Article

Modified Neural Architecture Search (NAS) Using the Chromosome Non-Disjunction

Kang-Moon Park ¹ , Donghoon Shin ^{2,*} and Sung-Do Chi ^{3,*}

¹ Department of Electronic Engineering, Korea National University of Transportation, Chungju 27469, Chungcheongbuk-do, Korea; kmpark@ut.ac.kr

² Department of Mechanical Systems Engineering, Sookmyung Women's University, Seoul 04310, Korea

³ Department of Software, Korea Aerospace University, Goyang 10540, Gyeonggi-do, Korea

* Correspondence: dhshin@sookmyung.ac.kr (D.S.); sdchi@kau.ac.kr (S.-D.C.)

Abstract: This paper proposes a deep neural network structuring methodology through a genetic algorithm (GA) using chromosome non-disjunction. The proposed model includes methods for generating and tuning the neural network architecture without the aid of human experts. Since the original neural architecture search (henceforth, NAS) was announced, NAS techniques, such as NASBot, NASGBO and CoDeepNEAT, have been widely adopted in order to improve cost- and/or time-effectiveness for human experts. In these models, evolutionary algorithms (EAs) are employed to effectively enhance the accuracy of the neural network architecture. In particular, CoDeepNEAT uses a constructive GA starting from minimal architecture. This will only work quickly if the solution architecture is small. On the other hand, the proposed methodology utilizes chromosome non-disjunction as a new genetic operation. Our approach differs from previous methodologies in that it includes a destructive approach as well as a constructive approach, and is similar to pruning methodologies, which realizes tuning of the previous neural network architecture. A case study applied to the sentence word ordering problem and AlexNet for CIFAR-10 illustrates the applicability of the proposed methodology. We show from the simulation studies that the accuracy of the model was improved by 0.7% compared to the conventional model without human expert.

Keywords: deep learning; neural network structuring; genetic algorithm; chromosome non-disjunction

Citation: Park, K.-M.; Shin, D.; Chi, S.-D. Modified Neural Architecture Search (NAS) Using the Chromosome Non-Disjunction. *Appl. Sci.* **2021**, *11*, 8628. <https://doi.org/10.3390/app11188628>

Academic Editor: Giancarlo Mauri

Received: 6 August 2021

Accepted: 10 September 2021

Published: 16 September 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Deep learning techniques have been successfully applied in multiple fields, including engineering, in recent years due to their state-of-the-art problem-solving performance [1,2]. Examples include image recognition [3,4], natural language processing (NLP) [5,6], game artificial intelligence (AI) [7], self-driving systems [8,9], and agriculture [10]. Furthermore, open-source toolkits for deep learning have become more varied and easier to use [11]. These programming libraries are easily accessible to non-experts.

The majority of recent research in deep learning has been focused on finding better architectures [12]. In other words, designing neural network architecture plays a vital role in deep learning [13]. However, since most of the available architectures have been developed manually, the entire process is time-consuming and susceptible to errors [14]. Furthermore, the larger scale of current deep neural networks (DNNs) results in the more complex structure of the DNNs [15]. Thus, the design and optimization of architecture have become challenging for designers.

To solve this problem, many researchers have tried to automate DNN design [15–17]. The automated designing methods for effective DNN architecture are state-of-the-art technologies in studies on deep learning architecture [18], which can reduce the costs and time for the model deployment [18].

Neural architecture search (NAS) is a technique used to automate the design process of DNNs. NAS has been known as a subcategory of automated machine learning (Au-

toML), and it contains significant overlap with hyper-parameter optimization [14]. NAS encompasses a broad set of techniques that design DNN architecture automatically for a given task [18,19]. NAS has three challenges: designing network architecture, putting components into a network topology, and hyper-parameter tuning [15]. Optimization and study of each factor are necessary for generating of a new task.

There are several types of approaches to NAS, including grid search [20], the Bayesian-based Gaussian process (BGP) [17,21,22], tree-structured Parzen estimators [23], and the evolutionary algorithm (EA). Grid search tests all combinations of the parameters to determine the best one. For this reason, it is difficult for grid search to evaluate all combinations within an acceptable amount of time [24]. The BGP and tree-structured Parzen estimators are suitable for hyper-parameter optimization (HPO). Bayesian optimization (BO) is a representative case of the HPO methodology [16,17].

An EA is a type of population-based heuristic computational paradigm [25,26]. It has been widely used in the area of optimization problems [27–29]. In evolutionary deep learning (EDL) methodologies, each individual is transformed into a DNN with the corresponding architecture through the mapping from genotype and phenotype [30]. Several EAs, such as particle swarm optimization (PSO) [31–34], the gravitational search algorithm (GSA) [35–37], grey wolf optimizer (GWO) [38–40], firefly algorithm (FA) [41], and genetic algorithm (GA) [12,28,29,42] have been implemented to design neural architectures.

A GA is a main stream of the EAs, which does not require rich domain knowledge [25,43], and is applied to various fields due to their characteristics of being gradient-free and insensitive to local minima [44]. A GA contains several genetic operators, such as selection, mutations, and cross-overs. These operators realize the GA suitable for the optimization of DNN architecture design [24]. However, most of these EDLs offer methods that stack layers without dynamic links.

