A Divide-and-Conquer Genetic Programming Algorithm with Ensembles for Image Classification

Ying Bi, *Member, IEEE*, Bing Xue, *Senior Member, IEEE*, and Mengjie Zhang, *Fellow, IEEE*

***Abstract*—Genetic programming (GP) has been applied to feature learning in image classification and achieved promising results. However, one major limitation of existing GP-based methods is the high computational cost, which may limit their applications on large-scale image classification tasks. To address this, this paper develops a divide-and-conquer GP algorithm with knowledge transfer and ensembles to achieve fast feature learning in image classification. In the new algorithm framework, a divide-and-conquer strategy is employed to split the training data and the population into small subsets or groups to reduce computational time. A new knowledge transfer method is proposed to improve GP learning performance. A new fitness function based on log-loss and a new ensemble formulation strategy are developed to build an effective ensemble for image classification. The performance of the proposed approach has been examined on 12 image classification datasets of varying difficulty. The results show that the new approach achieves better classification performance in significantly less computation time than the baseline GP-based algorithm. The comparisons with state-of-the-art algorithms show that the new approach achieves better or comparable performance in almost all the comparisons. Further analysis demonstrates the effectiveness of ensemble formulation and knowledge transfer in the proposed approach.**

***Index Terms*—Genetic Programming; Feature Learning; Knowledge Transfer; Ensemble Learning; Divide-and-Conquer; Image Classification.**

I. Introduction

IMAGE classification is a supervised learning task, where labelled training data are used to build a classifier for classifying images based on their content in the images. The original training data often consist of n images with class labels, i.e., $\mathcal{D}_{train} = \{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_n}, y_n)\}$, where $\boldsymbol{x} \in \mathbb{R}^{m \times l}$ represents a grey-scale image with size $m \times l$, and $y \in \mathbb{Z}$ represents the class label (the total number of classes is C). The pixel values of the images can be directly used for classification, but may not be effective because image data often have high variations in scale, rotation, viewpoint, and illumination. Instead, high-level image features, which are invariant to certain variations, are often used for effective classification. Image features can be extracted from images using manually designed descriptors or learned using feature learning techniques [1]. Typically, the learned features are often more effective for image classification than the manually designed features and do not require domain knowledge [2, 3]. Typical methods that can learn image features include convolutional neural network (CNNs) [2, 3] and genetic programming (GP) [4, 5, 6]. CNNs often require extensive expertise to design the architecture, which affects the performance of CNNs on particular data [7]. Compared with CNNs, GP has a flexible variable-length representation and can automatically find solutions without predefined structures. In addition, the solutions of GP often provide high interpretability, which can be seen from the example GP trees or the learned features visualised in [5, 8].

As an evolutionary computation (EC) algorithm, GP automatically evolves computer programs to solve problems without predefined solution structures [9]. GP uses a tree-based representation and iteratively searches for the best tree based on the principles of biologic evolution. GP has been used to automatically learn effective features for image classification and achieved promising results [5, 6, 10]. In these GP methods, each tree is often used to transform images into a set of informative features for classification [4, 5, 6, 10]. Typically, each GP tree can be formulated as $\Phi_{f,t}(\cdot) : \mathbb{R}^{m \times l} \rightarrow \mathbb{R}^d$, which consists of functions/operators (f), variables and constants (t). d denotes the number of learned features. A performance measure $\mathcal{L}$ is often used to evaluate the effectiveness of the learned features and to guide the search of GP towards finding the best tree ($\Phi^*(\cdot)$). Given $\mathcal{D}_{train}$ and $\mathcal{L}$, the feature learning task using GP can be formulated as Eq. (1).

$$\Phi^*(\cdot) = \underset{f \in \mathcal{F},\ t \in \mathcal{T}}{\operatorname{argmax}} \ \mathcal{L}(\Phi_{f,t}(\cdot), \mathcal{D}_{train}). \tag{1}$$

where $\Phi^*(\cdot)$ denotes the best tree found by GP. $\mathcal{F}$ means the function set and $\mathcal{T}$ means the terminal set. After the evolutionary search process, the best tree $\Phi^*(\cdot)$, which obtains the best $\mathcal{L}$ on $\mathcal{D}_{train}$, is tested on an unseen dataset $\mathcal{D}_{test}$ to show its generalisation performance.

Although GP has achieved promising results in different image classification tasks, one important problem is how to improve its computational efficiency. The desire of improving GP's efficiency is driven from the rapidly increasing number of images/instances in image classification datasets. Many well-known benchmark datasets have a very large number of images for training or testing. For example, MNIST has 60,000 training images [11], and ImageNet has 1.2 million training images [12]. Thanks to the graphics processing unit

This work was supported in part by the Marsden Fund of New Zealand Government under Contracts VUW1509, VUW1615, VUW1913 and VUW1914, the Science for Technological Innovation Challenge (SfTI) fund under grant E3603/2903, the University Research Fund at Victoria University of Wellington grant number 223805/3986, MBIE Data Science SSIF Fund under the contract RTVU1914, and National Natural Science Foundation of China (NSFC) under Grant 61876169.

The authours are with School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand (e-mail: ying.bi@ecs.vuw.ac.nz; bing.xue@ecs.vuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

(GPU) acceleration, many deep learning methods, particularly deep CNNs, can be applied to such large datasets. However, the current GP-based image classification methods are only based on the central processing unit (CPU), whose has lower computation speed than GPUs. Therefore, it is computationally expensive or even impossible to apply GP to image classification datasets with a large number of images based on CPUs. On the other hand, GP is a population-based search technique that needs a large number of expensive fitness evaluations. These form the motivations of developing a new GP-based approach to image classification with the goal of improving computational efficiency.

Existing methods often cannot address well the problem of improving computational efficiency of GP without sacrificing the generalisation performance. These methods include using a small number of instances for fitness evaluations [4], approximating the fitness values using surrogate models [13], or using subtree caching [14]. Using a small number of instances directly reduces the fitness evaluation cost, but it may lead to poor generalisation performance on unseen data. Surrogates have been used to fitness approximate in many evolutionary algorithms for computationally expensive problems [15]. But it may introduce uncertainty into the evaluation process. In addition, the design of surrogate models in GP is not easy because of the tree-based variable-length representation of the solutions. Subtree caching is a method that stores the evaluated GP trees and their fitness values in a hash table to avoid re-evaluating the trees that have been stored in the hash table [14]. But this method cannot significantly reduce the computational cost because many new trees may be generated and it requires additional searching time. To this end, this paper attempts to find better ways to improve the efficiency of GP for feature learning in image classification without affecting performance.

Divide-and-conquer is known as a decomposition method that decomposes a large problem into a set of smaller and simpler subproblems to be addressed in an iterative manner, and has been widely applied to large-scale optimisation problems [16]. Inspired by this method, this paper proposes a new feature learning framework for image classification. It splits the original training set into several non-overlapping small subsets such that a small population can be used to learn from each small subset. We term the new framework as DCFL, indicating Divide-and-Conquer Feature Learning. During evolution, each small population is evaluated on a small subset of the training set, which intuitively reduces the evaluation time. Since multiple populations are used in the new framework and each population uses a different subset for feature learning, a knowledge transfer method is developed to improve the learning performance. In addition, without loss of generalisation performance, a new fitness function and an ensemble formulation strategy are proposed in DCFL to achieve effective image classification. It is noted that the idea of using multiple small populations to perform search and the information exchange across these populations are similar to island-based EC algorithms [17], so DCFL can be a special case of them. Different from them, DCFL splits the training set into several small subsets and build ensembles to solve the learning problem with the goal of improving the efficiency.

The key characteristics of DCFL are summarised as follows:

1) DCFL splits the training set into several non-overlapping subsets and the whole population into multiple small populations. Each small population is used to learn features from a subset of the training set. Besides, another small population is used to learn features from the whole training set. Theoretically and empirically, this design reduces the overall computation time. DCFL is suitable for different GP methods using various tree representations for image classification.
2) DCFL shares the knowledge across these multiple populations to improve the learning performance of multiple small populations and the overall performance.
3) DCFL employs a new log-loss-based fitness function to obtain more accurate information on the effectiveness of the learned features in guiding the GP search.
4) DCFL returns the best tree found by each small population and creates an effective ensemble for image classification using a new ensemble formulation strategy. The new strategy selects better trees and calculates weights of the classifiers built from the selected trees to achieve high generalisation performance.

DCFL can cooperate with the GP methods using different tree representations to achieve fast feature learning for image classification. In this paper, the tree representation of the GP approach with image-related operators (i.e., FGP in short) proposed in [6] is used together with DCFL for image classification. Therefore, the new approach is also termed DCFL-FGP, which denotes the use of the DCFL framework and the FGP tree representation. Extensive experiments are conducted to show the effectiveness of DCFL-FGP in image classification.

II. Background and Related Work

A. GP for Image Feature Learning

In general, the overall process of the GP-based image feature learning algorithm can be summarised as follows.

Step 1: Randomly initialise a population of trees. Each tree is generated using a tree generation method based on the program structure, the function set and the terminal set;

Step 2: Evaluate the fitness of each tree using a fitness function and a set of training images with class labels;

Step 3: Use elitism to copy the best trees to the next generation. New trees are generated from the trees selected by a selection method (i.e., tournament selection) using the subtree crossover and subtree mutation operators; and

Step 4: Go to step 2 if the termination condition is not satisfied. Otherwise, terminate the evolutionary learning process and return the best tree.

The above process is similar to standard GP except for the tree generation in steps 1 and 3 and the fitness evaluation in step 2. The fitness evaluation varies with the algorithm design and the target task. The tree generation is different because most GP-based image feature learning algorithms have a different representation compared with standard GP. Standard GP constructs solutions by using arithmetic operators, e.g., $+, -$, and $\times$, as internal nodes and using features of the dataset or random constants as leaf nodes. Most GP-based

image feature learning algorithms have a special representation consisting of domain-specific functions as internal or root nodes [5]. The representation often determines how the image is transformed into features, e.g., through filtering and pooling operations in [5, 18], which is the key to success.

1) GP-based Feature Learning Algorithms: The proposed DCFL framework can cooperate with different GP-based algorithms using various representations for image classification to speed up their learning processes. In this paper, we use the representation newly proposed in FGP [6] because it has achieved promising results in many different image classification tasks.

The FGP method [6] is based on strongly typed GP (STGP) [19] and has a multi-layer program structure and a function set containing many image-related operators. The program structure has input, filtering, pooling, feature extraction, feature concatenation, and output layers, are shown in Fig. 1. In these layers, various image filters and descriptors are employed as functions (internal nodes of GP trees), respectively. These designs allow FGP to extract high-level features from raw images through multiple simple or complex transformations. Since the transformations are based on the operators from the image domain, FGP can learn effective features for classification. The results have shown its effectiveness on different tasks [6].

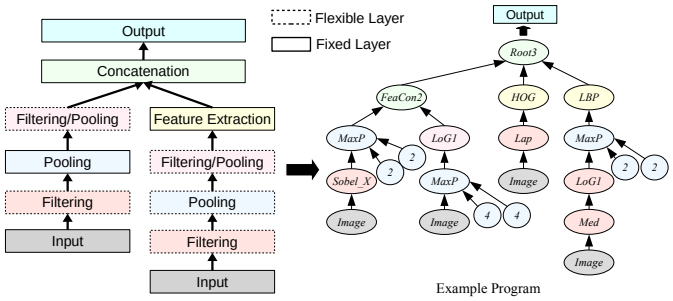


Fig. 1. The program structure and an example program of FGP [6]. The internal nodes in the example program are image-related operators.

