Towards a Time-efficient CNN Design with Cartesian Genetic Programming

https://github.com/terry00123/CS454_Project

Team 7

20160458 Sanghyun Lee
School of Computing, KAIST
lsh980504@kaist.ac.kr

20160534 Taeckyung Lee
School of Computing, KAIST
terry00123@kaist.ac.kr

20170072 Kihong Kim
School of Computing, KAIST
hongkim9815@kaist.ac.kr

20170305 Hyoungjo Bhang
School of Computing, KAIST
bhanghj3094@kaist.ac.kr

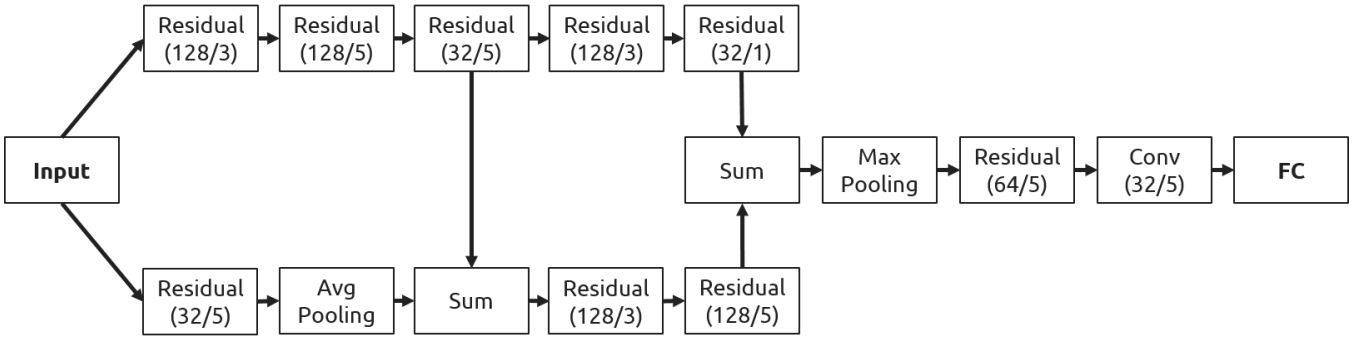

Figure 1. CNN architecture generated by our Enlarged Parent method. This architecture shows the best accuracy (92.44%) among every CGP results in our report.

1 Introduction

After years of the rising importance of deep learning, it has now melted into all areas of computer science; computer vision, autonomous driving, data generations, word/speech processing, and so on. Adaptive switching to different usages of deep learning models has required various implementations of structures. Not only according to their types of learning like CNN, RNN, and GAN, each type also requires diverse modeling to be effective on individual objectives.

These emerging attempts and contributions in uniting deep learning and traditional scientific methods have shown unprecedented possibilities, but at the same time, still very demanding for expert knowledge on modeling neural network architectures. To relieve this burden, recent researches on simplifying the creation of deep learning models have emerged. Among them is Genetic Programming (GP), which is an area of artificial intelligence for evolutionary computation. With the recent work of applying GP to automatic designing of deep learning models for image classification, we present more thoughtful analysis and experiments on genetic architecture.

In this piece of work, the main focus is the fact that this approach requires extremely heavy computations of GP combined with deep learning model training. Therefore, with limited number of evaluations, we tried to demonstrate the methodology for building the model with the best accuracy. The details are as follows: 1) Initialisation based on Statistical Analysis, 2) Strong Neutral Mutation, and 3) Enlarging Parent Population Size.

This paper is organized as follows: In *Section 2*, background of Cartesian Genetic Programming and related work is presented. Then, the detail method for implementation we have tried is in *Section 3*. The results and insights are documented and summarized in *Section 4*, and conclusions and discussions are in *Section 5*.

2 Background

2.1 CGP

For representing CNN architecture and its connectivity more easily, we utilized Cartesian Genetic Programming (CGP) which is a graph based genetic programming encoding schemes. Fig 2 shows how the algorithm is constructed with functional node. Each node represents a function, and these nodes are deployed in a grid which consists of row(Nr) and column(Nc).

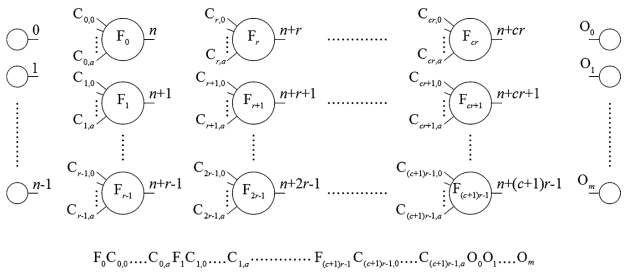

Figure 2. Mimetic diagram of Cartesian Genetic Programming [1].

