AN IMAGE RECOGNITION PROGRAM SYNTHESIS SYSTEM

KRONGKRAN PHUETPHAN
CHAOWAT PETCHDUM
PAVASORN NOIPHAN

Bachelor of Engineering Program in Software Engineering
International College
King Mongkut's Institute of Technology Ladkrabang
2012

Thesis – Academic Year 2012

B.Eng. in Software Engineering

International College

King Mongkut's Institute of Technology Ladkrabang

Title: AN IMAGE RECOGNITION PROGRAM SYNTHESIS SYSTEM

Authors:

1. Ms. Krongkran Phuetphan Student ID : 52090002
2. Mr. Chaowat Petchdum Student ID : 52090008
3. Mr. Pavasorn Noiphan Student ID : 52090016

Approved for submission

.....

(Dr. Ukrit Watchareeruetai)

Advisor

Date ..17/9/2013..

INTERNATIONAL COLLEGE KMITL

An Image Recognition Program Synthesis System

Ms. Krongkran Phuetphan 52090002
Mr. Chaowat Petchdum 52090008
Mr. Pavasorn Noiphan 52090016

Dr. Ukrit Watchareeruetai Advisor
Academic Year 2012

ABSTRACT

Designing a feature extraction for solving image recognition problem is a waste of time and cost. Feature extraction represents the interesting points which are found and compared with other interesting points (features) in the image. Moreover, in order to design feature extraction program, it usually requires a specialist with specific knowledge related to given problem. Most of human-designed feature extraction programs will be domain-specific, which means that it needs some specific knowledge to deal with the problem. Consequently, domain specific program is difficult to reuse and re-design for solving other problems. In this project, we changed from design problem into optimal search problem using machine learning, which give a system's ability to learn data by itself, and evolutionary techniques, which are techniques that been used to improve feature extraction programs in the system.

This project involves a Genetic Programming (GP) approach for constructing feature extraction programs. Linear genetic programming (LGP) is a kind of GP where each feature extraction program is a sequential program. LGP is adopted to search for optimal feature extraction program. Such approach can automatically construct feature extraction programs without domain-specific knowledge by giving information to the system to learn and find solutions by itself. In particular, the user only needs to input some information to the system for program evaluation.

The main purpose of this system is to find optimal feature extraction program for solving a given problem to be used in further works. In summary, in some experiments, for example, detecting a flower from its background. This system can find feature extraction program that is able to generate 98% precision close from such experiment. Therefore, this system can be adopted to use in further work in other fields of study or even in real work.

Acknowledgement

We would like to thank and offer our sincere gratitude to our advisor, Dr. Ukrit Watchareeruetai. We greatly appreciated the guidance, support, and encouragement that were offered throughout the length of the study. We also would like to give special thanks to all thesis committee members, and we are grateful for their helpful comments.

Thank you all staffs at the Faculty of International College for their help, and also thank you our classmates for the friendship and encouragement given. We would like to give special thanks to our family for their endless love, understanding and support.

Finally, we would like to express our gratitude to everyone else not named in this thesis, who supported us until the completion of this thesis.

Krongkran Phuetphan, Chaowat Petchdum and Pavasorn Noiphan

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Designing a feature extraction for solving image recognition problem a wasted of time and cost. Moreover, in order to design feature extraction program, it usually requires specialist with specific knowledge related to the given problem.

Usually, the techniques and tools that experts use to design the program mainly depend on their knowledge and experience. However, in some cases, the designer may overlook and ignore important or useful feature, due to the time constraints. Moreover, an image recognition program designed by expert is usually domain-specific. Therefore, if there are some changes in the problem, it can cause the designer to re-design the program. Also, it is very difficult to reuse the designed program in other tasks.

Due to reasons mentioned above, it would be more flexible if we change from a design problem into an optimal search problem instead. In this project, we choose to create a system that can generate the flowchart of image recognition program, by using machine learning and evolutionary computation techniques.

In this system, a user inputs only needed information, which will be used to evaluate the programs' performance. Then the system will use some functions from an image library to co-operate with genetic programming to find and improve the programs' performance.

The main purpose of this system is to find the optimal feature extraction program for solving a given problem. In summary, this system should be able to find feature extraction program that is good enough for solving the given problem. Therefore, this system can also be adopted to use in further work in other fields of study or even in real work.

1.2 Objectives

This project aims to develop a synthesis system that can automatically construct an image recognition program for a given problem without domain specific knowledge by using the ideas from machine learning and evolutionary computation.

The system needs to take some inputs from the user. After processed, the system will generate the flow-chart of generated image recognition programs that is appropriate for solving the given task for the user.

1.3 Scope of Work

To create a synthesis system that can automatically construct an image recognition program for a given problem without domain specific knowledge and develop a GUI that is able to show process while the system is processing.

1.4 Procedure

The following lists summarize the tasks which have been carried out (Either completely or partially) since September 2013 to May, 2013 and by whom.

Table 1.1 Tasks work

ID	Task name	Duration	Resource Names
1.	**Requirement gathering**	**15 days**	
	- Project background	1 day	Krongkran,Chaowat,Pavasorn
	- Statement of Problem	2 days	Pavasorn
	- Objectives	4 days	Pavasorn
	- Scope of work	2 days	Krongkran,Chaowat
	- Mock up and Scenario	6 days	Krongkran,Chaowat
2.	**Analysis**	**10 days**	
	- Use case	3 days	Krongkran
	- Activity Diagram	5 days	Chaowat
	- Package Diagram	3 days	Pavasorn
3.	**Design**	**10 days**	
	- Class Diagram	10 days	Pavasorn
4.	**Implementation**	32 days	
	- Genetic Programming	25 days	
	- Controller	25 days	Pavasorn
	- Representation	16 days	Pavasorn
	- Classifier	15 days	
	- Decode	7 days	Krongkran
	- Evaluator	15 days	Krongkran
	- User Interface	15 days	Chaowat
	- Program Statistic	7 days	Chaowat
5.	**Verification**	**14 days**	
	- Unit Testing	8 days	Krongkran,Chaowat,Pavasorn
	- Integration Testing	14 days	Krongkran,Chaowat,Pavasorn
6.	**Documentation and Presentation**	**61 days**	
	- Progress Examination	9 days	Krongkran,Chaowat,Pavasorn
	- Purpose Presentation	9 days	Krongkran,Chaowat,Pavasorn
	- Final Examination	48 days	
	- Draft Report	31 days	Krongkran,Chaowat,Pavasorn
	- Final Report	10 days	Krongkran,Chaowat,Pavasorn
	- Final Presentation	7 days	Krongkran,Chaowat,Pavasorn

1.5 Review of related works

Many different automatic systems for construction of image recognition program have been proposed. Evolutionary algorithms (EA) techniques that have been adopted include, genetic programming (GP), tree-based GP (TGP), graph-based GP (GGP), or linear GP (LGP).

Similar project ideas, which also use LGP as a tool to create an image recognition program using different EA, are present in other research.

Table 1.1 State of the art

	EA	Primitive operations
Robert & Claridge [3]	TGP	Arithmetic & mathematical & branch
Ando & Nagano [4]	TGP	Image processing
Shirakawa & Nagano [5]	GGP	Image processing
Krawiec & Bhanu[6]	LGP	Image processing
This project	LGP	Image processing

Instead of tree structures, Shirakawa and Nagao adopted a genetic image network (GIN)[5]. They demonstrate that even though the evolved image transformation is compact, it can perform complex image processing task. They also studied multiple-output GIN in this work, but its results were unimpressive.

Ando and Nagao used special hardware[4]. such as a genera-purpose graphics processing unit (GPGPU), which was designed for computer graphics (CG), to accelerate the construction speed. GPGPU is the inverse process of image processing, where it can be adopted to compute image-processing task efficiently.

Krawiec and Bhanu have proposed evolutionary and co-evolutionary systems based on LGP to construct feature extraction programs. POs (primitive operations) is used for basic image processing [6]. To perform object recognition, a classifier has been adopted.

1.6 Structure of the thesis

The rest of this thesis is organized as follows:

Chapter 2 describes the background knowledge and explains the theories related to GP and image processing.

Chapter 3 explains the functional and non-functional requirements of the system, and presents the use case and activity diagrams.

In Chapter 4, describes the overall architecture, including class diagrams and detail.

In Chapter 5, explains about the development process about GP and tools (OpenCV and Qt) including the algorithm used to show graph and flow chart.

Chapter 6 shows screenshots and describes the result of the experiment. And describes the effectiveness and the deficiencies of the software and evaluate the experimentation result.

Chapter 7 summarizes what had been done and what we had learned from this thesis. It also explains the problems occurred during the project and how we solved them.

Chapter 2
Background Knowledge

This chapter describes the background knowledge and explains the theories related to genetic programming and image processing

2.1 Genetic Programming

Genetic programming (GP) is a methodology to find computer programs that are able to perform a problem task. GP is a variation of genetic algorithm (GA) where each solution is a computer program.

It is a technique to find solutions based on a biological theory of natural selection, which means that only the optimal living organisms can survive. Some mechanisms in biological theory can also improve the living organisms' ability, such as adaptation to change of the environment. This can lead them to create or reproduce more successfully within their environment.

Additionally, GP is a computer simulation for solving optimization problem by representing a solution as a chromosome. A change or improvement will be processed through a chromosome set (called individual) using various genetic operators, which are mechanisms to reproduce an individual. These mechanisms consist of reproduction, mutation and crossover. Therefore, comparing biological theory to GP, a living organism is an individual in a system and an environment of living organisms is a problem task.

In GP, evolutionary process is required to find the optimal solution. This evolutionary process starts with a randomly created population that reproduce in each generation. In particular, a system will randomly select individuals to be modified using genetic operator. Selected individuals in the current population will be selected by tournament selection technique, which will randomly select some individuals from current population and choose the best one from selected individuals. After the system gets two selected individuals, it will be added to the next generation after modification. Therefore, this process can lead to new generation which might give higher performance that is able to solve the given problem. The evolutionary process will finish when it reached the termination condition or the solution to the given problem has been found.

2.2 Linear Genetic Programming

Linear genetic programming (LGP) is one kind of GP where each program is a sequential program. It consists of population, individual and instruction set. Moreover, it also needs parameters to perform a process, the parameters including population size, generation number and register size.

In LGP, each individual in a population will keep a sequential of functions that a system will perform. It will first initialize a register to keep an output of each has been functioned that been functioned, which makes it able to perform parallel tasks.

An instruction set is a function that will be processed in the system. It consists of operation code, first input for an operation code, second input for an operation code and output for an operation code, where every parameter will be initialized in number to refer to an index of register while the system process. Individual will keep instruction sets inside and process each instruction set sequentially. In particular, each individual is a program. Population contains individuals. It will keep some amount of individual inside depend on the size of population.

In addition, population size is needed to determine a size of population that LGP will perform in the system, and number of generation is also needed in order to determine a termination condition of LGP system.

2.3 Image Recognition

Image recognition is the process of identifying or detecting objects and features in an image in order to distinguish different images apart from each other. Image recognition may be obtained from the measuring width and length.

There are many applications of image recognition, such as detection of liver disease, fingerprint, face recognition and license plate. The steps of image recognition (Fig. 2.1) is shown in the thesis as follow:

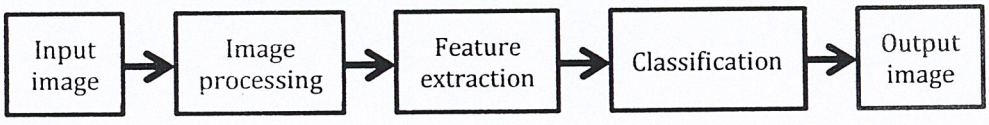


Figure 2.1 Step of Image Recognition

2.3.1 Image processing

Image processing is the improvement or modification of image data in order to improve quality and make it more appropriate for the human eye. Because an input image may contain noises, the image requires quality improvement before it can be used for processing with computer vision. This would increase processing efficiency.

The following techniques are frequently used to improve image quality:

- Image filtering: Improve images containing noises, such as small dots, which can be eliminated using image filtering.
- Brightness adjustment: Because the input image may contain too high or too low brightness, which could make the image unclear, adjusting brightness intensity in specific area within the image is required.
- Contrast adjustment: This technique is used to increase clarity and sharpness of the borders and lines within the image.
- Edge Enhancement: Adjusting borders on the image.

2.3.2 Feature extraction

Feature extraction represents the interesting points which are found and compared with other interesting points (features) in the image. Feature extraction is commonly used in image, including:

- Pixel-level features: Features calculated at each pixel, e.g. color, location.
- Local features: Features calculated the results of division of the image by size of feature.
- Global features: Features calculated over the entire image or just regular sub-area of an image.

For example, the application of colors to distinguish objects may be insufficient because colors may have problems from the variation of brightness, image perspective, or an object may be concealed by others, may cause object extraction using color to be misleading.

Those features, which are likely to assist in discrimination, are selected and used in the classification task. Feature extraction is the most critical because the particular features made available for discrimination directly influence the efficacy of the classification task. The end result of the extraction task is a set of features.

2.3.3 Classification

The goal of classification process is to categorize all pixels in a digital image or theme. This makes the computer learn the format of the data. Image classification is perhaps the most important part of digital image analysis. Classification can be divided into two categories: supervised classification and unsupervised classification.

Bayesian classification, which is a kind of supervised classification, is used in the thesis. Bayesian classification is very effective in classifying information by learning a problem and creating the conditions for classification using conditional probability

In our thesis, step of classification (Fig. 2.2), we classify each pixel and decision about which of the Training Image it resembles most.

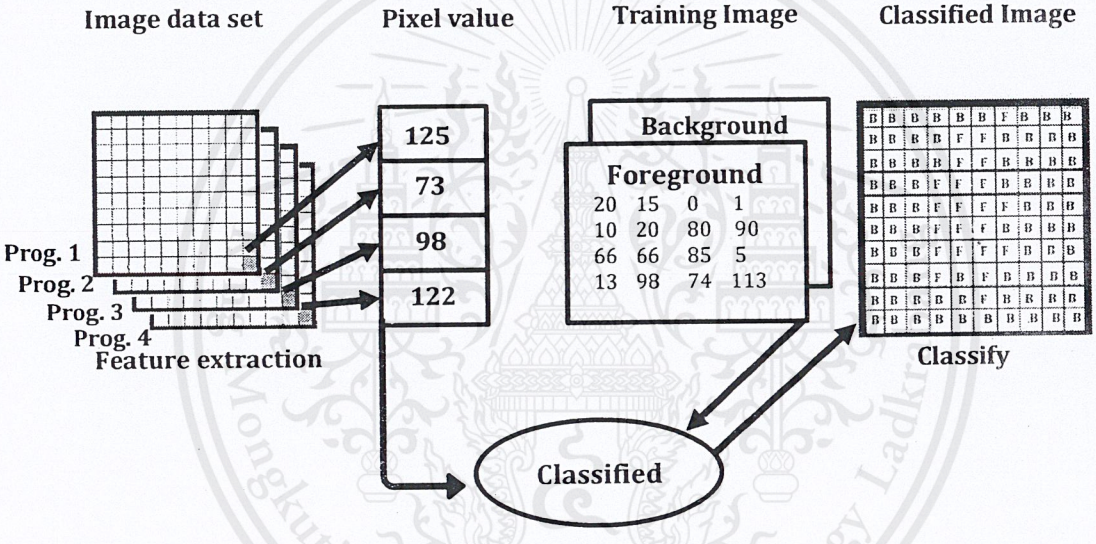


Figure 2.2 Step in Classification

Table 2.1. shows primitive operations used in the thesis and presents the type of input in each operations. Primitive operatiors can divided into four types (Table2.2).

Table2.1 List of primitive operations

One-input operations	Two-input operations
Image → Image	**Image + Image → Image**
HighPass filter	Image addition
HighPass laplacin filter	Image subtraction
Sobel operation	Image multiplication
Scharr operation	Image division
Histogram equalization	**Image + Value → Image**
Canny edge detection	Lowpass filter
Adaptive thresholding	Lowpass gaussian filter
Image square root	Median filter
Image absolute	Thresholding
Image scaling	Morphological dilation
Image negative	Morphological erosion
Convert to HSV	Morphological opening
Convert to XYZ	Morphological closing
Convert to YCrCb	Insert Channel
Convert to HLS	Extract Channel
Convert to Lab	Local histogram
Image → Value	Local variance
Global mean	Local skewness
Global standard deviation	Local kurtosis
Global variance	Local maximum
Global skewness	Local minimum
Global kurtosis	Local mode
Global maximum	Local range
Global minimum	Local entropy
Global median	
Global mode	
Global range	
Global entropy	

Table2.2 List of type primitive operatiors

Type	Input 1	Input 2	Output
1.	image	-	image
2.	image	-	value
3.	image	image	image
4.	image	value	image

2.4 Image Recognition Program Synthesis System

This system aims to find feature extraction programs that can solve the given task. In order to do so, the system needs some input for user such as images for machine learning or parameters to determine termination conditions for the system.

The system will take a pairs of images, a problem image and ground truth for the system to learn. Once the system learned, it will be able to solve the problem according to what it has learned.

In the learning process, the system will first initialize a population and evaluate population that had been generated. In addition, four elements of each instruction set will be initialized in real-value, the first element represents an operation code and the rest represent an index of register that it will access to. The system will select individual in the population one by one and process each instruction set in individual sequentially by using image function in image library depending on the information in an instruction set. In addition, it will also get an input and save an output of each instruction set according to its information. After evaluated, the system will select individual in the current population using tournament selection technique to do an evolutionary process.

This evolutionary process starts with a randomly created population and will be repeatedly re-create in each generation. In particular, a system will randomly select individuals to be modified using genetic operator. After the system gets two selected individuals, it will be added to the next generation after there are modified until a new generation reached its size. Moreover, evolutionary process can also be customized. The system is able to change an algorithm to perform methods that will be affected when system performs genetic operators. For example, crossover type is able to change from one-point crossover method to two-point crossover method, and the size of tournament selection technique can be set by the user.

In conclusion, the system will create feature extraction programs by combining many image operations together, calculate the performance of each program and it will stop when it reaches termination conditions or it founded the solution to the given problem.

Chapter 3

Requirements and Analysis

This chapter explains the requirement of the system, and presents the use case and activity diagrams.

3.1 Requirements

3.1.1 User requirements

Functional requirements

- User can select objective functions, parameter setting, length of individual, training sets, test sets and primitive operations.
- User can set population size, number of generations, crossover type, mutation operator type, selection type, crossover rate, mutation rate, tournament size, max length, number of image registers and number of numerical registers.
- User can view statistical information of a GP run such as best-so-far fitness, average fitness, standard deviation of fitness, number of evolved programs and parameters of the current GP.

3.1.2 System Requirements

Functional requirements

- System can save the statistical information of a GP run into Excel.
- System can show the graph of the best-so-far fitness and average fitness.
- System can show the flowchart of the best-so-far individual.
- System can show the training images being used, their corresponding ground truth, and their corresponding output produced by the current individual.

Non-functional requirements

- Graphic user interface (GUI): easy to use and easy to understand.

