Large Scale Image Classification Using GPU-based Genetic Programming

Peng Zeng
zengpeng7@stu.scu.edu.cn
College of Computer Science, Sichuan University
Chengdu, China

Andrew Lensen
andrew.lensen@ecs.vuw.ac.nz
School of Engineering and Computer Science, Victoria University of Wellington
Wellington, New Zealand

Yanan Sun*
ysun@scu.edu.cn
College of Computer Science, Sichuan University
Chengdu, China

ABSTRACT

Genetic programming (GP) has been applied to image classification and achieved promising results. However, most GP-based image classification methods are only applied to small-scale image datasets because of the limits of high computation cost. Efficient acceleration technology is needed when extending GP-based image classification methods to large-scale datasets. Considering that fitness evaluation is the most time-consuming phase of the GP evolution process and is a highly parallelizable process, this paper proposes a CPU multi-processing and GPU parallel approach to perform the process, and thus effectively accelerate GP for image classification. Through various experiments, the results show that the highly parallelized approach can significantly accelerate GP-based image classification without performance degradation. The training time of GP-based image classification method is reduced from several weeks to tens of hours, enabling it to be run on large-scale image datasets.

CCS CONCEPTS

• **Computing methodologies** → **Genetic programming**; **Parallel algorithms**.

KEYWORDS

genetic programming, image classification, parallel algorithms

ACM Reference Format:
Peng Zeng, Andrew Lensen, and Yanan Sun. 2022. Large Scale Image Classification Using GPU-based Genetic Programming. In *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3520304.3528892

1 INTRODUCTION

Genetic programming (GP) [8] is one of the evolutionary computation (EC) techniques based on the Darwinian theory of evolution. Due to its flexible representation and good global search ability, GP has multiple applications in machine learning tasks including image classification [6].

*Corresponding author

GECCO '22 Companion, July 9–13, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9268-6/22/07.
https://doi.org/10.1145/3520304.3528892

Although GP has been applied to different image classification tasks and achieved promising results, one notable problem is how its computational efficiency can be improved to allow its use on large-scale datasets. Many well-known image classification benchmark datasets, such as CIFAR-10 (50000 training images) and ImageNet (1.2 million training images), have a large number of images for training and testing, while most GP-based image classification are only applied to small-scale image datasets, such as FS (1300 training images) and VDGB (247 training images). GP-based image classification is hard to scale because of high computation cost. Fitness evaluation is the major bottleneck causing the GP algorithm to be computationally expensive. According to [5], approximately 94% of the training time in GP algorithms is taken by the evaluation stage. Specifically, during each iteration, every individual in the population need to be evaluated. In the image classification tasks, the fitness of each GP individual is often evaluated on the whole training dataset. When the number of images in the training dataset is large, the evaluation cost of even a single individual is rapidly increased.

To address the issue, in this paper, we propose a new two-level parallelism approach consisting of CPU multi-processing and GPU parallel to accelerate GP-based image classification. Specifically, we use CPU multi-processing techniques to perform fitness evaluation of multiple GP trees in the population in parallel. And in the process of one GP tree's fitness evaluation, the transformation of all images in the training dataset into thousands of feature vectors will be executed simultaneously using the mechanism of GPU hyper multi-threaded parallelism. Through various experiments, the results show that the speed up brought by the proposed approach outperform other accelerating GP-based image classification approaches. The proposed approach would not reduce the scale of fitness evaluation and thus not lead to performance degradation.

2 THE PROPOSED METHOD

The overall framework of the proposed GP-based image classification is composed of several steps including population initialization, fitness evaluation, selection and offspring generation. The population consists of hundreds of GP programs. In fitness evaluation, the classification accuracy on the training dataset is calculated and used as the fitness of each program. Then GP programs are selected based on the fitness and perform the genetic operators with a certain probability to generate offspring. The fitness evaluation and population generation repeat until a pre-defined termination criterion, for example, max generation, is satisfied and then the best program is returned as the solution found by the GP algorithm.

The best GP program is then evaluated on the test dataset to verify its classification performance.