Except for FGP, many GP methods with different representations have been developed for image classification. Shao et al. [5] developed a multi-objective GP algorithm (MOGP) for feature learning in image classification. MOGP has multiple layers, i.e., an input layer, a filtering layer, a pooling layer, a concatenation layer, and an output layer, to learn image features. MOGP achieved better results than a variety of methods on different tasks. This method could learn a large number of image features and require additional dimensionality reduction. Bi et al. [18] proposed a GP method to detect small regions from images and describe features using Gaussian-based filters and pooling operators simultaneously. The method achieved better performance on six different image classification datasets. Al-Sahaf et al. [4] developed a GP method with a special root node to learn texture features for classification. The learned GP tree can describe texture features in a way that is similar to local binary patterns (LBP), which is a well-known texture descriptor. This method achieved better results than LBP and its variants on several texture classification datasets. However, since this method can only extract texture features, it may not be effective for other types of image classification tasks. Bi et al. [20] developed a GP method with a multi-layer representation to achieve simultaneous feature learning and ensemble learning for image classification. This method used a set of image-related operators and classification algorithms to build GP trees, achieving better performance than many other effective algorithms on different image classification tasks. However, with the use of complex operators as GP functions, the computational cost of this method is high.

GP-based feature learning, including feature extraction and construction, has also been widely applied to other tasks. Rodriguez-Coayahuitl et al. [21] investigated a structured layered GP approach to representation learning. A GP-based autoencoder, was developed to construct a representation of the data using an encoding forest and a decoding forest. This GP autoencoder achieved promising results on three image datasets. Tran et al. [22] developed GP-based methods for multiple feature construction in high-dimensional classification. The GP methods with a multi-tree representation was used to construct multiple features for classification based on class-independent and class-dependent fitness functions. The GP methods achieved better results in several real-world applications. La Cava et al. [23] developed a multidimensional GP method to learn multiple features for multi-class classification. A new program representation was developed to allow GP to produce multiple features and a distance-based classifier was designed for classification. This method achieved better performance than other algorithms on several classification problems. More related work can be found from [8, 24].

Several methods have been developed to improve the efficiency of GP. Nguyen et al. [13] proposed a surrogate-assisted GP method for dynamic job shop scheduling. The fitness values of GP trees were approximated using simple simulation models that have a small number of jobs. This method could reduce the computational cost of GP without significant performance decrease. However, GP for feature learning is different from GP for job shop scheduling and does not use simulation models for fitness evaluations. In [14], subtree caching was developed to improve the efficiency of GP. The evaluated subtrees and trees, and their fitness values were stored in a hash table. This method can avoid re-evaluating the subtree that have been evaluated and stored in the hash table. However, it requires time to perform a loop to check whether the subtree is in this hash table so it cannot significantly reduce the computational cost. Other methods such as surrogates for fitness approximation, have also been used in EC algorithms [15]. However, surrogates often introduce uncertainty into the evaluation process and more investigation is needed to better use them because GP has a variable-length representation, which is different from other evolutionary algorithms.

In summary, most GP-based feature learning algorithms for image classification focus on the GP representations with the goal of improving the classification performance [4, 5, 6]. The high computational cost of GP for image classification has seldom been addressed. Since many image datasets tend to be very large, e.g., MNIST [11] has 60,000 training images and ImageNet [12] has 1.2 million training images of 1,000 classes, the computational cost of GP on such large datasets

could be very high. It is necessary to develop a new GP approach applicable to these large datasets with less computational cost. Existing work on improving the efficiency of GP have limitations. Therefore, in this paper, inspired by the ideas of divide-and-conquer [25] and island-based models [17, 26], we develop a new GP approach to improving the computational efficiency in image classification.

B. Knowledge Transfer

Knowledge transfer, or transfer learning, can extract useful knowledge from a source domain or task and use the knowledge to help improve the learning on a target domain or task. It has been used to improve the learning performance of EC algorithms. Gupta et al. [27] summarised three typical paradigms of knowledge transfer in optimisation problems, i.e., sequential transfer, multitask optimisation, and multiform optimisation. Iqbal et al. [28] extracted the code fragments of GP trees learned from a source task and reused them in the initialisation and mutation steps of GP to improve its learning performance in a target task. Jiang et al. [29] developed a new transfer learning method to improve the performance of multi-objective optimisation algorithms. The idea considered that the non-dominated solutions at different times have different distributions and there are relationships between these probability distributions. This method can find a latent space where a global feature of the source and target domains is small and the transfer learning idea can improve the algorithm performance. Gupta et al. [30] proposed an evolutionary multitask optimisation approach, where a population was used to solve multiple tasks and the genetic materials were transferred implicitly through chromosomal crossover across different tasks. Ding et al. [31] developed a new framework for evolutionary multitask optimisation, where the knowledge extracted from cheap optimisation problems were used to improve the optimisation performance on computationally expensive problems. In this study, the new divide-and-conquer algorithm framework uses multiple populations to search for the best trees from each small subset of the training set. To improve the learning performance, a knowledge transfer method is developed to achieve knowledge transfer between these small population under the new algorithm framework.

C. Ensemble Methods for Classification

Ensemble methods typically use a set of classifiers to solve a classification problem [32]. Well-known ensemble methods are bagging or boosting methods, which train (base) classifiers to build an ensemble in parallel or sequentially. After obtaining a set of classifiers, a combination method, such as voting and averaging, is used to combine the outputs of different base learners to make the final prediction. Many EC algorithms, including GP, have been developed to construct ensembles for classification. Rosales-Pérez et al. [33] proposed an evolutionary multi-objective algorithm to find multiple solutions to instance selection and built an ensemble of SVM classifiers using these solutions. Five ensemble formulation strategies were investigated. The constructed ensembles achieved better results than many other algorithms. Zhang et al. [34] developed a new GP method to evolve a set of similarity functions that were used to form an ensemble for text classification. This method can find multiple similarity functions using a single population, achieving better performance than the other fusion methods such as GAs. Nag and Pal [35] developed a multi-objective GP-based method to simultaneous feature selection and diverse classifier designing for classification. This method optimised three objectives, i.e., false positive, false negative, and the number of leaf nodes in the GP tree. This method built multiple binary classifiers with concise rules to construct ensembles. This method achieved better performance on many classification tasks in most comparisons. Rodrigues et al. [36] proposed an ensemble GP method in which the population has a different structure, fitness function, and genetic operators to simultaneously evolve ensembles for classification. This method achieved better results than standard GP and other ensemble-based GP methods on eight binary classification tasks. Bi et al. [37] developed a GP method to automatically extract image features and evolve ensembles of random forest classifiers for image classification. This method achieved better performance than traditional methods and deep random forest on several image classification datasets.

In the proposed DCFL framework, multiple best GP trees can be evolved and obtained after the evolutionary process. These trees can be used to build an ensemble for classification with high generalisation performance. This paper will develop a new ensemble formulation strategy to combine these classifiers built from the best trees to create an effective ensemble for image classification.

III. The Proposed Approach

This section introduces the proposed DCFL framework in detail. Then the knowledge transfer method is described, followed by the descriptions of the new fitness function and the new ensemble method.

A. Algorithm Framework

The overall algorithm framework of DCFL is shown in Fig. 2. Based on the idea of divide-and-conquer, the whole training set $\mathcal{D}_{train}$ is randomly split into N non-overlapping subsets preserving the class ratio to form subsets $\mathcal{D}_1,\ \ldots,\ \mathcal{D}_N$ and the whole population with a size of S is equally partitioned into $N+1$ small populations, i.e., $P_0,\ P_1,\ \ldots,\ P_N$. The population size for each small population is $[S/(N+1)]$ (use the integer part if S is indivisible by $N+1$). It is noted that, unlike other EC algorithms, e.g., GAs and particle swarm optimisation (PSO), GP often uses a larger population size, such as 500 in [6] or 1,000 in [13]. Therefore, it is applicable to split the whole population into several small populations to perform the search.

In DCFL, each small population uses a standard evolutionary process to search for the best tree/individual by updating and evaluating the population at every generation. At the initialisation step, DCFL randomly generates multiple small populations of trees using a tree generation method based on the predefined GP program structure, function set, and terminal set. Then each population is evaluated on a specific training

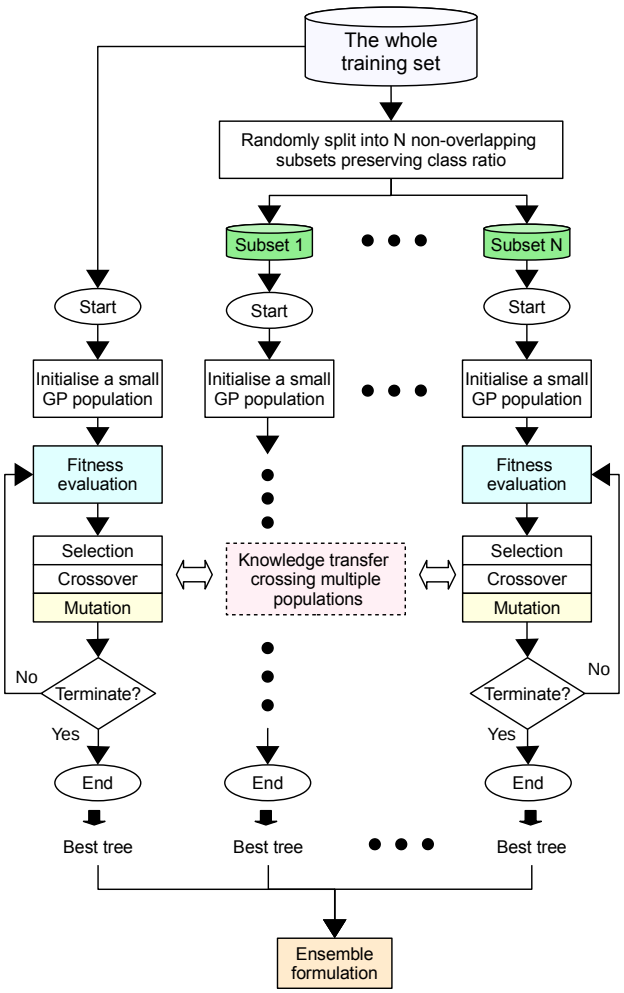


Fig. 2. The overall algorithm framework of DCFL. The whole training set is split into N non-overlapping subsets. This framework uses $N+1$ small populations to learn features from the whole training set and the N subsets, respectively. Knowledge sharing is performed across these $N+1$ populations during the evolutionary process. The best tree found by each small population is used to form an effective ensemble for image classification.

set to obtain the fitness of the trees using a new log-loss-based fitness function. DCFL uses crossover and mutation operators to generate a new population at each generation. During this process, DCFL extracts and transfers knowledge across these multiple populations to improve the learning performance of multiple small populations. After the evolutionary process, i.e., when the maximal number of generations is reached, the best tree found by each population is returned. Based on these best trees, DCFL creates an effective ensemble for image classification using a new ensemble formulation strategy. The new strategy can automatically select the best trees and calculate weights of the classifiers built from the selected trees to achieve high generalisation performance.

During the evolutionary learning process, the P_0 population uses the whole training set to learn features, while the other N populations use the N non-overlapping subsets to learn features, respectively. In other words, P_0 is evaluated on $\mathcal{D}_{train}$ and P_i is evaluated on $\mathcal{D}_i$ (with $\mathcal{D}_1 \cup \mathcal{D}_2 \cup \cdots \cup \mathcal{D}_N = \mathcal{D}_{train}$ and $\mathcal{D}_i \cap \mathcal{D}_j = \emptyset,\ i,j \in [1,N], i \neq j$) at each generation. In contrast to evaluating the whole population on $\mathcal{D}_{train}$, DCFL evaluates multiple small populations on small subsets of $\mathcal{D}_{train}$, which directly reduces the computational cost.