1.2 Use Case Diagram

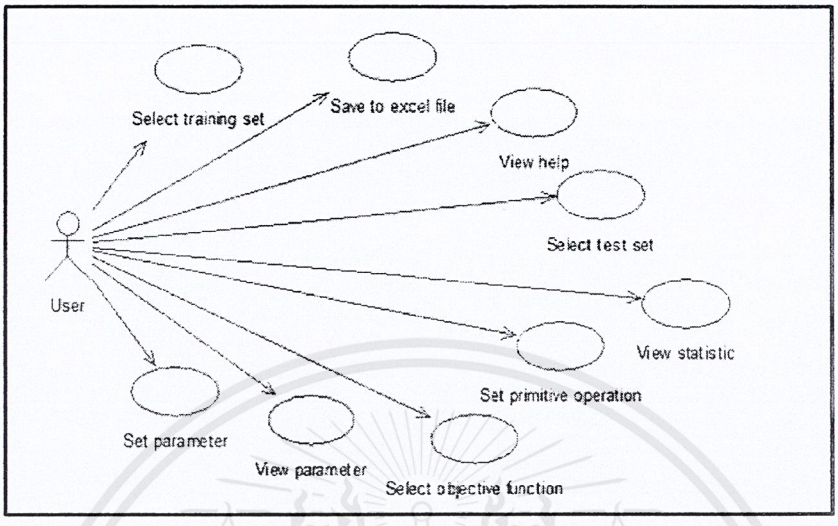


Figure 3 .1 Use case diagram

Use case is a graphical representation relationship between the user and the sub system within the system. Use case diagram is composed of an actor, use case and straight line to represent the relationship between use case and actor. In this project (Fig. 3.1), there is one user to use the system and nine use-case representation functions of the system.

- *Select training set* use case is a use case to select training image in your computer to use in the system. User can choose settings on the menu bar and choose select training set. The system will open select training set page. User can click add button to select image in your computer and click open to show thumbnail. After the user finishes, the user can click on the OK button to set image to the system.
- *Select test set use case* is a use case to select test image in your computer to use in the system. User can choose settings on menu bar and choose select test set. The system will open select test set page. User can click add button to select image in your computer and click open to show thumbnail. After the user finishes, the user can click on the OK button to set image to the system.
- *View help use case* is a use case that displays guideline or instruction for user to use the program. The user can choose help on menu bar. The system will open the help page, which shows guideline or instruction of the system.
- *View parameter use case* is a use case to show the parameter of the system. The parameter page will show training set, test set, objective function, primitive operation

and parameter. The user can choose info on menu bar and choose parameter. The system will open parameter page.

- *View statistics use case* is a use case to show best-so-far fitness, average fitness etc. of the program. User can choose info on menu bar and choose statistics. The system will open statistics page.

- *Select objective function use case* is a use case to select the goal of the system that the user want to solve and input fitness value (performance of generated program that user wants). User can choose settings on menu bar and choose select objective function. The system will open select objective function page. User can set objective function and fitness value in this page that user want and click OK button to set into the system.

- *Set primitive operation* use case is a use case to set function of image processing that user want to solve such as high pass, low pass. User can choose setting on menu bar and choose set primitive operation. The system will show set primitive operation page. User can set primitive operation that user want and click OK button to set into the system.

- *Set parameter use case* is a use case to set parameter of genetic programming that the user wants to solve. User can choose setting on menu bar and choose set parameter. The system will show set parameter page. User can set parameter that the user wants and click on the OK button to set into the system.

- *Save to excel file use case* is a use case to save the parameter of the system into the excel file that user wants. User can choose file on menu bar and choose save. The system will save parameter into excel file.

3.3 Activity Diagram

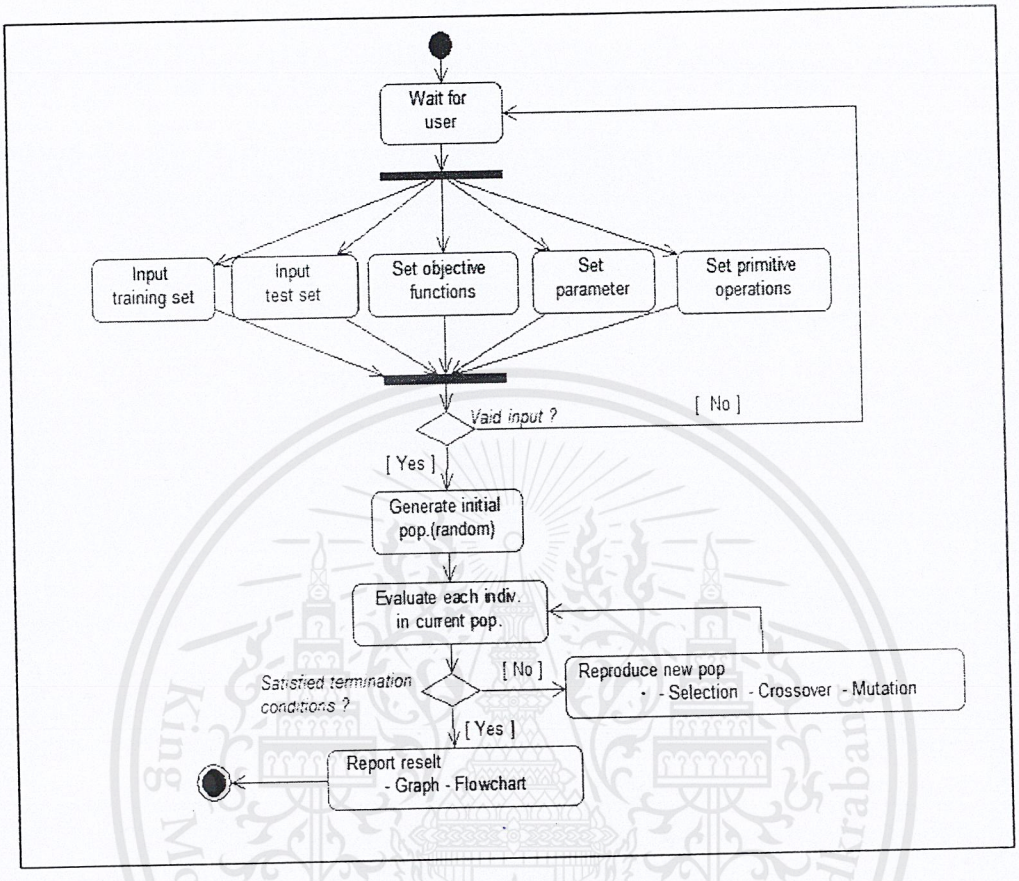


Figure 3.2 Activity diagram

Activity diagram it used to describe the step-by-step workflow of components in a system and shows the overall flow of control.

In the project, the system starts and waits for the user to select training set, select test set, set objective function, set parameter and set primitive operation. After the user finishes, the system can check the validity of the input. If the input is invalid, the system can return to prompt the user to input again. If the input is valid, the system randomly generates an initial population. After the system finishes the evaluating each individual in the current population, it satisfies termination condition. Otherwise, the system reproduces a new population using selection, crossover and mutation. The system goes back to evaluating each individual in the current population again. If satisfies termination condition, the system will report the result by graph and flowchart.

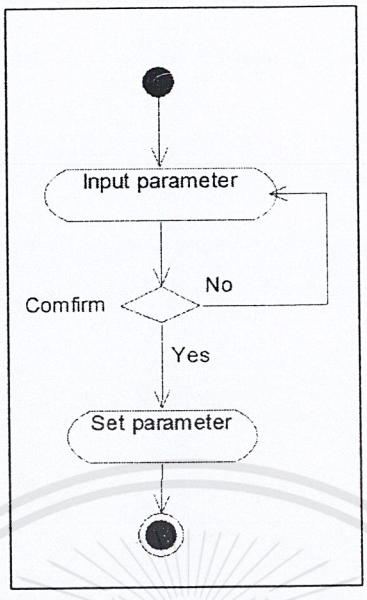


Figure 3.3 Set parameter activity

Fig. 3.3 displays the process of setting the parameter. After the user finishes, the system can check the validity of the input. If the input is invalid, the system can return to prompt the user to input the parameter again. If the input is valid, the system can set parameter to the system.

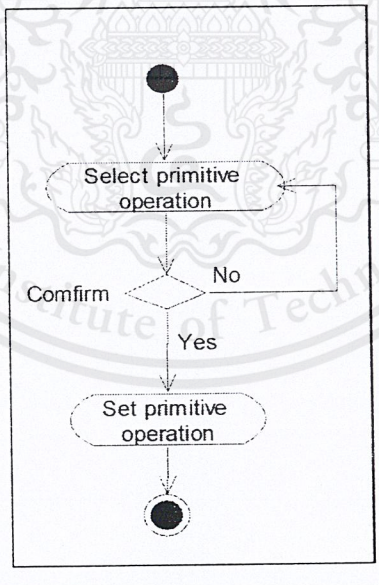


Figure 3.4 Set primitive operation activity

Fig. 3.4 displays the process of selecting the primitive operation. After the user finishes, the system can check the validity of the input. If the input is invalid, the system can return to prompt the user to input the primitive operation again. If the input is valid, the system can set primitive operation to the system.

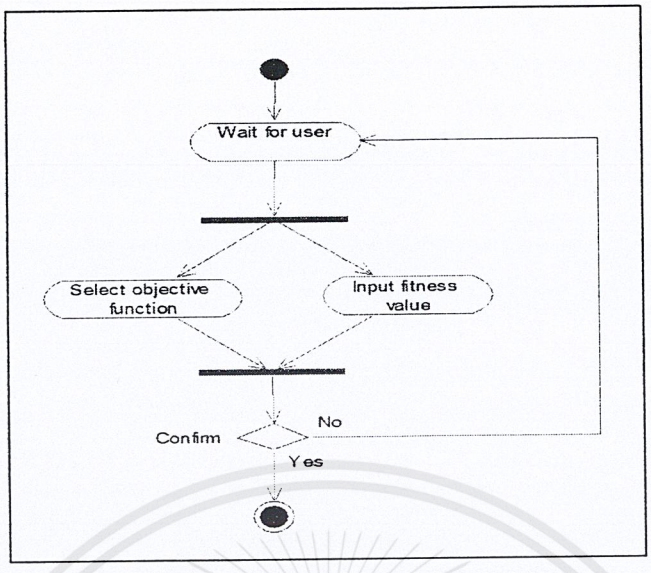


Figure 3.5 Set objective function activity

Fig. 3.5 displays the process of setting the objective function. After the user finishes, the system can check the validity of the input. If the input is invalid, the system can return to prompt the user to input the objective function again. If the input is valid, the system can set objective function to the system.

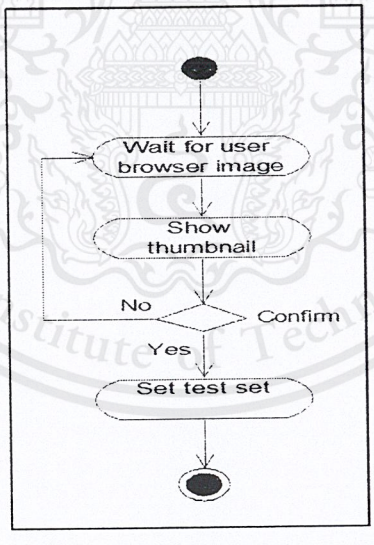


Figure 3.6 Select test set activity

Fig. 3.6 displays the process of selecting the test set. After the user finishes, the system will show thumbnail and then it can check the validity of the input. If the input is invalid, the system can return to prompt the user to input the test set again. If the input is valid, the system can set test set to the system.

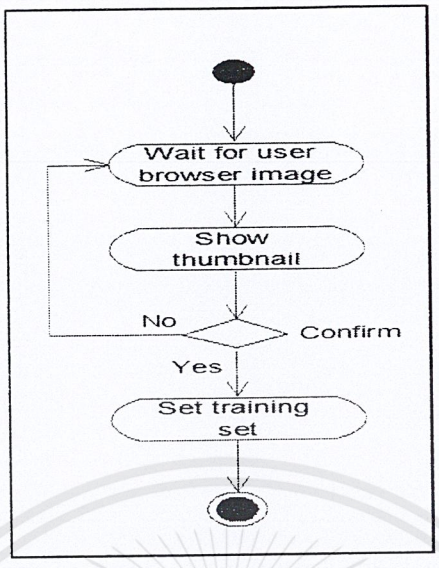


Figure 3.7 Select training set activity

Fig. 3.7 displays the process of selecting the training set. After the user finishes, the system will show thumbnail and then it can check the validity of the input. If the input is invalid, the system can return to prompt the user to input the training set again. If the input is valid, the system can set training set to the system.

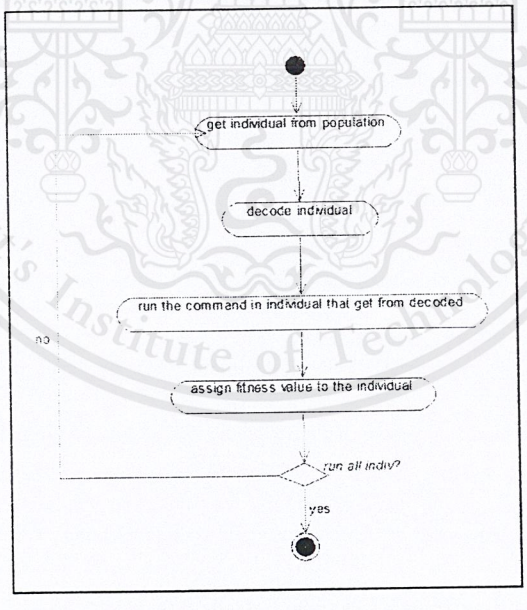


Figure 3.8 Initial population activity

Fig. 3.8 displays initialize population activity. Each individual is kept in the population until the current population is filled.

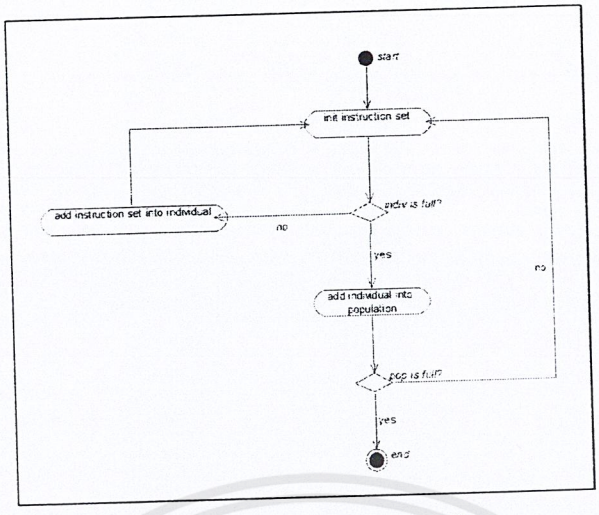


Figure 3.9 Evaluation activity

Fig. 3.9 the evaluator receives individuals from the system and uses the decoder to decode information in each individual, in order to evaluate information. Then the system returns the performance value of each individual that has been evaluated.

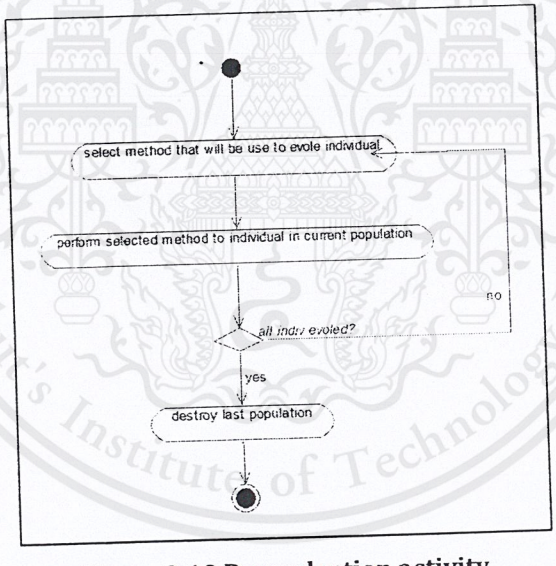


Figure 3.10 Reproduction activity

Fig. 3.10 The reproduction will create a temporary population and selects the method that will be used in to evolutionary process. Then it will select the individual in the last population through the selection method and perform mutation or crossover method to create individual in the temporary population.

The system keeps repeating this process until the temporary population is filled; then the last population will be destroyed and the temporary population is return to the system.

Chapter 4

System Design

After all requirement are met, the next step is to design the structure of the software, by designing a class and a package diagram.

4.1 The System Overview

When the system started, it will wait for the user to select training and test sets, set objective functions, GP parameters and primitive operations. After receiving, all parameters, the system will check the input whether or not it is valid. Then, the system needs to initialize the instruction sets, add individuals and keep in the population until the current population is full. (Fig. 4.1)

The system will keep evaluating each individual in the current population. In the process, the system will send individuals in current population to the evaluator and use decoder to decode the information. After decoding, it will evaluate the information using some image processing methods and return performance value of the current individual that had been evaluated to the system.

When all individuals in the current population have been processed, the system will use a reproducer object to create a temporary population and select the genetic operation to be used in evolutionary process. After that, it will select the individual in the last population through the selection method and perform mutation or crossover to create new individuals in the temporary population.

It will keep repeating this process until the temporary population is full. Then it will destroy the last population and return the temporary population to the system and process the new population again. If the performance value of the individual satisfies termination conditions, the system will return the result and represent the flow-chart of the best individual.

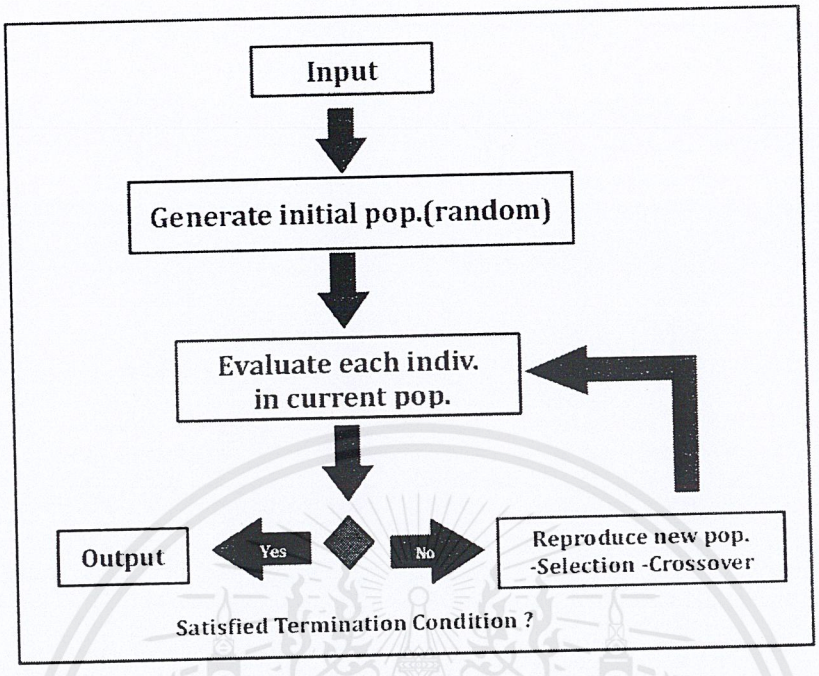


Figure 4.1 the system overview

4.2 Package Diagram and Class Diagram

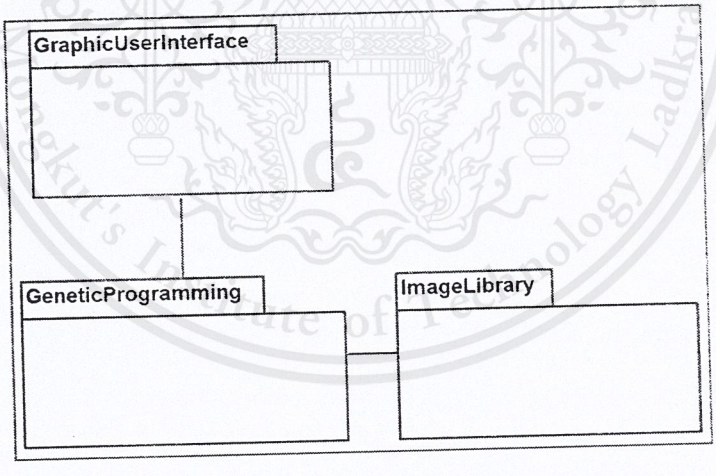


Figure 4.2 Package Diagram

The package diagram in Fig 4.2 shows that the classes used in this project can be divided into three packages: GraphicUserInterface, GeneticProgramming and ImageLibrary package.

