IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 28, 2021, accepted June 17, 2021, date of publication June 23, 2021, date of current version July 1, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3091729

An Automatic Convolutional Neural Network Optimization Using a Diversity-Guided Genetic Algorithm

TIRANA NOOR FATYANOSA[1], (Graduate Student Member, IEEE), AND MASAYOSHI ARITSUGI[2], (Member, IEEE)

[1]Graduate School of Science and Technology, Kumamoto University, Kumamoto 860-8555, Japan
[2]Faculty of Advanced Science and Technology, Kumamoto University, Kumamoto 860-8555, Japan

Corresponding author: Tirana Noor Fatyanosa (fatyanosa@dbms.cs.kumamoto-u.ac.jp)

ABSTRACT Hyperparameters and architecture greatly influence the performance of convolutional neural networks (CNNs); therefore, their optimization is important to obtain the desired results. One of the state-of-the-art methods to achieve this is the use of neuroevolution that utilizes a genetic algorithm (GA) to optimize a CNN. However, the GA is often trapped into a local optimum resulting in premature convergence. In this study, we propose an approach called the ''diversity-guided genetic algorithm-convolutional neural network (DGGA-CNN)'' that uses adaptive parameter control and random injection to facilitate the search process by exploration and exploitation while preserving the population diversity. The alternation between exploration and exploitation is guided by using an average pairwise Hamming distance. Moreover, the DGGA fully handles the architecture of the CNN by using a novel finite state machine (FSM) combined with three novel mutation mechanisms that are specifically created for architecture chromosomes. Tests conducted on suggestion mining and twitter airline datasets reveal that the DGGA-CNN performs well with valid architectures and a comparison with other methods demonstrates its capability and efficiency.

INDEX TERMS Convolutional neural networks, genetic algorithms, hyperparameter optimization, text classification.

I. INTRODUCTION

Despite the noteworthy achievements of convolutional neural networks (CNNs) in recent years (e.g., [1]–[5]), the selection of their hyperparameters and architecture remains a challenging task. Traditional approaches usually use uninformed searches such as grid search [6], [7] and random search [8]. These optimization methods are popular as they are simple and easy to use [8]. However, using these methods can be ineffective if the combination of hyperparameters is unsuitable [9].

An alternative method, called neuroevolution, utilizes an evolutionary algorithm (EA) to optimize a neural network. There are several EA variants that were used in neuroevolution research such as CoDeepNEAT [10], evolution strategies (ES) [11], particle swarm optimization (PSO) [12], differential evolution (DE) [12], covariance matrix adaptation evolution strategy (CMA-ES) [13], genetic programming (GP) [14], and genetic algorithm (GA) [12], [15]–[20]. GA is a popular method used by most researchers mainly because of its ease of implementation, supported by many libraries [21], [22]. Also, since the GA is the most basic and general approach for evolutional computation [23], developed ideas to the GA can be of benefit to existing EAs. Moreover, GA is also resource-friendly as it effectively finds better solutions faster than the other EAs [24], [25]. Therefore, we adopted this algorithm as the base of our approach in this study.

A potential drawback of the GA is that it is often trapped into a local optimum resulting in suboptimal solutions and premature convergence. Possible causes include high selection pressure, limited search directions, or loss of diversity [26]. Several strategies were used to enable the algorithm to escape from the local optimum and preserve the diversity by maintaining or controlling it [27]. These strategies demonstrated the dynamic nature of the optimization process and enhanced the diversity. In this study, we propose the ''diversity-guided genetic algorithm-convolutional neural network (DGGA-CNN)'' which has three types:

The associate editor coordinating the review of this manuscript and approving it for publication was Christian Pilato.

DGGA-CNN1, DGGA-CNN2, and DGGA-CNN3. The DGGA-CNN1 utilizes opposite trends of increase or decrease in the crossover and mutation rates; DGGA-CNN2 uses concurrent rates of increment or decrement; DGGA-CNN3 utilizes random injection.

Our work is committed to the development of the DGGA-CNN that would spread a solution adaptively throughout the search space without prior knowledge of any specific search area. The search direction must be determined by the algorithm itself based on the present situation. The main features of this paper are summarized as follows:

- The DGGA-CNN is fully automated in designing the architecture by using a finite state machine FSM) specifically for text classification. There is no predetermined depth of the CNN in the DGGA-CNN; instead, it is determined by the architecture produced from the FSM.
- Our DGGA-CNN can optimize architecture and all of the hyperparameters for the optimized architecture.
- Our DGGA-CNN uses adaptive parameter control and random injection, guided by an average pairwise Hamming distance to facilitate the search process.
- We build three novel mutation mechanisms created particularly for architecture chromosomes, namely, elimination, exchange, and insertion.
- Our experimental results suggested that the DGGA-CNNs were superior to the other methods in terms of the classification results, efficiency, and handling of premature convergence.

The rest of the paper is organized as follows: Section II provides the related work and Section III presents the development of the proposed DGGA-CNNs. The results and discussion are detailed in Section IV. The conclusions and future work are given in Section V.

II. RELATED WORK

This section provides an overview of the optimization of hyperparameters and architecture, deep neuroevolution, exploration and exploitation strategies in the EA, and population diversity in the EA.

A. OPTIMIZATION OF HYPERPARAMETERS AND ARCHITECTURE

The optimization of hyperparameters and architecture is a laborious, yet important, task as their selection can assess the performance of a neural network. Several studies optimized hyperparameters [6], [8], [13], [19], [28], [29] or architecture [15], and most recent works optimized both [10]–[12], [16], [17], [20], [30]–[33]. In this study, we optimize hyperparameters as well as architecture as this was shown to improve the performance of a neural network.

In general, hyperparameters and architecture optimization methods are classified based on known information, namely, uninformed and informed searches. An uninformed search is one that does not learn from prior results; grid search [6], [7] and random search [8] are included in this category. A grid search tries all combinations within the sets of values, whereas a random search randomly selects a subset of a combination within the defined range. These methods are popular owing to their simplicity and ease of use [8]; however, they can be ineffective if the combination of hyperparameters is inappropriate [9]. Moreover, the computational cost tends to be high [34].

The above drawbacks render the method of uninformed search inferior to that of informed search, as shown by [18], [35], [36]. The informed search uses and learns the knowledge obtained from previous results. One of the searches that falls into this category is Bayesian optimization. This method decides the next evaluated values based on a generated probabilistic model of the optimized function. Several studies used the Bayesian method for the hyperparameter search problem [7], [35], [37]–[39]. Another approach that is also categorized as an informed search is the EA that is a population-based approach that initializes population randomly and uses mutation, crossover, and selection operators to obtain near-optimal solutions. In recent years, the EA has achieved state-of-the-art performances in optimizing deep neural networks [10]–[13], [15]–[20]. It has also proved its superiority over Bayesian optimization [20], [40], [41], which is not applicable to solving a sequential decision problem such as the architecture of neural networks [38]. Moreover, the performance of Bayesian optimization is poor when using a high number of hyperparameters [13]. Therefore, we utilized the EA in this study.

B. DEEP NEUROEVOLUTION

Neuroevolution is the utilization of an EA to optimize neural networks [42]. Recently, numerous EA variants were developed to optimize the hyperparameters and architecture of deep neural networks as mentioned above, of which the GA was found to be most used. Therefore, we adopted the GA as the base of our approach.

Most of the recent studies on deep neuroevolution are applied to image classification [10], [11], [13]–[18] and only a few to text classification [12], [19], [20]. Text data faces different challenges as compared to image data, as it has a large number of words, which leads to sparsity and high computational cost [43], [44]. We attempted to highlight the usability of our approach to text classification and overcome these challenges.

Furthermore, we generated CNN architecture automatically by using an FSM that created similar to that in [32]. The difference being the use of an embedding layer as we focused on text classification, a global max-pooling layer as an intermediate layer between the top and bottom layers, and the specific use of a convolutional layer in the architecture. We also created three novel mutations for the architecture, namely, elimination, exchange, and insertion.

C. EXPLORATION AND EXPLOITATION STRATEGIES IN THE EA

There are two substantial aspects of the EA, namely, exploration and exploitation, that are opposite to each other.

Exploration identifies possible individuals in other search spaces and exploitation observes potential individuals in neighboring areas. A crucial issue faced by the EA is to adjust the exploration and exploitation in the search space. An imbalance of these can cause premature convergence that occurs when diversity in the population decreases owing to identical individuals controlling it. There are several approaches to avoid premature convergence and diversify the population by maintaining or controlling it [27]. In this study, we control the population diversity by using adaptive parameter control and random injection.

Our adaptive parameter control strategy operates by increasing or decreasing the crossover rate (CR) and mutation rate (MR) adaptively, based on the resulting diversity. The merit of this mechanism is that it automates the alteration of the values of CR and MR, but the disadvantage is that the exact global optimum is usually obscure; hence, changing the search direction to exploration or exploitation becomes more difficult. A generally known assumption is that early generations always start with exploration as all the individuals are generated randomly and as the diversity reduces, exploitation takes over the search direction [27]; however, the possibility of occurrence of premature convergence becomes higher because the exploration ability decreases. To avoid this, we must stimulate exploration by increasing the population diversity.

There are two notable perspectives regarding the difference between the use of crossover and mutation, which are operators that determine exploration and exploitation in the GA. Both perspectives lead to different treatments when directing the search towards exploration or exploitation. The first defines a clear boundary between crossover and mutation: crossover is a tool for exploitation, whereas mutation is used for exploration [45], [46]. This initiated the concept of decreasing CR and increasing MR when the population needs to explore more, and increasing CR and decreasing MR when the population needs to emphasize exploitation, as in [47]–[49]. The second is that there is no clear boundary between the crossover and mutation operators as they have roles in both the exploration and exploitation of the search space [27], [50]. This introduces the concept of an increment or decrement in both rates when the population requires to boost exploration or exploitation, respectively, as in [51] and [52]. In this study, we compared both the perspectives to ascertain the one that works best.

Besides adaptive parameter control, we also utilized random injection that is a replacement strategy to maintain diversity, and has been applied in previous studies [53], [54]. Instead of using all the individuals from the prior generation, a random injection replaces the worse individuals with newly generated ones. The new individuals are stepping stones that help the current population to produce better offspring and help to explore high search space dimensions.

D. POPULATION DIVERSITY IN THE EA

In the EA, population diversity is important to prevent premature convergence [46], [55]. Despite its importance, population diversity is still widely used for analysis [46], [56]. Only a few papers use it to help the search in EA, such as to switch between exploration and exploitation [46], alter the fitness function [55], or select the parent population for the next generation [57]. Based on those studies, the use of diversity helps discover better solutions. However, diversity-guided mechanisms have not yet been applied to a more complex problem. Therefore, we investigate the diversity-guided mechanisms on hyperparameter optimization for binary text classification tasks, which are known for their challenges: figurative expressions, context-dependency, imbalanced class distributions, and complex sentences [58]. Moreover, the effectiveness of diversity-guided mechanisms has not yet been utilized to change the CR and MR. The other diversity-guided papers, [46], [55], [57], used fixed parameters and stated that adaptive parameter control would improve the performance and be considered for their future work. Therefore, in this paper, we control the CR and MR values adaptively based on diversity as adaptive parameter control has been proven to improve the performance of EA [49]. Our GA was guided by a simple population diversity approach calculated by using the average Hamming distance among individual pairs in a population. Such an approach alternates between exploration and exploitation using a predetermined diversity threshold.