Typically, using a small population has worse search ability than using a large population. Since multiple populations are used in DCFL and each population uses a different dataset for learning, the knowledge across the different populations can be exchanged to improve the learning performance. The knowledge transfer method will be introduced in Section III-B.

Logistic regression (LR) can provide the prediction probabilities of different classes, which is better for constructing an ensemble. Therefore, LR is used in DCFL as a classification algorithm and a new fitness function is developed to better evaluate the performance of the trees. The new fitness function is based on log-loss, which can provide accurate and diverse information to evaluate the trees when the training set is small. Section 6 will describe the new fitness function in details.

In DCFL, these small populations use limited information from a small training set to search for the best trees. Image data often have high variations. The best tree found using limited information of the training set may have poor generalisation performance on unseen data. An ensemble consists of multiple classifiers to solve a problem and can achieve better generalisation performance than a single classifier [32]. Therefore, DCFL creates an effective ensemble from the best trees returned by these multiple small populations in order to obtain high generalisation performance on the unseen dataset. The ensemble is built using a new ensemble formulation strategy, which will be introduced in Section III-D.

*1) **Approximated Running/Computation Time Speed-Up Analysis**:* We theoretically compare the approximated running time of DCFL with a GP method having a standard algorithm framework (i.e., SGP in short), i.e., evaluating the whole population on the whole training set, to show its efficiency. Typically, the main computation time of GP for image classification is spent on fitness evaluations, including transforming images into features via GP trees and performing classification using those features. Therefore, in this analysis, only the image transformation time and the classification time are counted and compared. Because GP evolves variable-length solutions that transform the images into a set of features, exact computation time is very difficult to calculate. For simplicity, the computation time of using a GP tree to transform n images is $w_1 n$, where w_1 indicates the weight/multiplicative factor. In addition, we assume that DCFL and SGP both use SVM for classification. Typically, the computation time of SVM to classify n instances, each of which has f features, is $w_2 n^2 f$ (the computational complexity of SVM is $O(n^2 f)$ [38, 39]), where w_2 indicates the weight/multiplicative factor.

Let S denotes the whole population size, n denotes the number of instances in $\mathcal{D}_{train}$, and G denotes the maximal number of generations. At every generation, SGP evaluates S individuals on $\mathcal{D}_{train}$ with n images. The approximated computation time of SGP of a single run is

$$T_{SGP} = GSw_1 n + GSw_2 n^2 f, \tag{2}$$

At every generation, DCFL evaluates $S/(N+1)$ trees on $\mathcal{D}_{train}$ and $S/(N+1)$ trees on $\mathcal{D}_1, \ldots, \mathcal{D}_N$, respectively. In

$\mathcal{D}_1, \ldots, \mathcal{D}_N$, the number of instances is n/N (use the integer part if n is indivisible by N). The approximated computation time of DCFL of a single run is

$$T_{DCFL} = \frac{GS}{N+1}(w_1 n + w_2 n^2 f) + \frac{GSN}{N+1}(\frac{w_1 n}{N} + w_2(\frac{n}{N})^2 f) = GS\frac{2w_1 n}{N+1} + GS\frac{w_2 n^2 f}{N} \quad (3)$$

Based on Equations (2) and (3), we can have $T_{DCFL} < T_{SGP}$ on the condition that $N > 1$. A more detailed proof can be found in the supplementary material.

Taking $N = 4$ as an example, the approximated computation time of DCFL is $2GSw_1 n/5 + GSw_2 n^2 f/4$. Comparing with SGP (i.e., $GSw_1 n + GSw_2 n^2 f$), DCFL is much faster. Theoretically, DCFL is faster than SGP. In Section V, the empirical running time speed-up comparisons will be provided to further demonstrate the efficiency of DCFL.

B. Knowledge Transferring/Sharing in DCFL

The DCFL approach has $N + 1$ small populations, which can learn different knowledge from various data $\mathcal{D}_{train}, \mathcal{D}_1, \ldots, \mathcal{D}_N$. Each dataset can be considered as a single task with different instances. Knowledge transfer across these populations can help improve learning performance. But the key questions are what to transfer, how to transfer and when to transfer [40]. In DCFL, we develop a simple method to achieve fast knowledge transfer between the $N + 1$ small populations at each generation, which is different from the current work on knowledge transfer in GP.

Typically, the GP trees or subtrees contain/represent the knowledge learned/discovered from the tasks [28]. The knowledge can be extracted for reuse [28]. DCFL has multiple populations, which indicates that there are several possible sources to extract trees and reuse them. To achieve fast and effective knowledge transfer, the relationship between the datasets can be used to provide intuitive hints towards how to extract and use the knowledge across different populations. Based on the relationship $\mathcal{D}_i \subset \mathcal{D}_{train}$, $\mathcal{D}_1 \cup \mathcal{D}_2 \cup \cdots \cup \mathcal{D}_N = \mathcal{D}_{train}$, $\mathcal{D}_i \cap \mathcal{D}_j = \emptyset$ $(i, j \in [1, N]; i \neq j)$, the knowledge transfer route in DCFL is defined as Fig. 3.

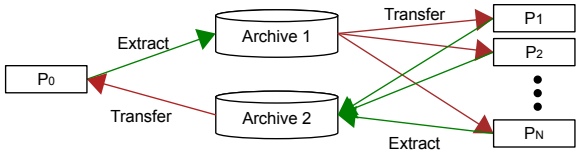


Fig. 3. The knowledge transfer route in DCFL across the $N + 1$ multiple populations at each generation. P_0 uses the whole training set for learning and the other populations use the N subsets of the training set for learning, respectively.

In DCFL, two archives (i.e., $Archive1$ and $Archive2$) are used to store the extracted trees (including subtrees) for transferring. $Archive1$ stores $[S/(N+1)]$ best trees extracted from P_0 and $Archive2$ stores $[S/(N+1)]$ best trees extracted from $P_1, \ldots, P_N$. Specifically, $[S/N(N+1)]$ best trees are extracted from each $P_i (i \in [1, N])$ to form $Archive2$. Note that we limit the storage of $Archive1$ and $Archive2$ to $[S/(N+1)]$ trees in order to save memory. The trees in $Archive1$ and $Archive2$ are the best trees of P_0, $P_1, \ldots, P_N$ of all the past generations. $Archive1$ and $Archive2$ are updated at each generation after fitness evaluation.

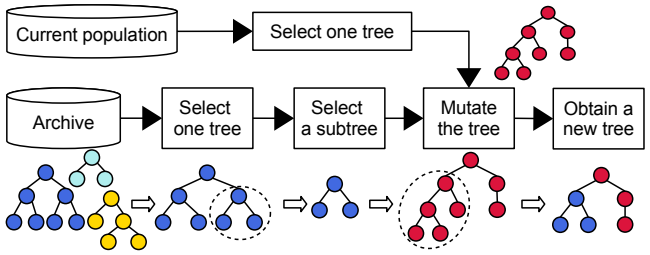


Fig. 4. The process of knowledge transfer in the mutation operation on a GP tree to generate a new tree.

The knowledge transfer is performed in the mutation operation at each generation. As $Archive1$ and $Archive2$ can store best trees, only the crossover and mutation operators are used to generate new populations in DCFL. Standard mutation operator randomly generates a new subtree to replace the subtree from a randomly selected node of the tree, which can add new genetic materials into the population. In DCFL, a new mutation operator is proposed to use the subtrees from the archive. The overall process of knowledge transfer in the mutation process is shown in Fig. 4. The new operator starts with selecting one tree from the archive using tournament selection. Then a subtree is randomly selected from the selected tree. The subtree of the tree to be mutated is replaced by the selected subtree to generate a new tree. The new tree has a subtree selected from the archive, which indicates that the knowledge is successfully transferred. The mutation process is somehow similar to the crossover process, but it focuses on one selected parent each time and only returns one of the offspring. This mutation operator can reduce the randomness of the new population compared to the traditional mutation operator by using transferred knowledge and allow the new approach to have a good exploitation ability. On the other hand, it might reduce the diversity of the population and result in achieving a local optimal. Balancing the diversity and the goodness of the population is typically difficult to achieve in an EC algorithm. In DCFL, the mutation operator with knowledge transfer shows promising results and its effectiveness will be analysed and discussed in Section VI-B.

C. Fitness Evaluation

In many GP-based feature learning algorithms [5, 6], SVM is often used for classifying images. SVM builds multiple strong binary classifiers for a classification problem and cannot directly provide probabilities of prediction. Compared with SVM, LR, also known as softmax regression, can predict confidence scores/probabilities of different classes for each instance, which may be better for building an ensemble. DCFL uses multinomial LR for image classification. In multinomial LR, the probability of an instance $\boldsymbol{x}$ to belong to one of the $C-1$ classes is $P\{y = c|\boldsymbol{x}\}$ and the probability of $\boldsymbol{x}$ to belong

to class C is $P\{y = C|\boldsymbol{x}\}$, which are shown in Eq. (4) and Eq. (5), respectively.

$$P\{y = c|\boldsymbol{x}\} = \frac{e^{\boldsymbol{w_c}^T \cdot \boldsymbol{x} + \beta_c}}{1 + \sum_{c=1}^{C-1} e^{\boldsymbol{w_c}^T \cdot \boldsymbol{x} + \beta_c}}, c = 1, 2, 3, \cdots, C-1, \tag{4}$$

$$P\{y = C|\boldsymbol{x}\} = \frac{1}{1 + \sum_{c=1}^{C-1} e^{\boldsymbol{w_c}^T \cdot \boldsymbol{x} + \beta_c}}, \tag{5}$$

where $\boldsymbol{w_c} \in \mathbb{R}^S$ represents the weights (i.e., $\{w_{c,1}, w_{c,2}, \ldots, w_{c,S}\}$) and β_c represents the bias of the linear model: $\mathbf{w_c}^T \cdot \boldsymbol{x} + \beta_c$. $\boldsymbol{w_c}^T$ represents the vector transpose. The weights $\{\boldsymbol{w_1}, \boldsymbol{w_2}, \ldots, \boldsymbol{w_{C-1}}\}$ and biases $\{\beta_1, \beta_2, \ldots, \beta_{C-1}\}$ can be estimated from training data using maximum likelihood estimation.

The outputs of a LR classifier for an instance x are the probabilities of different classes, such as $P\{y = 0|\boldsymbol{x}\} = 0.1$ and $P\{y = 1|\boldsymbol{x}\} = 0.9$. The commonly used log-loss (also known as cross-entropy loss) function is more suitable for evaluation than the accuracy metric. The accuracy metric only considers the total number of correctly classified instances, which, for example, considers whether $P\{y = 0|\boldsymbol{x}\}$ is larger or smaller than $P\{y = 1|\boldsymbol{x}\}$. However, the log-loss function can provide more information by quantifying the penalty if the predicted probability for the correct class is not one. Specifically, a smaller loss indicates a better classification performance. A zero log-loss indicates that all the instances have predicted probabilities of one in the correct class and zero in the other classes. In DCFL, the fitness function is based on the negative log-loss function, which is to be maximised. In the fitness evaluation, the stratified k-fold cross-validation method is employed to obtain the value of log-loss. Therefore, the fitness function for DCFL (to be maximised) is formulated as Eq. (6).

$$\begin{aligned} Fit &= \frac{1}{K}\sum_{i=1}^{K}(L_i) \\ &= \frac{1}{K}\sum_{i=1}^{K}(\frac{1}{M_i}\sum_{m=1}^{M_i}\sum_{c=1}^{C} y_{m,c}\log p_{m,c}). \end{aligned} \tag{6}$$

where L_i represents the negative log-loss (i.e., log-loss is $-L_i$) on the ith fold. $y_{m,c} \in \{0,\ 1\}$ represents the true label of instance m on class c. $p_{m,c} \in [0,\ 1]$ represents the predicted probability of instance m on class c and $\sum_{c=1}^{C} p_{m,c} = 1$. K represents the number of folds in K-fold cross-validation and M_i represents the total number of instances in fold i. The values of Fit is in the range of $(-\infty,\ 0]$.