Each node has *a* inputs that depend on the function in the node, and levels-back parameter (*l*) that determines how far this input can reach behind. For example, *c*-th column's nodes can be connected to nodes from (*c*-*l*) to (*c*-*1*)-th column's nodes.

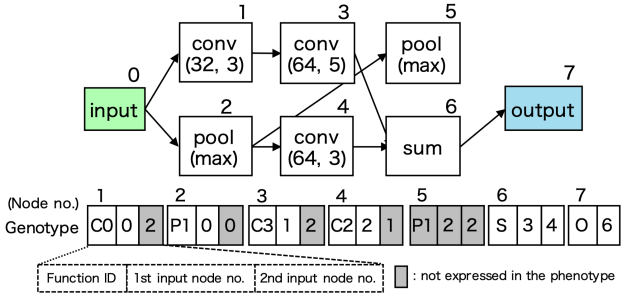

Figure 3. Example of CGP model [4]. Node 1 ~ 7 are represented as genotype with function ID, and input numbers. The order of nodes are from left column to right, and from upper nodes to down. Parts of nodes expressed in silver are inactive numbers that cannot affect the output values.

2.2 A Genetic Programming Approach to Designing Convolutional Neural Network Architectures

A paper "A Genetic Programming Approach to Designing Convolutional Neural Network Architectures [4]" is the first attempt to construct a CNN architecture automatically using genetic programming approach. The paper generated the architectures to outperform state-of-art CNN models like VGGNet and ResNet.

To generate the CNN architecture, the paper utilized CGP with six types of blocks: ConvBlock, ResBlock, max pooling, average pooling, concatenation, and summation.

- **ConvBlock and ResBlock**
  ConvBlock consists of convolutional computation, batch normalization, and ReLU. For a kernel size of 3 and 5, input (M, N, C) is mapped to output (M, N, C'). ResBlock has extra summation layer on the ConvBlock as the Fig 4.

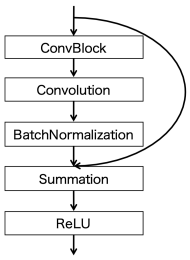

Figure 4. Architecture of ResBlock [4].

- **Max and Average pooling**
  Pooling layer with 2 x 2 receptive field and stride size 2 results in image field size as 1/4 of the original.

- **Concatenation and Summation**
  Two types of node is the main differences of CGP-based architecture and state-of-the-art architectures. Since CGP does not consider the size of each layer to be connected, concatenation and summation cannot be done easily.
  Concatenation node concatenates two inputs of size $M_1 \times N_1 \times C_1$ and $M_2 \times N_2 \times C_2$ to generate output feature map $min(M_1, M_2) \times min(N_1, N_2) \times (C_1 + C_2)$. Larger input feature map is down-sampled to fit to the smaller feature map.
  Summation node performs element-wise summation two inputs of size $M_1 \times N_1 \times C_1$ and $M_2 \times N_2 \times C_2$ to generate output feature map $min(M_1, M_2) \times min(N_1, N_2) \times max(C_1, C_2)$. Smaller input feature map is zero-padded to sum with the larger feature map.

They also referred about genetic algorithm operators such as mutation and selection. It implements default algorithm of point mutation, which replaces only one node. They applied a **forced** mutation to generate λ siblings; while they applied a **neutral** mutation to the parents that survived. Among *Nr* by *Nc* nodes, there exist inactive nodes that do not affect the phenotype, while active nodes that actually participate in the phenotype architecture. Neutral mutation is applied at inactive nodes to preserve the phenotype; while forced mutation is applied to active nodes to change the phenotype.

2.3 Crossover in CGP

Crossover operators are widely used in GP area, but there is no appropriate crossover operator for CGP [2]. Also, crossover for CNN may not be applicable since CNN layers are connected to each other to generate own characteristics. In other words, crossover does not have significant reason to use. Therefore, the original paper [4] does not utilize crossover in CGP-based CNN architecture generation, and we decided to follow the paper.

3 Methods

To achieve the highest accuracy with limited number of evaluations, we tried following approaches: *1) Initialisation based on Statistical Analysis*, *2) Strong Neutral Mutation*, and *3) Enlarging Parent Population Size*.