Population diversity can be measured with respect to its genotype or phenotype levels: the genotype level views gene differences within a population, whereas the phenotype level considers the differences in the fitness values within the population [27]. In this paper, the diversity was measured at the genotype level using an average pairwise Hamming distance. The genotype-based measurement is more appropriate for our research as the CNN produces different results for each run, even with the same values of hyperparameters and architecture, i.e., even if two individuals are identical, their fitness values may be different. Therefore, if we compare the distance between these individuals, the distance at the phenotype level is higher than that at the genotype level. Eventually, this affects the search space exploration and exploitation as they depend on distance. Based on these considerations, we used the genotype-based measurement, to avoid a misdirected search.

III. DIVERSITY-GUIDED GENETIC ALGORITHM-CONVOLUTIONAL NEURAL NETWORK

In this section, we first define in more detail the development of our proposed solution. The approach consists of three basic phases, as shown in Fig. 1. In the beginning, the dataset is preprocessed, the ranges and lists of all the optimized hyperparameters are determined, and an FSM is built. Then, through the DGGA-CNN algorithm, the best combination of

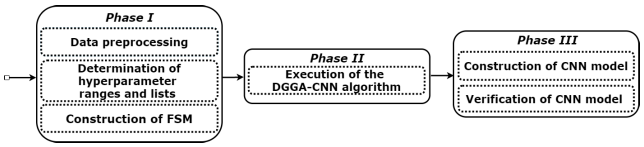


FIGURE 1. Workflow of the proposed DGGA-CNN algorithm.

CNN hyperparameters and architecture is deduced and finally, using the best combination, a CNN model is constructed and verified using test data.

A. PHASE I

1) DATA PREPROCESSING

Data preprocessing is one of the most important steps in text classification. The text is subjected to preprocessing before applying the classification method. We applied different preprocessing for the two datasets. The suggestion mining dataset used 11 steps in sequence, namely, lowercasing, expanding contractions, removing URL, removing numbers, removing hashtags, removing punctuation, removing underscore, removing stopwords, removing non-alphabetic characters, removing words having less than three characters, and correcting spelling. The twitter airline dataset utilized special character removal, contractions expansion, character entity references with symbols replacement, slang correction, typos correction, informal abbreviations expansion, hashtags expansion, usernames expansion, URL removal, words and punctuations separation, and acronym expansion.

2) DETERMINATION OF HYPERPARAMETER RANGES AND LISTS

In this paper, we optimized a total of 22 different types of hyperparameters. We divided those hyperparameters into three categories: global, layer, and architecture hyperparameters. We followed the hyperparameters along with ranges and lists of the global and layer hyperparameters from [20]. Meanwhile, the ranges of the architecture hyperparameters in Table 1 were defined by ourselves as the architecture generation was done automatically.

The first category, global hyperparameters, are those that affect the overall model. Included in this category are the number of epochs (NE), batch size (BS), optimizer (OP), learning rate (LR), and momentum (MO). The second category, layer hyperparameters, determine the hyperparameters at the layer level. Hyperparameter in the embedding layer is the output dimension (OD) that selects among 50, 100, 200, and 300 dimensions in the pre-trained word embedding GloVe [59]. Next, hyperparameters in the convolutional layer are the number of filters (NF), kernel size (KS), activation function (AFC), kernel initializer (KIC), and weight constraint (WCC). Hereafter, there are four hyperparameters for dense layer, viz., the number of neurons (NN), activation function (AFD), kernel initializer (KID), and weight constraint (WCD). As for the dropout layer, max-pooling layer, and output layer, the hyperparameters are dropout rate (DR), pool size (PS), and kernel initializer (KIO), respectively. Finally, the third category, architecture hyperparameters, define the depth of the CNN architecture. The hyperparameters included in this category are shown in Table 1.

TABLE 1. Range of architecture hyperparameters.

Hyperparameter	Definition	Range
NC	Number of convolutional layers	Min: 1, Max: *
NM	Number of max-pooling layers	Min: 0, Max: *
ND	Number of dense layers	Min: 0, Max: *
NDO	Number of dropout layers	Min: 0, Max: *

Notes: *) Depends on the generated architecture.

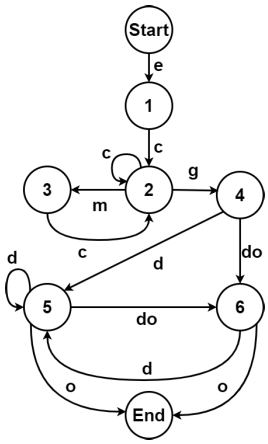


FIGURE 2. Finite state machine (FSM). The labels denote the types of layers: embedding (e), dropout (do), convolutional (c), max-pooling (m), global max-pooling (g), dense (d), and output (o).

3) CONSTRUCTION OF FSM

We constructed a finite state machine (FSM) to generate the architecture automatically, as shown in Fig. 2. The architecture starts with an embedding layer and must have at least one convolutional layer, as required for a CNN classifier. A max-pooling or global max-pooling layer can then be added. The global max-pooling layer acts as an intermediary layer between the top and bottom layers, i.e., it is a flattened layer that converts the output from the convolutional layer into a compatible input for the dense layer. Therefore, the last convolutional layer must be followed by a global max-pooling layer. A dense or dropout layer may or may not be added to the top layers. The architecture ends with an output layer consisting of a one-neuron dense layer with the sigmoid activation function and binary cross-entropy loss function.

B. PHASE II

Phase II involves the execution of the DGGA-CNN. The logical flow of this phase is illustrated in Fig. 3. The main processes of the DGGA-CNN consist of population initialization, diversity-guided mechanism, crossover, mutation, evaluation, and selection. The parameter settings of phase II are defined in Table 2. The low settings for *PopSize* and *Ngen* were chosen to counteract the high cost of computation of the

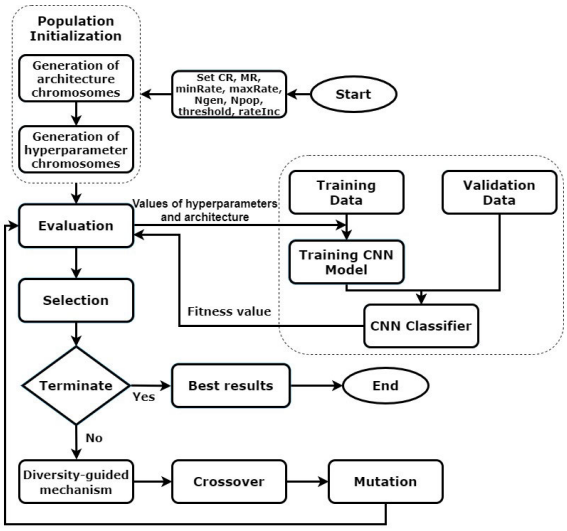


FIGURE 3. Logical flow of the DGGA-CNN algorithm.

TABLE 2. Parameter settings.

Parameter	Description	Value
CR	Crossover rate	0.8
MR	Mutation rate	0.2
minRate	Minimum rate	0.01
maxRate	Maximum rate	1.0
Ngen	Number of generations	100
PopSize	Population size	30
θ	Threshold of average distance	0.5
UR	Update rate	0.01
pctIndiv	Percentage of injected individuals	10%

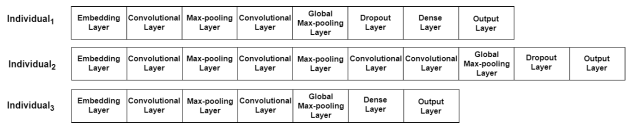


FIGURE 4. Example of architecture chromosomes.

DGGA-CNN. These low settings were also used in several studies [12], [15]–[17], [20] without any degradation in the performance. The values of CR and MR were selected based on recommendations from several papers [12], [15], [20]. The output of this phase was the best combination of CNN hyperparameters and architecture.

1) POPULATION INITIALIZATION

Population initialization involves architecture and hyperparameter chromosomes. The distinction was made due to the different ways of initialization. Architecture chromosomes were initialized for each individual using the FSM in Fig. 2. We do not limit the depth of CNN architecture; instead, its depth is constructed randomly through the initialization process, which starts from the embedding layer and ends in the output layer. Consequently, the depth can be different for each individual, as exemplified in Fig. 4.

After generating the architecture chromosomes, we define the hyperparameter chromosomes. To simplify the crossover and mutation processes, we equate the length of the hyperparameter chromosomes of all the individuals. The chromosome length is defined by using the maximum number of convolutional layers (MNC), max-pooling layers (MNM), dense layers (MND), and dropout layers (MNDO) obtained from the architecture of the first population.

For example, from Fig. 4, we can extract the maximum number of each of the layers: MNC is 4, MNM is 2, MND is 1, and MNDO is 1. These maximum values are then used to define the chromosome length for all the individuals by using Fig. 5 as a reference.

TABLE 3. Types of the DGGA-CNN algorithm.

	DGGA-CNN1	DGGA-CNN2	DGGA-CNN3
$\overline{H}>\theta$	CR++ MR- -	CR- - MR- -	-
$\overline{H}<\theta$	CR- - MR++	CR++ MR++	Random injection

2) DIVERSITY-GUIDED MECHANISM

We employed two types of diversity control strategies, namely, adaptive parameter control and random injection. Adaptive parameter control involves increasing or decreasing the crossover and mutation rates based on the present situation and random injection operates by replacing the worse individuals with newly generated ones when the algorithm needs to explore further. The circumstances for applying these strategies must be identified, i.e., we must define the case where the algorithm switches to exploration or exploitation. Therefore, we utilized an average pairwise Hamming distance to measure the diversity within the population. This measurement was based on the average distance between individual pairs in a population, as shown in (1).

$$\overline{H} = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_h(x_i, x_j)}{\binom{n}{2}} \tag{1}$$

where, x is an individual, $d_h(x_i, x_j)$ is the Hamming distance between a pair of individuals, and n is the number of individuals.

To guide the alternation between exploration and exploitation, we set a threshold (θ) as the upper limit of significant difference in the average diversity. If $\overline{H}$ of the previous generation attains the value θ, it means the diversity has declined, and the algorithm needs to explore further so that the population can be more diverse. Otherwise, the algorithm must stress on exploitation.

Harmony between exploration and exploitation is necessary to drive the algorithm towards better results. We compared three types of DGGA-CNN, namely, DGGA-CNN1, DGGA-CNN2, and DGGA-CNN3, to establish which strategy was better. The differentiation among these DGGA-CNNs is detailed in Table 3.

In Section II-C, we defined two notable aspects concerning the difference between the use of crossover

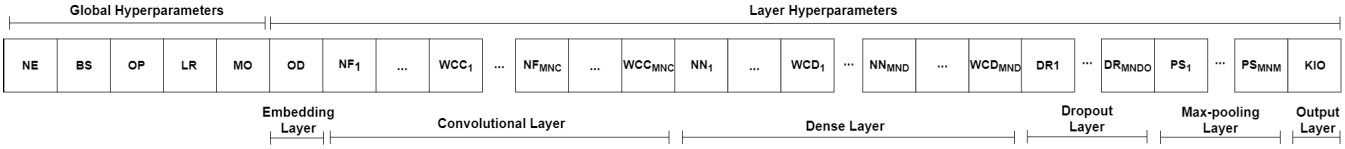


FIGURE 5. Initialization of population of hyperparameter chromosomes of the DGGA-CNN algorithm.