2.1 The GP Program Structure, Function Set, and Terminal Set

In order to make the programs evolved by the GP algorithm applicable for image classification tasks, we develop a flexible GP program structure. We also specifically design the function set and the corresponding terminal set to make sure each GP program can be executed independently on different images, which is needed to achieve image parallel.

2.1.1 GP Program Structure. The proposed GP program structure is based on strongly typed GP (STGP) [9] and can be divided into four layers including the input layer, the intermediate layer, the concatenation layer and the output layer. In the input layer, the original image is taken as the input of the GP tree. Some randomly generated constants are also taken as the input for the execution of some functions, such as *conv*, *maxpool*, etc. In the intermediate layer, there are multiple types of functions employed for region cropping and selection, image processing, feature extraction, and feature compression. Unlike CNNs, functions in the intermediate layer are not fixed, which means the specific GP program structure is determined through evolution. After the intermediate layer, the concatenation layer concatenates all the features from subtrees into a feature vector. At last, the output layer returns the final feature vector. Compared with the previous GP-based image classification methods, the GP program structure in the proposed method is not explicitly divided by steps or in a predefined order, e.g. region selection first, then feature extraction. Moreover, various processing steps can be reused when building the GP program. This change brings more flexibility for GP to evolve the program structure.

2.1.2 Function Set and Terminal Set. The functions designed in the proposed method is inspired by CNNs. Specifically, the *concat* function is used in the concatenation layer, by concatenating all the features from subtrees into a feature vector. Other functions are all used in the intermediate layer. Particularly, The *regions* and *regionr* function directly perform on the image and select a square or rectangular area of interest. The *conv*, *maxpool* and *ReLU* functions can be used for both raw images and processed images, and the order between them is not restricted. The gen_filter function generate filters of different size and different type to perform convolution. The *mean* and *std* function obtain the statistics like mean, standard deviation of pixel matrix. The *pixeladd*, *pixelsub*, *pixelmul* and *pixeldiv* function directly perform on images to obtain some pixel calculation results. Instead, the *numadd*, *numsub*, *nummul*, *numdiv* and *numneg* function are mainly used to calculate results from statistics. The if_else function introduce conditional statements into the GP program. The *mixadd* and *mixsub* perform addition and subtraction with weights.

The terminals includes the following. The inputs of the proposed method are raw images. The *row* and *col* terminals are used to locate a specific pixel, so the range of them is from 0 to the height or width of image. The *randomd* terminal is used to generate random numbers for using in some functions. The *windowx*, *windowy* and *windowsize* terminals are used in the *regions* and *regionr* function, where the *windowx* and *windowy* represent the coordinates of the upper left point and the *windowsize* represent the length and width of the selected area. The $filtersize$ and $filtertype$ are used to determine the convolution filters used in the *conv* functions. The $filtertype$ includes *random*, *mean*, *gauss*, *prewitt*, *sobel* and *laplace*.

2.2 Fitness Evaluation and the Parallel Approach

In the proposed method, the fitness of a GP program is evaluated using training classification accuracy. Specifically, the GP program is used to transform each image in the training dataset into feature vectors. Since feature vectors from different subtrees may not be on the same order of magnitude and in order to avoid some of the feature vectors dominating the classification results, all feature vectors are then normalized using the min-max normalization method. Since the normalized feature vectors can not directly reflect classification effect, we feed the feature vectors into a linear support vector machine (SVM) [4] to calculate the classification accuracy. To improve the generalization ability of the evolved program, 5-fold cross-validation is used for SVM, and the mean accuracy of 5 folds is obtained and employed as the fitness value of the GP program. Since fitness evaluation is the bottleneck causing GP algorithm computationally expensive, we analyze the process and find it is a highly parallelized process, which can be summarized as *individual parallel* and *image parallel*:

- *Individual parallel* means that the same fitness evaluation function needs to be performed on each individual in the whole population independently. The fitness evaluation of one individual is not influenced by other individuals, so we can easily conduct the fitness evaluation of multiple individuals in parallel.
- *Image parallel* means that thousands of images are transformed into a number of feature vectors through the same GP program in the process of one individual's fitness evaluation, which means the same functions are performed on different images. As a result, we can conduct the execution of a GP individual on multiple images in parallel.