The fitness evaluation process in [6] uses a hash table to store a number of evaluated trees to avoid evaluating the same trees again. The fitness evaluation process of DCFL also employs this strategy. Since there are $N + 1$ populations in DCFL, $N + 1$ hash tables are built to store trees and their fitness values in the past generations, respectively. Based on the hash table and the new fitness function, the overall fitness evaluation process is shown in Algorithm 1.

Algorithm 1: Fitness Evaluation

Input : CT_n: the hash table of population n;
$\mathcal{D}_n = \{(x_1, y_1), \ldots, (x_m, y_m)\}$: the training data for population n ($\mathcal{D}_0$ is $\mathcal{D}_{train}$). p: the evaluated tree.
Output : $f(p)$.

```
if p in CT_n then
    f(p) ← the fitness value of CT_n(p);
else
    Use p to transform {x_1, ..., x_m} into features {f_1, ..., f_m};
    Standardise {f_1, ..., f_m} according to the mean and standard
      deviation values;
    Split {f_1, ..., f_m} and {y_1, ..., y_m} into K = 5 folds
      preserving the class ratio;
    for i = 1 to K do
        Feed four folds of data except for the ith fold into logistic
          regression to train a classifier;
        Test the classifier on the ith fold of data;
        L_i ← Calculate the values of negative log-loss on the ith
          fold according to Eq. (6);
    end
    f(p) ← Calculate the average negative log-loss values;
end
Return f(p).
```

D. Ensemble Formulation for Image Classification

After the evolutionary process, DCFL returns the best tree found by each population. Specifically, $N + 1$ best trees, i.e., $I_0, \ldots, I_N$, are obtained. To achieve good generalisation performance on test data, a weighted ensemble is created based on these trees and their weights for image classification. The outline of ensemble formulation is showed in Fig. 5.

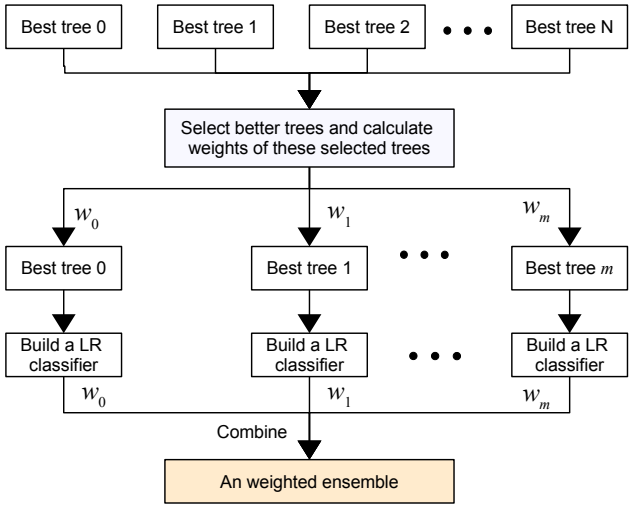


Fig. 5. Outline of ensemble formulation. It selects m trees from the $N + 1$ trees to build m LR classifiers. A weighted ensemble with these m LR classifiers is built for image classification.

In the ensemble formulation process, the performance of trees is calculated and better trees are selected to construct an effective ensemble of more accurate classifiers. The selection is based on the performance of the trees on the whole training set using the fitness function. The better trees are selected by comparing their fitness values (f) with the fitness value (f_0) of the best tree of the first population P_0. It is noted that the fitness values have a large range (i.e., $(-\infty,\ 0]$) and vary with the datasets. We cannot set a fixed threshold for selection. Therefore, f is compared with f_0, and f_0 is typically larger than f. For simplicity purposes, the tree will be selected if its fitness value is in the range $[a * f_0, 0]$. In this approach, we

set a as 2 to the selection and removal of trees. When $a = 2$, the trees with fitness values in $[2f_0,\ 0]$ will be selected.

The main steps to select trees and calculate their weights are summarised as follows:

Step 1: Evaluate each tree I_i ($i \in [1, N]$) on $\mathcal{D}_{train}$ using the fitness function to obtain fitness values f_i. Note that I_0 has its fitness value f_0 on $\mathcal{D}_{train}$ so that it is not evaluated again;

Step 2: Select the trees if their fitness values are between $[2f_0,\ 0]$. Then $m + 1$ trees with their fitness values $\{f_0, \ldots, f_m\}$ ($m \in [0, N]$) are obtained;

Step 3: Calculate the weights of each classifier. First, the fitness values of m selected trees are transformed to positive using $(\sum_{i=0}^{m} f_i)/f_i$ to obtain $\{p_i\}_{i=0}^{m}$ (note that the rank of $\{f_i\}_{i=0}^{m}$ does not change). Second, p_i is scaled into $[0, 1]$ using $p_i/\sum_{i=0}^{m} p_i$ to obtain $\{w_i\}_{i=0}^{m}$, where $\sum_{i=0}^{m} w_i = 1$.

According to the $m + 1$ selected trees and the weights, we can build $m + 1$ classifiers using the corresponding trees and create an ensemble based on the classifiers and the weights. Each classifier is built from a tree I_i ($i \in [0, m]$). In this process, the tree I_i is used to transform images in $\mathcal{D}_{train}$ to features, i.e., $\mathcal{D}^i_{train}$. The features in $\mathcal{D}^i_{train}$ are standardised. Then $\mathcal{D}^i_{train}$ is fed into LR to train the classifier i. Repeating this process, the $m + 1$ classifiers trained using $\mathcal{D}_{train}$ are obtained. The weight for classifier i is w_i ($i \in [0, m]$).

In the test process, the $m+1$ trained classifiers are combined to obtain an ensemble for classifying images in the test set $\mathcal{D}_{test}$ using weighted voting [32]. In the test process, the test set $\mathcal{D}_{test}$ is transformed into features $\mathcal{D}^i_{test}$ using the corresponding tree I_i. The $\mathcal{D}^i_{test}$ is standardised according to $\mathcal{D}^i_{train}$. Then an instance $\boldsymbol{x}$ in $\mathcal{D}^i_{test}$ can be fed into a classifier i to obtain the predicted probabilities for x, i.e., $\{P^i(y = 0|\boldsymbol{x}), \ldots, P^i(y = C|\boldsymbol{x})\}$. Repeating this process, $m+1$ sets of predicted probabilities for instance $\boldsymbol{x}$ are obtained from the $m + 1$ classifiers.

According to the weights $\{w_i\}_{i=0}^{m}$ and the probabilities from m classifier, the weighted probabilities for instance $\boldsymbol{x}$ are calculated using Eq. (7).

$$P(y = c|\boldsymbol{x}) = \sum_{i=0}^{m} w_i \times P^i(y = c|\boldsymbol{x}),\ c = 0, \ldots, C, \quad (7)$$

Based on the weighted probabilities, we can obtain the class label c^* for instance $\boldsymbol{x}$ returned by Eq. (8).

$$c^* = \underset{P(y=c|\boldsymbol{x})}{\operatorname{argmax}}\ \{P(y = 0|\boldsymbol{x}), \ldots, P(y = C|\boldsymbol{x})\}. \quad (8)$$

Finally, the class label for each instance in $\mathcal{D}_{test}$ and the overall classification accuracy on the test set can be calculated.

IV. Experiment Design

A number of experiments have been conducted to show the effectiveness of the proposed approach. This section presents the experiment design, including benchmark datasets, benchmark methods, and parameter settings.

A. Benchmark Datasets

In the experiments, 12 image classification datasets of varying difficulty are employed as benchmark datasets to examine the performance of DCFL. These datasets are FEI_1 [41], FEI_2 [41], KTH [42], FS [43], Amazon [44], MB [45], MRD [45], MBR [45], MBI [45], Rectangle [45], RI [45], and Convex [45]. These datasets represent various representative image classification tasks, i.e., facial expression classification (FEI_1 and FEI_2), texture classification (KTH), scene classification (FS), digit recognition (MB, MRD, MBR, and MBI), and object classification (Amazon, Rectangle, RI and Convex). In addition, these datasets include a wide range of image variations, i.e., rotation (Amazon, MRD), scale (Amazon, KTH), illumination (KTH and FS), noise or background changes (MBR, MBI, FS, and RI). Therefore, these datasets are very suitable for evaluating the performance of DCFL without losing generality. The detailed information about these datasets are listed in Table I. The FEI_1, FEI_2, KTH, FS, and Amazon datasets are randomly split to have 50, 50, 48, 100, and 30 images per class to form the training sets and the remaining images to form the test sets, respectively [6]. Example images from these datasets are shown in Fig. 6. Note that we have also tested the performance of DCFL-FGP on two additional large datasets, i.e., MNIST [11] and Fashion_MNIST [46]. The classification performance and the comparisons between DCFL-FGP and the other reference methods [2, 3] on such datasets are discussed in the supplementary materials.

TABLE I
SUMMARY OF BENCHMARK DATASETS

No.	Dataset	Image Size	Training Set Size	Test Set Size	#Class
1	FEI_1	60×40	150	50	2
2	FEI_2	60×40	150	50	2
3	KTH	50×50	480	330	10
4	FS	55×55	1,300	2,559	13
5	Amazon	50×50	930	1,887	31
6	MB	28×28	12,000	50,000	10
7	MRD	28×28	12,000	50,000	10
8	MBR	28×28	12,000	50,000	10
9	MBI	28×28	12,000	50,000	10
10	Rectangle	28×28	1,200	50,000	2
11	RI	28×28	12,000	50,000	2
12	Convex	28×28	8,000	50,000	2

B. Benchmark Methods

A large number of benchmark methods are employed to show the effectiveness of the proposed approach. The benchmark methods on datasets 1-5 are 14 methods, i.e., six classification algorithms using raw pixels, four SVMs using different preextracted features, two CNNs with different architectures, and two GP-based methods. These methods are SVM, KNN, LR, adaptive boosting (AdaBoost), random forest (RF), extremely randomised trees (ERF), SVM using uniform LBP features (uLBP+SVM), SVM using LBP features (LBP+SVM), SVM using histogram of oriented gradient (HOG) features (HOG+SVM), SVM using scale-invariant feature transform (SIFT) features (SIFT+SVM), CNN with five layers (CNN-5), CNN with eight layers (CNN-8), EGP

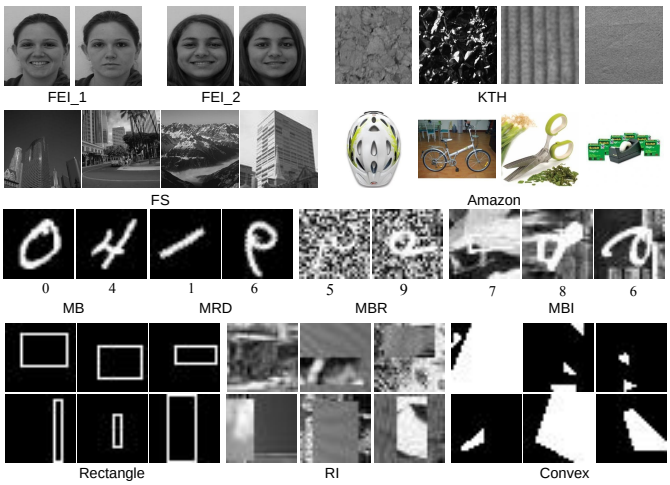


Fig. 6. Example images from the 12 benchmark image classification datasets, respectively. The corresponding class labels of digit images are under the example images.