Algorithm 1 DGGA-CNN1 and DGGA-CNN2

Input:

· Average pairwise Hamming distance from the previous generation $\overline{H}$
· Threshold of average distance θ
· Previous crossover rate CR
· Previous mutation rate MR
· The minimum value of crossover and mutation rates $minRate$
· The maximum value of crossover and mutation rates $maxRate$
· Update rate UR

Output:

· New crossover rate CR
· New mutation rate MR

if $\overline{H} > \theta$ **then**
  if DGGA-CNN1 **then**
    $CR \leftarrow CR + UR$
  else if DGGA-CNN2 **then**
    $CR \leftarrow CR - UR$
  end if
  $MR \leftarrow MR - UR$
else
  if DGGA-CNN1 **then**
    $CR \leftarrow CR - UR$
  else if DGGA-CNN2 **then**
    $CR \leftarrow CR + UR$
  end if
  $MR \leftarrow MR + UR$
end if
if $CR > maxRate$ **then**
  $CR \leftarrow maxRate$
else if $CR < minRate$ **then**
  $CR \leftarrow minRate$
end if
if $MR > maxRate$ **then**
  $MR \leftarrow maxRate$
else if $MR < minRate$ **then**
  $MR \leftarrow minRate$
end if

Algorithm 2 DGGA-CNN3

Input:

· Average pairwise Hamming distance from the previous generation $\overline{H}$
· Threshold of average distance θ
· Percentage of injected individuals $pctIndiv$
· Population Size $PopSize$
· Previous population pop

Output:

· New population pop

$n = pctIndiv * PopSize$
if $\overline{H} < \theta$ **then**
  $newPop \leftarrow$ generated n new individuals
  Calculate the fitness values of $newPop$.
  Change the worst n individuals in pop with the $newPop$.
end if

and mutation. We differentiate between DGGA-CNN1 and DGGA-CNN2 based on these, as summarized in Algorithm 1. The DGGA-CNN1 utilizes opposite trends of increment or decrement of CR and MR values, i.e., the CR increases if $\overline{H} > \theta$ and vice versa and the MR decreases if $\overline{H} > \theta$ and vice versa. Meanwhile, the DGGA-CNN2 executes a simultaneous variation in the values of CR and MR, i.e., the CR and MR decrease if $\overline{H} > \theta$ and vice versa. For both the DGGA-CNNs, the initial values of CR and MR were 0.8 and 0.2, respectively. These values were selected based on previous studies [12], [15], [20]. The increment and decrement rate (UR) for CR and MR values were both 0.01. Moreover, to prevent invalid rates, a minimum rate (minRate) and maximum rate (maxRate) were utilized for the CR as well as MR.

The procedure for DGGA-CNN3 is shown in Algorithm 2. This is a GA-based CNN optimization method that utilizes a random injection strategy that can be easily implemented in the GA. There are five important points to be considered: First, the new individuals are generated randomly, as in the initialization of population. Second, the number of new individuals is only ten percent of the population size (pctIndiv). Third, individuals with the lowest ten percent of fitness values are replaced by new individuals. Fourth, the random injection is only executed if $\overline{H}$ of the prior population is lower than θ. Finally, the random injection is only implemented on hyperparameter chromosomes.

3) CROSSOVER

Crossover requires two parent individuals to produce offspring. The individuals were selected randomly from the current population based on the fitness function. To determine whether this pair of individuals will undergo crossover, a random float value is generated and compared with the

Algorithm 3 Crossover

Input:
- Selected parent individuals *ind1, ind2*
- A random integer between 0 and 2 *randInt*
- A random float between 0 and 1 *randFloat*
- Crossover rate *CR*

Output:
- Offspring individuals *offspring1, offspring2*

if *randFloat* $<$ *CR* **then**
  Select the hyperparameter chromosomes of *ind1* and *ind2*.
  if *randInt* $= 0$ **then**
    $offspring1, offspring2 \leftarrow OnePointCrossover(ind1, ind2)$
  else if *randInt* $= 1$ **then**
    $offspring1, offspring2 \leftarrow TwoPointCrossover(ind1, ind2)$
  else if *randInt* $= 2$ **then**
    $offspring1, offspring2 \leftarrow UniformCrossover(ind1, ind2)$
  end if
  Select the architecture chromosomes of *ind1* and *ind2*.
  Exchange the architecture chromosomes of *ind1* and *ind2* using one-point crossover.
  Calculate the fitness values for *offspring1* and *offspring2*.
end if

CR value. If the random value is smaller, then a crossover will be performed.

The crossover operator performs differently for different types of chromosomes. Algorithm 3 shows the details of the crossover operator. For the hyperparameter chromosomes, we utilized three types of crossovers, namely, one-point, two-point, and uniform crossovers. The crossover type was selected randomly to produce diverse offspring. The cutting point for the one-point and two-point crossovers was also selected randomly within the chromosome length. In the case of architecture chromosomes, we applied the one-point crossover. The cutting point was not selected randomly; instead, we determined it beforehand to ensure valid architectures of the resulting offspring. For each individual, the cutting point was selected after the global max-pooling layer. Fig. 6 shows an example of the one-point crossover that was utilized for architecture chromosomes.

4) MUTATION

Mutation generates a new solution from the offspring produced by the crossover operator. Before mutating the offspring, a random float value is generated and then compared with the MR. The offspring will be mutated if the generated random float value is less than the MR.

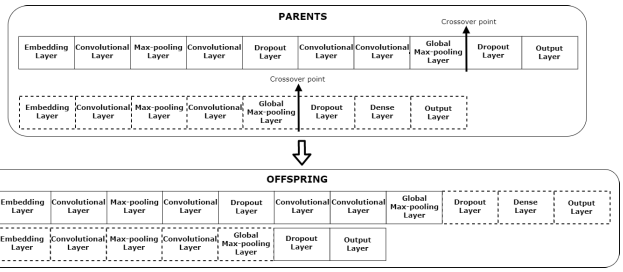


FIGURE 6. Example of one-point crossover for architecture chromosomes.

Algorithm 4 Mutation

Input:
- Selected individual *ind*
- Mutation rate *MR*
- A random integer between 0 and 2 *randInt*
- A random float between 0 and 1 *randFloat*

Output:
- Offspring *offspring*

if *randFloat* $<$ *MR* **then**
  hyperChromInd $\leftarrow$ *hyperparameter chromosomes of ind*
  $offspring \leftarrow UniformMutation(hyperChromInd)$
  arcChromInd $\leftarrow$ *architecture chromosomes of ind*
  if *randInt* $= 0$ **then**
    $offspring \leftarrow EliminationMutation(arcChromInd)$
  else if *randInt* $= 1$ **then**
    $offspring \leftarrow ExchangeMutation(arcChromInd)$
  else if *randInt* $= 2$ **then**
    $offspring \leftarrow InsertionMutation(arcChromInd)$
  end if
end if
Calculate the fitness value for the *offspring*.

The mutation operator is applicable to both types of chromosomes. Algorithm 4 shows the details of the mutation operator. For hyperparameter chromosomes, we utilize uniform mutation that works by selecting random values within the range or list for the selected gene of the selected individual.

We created three types of mutation mechanisms for the architecture chromosomes. These are (1) removing a layer (elimination mutation), (2) changing a layer (exchange mutation), and (3) adding a layer (insertion mutation). Four CNN layer types undergo architecture mutation, namely, the convolutional layer, max-pooling layer, dense layer, and dropout layer. The embedding layer, global max-pooling layer, and output layer cannot be removed, changed, or added because the architecture would become invalid. The mutation type was selected randomly to improve diversity. Moreover, when applying the mutation, we ensured that the number of layers did not exceed the maximum layers specified in the architecture of the initial population.

Though the key idea of the three types of mutation is simple and understandable, there is one issue to be addressed for

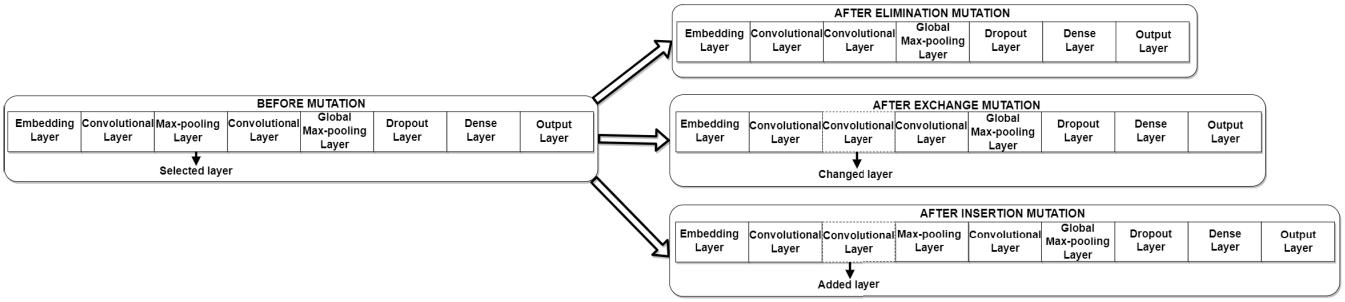


FIGURE 7. Example of mutation for architecture chromosomes.

TABLE 4. Elimination mutation.

Selected Layer	Previous Layer	Following Layer
Convolutional layer	Convolutional layer	Convolutional layer
	Convolutional layer	Max-pooling layer
	Max-pooling layer	Convolutional layer
	Convolutional layer	Global max-pooling layer
Max-pooling layer	Convolutional layer	Convolutional layer
Dense layer	Global max-pooling layer	Dense layer
	Global max-pooling layer	Dropout layer
	Dense layer	Dense layer
	Dense layer	Dropout layer
	Dropout layer	Dense layer
	Dense layer	Output layer
	Dropout layer	Output layer
Dropout layer	Global max-pooling layer	Dense layer
	Global max-pooling layer	Dropout layer
	Dense layer	Dense layer
	Dense layer	Output layer

TABLE 5. Exchange mutation.

Selected Layer	Previous Layer	Following Layer	Substitute Layer
Convolutional layer	Convolutional layer	Convolutional layer	Max-pooling layer
Max-pooling layer	Convolutional layer	Convolutional layer	Convolutional layer
Dense layer	Global Max-pooling layer Dense layer	Dense layer	Dropout layer
Dropout layer	Global Max-pooling layer Dense layer	Dense layer	Dense layer

TABLE 6. Insertion mutation.

Selected Layer	Previous Layer	Additional Layer
Convolutional layer	Convolutional layer	Convolutional layer Max-pooling layer
Max-pooling layer	Convolutional layer	Convolutional layer
Dense layer	Global Max-pooling layer Dense layer	Dense layer Dropout layer
Dropout layer	Global Max-pooling layer Dense layer	Dense layer

each type, i.e., to define a procedure so that the mutation process does not produce an invalid CNN architecture. This means we need to identify the circumstance in which a layer can be removed, changed, or added. Therefore, as part of the mutation process, we employed several rules to ensure a valid mutated architecture. We extracted the rules from the FSM in Fig. 2 and prepared Table 4 for the elimination mutation, Table 5 for the exchange mutation, and Table 6 for the insertion mutation.

In elimination mutation, the selected layer can be removed if the previous and following layers satisfy the condition in Table 4, i.e., the algorithm checks whether the surrounding layer would still be valid if the selected layer is removed. For example, in Fig. 7, the selected layer is the max-pooling layer enclosed by convolutional layers. If the max-pooling layer is removed, then the architecture is still valid as the previous convolutional layer can be connected with the other convolutional layer.

In exchange mutation, the selected layer can be changed to a substitute layer only if the previous and following layers satisfy the condition in Table 5. For example, in Fig. 7, the max-pooling layer can be changed to the substitute layer only if the previous and following layers are convolutional layers. Therefore, the architecture remains valid when the max-pooling layer is changed to the convolutional layer.