4.2.1 GeneticProgramming Class Diagram

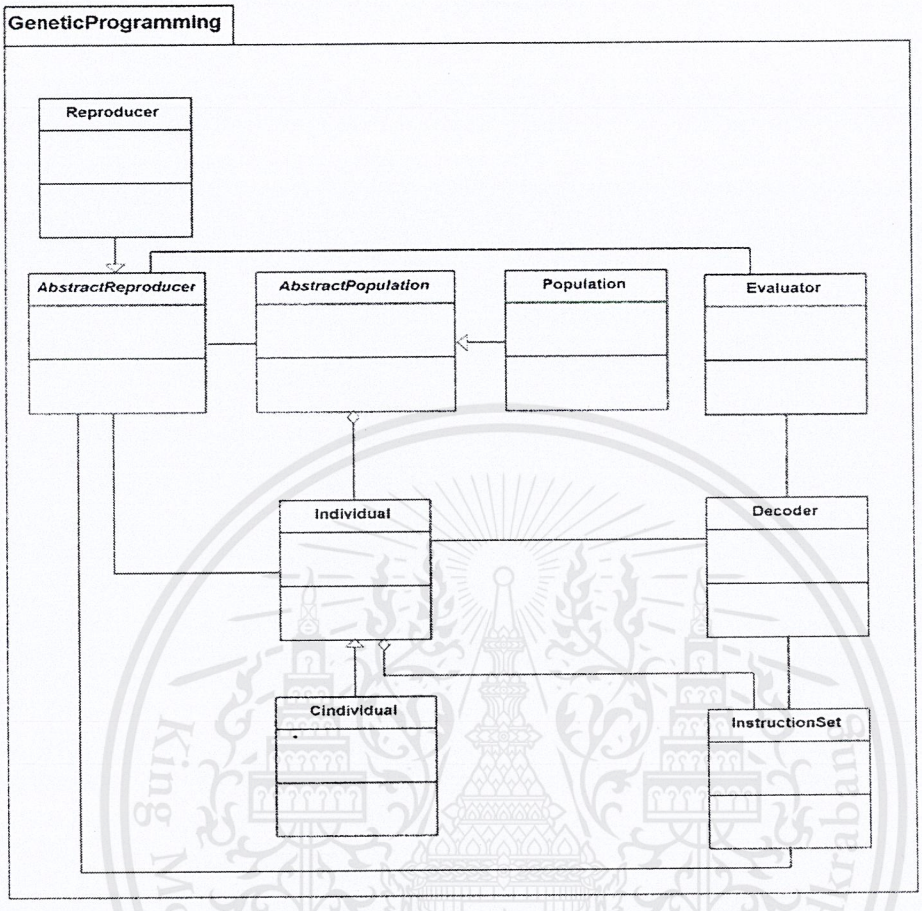


Figure 4.3 GeneticProgramming Class Diagram

This class diagram presents the structure of GP implemented in this project. The InstructionSet class, which consists of image operation methods, is a part of the Individual class. The Individual class consists of many operations, for example, an individual is a feature extraction program. The Cindividual class inherited properties from the Individual class.

The AbstractPopulation class consists of individuals and the population class. The population class will be implemented according to the individual that have been used. The AbstractReproducer class depends on the Population, Individual and InstructionSet class because in some cases, it needs to edit some information in other classes and the implementation will also be implemented according to individual that has been used. The evaluator class will calculate the performance value for each individual in the population. The decoder class depends on the Individual and InstructionSet class because it needs to decode information in other classes and send decoded information to the evaluator to process or evaluate.

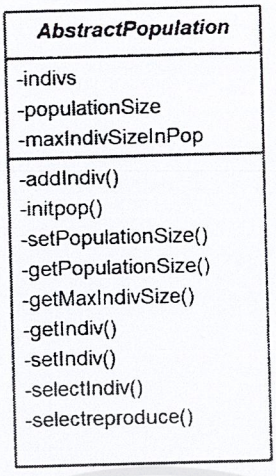


Figure 4.4 AbstractPopulation Class

Table 4.1 Attribute of Abstract Class

Attribute	Type
indivs	vector<Individual>
population Size	integer
maxIndivSizeInPop	integer

Table 4.2 Operation of AbstractPopulation Class

Operation	Return type	Parameters	Details
addIndiv()	void	individual	Add individual into population and keep in attribute name "indivs"
initpop()	void	void	Initial individuals and add into population until the population is full
setPopulationSize()	void	integer	Set attribute "populationSize" from input integer
getPopulationSize()	integer	void	Return attribute "populationSize"
getMaxIndivSize()	integer	void	Return the number of maximum size of individual in population
setMaxIndivSize()	void	integer	Set the number of maximum size of individual in population
getIndiv()	individual	integer	Get one individual which index of that individual equal to input integer
setIndiv()	void	integer, individual	Put individual into "indivs" where the index equals to the input integer

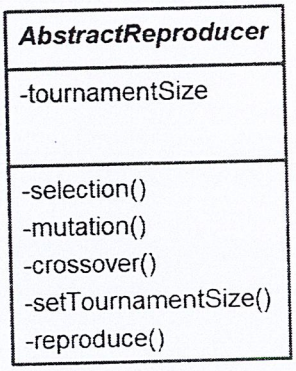


Figure 4.5 AbstractReproducer Class

Table 4.3 Attribute of AbstractReproducer Class

Attribute	Type
tournamentSize	integer

Table 4.4 Operation of AbstractReproducer Class

Operation	Return type	Parameters	Details
selection()	individual	population	Take population as an input and randomly choose individuals depend on "tournament Size" and return the best one
mutation()	individual	population	Randomly edit some information inside individual that has been chosen
crossover()	vector<individual>	population	Select two individuals to be parents and select a "cut point" to exchange instruction set inside two individuals
setTournamentSize()	void	integer	Take input integer and set the "tournament Size"
reproduce()	population	population	Take population and use the methods above to create a new population and destroy the last population

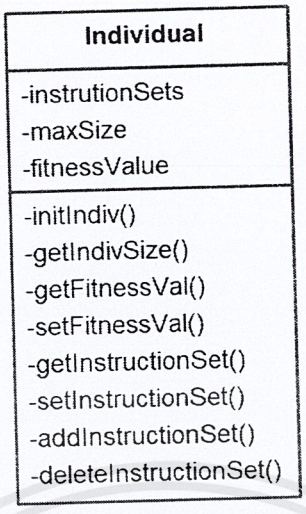


Figure 4.6 Individual Class

Table 4.5 Attribute of Individual Class

Attribute	Type
instuructionSets	vector<instructionSets>
maxSize	integer
fitness	float

Table 4.6 Operation of Individual Class

Operation	Return type	Parameters	Details
initIndiv()	void	integer	Initialize instruction set and add individuals
getIndivSize()	integer	void	Return individual size
getFitnessVal()	float	void	Return fitness value
setFitnessVal()	void	float	Take number and set it as a fitness value of individual
getInstructionSet()	instructionset	integer	Return instruction set where index equals to the input integer
setInstructionSet()	void	integer ,instructionset	Set instructions in "instructionSets" where index equals to the input integer
addInstructionSet()	void	instructionset	Add input instruction set to "instructionSets"
deleteInstructionSet()	void	void	Delete the last instruction set in "instructionSets"

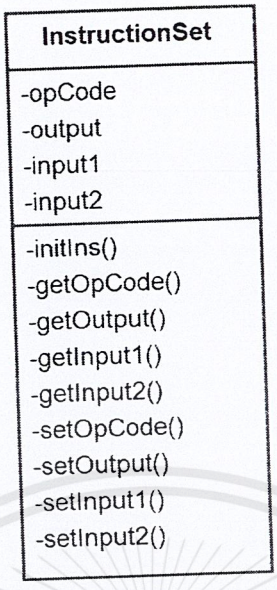


Figure 4.7 Instruction Set Class

Table 4.7 Attribute of Instruction Set Class

Attribute	Type
opCode	integer
output	integer
input1	integer
Input2	integer

Table 4.8 Operation of Instruction Set Class

Operation	Return type	Parameters	Details
initIns()	void	void	Randomly assign a number to every attribute in InstructionSet
getOpCode()	integer	void	Return opcode as an integer
getOutput()	integer	void	Return output as an integer
getInput1()	integer	void	Return input1 as an integer
getInput2()	integer	void	Return input2 as an integer
setOpCode()	void	integer	Set input integer as an opcode
setOutput ()	void	integer	Set input integer as an output
setInput1 ()	void	integer	Set input integer as an input1
setInput2()	void	integer	Set input integer as an input2

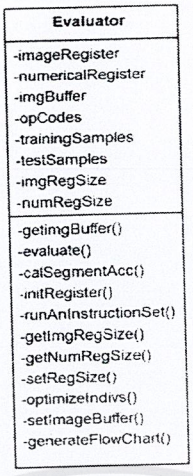


Figure 4.8 Evaluate Class

Table 4.9 Attribute of Evaluate Class

Attribute	Type
imageRegister	vector<cv::Mat>
numericalRegister	vector<float>
imgBuffer	vector<cv::Mat>
opCodes	vector<string>
TrainingSamples	vector<string>
testSamples	vector<string>
imgRegSize	integer
numRegSize	integer

Table 4.10 Operation of Evaluate Class

Operation	Return type	Parameters	Details
getImgBuffer ()	vector<cv::mat>	void	Return "imgBuffer"
evaluate ()	float	individual	Take individual to process and return result as a floating point
calSegmentAcc ()	float	cv::mat,cv::mat	Take two images and calculate the difference
initRegister ()	void	cv::mat	Initial image register and numerical register
runAnInstructionSet()	void	string	Run each instruction set that has been decoded from the decoder
getImgRegSize ()	integer	void	Return the size of image register
getNumRegSize ()	integer	void	Return the size of numerical register
setRegSize ()	void	integer, integer	Set size of register for both numerical and image register
optimizeIndivs()	vector<string>	vector<string>	Delete some instruction sets in individuals that is not used
setImageBuffer()	void	void	Initialize images in buffer to use repeatedly
generateFlowChart()	vector<string>	vector<string>	Generate flowchart protocol to use in GUI

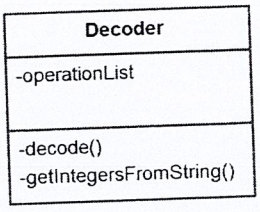


Figure 4.9 Decoder Class

Table 4.11 Attribute of Decoder Class

Attribute	Type
operationList	vector<string>

Table 4.12 Operation of Decoder Class

Operation	Return type	Parameters	Details
decode ()	vector<string>	individual	Take individual and decode it into sets of operations
getIntegersFromString()	vector<integer>	string	Decode each operation into sets of index number

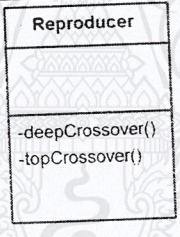


Figure 4.10 Reproducer Class

Table 4.13 Operation of CReproducer Class

Operation	Return type	Parameters	Details
deepCrossover()	vector<indiv>	population	It will change information in each feature in individual
topCrossover()	vector< indiv >	population	It will change the hold feature by not change its information

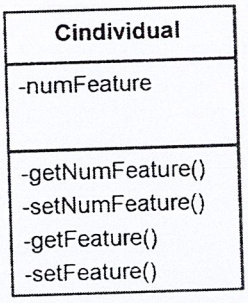


Figure 4. 11 CIndividual Class

Table 4.14 Attribute of CIndividual Class

Attribute	Type
numFeature	Integer

Table 4.15 Operation of CIndividual Class

Operation	Return type	Parameters	Details
getNumFeature()	void	integer	Return NumFeature
setNumFeature()	integer	void	Set NumFeature
getFeature()	individual	integer	Return Feature
setFeature()	void	integer, individual	Set Feature

4.2.2 GraphicUserInterface Class Diagram

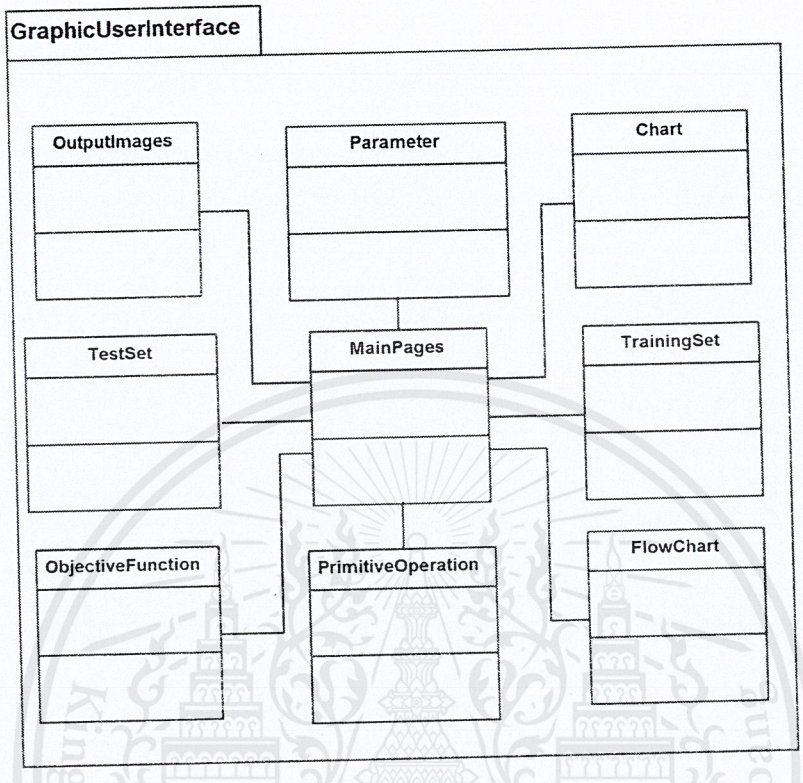


Figure 4.12 GraphicUserInterface Class Diagram

The class diagram represents the structure of the GUI implemented in this project (Fig. 4.12). The MainPages class is the main of the GUI. This class is used to control the system. The MainPages class calls the other classes to receive input to the system.

The TrainingSet and TestSet classes are classes used to add image of the training set to the system. The ObjectiveFunction, PrimitiveFunction and Parameter classes are classes that receives input parameter of GP and image processing to the system.

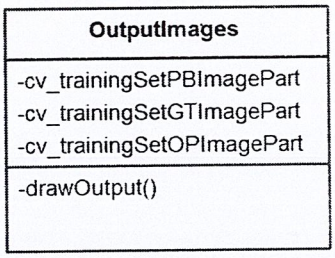


Figure 4.13 OutputImages Class

Table 4.16 Attribute of OutputImages Class

Attribute	Type
trainingSetPBImagePart	vector<QString>
trainingSetGTImagePart	vector<QString>
trainingSetOPImagePart	vector<QString>

Table 4.17 Operation of OutputImages Class

Operation	Return type	Parameters	Details
drawOutput()	void	void	Shows training image, ground truth and output image on the widget

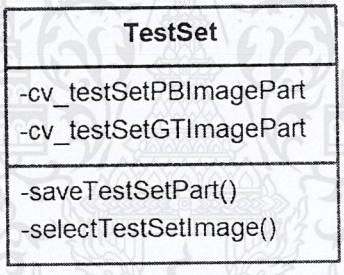


Figure 4.14 TestSet Class

Table 4.18 Attribute of TestSet Class

Attribute	Type
cv_testSetPBImagePart	vector<QString>
cv_testSetGTImagePart	vector<QString>

Table 4.19 Operation of TestSet Class

Operation	Return type	Parameters	Details
saveTestSetPart()	Void	void	Set the test set problem image and ground truth image to the system
selectTestSetImage()	Void	void	Select test set problem images from the computer and show preview on the screen

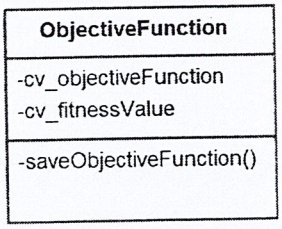


Figure 4.15 ObjectiveFunction Class

Table 4.20 Attribute of ObjectiveFunction Class

Attribute	Type
cv_objectiveFunction	QString
cv_fitnessValue	QString

Table 4.21 Operation of ObjectiveFunction Class

Operation	Return type	Parameters	Details
saveObjectiveFunction	void	void	Set the objectivefunction and desired fitness value to the system

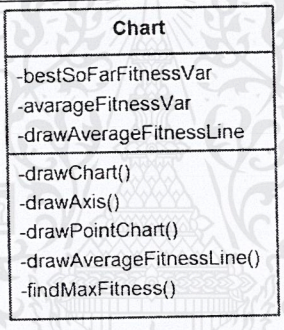


Figure 4.16 Chart Class

Table 4.22 Attribute of Chart Class

Attribute	Type
bestSoFarFitnessVar	vector<QString>
avarageFitnessVar	vector<QString>

Table 4.23 Operation of Chart Class

Operation	Return type	Parameters	Details
drawChart()	void	void	Draw the graph
drawAxis()	void	void	Draw the axis of the graph
drawPointChart()	void	integer, float	Draw the point on the axis of the graph
drawBestSoFarLine()	void	vector<float>, float	Draw the Best-So-Far line on the graph
drawAvarageLine()	void	vector<float>, float	Draw the average line on the graph
findMaxFitness()	float	void	Find the maximum of fitness value