This paper represents a modified NAS as a DNN structuring methodology through the GA using chromosome non-disjunction [12]. This method does not include hyper-parameter optimization. Instead, it focuses on an automated design process of the full network topology containing dynamic links via the EA.

Modifying and improving the previous NAS method by using the genetic algorithm with chromosome non-disjunction [12] are the main contributions of this research. This dramatically improves the complex problem solving performance, which cannot be solved through conventional artificial neural network (ANN). The aim of this paper is to present a novel methodology for tuning an existing DNN.

The purpose of this study is to propose an automated method for improving the performance of conventional DNNs. Our approach differs from others in that (1) it includes the destructive approach as well as the constructive approach to decrease the searching cost and time, similar to pruning methodologies [45–47], (2) it utilizes a chromosome non-disjunction operation to preserve information from both parents without any information loss, and (3) it offers mutation of links between several layers. The performance can be improved by tuning the previous neural network architecture with residual layers. Note that the previous cross-over operation must lose information when it obtains new information.

The rest of this paper is organized as follows. Section 2 presents the background and related works. Section 3 describes the overall methodologies. Section 4 illustrates the case study applied to the sentence word order. Finally, conclusions and future works are derived in Section 5.

2. Background and Related Works

In this study, we introduce a neural architecture search method using a GA with chromosome non-disjunction. Before introducing our method in Section 3, this section reviews NAS.

2.1. Automated Machine Learning (AutoML)

The ultimate goal of AutoML is to determine the best-performing learning algorithm without human intervention [48]. To exclude human intervention in the auto-design process, there are several challenges, including hyper-parameter optimization (HPO), meta-learning, and NAS [49].

HPO aims to find the optimal hyper-parameters in a specific DNN. Every machine learning system has several hyper-parameters, and they can represent a complex neural network architecture [18,49]. HPO can define a neural network architecture. Bayesian optimization (BO) is a well-known method for HPO.

Meta-learning can improve the performance of an ANN by using meta-data. However, it cannot search for neural architecture significantly faster than HPO or NAS. The challenge for meta-learning is to learn from prior experience in a systematic, data-driven way [49]. Collecting meta-data is required to form prior learning tasks, and the system can then learn from this meta-data.

NAS techniques provide the automated generating of DNN architecture. Recent DNNs have increased in complexity and variety. However, currently employed neural network architectures have mostly been designed manually [49]. This includes a time-consuming and error-prone process [12]. NAS can design a neural architecture automatically. This includes the HPO process. However, it is not aimed at HPO. To generate the DNN architecture topology, NAS usually uses EAs.

2.2. Evolutionary Algorithm for Neural Network

Evolutionary architecture search is also called NeuroEvolution, which uses an EA. Evolutionary architecture search is a combination of ANN and the evolutionary strategy [50]. The purpose of an EA is to optimize the neural network architecture topology.

The PSO is a standard optimization algorithm that consists of a particle population [32,51]. It has been used to optimize hyper-parameters of neural networks. Recently, improved PSO algorithms have been used to enhance the ANNs by HPO or stacking layers [31–34]. It is useful; however, only few researchers have investigated the use of PSO for evolutionary design of ANNs because PSO is more successful on smaller networks than GA [33,34].

GWO is one of the latest bio-inspired optimization algorithms, which mimics the hunting activity of gray wolves [39,40]. GWO determines the most important decision through social dominant hierarchy. It has been applied to the HPO of several types of neural networks [38,39]. GWO is used to find the optimal number of hidden layers, nodes, initial weight and biases. In this research, the best result is obtained by GA; however, GWO shows a better average result.

GSA shows a good performance in exploration search but a weak one in exploitation search. In order to overcome this, some hybrid methods have been proposed [35,37]. Those hybrid algorithms show advanced results for optimizing the initial weights or parameters. GAs has been the most commonly adopted method for designing ANN architectures in recent neuro-evolutionary approaches [12,15,52,53]. An EA evolves the population of the ANN architectures; automatically designed ANN architectures are trained and tested in each generation. Offspring inherit information of layers and links from their parents with mutations.

In addition, some methods for tuning the granular neural network using EA have been proposed [38,41,42,54]. Granular neural networks are defined by sub-granules as sub-hidden layers. An architecture of DNN is presented as stacked sub-granules. EA algorithms such as GWA, FA, or GA are applied to this granule neural networks to tune the hyper-parameters.

Most of these methods are aimed to optimize hyper-parameters or to stack layers. However, we aim for the structuring of neural architecture. In order to structure an existing neural network, GA is used in this study. In addition, a chromosome non-disjunction operator is added to the dynamic linkage.

2.3. Neural Architecture Search Using Genetic Algorithm

GA has been the most common method used for designing neural network architectures [14,44,55–59]. In these studies, GA is used to auto-design the architecture of neural networks within neurons and links. Each of the algorithms have their own phenotype and genotype in order to present a neural network. All neurons and links are expressed in early studies. However, layers and their links are expressed instead of neurons in recent studies because it is hard to express and requires high computing power.