In insertion mutation, a layer can be inserted before the selected layer only if the selected and previous layers satisfy the condition in Table 6. For example, in Fig. 7, a convolutional layer can be added before the max-pooling layer if the previous layer is a convolutional layer. Therefore, the architecture is still valid when a convolutional layer is inserted before the max-pooling layer.

5) EVALUATION

The DGGA-CNN is driven by a fitness function that calculates the closeness of the solution to the objective. The calculated result then decides whether the solution can be used for the next generation or not. As the fitness function, we used the F1-Score for the suggestion mining dataset and the accuracy for the twitter airline dataset. The evaluation process starts by forwarding the produced hyperparameters and architecture to the CNN. After the training, the fitness value is returned. If the fitness value of an individual is high, it has a higher chance of being added to the next generation.

6) SELECTION

We used elitism selection that works by sorting the fitness values of all the parent and offspring individuals. Then, the individuals are selected to the top 30 (*PopSize*) for the next generation.

C. PHASE III

In this phase, the best hyperparameters and architecture from phase II are used to train the CNN model. The details of this phase are illustrated in Fig. 8. After constructing the CNN

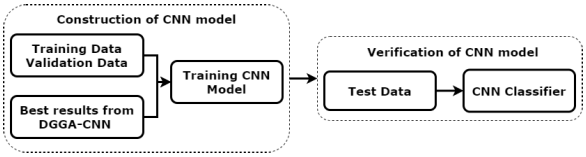


FIGURE 8. Phase III.

TABLE 7. Distribution of datasets.

Suggestion Mining Dataset				
Class	Training	Validation	Test	Total
Non-Suggestion	6386	700	1222	8308
Suggestion	2114	700	435	3249
All Classes	8500	1400	1657	11557
Twitter Airline Dataset				
Class	Training	Validation	Test	Total
Negative	5857	1432	1889	9178
Positive-Neutral	3512	911	1039	5462
All Classes	9369	2343	2928	14640

model using the training and validation data, the model was verified using test data.

IV. RESULTS AND DISCUSSION

A. TOOLS AND LIBRARIES

All the methods were implemented in the Python programming language. The metrics were calculated using SciKit-Learn [6]. Our data preprocessing was carried out on NLTK [60] and Pandas [61]. The Wilcoxon signed-rank test and the Spearman rank correlation coefficient were conducted using the SciPy library [62]. For correction of spellings, we made use of TextBlob [63]. To build the deep learning model, we utilized Keras [64] and Tensorflow [65]. The GA was created using DEAP [21] and we carried out all our experiments on GPU Titan V. The source code of our approach is available on GitHub.[1]

B. DATASET

To evaluate the effects of the DGGA-CNN for text classification, we used the suggestion mining dataset [58] and the twitter airline dataset [66]. The distribution of the datasets is given in Table 7. For the suggestion mining dataset, we used all subtasks of their trial data as validation. We also combined two subtasks of their evaluation data for test data to comprehensively evaluate our proposed solution. Meanwhile, the twitter airline dataset was divided into training, validation, and testing with a ratio of 64:16:20. We used *random_state* of 42 to determine the number of data for each class in training, validation, and testing. This dataset is binarized by considering two classes: negative and merged positive-neutral sentiments.

It is important to mention that to tackle overfitting, we utilized a class weighting, an early stopping mechanism to halt the training of the CNN with *patience* of 10, and a model checkpoint strategy to save the best model throughout epochs.

[1]https://github.com/fatyanosa/DGGA-CNN

TABLE 8. Hyperparameter value sets of grid search.

Hyperparameter	Set of values
NE	[20, 40, 60, 80, 100]
BS	[32, 64, 128, 256]
OP	[‘adam’, ‘nadam’, ‘adamax’, ‘sgd’, ‘rmsprop’]
LR	[1e-4, 1e-3]
OD	[100, 200, 300]
AFC	[‘relu’, ‘elu’, ‘selu’, ‘tanh’, ‘sigmoid’]

C. CNN OPTIMIZATION

In this section, we examine our approach by performing three comparisons. We compare our DGGA-CNNs with (1) uninformed searches such as grid and random searches; (2) other GA-based CNN optimizations such as standard GA (GA-CNN) [20] and fitness-guided adaptive GA (AGA-CNN) [51], [67], [68]; (3) existing automatic machine learning (autoML) packages such as auto-sklearn [69] and TPOT [70].

Different settings were performed for grid search owing to time-consuming characteristics [34]. We limited the number of hyperparameters to six and used the defined sets of values, as shown in Table 8. The grid search used a fixed CNN layer order, i.e., input layer, embedding layer, convolutional layer, global max-pooling layer, dense layer, dropout layer, and output layer. The random search used the same range and list of hyperparameters as the DGGA-CNN, and its architecture was generated using our FSM as constructed in Fig. 2.

To ensure a fair comparison, we used two types of termination that were based on (1) the time limit and (2) number of executed CNNs. The termination based on the time limit stops all methods within 1, 2, and 3 h. This type of termination was executed to compare the CNN optimization methods, namely, grid search, random search, GA-CNN, AGA-CNN, DGGA-CNN1, DGGA-CNN2, and DGGA-CNN3, and non-CNN optimization methods, namely, auto-sklearn and TPOT. The default parameters were used for auto-sklearn except for *time_left_for_this_task*, which was set to 3600, 7200, and 10800 s, *metric* to a custom scorer of F1-Score for the suggestion mining dataset, and *metric* to *accuracy* for the twitter airline dataset. TPOT also used the default parameters except for *max_time_mins*, which was set to 60, 120, and 180 m, *cv* to the predetermined training and validation data, *scoring* parameter to *f1* for the suggestion mining dataset, and *scoring* parameter to *accuracy* for the twitter airline dataset.

The termination based on the number of executed CNNs performed the same number of CNNs, i.e., 3000. The number of executed CNNs for grid search was obtained from the multiplication of the NE, BS, OP, LR, OD, and AFC sets, i.e., $(5 * 4 * 5 * 2 * 3 * 5) = 3000$. The random search was run for 3000 iterations. The number of executed CNNs for all the GA-based CNN optimization methods was the multiplication of *Ngen* and *PopSize*, i.e., $(100 * 30) = 3000$. It is important to highlight that the number of evaluations for all the GA-based CNN optimization methods, theoretically, is equal to *Ngen*PopSize* and practically, is less than or equal

TABLE 9. Results of performance of hyperparameter optimization methods based on time limits.

Method	Suggestion mining						Twitter airline					
	Best			Average			Best			Average		
	1 h	2 h	3 h	1 h	2 h	3 h	1 h	2 h	3 h	1 h	2 h	3 h
Auto-Sklearn	0.4572	0.4639	0.4641	0.4291	0.4451	0.4454	0.7435	0.7452	0.7435	0.7379	0.7362	0.7345
TPOT	0.6945	0.6958	0.6958	0.6834	0.6896	0.6883	0.7499	0.7503	0.7469	0.7446	0.7452	0.7449
Grid search	0.7201	0.7205	0.7234	0.7095	0.7153	0.7209	0.8361	0.8361	0.8361	0.8327	0.8333	0.8333
Random search	0.7319	0.7319	0.7327	0.7224	0.7227	0.7252	0.8421	0.8425	0.8425	0.8405	0.8414	0.8414
GA-CNN	0.7338	0.7397	0.7430	0.7249	0.7328	0.7360	0.8451	0.8464	0.8485	0.8416	0.8442	0.8457
AGA-CNN	0.7412	0.7518	0.7518	0.7276	0.7393	0.7400	0.8442	0.8459	0.8481	0.8420	0.8434	0.8448
DGGA-CNN1	0.7440	**0.7522**	0.7522	0.7295	0.7419	0.7459	0.8434	0.8476	0.8476	0.8429	0.8446	0.8452
DGGA-CNN2	0.7423	**0.7522**	0.7566	**0.7373**	**0.7467**	**0.7487**	**0.8459**	**0.8493**	**0.8493**	**0.8436**	**0.8458**	**0.8470**
DGGA-CNN3	**0.7510**	0.7515	**0.7586**	**0.7373**	0.7393	0.7442	0.8446	0.8446	0.8489	0.8417	0.8436	0.8459

to *Ngen*PopSize*. To our best knowledge, there is no way to know the exact number of evaluations, and a different experiment may have a different number of evaluations. Therefore, we only consider the worst-case evaluation cost and assume all the methods have the same number of evaluations. Owing to the different classifiers and optimization algorithms, it was impractical to compare the CNN optimization methods with auto-sklearn and TPOT using this termination; therefore, this was executed for an in-depth comparison of the CNN optimization methods. It is important to note that we would have preferred to run all the approaches several times owing to the non-deterministic optimization behavior; however, considering the expense of computation, we ran them only 5 times.

1) COMPARISON OF FITNESS VALUES BASED ON TIME LIMIT

Table 9 compares the performance of all the hyperparameter optimization methods within 1, 2, and 3 h. The DGGA-CNNs were able to locate powerful CNNs that achieved the highest best and average fitness values in all the time limits for the two datasets. These results demonstrated that our DGGA-CNNs can be used within low time limits without compromising on the performance.

Moreover, from the results, it is clear that there are trade-offs between the time taken and level of performance of the methods. Along with the addition of time limits, the fitness values increased monotonically. This indicates that if more time is spent, then more effective hyperparameters and architecture are obtained. Further, the population diversity may still be greater than θ within low time limits, and thus the DGGA-CNNs did not yet apply the scheme when the diversity was below θ. More execution time is needed to find out whether the scheme is successful or not. Therefore, the next subsections discuss the comparisons among the CNN optimization methods with the same number of executed CNNs that take a longer time.

2) COMPARISON OF FITNESS VALUES BASED ON THE NUMBER OF EXECUTED CNNs

Table 10 shows a comparison of the best and average fitness values using the training and validation data within the same number of executed CNNs. The best and average fitness values of all the CNN optimization methods were obtained from all the 5 runs.

TABLE 10. Results of performance of CNN optimization methods based on the number of executed CNNs.

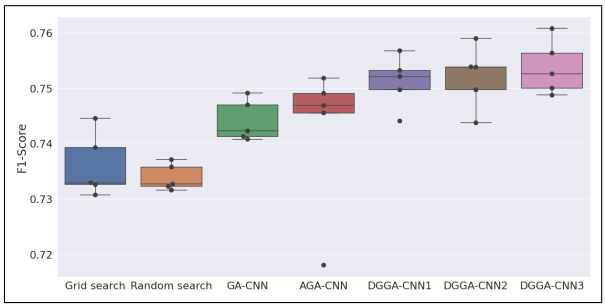
Method	Suggestion mining		Twitter airline	
	Best	Average	Best	Average
Grid search	0.7445	0.7360	0.8391	0.8372
Random search	0.7371	0.7339	0.8472	0.8449
GA-CNN	0.7491	0.7448	0.8498	0.8484
AGA-CNN	0.7518	0.7423	0.8481	0.8455
DGGA-CNN1	0.7568	0.7512	0.8502	0.8476
DGGA-CNN2	0.7590	0.7520	**0.8515**	**0.8498**
DGGA-CNN3	**0.7608**	**0.7537**	0.8506	0.8493

A comparison between the uninformed and informed searches shows that the GA-based CNN optimization methods perform better than the grid and random searches, which proves the superiority of informed search over uninformed search. A possible explanation of this outcome is that an uninformed search only has access to the problem statement without considering previous solutions whereas an informed search has access to both. An access to prior solutions means that the informed search can learn how to choose the best solution based on the results of the previous generation and this helps in improving the results in the next generation.