According to the high parallelism in GP algorithm, we propose a two-level parallel approach with CPU multi-processing and GPU parallel to accelerate the process. The details of the proposed two-level parallel approach is illustrated in Fig. 1. First, the individual parallel is performed utilizing the CPU multi-processing mechanism. Specifically, after the population initialization, all GP programs in the population need to be evaluated. We launch a multi-processing pool where there are multiple subprocesses to perform the fitness evaluation. The fitness evaluation function are mapped to different GP programs. The evaluation process of different GP programs are independent, so a subprocess carries on a GP program's fitness evaluation and all subprocesses can execute in parallel. If the multi-processing pool is not full, a program's fitness evaluation would be performed immediately using a new subprocess instead of waiting for other programs' fitness evaluation to end. If sufficient resources are available, fitness evaluation of all programs can be performed simultaneously.

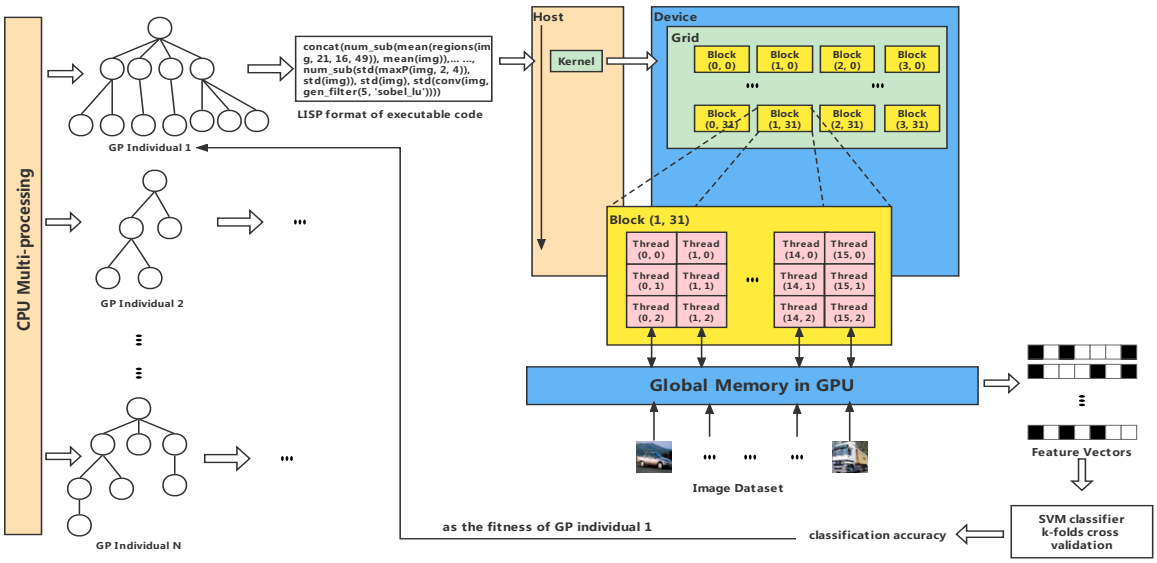

Figure 1: The proposed CPU multi-processing and GPU parallel approach.

In the fitness evaluation of a GP program, thousands of images are transformed into a number of feature vectors via the same instructions (GP program), which is in line with the SIMD intrinsic of GPU [1]. Therefore, we perform image parallel using the mechanism of GPU hyper-multithreaded parallelism. This means that one GPU thread represents the execution of a GP program on an image.