NeuroEvolution of augmenting technologies (NEAT) is the most famous NeuroEvolution technique using a GA [57]. For example, NEAT has produced successful results in solving the XOR problem by evolving a simple neural network architecture within a minimum architecture. In addition, coDeepNEAT, which uses the NEAT methodology, can even automatically design convolutional neural network (CNN) architecture through evolution [15]. CoDeepNEAT uses a node as a layer in a DNN, and it has been very successful since it simplifies a complex neural network. However, it cannot start from the middle state of the architecture because NEAT is a constructive algorithm that must be initiated from a minimum architecture. For this reason, many researchers have added operations such as a delete node. However, the simple addition of a destructive method to the evolutionary architecture search cannot force it to start from the middle of the architecture.

A variable chromosome GA using chromosome non-disjunction is a solution [32]. A new genetic operation named chromosome non-disjunction (or attachment) makes changes in the numbers of chromosomes. This genetic operation decreases or increases the volume of genetic information, and this causes changes in the ANN architecture. This characteristic of the operation enables the ability to start from a middle state in an ANN architecture evolution process. Due to this characteristic, it can be useful for tuning an existing DNN. However, the previous evolutionary architecture search with a variable chromosome GA using chromosome non-disjunction cannot present the DNN architecture, since it uses nodes and links.

3. Methodology

We have proposed a modified NAS to deal with the aforementioned minimum architecture and structuring issues. It looks like a conventional pruning method. However, our methodology adopts the special genetic operation called chromosome non-disjunction to allow the destructive searching not possible in conventional auto-design ANN architectures [12]. The chromosome non-disjunction operation provides the variability for designing an ANN architecture. Consequently, our approach does not need to start from the minimum architecture. Instead, the designer should define the initial ANN architecture. In addition, this method does not consider the speed or acceleration of searching.

3.1. Overall System

Figure 1 shows the system configuration of a modified NAS. It consists of a GA generator and a group of DNN generators and evaluators. The GA generator has a role in the control population. It creates new generations of individuals that present ANN architectures. The genetic controller is divided into a selector, genetic operator, and gene pool. The selector selects appropriate individuals to survive by applying the fitness function. The architecture designer should configure the suitable fitness function based on the design goal.

Figure 1. System configuration of NAS using a genetic algorithm with chromosome non-disjunction.

The genetic operator has three operations: (i) a cross-over operation that blends information of parents to make various but parent-like offspring; (ii) a mutation operation that changes a random gene to differentiate it from the parents; and (iii) a non-disjunction operation [12] that makes two offspring where one has less information and the other one has more information.

The DNN generator designs neural architectures from chromosomes in individuals and evaluates them. The deep neural architecture generator interprets chromosomes with a node checker and link checker. Every individual has several chromosomes, which mean layers and links. However, not all individuals are runnable and learnable. Some have broken connections, and some have unmatched inputs/outputs. The model checker and link checker classify these inappropriate neural architectures. The DNN architecture evaluator implements training and testing with deep learning. Users must prepare preprocessed training data and testing data. Next, fitness values, which are calculated by the DNN generators and evaluators, are sent to the GA generator.

3.2. Evolutionary Process

Figure 2 shows the overall evolutionary process, where six steps are processed. In Phase I, users have to initialize some parameters, such as the termination condition, training data, testing data, learning rate, initial chromosomes, mutation rate, crossover rate, and non-disjunction rate. In Phase II, the system selects individuals to survive and crossbreed. The selector uses a fitness value for selection. Selected individuals make offspring according to the given rules. For example, the top 10% of the population can crossbreed, or the bottom 10% of the population can crossbreed. In Phase III, genetic operations are applied to offspring from selected individuals. Mutation, cross-over, and non-disjunction operations ensure that the offspring are different from their parents. These offspring are generated to the DNN architecture in Phase IV. The DNN architecture generator designs the DNN architecture from the chromosomes of individuals through the node checker and link checker. If the node checker of the link checker concludes that generating the DNN architecture from an individual is impossible, the DNN architecture generator will stop. Generated DNN architectures will be trained in Phase V. In Phase VI, the fitness values of the trained DNN architectures are calculated from test and validation results. After Phase VI, individuals sorted by their fitness values are sent to Phase II.

Figure 2. Overall evolutionary process.

DNN architectures can be presented as chromosomes. Figure 3 shows an example of matching between chromosomes and the DNN architecture. Each layer and link must be presented as one chromosome. In addition, each chromosome has to contain all of the information for a layer or link.

Algorithm 1 presents the pseudocode of the genetic operation. Indchild takes genes from each parent randomly in order to widen the exploration. It helps this algorithm find unexpected models [25,26]. In this process, a gene from a parent can be mutated and thereby differ from the parent. Non-disjunction can occur in the same way.

Figure 3. Example of matching between chromosomes and DNN architecture.