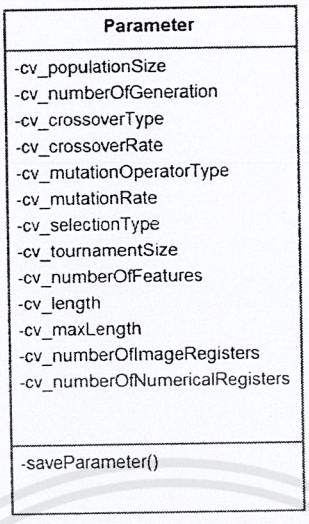


Figure 4.17 Parameter Class

Table 4.24 Attribute of Parameter Class

Attribute	Type
cv_populationSize	QString
cv_numberOfGeneration	QString
cv_crossoverType	QString
cv_crossoverRate	QString
cv_mutationOperatorType	QString
cv_mutationRate	QString
cv_selectionType	QString
cv_tournamentSize	QString
cv_numberOfFeatures	QString
cv_length	QString
cv_maxLength	QString
cv_numberOfImageRegisters	QString
cv_numberOfNumericalRegisters	QString

Table 4.25 Operation of Parameter Class

Operation	Return type	Parameters	Details
saveParameter()	void	void	Set the parameter, such as population size, number of generation, crossover type etc. to the system

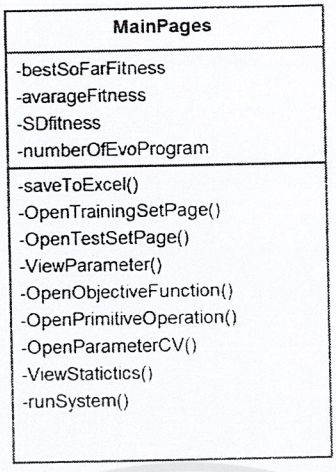


Figure 4.18 MainPage Class

Table 4.26 Attribute of MainPage Class

Attribute	Type
bestSoFarFitness	QString
avarageFitness	QString
SDfitness	QString
numberOfEvoProgram	QString

Table 4.27 Operation of MainPage Class

Operation	Return type	Parameters	Details
saveToExcel()	void	void	Save parameter to excel file
openTrainingSetPage()	void	void	Open training set page to set image of training set to the system
openTestSetPage()	void	void	Open test set page to set image of test set to the system
viewParameter()	void	void	Open page to view all parameter that user add
openObjectiveFunction()	void	void	Open page to add objective function and fitness value to the system
openPrimitiveOperation()	void	void	Open page to add primitive operation of image processing to the system
openParameterCV()	void	void	Open page to add parameter to the system
viewStatictics()	void	void	Show statistics
runSystem()	void	void	Run the system

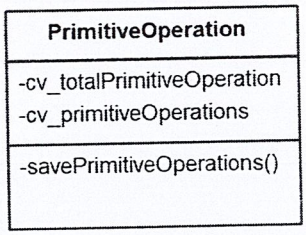


Figure 4.19 PrimitiveOperation Class

Table 4.28 Attribute of PrimitiveOperation Class

Attribute	Type
cv_totalPrimitiveOperation	Integer
cv_primitiveOperation	vector<QString>

Table 4.29 Operation of PrimitiveOperation Class

Operation	Return type	Parameters	Details
savePrimiriveOperations()	void	void	Set the primitive operations to the system

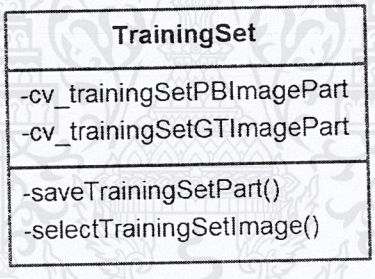


Figure 4.20 TrainingSet Class

Table 4.30 Attribute of TrainingSet Class

Attribute	Type
cv_trainingSetPBImagePart	vector<QString>
cv_trainingSetGTImagePart	vector<QString>

Table 4.31 Operation of TrainingSet Class

Operation	Return type	Parameters	Details
saveTrainingSetPart()	void	void	Set the training set problem image and ground truth image to the system
selectTrainingSetImage()	void	void	Select training set problem images from the computer and show preview on the screen

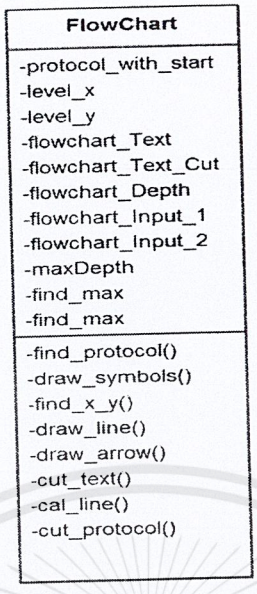


Figure 4.21 Flowchart Class

Table 4.32 Attribute of Flowchart Class

Attribute	Type
protocol_with_start	vector<QString>
level_x	vector< Integer >
Level_y	vector< Integer >
Flowchart_Text	vector<QString>
Flowchart_Text_Cut	vector<QString>
FlowchartDepth	vector< Integer >
Flowchart_Input_1	vector<QString>
Flowchart_Input_2	vector<QString>
maxDepth	integer

Table 4.33 Operation of Flowchart Class

Operation	Return type	Parameters	Details
find_protocol()	void	void	
draw_symbols()	void	void	Draw symbols
find_x_y()	void	void	
draw_line()	void	integer, integer, integer, integer	Draw lines on the flowchart
draw_arrow()	void	integer, integer, integer	Draw arrows of the line on the flowchart
cut_text()	void	vector<QString>	
cal_line	void	void	Calculate the point to draw the line on the flowchart
cut_protocol()	vector<QString>	QString	Cut the flowchart protocol

4.2.3 ImageLibrary Class Diagram

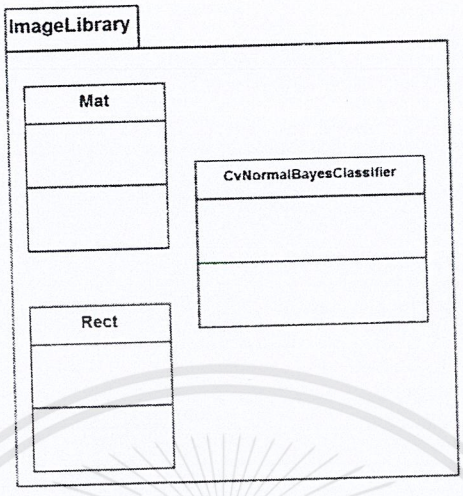


Figure 4.22 ImageLibrary Class Diagrams

Fig. 4.22 represents ImageLibrary Package diagrams that consist of three classes: Mat, Rect and CvNormalBayesClassifier.

Chapter 5

Development

In this chapter, we describe GP techniques and other tools such as OpenCV and Qt. We also explain algorithms used to show the graph and flow chart of best-so-far program.

5.1 Tools

The software tools listed below are used in design and development of the program.

5.1.1 Microsoft Visual Studio 2010

Microsoft Visual Studio is an integrated development environment (IDE), created by Microsoft to help developers to create and design applications. Microsoft Visual Studio can support several languages, such as C++, C#, Net framework etc. In this thesis, we use Microsoft Visual Studio to support image processing (OpenCV) and Graphic user interface (Qt).

5.1.2 OpenCV

OpenCV (Open source Computer Vision) library is an open source library developed by Intel Corporation. The official manual is freely available on OpenCV website. This library allows high-level functions for computer vision and image processing. OpenCV can run on many computer platforms. It focuses mainly on real-time image processing.

Support C and C++ classes that implement some popular image processing and computer vision algorithms. It provides high-speed implementations of functions used in image processing.

5.1.3 Qt SDK

Qt is a tool to create an application and graphical user interface. It works on Desktop PCs, smart phones and embedded systems. Qt is a cross-platform program, meaning that it works on multiple operating systems (OS).

Nowadays, there are various methods for creating and developing GUIs for an application or a system. Also, there are different tools to choose from, such as, VC# and

VB.NET on WIN CE. Qt is an interesting alternative because the user may choose from its vast API and libraries. It is similar to Microsoft's MSDN, but it is open source, where it doesn't require the user to restart from the beginning and avoids losing valuable time. Qt contains different application programming interface (API) and libraries, which are written using C++.

Consequently, the user is able to develop applications and it also contributes to the development of C++, Java, Python, Perl, Pascal, and PHP. Nevertheless, the user chooses the language that would be implemented in the development of the application. Qt also has special capabilities aside from GUI, for instance, connecting with SQL database, reading XML file, managing thread, network and managing files.

5.1.4 Libxl Library

Libxl library is a library to use to read and write excels files. In this project used to consume to exporting and extracting parameter to excel file. Library used in C++ and support excels formats xls and xlsx. Libxl library are many command to use such as add picture use to add image to excel.

5.2 Techniques

5.2.1 Genetic programming

This project aims to develop a synthesis system that is able to find feature extraction programs for real-world object. In addition, the system would be able to find feature extraction programs that can solve the given task automatically by receiving some inputs from the user and generates a result in the flow-chart of feature extraction program. Moreover, the system must be able to show the performance of best-so-far program while the system is running. Due to the goal of the project, we found that Genetic programming (GP) is suitable for creating such system.

GP is a machine learning technique to find the optimal solution. In our system, it needs a pair of images; a problem image and a solution of given problem image (called "ground truth") for the system to learn. Once the system learned, it will be able to solve the problem according to what it learned.

GP consists of two main parts; one is responsible for generating population and the other evaluates the generated population. After the population was evaluated, the system will receive a performance value of all members in the population. If performance value does not satisfy the conditions, the system will re-generate the population again.

In the system, a feature extraction is created by combining many image operations together. Each image operation is one chromosome (called "instruction set"), meaning that, each set of chromosome is a feature extraction program (called "individual"). An instruction set consists of four parts; operation code, first input for the operation, second input for the operation, and output of the operation. Four of them will be first generated in real number and will be used in evaluation phase. As mentioned above, an individual will store many instruction sets inside. Finally, each generated individual will be kept in a population.

The evaluation phase will use some functions in image library depending on an instruction set. The system will change the real number generated in the instruction set into a readable message. Then, it will take a problem image and run image operation according to the information in each instruction set. After the image is processed, the system will compare the output image to the ground truth image to calculate the performance value (called "fitness value").

While the system is processing, it will keep changing the flow-chart of the best-so-far program that it can generated and also shows the highest fitness value of individual and the average fitness value in each population via graph.

The system will stop when it reaches termination conditions or it found the solution to the given problem.

5.2.2 Graph

In an image recognition synthesis system, a graph is a graphical representation of the best-so-far fitness and average fitness, which are generated by the system. As shown in Fig. 5.1,the x-axis is the number of generation and the y-axis is the fitness value, while the best-so-far and average fitness are shown as the blue and red lines, respectively.

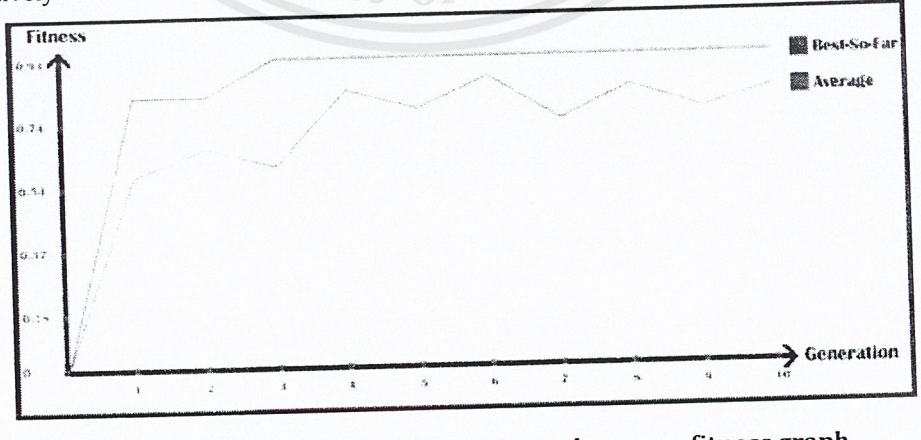


Figure 5.1 An example of best-so-far and average fitness graph.

This graph is created using QPainter, which is a library of Qt. It is used to paint on widgets. The system will generate best-so-far fitness protocol and average fitness protocol to the function, as shown in Fig. 5.2.

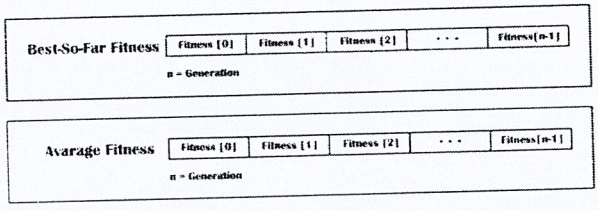


Figure 5.2 Best-So-Far and average fitness protocols.

After the function draws the graph, the steps in creating a graph are as follow:

1) Create a widget using QWidget for drawing graphs. (Fig. 5.3)

Figure 5.3 Widget for drawing graph.

2) Draw x and y axis using QPainter on the widget. (Fig 5.4)

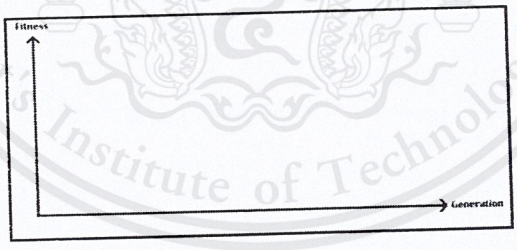


Figure 5.4 Widget with the axis.

3) Calculate the distance between every point on the x-axis and y-axis.

• The distance between each point on the x-axis can be calculated using the size of the best-so-far protocol or average fitness protocol to find maximum generation. Then, we calculate the distance between every point on the x-axis by using the equation (5-1).

$$\frac{A}{B} = C, \qquad (5\text{-}1)$$

where A is size of x-axis, B is maximum generation and C is distance between each point on the x-axis.

- The distance between every point on the y-axis can be calculated using the maximum fitness of the best-so-far fitness protocol and average fitness protocol. After finding the maximum fitness, we find the distance of each point on the y-axis by using the equation (5-2).

$$\frac{A}{B} = C, \tag{5-2}$$

where A is size of y-axis, B is maximum fitness and C is distance between each point on the y-axis.

4) Draw a dot on the x-axis and y-axis according to the calculation made in number three. As shown in Fig. 5.5

Figure 5.5 Graph with axis and point.

5) Calculate the value of the best-so-far fitness line and average fitness line. It is necessary to calculate the x and y axis of each generation.

- Find the x-axis of every generation. It can be calculated by equation (5-3).

$$\left(\left(\frac{A}{B}\right) \times 100\right) \times C = D, \tag{5-3}$$

where A is generation number, B is maximum generation, C is range of x-axis and D is x-position of each generation.

- Find the y-axis of every generation. It can be calculated by equation (5-4).

$$\left(\left(\frac{A}{B}\right) \times 100\right) \times C = D, \tag{5-4}$$

where A is fitness number, B is maximum fitness, C is range of y-axis and D is y-position of each generation.

6) Draw the best-so-far fitness line and average fitness according to the calculation made in number 5. (Fig. 5.6)

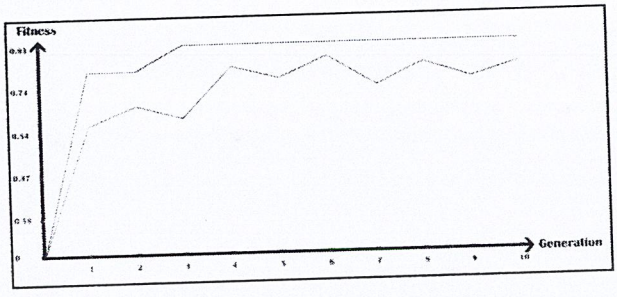


Figure 5.6 A Complete graph

When the function receives new values from the system, the function will recalculate the positions of x-axis and y-axis every time and redraw the graph.

5.2.3 Flow chart

In an image recognition synthesis system, a flowchart is a graphical representation of the working steps of image processing functions, which is generated by the system. As shown in Fig. 5.7, the flowchart displays the working process of the system and the relationship between each step in order.

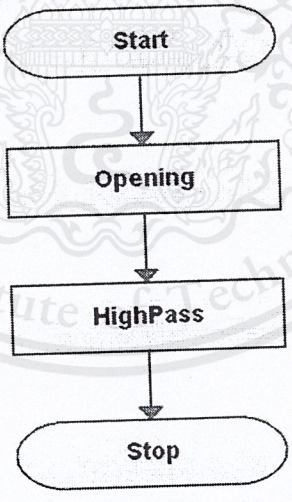


Figure 5.7 an example of Flowchart

After the system sends a flowchart protocol, as shown in fig 5.8, functions will decode it into four parts, including: name, depth, first input and second input, as shown in fig 5.9.

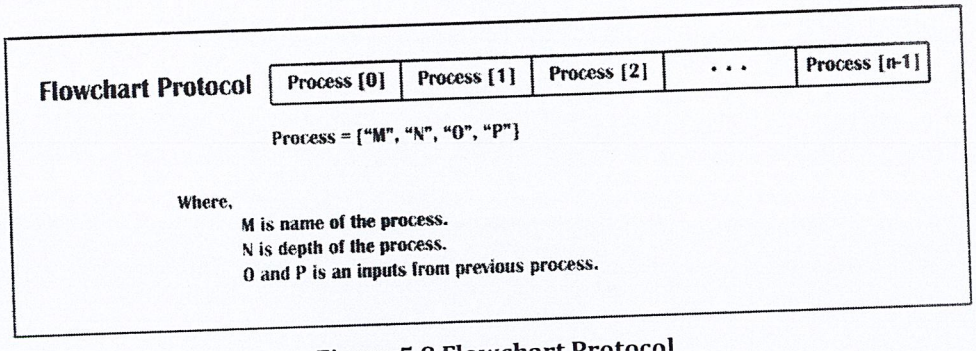


Figure 5.8 Flowchart Protocol

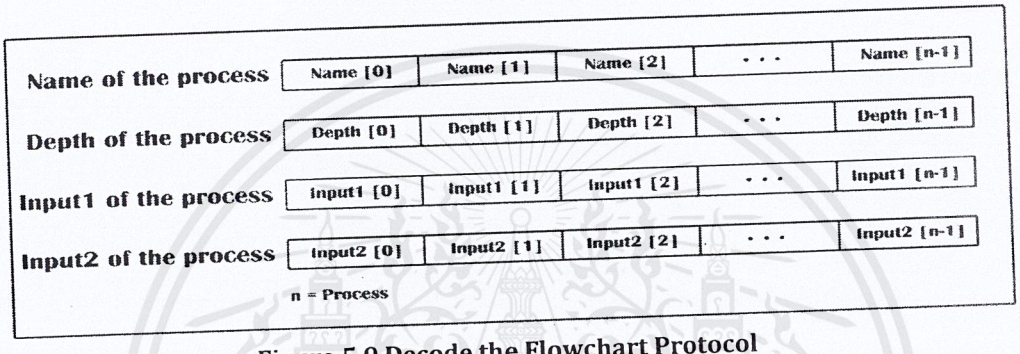


Figure 5.9 Decode the Flowchart Protocol

After the function decodes the data, the steps in creating a flowchart are as follow:

1. Create a widget using QWidget for drawing a flowchart (Fig. 5.10).

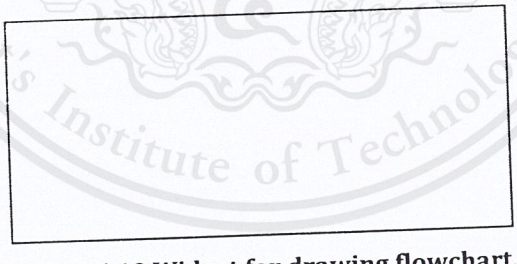


Figure 5.10 Widget for drawing flowchart.