In the proposed approach, we take the transformation of an image into a feature vector through a GP program as a kernel function and the rest part of fitness evaluation function as the host function. The kernel function is executed on the GPU, while the host function is performed on the CPU. In order to conduct the kernel function (mainly GP program) on the GPU, we implement all the functions in the function set on CUDA. Considering the organization of threads, we launch a grid in GPU for each kernel function. There are multiple blocks in the grid and in each block there are hundreds of threads. Since CUDA executes instructions using warps of 32 threads in parallel, the number of threads in the same block is better set to a multiple of 32. Since a GPU thread processes on an image, the exact number of blocks per grid and threads per block is determined by the size of image dataset. After completing the organization of GPU threads, we copy the entire image dataset to the global memory of GPU. Specifically, an image is assigned to a thread, and when a thread executes, it finds the corresponding image in the memory according to three CUDA built-in structures including *blockIdx*, *blockDim* and *threadIdx*. We also allocate an empty device array on GPU, whose size is determined by the size of image dataset, to store the results obtained by each thread processing the image. When the kernel function is finished, the whole results matrix would be copied back to host, then the classification accuracy would be calculated as the fitness of the GP program.

3 EXPERIMENTS AND RESULTS

3.1 Experiment Design

3.1.1 Benchmark Datasets. The proposed method is evaluated on five popular image classification benchmark datasets of different types of tasks and various sizes including the facial expression classification dataset (FEI_1), the texture classification dataset (KTH), the digit recognition dataset (MNIST), the object classification dataset (CIFAR-10), and the real-world digit recognition dataset (SVHN). Among these datasets, FEI_1 and KTH are commonly used in existing GP-based image classification methods. MNIST, CIFAR-10 and SVHN are not widely used because of their large size.

3.1.2 Baseline Methods. In the small-scale dataset such as FEI_1 and KTH, we compare the proposed method with the latest GP-based image classification methods including EGP, COGP, and FGP [3]. Multiple existing GP libraries including DEAP [7] and gplearn [10] provide support for accelerating GP algorithms, we compare the proposed method with these libraries. Since DEAP and gplearn are only libraries instead of specific image classification methods, we implement a GP-based image classification algorithm on them.

As is mentioned in Section 1, existing GP-based image classification methods are almost impossible to apply to large-scale datasets. For instance, the training time of COGP on CIFAR-10 is more than 2,000 hours for one time run. Therefore, for large-scale datasets such as MNIST, CIFAR-10, and SVHN, we add two efficient acceleration strategies including Instance Selection-Based Surrogate-Assisted (ISS) GP and Divide-and-Conquer (DC) GP [2] for comparison. In ISS, five surrogate subsets are selected from the whole training dataset using instance selection. In DC, the training dataset is randomly split into four non-overlapping subsets preserving class ratio. For a fair comparison, the GP program structure as well as the corresponding function set and terminal set in ISS and DC are identical with the proposed method.

3.1.3 Hardware Setup and Parameter Settings. All experiments were carried out in the computer with GeForce RTX 2080Ti and the number of CPU cores is 16.To implement the proposed method, some key parameters are defined below. Firstly, we use the ramped half-and-half method to generate the population of 500 individuals. The number of generations is set to 50. Secondly, for genetic operators, the genetic operators used in our approach include elitism, crossover, and mutation, and their probabilities are 1%, 70%, 29%, respectively. The mutation operators used in the proposed method includes point mutation and subtree mutation. Lastly, for selection, we use the tournament scheme to perform selection, and the tournament size is set to 5.

3.2 Results and Discussion

Table 1: Training Time Cost (hours) and Classification Accuracy (%) of the Proposed Method and Baseline Methods on Five Benchmark Datasets

Dataset	Method	Training Time (h)	Classification Accuracy (%)
FEI_1	EGP	17.86	96.0
	COGP	1.67	94.0
	FGP	1.99	96.0
	Gplearn	3.23	94.0
	DEAP	0.59	94.0
	Our Method	0.18	94.0
KTH	EGP	117.02	78.02
	COGP	27.38	78.52
	FGP	21.29	96.88
	Gplearn	21.12	76.36
	DEAP	6.72	77.27
	Our Method	3.32	78.48
MNIST	DEAP	244.26	83.94
	ISS	58.71	82.38
	DC	48.53	81.12
	Our Method	21.06	87.08
CIFAR-10	DEAP	303.25	39.47
	ISS	110.70	41.64
	DC	74.62	39.96
	Our Method	32.74	40.23
SVHN	DEAP	434.81	60.38
	ISS	85.35	58.20
	DC	82.96	59.12
	Our Method	35.46	61.93