Algorithm 1 GA generator

```

Input:  $Ind_{p1}, Ind_{p2}$ 
Output:  $Ind_{child}$ 
if  $Ind_{p1}.chromosomes.num \geq Ind_{p2}.chromosomes.num$  then
     $size := Ind_{p1}.chromosomes.num$ 
else
     $size := Ind_{p2}.chromosomes.num$ 
end if
 $r := \text{random}(0,1)$ 
while True do
    if  $r = 0$  then
         $feedChrom := Ind_{p1}.chromosomes[i]$ 
    else if  $r = 1$  then
         $feedChrom := Ind_{p2}.chromosomes[i]$ 
    end if
    if isNonDisjunction = True then
        Add another  $chromosome[i]$  to  $Ind_{child}$ 
    end if
    if isMutation = True then
        Mutate  $feedChrom$ 
    end if
     $feedChrom$  to  $Ind_{child}$ 
end while
return  $Ind_{child}$ 

```

Algorithm 2 presents the pseudocode of the DNN structuring. The most important factor in this process is to check the possibility of developing an architecture and its connectivity. Every input entering into a specific layer is added together because we adopt a FusionNet method for the residual layer [60,61].

Algorithm 2 DNN generator

```

Input: Individual
Output: DNN Architecture
if checkLink is True then
    dead := False
else
    dead := True
end if
if dead = False then
    for each links as l do
        if Individual.chromosomes[l[0]].type = "layer" then
            Add layer to layers
        end if
    end for
    if checkModel is True then
        dead := False
    else
        dead := True
    end if
end if
for each links as l do
    if l[1] is output then
        sum := sum + FINDLINK (x, link)
    end if
end for
procedure FINDLINK(x, link)
    for each links as l do
        if l[1] = link[0] then
            sum := sum + FINDLINK (x, l)
        end if
    end for
    x = layers[link[0]](x)
    if l[1] is not output then
        if layers[link[0]] has activation function then
            actFunc[link[1]]
        end if
    end if
end procedure
return x

```

4. Case Studies

To verify whether the system is capable of generating and/or tuning the previous ANN architecture, we have proposed two pilot studies: (1) a simple linguistic phenomena and (2) CIFAR-10. Case Study 1 uses simple linear models to train Korean grammaticality tasks. Case Study 2 uses some convolutional layers to train the CIFAR-10 dataset.

In Case 1, we verify that the proposed algorithm can search for the same architecture in any case to validate that our methodology can use a constructive as well as destructive approach to tune the DNN architecture. In Case 2, we apply our algorithm to an existing DNN model to improve it.

4.1. Case 1: Korean Grammaticality Task

This case study is intended to further analyze our previous study [62]. The purpose of case study 1 is to verify whether starting with a different network converges to the same result as the previous study. The selected data contain seven different syntactic categories (Noun Phrase, Verb Phrase, Preposition Phrase, Adjective Phrase, Adverbs, Complementizer Phrase, and Auxiliary Phrase) in four-word level sentences [62]. Thus, we

obtained 2401 combinations. In determining those combinations, we consulted the Sejong Corpus as well as linguists to verify the grammaticality. Because Korean displays many underlying operations, such as scrambling and argument ellipsis, as illustrated below, only 113 combinations out of the 2401 resulted in grammatical sentences. More details about this dataset are explained in our previous paper [62]. Due to the fact that the used dataset has low complexity, a simple multiple layer perceptron (MLP) with error backpropagation was implemented in this experiment.

We conducted two cases of experiments with the data to investigate whether the neural networks can be generated. We directly compared one case with a minimum architecture and another without it. We initiated a simple linear layer architecture that contains four inputs. Each input refers to a syntactic category. The output was set for the grammaticality of each combination. The entire population of the first generation had the same initial architecture. The parameters of these two experiments are shown in Table 1. We limited ourselves in population and generation size due to the limited capabilities of the computational environment, although a larger size would yield a better result.

Table 1. Example of Korean sentence with four words.

a.	May-ka May-NOM 'May said that Jake is pretty.'	Jake-ka Jake-NOM	Yepputako Pretty-C	Malhassta said
b.	Argument ellipsis May-ka	()	Yepputako	Malhassta
c.	Scrambling Jake-ka	Yepputako	May-ka	Malhassta

Fitness function (Equation (1)) is defined as follows:

$$fitness = (1 - loss) - R \times \left(\frac{num_{layer}}{num_{avg}} \right) \times (2 \times (1 - loss) - 1) \quad (1)$$

where R is the coefficient of the dependency rate of the number of layers between 0 and 1, num_{layer} is the number of layers, and num_{avg} is the average of the number of layers. This simplifies the DNN architecture while maintaining loss.

Figure 4 shows the initial architecture of Case 1. It has three layers: one input layer, one output layer, and one hidden linear layer with five nodes. It takes word ordering data as inputs and determines whether this input is correct. Since it is a minimum architecture, the overall structure is simple.

Figure 5 shows the initial architecture of Case 2. It has one input layer, one output layer, and two hidden linear layers. One of the hidden layers has a skip connection, and it is added to the linear_2 layer in front of the output layer.

Figure 4. Initial architectures of first experiment.

Figure 5. Initial architectures of second experiment.

4.2. Case 2: CIFAR-10 Dataset

CIFAR-10 is currently one of the most widely used datasets in machine learning and serves as a test ground for many computer vision methods [63]. To evaluate our methodology, we tuned a five-layer AlexNet [64] using the proposed NAS algorithm. All layers and connections are presented as chromosomes in order to apply GA. The accuracy of the generated architecture was calculated by Equation (1). After that, the evaluated models were mated through genetic operators.