[47], and FGP [6]. These methods are representative methods for image classification. On datasets 6-12, there are 21 benchmark methods for comparisons. They are SVM+RBF [45], SVM+Poly [45], SAE-3 [48], DAE-b-3 [48], CAE-2 [48], SPAE [49], RBM-3 [48], ScatNet-2 [50, 51], RandNet-2 [51], PCANet-2 (softmax) [51], LDANet-2 [51], NNet [45], SAA-3 [45], DBN-3 [45], FCCNN [52], FCCNN (with BT) [52], SPCN [53], EvoCNN [7], EGP [47], IEGP [20], and FGP [6]. These methods include auto-encoders (SAE-3, DAE-b-2, CAE-2, and SPAE), and CNNs (CNN-5, CNN-8, FCCNN, FCCNN(with BT), SPCN, and EvoCNN). The EvoCNN method is a state-of-the-art deep learning method that uses an evolutionary algorithm to automatically find the architectures of CNNs for image classification. It searches for the best CNN architecture rather than using a predefined architecture to achieve impressive classification performance. It is noted that EvoCNN has achieved better results than several well-known CNNs, including VGG16, GoogleNet, AlexNet, and SqueezeNet [7]. These methods also include state-of-the-art GP-based methods, i.e., EGP (an ensemble method), IEGP (an ensemble methods) and FGP.

Note that the benchmark methods on datasets 1-5 are different from those on datasets 6-12. Datasets 6-12 have publicly available training and test sets so that there are many reported classification results in the literature. The classification results of the 21 benchmark methods are collected from the corresponding papers. These results are state-of-the-art on the corresponding dataset. On datasets 1-5, all the benchmark methods run on the training set and the classification results are obtained on the test set, following the same settings in [6].

C. Parameter Settings

The DCFL framework is used with the FGP tree representation [6] for image classification. The new approach is termed DCFL-FGP in the following sections. The program structure, the function set and the terminal set of FGP can be found in [6]. The parameter settings for DCFL-FGP are based on commonly used settings for GP [6, 28]. In FGP, the population size is 500 and the maximal number of generation is 50 [6]. To achieve fair comparisons, the total population size in DCFL-FGP is 500. Based on preliminary experiments, N is set to 4 as this value induces a good balance between efficiency and effectiveness. With $N = 4$, DCFL-FGP has five small populations with a size of 100, respectively. The crossover rate is 0.8 and the mutation rate is 0.2. The best trees can be stored in the archives (i.e., $Archive1$ and $Archive2$) so that DCFL-FGP does not need elitism. The selection method is tournament selection with size 7. The tree generation method is ramped half-and-half. The tree depth at the initialisation step is 2-6 and the maximal tree depth is 8. Note that DCFL-FGP may evolve trees with depth above 8 because the type constraints are more important than the depth constraint in STGP. In the LR classification algorithm employed for fitness evaluation, stochastic average gradient (SAG) is used for optimising weights and bias as it is fast on large dataset [54]. The number of iterations in LR is set to 50 during the evolutionary learning process to save computational cost and set to 100 in the test process. The other parameters for LR are based on the default settings of the algorithm implemented in *scikit-learn* [38].

The parameter settings for the benchmark methods, i.e., SVM, KNN, LR, RF, AdaBoost, and ERF, refer to [4, 55, 56], where the number of nearest neighbours is 1 in KNN, the number of trees is 500 and the maximal tree depth is 100 in RF and ERF. These algorithms are implemented based on the machine learning package *scikit-learn* [38]. The other parameters are the default ones in this package. The CNN-5 and CNN-8 methods [6] are implemented in *Keras* [57]. ReLU is used as the activation function and softmax is used for classification in the final layer. Dropout is added after the pooling and the first fully connected layer with 0.25 and 0.5 probabilities, respectively, to avoid overfitting [58]. Note that CNN-5 and CNN-8 are just examples of CNNs to show whether the proposed method can achieve better results than simple CNNs on datasets 1-5. The performance of CNNs on datasets 1-5 might be further improved by using deep CNNs with transfer learning or manual tuning of architectures, which are out of the scope of this study.

The DCFL-FGP approach is implemented using the DEAP (*Distributed Evolutionary Algorithm in Python*) [59] package. To further speed up DCFL-FGP, the SCOOP package [60] is employed to use multiple cores for fitness evaluations, i.e., each tree is evaluated on one core and multiple trees are evaluated at the same time using multiple cores. In the experiments, four cores are used to run DCFL-FGP. On each dataset, 30 independent runs of DCFL-FGP have been executed, according to the convention of the EC communities. The benchmark methods on datasets 1-5 also run 30 times for comparisons. The best and average results of the 30 runs are reported. Note that all the reported results are on the test sets, which are unseen to the training and evolutionary processes.

V. Results and Discussions

This section presents the experimental results obtained by the proposed DCFL-FGP approach and the benchmark

TABLE II
CLASSIFICATION ACCURACY (%) OF DCFL-FGP AND FGP ON THE 12 DATASETS. "+"/"–" INDICATES THAT DCFL-FGP ACHIEVES SIGNIFICANTLY BETTER/WORSE RESULTS THAN FGP. "=" INDICATES THAT DCFL-FGP ACHIEVES SIMILAR RESULTS TO FGP

	FGP [6]		**DCFL-FGP**	
Dataset	Max	Mean±St.dev	Max	Mean±St.dev
FEI_1	98.00	94.47±2.67=	98.00	**94.80±2.61**
FEI_2	96.00	91.33±3.36=	**98.00**	**91.40±3.02**
KTH	98.79	96.07±1.13+	**99.09**	**97.84±0.78**
FS	74.48	71.59±1.74+	**80.23**	**78.52±1.09**
Amazon	61.31	57.80±1.35+	**63.01**	**61.07±0.93**
MB	98.82	98.70±0.06+	**98.95**	**98.85±0.05**
MRD	**92.63**	91.56±0.61+	92.58	**91.99±0.40**
MBR	93.46	92.66±0.43=	**93.73**	**92.79±0.41**
MBI	92.52	**89.65±1.44=**	**93.16**	89.49±0.92
Rectangle	100.0	99.88±0.11+	100.0	**99.99±0.02**
RI	93.90	92.66±0.62=	**95.11**	**93.05±1.03**
Convex	98.46	98.16±0.19=	**98.54**	**98.22±0.14**
Total		6+, 6=		

methods on the 12 datasets. This section compares the DCFL-FGP approach with the benchmarks methods in classification accuracy and training time on these benchmark datasets.

A. Classification Performance

1) Comparison with FGP: The classification results, i.e., maximal accuracy (Max), mean accuracy and standard deviation (Mean±St.dev), of DCFL-FGP and FGP are listed in Table II. The Wilcoxon rank-sum test with a 95% significance interval is used to compare DCFL-FGP with FGP. In Table II, the "+" and "=" symbols indicate that DCFL-FGP achieves significantly better, worse and similar results than/to FGP. It is noted that FGP and DCFL-FGP use the same individual representation for feature learning. Unlike FGP, DCFL-FGP uses the proposed DCFL framework.

Table II shows that DCFL-FGP achieves significantly better results than FGP on six datasets and similar results on the remaining six datasets. DCFL-FGP obtains higher maximal accuracy than FGP on 11 datasets except for MRD. It achieves better mean accuracy than FGP on 11 datasets except for MBI. More importantly, DCFL-FGP improves the mean accuracy by 6.93% on FS, for which the challenging task is understanding natural scene images. The classification results show that DCFL-FGP is more effective than FGP on the 12 different image classification tasks. Compared with FGP, DCFL-FGP uses the same tree representation, function set, terminal set, and parameter settings. The results show that the new DCFL framework, i.e., including the knowledge transfer, the fitness function, and the ensemble formulation, is effective for improving the performance of FGP in image classification. With these new designs, DCFL-FGP can find the best trees from multiple small populations and construct an effective ensemble of diverse and effective classifiers using these trees to achieve higher generalisation performance on different image classification datasets.

2) Comparison with Benchmark Methods on Datasets 1-5: The classification results of DCFL-FGP and 13 benchmark methods on datasets 1-5 are listed in Table III. The results show that DCFL-FGP obtains significantly better results than most benchmark methods on these five image classification datasets of varying difficulty. Specifically, DCFL-FGP is significantly better than eight methods and similar to four methods out of the 13 benchmark methods on FEI_1. On FEI_2, DCFL-FGP is significantly better than nine methods and similar to two methods. On KTH, FS, and Amazon, DCFL-FGP performs significantly better than any of the benchmark methods. FEI_1 and FEI_2 are facial expression classification tasks, which are relatively easy so that many benchmark methods achieve high accuracy, e.g., over 90%. KTH and FS are texture classification and scene classification tasks. The benchmark methods achieve very low accuracy on these two datasets, i.e., the mean accuracy is less than 83% on KTH and less than 62% on FS, while DCFL-FGP achieves 97.84% mean accuracy on KTH and 78.52% mean accuracy on FS. Amazon represents a difficult object classification task so that most benchmark methods achieve very low accuracy. On Amazon, DCFL-FGP achieves significantly better accuracy than any of the benchmark methods. The results show that DCFL-FGP is effective for solving different types of image classification tasks.

3) Comparison with Benchmark Methods on Datasets 6-12: The classification accuracy of DCFL-FGP and 21 benchmark methods on datasets 6-12 are listed in Table IV. The datasets 6-12 have public training and test sets so that the accuracy of these benchmark methods on the test set is collected from the corresponding papers. On these datasets, we only compare the best results of these methods because most benchmark methods only report the best results.

In Table IV, we can observe that DCFL-FGP achieves better results than most benchmark methods on the seven datasets. Since these benchmark methods have achieved high accuracy on these datasets, any further improvement in accuracy is difficult. In addition, most benchmark methods are (deep) neural network-based methods, which are known as powerful methods for image classification. Compared with these benchmark methods, DCFL-FGP, as a pure GP method, ranks first among all the methods on four datasets, i.e., MB, Rectangle, RI, and Convex, ranks second on the two datasets, i.e., MBR and MBI, and ranks fourth on the remaining one dataset, i.e., MRD. On MB, DCFL-FGP and LDANet-2 find the best accuracy of 98.95%. However, LDANet-2 only achieves 87.58% on MBI, which is a variant of MB by adding additional image background, and 83.80% on RI, which is a variant of Rectangle by adding additional background noise. This indicates that the performance of LDANet-2 is significantly affected by these additional noises in the MBI and RI datasets. Compared with LDANet-2, DCFL-FGP is less affected by achieving 93.16% accuracy on MBI and 95.11% accuracy on RI. The EvoCNN method, which is a state-of-the-art deep learning method that uses an evolutionary algorithm to search for the best CNN architecture for image classification achieves the best results on the MRD, MBR and MBI datasets. These three datasets are difficult due to additional variations. EvoCNN requires graphics processing unit (GPU) implementation and uses extensive computational resources to find the best architecture for CNN so that it can achieve better classification accuracy. In contrast, DCFL-

TABLE III
CLASSIFICATION ACCURACY (%) OF DCFL-FGP AND 13 BENCHMARK METHODS ON DATASETS 1-5 . "+"/"–" INDICATES THAT DCFL-FGP ACHIEVES SIGNIFICANTLY BETTER/WORSE RESULTS THAN THE COMPARED METHOD. "=" INDICATES THAT DCFL-FGP ACHIEVES SIMILAR RESULTS TO THE COMPARED METHOD