3.1 Initialisation based on Statistical Analysis

Initialisation is a critical factor of genetic programming, because it is closely related to the convergence speed. However, the original paper used random initialisation without any reasonable explanations. Since initialisation can be replaced by sampling the architectures from CGP executions, we conducted an experiment to calculate the maximum expectation of fitness with fixed number of fitness evaluations.

Dataset Preparation Inspecting trial executions on original CGP method, the accuracy of classification model converged after 30 generations. Therefore, we set our maximum fitness evaluations as 30, and repeated 48 executions of 30 generations to collect the fitness of each generations.

Main Goal Our two main objectives are 1) find the initialisation method of best expected fitness of limited evaluations, and 2) interpret the result to understand the nature of CGP-based CNN model generation. Elaborating on the second objective, each initial generation of genotype can be stuck in local optima. Therefore, the right balance between trying several mutations and finding new random initialisation is important.

Expectation of Max Fitness We assumed the distribution to normal distribution of the mean and standard deviation from the sample of each generation. Let X as random variable of fitness, and $X_{max,n}$ as max fitness of n samples. $P(X_{max,n} \leq x) = F(x)^n$, when $F(x)$ is CDF of variable X. Therefore, $E(X_{max,n}) = \int_{-\infty}^{\infty} x[F(x)^n]'dx$

3.2 Strong Neutral Mutation

Strong neutral mutation (SNM) is our new approach on CGP to intend the fast escape from the local optima. We observed that parents does not change for 5~20 steps. Therefore, we intended to give strong evolutionary pressure to the parents by repeating the neutral mutation. The steps for our algorithm are:

- Initialize *best_is_parent* count.
- For each generations: If the parent survives, increment *best_is_parent* count. Else, reset the count.
- When *best_is_parent* reaches 5, apply 10 neutral mutations and reset its count.

3.3 Enlarging Parent Population Size

For previous research, only one parent exists, so the best individual survives each generation. However, considering the diversity of the population and possibility of better siblings can be generated from slightly worse individuals, we decided to enlarge the parent population size as well as the entire population size.

For example, if the original CGP uses one parent with one siblings at each generation, we can enlarge by the factor of 2 so there exists two parents with one siblings each to generate total two siblings.

4 Evaluation

For the evaluation, we used two RTX 2080 machines with Intel i7-6700K CPU and 15GB RAM. Our program was operated on Ubuntu 16.04 LTS, CUDA 10.0 and PyTorch 0.4.1 [3]. Details can be found at our GitHub repository[1]. Total experiment procedure had taken almost 30 days.

As genetic operators, we used $Nc = 30, Nr = 5, l = 10$, and $\lambda = 1$ (1 parent with 1 sibling).

4.1 Initialisation based on Statistical Analysis

We evaluated 45 samples of 30 generations. Total execution time of 45 samples was 1,548,634 seconds. The average execution time of 1 evaluation was 1,147 seconds. Table 1 shows the mean and standard deviation of each generation.

gen	Mean	Std
1	0.8460	0.0518
2	0.8604	0.0457
3	0.8676	0.0425
4	0.8725	0.0345
5	0.8765	0.0312
10	0.8872	0.0234
20	0.8982	0.0122
30	0.9036	0.0103

Table 1. Mean and standard deviation of each generation

We calculated the expectation value with assumption that takes 30 evaluation. For example, we calculated the expectation value of max fitness with 30 samples in generation 1, because it need 1 evaluation to reach generation 1. Like this, there would be 3 samples in 10 generation, 2 samples in 15 generation, 1 samples in 30 generation. Fig 5 shows graph of the max fitness in 30 generations.

However, expectation value is much different from the real statistics. The expectation value of generation 1 is 0.9517, but there was only one sample which is over 0.9 and its accuracy was 0.9164. The *p-value* of 0.9164 is 0.0171 in generation 1 distribution, so we need to revise the samples.