The objective of this case study was to show it is possible to tune the previous DNN model using our methodology. The parameters of the experiment are shown in Table 2. These parameters were tuned based on our previous research [12] because of improvement of the complexity of the problem domain. We performed the experiment several times in order to optimize those parameters. The initial model for AlexNet is shown as Figure 6.

Figure 6. Initial model of AlexNet.

Table 2. Parameters of experiments.

Parameter	Value
Population	50
Generations	30
Mutant rate	0.05
Cross-over rate	0.05
Non-disjunction rate	0.1
Learning rate	0.01
Criterion	MSELoss

The purpose of this case study was to tune the existing DNN to improve performance even slightly without adding layers. It can help existing DNNs lighter while maintaining or improving their performance. The parameters and fitness function of the experiments were identical to those in Experiment 1. In addition, this experiment was trained for 50 epochs.

5. Experiment Results

5.1. Case 1 Results

Figure 7 plots the loss and the number of layers and links of the neural network architecture with the lowest loss for each generation of Case 1. The loss of the highest performing species started from 0.27 and decreased to almost 0.1. This system searched

an architecture for word ordering tasks within seven generations. This architecture had a linear layer with five nodes and four links, which started from a linear layer to an output layer. It started adding each link to the output layer from the hidden layer in every evolution step after Generation 6 and then started converging after Generation 7.

Figure 7. Evolutionary process of first experiment.

Figure 8 plots the loss and the number of layers and links of the neural network architecture with the lowest loss for each generation of Case 2. The loss of the highest performing species started from 0.32 and decreased to almost 0.1. This system searched for an architecture for word ordering tasks within five generations. As in Case 1, the architecture in this experiment had a linear layer with five nodes and four links, which started from a linear layer to an output layer. A skip connection layer was eliminated in Generation 2. It started adding a link to the output layer from a hidden layer in every evolution step after Generation 6 and started converging after Generation 5. It is possible that convergence will happen in different generations due to the randomness in the genetic operators.

Figure 9 depicts the final neural architecture. As shown, the two cases generated the same final architecture. This means that the proposed system can find a correct answer with a destructive method as well as with a constructive method. Moreover, it determined a unique architecture that looks like a fork. We leave the detailed investigation of this unique architecture to future research.

Figure 8. Evolutionary process of second experiment.

Figure 9. Evolved final neural architecture of both experiments.

5.2. Case 2 Results

Some tuned AlexNet models were generated after 30 generations. Figure 10 shows some of these architectures, which obtained high fitness value rankings. As Figure 7 shows, these architectures had five convolutional layers, which is also true with AlexNet. However, they show a novel structure with several links between the same layers, which takes on the appearance of a fork. Model 1 shown in Figure 10a has a three-lever fork link before the first flattened layer. Figure 10b has a five-lever fork link before the first flattened layer. Model 3 shown in Figure 10c has two three-lever fork links before the last convolutional layer and flattened layer. The layers of all of these tuned models are perfectly identical to the initial AlexNet model. However, as Figure 8 shows, they have different links between layers. This creates greater accuracy.

Figure 10. Evolved architectures of case 2. (a) Model (a) has a 3-lever fork link before the flattened layer. (b) Model (b) has a 5-lever fork link before the flattened layer. (c) Model (c) has two 3-lever fork links before the last convolutional layer and flattened layer.

Figure 11 demonstrates the model accuracies of 50 times of independent running of the model. The average accuracy of the original AlexNet model with 50 epochs was 0.74582, and the maximum accuracy was 0.7536. The average accuracies of the tuned models with fork-like links were improved compared to the original model. The maximum accuracies were also improved, except for in model (a). Table 3 presents the values of Figure 9. It shows that the proposed algorithm improved upon the conventional model by almost 0.7% without a human expert. Although it seems to be a small improvement compared to the effort, it can be substantiated that it is a meaningful result since it has been achieved without human experts.

Table 3. Parameters of experiments.

Accuracy	Original Model	Model (a)	Model (b)	Model (c)
Average	0.74582	0.75026	0.75292	0.75158
Max	0.7536	0.7534	0.7582	0.7566
Computation time (s)	1.2	1.2	1.2	1.21

Figure 11. Comparison between initial model and generated models.

6. Conclusions

The goal of this paper was to modify our previous NAS methodology for DNN structuring and present the systematic specifications of the proposed methodology. This goal was successfully achieved by proposing a modified neural network architecture search system using a chromosome non-disjunction operation. The main contribution of proposed method is the novel method for tuning an existing DNN model. It can be said that tuning of the already existing DNN without the help of human experts is the main contribution of our research. Additionally, we can find a novel structure of DNN that can improve accuracy without adding layers.

Our approach differs from previous methodologies in that it includes a destructive approach as well as a constructive approach, which is similar to pruning methodologies. In addition, it utilizes a chromosome non-disjunction operation so that it can preserve information for both parents without losing any information. Chromosome non-disjunction is a novel genetic operator we proposed that can reduce or expand chromosomes from their parents [12]. It helps neural networks to be tuned flexibly. As such, the NAS system with our approach can determine the same architecture with both constructive and destructive methods. This methodology was validated using the results of case studies to determine a more effective deep learning model in specific cases. In addition, we applied our method to AlexNet for a CIFAR-10 benchmark test to tune the existing model. We were able to determine a novel DNN structure via this case study. In addition, some of these tuned models demonstrated improved accuracy without adding layers. This is an astonishing result because adding layers is a general method to improve accuracy. Accordingly, this research should be useful for tuning an existing neural architecture. More details of these models will be presented in our next paper.