2. Calculate the x and y position of every process. The y position is the flowchart depth value and the x position can be found by calculating the repeated values in flowchart depth.

3. Draw the process according to the calculations made in number 2 using QPainter on the widget. (Fig. 5.11)

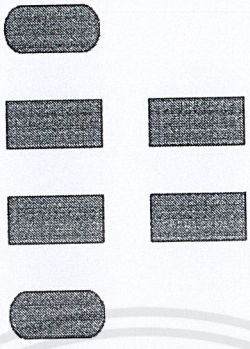


Figure 5.11 Widget with the process.

4. Calculate the position for drawing the lines, which is calculated according to the line's type. There are 6 types of lines; as shown in Fig. 5.12

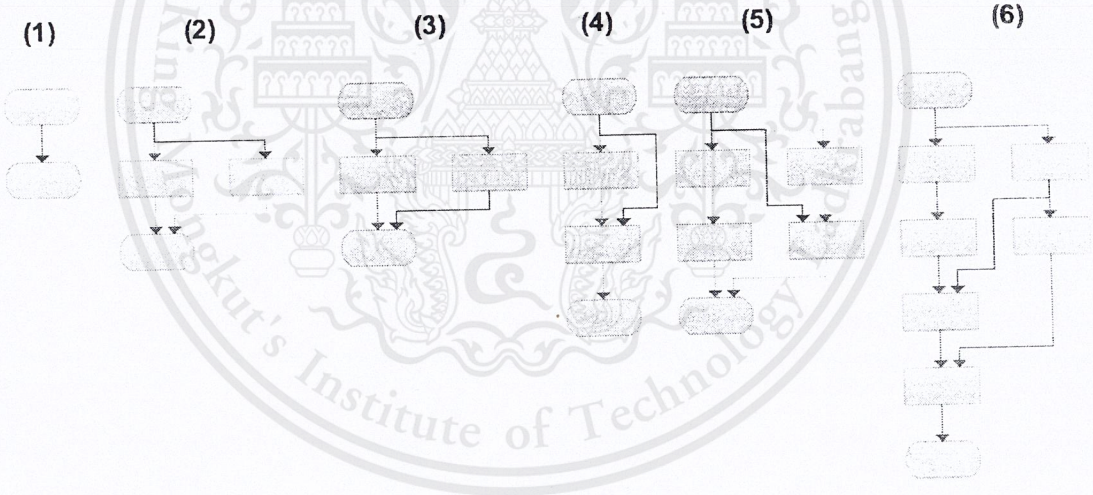


Figure 5.12 Type of lines.

5. In Fig. 5.13, an algorithm is used to find the position of lines and draw lines to the flowchart.

```
ALGORITHM FIND_POSITION_OF_LINES

VARIABLES:      begin_x_co, begin_y_co, end_x_co, end_y_co

                angle_x1_co, angle_y1_co, angle_x2_co, angle_y2_co

                angle_x3_co, angle_y3_co, angle_x4_co, angle_y4_co

GET: begin_x_position, begin_y_position, end_x_positon, end_y_positon


begin_x_co = (begin_x_position * distance between box) + (size of box / 2)

end_x_co = (end_x_position * distance between box) + (size of box / 2)

begin_y_co = begin_y_position * distance between box

end_y_co = end_y_positon*distance between box

angle_x1_co = begin_x_co
angle_y1_co = begin_y_co + (distance between box/2)

angle_x4_co = end_x_co
angle_y4_co = end_y_co – (distance between box/2)

IF end_y_positon - begin_y_position != 1 THEN
        IF begin_x_position <=  end_x_positon THEN
                angle_x2_co = angle_x1_co+
        ELSE
                angle_x2_co = angle_x1_co-
        ENDIF
        angle_x3_co = angle_x2_co

ELSE
        angle_x2_co = angle_x1_co
        angle_x3_co = angle_x4_co
ENDIF

angle_y2_co = angle_y1_co
angle_y3_co = angle_y4_co

draw_line(begin_x_co, begin_y_co, angle_x1_co, angle_y1_co)
draw_line(angle_x1_co, angle_y1_co, angle_x2_co, angle_y2_co)
draw_line(angle_x2_co, angle_y2_co, angle_x3_co, angle_y3_co)
draw_line(angle_x3_co, angle_y3_co, angle_x4_co, angle_y4_co)
draw_line(angle_x4_co, angle_y4_co, , end_x_co, end_y_co)

END FIND_POSITION_OF_LINES
```

Figure 5.11 Algorithm to draw lines

When the function receives new values from the system, the function will recalculate every time and redraw the flowchart.

Chapter 6

Evaluations and Discussions

This chapter represents screenshots of the program and describes the effectiveness and the deficiencies of the software and an evaluate the experimentation results.

6.1 Screenshots

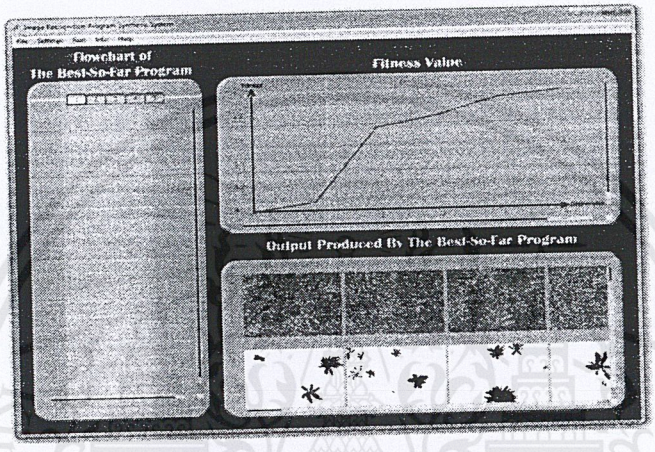


Figure 6.1 Main page

Figure 6.2 Training set pages

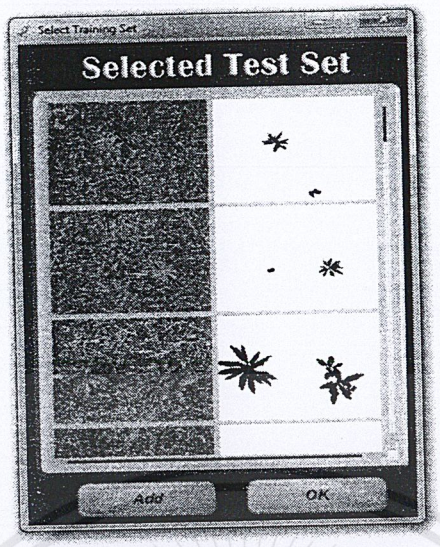


Figure 6.3 Test set page

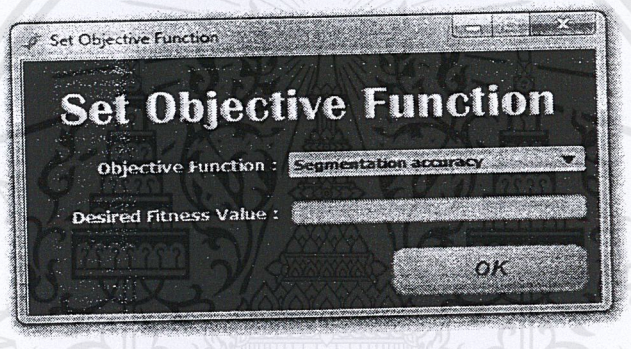


Figure 6.4 Objective function pages

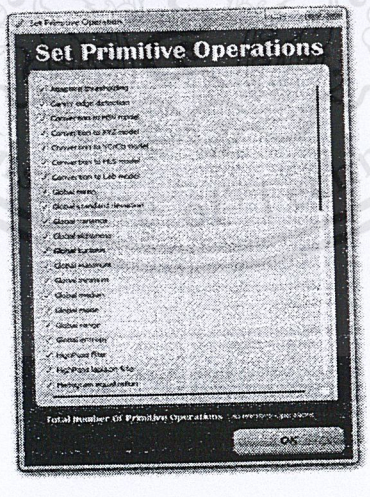


Figure 6.5 Primitive operation pages

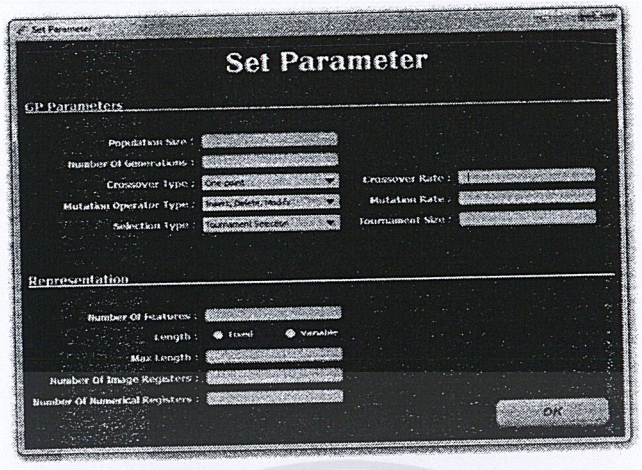


Figure 6.6 Set parameter page

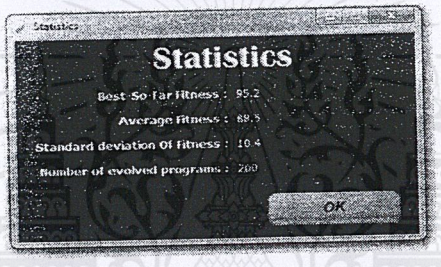


Figure 6.7 View statistics

6.2 Evaluations

The system used genetic programing as a technique in order to find an optimal solution. First, we only used one set of image operation to construct the image recognition program. After that, we tried to improve our system by adding KNN classifier as a tool to help and improve system's efficiency. But KNN classifier takes too much time than we expected in processing the image. After discussion, we decided to change the tool from KNN classifier to Bayes classifier instead.

In each solution, we found that some of the image operations have neither been used nor operated. Therefore, we designed an algorithm and created functions for the system to eliminate useless image operations in each solution that been generated.

Step 1: created ground truth Image.

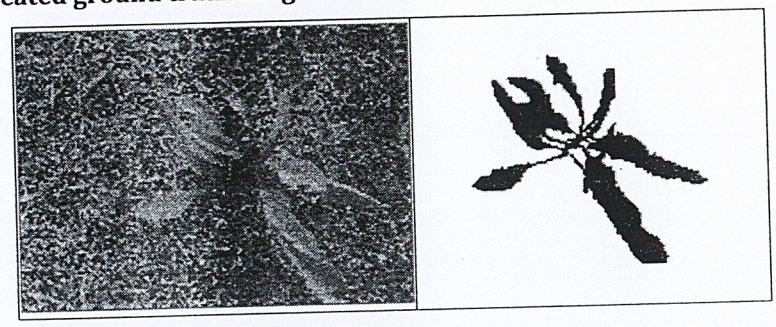


Figure 6.8 Created ground truths Image

We need to find problem image and create ground truth image for our system because genetic programing is a machine learning technique and our system needs to find feature extraction program.

In order to do so, we need to give good input for the system to learn to make the system good enough to find the solution that is able to generate the program that we want. If the training image is not good enough, it would be very difficult or impossible for the system to find the best solution.

Step 2 : input parameters into system.

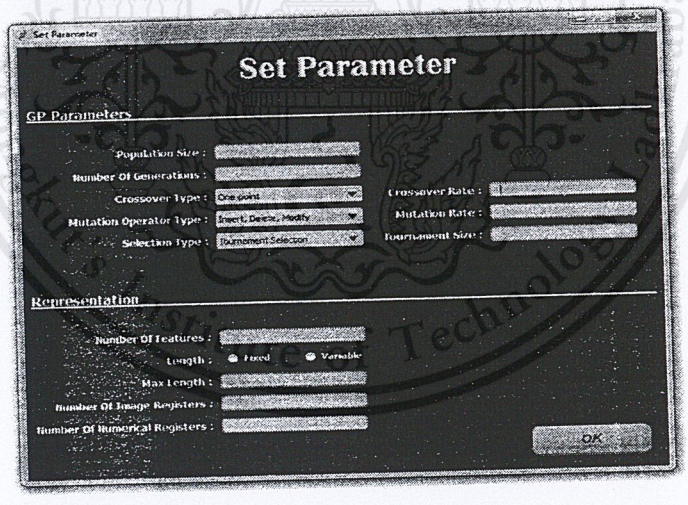


Figure 6.9 Input parameters

In Fig 6.9, the system receives input from the user. Because the system is quite random, the system may not be able to find the best solution if the input is not good enough.

Each parameter in our system is very important; it can effects the efficiency and effectiveness of our system and the solution that the system generates.

Step 3 : train the system to learn by use images that been input.

After the system received all input including all parameters and train data, system take time to process and will generate the output for user.

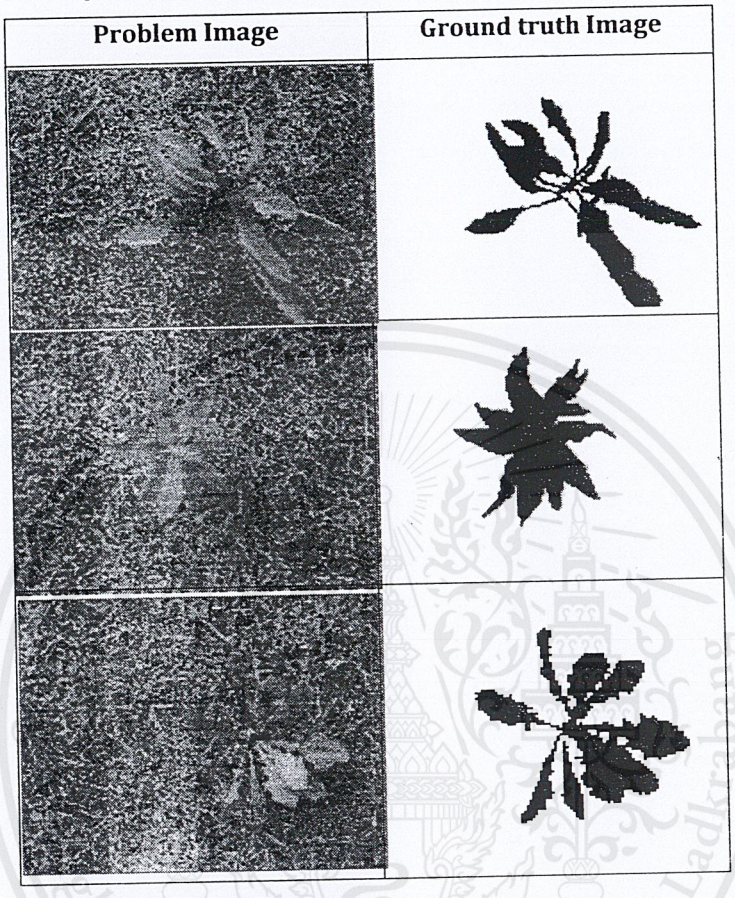


Figure 6.10 Problem Images and Ground truth Images

Problem images and ground truth images (Fig. 6.10) have been used for system training to generate.

The parameters that been set in this operation listed below; Population size = 50, Number of generations =50, Size of individual =20, and Primitive operations 50:

1. Adaptive thresholding
2. Image addition
3. Canny edge detection
4. Image subtraction
5. Global mean
6. Image multiplication
7. Global standard deviation
8. Image division
9. Global variance
10. Local histogram
11. Global skewness
12. Local variance
13. Global kurtosis
14. Local skewness
15. Global maximum
16. Local kurtosis
17. Global minimum
18. Local maximum
19. Global median
20. Local minimum
21. Global mode
22. Local mode
23. Global range
24. Local range
25. Global entropy
26. Local entropy
27. HighPass filter
28. Lowpass filter
29. HighPass laplacin filter
30. Lowpass gaussian filter
31. Histogram equalization
32. Median filter
33. Image square root
34. Morphological dilation
35. Image absolute
36. Morphological erosion
37. Image scaling
38. Morphological opening
39. Image negative
40. Morphological closing
41. Sobel operation
42. Scharr operation
43. Thresholding
44. Convert to HSV
45. Convert to XYZ
46. Convert to YCrCb
47. Convert to HLS
48. Convert to Lab
49. Insert Channel
50. Extract Channel

Table 6.1 Weed database

Experimental no.	Accuracy (%)	
1	90.67	
2	93.16	
3	92.67	
4	92.87	
5	92.17	
6	91.07	
7	93.87	#2nd
8	93.97	#1st
9	92.47	
10	93.51	#3rd
AVG	**92.643**	
STD	**1.1034**	
Max	**93.97**	
Min	**90.67**	

The result of ten experimental program from the system is shown in Table 6.1. In addition, it also shows the accuracy of the best and worst program, including STD and average.