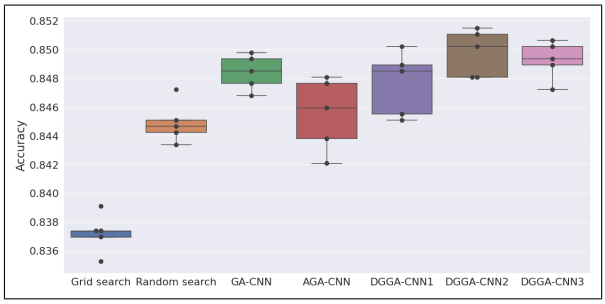
From the boxplots in Fig. 9, it is clear that after applying adaptive parameter control and random injection, there was an improvement in the fitness value, which indicates that these methods are reliable to lead the search into the appropriate search areas. If the DGGA-CNN focuses on a particular area, the frequency of searching that area may become very large and the search area may become limited; however, if it focuses on a few areas, it may fail to locate the global optimum. Therefore, the application of adaptive parameter control and random injection offers a search strategy to obtain higher fitness values.

A comparison between the three DGGA-CNNs shows that DGGA-CNN3 and DGGA-CNN2 deliver promising results for the suggestion mining and twitter airline datasets, respectively. It is also evident from Fig. 9 that the DGGA-CNN2 and DGGA-CNN3 perform better, with higher interquartile ranges.

Even though our DGGA-CNNs show improvement for the two datasets, the best type is different. One implication of this is that the characteristics of datasets may influence the most appropriate one. This conjecture is supported by research



(a) Suggestion mining



(b) Twitter airline

FIGURE 9. Boxplots of the best fitness values of the CNN optimization methods for 5 runs for each dataset.

from [71] which suggests the need to consider the characteristics of datasets when choosing a classification algorithm. However, this would need further validation, and thus, this is left for future work.

The comparison of the fitness values based on the number of executed CNNs leads to three important conclusions: (1) Informed searches, i.e., GA-based CNN optimization methods, are superior to uninformed searches, i.e., grid search and random search. (2) The DGGA-CNNs outperform the GA-CNN and AGA-CNN. (3) The DGGA-CNN3 and DGGA-CNN2 yield better results for the suggestion mining and twitter airline datasets, respectively.

3) COMPARISON OF EFFICIENCY

In this section, we evaluate the efficiency of all the methods of optimization of hyperparameters by comparing their average execution time, as shown in Table 11. The average execution time reflects the time needed to finish 3000 CNNs in all the 5 runs. Although all the methods executed the same number of CNNs, the average execution times were different. The results in Table 11 demonstrate two conclusions: (1) the grid search needs a higher execution time than the other methods and (2) the DGGA-CNNs require a relatively high execution time. However, it is promising that the results of the DGGA-CNNs can be improved upon by utilizing adaptive parameter control and random injection.

A comparison of the efficiency showed that uninformed searches were not efficient for hyperparameter optimization as they generally needed an excessive amount of time but were unable to exhibit higher performance. The GA-based CNN optimization methods were better as they found higher fitness values within the same number of executed CNNs.

TABLE 11. Average execution time of CNN optimization methods based on the number of executed CNNs.

Method	Suggestion mining	Twitter airline
Grid search	30.43 h	49.97 h
Random search	25.22 h	27.93 h
GA-CNN	13.58 h	30.45 h
AGA-CNN	6.9 h	10.40 h
DGGA-CNN1	19.23 h	33.90 h
DGGA-CNN2	20.04 h	36.92 h
DGGA-CNN3	16.04 h	39.20 h

4) COMPARISON OF THE BEST CNN HYPERPARAMETERS AND ARCHITECTURE

A representation of the selected architectures of all the CNN optimization methods is given in Fig. 10. All the resulting CNN architectures are quite small and straightforward and consist of 6–8 layers with mainly 1 convolutional layer, 1 dropout layer, and 1–3 dense layers. The number of hyperparameters is also relatively small for the two datasets, which only optimized 6 to 24 hyperparameters (6-17 types of hyperparameters). Even so, a smaller architecture is known for its efficiency [72] and is applicable for a small dataset, and acceptable results can still be obtained as the CNN has the translation invariance property and learns locally [73]. The resulting hyperparameters and architecture demonstrated that small architectures are appropriate to classify the two datasets.

Moreover, we compared the largest number of layers for each layer type in each generation of the best run, as shown in Fig. 11 and Fig. 12. This comparison aims to see the advantages of using the FSM and three novel mutation mechanisms for architecture chromosomes. Towards the end of the generation, using the suggestion mining dataset, all the final populations of the GA-based CNN optimization methods have 1 convolutional layer, 0 max-pooling layers, 1-4 dense layers, and 0-1 dropout layer. Meanwhile, for the twitter airline dataset, all the final populations have 1 convolutional layer, 0 max-pooling layers, 0 dense layers, and 1 dropout layer. The striking difference between the two datasets is the use of the dense layer. In suggestion mining dataset, the structures of the final populations are mainly made of dense layers. Meanwhile, in the twitter airline dataset, all the GA-based CNN optimization methods did not use dense layers at the final architecture. Furthermore, it can be seen from the fluctuation that the three mutation mechanisms play an important role in increasing and decreasing the number of layers.

5) COMPARISON OF HANDLING OF PREMATURE CONVERGENCE

It can be seen from Fig. 13a and 13b that most of the grid and random search solutions produce fitness values of 0.0 and 0.6667 for the suggestion mining dataset. Meanwhile, as shown in Fig. 14a and 14b, both grid and random searches mostly fall into a fitness value of 0.6112 using the twitter airline dataset. Apart from that, no other patterns could

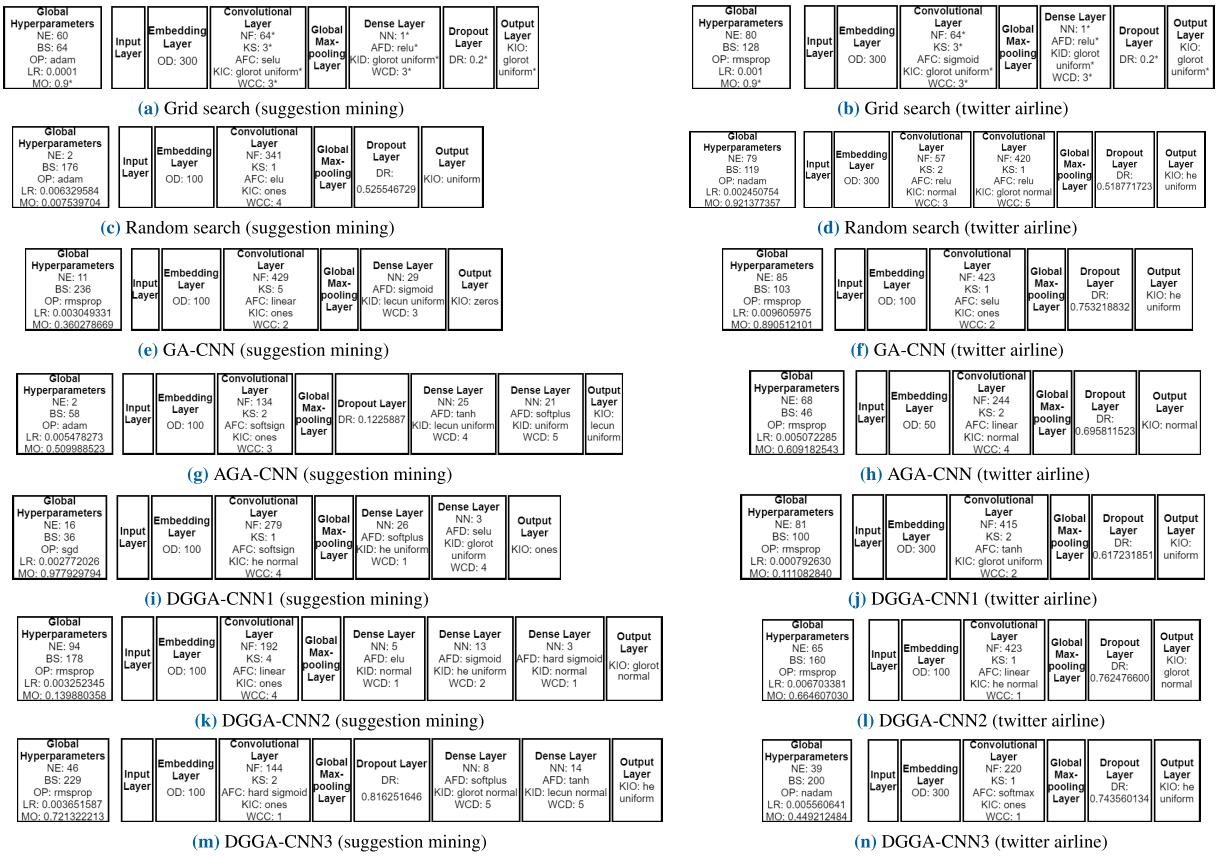
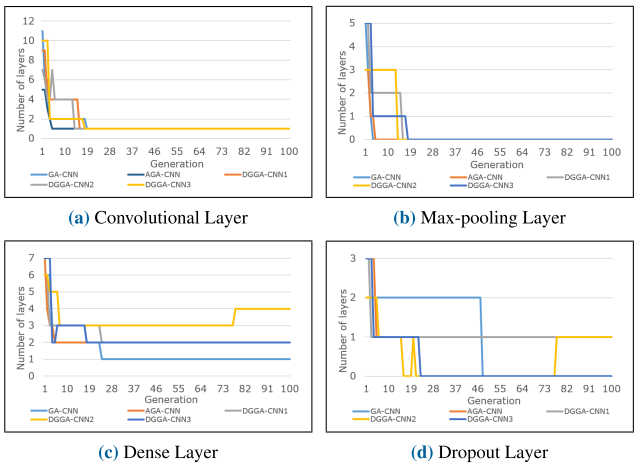


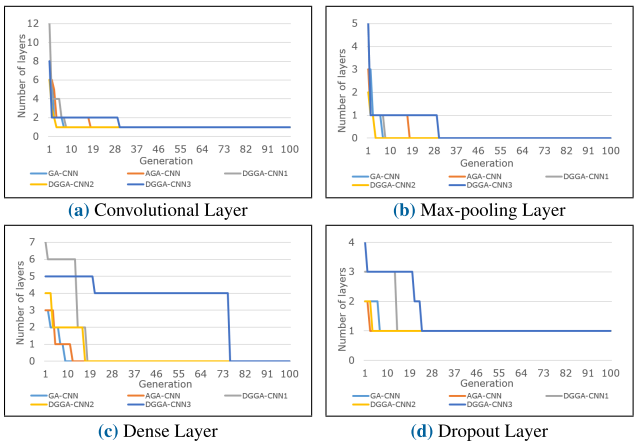
FIGURE 10. Best hyperparameter values and architecture of each CNN optimization method.



(a) Convolutional Layer (b) Max-pooling Layer

(c) Dense Layer (d) Dropout Layer

FIGURE 11. Comparison of the largest number of layers in each generation of the best run for the suggestion mining dataset.