In this paper, we investigated the proposed modified NAS methodology only with simple examples. However, real cases are larger than those in our case studies because most datasets are significantly larger in the current era. To handle these datasets using our methodology, the proposed method has to be extended. However, it has a limitation in that it has high costs, such as time and resources, despite its improved accuracy. It has data sensitivity issues, which is directly affected by the size of the dataset available [65]. Auto-tuning of the deep neural network is still a time-consuming job when it comes to dealing with massive amounts of real-world data.

Studying the impact of the dataset by considering the nature, size, and complexity on the performance of the proposed approach is the topic of our future research. In order to more clearly substantiate the improved performance of our proposed model, we will compare it with some conventional methods in our future research. In addition, we will apply a recurrent neural network (RNN) to our system. After that, several existing models will be experimented with to tune to improve the results. In addition, the fork-like architecture will be investigated from various perspectives. It can help to find new ways to improve DNN models.

Author Contributions: Conceptualization, K.-M.P. and S.-D.C.; methodology, K.-M.P.; software, K.-M.P.; validation, K.-M.P., D.S. and S.-D.C.; formal analysis, D.S.; investigation, S.-D.C.; resources, D.S.; data curation, K.-M.P.; writing—original draft preparation, K.-M.P.; writing—review and editing, D.S. and S.-D.C.; visualization, K.-M.P.; supervision, S.-D.C.; project administration, S.-D.C.; funding acquisition, S.-D.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: This was supported by Korea National University of Transportation in 2021, National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT; No. 2021R1F1A1048133), and by the Sookmyung Women's University Research Grants (1-2103-1077).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wang, T.; Wen, C.-K.; Jin, S.; Li, G.Y. Deep learning-based CSI feedback approach for time-varying massive MIMO channels. *IEEE Wirel. Commun. Lett.* **2018**, *8*, 416–419. [\[CrossRef\]](#)
2. Hohman, F.; Kahng, M.; Pienta, R.; Chau, D.H. Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE Trans. Vis. Comput. Graph.* **2018**, *25*, 2674–2693. [\[CrossRef\]](#) [\[PubMed\]](#)
3. Li, A.A.; Trappey, A.J.; Trappey, C.V.; Fan, C.Y. E-discover State-of-the-art Research Trends of Deep Learning for Computer Vision. In Proceedings of the 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), Bari, Italy, 6–9 October 2019; pp. 1360–1365.
4. Han, X.; Laga, H.; Bennamoun, M. Image-based 3D Object Reconstruction: State-of-the-Art and Trends in the Deep Learning Era. *IEEE Trans. Pattern Anal. Mach. Intell.* **2019**, *43*, 1578–1604. [\[CrossRef\]](#) [\[PubMed\]](#)
5. Lopez, M.M.; Kalita, J. Deep Learning applied to NLP. *arXiv* **2017**, arXiv:1703.03091.
6. Young, T.; Hazarika, D.; Poria, S.; Cambria, E. Recent trends in deep learning based natural language processing. *IEEE Comput. Intell. Mag.* **2018**, *13*, 55–75. [\[CrossRef\]](#)
7. Justesen, N.; Bontrager, P.; Togelius, J.; Risi, S. Deep learning for video game playing. *IEEE Trans. Games* **2019**, *12*, 1–20. [\[CrossRef\]](#)
8. Hatcher, W.G.; Yu, W. A survey of deep learning: platforms, applications and emerging research trends. *IEEE Access* **2018**, *6*, 24411–24432. [\[CrossRef\]](#)
9. Simhambhatla, R.; Okiah, K.; Kuchkula, S.; Slater, R. Self-Driving Cars: Evaluation of Deep Learning Techniques for Object Detection in Different Driving Conditions. *SMU Data Sci. Rev.* **2019**, *2*, 23.
10. Kamilaris, A.; Prenafeta-Boldú, F.X. Deep learning in agriculture: A survey. *Comput. Electron. Agric.* **2018**, *147*, 70–90. [\[CrossRef\]](#)
11. Rebordera, M.A.; Fajardo, A.C. An Enhanced Deep Learning Approach in Forecasting Banana Harvest Yields. *Int. J. Adv. Comput. Sci. Appl.* **2019**, *10*, 275–280. [\[CrossRef\]](#)
12. Park, K.; Shin, D.; Chi, S. Variable Chromosome Genetic Algorithm for Structure Learning in Neural Networks to Imitate Human Brain. *Appl. Sci.* **2019**, *9*, 3176. [\[CrossRef\]](#)