	FEI_1		**FEI_2**		**KTH**		**FS**		**Amazon**	
Method	Max	Mean±St.dev	Max	Mean±St.dev	Max	Mean±St.dev	Max	Mean±St.dev	Max	Mean±St.dev
SVM	90.00	90.00±0.00+	88.00	88.00±0.00+	46.97	44.59±2.83+	20.63	20.30±0.15+	32.17	31.37±0.31+
KNN	32.00	32.00±0.00+	8.00	8.00±0.00+	34.24	34.24±0.00+	24.35	24.35±0.00+	29.57	29.57±0.00+
LR	92.00	92.00±0.00+	88.00	88.00±0.00+	48.79	48.79±0.00+	23.49	23.49±0.00+	35.19	35.19±0.00+
RF	98.00	**97.07±1.01**–	90.00	89.20±1.13+	60.00	57.81±0.83+	37.36	36.53±0.49+	46.90	46.16±0.41+
AdaBoost	80.00	78.67±1.32+	80.00	76.00±3.44+	37.88	33.44±1.37+	17.47	13.04±1.47+	7.47	7.42±0.01+
ERF	94.00	93.27±0.98+	92.00	90.60±0.93=	61.52	59.83±0.86+	37.94	37.15±0.36+	48.49	47.55±0.38+
uLBP+SVM	66.00	56.73±3.66+	68.00	62.53±3.52+	78.79	73.29±4.18+	49.79	33.27±8.90+	35.14	35.14±0.00+
LBP+SVM	68.00	64.60±1.83+	74.00	69.80±0.00+	83.64	82.71±0.51+	53.50	50.45±1.80+	39.90	39.90±0.00+
HOG+SVM	96.00	96.00±0.00=	82.00	82.00±0.00+	57.27	55.96±0.64+	12.11	7.91±2.47+	26.87	26.87±0.00+
SIFT+SVM	56.00	56.00±0.00+	62.00	62.00±0.00+	65.76	65.76±0.00+	60.92	60.92±0.00+	51.46	51.46±0.00+
CNN-5	98.00	95.40±1.30=	98.00	95.27±1.62–	85.76	82.56±1.87+	50.14	48.03±1.16+	43.19	40.06±1.26+
CNN-8	98.00	95.33±1.32=	96.00	90.93±1.87=	76.36	71.63±3.18+	49.16	46.79±1.01+	46.26	43.94±1.26+
EGP [47]	**100.0**	96.20±2.06=	**100.0**	**98.07±1.70**–	87.88	77.53±5.17+	67.17	61.07±2.91+	**63.75**	56.77±2.87+
DCFL-FGP	98.00	94.80±2.61	98.00	91.40±3.02	**99.09**	**97.84±0.78**	**80.23**	**78.52±1.09**	63.01	**61.07±0.93**
Overall		8+, 4=, 1–		9+, 2=, 2–		13+		13+		13+

TABLE IV
CLASSIFICATION ACCURACY (%) OF DCFL-FGP AND 21 BENCHMARK METHODS ON DATASETS 6-12. "↑" INDICATES THAT DCFL-FGP ACHIEVES BETTER ACCURACY THAN THE COMPARED METHOD

Method	**MB**	**MRD**	**MBR**	**MBI**	**Rectangle**	**RI**	**Convex**
SVM+RBF [45]	96.97 (↑)	88.89 (↑)	85.42 (↑)	77.39 (↑)	97.85 (↑)	75.96 (↑)	80.87 (↑)
SVM+Poly [45]	96.31 (↑)	84.58 (↑)	83.38 (↑)	75.99 (↑)	97.85 (↑)	75.95 (↑)	80.18 (↑)
SAE-3 [48]	96.54 (↑)	89.70 (↑)	88.72 (↑)	77.00 (↑)	97.86 (↑)	75.95 (↑)	–
DAE-b-3 [48]	97.16 (↑)	90.47 (↑)	89.70 (↑)	83.32 (↑)	98.01 (↑)	78.41 (↑)	–
CAE-2 [48]	97.52 (↑)	90.34 (↑)	89.10 (↑)	84.50 (↑)	98.79 (↑)	78.46 (↑)	–
SPAE [49]	96.68 (↑)	89.74 (↑)	90.99 (↑)	86.76 (↑)	–	–	–
RBM-3 [48]	96.89 (↑)	89.70 (↑)	93.27 (↑)	83.69 (↑)	97.40 (↑)	77.50 (↑)	–
ScatNet-2 [50, 51]	98.73 (↑)	92.52 (↑)	87.70 (↑)	81.60 (↑)	99.99 (↑)	91.98 (↑)	93.50 (↑)
RandNet-2 [51]	98.75 (↑)	91.53 (↑)	86.53 (↑)	88.35 (↑)	99.91 (↑)	83.00 (↑)	94.55 (↑)
PCANet-2 (softmax) [51]	98.60 (↑)	91.48 (↑)	93.15 (↑)	88.45 (↑)	99.51 (↑)	86.61 (↑)	95.81 (↑)
LDANet-2 [51]	**98.95**	92.48 (↑)	93.19 (↑)	87.58 (↑)	99.86 (↑)	83.80 (↑)	92.78 (↑)
NNet [45]	95.31 (↑)	81.89 (↑)	79.96 (↑)	72.59 (↑)	92.84 (↑)	66.80 (↑)	67.75 (↑)
SAA-3 [45]	96.54 (↑)	89.70 (↑)	88.72 (↑)	77.00 (↑)	97.59 (↑)	75.95 (↑)	81.59 (↑)
DBN-3 [45]	96.89 (↑)	89.70 (↑)	93.27 (↑)	83.69 (↑)	97.40 (↑)	77.50 (↑)	81.37 (↑)
FCCNN [52]	97.57 (↑)	91.09 (↑)	93.55 (↑)	86.77 (↑)	–	–	–
FCCNN (with BT) [52]	97.32 (↑)	90.41 (↑)	93.03 (↑)	89.20 (↑)	–	–	–
SPCN [53]	98.18 (↑)	90.19 (↑)	94.16	90.45 (↑)	99.81 (↑)	89.40 (↑)	–
EvoCNN (best) [7]	98.82 (↑)	**94.78**	**97.20**	**96.47**	99.99 (↑)	94.97 (↑)	95.18 (↑)
EGP (best) [47]	97.19 (↑)	–	–	–	99.91 (↑)	–	93.97 (↑)
IEGP (best) [20]	98.82 (↑)	94.28	93.59 (↑)	89.41 (↑)	**100**	94.88 (↑)	98.26 (↑)
FGP (best) [6]	98.82 (↑)	92.63	93.46 (↑)	92.52 (↑)	**100**	93.90 (↑)	98.46 (↑)
DCFL-FGP (best)	**98.95**	92.58	93.73	93.16	**100**	**95.11**	**98.54**
Rank	**1/22**	4/21	2/21	**2/21**	**1/19**	**1/18**	**1/14**

FGP is implemented on central processing unit (CPU) and the computational cost is affordable (or even less). Although DCFL-FGP achieves slightly worse results on these three datasets, it is noted that it achieves better results than EvoCNN on the remaining four datasets. Compared with EGP, IEGP and FGP, which are state-of-the-art GP-based algorithms, DCFL-FGP achieves better results on six datasets except for MRD, on which IEGP achieves slightly better accuracy. IEGP is an ensemble method that automatically learns features and builds many different classifiers (e.g., SVM, RF and LR) to construct ensembles. DCFL-FGP only uses the LR classifier to construct ensembles but achieves similar or even better performance than IEGP. The results on these seven datasets demonstrate that DCFL-FGP is more effective than existing methods by achieving better results in almost all the comparisons.

B. Training Time

The DCFL-FGP approach is sped up by the SCOOP [60] package to use four CPU cores for fitness evaluations. To achieve fair comparisons, we also run FGP using four cores and the SCOOP package. Due to the high computational cost of FGP, we only run it on the FEI_1, FEI_2, KTH, FS, Amazon, and Rectangle datasets. The training time of DCFL-FGP and FGP on the six datasets are shown in Fig. 7. The computation time of DCFL-FGP on all the 12 benchmark datasets are shown in Fig. 8.

Figure 7 shows that DCFL-FGP uses significantly shorter training time than FGP on these six datasets. Specifically, DCFL-FGP is nearly twice faster than FGP on the FEI_1 and Amazon datasets, nearly three times faster on the FEI_2 and Rectangle datasets, three times faster on the KTH dataset, and

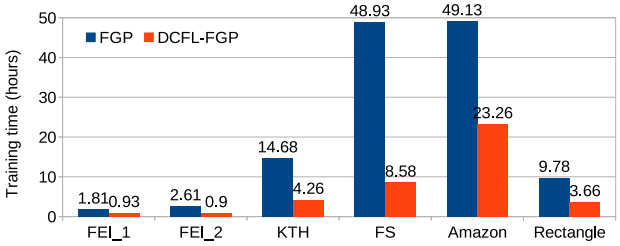


Fig. 7. Training/computation time (hours) of DCFL-FGP and FGP on the FEI_1, FEI_2, KTH, FS, Amazon and Rectangle datasets.

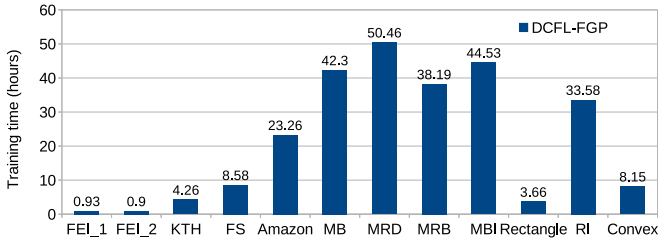


Fig. 8. Training/computation time (hours) of DCFL-FGP on the 12 benchmark datasets.

five times faster on the FS dataset. This indicates that DCFL-FGP significantly improved the computational efficiency of FGP. Compared with FGP, DCFL-FGP has a new algorithm framework that splits the training set and the whole population into small subsets and small populations. By evaluating small populations on small subsets, DCFL-FGP uses significantly shorter training time than FGP on these datasets. These comparisons show that the objective of improving the efficiency of GP for image classification has been successfully achieved.

To further analyse the computational cost of DCFL-FGP, we compare the running time of DCFL-FGP with a state-of-the-art deep learning method and many other benchmark methods, including EvoCNN, DAE-b-3, SAE-3, DBN-3, PCANet, and FCCNN. Figure 8 shows that DCFL-FGP uses about 38 to 50 hours on the MB, MRD, MBR, and MBI datasets, 33.6 hours on the RI dataset, and 8.2 hours on the Convex dataset. The state-of-the-art method, EvoCNN, needs 2 to 3 days to run the experiments using two identical GTX1080 GPU cards on these benchmarks [7]. The training time of DCFL-FGP is shorter than or similar to that of EvoCNN if we do not consider other factors. However, the computational speed of GPUs is typically faster than that of CPUs for CNNs and image operations [61, 62] (i.e., CNNs for image classification is high inherent parallelism) and the comparisons are not entirely fair. The training time of some benchmark methods, i.e., DAE-b-3, SAE-3, DBN-3, PCANet, and FCCNN, can be found from the literature [51, 52]. DAE-b-3, SAE-3 and DBN-3 use more than three hours on the MB dataset. PCANet uses less than one hour on MB. FCCNN uses 5 to 30 minutes on MB. Compared with these methods, DCFL-FGP uses longer training time on MB. However, such simple comparisons are not entirely fair due to lack of detailed information of the experiment running environment. DCFL-FGP is an EC-based method, which is typically more time-consuming than non-EC-based methods due to a large number of fitness evaluations during the evolutionary process. It is noted that the training of DCFL-FGP can be offline and its test process is fast.

C. Summary

To sum up, the results show that DCFL-FGP is an effective and promising approach to feature learning for image classification. Compared with the original FGP method, DCFL-FGP not only improves classification performance but also significantly reduces computation time. Compared with a large number of existing image classification algorithms, DCFL-FGP achieves better results in most comparisons, which further indicates the effectiveness of DCFL-FGP. The extensive experimental results on different datasets demonstrate that DCFL-FGP can effectively solve different types of image classification tasks. The results indicate that DCFL-FGP achieves high generalisation performance by constructing an effective ensemble of diverse and accurate classifiers using the best trees found by small populations from different subsets of training data. The comparisons of DCFL-FGP with FGP and EvoCNN in training time show that the efficiency of DCFL-FGP is improved by using small populations to learn features from small subsets of the training data.