(a) Convolutional Layer (b) Max-pooling Layer

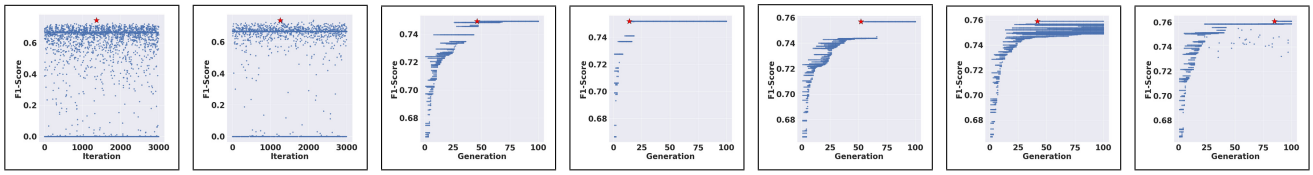
(c) Dense Layer (d) Dropout Layer

FIGURE 12. Comparison of the largest number of layers in each generation of the best run for the twitter airline dataset.

be analyzed from the fitness values of grid search or random search as each solution was independent. This is due to the absence of assistance from previous iterations, thus causing the algorithms to fall repeatedly to the local optimums.

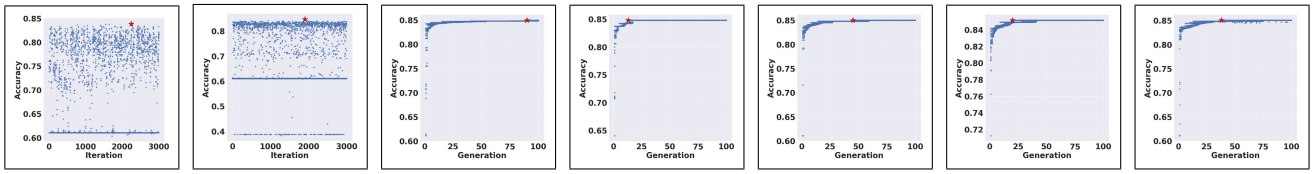
Fig. 13 and Fig. 14 also show the fitness value transformations for the GA-based CNN optimization methods in every generation. Since the trend is pretty much the same for all the 5 runs, we show only the transformations of the best run. The fitness values of all the GA-based CNN optimization methods have a rising trend due to their ability to learn from previous generations, which is advantageous to keep track of the search area. Eventually, they have a higher possibility of obtaining better results in later generations, as shown in Table 10.

The ability to learn from previous generations helps to obtain better results, but if carried out continuously, it can bring about uniformity in all the individuals and



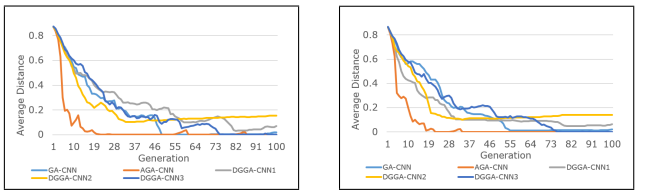
(a) Grid search (b) Random search (c) GA-CNN (d) AGA-CNN (e) DGGA-CNN1 (f) DGGA-CNN2 (g) DGGA-CNN3

FIGURE 13. Fitness value transformations throughout the generations of all the CNN optimization methods for the suggestion mining dataset.



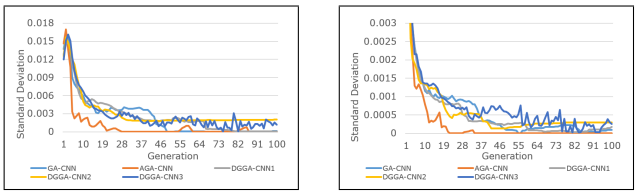
(a) Grid search (b) Random search (c) GA-CNN (d) AGA-CNN (e) DGGA-CNN1 (f) DGGA-CNN2 (g) DGGA-CNN3

FIGURE 14. Fitness value transformations throughout the generations of all the CNN optimization methods for the twitter airline dataset.



(a) Suggestion mining (b) Twitter airline

FIGURE 15. Comparison of the average distance of every generation of all the 5 runs.



(a) Suggestion mining (b) Twitter airline

FIGURE 16. Comparison of the average standard deviation of every generation of all the 5 runs.

cause premature convergence. This is seen in the results of the AGA-CNN. Fig. 13d and 14d demonstrates that the AGA-CNN converged quickly from the 14th and 13th generations for the suggestion mining and twitter airline datasets, respectively. The fitness-guided adaptive CR and MR were unable to assist the AGA-CNN to break free from the local optimum. A rationale behind this is that the AGA-CNN adjusts the CR and MR based on the fitness value of each individual that makes the algorithm focus more on exploitation than exploration. In addition, when a population in the AGA-CNN converges, the values of CR and MR are zero, which caused no individual to undergo crossover and mutation; thus, the population remained stagnant. Our experimental results revealed that the AGA-CNN tended to suffer from premature convergence and was stuck in a local optimum.

On the contrary, most of the DGGA-CNNs were able to escape from local optimum and avoid premature convergence as shown in Fig. 13 and 14. Moreover, from Fig. 15, it is seen that the DGGA-CNN1 and DGGA-CNN2 have the highest diversity at the end of the generation and were able to keep the diversity from becoming zero. Even though the diversity dropped dramatically in the early generations, the adaptive parameter control was able to help DGGA-CNN1 and DGGA-CNN2 to stabilize and even increase the diversity after the decline. Fig. 16 also shows that most of the DGGA-CNNs have a higher standard deviation than the GA-CNN and AGA-CNN, which indicates their high variation among individuals. This high variation allows the algorithms to escape from local optimum and proves that they can avoid premature convergence until the end of the generation.

A comparison between the two DGGA-CNNs with adaptive parameter control shows that DGGA-CNN2 outperforms DGGA-CNN1 in terms of performance and population diversity. It appears possible that the nature of the GA may cause this superiority. Typically, the GA population starts with exploration and gradually transforms into exploitation [27]. In the beginning, when the population is more diverse, decreasing the CR and MR can help the algorithm to search for potential individuals near the current solution. If the CR and MR are increased in this phase, there is a possibility that the crossover and mutation operators disrupt the process of finding near-optimum solutions [51]. Therefore, to avoid this, it is necessary to decrease the values of CR and MR. Later, when entering a phase that requires exploration, increasing the values of CR and MR help the algorithm to produce more diverse offspring faster [52] and explore other search spaces [74].

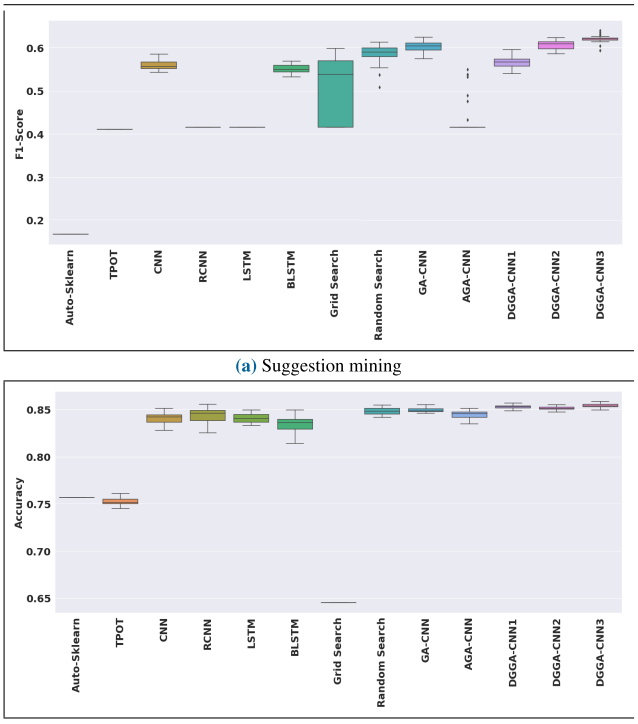
In DGGA-CNN3, the injected individuals replace the worse individuals, ensuring variations when diversity is low. The random injection provides exploration, whereas the current state contributes exploitation. Therefore, a proper balance between exploration and exploitation is attained. The best possible situation is when injected and old individuals

TABLE 12. Spearman rank correlation coefficient of the best run.

Method	Suggestion mining			Twitter airline		
	ρ	p	Result	ρ	p	Result
GA-CNN	-0.927	0.000	Correlated	-0.927	0.000	Correlated
AGA-CNN	-0.963	0.000	Correlated	-0.993	0.000	Correlated
DGGA-CNN1	-0.928	0.000	Correlated	-0.812	0.000	Correlated
DGGA-CNN2	0.310	0.002	Correlated	-0.766	0.000	Correlated
DGGA-CNN3	-0.796	0.000	Correlated	-0.958	0.000	Correlated

TABLE 13. Comparison of best and average fitness values of test data.

Method	Suggestion mining		Twitter airline	
	Best	Average	Best	Average
Auto-Sklearn	0.1667	0.1667	0.7568	0.7568
TPOT	0.4103	0.4103	0.7609	0.7524
CNN	0.5846	0.5595	0.8518	0.8454
RCNN	0.4159	0.4159	0.8555	0.8426
LSTM	0.4159	0.4159	0.8497	0.8408
BLSTM	0.5689	0.5503	0.8494	0.8340
Grid search	0.5978	0.5080	0.8501	0.6690
Random search	0.6130	0.5856	0.8548	0.8482
GA-CNN	0.6240	0.6017	0.8538	0.8487
AGA-CNN	0.5497	0.4373	0.8514	0.8445
DGGA-CNN1	0.5952	0.5659	0.8579	0.8528
DGGA-CNN2	0.6229	0.6063	0.8552	0.8513
DGGA-CNN3	**0.6393**	**0.6199**	**0.8596**	**0.8543**



(a) Suggestion mining

(b) Twitter airline

FIGURE 17. Comparison of boxplots of test data.

produce offspring through crossover and mutation. This eventually helps the DGGA-CNN3 to produce potential solutions with higher fitness values. Moreover, as shown in Fig. 13g and Fig. 14g, DGGA-CNN3 has many outliers scattered far away from other individuals. Outliers are caused by the possibility of lower or higher fitness values of the injected individuals resulting in some individuals being located very far as compared to others. However, outliers are part of the strategy as they allow the algorithm to explore other search spaces and obtain better results.

We also tested the relationship between average distance and average fitness value of the best run using the Spearman rank correlation coefficient with $\alpha = 0.05$. Table 12 shows the Spearman rank correlation coefficients (ρ) and probabilities (p) for all the GA-based CNN optimization. These results suggest that the average distance and average fitness value for each method are correlated. This indicates that diversity indeed affects the performance of the GA-based CNN optimization methods.

In conclusion, DGGA-CNNs are capable of handling premature convergence better than the other methods. The reason that the DGGA-CNNs are superior may be their ability to alternate between exploration and exploitation. In the DGGA-CNN1 and DGGA-CNN2, this alternation influences changes in the values of CR and MR, whereas in DGGA-CNN3, the alternation determines whether the method requires the injection of new individuals. This conclusion is consistent with the results found in Sections IV-C1 and IV-C2, thus underlining the advantage of DGGA-CNNs over auto-sklearn, TPOT, grid search, random search, GA-CNN, and AGA-CNN.

D. VERIFICATION OF CNN MODEL

We evaluated our proposed method using test data and executed it 30 times to further demonstrate its ability to classify unseen datasets. We used the best combination of hyperparameters and architecture of each CNN optimization method given in Fig. 10. Regarding auto-sklearn and TPOT, we implemented the best pipeline found within 3 h. We also compared our approach with other deep learning methods such as CNN, recurrent CNN (RCNN), LSTM, and bidirectional LSTM (BLSTM). Similar to our GA-based CNN optimization, all the deep learning methods utilized class weighting, early stopping, model checkpoint mechanisms, F1-Score metrics for the suggestion mining dataset, and accuracy metrics for the twitter airline dataset. The optimization of hyperparameters and architecture was not examined for the other deep learning methods mentioned above; instead, we followed the settings from the sentiment classification examples in [64]. This comparison aimed to perceive the benefits of automatic optimization over manual tuning. It is impractical to compare the results of the other submissions in [58] as this paper used multidomain datasets from two subtasks in the validation and test data for the suggestion mining dataset.