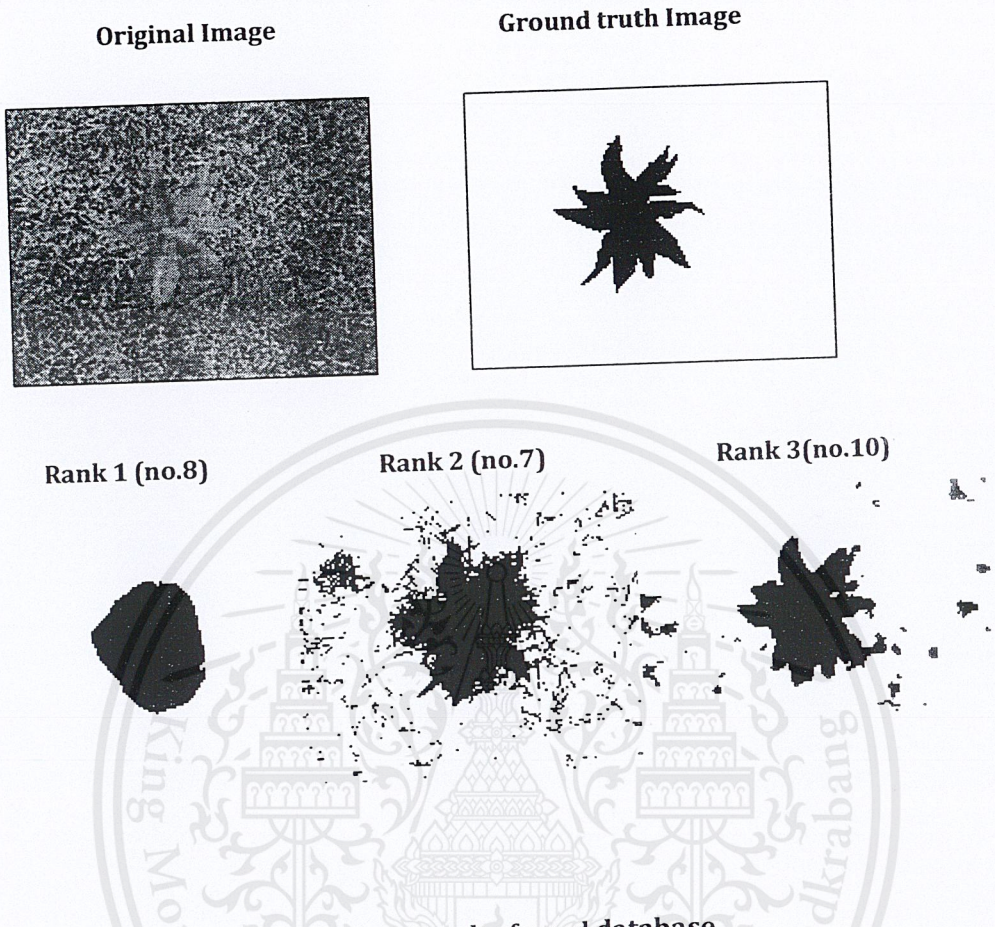
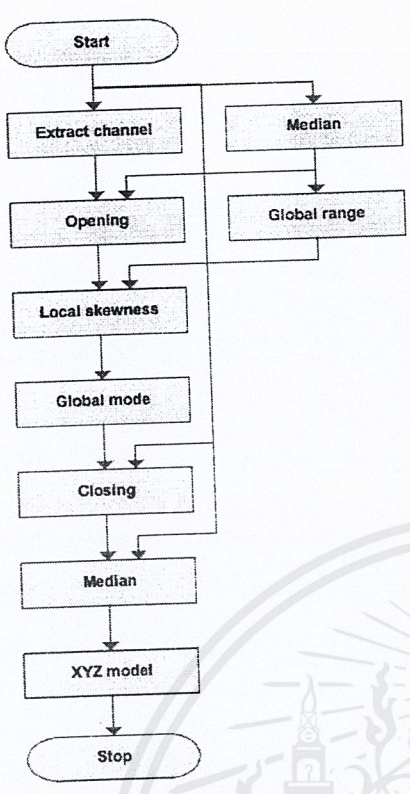


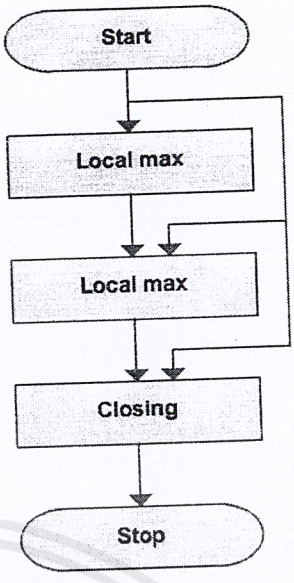
Figure 6.11 Result of weed database

After all accuracy values are collected (Table 6.1), they will be sorted from the best to the worst and be shown in the Fig 6.11 with their ranks result. It shows the original and the ground truth image, that is use for training the program.

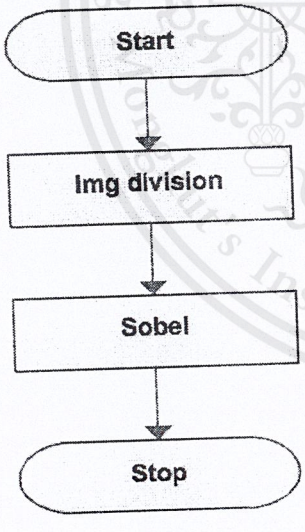
Moreover, the program can generate flowcharts of each feature before sending it to the classifier to process. Each flowchart is shown in the following pages (Figs. 6.12 – Fig 6.14).



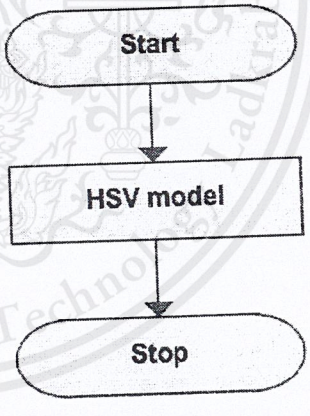
(sub program a)



(sub program b)

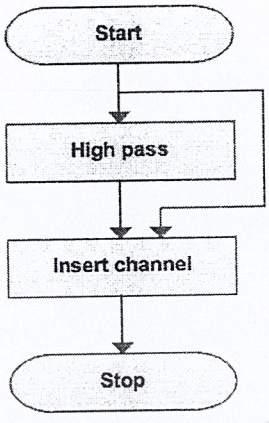


(sub program c)

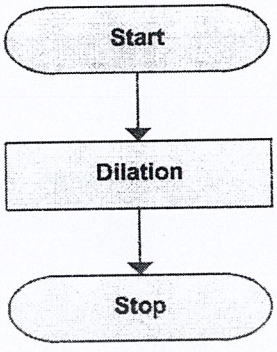


(sub program b)

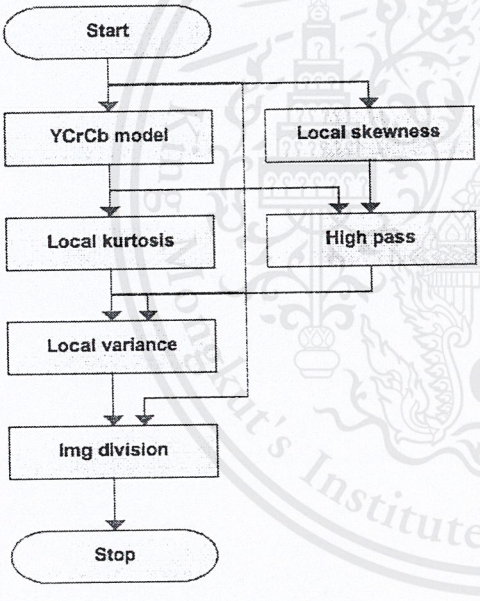
Figure 6.12 Flowchart of best-so-far individual
(no. is obtained from the experiment no.8 (Rank 1))



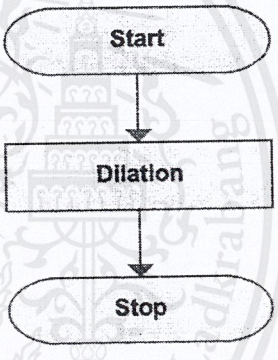
(sub program a)



(sub program b)



(sub program c)



(sub program d)

Figure 6.13 Flowchart of best-so-far individual
(no. is obtained from the experiment no.7 (Rank 2))

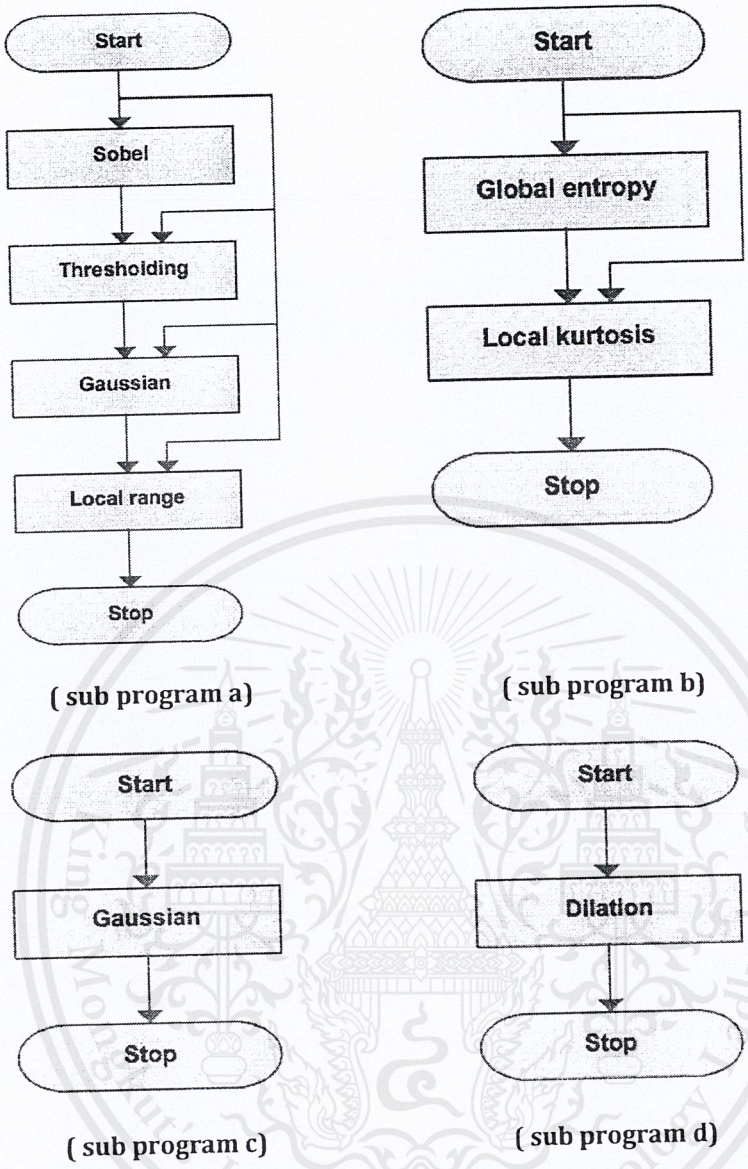


Figure 6.14 Flowchart of best-so-far individual
(no. is obtained from the experiment no.10 (Rank 3))

The Second experimental, the result of generated program from the system 10 experimental, Shown in Table 6.2.The parameters that been set in this operation listed below; Population size = 50, Number of generations =50, Size of individual =10, and Primitive operations 50:

1. Adaptive thresholding
2. Image addition
3. Canny edge detection
4. Image subtraction
5. Global mean
6. Image multiplication
7. Global standard deviation
8. Image division
9. Global variance
10. Local histogram
11. Global skewness
12. Local variance
13. Global kurtosis
14. Local skewness
15. Global maximum
16. Local kurtosis
17. Global minimum
18. Local maximum
19. Global median
20. Local minimum
21. Global mode
22. Local mode
23. Global range
24. Local range
25. Global entropy
26. Local entropy
27. HighPass filter
28. Lowpass filter
29. HighPass laplacin filter
30. Lowpass gaussian filter
31. Histogram equalization
32. Meadian filter
33. Image square root
34. Morphological dilation
35. Image absolute
36. Morphological erosion
37. Image scaling
38. Morphological opening
39. Image negative
40. Morphological closing
41. Sobel operation
42. Scharr operation
43. Thresholding
44. Convert to HSV
45. Convert to XYZ
46. Convert to YCrCb
47. Convert to HLS
48. Convert to Lab
49. Insert Channel
50. Extract Channel

Table 6.2 Flower database

Experimental no.	Accuracy (%)	
1	98.76	#2nd
2	86.57	
3	88.47	
4	92.71	
5	94.73	#3rd
6	98.96	#1st
7	82.87	
8	80.05	
9	93.63	
10	86.83	
AVG	**90.358**	
STD	**6.4345**	
Max	**98.96**	
Min	**80.05**	

In addition, Table 6.2 shows the accuracy of the best and worst of program, including STD and average.

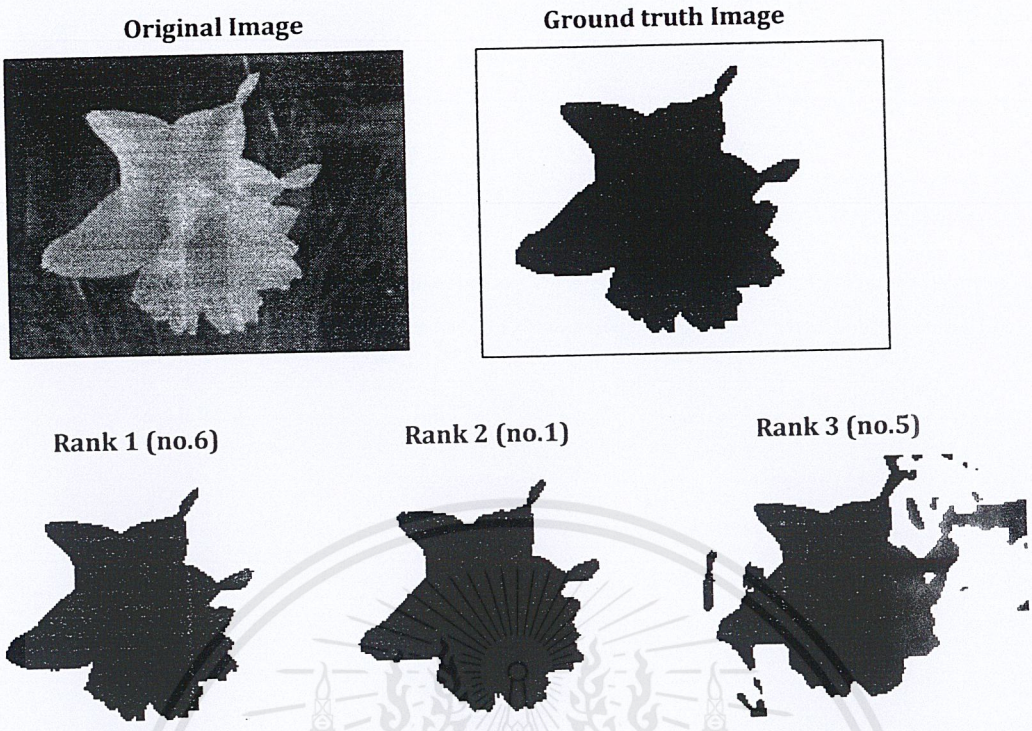
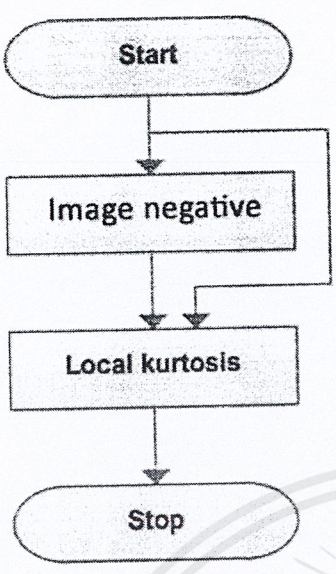


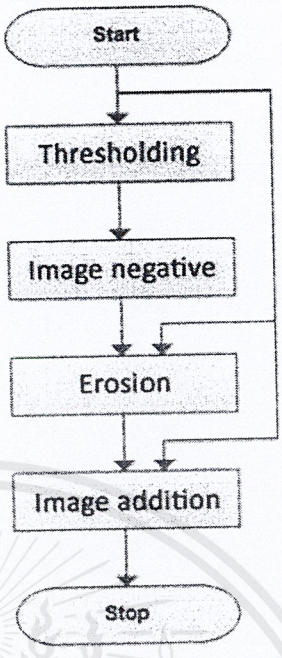
Figure 6.15 Result of flower database

After all accuracy values are collected (Table 6.2), they will be sorted from the best to the worst and be shown in the Fig 6.15 with their rank results. It shows the original and ground truth image, that is use for training the program.

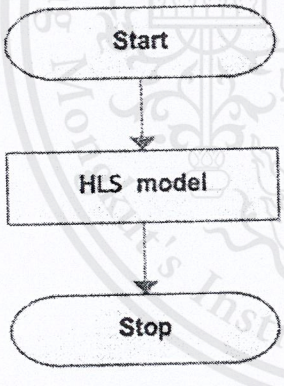
Moreover, the program can generate flowcharts of each feature before sending it to the classifier to process. Each flowchart is shown in following pages(Figs. 6.16-6.18).



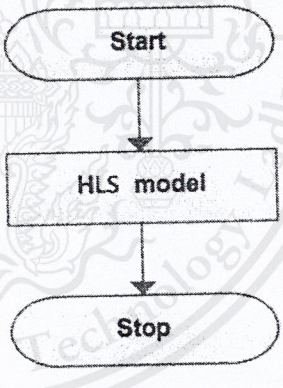
(sub program 1)



(sub program 2)



(sub program 3)



(sub program 4)

Figure 6.16 Flowchart of best-so-far individual
(no. is obtained from the experiment no.6 (Rank 1))

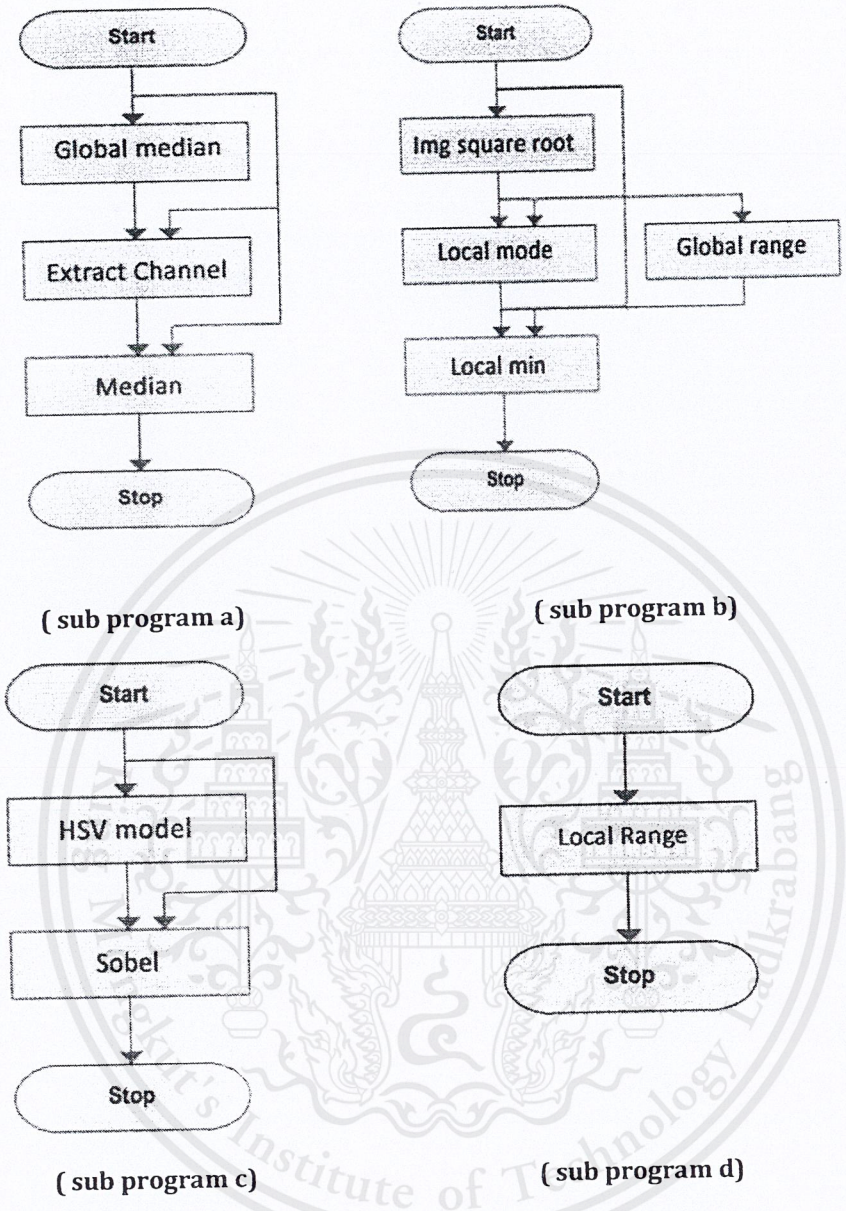


Figure 6.17 Flowchart of best-so-far individual
(no. is obtained from the experiment no.1 (Rank 2))