VI. Further Analysis

This section further analyses the ensembles constructed from the trees of DCFL-FGP to show why it can achieve good classification performance. Further empirical analysis is conducted to analyse whether and why the knowledge transfer is effective in DCFL-FGP.

A. Analysis on the Constructed Ensembles

DCFL-FGP outputs a set of best trees found by multiple small populations and creates an ensemble based on these trees for image classification. Note that N is set to 4 in DCFL-FGP so that there are five small populations that return five trees. In each constructed ensemble, there are at most five classifiers. Each classifier is obtained using one GP tree. Therefore, the performance of the five classifiers on the test set and the weights of the five classifiers in the constructed ensemble are analysed to show why DCFL-FGP can achieve high generalisation performance.

Fig. 9 shows the distributions of the classification accuracy (%) obtained by every single classifier and the constructed ensemble on the test set of MB. Note that the results have been obtained in 30 independent runs. From Fig. 9, it can be observed that the classification performance of all the five classifiers is between 98.1% to 98.8%. Among all the five classifiers, the best one is the first one, which is obtained using the best tree found from the whole training set. The other four classifiers are obtained using the best trees found from the small subsets of the training set. This shows that using a small number of training instances could reduce the generalisation performance of the trees. However, the differences in the accuracy between all the classifiers are not very big. One reason may be that the knowledge transfer between the multiple small populations improves generalisation performance.

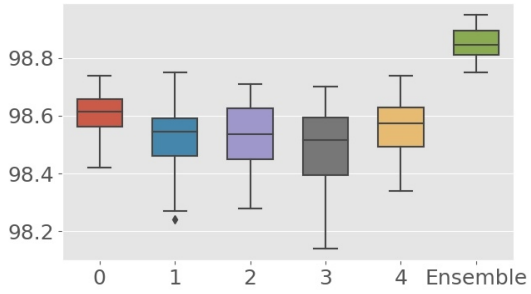


Fig. 9. The distributions of classification accuracy (%) obtained by every single classifier and the constructed ensemble on the MB dataset. Every single classifier is obtained using the best tree found by a small population in DCFL-FGP. The ensemble is built from the five classifiers using the ensemble formulation strategy.

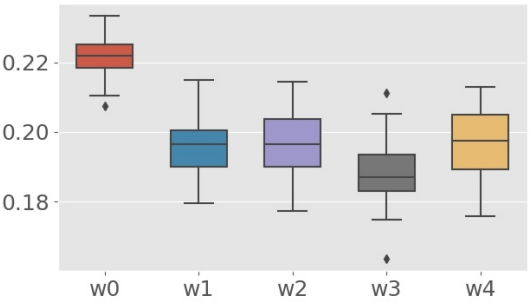


Fig. 10. The distributions of weights of the five classifiers in the constructed ensemble on the MB dataset. The five classifier are obtained using the five best trees found by DCFL-FGP.

Comparing the results of the five classifiers with the ensemble in Fig. 9, it is clear that the constructed ensemble achieves better generalisation performance than any of the five classifiers. More importantly, the lowest accuracy of the ensembles in the 30 runs is better than the highest accuracy of the five classifiers on the MB dataset. In general, to obtain a good ensemble, the classifiers should be diverse and accurate [56]. Since each classifier is trained using trees from the different small populations, it is clear that the diversity of the classifiers in the ensemble is high. Since the accuracy of every single classifier is also high, the generalisation performance of the constructed ensemble is better than any of the single classifiers. It is noted that the accuracy of the constructed ensemble is better than any of the benchmark methods on MB although the best accuracy of the five classifiers is worse than some benchmark methods, such as FGP and RandNet-2.

Fig. 10 shows the distributions of weights of the five classifiers in the ensembles on the MB dataset. The weights are calculated according to the performance of the trees found by the small populations on the whole training set, which has been described in Section III-D. From Fig. 10, it is obvious that the weight of the first classifier is higher than those of the other four classifiers. From Fig. 9 and Fig. 10, we can find that the classifier that has a larger weight in the ensemble also obtains a higher accuracy on the test set of MB. This shows that the ensemble formulation strategy can combine the classifiers well to obtain an effective ensemble with high generalisation performance.

B. Effectiveness of Knowledge Transfer

To analyse the effectiveness of knowledge transfer in DCFL-FGP, the baseline method without knowledge transfer is used for comparisons. Six different datasets of various numbers of instances are used to conduct the experiments. The parameter settings for DCFL-FGP with and without knowledge transfer (KT) are the same. The classification results of these two methods are listed in Table V. It can be observed that DCFL-FGP with KT achieves significantly better or similar performance than/to that without KT on the six different datasets. On five of the six datasets, DCFL-FGP with KT achieves better maximal accuracy than DCFL-FGP without KT. On the remaining dataset (FEI_1), DCFL-FGP with KT achieves the same maximal accuracy to that without KT. On the FEI_1, FEI_2 and Convex datasets, DCFL-FGP with KT achieves better mean accuracy than DCFL-FGP without KT. On the KTH, FS and MB datasets, DCFL-FGP without KT achieves better mean accuracy than DCFL-FGP with KT, but the results of the 30 runs are not significantly different. From Table V, it is noted that DCFL-FGP without KT can achieve comparable performance on some datasets. Without KT, the trees found by DCFL-FGP are more diverse but less accurate, which might lead to creating an effective ensemble of diverse classifiers to achieve high generalisation performance in image classification. In DCFL-FGP with KT, subtrees of the best trees found by small populations can be extracted and reused in the mutation operation during the evolutionary process. This can help improve the learning performance of every single small population and find the best tree with better performance. With multiple effective trees, an effective ensemble can be constructed for image classification to achieve high generalisation performance. The results show that knowledge transfer can improve the learning performance of DCFL-FGP.

TABLE V
CLASSIFICATION ACCURACY (%) OF DCFL-FGP WITH AND WITHOUT KNOWLEDGE TRANSFER (KT) ON SIX DATASETS. "+" DENOTES THAT DCFL-FGP IS SIGNIFICANTLY BETTER AND "=" DENOTES DCFL-FGP ACHIEVES SIMILAR RESULTS TO DCFL-FGP WITHOUT KT

	DCFL-FGP without KT		**DCFL-FGP with KT**	
Dataset	Max	Mean±St.dev	Max	Mean±St.dev
FEI_1	98.00	93.47±2.83=	98.00	94.80±2.61
FEI_2	96.00	88.73±5.08+	**98.00**	**91.40±3.02**
KTH	98.79	98.16±0.43=	**99.09**	97.84±0.78
FS	79.95	78.63±0.74=	**80.23**	78.52±1.09
MB	98.91	98.86±0.04=	**98.95**	98.85±0.05
Convex	98.12	97.86±0.24+	**98.54**	**98.22±0.14**
Total		2+, 4=		

VII. CONCLUSIONS

The overall goal of this paper was to develop a new algorithm framework than can effectively improve the computational efficiency of the GP-based feature learning algorithms without sacrificing the generalisation performance in image classification. This goal has been successfully achieved by developing the divide-and-conquer feature learning (DCFL)

framework and combining it with a GP representation (FGP) to solve different image classification tasks of varying difficulty. DCFL splits the training set into several non-overlapping subsets and the whole population into several small populations of trees so that each tree is evaluated on a small number of instances. To improve the generalisation performance, a knowledge transfer method, a fitness function, and an ensemble method were developed in the DCFL approach. With these designs, DCFL can build an effective ensemble using the best trees found by the different small populations for image classification. Extensive experiments were conducted to demonstrate the effectiveness of DCFL-FGP on different types of image classification tasks.

Compared with FGP without the DCFL framework, DCFL-FGP obtained better generalisation performance at a much lower computational cost. Compared with other effective methods, DCFL-FGP achieved better performance in almost all the comparisons on the 12 different image classification datasets. The results showed that DCFL-FGP is a promising and effective approach to image classification. Compared with a state-of-the-art deep learning method (i.e. EvoCNN), DCFL-FGP achieved better or comparable performance and used fewer computational resources. The analysis also showed that the constructed ensembles obtained better performance than single individual classifiers, which confirmed the effectiveness of the proposed ensemble formulation strategy in DCFL. Further analysis showed that knowledge transfer can improve the learning performance of DCFL-FGP, leading to an effective ensemble for image classification.

The DCFL method improved the computational efficiency of GP for image classification. However, the problem of the high computational cost of GP has not been completely solved. In the future, it is still necessary to develop new computationally cheap GP methods for image classification. Other future work will focus on exploring the ideas of island-based models and federated learning in GP for image classification and extending DCFL for online learning.

REFERENCES

[1] A. Latif, A. Rasheed, U. Sajid, J. Ahmed, N. Ali, N. I. Ratyal, B. Zafar, S. H. Dar, M. Sajid, and T. Khalil, "Content-based image retrieval and feature extraction: A comprehensive review," *Math. Probl. Eng.*, no. 9658350, p. 21pp, 2019.

[2] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[3] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1, no. 2.

[4] H. Al-Sahaf, A. Al-Sahaf, B. Xue, M. Johnston, and M. Zhang, "Automatically evolving rotation-invariant texture image descriptors by genetic programming," *IEEE Trans. Evol. Comput.*, vol. 21, no. 1, pp. 83–101, 2017.

[5] L. Shao, L. Liu, and X. Li, "Feature learning for image classification via multiobjective genetic programming," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 7, pp. 1359–1371, 2014.

[6] Y. Bi, B. Xue, and M. Zhang, "Genetic programming with image-related operators and a flexible program structure for feature learning to image classification," *IEEE Trans. Evol. Comput.*, vol. 25, no. 1, pp. 87–101, 2021.

[7] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 1–14, 2019.

[8] Y. Bi, B. Xue, and M. Zhang, *Genetic Programming for Image Classification: An Automated Approach to Image Classification*. Springer International Publishing, XXVIII, 258pp, 2021, DOI: https://doi.org/10.1007/978-3-030-65927-1.

[9] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT press, Cambridge, 1992.

[10] Y. Bi, B. Xue, and M. Zhang, "Genetic programming for automatic global and local feature extraction to image classification," in *Proc. IEEE CEC*, 2018, pp. 1–8.

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

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. IEEE CVPR*, 2009, pp. 248–255.

[13] S. Nguyen, M. Zhang, and K. C. Tan, "Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules," *IEEE Trans. Cybern.*, vol. 47, no. 9, pp. 2951–2965, 2017.

[14] M. E. Roberts, "The effectiveness of cost based subtree caching mechanisms in typed genetic programming for image segmentation," in *Proc. Workshop Appl. Evol. Comput.* Springer, 2003, pp. 444–454.

[15] H. Wang, Y. Jin, and J. O. Jansen, "Data-driven surrogate-assisted multiobjective evolutionary optimization of a trauma system," *IEEE Trans. Evol. Comput.*, vol. 20, no. 6, pp. 939–952, 2016.

[16] A. Song, W. Chen, Y. Gong, X. Luo, and J. Zhang, "A divide-and-conquer evolutionary algorithm for large-scale virtual network embedding," *IEEE Trans. Evol. Comput.*, vol. 24, no. 3, pp. 566–580, 2020.

[17] F. Lardeux and A. Goëffon, "A dynamic island-based genetic algorithms framework," in *Proc. SEAL*. Springer, 2010, pp. 156–165.