A comparison of the best scores given in Table 13 and boxplots in Fig. 17 show clearly that most of the DGGA-CNNs are better than the other methods. Superior results are exhibited by DGGA-CNN3 owing to higher fitness values, higher median, and higher interquartile range in both boxplots. The results of auto-sklearn, TPOT, and grid search are the worst.

We also evaluated the statistical significance of the methods by using a Wilcoxon signed-rank test with $\alpha = 0.05$. The Wilcoxon signed-rank test shows that all the DGGA-CNNs demonstrate statistically significant differences when paired with other methods. These results suggest the superiority of our DGGA-CNNs over other methods.

V. CONCLUSION AND FUTURE WORK

In this study, we proposed a new algorithm (DGGA-CNN) to optimize CNN hyperparameters and architecture using the diversity-guided GA. We employed three types of DGGA-CNN, namely, DGGA-CNN1, DGGA-CNN2, and DGGA-CNN3. There were several benefits of using DGGA-CNNs. First, they automatically handled the architecture of the CNN by using the novel FSM and three mutation mechanisms. Second, the classification using the suggestion mining and twitter airline datasets appeared to prove their superiority over other methods in terms of fitness values and handling of premature convergence. Lastly, the execution time and computational resources were used efficiently as compared to uninformed searches, namely, grid search and random search. To conclude, the use of DGGA to optimize CNN in text classification appears to be a promising approach, as highlighted by our experimental results.

Despite the promising performance of DGGA-CNN, further studies must be conducted. Firstly, we left for future work discussing which DGGA-CNN approach is better for given characteristics of datasets. Secondly, we optimized all the hyperparameters, including the unimportant hyperparameters; the important hyperparameters may have a higher contribution towards improvement in the performance [75], [76]. Therefore, a future extension of this work is an automatic selection of hyperparameters that are important to optimize for certain datasets. Thirdly, we deliberately defined wide ranges for several hyperparameters because of the problem-specific characteristic. For instance, a suggested range for the dropout rate (DR) is between 0.2 and 0.5 (between the probability of 0.5 and 0.8 for retaining a unit) [77]. However, in our experiments, we found that the best dropout rate can be lower or higher than that range (see Fig. 10). Therefore, we believe that expanding the range is necessary, and it would be interesting if the algorithm could automatically determine the ranges based on the present situation. Fourthly, while we have shown the reliability of DGGA to optimize CNN, further comparisons should be investigated to have a deeper understanding of the capabilities of our proposed algorithm. Therefore, we will consider a comparison with other neural architecture searches [10]–[12], [15]–[17], [20], [30]–[33] and other existing diversity-guided evolutionary algorithms [46], [55], [57]. Fifthly, the applicability of the proposed algorithm was demonstrated on text classification datasets. However, it can be assumed that our methods are not limited to them and generally applicable to any possible CNN-based application. Thus, it would be interesting to apply our algorithm to different data types, such as image, video, or audio. Moreover, we only applied the proposed algorithm on binary text classification datasets. In future work, we intend to use it on multi-class benchmark datasets [78], [79] to show the general applicability and capabilities of our approach. Finally, another direction of future work is to investigate possibilities to accelerate the algorithm.

REFERENCES

[1] R. Ranjan, A. Bansal, J. Zheng, H. Xu, J. Gleason, B. Lu, A. Nanduri, J.-C. Chen, C. Castillo, and R. Chellappa, "A fast and accurate system for face detection, identification, and verification," *IEEE Trans. Biometrics, Behav., Identity Sci.*, vol. 1, no. 2, pp. 82–96, Apr. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8680708/

[2] H. Kim, "Multiple vehicle tracking and classification system with a convolutional neural network," *J. Ambient Intell. Hum. Comput.*, vol. 2, pp. 1–12, Aug. 2019, doi: 10.1007/s12652-019-01429-5.

[3] H. Kim and Y.-S. Jeong, "Sentiment classification using convolutional neural networks," *Appl. Sci.*, vol. 9, no. 11, p. 2347, Jun. 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/11/2347

[4] A. Aghaebrahimian and M. Cieliebak, "Hyperparameter tuning for deep learning in natural language processing," in *Proc. 4th Ed. Swiss Text Anal. Conf.*, 2019. [Online]. Available: http://ceur-ws.org/Vol-2458/paper5.pdf

[5] A. Wahab, H. Tayara, Z. Xuan, and K. T. Chong, "DNA sequences performs as natural language processing by exploiting deep learning algorithm for the identification of N4-methylcytosine," *Sci. Rep.*, vol. 11, no. 1, Dec. 2021, Art. no. 212. [Online]. Available: http://www.nature.com/articles/s41598-020-80430-x

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Nov. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=1953048.2078195

[7] A. Klein and F. Hutter, "Tabular benchmarks for joint architecture and hyperparameter optimization," 2019, *arXiv:1905.04970*. [Online]. Available: http://arxiv.org/abs/1905.04970

[8] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2503308.2188395

[9] E. Lewinson, *Python for Finance Cookbook*. Birmingham, U.K.: Packt, 2020.

[10] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Amsterdam, The Netherlands: Elsevier, 2019, pp. 293–312. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9780128154809000153

[11] P. Vidnerová and R. Neruda, "Evolution strategies for deep neural network models design," in *Proc. Inf. Technol.-Appl. Theory Conf.*, 2017, pp. 159–166. [Online]. Available: http://ceur-ws.org/Vol-1885/159.pdf

[12] A. Dahou, M. A. Elaziz, J. Zhou, and S. Xiong, "Arabic sentiment classification using convolutional neural network and differential evolution algorithm," *Comput. Intell. Neurosci.*, vol. 2019, pp. 1–16, Feb. 2019. [Online]. Available: https://www.hindawi.com/journals/cin/2019/2537689/

[13] I. Loshchilov and F. Hutter, "CMA-ES for hyperparameter optimization of deep neural networks," *CoRR*, vol. abs/1604.07269, pp. 1–5, Apr. 2016. [Online]. Available: http://arxiv.org/abs/1604.07269

[14] M. Suganuma, M. Kobayashi, S. Shirakawa, and T. Nagao, "Evolution of deep convolutional neural networks using Cartesian genetic programming," *Evol. Comput.*, vol. 28, no. 1, pp. 141–163, Mar. 2020. [Online]. Available: https://www.mitpressjournals.org/doi/full/10.1162/evco_a_00253

[15] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3840–3854, Sep. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9075201/

[16] T. Hinz, N. Navarro-Guerrero, S. Magg, and S. Wermter, "Speeding up the hyperparameter optimization of deep convolutional neural networks," *Int. J. Comput. Intell. Appl.*, vol. 17, no. 2, Jun. 2018, Art. no. 1850008, doi: 10.1142/S1469026818500086.

[17] G. C. Felbinger, "Optimal CNN hyperparameters for object detection on NAO robots," M.S. thesis, Techn. Univ. Hamburg-Harburg, Hamburg, Germany, 2018. [Online]. Available: https://hulks.de/_files/MA_Georg-Felbinger.pdf

[18] F. Mattioli, D. Caetano, A. Cardoso, E. Naves, and E. Lamounier, "An experiment on the use of genetic algorithms for topology selection in deep learning," *J. Electr. Comput. Eng.*, vol. 2019, Jan. 2019, Art. no. 3217542. [Online]. Available: https://www.hindawi.com/journals/jece/2019/3217542/

[19] N. Bansal, A. Sharma, and R. K. Singh, "An evolving hybrid deep learning framework for legal document classification," *Ingénierie des Syst. Inf.*, vol. 24, no. 4, pp. 425–431, Oct. 2019. [Online]. Available: http://www.iieta.org/journals/isi/paper/10.18280/isi.240410

[20] T. N. Fatyanosa and M. Aritsugi, "Effects of the number of hyperparameters on the performance of GA-CNN," in *Proc. IEEE/ACM Int. Conf. Big Data Comput., Appl. Technol. (BDCAT)*, Dec. 2020, pp. 144–153. [Online]. Available: https://ieeexplore.ieee.org/document/9302552/

[21] F.-A. Fortin, F.-M. De Rainville, M.-A. G. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *J. Mach. Lang. Res.*, vol. 13, pp. 2171–2175, Jul. 2012.

[22] F. Wilhelmstötter. (2021). *Jenetics Library User's Manual 6.2*. [Online]. Available: https://jenetics.io/

[23] M. A. A. Albadr, S. Tiun, M. Ayob, and F. T. AL-Dhief, "Spoken language identification based on optimised genetic algorithm–extreme learning machine approach," *Int. J. Speech Technol.*, vol. 22, no. 3, pp. 711–727, Sep. 2019. [Online]. Available: https://link.springer.com/article/10.1007/s10772-019-09621-w

[24] W. Bi, Y. Xu, and H. Wang, "Comparison of searching behaviour of three evolutionary algorithms applied to water distribution system design optimization," *Water*, vol. 12, no. 3, p. 695, Mar. 2020. [Online]. Available: https://www.mdpi.com/2073-4441/12/3/695

[25] A. Gogna and A. Tayal, "Comparative analysis of evolutionary algorithms for image enhancement," *Int. J. Metaheuristics*, vol. 2, no. 1, pp. 80–100, Jul. 2012, doi: 10.1504/IJMHEUR.2012.048219.

[26] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: Past, present, and future," *Multimedia Tools Appl.*, vol. 80, no. 5, pp. 8091–8126, Feb. 2021.

[27] M. Ærepináek, S.-H. Liu, and M. Mernik, "Exploration and exploitation in evolutionary algorithms: A survey," *ACM Comput. Surv.*, vol. 45, no. 3, pp. 1–33, Jun. 2013, doi: 10.1145/2480741.2480752.

[28] I. Ilievski, T. Akhtar, J. Feng, and C. A. Shoemaker, "Efficient hyperparameter optimization of deep learning algorithms using deterministic RBF surrogates," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 822–829.

[29] X. Zhang, X. Chen, L. Yao, C. Ge, and M. Dong, "Deep neural network hyperparameter optimization with orthogonal array tuning," 2019, *arXiv:1907.13359*. [Online]. Available: http://arxiv.org/abs/1907.13359

[30] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 2902–2911.

[31] T. Desell, "Developing a volunteer computing project to evolve convolutional neural networks and their hyperparameters," in *Proc. IEEE 13th Int. Conf. E-Sci.*, Oct. 2017, pp. 19–28. [Online]. Available: http://ieeexplore.ieee.org/document/8109119/

[32] A. Martín, R. Lara-Cabrera, F. Fuentes-Hurtado, V. Naranjo, and D. Camacho, "EvoDeep: A new evolutionary approach for automatic deep neural networks parametrisation," *J. Parallel Distrib. Comput.*, vol. 117, pp. 180–191, Jul. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0743731517302605

[33] I. De Falco, G. De Pietro, A. Della Cioppa, G. Sannino, U. Scafuri, and E. Tarantino, "Evolution-based configuration optimization of a deep neural network for the classification of obstructive sleep apnea episodes," *Future Gener. Comput. Syst.*, vol. 98, pp. 377–391, Sep. 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167739X1832805X

[34] J. P. Mueller and L. Massaron, *Python for Data Science*, 2nd ed. Hoboken, NJ, USA: Wiley, 2019.

[35] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, vol. 2. Red Hook, NY, USA: Curran Associates, 2012, pp. 2951–2959.