13. Liang, J.; Meyerson, E.; Mikkilainen, R. Evolutionary architecture search for deep multitask networks. In Proceedings of the Genetic and Evolutionary Computation Conference, Kyoto, Japan, 15–19 July 2018; pp. 466–473.
14. Elsken, T.; Metzen, J.H.; Hutter, F. Neural architecture search: A survey. *arXiv* **2018**, arXiv:1808.05377.
15. Mikkilainen, R.; Liang, J.; Meyerson, E.; Rawal, A.; Fink, D.; Francon, O.; Raju, B.; Shahrzad, H.; Navruzian, A.; Duffy, N.; et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*; Elsevier: Amsterdam, The Netherlands, 2019; pp. 293–312.
16. Kandasamy, K.; Neiswanger, W.; Schneider, J.; Poczos, B.; Xing, E.P. Neural architecture search with bayesian optimisation and optimal transport. *arXiv* **2018**, arXiv:1802.07191.
17. Ma, L.; Cui, J.; Yang, B. Deep neural architecture search with deep graph bayesian optimization. In Proceedings of the 2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI), Thessaloniki, Greece, 14–17 October 2019; pp. 500–507.
18. Adam, G.; Lorraine, J. Understanding neural architecture search techniques. *arXiv* **2019**, arXiv:1904.00438.
19. Dong, X.; Yang, Y. Nas-bench-102: Extending the scope of reproducible neural architecture search. *arXiv* **2020**, arXiv:2001.00326.
20. Bergstra, J.; Bengio, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* **2012**, *13*, 281–305.
21. Williams, C.K.; Rasmussen, C.E. *Gaussian Processes for Machine Learning*; MIT Press: Cambridge, MA, USA, 2006; Volume 2.
22. Moćkus, J. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*; Springer: Berlin/Heidelberg, Germany, 1975; pp. 400–404.
23. Bergstra, J.S.; Bardenet, R.; Bengio, Y.; Kégl, B. Algorithms for hyper-parameter optimization. *Adv. Neural Inf. Process. Syst.* **2011**, *24*, 2546–2554.
24. Sun, Y.; Xue, B.; Zhang, M.; Yen, G.G. Evolving deep convolutional neural networks for image classification. *IEEE Trans. Evol. Comput.* **2019**, *24*, 394–407. [[CrossRef](#)]
25. Back, T. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*; Oxford University Press: Oxford, UK, 1996.
26. Schmitt, L.M. Theory of genetic algorithms. *Theor. Comput. Sci.* **2001**, *259*, 1–61. [[CrossRef](#)]
27. Sun, Y.; Yen, G.G.; Yi, Z. Improved regularity model-based EDA for many-objective optimization. *IEEE Trans. Evol. Comput.* **2018**, *22*, 662–678. [[CrossRef](#)]
28. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197. [[CrossRef](#)]
29. Ruiz-Rangel, J.; Hernandez, C.J.A.; Gonzalez, L.M.; Molineros, D.J. ERNEAD: Training of artificial neural networks based on a genetic algorithm and finite automata theory. *Int. J. Artif. Intell.* **2018**, *16*, 214–253.
30. Sun, Y.; Wang, H.; Xue, B.; Jin, Y.; Yen, G.G.; Zhang, M. Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor. *IEEE Trans. Evol. Comput.* **2019**, *24*, 350–364. [[CrossRef](#)]
31. da Silva, G.L.F.; Valente, T.L.A.; Silva, A.C.; de Paiva, A.C.; Gattass, M. Convolutional neural network-based PSO for lung nodule false positive reduction on CT images. *Comput. Methods Programs Biomed.* **2018**, *162*, 109–118. [[CrossRef](#)]
32. Chatterjee, S.; Hore, S.; Dey, N.; Chakraborty, S.; Ashour, A.S. Dengue fever classification using gene expression data: A PSO based artificial neural network approach. In *Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications*; Springer: Singapore, 2017; pp. 331–341.
33. Kiranyaz, S.; Ince, T.; Yildirim, A.; Gabbouj, M. Evolutionary artificial neural networks by multi-dimensional particle swarm optimization. *Neural Netw.* **2009**, *22*, 1448–1462. [[CrossRef](#)]
34. Yu, J.; Wang, S.; Xi, L. Evolving artificial neural networks using an improved PSO and DPSO. *Neurocomputing* **2008**, *71*, 1054–1060. [[CrossRef](#)]
35. Yin, B.; Guo, Z.; Liang, Z.; Yue, X. Improved gravitational search algorithm with crossover. *Comput. Electr. Eng.* **2018**, *66*, 505–516. [[CrossRef](#)]
36. Pelusi, D.; Mascella, R.; Tallini, L.; Nayak, J.; Naik, B.; Abraham, A. Neural network and fuzzy system for the tuning of Gravitational Search Algorithm parameters. *Expert Syst. Appl.* **2018**, *102*, 234–244. [[CrossRef](#)]
37. Deepa, S.N.; Baranilingesan, I. Optimized deep learning neural network predictive controller for continuous stirred tank reactor. *Comput. Electr. Eng.* **2018**, *71*, 782–797. [[CrossRef](#)]
38. Sánchez, D.; Melin, P.; Castillo, O. A grey wolf optimizer for modular granular neural networks for human recognition. *Comput. Intell. Neurosci.* **2017**, *2017*, 4180510. [[CrossRef](#)] [[PubMed](#)]
39. Turabieh, H. A hybrid ann-gwo algorithm for prediction of heart disease. *Am. J. Oper. Res.* **2016**, *6*, 136–146. [[CrossRef](#)]
40. Mirjalili, S.; Mirjalili, S.M.; Lewis, A. Grey wolf optimizer. *Adv. Eng. Softw.* **2014**, *69*, 46–61. [[CrossRef](#)]
41. Sánchez, D.; Melin, P.; Castillo, O. Optimization of modular granular neural networks using a firefly algorithm for human recognition. *Eng. Appl. Artif. Intell.* **2017**, *64*, 172–186. [[CrossRef](#)]
42. Sánchez, D.; Melin, P.; Castillo, O. Optimization of modular granular neural networks using a hierarchical genetic algorithm based on the database complexity applied to human recognition. *Inf. Sci.* **2015**, *309*, 73–101. [[CrossRef](#)]
43. Mitchell, M. *An Introduction to Genetic Algorithms*; MIT Press: Cambridge, MA, USA, 1998.
44. Yao, X. Evolving artificial neural networks. *Proc. IEEE* **1999**, *87*, 1423–1447.
45. Ai, F. A new pruning algorithm for feedforward neural networks. In Proceedings of the Fourth International Workshop on Advanced Computational Intelligence, Wuhan, China, 19–21 October 2011; pp. 286–289.