[18] Y. Bi, B. Xue, and M. Zhang, "A gaussian filter-based feature learning approach using genetic programming to image classification," in *Proc. Austra. Joint Conf. Art. Intell.* Springer, 2018, pp. 251–257.

[19] D. J. Montana, "Strongly typed genetic programming," *Evol. Comput.*, vol. 3, no. 2, pp. 199–230, 1995.

[20] Y. Bi, B. Xue, and M. Zhang, "Genetic programming with a new representation to automatically learn features and evolve ensembles for image classification," *IEEE Trans. Cybern.*, vol. 51, no. 4, pp. 1769–1783, 2021.

[21] L. Rodriguez-Coayahuitl, A. Morales-Reyes, and H. J. Escalante, "Structurally layered representation learning: Towards deep learning through genetic programming," in *Proc. EuroGP*. Springer, 2018, pp. 271–288.

[22] B. Tran, B. Xue, and M. Zhang, "Genetic programming for multiple-feature construction on high-dimensional classification," *Pattern Recognit.*, vol. 93, pp. 404–417, 2019.

[23] W. La Cava, S. Silva, K. Danai, L. Spector, L. Vanneschi, and J. H. Moore, "Multidimensional genetic programming for multiclass classification," *Swarm Evol. Comput.*, vol. 44, pp. 260–272, 2019.

[24] H. Al-Sahaf, Y. Bi, Q. Chen, A. Lensen, Y. Mei, Y. Sun, B. Tran, B. Xue, and M. Zhang, "A survey on evolutionary machine learning," *J. Roy. Soc. New Zeal.*, vol. 49, no. 2, pp. 205–228, 2019.

[25] D. Yazdani, M. N. Omidvar, J. Branke, T. T. Nguyen, and X. Yao, "Scaling up dynamic optimization problems: A divide-and-conquer approach," *IEEE Trans. Evol. Comput.*, vol. 24, no. 1, pp. 1–15, 2020.

[26] K. Ono, Y. Hanada, M. Kumano, and M. Kimura, "Enhancing island model genetic programming by controlling frequent trees," *J. Artif. Intell. Soft Comput. Res.*, vol. 9, no. 1, pp. 51–65, 2019.

[27] A. Gupta, Y.-S. Ong, and L. Feng, "Insights on transfer optimization: Because experience is the best teacher," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, no. 1, pp. 51–64, 2017.

[28] M. Iqbal, B. Xue, H. Al-Sahaf, and M. Zhang, "Cross-domain reuse of extracted knowledge in genetic programming for image classification," *IEEE Trans. Evol. Comput.*, vol. 21, no. 4, pp. 569–587, 2017.

[29] M. Jiang, Z. Huang, L. Qiu, W. Huang, and G. G. Yen, "Transfer learning-based dynamic multiobjective optimization algorithms," *IEEE Trans. Evol. Comput.*, vol. 22, no. 4, pp. 501–514, 2017.

[30] A. Gupta, Y.-S. Ong, and L. Feng, "Multifactorial evolution: toward evolutionary multitasking," *IEEE Trans. Evol. Comput.*, vol. 20, no. 3, pp. 343–357, 2015.

[31] J. Ding, C. Yang, Y. Jin, and T. Chai, "Generalized multitasking for evolutionary optimization of expensive problems," *IEEE Trans. Evol. Comput.*, vol. 23, no. 1, pp. 44–58, 2017.

[32] Z.-H. Zhou, *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC, 2012.

[33] A. Rosales-Pérez, S. García, J. A. Gonzalez, C. A. C. Coello, and F. Herrera, "An evolutionary multiobjective model and instance selection for support vector machines with Pareto-based ensembles," *IEEE Trans. Evol. Comput.*, vol. 21, no. 6, pp. 863–877, 2017.

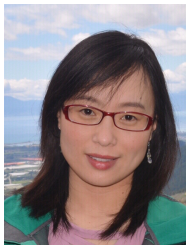
[34] B. Zhang, Y. Chen, W. Fan, E. A. Fox, M. Gonçalves, M. Cristo, and P. Calado, "Intelligent GP fusion from multiple sources for text

classification," in *Proc. CIKM*, 2005, pp. 477–484.
[35] K. Nag and N. R. Pal, "A multiobjective genetic programming-based ensemble for simultaneous feature selection and classification," *IEEE Trans. Cybern.*, vol. 46, no. 2, pp. 499–510, 2016.
[36] N. M. Rodrigues, J. E. Batista, and S. Silva, "Ensemble genetic programming," in *Proc. EuroGP*. Springer, 2020, pp. 151–166.
[37] Y. Bi, B. Xue, and M. Zhang, "Evolving deep forest with automatic feature extraction for image classification using genetic programming," in *Proc. PPSN*. Springer, 2020, pp. 3–18.
[38] F. Pedregosa, G. Varoquaux, A. Gramfort, and et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
[39] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, p. 27, 2011.
[40] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data En.*, vol. 22, no. 10, pp. 1345–1359, 2009.
[41] C. E. Thomaz, "Fei face database," *online: http://fei.edu.br/˜cet/facedatabase.html*, 2012.
[42] P. Mallikarjuna, A. T. Targhi, M. Fritz, E. Hayman, B. Caputo, and J.-O. Eklundh, "The kth-tips2 database," *Computational Vision and Active Perception Laboratory, Stockholm, Sweden*, pp. 1–10, 2006.
[43] L. Fei-Fei and P. Perona, "A bayesian hierarchical model for learning natural scene categories," in *Proc. IEEE CVPR*, vol. 2, 2005, pp. 524–531.
[44] K. Saenko, B. Kulis, M. Fritz, and T. Darrell, "Adapting visual category models to new domains," in *Proc. ECCV*. Springer, 2010, pp. 213–226.
[45] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proc. ICML*. ACM, 2007, pp. 473–480.
[46] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.
[47] Y. Bi, B. Xue, and M. Zhang, "An automated ensemble learning framework using genetic programming for image classification," in *Proc. GECCO*, 2019, pp. 365–373.
[48] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Proc. ICML*. Omnipress, 2011, pp. 833–840.
[49] T. Yu, C. Guo, L. Wang, S. Xiang, and C. Pan, "Self-paced autoencoder," *IEEE Signal Proc. Let.*, vol. 25, no. 7, pp. 1054–1058, 2018.
[50] J. Bruna and S. Mallat, "Invariant scattering convolution networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1872–1886, 2013.
[51] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "PCANet: A simple deep learning baseline for image classification?" *IEEE Trans. Image Process.*, vol. 24, no. 12, pp. 5017–5032, 2015.
[52] G. Qian and L. Zhang, "A simple feedforward convolutional conceptor neural network for classification," *Appl. Soft Comput.*, vol. 70, pp. 1034–1041, 2018.
[53] H. Li and M. Gong, "Self-paced convolutional neural networks," in *Proc. IJCAI*, 2017, pp. 2110–2116.
[54] M. Schmidt, N. Le Roux, and F. Bach, "Minimizing finite sums with the stochastic average gradient," *Math. Program.*, vol. 162, no. 1-2, pp. 83–112, 2017.
[55] S. Young, T. Abdou, and A. Bener, "Deep super learner: A deep ensemble for classification problems," in *Proc. 31st Can. Conf. Art. Intell.* Springer, 2018, pp. 84–95.
[56] Z.-H. Zhou and J. Feng, "Deep forest," *Natl. Sci. Rev.*, vol. 6, no. 1, pp. 74–86, 2018.
[57] F. Chollet *et al.*, "Keras," https://keras.io, 2015.
[58] 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, 2014.
[59] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *J. Mach. Learn. Res.*, vol. 13, no. Jul, pp. 2171–2175, 2012.
[60] Y. Hold-Geoffroy, O. Gagnon, and M. Parizeau, "Once you SCOOP, no need to fork," in *Proc. ACM XSEDE*, 2014, p. 60.
[61] E. László, P. Szolgay, and Z. Nagy, "Analysis of a gpu based cnn implementation," in *Proc. IEEE CNNA*, 2012, pp. 1–5.
[62] S. Potluri, A. Fasih, L. K. Vutukuru, F. Al Machot, and K. Kyamakya, "Cnn based high performance computing for real time image processing on gpu," in *Proc. IEEE INDS'11 & ISTET'11*, 2011, pp. 1–7.

Ying Bi (M'17) received the B.Sc.degree in 2013 from Wuhan Polytechnic University, Hubei, China, the M.Sc. degree in 2016 from Shenzhen University, Shenzhen, China, and the Ph.D. degree in 2020 from Victoria University of Wellington, New Zealand.

She is currently a Post-Doctoral Research Fellow with the School of Engineering and Computer Science, Victoria University of Wellington. Her current research interests include evolutionary computation, computer vision, and machine learning. She has published over 30 papers in this field, including top journals and conference papers.

She is a member of the IEEE Computational Intelligence Society and has been severing as reviewers for top international journals and conferences, such as IEEE Transactions on Evolutionary Computation, IEEE Transactions on Cybernetics, IEEE Congress on Evolutionary Computation (CEC), and the Genetic and Evolutionary Computation Conference (GECCO).



Bing Xue (M'10-SM'21) received the B.Sc. degree from the Henan University of Economics and Law, Zhengzhou, China, in 2007, the M.Sc. degree in management from Shenzhen University, Shenzhen, China, in 2010, and the Ph.D. degree in computer science in 2014 at Victoria University of Wellington (VUW), New Zealand.

She is currently a Professor in Computer Science, and Program Director of Science in the School of Engineering and Computer Science at VUW. She has over 300 papers published in fully refereed international journals and conferences and her research focuses mainly on evolutionary computation, machine learning, classification, symbolic regression, feature selection, evolving deep neural networks, image analysis, transfer learning, multi-objective machine learning.

Dr Xue is currently the Chair of IEEE Computational Intelligence Society (CIS) Task Force on Transfer Learning & Transfer Optimization, and Vice-Chair of IEEE CIS Evolutionary Computation Technical Committee, Editor of IEEE CIS Newsletter,. Vice-Chair of IEEE Task Force on Evolutionary Feature Selection and Construction, and Vice-Chair IEEE CIS Task Force on Evolutionary Deep Learning and Applications. She has also served as associate editor of several international journals, such as IEEE Computational Intelligence Magazine and IEEE Transactions on Evolutionary Computation and IEEE Transactions on Artificial Intelligence.

Mengjie Zhang (M'04-SM'10-F'19) received the B.E. and M.E. degrees from Artificial Intelligence Re- search Center, Agricultural University of Hebei, Hebei, China, and the Ph.D. degree in computer science from RMIT University, Melbourne, VIC, Australia, in 1989, 1992, and 2000, respectively.

He is currently Professor of Computer Science, Head of the Evolutionary Computation Research Group, and the Associate Dean (Research and Innovation) in the Faculty of Engineering. His current research interests include evolutionary computation, particularly genetic programming, particle swarm optimization, and learning classifier systems with application areas of image analysis, multi-objective optimization, feature selection and reduction, job shop scheduling, and transfer learning. In these areas, he has published over 700 research papers in refereed international journals and conferences.

Prof. Zhang is a Fellow of Royal Society of New Zealand, a Fellow of IEEE, an IEEE CIS Distinguished Lecturer, and have been a Panel member of the Marsden Fund (New Zealand Government Funding). He was the chair of the IEEE CIS Intelligent Systems and Applications Technical Committee, and chair for the IEEE CIS Emergent Technologies Technical Committee and the Evolutionary Computation Technical Committee, and a member of the IEEE CIS Award Committee. He is a vice-chair of the IEEE CIS Task Force on Evolutionary Feature Selection and Construction, a vice-chair of the Task Force on Evolutionary Computer Vision and Image Processing, and the founding chair of the IEEE Computational Intelligence Chapter in New Zealand. He is also a committee member of the IEEE NZ Central Section.