[36] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1437–1446. [Online]. Available: https://arxiv.org/abs/1807.01774

[37] A. Quitadadmo, J. Johnson, and X. Shi, "Bayesian hyperparameter optimization for machine learning based eQTL analysis," in *Proc. 8th ACM Int. Conf. Bioinf., Comput. Biol., Health Informat.* New York, NY, USA: Association for Computing Machinery, 2017, pp. 98–106, doi: 10.1145/3107411.3107434.

[38] W. Jia, C. Xiu-Yun, Z. Hao, X. Li-Dong, L. Hang, and D. Si-Hao, "Hyperparameter optimization for machine learning models based on Bayesian optimization," *J. Electron. Sci. Technol.*, vol. 17, no. 1, pp. 26–40, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1674862X19300047

[39] T. Theckel Joy, S. Rana, S. Gupta, and S. Venkatesh, "Fast hyperparameter tuning using Bayesian optimization with directional derivatives," 2019, *arXiv:1902.02416*. [Online]. Available: http://arxiv.org/abs/1902.02416

[40] N. Mori, M. Takeda, and K. Matsumoto, "A comparison study between genetic algorithms and Bayesian optimize algorithms by novel indices," in *Proc. 7th Annu. Conf. Genetic Evol. Comput.* New York, NY, USA: Association for Computing Machinery, 2005, pp. 1485–1492, doi: 10.1145/1068009.1068244.

[41] M. Schmidt, S. Safarani, J. Gastinger, T. Jacobs, S. Nicolas, and A. Schulke, "On the performance of differential evolution for hyperparameter tuning," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/document/8851978/

[42] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Jun. 1999. [Online]. Available: http://ieeexplore.ieee.org/document/784219/

[43] R. Johnson and T. Zhang, "Effective use of word order for text categorization with convolutional neural networks," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, 2015, pp. 103–112. [Online]. Available: https://www.aclweb.org/anthology/N15-1011

[44] K. Kowsari, K. J. Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, "Text classification algorithms: A survey," *Information*, vol. 10, no. 4, p. 150, Apr. 2019. [Online]. Available: https://www.mdpi.com/2078-2489/10/4/150

[45] K.-H. Lee, K.-S. Leung, C.-W. Ho, and Y.-Y. Wong, "A novel approach in parameter adaptation and diversity maintenance for genetic algorithms," *Soft Comput. Fusion Found., Methodol. Appl.*, vol. 7, no. 8, pp. 506–515, Aug. 2003. [Online]. Available: http://link.springer.com/10.1007/s00500-002-0235-1

[46] R. K. Ursem, "Diversity-guided evolutionary algorithms," in *Parallel Problem Solving From Nature*, J. J. M. Guervós, P. Adamidis, H.-G. Beyer, H.-P. Schwefel, and J.-L. Fernández-Villacañas, Eds. Berlin, Germany: Springer, 2002, pp. 462–471.

[47] E. Pellerin, L. Pigeon, and S. Delisle, "Self-adaptive parameters in genetic algorithms," in *Data Mining and Knowledge Discovery: Theory, Tools, and Technology VI*, vol. 5433, B. V. Dasarathy, Ed. Bellingham, WA, USA: SPIE, 2004, pp. 53–64, doi: 10.1117/12.542156.

[48] M. Vannucci and V. Colla, "Fuzzy adaptation of crossover and mutation rates in genetic algorithms based on population performance," *J. Intell. Fuzzy Syst.*, vol. 28, no. 4, pp. 1805–1818, 2015. [Online]. Available: https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/IFS-141467

[49] A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. Hammouri, and V. B. S. Prasath, "Choosing mutation and crossover ratios for genetic algorithms—A review with a new dynamic approach," *Information*, vol. 10, no. 12, p. 390, Dec. 2019. [Online]. Available: https://www.mdpi.com/2078-2489/10/12/390

[50] E. Osaba, R. Carballedo, F. Diaz, E. Onieva, I. de la Iglesia, and A. Perallos, "Crossover versus mutation: A comparative analysis of the evolutionary strategy of genetic algorithms applied to combinatorial optimization problems," *Sci. World J.*, vol. 2014, Aug. 2014, Art. no. 154676. [Online]. Available: http://www.hindawi.com/journals/tswj/2014/154676/

[51] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, no. 4, pp. 656–667, Apr. 1994. [Online]. Available: http://ieeexplore.ieee.org/document/286385/

[52] C.-Y. Chang and D.-R. Chen, "Active noise cancellation without secondary path identification by using an adaptive genetic algorithm," *IEEE Trans. Instrum. Meas.*, vol. 59, no. 9, pp. 2315–2327, Sep. 2010. [Online]. Available: http://ieeexplore.ieee.org/document/5492196/

[53] W. F. Mahmudy, R. M. Marian, and L. H. S. Luong, "Hybrid genetic algorithms for part type selection and machine loading problems with alternative production plans in flexible manufacturing system," *ECTI Trans. Comput. Inf. Technol.*, vol. 8, no. 1, pp. 80–93, Jan. 1970.

[54] Y. A. Auliya, W. F. Mahmudy, and Sudarto, "Improve hybrid particle swarm optimization and K-means for clustering," *J. Inf. Technol. Comput. Sci.*, vol. 4, no. 1, p. 42, 2019.

[55] T. Gabor, L. Belzner, and C. Linnhoff-Popien, "Inheritance-based diversity measures for explicit convergence control in evolutionary algorithms," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2018, pp. 841–848.
[56] W. M. Spears, "The role of mutation and recombination in evolutionary algorithms," Ph.D. dissertation, Dept. Comput. Sci., George Mason Univ., Fairfax, VA, USA, 1998.
[57] C. Chen, Z. Yang, Y. Tan, and R. He, "Diversity controlling genetic algorithm for order acceptance and scheduling problem," *Math. Problems Eng.*, vol. 2014, Feb. 2014, Art. no. 367152.
[58] S. Negi, T. Daudert, and P. Buitelaar, "SemEval-2019 task 9: Suggestion mining from online reviews and forums," in *Proc. 13th Int. Workshop Semantic Eval.* Minneapolis, MI, USA: Association for Computational Linguistics, Jun. 2019, pp. 877–887. [Online]. Available: https://www.aclweb.org/anthology/S19-2151
[59] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Doha, Qatar, Oct. 2014, pp. 1532–1543. [Online]. Available: https://www.aclweb.org/anthology/D14-1162
[60] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. Newton, MA, USA: O'Reilly Media, 2009.
[61] W. McKinney, "Data structures for statistical computing in Python," in *Proc. 9th Python Sci. Conf.*, S. van der Walt and J. Millman, Eds., 2010, pp. 51–56.
[62] P. Virtanen, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, pp. 261–272, Feb. 2020. [Online]. Available: http://www.nature.com/articles/s41592-019-0686-2
[63] S. Loria. (2018). *Textblob: Simplified Text Processing*. [Online]. Available: https://textblob.readthedocs.io/en/dev/
[64] F. Chollet. (2015). *Keras*. [Online]. Available: https://keras.io
[65] M. Abadi. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. [Online]. Available: https://www.tensorflow.org/
[66] F. Eight. (2019). *Twitter US Airline Sentiment*. [Online]. Available: https://www.kaggle.com/crowdflower/twitter-airline-sentiment
[67] S. Saha and S. Bandyopadhyay, "A symmetry based multiobjective clustering technique for automatic evolution of clusters," *Pattern Recognit.*, vol. 43, no. 3, pp. 738–751, Mar. 2010. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0031320309002714
[68] T. Wang, Z. Liu, Y. Chen, Y. Xu, and X. Dai, "Load balancing task scheduling based on genetic algorithm in cloud computing," in *Proc. IEEE 12th Int. Conf. Dependable, Autonomic Secure Comput.*, Aug. 2014, pp. 146–152. [Online]. Available: https://ieeexplore.ieee.org/document/6945680
[69] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter, "Auto-sklearn: Efficient and robust automated machine learning," in *Automated Machine Learning: Methods, System, Challenges*, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds. Cham, Switzerland: Springer, 2019, pp. 113–134, doi: 10.1007/978-3-030-05318-5_6.
[70] R. S. Olson and J. H. Moore, "TPOT: A tree-based pipeline optimization tool for automating machine learning," in *Automated Machine Learning: Methods, System, Challenges*. Cham, Switzerland: Springer, 2019, pp. 151–160. [Online]. Available: http://link.springer.com/10.1007/978-3-030-05318-5_8
[71] Y. Choi and H. Lee, "Data properties and the performance of sentiment classification for electronic commerce applications," *Inf. Syst. Frontiers*, vol. 19, no. 5, pp. 993–1012, Oct. 2017.
[72] F. Iandola and K. Keutzer, "Small neural nets are beautiful: Enabling embedded systems with small deep-neural-network architectures," in *Proc. 12th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth. Companion*. New York, NY, USA: ACM Press, 2017, pp. 1–10. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3125502.3125606
[73] F. Chollet and J. J. Allaire, *Deep Learning With R*, 1st ed. Greenwich, CT, USA: Manning, 2018.
[74] J. Zhang, H. S.-H. Chung, and W.-L. Lo, "Clustering-based adaptive crossover and mutation probabilities for genetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 11, no. 3, pp. 326–335, Jun. 2007. [Online]. Available: http://ieeexplore.ieee.org/document/4220690/
[75] A. Biedenkapp, M. Lindauer, K. Eggensperger, F. Hutter, C. Fawcett, and H. H. Hoos, "Efficient parameter importance analysis via ablation with surrogates," in *Proc. 21st AAAI Conf. Artif. Intell.*, 2017, pp. 773–779.
[76] J. N. van Rijn and F. Hutter, "Hyperparameter importance across datasets," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*. New York, NY, USA: Association for Computing Machinery, 2018, pp. 2367–2376, doi: 10.1145/3219819.3220058.
[77] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2627435.2670313
[78] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "RCV1: A new benchmark collection for text categorization research," *J. Mach. Learn. Res.*, vol. 5, pp. 361–397, Dec. 2004.
[79] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in Neural Information Processing Systems*, vol. 28, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds. New York, NY, USA: Curran Associates, 2015.

TIRANA NOOR FATYANOSA (Graduate Student Member, IEEE) received the B.S. and M.S. degrees in computer science from Brawijaya University, Indonesia, in 2013 and 2017, respectively. She is currently pursuing the Ph.D. degree with the Graduate School of Science and Technology, Kumamoto University, Japan. From 2013 to 2018, she also embarked on her technical career with several IT companies: Mitrais and Anomadic, working as a Software Engineer and a Quality Assurance Specialist. Also, she has worked with Kata.ai, as an AI Research Scientist Intern. Her research interests include deep neuroevolution, focusing on the automatic hyperparameters and architecture optimization of deep neural networks using an evolutionary algorithm.

During her graduate study, she received the Best Presenter Award at the SIET, in 2016, and the Best Session Presenter Award at the ICACSIS, in 2017.

MASAYOSHI ARITSUGI (Member, IEEE) received the B.E. and D.E. degrees in computer science and communication engineering from Kyushu University, Japan, in 1991 and 1996, respectively. From 1996 to 2007, he was with the Department of Computer Science, Gunma University, Japan. Since 2007, he has been a Professor with Kumamoto University, Japan. His research interests include database systems and parallel/distributed data processing.

Prof. Aritsugi is a Senior Member of IPSJ and IEICE and a member of ACM and DBSJ. He was a recipient of the COMPSAC 2015 Best Paper Award, the Best Paper Award in image processing and understanding in 13th IEEE International Conference on Signal Processing (ICSP2016), and the Best Paper Award at the 2019 IEEE International Cyber Science and Technology Congress (CyberSciTech).