There were some samples which accuracy of first generation is much lower than other samples, because accuracy of CNN is very unpredictable process. This samples did not

[1] https://github.com/terry00123/CS454_Project

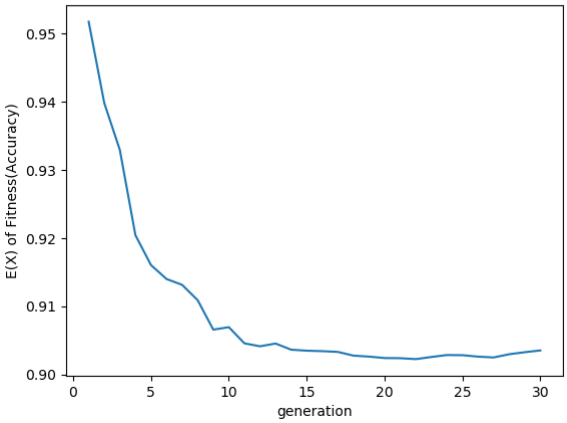

Figure 5. Expectation value of maximum fitness in 30 evaluation

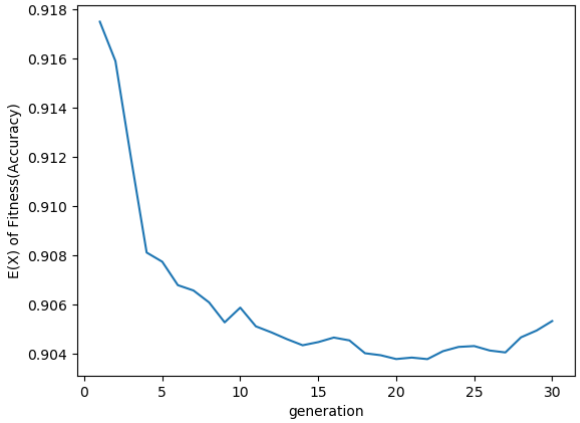

Figure 6. Expectation value of maximum fitness in 30 evaluation

affect to real max value much, because it just cost 1 evaluation in large amount of evaluation. However, they made standard deviation much larger, so the statistical expectation value become much higher. For this reason, we remove the samples which accuracy of first generation is lower than 0.8.

gen	Mean	Std
1	0.8640	0.0262
2	0.8733	0.0245
3	0.8787	0.0216
4	0.8816	0.0191
5	0.8844	0.0184
10	0.8911	0.0174
20	0.8998	0.0119
30	0.9053	0.0094

Table 2. Revised mean and standard deviation of each generation

Table 2 shows mean and standard deviation after revision process, and Fig 6 shows graph of expectation value. The Fig 6 shows similar trend with Fig 5, but the expectation value of early generation is much lower even we removed the low accuracy samples. The expectation value of first generation is 0.9175 and the *p-value* of 0.9164 is 0.4224 in this distribution, so we suggest the revision of statistics is reasonable.

Surprisingly, we get the best value in the first generation. The accuracy is decreased rapidly during the first 5 generations, and it is increased slowly after generation 20. We think the reason of result is that CGP of CNN has a strong randomness, so it is hard to find better solution in a few local search. Therefore, the pure random search is more efficient in initialisation process. However, the growth of accuracy

gen	E(X)
1	0.9175
2	0.9159
3	0.9120
4	0.9081
5	0.9077
10	0.9059
20	0.9038
30	0.9053

Table 3. Expectation value of max accuracy in each generation

after generation 20 can demonstrate that the CGP effect after enough local search evaluation.

4.2 Strong Neutral Mutation

We compare experiment results on the original CGP and our SNM-based approach. To compare our result with the original one, the initial population should be fixed. Therefore, we fixed the seed with the value 20160458. Genetic programming is performed over 250 generations.

SNM-based method consumed 1,214s per generation for average, and the original method consumed 1,520s per generation for average.

As shown in Fig 7, even though we fixed the seed so the initial CNN architectures were identical, the initial accuracy showed different due to the randomness in Pytorch. Also, the result indicates that our SNM-based approach does not show the significant improvement over the original version.

4.3 Enlarging Parent Population Size

We execute the original CGP twice; and execute our model with parent population size enlarged to 2 three times.

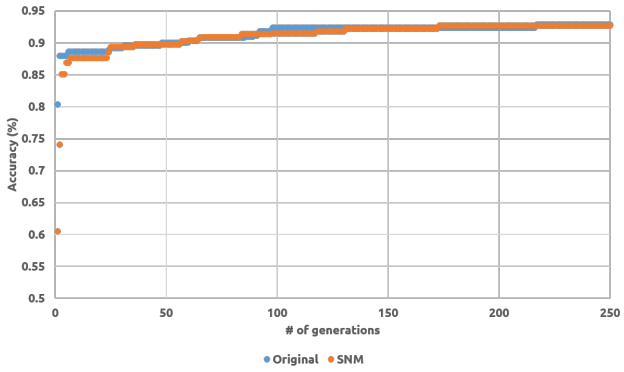

Figure 7. Comparison of SNM and the original CGP with seed 20160534 over 250 generations.