46. Han, H.-G.; Zhang, S.; Qiao, J.-F. An adaptive growing and pruning algorithm for designing recurrent neural network. *Neurocomputing* **2017**, *242*, 51–62. [CrossRef]
47. Zemouri, R.; Omri, N.; Fnaiech, F.; Zerhouni, N.; Fnaiech, N. A new growing pruning deep learning neural network algorithm (GP-DLNN). *Neural Comput. Appl.* **2020**, *32*, 18143–18159. [CrossRef]
48. Wong, C.; Houlsby, N.; Lu, Y.; Gesmundo, A. Transfer learning with neural automl. *arXiv* **2018**, arXiv:1803.02780.
49. Hutter, F.; Kotthoff, L.; Vanschoren, J. *Automated Machine Learning*; Springer: Berlin/Heidelberg, Germany, 2019.
50. Zhang, H.; Yang, C.-H.H.; Zenil, H.; Kiani, N.A.; Shen, Y.; Tegner, J.N. Evolving Neural Networks through a Reverse Encoding Tree. *arXiv* **2020**, arXiv:2002.00539.
51. Zitzler, E.; Thiele, L. An Evolutionary Algorithm for Multiobjective Optimization: The Strength Pareto Approach. TIK-Report. 1998. Volume 43. Available online: <https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/145900/eth-24834-01.pdf> (accessed on 15 September 2021).
52. Elsken, T.; Metzen, J.H.; Hutter, F. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv* **2018**, arXiv:1804.09081.
53. Xie, L.; Yuille, A. Genetic cnn. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 1379–1388.
54. Melin, P.; Sánchez, D. Multi-objective optimization for modular granular neural networks applied to pattern recognition. *Inf. Sci.* **2018**, *460*, 594–610. [CrossRef]
55. Larrañaga, P.; Poza, M.; Yurramendi, Y.; Murga, R.H.; Kuijpers, C.M.H. Structure learning of Bayesian networks by genetic algorithms: A performance analysis of control parameters. *IEEE Trans. Pattern Anal. Mach. Intell.* **1996**, *18*, 912–926. [CrossRef]
56. Kwok, T.-Y.; Yeung, D.-Y. Constructive algorithms for structure learning in feedforward neural networks for regression problems. *IEEE Trans. Neural Netw.* **1997**, *8*, 630–645. [CrossRef]
57. Stanley, K.O.; Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evol. Comput.* **2002**, *10*, 99–127. [CrossRef]
58. Miller, G.F.; Todd, P.M.; Hegde, S.U. Designing Neural Networks using Genetic Algorithms. *ICGA* **1989**, *89*, 379–384.
59. Yang, S.-H.; Chen, Y.-P. An evolutionary constructive and pruning algorithm for artificial neural networks and its prediction applications. *Neurocomputing* **2012**, *86*, 140–149. [CrossRef]
60. Hegde, V.; Zadeh, R. Fusionnet: 3d object classification using multiple data representations. *arXiv* **2016**, arXiv:1607.05695.
61. Quan, T.M.; Hildebrand, D.G.; Jeong, W.-K. Fusionnet: A deep fully residual convolutional neural network for image segmentation in connectomics. *arXiv* **2016**, arXiv:1612.05360.
62. Park, K.; Shin, D.; Yoo, Y. Evolutionary Neural Architecture Search (NAS) Using Chromosome Non-Disjunction for Korean Grammaticality Tasks. *Appl. Sci.* **2020**, *10*, 3457. [CrossRef]
63. Recht, B.; Roelofs, R.; Schmidt, L.; Shankar, V. Do cifar-10 classifiers generalize to cifar-10? *arXiv* **2018**, arXiv:1806.00451.
64. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1097–1105. [CrossRef]
65. Nasir, V.; Sassani, F. A review on deep learning in machining and tool monitoring: Methods, opportunities. *Int. J. Adv. Manuf. Technol.* **2021**, *115*, 2683–2709. [CrossRef]