*, "Building a manga dataset "manga109" with annotations for multimedia applications," *IEEE MultiMedia*, vol. 27, no. 2, pp. 8–18, April 2020.

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

[21] J. F. Miller and S. L. Harding, "Cartesian genetic programming," in *Proc. the Genetic and Evolutionary Computation Conference*, 2008, pp. 2701–2726.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proc. the 2015 IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034.

[23] A. Vaswani, N. Shazeer, N. Parmar *et al.*, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017, pp. 5999–6009.

[24] F. Scarselli, M. Gori, A. C. Tsoi *et al.*, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, January 2008.