The experimental results of the proposed method and the baseline methods are shown in Table 1. In terms of training time, which is the main concern of the paper, Table 1 shows that the proposed method uses significantly shorter training time than baseline methods on all the five benchmark datasets. Specifically, in small-scale datasets like KTH, existing GP-based image classification methods typically require training time more than twenty hours, while the proposed method can finish training in several hours. In addition, the acceleration effect of the proposed method becomes more effective when the dataset size gets larger. For instance, the training time of the proposed method on KTH is half of that in DEAP. While in large-scale datasets such as MNIST and CIFAR-10, the training time of the proposed method is only about one tenth of that of DEAP. This is because as the size of datasets increases, the proposed method launches more threads in GPU to deal with the execution of GP programs, which devotes more space and does not result in a dramatic increase in training time. Thus the massive data parallelism in GPU can be fully utilized when training on large-scale datasets. In conclusion, the proposed method can effectively accelerate the training of GP algorithm and is applicable to large-scale image datasets currently.

Table 1 shows that the proposed method can achieve promising results on the benchmark datasets in terms of classification accuracy. Firstly, comparing with other GP-based image classification methods, the proposed method achieves similar classification results to EGP and COGP. However, the proposed method takes significantly less training time than EGP and COGP with similar accuracy. FGP achieves much better results on KTH dataset. This is because FGP use more expert-designed feature extraction methods in the function set. Besides, compared with other acceleration strategies, the proposed method achieves comparable or better classification results than two effective acceleration methods including ISS and DC. According to the experimental results, we can conclude that the proposed method can significantly speed up GP-based image classification without degrading the performance.

4 CONCLUSION

The paper develops an effective approach to accelerate the evolution of GP for image classification with the aim of applying GP-based image classification methods to today's increasingly large datasets. We propose a new two-level parallelism approach consisting of CPU multi-processing and GPU parallel to accelerate GP-based image classification methods. Then we conduct a series of experiments to demonstrate that the proposed approach can effectively reduce GP evolution time without performance degradation, and successfully apply GP-based image classification to large-scale image datasets.

REFERENCES

[1] M. G. Arenas, Gustavo Romero, Antonio Mora, Pedro Castillo, and Juan Merelo Guervós. 2012. GPU Parallel Computation in Bioinspired Algorithms: A Review. *Studies in Computational Intelligence* 422 (2012), 113–134. https://doi.org/10.1007/978-3-642-30154-4_6

[2] Ying Bi, Bing Xue, and Mengjie Zhang. 2021. A Divide-and-Conquer Genetic Programming Algorithm with Ensembles for Image Classification. *IEEE Transactions on Evolutionary Computation* (2021), 1. https://doi.org/10.1109/TEVC.2021.3082112

[3] Ying Bi, Bing Xue, and Mengjie Zhang. 2021. *Genetic Programming for Image Classification.* Vol. 24. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-030-65927-1

[4] Richard G. Brereton and Gavin R. Lloyd. 2010. Support vector machines for classification and regression. *The Analyst* 135, 2 (2010), 230–267. https://doi.org/10.1039/B918972F

[5] Alberto Cano, Amelia Zafra, and Sebastian Ventura. 2012. Speeding up the evaluation phase of GP classification algorithms on GPUs. *Soft Computing* 16 (2012), 187–202. https://doi.org/10.1007/s00500-011-0713-4

[6] Pedro A. Castillo and Juan Luis Jiménez Laredo (Eds.). 2021. *Applications of Evolutionary Computation.* Springer International Publishing, Cham. https://doi.org/10.1007/978-3-030-72699-7

[7] Félix-Antoine Fortin, François-Michel de Rainville, M. A. Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research, Machine Learning Open Source Software* 13 (2012), 2171–2175.

[8] John R. Koza. 1992. *Genetic programming: on the programming of computers by means of natural selection.* MIT press.

[9] David J. Montana. 1995. Strongly Typed Genetic Programming. *Evolutionary Computation* 3, 2 (1995), 199–230. https://doi.org/10.1162/evco.1995.3.2.199

[10] Trevor Stephens. 2015. gplearn: Genetic programming in python, with a scikit-learn inspired api.