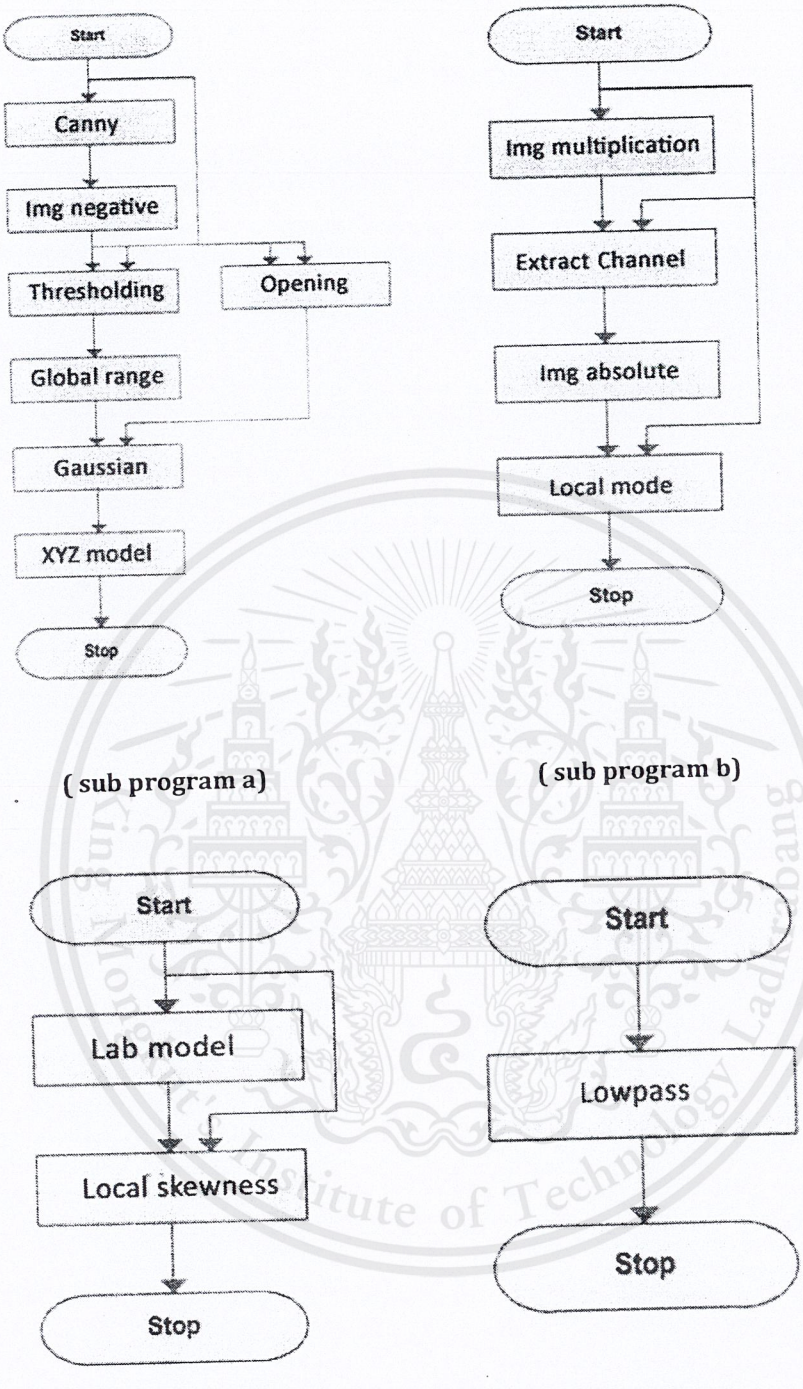


Figure 6.18 Flowchart of best-so-far individual
(no. is obtained from the experiment no.5 (Rank 3))

Chapter 7

Conclusions

7.1 Summary

In this system we use LGP along with classifier. The result of our system is quite impressive. In some experiment, for example, detecting a flower from its background, the system able to find feature extraction program that able to generate 98% precision from such experiment.

In conclusion, from the result of such experiment, we can conclude that our system is good enough to be used in further work or other fields of study.

7.2 Lessons learned

For almost 9 months we have been doing this project, we have learned a lot of lessons which are

- Time Management.
- Teamwork.

7.3 Problems and obstacles

First we have had problems about learning new things such as tools and techniques. We spent a lot of time on research and study. After we got acquainted to them, we have had another problem, which is time management. We don't know exactly how the tools manage its memory. Therefore, we faced memory leak problem and it is very difficult for the system to find and resolve such problem.

Integration between GP system and Graphics User Interface took a lot of time. Initially, because the system takes time to process, we separated the system to work into two threads. Then, we faced a resource sharing and communication problem between threads.

Bibliography

[1] Robert Laganière, OpenCV 2 Computer Vision Application Programming Cookbook, 2011

[2] JasminBlanchette and Mark Summerfield, C++ GUI Programming with Qt 4, 2006

[3] M.Roberts and E.Claridge, "A multistage approach to cooperatively coevolving feature construction and object detection," in: Applications of Evolutionary Computing, Franz Rothlauf et al. (eds.), LNCS 3449, Springer-Verlag, Berlin, pp. 396-406, 2005.

[4] J. Ando and T. Nagao, " Fast tree-structural image processing using GPU," in: Proceedings of IWAIT-2007, Bangkok, Thailand, pp. 423–428 ,2007

[5] S. Shirakawa and T. Nagano, "Genetic image network (GIN): automatically construction of image processing," in: Proceedings of IWAIT-2007, Bangkok, Thailand, pp. 643-348, 2007.

[6] K. Krawiec and B. Bhanu, " Visual learning by evolutionary and coevolutionary feature synthesis," IEEE Transactions on Evolutionary Computation, vol.11, No.5, pp. 635–650, 2007.

[7] U. Watchareeruetai, Y. Takeuchi, T. Matsumoto, H. Kudo, and N. Ohnishi, "Lawn weeds detection methods based on image processing techniques," in: IEICE Technical Report of PRMU2006, vol.106, no.301, pp.65-70, Tokyo, Japan, October 19-20, 2006.

[8] E. Saber, A.M. Tekalp, "Integration of color, edge and texture features for automatic region-based image annotation and retrieval," Electronic Imaging, 7, pp. 684–700, 199

Appendix A

Source Code

CIndividual.h

```
#ifndef CINDIVIDUAL_H
#define CINDIVIDUAL_H

#include "Individual.h"
#include "TypeConverter.h"

using namespace std;

class CIndividual {
private:
        vector<Individual> features;
        int numberOfFeature;
        int maxFeatureSize;
        float fitness;
public:
        void setFlowChartInFeature(int,vector<string>);
        vector<string> getFlowChartFromFeature(int);
        void setMaxFeatureSize(int);
        int getMaxFeatureSize();
        int getNumOfFeature();
        void setNumOfFeature(int);
        void initialCInd();
        float getFitness(void);
        void setFitness(float);
        Individual getFeature(int);
        void setFeature(int,Individual);
};
#endif
```

CIndividual.cpp

```
#include "CIndividual.h"

void CIndividual::initialCInd(){
        fitness = 0;
        Individual ind;
        if(!features.empty()){
                features.clear();
        }
        for(int i = 0 ; i < numberOfFeature ; i++){
                ind.initialInd(maxFeatureSize);
                features.push_back(ind);
        }
}
void CIndividual::setMaxFeatureSize(int featureSize){
        maxFeatureSize = featureSize;
}
void CIndividual::setFlowChartInFeature(int index,vector<string> flowchart){
        features[index].setFlowChart(flowchart);
}
vector<string> CIndividual::getFlowChartFromFeature(int index){
        return features[index].getFlowChart();
}
int CIndividual::getMaxFeatureSize(){
        return maxFeatureSize;
}
int CIndividual::getNumOfFeature(){
        return numberOfFeature;
}
void CIndividual::setNumOfFeature(int numOfF){
        numberOfFeature = numOfF;
}
float CIndividual::getFitness(){
        return fitness;
}
void CIndividual::setFitness(float fitnessVal){
        fitness = fitnessVal;
}
Individual CIndividual::getFeature(int index){
        return features[index];
}
void CIndividual::setFeature(int index,Individual feature){
        features[index] = feature;
}
```

CPopulation.h

```
#ifndef CPOPULATION_H
#define CPOPULATION_H

#include "CIndividual.h"
#include "TypeConverter.h"

using namespace std;

class CPopulation {
private:
        vector<CIndividual> cindivs;
        int populationSize;
        int numberOfFeature;
        int maxFeatureSize;
public:
        CIndividual getCIndiv(int);
        void addCInd(CIndividual);
        void initialCPop();
        void setCPopsize(int);
        int getCPopSize(void);
        void setCInd(int,CIndividual);
        int getCurrentCPopSize();
        int getMaxFeatureSize(void);
        void setMaxFeatureSize(int);
        int getNumberOfFeature();
        void setNumberOfFeature(int);
};
#endif
```

CPopulation.cpp

```
#include "CPopulation.h"

CIndividual CPopulation::getCIndiv(int index){
        return cindivs[index];
}
void CPopulation::addCInd(CIndividual indiv){
        cindivs.push_back(indiv);
}
void CPopulation::initialCPop(){
        CIndividual cind;
        if(!cindivs.empty()){
                cindivs.clear();
        }
        cind.setNumOfFeature(numberOfFeature);
        cind.setMaxFeatureSize(maxFeatureSize);
        for(int i = 0 ; i < populationSize ; i++){
                cind.initialCInd();
                cindivs.push_back(cind);
        }
}
void CPopulation::setCPopsize(int popSize){
        populationSize = popSize;
}
int CPopulation::getCPopSize(){
        return populationSize;
}
void CPopulation::setCInd(int index,CIndividual indiv){
        cindivs[index] = indiv;
}
int CPopulation::getCurrentCPopSize(){
        return cindivs.size();
}
int CPopulation::getMaxFeatureSize(void){
        return maxFeatureSize;
}
void CPopulation::setMaxFeatureSize(int size){
        maxFeatureSize = size;
}
int CPopulation::getNumberOfFeature(){
        return numberOfFeature;
}
void CPopulation::setNumberOfFeature(int numOfFeature){
        numberOfFeature = numOfFeature;
}
```

CReproducer.h

```
#ifndef CREPRODUCER_H
#define CREPRODUCER_H

#include "CPopulation.h"
#include "Reproducer.h"
#include "Population.h"
#include "TypeConverter.h"
#include "Random.h"

using namespace std;

class CReproducer {
private:
        Reproducer rep;
        int tournamentSize;
        CIndividual selection(CPopulation);
        vector<CIndividual> deepCrossover(CPopulation);
        vector<CIndividual> topCrossover(CPopulation);
        CIndividual mutation(CPopulation);
public:
        CReproducer();
        void setTournamentSize(int);
        int getTournamentSize();
        CPopulation reproduce(CPopulation);
};
#endif
```

CReproducer.cpp

```
#include "CReproducer.h"

CReproducer::CReproducer(){
	tournamentSize = 5;
}
void CReproducer::setTournamentSize(int size){
	tournamentSize = size;
}
int CReproducer::getTournamentSize(){
	return tournamentSize;
}
CPopulation CReproducer::reproduce(CPopulation curr){
	int selectedMedthod;
	CPopulation temp;
	temp.setMaxFeatureSize(curr.getMaxFeatureSize());
	temp.setCPopsize(curr.getCPopSize());
	vector<CIndividual> crossoveredCInd;
	while(temp.getCurrentCPopSize()!=curr.getCurrentCPopSize()){
		selectedMedthod = Random::random(3,1);
		if (selectedMedthod==1){
			temp.addCInd(mutation(curr));
		}
		else if (selectedMedthod==2){
			crossoveredCInd = deepCrossover(curr);
			if( temp.getCurrentCPopSize() != curr.getCurrentCPopSize() ){
				temp.addCInd(crossoveredCInd[0]);
			}
			if(temp.getCurrentCPopSize()!=curr.getCurrentCPopSize()){
			temp.addCInd(crossoveredCInd[1]);
			}
		}
		else {
			crossoveredCInd = topCrossover(curr);
			if( temp.getCurrentCPopSize()!=curr.getCurrentCPopSize() ){
				temp.addCInd(crossoveredCInd[0]);
			}
			if(temp.getCurrentCPopSize()!=curr.getCurrentCPopSize()){
			temp.addCInd(crossoveredCInd[1]);
			}
		}
	}
	return temp;
}
CIndividual CReproducer::selection(CPopulation pop){
	CIndividual highestFitnessInd;
	CIndividual temp;
	highestFitnessInd = pop.getCIndiv(Random::random(pop.getCPopSize()));
	for(int i = 0;,i < tournamentSize-1;i++){
		temp = pop.getCIndiv(Random::random(pop.getCPopSize()));
		if (highestFitnessInd.getFitness()<temp.getFitness()){
			highestFitnessInd = temp;
		}
	}
	return highestFitnessInd;
}
vector<CIndividual> CReproducer::deepCrossover(CPopulation pop){
	vector<Individual> temp;
	vector<CIndividual> result;
	result.push_back(selection(pop));
	result.push_back(selection(pop));
	int pos1,pos2;
	pos1 = Random::random(4,0);
	pos2 = Random::random(4,0);
```

```
            temp.push_back(result[0].getFeature(pos1));
            temp.push_back(result[1].getFeature(pos2));
            temp = rep.crossover(temp[0],temp[1],pop.getMaxFeatureSize());
            result[0].setFeature(pos1,temp[0]);
            result[1].setFeature(pos2,temp[1]);
            return result;
    }
    vector<CIndividual> CReproducer::topCrossover(CPopulation pop){
            Individual temp;
            vector<CIndividual> result;
            result.push_back(selection(pop));
            result.push_back(selection(pop));
            int pos1,pos2;
            pos1 = Random::random(4,0);
            pos2 = Random::random(4,0);
            for(int i = 0 ; i < pop.getNumberOfFeature();i++){
                    if(Random::dOn(30)){
                            temp = result[0].getFeature(pos1);
                            result[0].setFeature(pos1,result[1].getFeature(pos2));
                            result[1].setFeature(pos2,temp);
                    }
            }
            return result;
    }
    CIndividual CReproducer::mutation(CPopulation pop){
            Individual temp;
            CIndividual result;
            result = selection(pop);
            for(int i = 0 ; i < pop.getNumberOfFeature();i++){
                    if(Random::dOn(20)){// percent to mutate the individual
                            result.setFeature(i,rep.mutation(result.getFeature(i)));
                    }
            }
            return result;
    }
```

Decoder.h

```
#ifndef DECODER_H
#define DECODER_H

#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include "Individual.h"
#include "TypeConverter.h"

using namespace std;

class Decoder {
private:
        vector<string> operationList ;
public:
        Decoder() {}
        vector<string> decode(const Individual& indiv);
        vector<int> getIndexFromString(string);
};
#endif
```

Decoder.cpp

```
#include "Decoder.h"

vector<string> Decoder::decode(const Individual &indiv)
{
        if(!operationList.empty()){
                operationList.clear();
        }
        Individual ind = indiv;
        string command;
        for (int i = 0 ; i < ind.getIndSize();i++){
                command = "";
                command += integerToString(ind.getIns(i).getOpCode())+" ";
                command += integerToString(ind.getIns(i).getOutput())+" ";
                command += integerToString(ind.getIns(i).getInput1())+" ";
                command += integerToString(ind.getIns(i).getInput2());
                operationList.push_back(command);
        }
        return operationList;
}
vector<int> Decoder::getIndexFromString(string input){
        vector<int> indexNumber;
        string keep;
        int intkeep;
        stringstream str(input) ;
        while(!str.eof()){
                getline(str, keep ,' ');
                stringstream ss(keep);
                ss >> intkeep;
                indexNumber.push_back(intkeep);
        }
        return indexNumber;
}
```

Evaluator.h

```
#ifndef EVALUATOR_H
#define EVALUATOR_H

#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include "Decoder.h"
#include "openCVMethod.h"
#include <opencv2\imgproc\imgproc.hpp>
#include <opencv2\ml\ml.hpp>
#include "CIndividual.h"
#include <algorithm>
#include <iomanip>
#include "TypeConverter.h"

class Evaluator {
private:
        Decoder             decoder;
        vector<cv::Mat>     imgReg;
        vector<float>       numReg;
        vector<cv::Mat>     imgBuffer;
        vector<string>      opCodes;
        cv::Mat             tempImguc;
        cv::Mat             tempImgf;
        vector<string>      trainingSamples,GTtrainingSamples;
        vector<string>      testSamples,GTtestSamples;
        int                 regImgSize;
        int                 regNumSize;
public:
        int                 flag;
        int                 countIndiv;
        int                 countFeature;
        int                 countGen;
        ofstream            outfile;
        string              filename;
        Evaluator(){}
        vector<cv::Mat>     imgProbBuffer;
        vector<cv::Mat>     imgGTBuffer;
        int                 optimizeFilterSize(float);
        vector<cv::Mat>     testImgProbBuffer;
        vector<cv::Mat>     testImgGTBuffer;
        CvMat*              trainDataPtr;
        CvMat*              trainClassesPtr;
        CvMat*              testDataPtr;
        cv::Mat             trainData;
        cv::Mat             trainClasses;
        cv::Mat             testData;
        float               response;
        void                setImgBuffer();
        void                setData(CIndividual&);
        void                setTrainClass();
        float               processIndinCInd();
        vector<string>      optimizeIndiv(vector<string>);
        vector<string>      generateFlowChart(vector<string>);
        void addTrainingSample(string);
        void addTestSample(string);
        void addTrainingGTSample(string);
        void addTestGTSample(string);
        void addOpcode(string);
```

```
        void clearTrainingSample();
        void clearTestSample();
        vector<cv::Mat> getImgBuffer();
        float evaluate( const Individual& indlv );
        string findGTPath(string);
        void initReg ( cv::Mat inputImg );
        void runAnInstr( string instr );
        float calSegmentAcc( Mat& out, Mat& gt );
        int getImgRegSize();
        void setRegSize(int,int);
        int getNumRegSize();
};
#endif
```