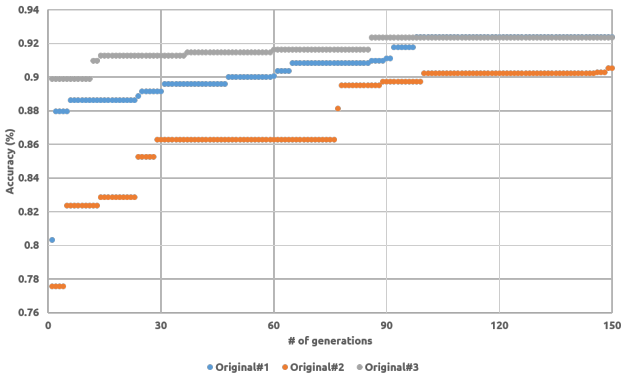

Figure 8. Three different results of the original CGP over 150 generations. Third execution was recorded up to 120 generations due to the time burden.

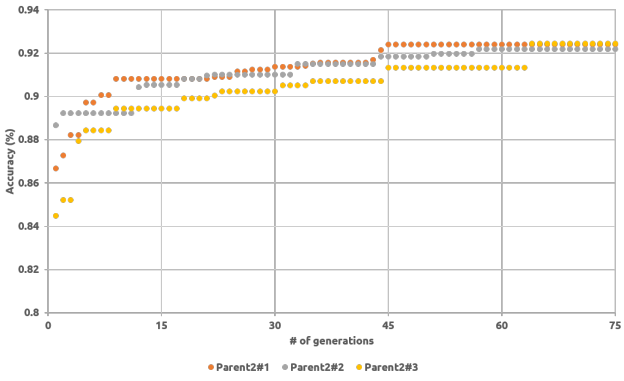

Figure 9. Three different results of parent population size 2 over 75 generations.

As shown in Fig 9, our parent population size enlarged version shows smoother curve with faster conversion comparing to the original execution in Fig 8. However, maximum

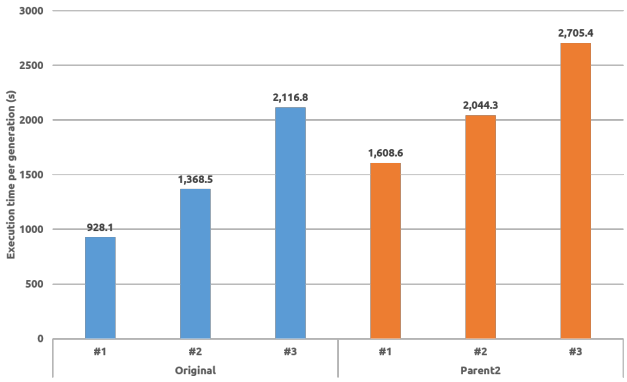

Figure 10. Execution time per generation.

accuracy is similar; original version has 0.9236 for 150 generations, while parent enlarged version has 0.9244 for 75 generations (which performs 150 fitness evaluations).

Execution time of both methods in Fig 10 were various since it is dependent on the CNN structure. Interestingly, our method showed execution time under twice of the original version. Further researches about relations between execution time and CGP structure would be interesting.

As the result, we could generate the model with highest accuracy 92.44% by our Enlarged Parent method as Fig 1. The CNN architecture contains a lot of residual blocks (Fig 4) but only a single convolution block; which matches the fact that state-of-the-art CNN models contain lots of residual layers rather than pure convolution layers.

5 Conclusion And Discussion

We tried three methods to generate the maximum result within limited time budget or fitness evaluations. We analyzed the expectation of maximum fitness over multiple executions, and found that the random initialisation is the best method on fixed number of fitness evaluations. Also, we found the weakness of neutral mutation and tried Strong Neutral Mutation, but failed to show improvements. We believe stronger mutations should be applied to generate variant phenotypes on the siblings. Finally, we enlarged the parent population size to result smoother and faster convergence comparing to the original method.

CNN is a future promising method that is widely researched and used in both academy and industry. We believe CGP-based CNN design method highly benefits on hand-designing CNN structures. We hope future researches would overcome the issues toward an automated CNN model generation.

References

[1] Julian Francis Miller. 2019. Cartesian genetic programming: its status and future. *Genetic Programming and Evolvable Machines* (2019), 1–40.

[2] Julian F Miller and Stephen L Smith. 2006. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation* 10, 2 (2006), 167–174.

[3] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[4] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. 2017. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 497–504.