Evaluator.cpp

```
#include "Evaluator.h"

void Evaluator::setRegSize(int ImgRegisterSize,int NumRegisterSize){
        regImgSize = ImgRegisterSize;
        regNumSize = NumRegisterSize;
}
void Evaluator::clearTrainingSample(){
        if (!trainingSamples.empty()){
                trainingSamples.clear();
        }
}
void Evaluator::clearTestSample(){
        if (!testSamples.empty()){
                testSamples.clear();
        }
}
void Evaluator::addTrainingSample(string trainingSample){
        trainingSamples.push_back(trainingSample);
}
void Evaluator::addTestSample(string testSample){
        testSamples.push_back(testSample);
}
void Evaluator::addTrainingGTSample(string trainSample){
        GTtrainingSamples.push_back(trainSample);
}
void Evaluator::addTestGTSample(string testGTSample){
        GTtestSamples.push_back(testGTSample);
}
void Evaluator::addOpcode(string opcode){
        opCodes.push_back(opcode);
}
vector<cv::Mat> Evaluator::getImgBuffer(){
        return imgBuffer;
}
float Evaluator::evaluate(const Individual & indiv)
{
        float fitness = 0, sumFitness = 0;
        string gtPath;
        cv::Mat problemImg;
        cv::Mat gtImg;
        vector<string> prog = decoder.decode(indiv);
        if (!imgBuffer.empty()){
                imgBuffer.clear();
        }
        for( int i=0; i < trainingSamples.size(); i++ )
        {
                cv::Mat problemImgLoad = cv::imread( trainingSamples[i] );
                gtPath = findGTPath(trainingSamples[i]);
                cv::Mat gtImgLoad = cv::imread( gtPath);
                initReg( problemImgLoad );
                for( int i=0; i<prog.size(); i++ ){
                        runAnInstr( prog[i] );
                }
                thresholding(imgReg[0],128,imgReg[0]);
                imgBuffer.push_back(imgReg[0]);
                sumFitness += calSegmentAcc( imgReg[0], gtImgLoad );
        }
        fitness = sumFitness/trainingSamples.size();
        return fitness;
}
string Evaluator::findGTPath(string s){
        string gtPath = "";
        for (int i =0 ; i<s.size();i++)
```

```
                if(s[i] == '.'){
                        gtPath +="GT.";
                }
                else{
                        gtPath += s[i];
                }
        }
        return gtPath;
}
void Evaluator::setImgBuffer(){
        if (!imgProbBuffer.empty()){
                imgProbBuffer.clear();
        }
        if (!imgGTBuffer.empty()){
                imgGTBuffer.clear();
        }
        if (!testImgProbBuffer.empty()){
                testImgProbBuffer.clear();
        }
        if (!testImgGTBuffer.empty()){
                testImgGTBuffer.clear();
        }
        for( int i=0; i<trainingSamples.size(); i++ )
        {
                cv::Mat problemImgLoad = cv::imread( trainingSamples[i],1 );
                cv::Mat gtImgLoad = cv::imread( GTtrainingSamples[i],0 );
                imgProbBuffer.push_back(problemImgLoad);
                imgGTBuffer.push_back(gtImgLoad);
        }
        for( int i=0; i<testSamples.size(); i++ )
        {
                cv::Mat problemImgLoad = cv::imread( testSamples[i],1 );
                cv::Mat gtImgLoad = cv::imread( GTtestSamples[i],0 );
                testImgProbBuffer.push_back(problemImgLoad);
                testImgGTBuffer.push_back(gtImgLoad);
        }
}
void Evaluator::setData(CIndividual& indivs){//setTrainData,setTestData
        trainData = cv::Mat( imgProbBuffer[0].rows*imgProbBuffer[0].cols*imgProbBuffer.size(),
indivs.getNumOfFeature(), CV_32FC1 );
        testData = cv::Mat(
testImgProbBuffer[0].rows*testImgProbBuffer[0].cols*testImgProbBuffer.size(), indivs.getNumOfFeature(),
CV_32FC1 );
        int rowCount = 0 ;
        int colCount = 0 ;
        int row = 0 ,col = 0 ;
        int count;
        int checkCount = 0;
        flag = 0;
        filename = "C:/Users/Chaowat/Desktop/Test_2/logfiles/genenation "+
integerToString(countGen) + "individual " + integerToString(countIndiv) + ".txt";
        for( int i=0; i < imgProbBuffer.size(); i++ )
        {
                colCount = 0 ;
                countFeature = 0 ;
                for (int j = 0 ;j < indivs.getNumOfFeature();j++){
                        countFeature += 1 ;
                        if(flag == 0 ){
                                outfile.open(filename.c_str(),ios::app);
                                outfile << "Feature Number:" + integerToString(j+1) << endl;
                                outfile.close();
                        }
                        count=checkCount;
                        vector<string> prog = decoder.decode(indivs.getFeature(j));
                        initReg( imgProbBuffer[i] );
                        prog = optimizeIndiv(prog);
```

```
                        Individual indiv;
                        indiv = indivs.getFeature(j);
                        indiv.setFlowChart(generateFlowChart(prog));
                        indivs.setFeature(j,indiv);
                        for( int k=0; k < prog.size(); k++ ){
                                runAnInstr( prog[k] );
                        }
                        row = 0;
                        for(int c = 0 ; c < imgProbBuffer[0].rows*imgProbBuffer[0].cols ; c++ ){
                                optimizeImgType(imgReg[0],CV_32FC1);
                                trainData.at<float>(count,colCount) = imgReg[0].at<float>(row,col);
                                count ++;
                                col+=1;
                                if(col == imgProbBuffer[0].cols){
                                        row+=1;
                                        col=0;
                                }
                        }
                        colCount += 1;
                        if(colCount==indivs.getNumOfFeature()){
                                checkCount += testImgProbBuffer[0].rows*testImgProbBuffer[0].cols;
                        }
                }
                flag = 1;
        }
        rowCount = 0 ;
        row = 0 ,col = 0 ;
        checkCount = 0;
        for( int i=0; i < testImgProbBuffer.size(); i++ )
        {
                colCount = 0 ;
                for (int j = 0 ;j < indivs.getNumOfFeature();j++){
                        count = checkCount;
                        vector<string> prog = decoder.decode(indivs.getFeature(j));
                        prog = optimizeIndiv(prog);
                        initReg( testImgProbBuffer[i] );
                        for( int k=0; k < prog.size(); k++ ){
                                runAnInstr( prog[k] );
                        }
                        row = 0;
                        for(int c = 0 ; c < testImgProbBuffer[0].rows*testImgProbBuffer[0].cols ; c++ ){
                                optimizeImgType(imgReg[0],CV_32FC1);
                                testData.at<float>(count,colCount) = imgReg[0].at<float>(row,col);
                                count++;
                                col+=1;
                                if(col == testImgProbBuffer[0].cols){
                                        row+=1;
                                        col=0;
                                }
                        }
                        colCount += 1;
                        if(colCount==indivs.getNumOfFeature()){
                                checkCount += testImgProbBuffer[0].rows*testImgProbBuffer[0].cols;
                        }
                }
        }
}
int Evaluator::optimizeFilterSize(float value){
        int result;
        result = int(abs(value))%10;
        result = (result*2)+3;
        return result;
}
void Evaluator::setTrainClass(){
        trainClasses = cv::Mat( imgGTBuffer[0].rows*imgGTBuffer[0].cols*imgGTBuffer.size(), 1,
CV_32FC1 );
```

```
                int count = 0;
                for( int i=0; i < imgGTBuffer.size(); i++ )
                {
                        cv::Mat floatImg;
                        imgGTBuffer[i].convertTo(floatImg,CV_32FC1 );
                        for( int j = 0 ; j < floatImg.rows ; j++ ){
                                for(int k = 0 ; k < floatImg.cols ; k++){
                                        trainClasses.at<float>(count,0) = floatImg.at<float>(j,k);
                                        count++;
                                }
                        }
                }
        }
        float Evaluator::processIndinClnd(){
                cv::Mat sample;
                CvMat cvmatSample;
                cv::Mat picResult;
                cv::Mat test1;
                cv::Mat test2;
                int picPixel =  imgGTBuffer[0].rows*imgGTBuffer[0].cols;
                int index = 0 ;
                float fitness = 0;
                int K = 9 ;
                int row = 0 ;
                int col = 0 ;
                picResult = cv::Mat( imgGTBuffer[0].rows, imgGTBuffer[0].cols, CV_32FC1 );
                sample = cv::Mat( 1 , testData.cols , CV_32FC1 );
                CvNormalBayesClassifier bayes;
                bayes.train(trainData,trainClasses);
                if(!imgBuffer.empty()){
                        imgBuffer.clear();
                }
                for (int i = 0; i < testData.rows;i++){
                        for (int nof = 0 ; nof < testData.cols ; nof++)
                        {
                                sample.at<float>(0,nof) = testData.at<float>(i,nof);
                        }
                        response = bayes.predict(sample);
                        if( i % picPixel == 0 && i !=
0||i+1==imgGTBuffer[0].rows*imgGTBuffer[0].cols*testImgGTBuffer.size()){
                                picResult.convertTo( picResult, CV_8UC1 );
                                fitness += calSegmentAcc(picResult,testImgGTBuffer[index]);
                                imgBuffer.push_back(picResult);
                                index++;
                                picResult.convertTo( picResult, CV_32FC1 );
                                row = 0;
                                col = 0;
                        }
                        picResult.at<float>(row,col) = response;
                        col+=1;
                        if (col == imgGTBuffer[0].cols){
                                row +=1;
                                col = 0;
                        }
                }
                outfile.open(filename.c_str(),ios::app);
                outfile << "fitness value : " << integerToString((fitness/index)*100) << "%" << endl;
                bayes.clear();
                return fitness/index;
}
void Evaluator::initReg( cv::Mat inputImg )
{
        if(!imgReg.empty()){
                imgReg.clear();
        }
        if(numReg.empty()){
```

```
                numReg.clear();
        }
        for(int i = 0 ; i<regNumSize ; i++){
                numReg.push_back(1.0);
        }
        for(int i = 0 ; i<regImgSize ; i++){
                imgReg.push_back(inputImg);
        }
}
vector<string> Evaluator::optimizeIndiv(vector<string> instr){
        vector<string> prog;
        vector<int> imgChecker;
        vector<int> numChecker;
        vector<int> instructionSet;
        vector<string> processes;
        string process = "";
        int maxLevel = 0;
        int nodeCount = 0;
        string opcode;
        int outputIndex;
        int input1Index;
        int input2Index;
        int countInput;
        if(regImgSize > regNumSize){
                imgChecker.push_back(1);
                for (int i =0 ;i < regImgSize -1;i++){
                        imgChecker.push_back(0);
                }
                for (int i =0 ;i < regImgSize;i++){
                        numChecker.push_back(0);
                }
        }
        else{
                imgChecker.push_back(1);
                for (int i = 0 ;i < regNumSize-1;i++){
                        imgChecker.push_back(0);
                }
                for (int i = 0 ;i < regNumSize;i++){
                        numChecker.push_back(0);
                }
        }
        for( int i = instr.size()-1 ; i >= 0 ; i-- ){
                process = "";
                instructionSet = decoder.getIndexFromString(instr[i]);
                opcode = opCodes[instructionSet[0]%opCodes.size()];
                countInput = countFunctionInput(opcode);
                outputIndex = instructionSet[1]%regImgSize;
                input1Index = instructionSet[2]%regImgSize;
                input2Index = instructionSet[3]%regImgSize;
                if(imgChecker[outputIndex]==1){
                        if(countInput == 0){
                                imgChecker[outputIndex] = 0;
                                imgChecker[input1Index] = 1;
                                numChecker[input2Index] = 1;
                                prog.push_back(instr[i]);
                        }
                        else if(countInput == 1){
                                imgChecker[outputIndex] = 0;
                                imgChecker[input1Index] = 1;
                                prog.push_back(instr[i]);
                        }
                        else if(countInput == 2){
                                imgChecker[outputIndex] = 0;
                                imgChecker[input1Index] = 1;
                                imgChecker[input2Index] = 1;
                                prog.push_back(instr[i]);
```

```
                        }
                    }
                    if(numChecker[outputIndex]==1){
                        if(countInput == -1){
                            numChecker[outputIndex] = 0;
                            imgChecker[input1Index] = 1;
                            prog.push_back(instr[i]);
                        }
                    }
            }
            std::reverse(prog.begin(),prog.end());
            return prog;
}
vector<string> Evaluator::generateFlowChart(vector<string> instr){
            vector<string> processes;
            string process;
            vector<int> imgNode;
            vector<int> numNode;
            vector<int> imgLevel;
            vector<int> numLevel;
            vector<int> instructionSet;
            int nodecount;
            int countInput;
            string opcode;
            int outputIndex;
            int input1Index;
            int input2Index;
            nodecount = 0;
            process = "";
            if(regImgSize > regNumSize){
                for (int i =0 ;i < regImgSize;i++){
                    imgNode.push_back(0);
                    numNode.push_back(0);
                    imgLevel.push_back(1);
                    numLevel.push_back(1);
                }
            }
            else{
                for (int i =0 ;i < regNumSize;i++){
                    imgNode.push_back(0);
                    numNode.push_back(0);
                    imgLevel.push_back(1);
                    numLevel.push_back(1);
                }
            }
            for(int i = 0 ; i < instr.size();i++){
                process = "";
                nodecount += 1;
                instructionSet = decoder.getIndexFromString(instr[i]);
                opcode = opCodes[instructionSet[0]%opCodes.size()];
                countInput = countFunctionInput(opcode);
                outputIndex = instructionSet[1]%regImgSize;
                input1Index = instructionSet[2]%regImgSize;
                input2Index = instructionSet[3]%regImgSize;
                if(countInput == 0){
                    process += opcode+",";
                    if(imgLevel[input1Index]>numLevel[input2Index]){
                        process += integerToString(imgLevel[input1Index]) + ",";
                        imgLevel[outputIndex] = imgLevel[input1Index]+1;
                    }
                    else{
                        process += integerToString(numLevel[input2Index]) + ",";
                        imgLevel[outputIndex] = numLevel[input2Index]+1;
                    }
                    process += integerToString(imgNode[input1Index]) + ",";
                    process += integerToString(numNode[input2Index]);
```

```
                imgNode[outputIndex] = nodecount;
                processes.push_back(process);
            }
            else if(countInput == 1){
                process += opcode+",";
                process += integerToString(imgLevel[input1Index]) + ",";
                imgLevel[outputIndex] = imgLevel[input1Index]+1;
                process += integerToString(imgNode[input1Index]) + ",";
                process += "-";
                imgNode[outputIndex] = nodecount;
                processes.push_back(process);
            }
            else if(countInput == -1){
                process += opcode+",";
                process += integerToString(imgLevel[input1Index]) + ",";
                numLevel[outputIndex] = imgLevel[input1Index]+1;
                process += integerToString(imgNode[input1Index]) + ",";
                process += "-";
                numNode[outputIndex] = nodecount;
                processes.push_back(process);
            }
            else if(countInput == 2){
                process += opcode+",";
                if(imgLevel[input1Index]>imgLevel[input2Index]){
                    process += integerToString(imgLevel[input1Index]) + ",";
                    imgLevel[outputIndex] = imgLevel[input1Index]+1;
                }
                else{
                    process += integerToString(imgLevel[input2Index]) + ",";
                    imgLevel[outputIndex] = imgLevel[input2Index]+1;
                }
                process += integerToString(imgNode[input1Index]) + ",";
                process += integerToString(imgNode[input2Index]);
                imgNode[outputIndex] = nodecount;
                processes.push_back(process);
            }
        }
        return processes;
}
void Evaluator::runAnInstr( string instr )
{
        int filterSize;
        vector<int> instructionSet = decoder.getIndexFromString(instr);
        string opcode = opCodes[instructionSet[0]%opCodes.size()];
        int outputIndex = instructionSet[1];
        int input1Index = instructionSet[2];
        int input2Index = instructionSet[3];
        if(flag == 0 ){
            outfile.open(filename.c_str(),ios::app);
            outfile << setw(15) << opcode << " ";
        }
        if( opcode == "highPass"){
            if(flag == 0 ){
            outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
            outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
            outfile << endl;
            outfile.close();
            }
            optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
            optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
            highPass(imgReg[input1Index%regImgSize],imgReg[outputIndex%regImgSize]);
        }
        else if( opcode == "highPass_laplacin"){
            if(flag == 0 ){
            outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
            outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
```

```
            outfile << endl;
            outfile.close();
            }
            optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
            optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
highPass_laplacin(imgReg[input1Index%regImgSize],imgReg[outputIndex%regImgSize]);
}
else if( opcode == "sobelImage"){
            if(flag == 0 ){
            outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
            outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
            outfile << endl;
            outfile.close();
            }
            optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
            optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
            sobelImage(imgReg[input1Index%regImgSize],imgReg[outputIndex%regImgSize]);
}
else if( opcode == "scharrImage"){
            if(flag == 0 ){
            outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
            outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
            outfile << endl;
            outfile.close();
            }
            optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
            optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
            scharrImage(imgReg[input1Index%regImgSize],imgReg[outputIndex%regImgSize]);
}
else if( opcode == "cannyImage"){
            if(flag == 0 ){
            outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
            outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
            outfile << endl;
            outfile.close();
            }
            optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
            optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
            cannyImage(imgReg[input1Index%regImgSize],imgReg[outputIndex%regImgSize]);
}
else if( opcode == "negativeImage"){
            if(flag == 0 ){
            outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
            outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
            outfile << endl;
            outfile.close();
            }
            optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
            optimizeImgType(imgReg[outputIndex%regImgSize],CV_32FC1);
            negativeImage(imgReg[input1Index%regImgSize],imgReg[outputIndex%regImgSize]);
}
else if( opcode == "adaptiveImage"){
            if(flag == 0 ){
            outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
            outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
            outfile << endl;
            outfile.close();
            }
            optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
            optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
            adaptiveImage(imgReg[input1Index%regImgSize],imgReg[outputIndex%regImgSize]);
}
else if( opcode == "histogramImage"){
            if(flag == 0 ){
            outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
            outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
```

```
            outfile << endl;
            outfile.close();
            }
            optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
            optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
            histogramImage(imgReg[input1Index%regImgSize],imgReg[outputIndex%regImgSize]);
}
else if( opcode == "convert2HSV"){
            if(flag == 0 ){
            outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
            outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
            outfile << endl;
            outfile.close();
            }
            optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC3);
            optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC3);
            convert2HSV(imgReg[input1Index%regImgSize],imgReg[outputIndex%regImgSize]);
}
else if( opcode == "convert2XYZ"){
            if(flag == 0 ){
            outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
            outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
            outfile << endl;
            outfile.close();
            }
            optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC3);
            optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC3);
            convert2XYZ(imgReg[input1Index%regImgSize],imgReg[outputIndex%regImgSize]);
}
else if( opcode == "convert2YCrCb"){
            if(flag == 0 ){
            outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
            outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
            outfile << endl;
            outfile.close();
            }
            optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC3);
            optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC3);
            convert2YCrCb(imgReg[input1Index%regImgSize],imgReg[outputIndex%regImgSize]);
}
else if( opcode == "convert2HLS"){
            if(flag == 0 ){
            outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
            outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
            outfile << endl;
            outfile.close();
            }
            optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC3);
            optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC3);
            convert2HLS(imgReg[input1Index%regImgSize],imgReg[outputIndex%regImgSize]);
}
else if( opcode == "convert2Lab"){
            if(flag == 0 ){
            outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
            outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
            outfile << endl;
            outfile.close();
            }
            optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC3);
            optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC3);
            convert2Lab(imgReg[input1Index%regImgSize],imgReg[outputIndex%regImgSize]);
}
else if( opcode == "sqrtImage"){
            if(flag == 0 ){
            outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
            outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
```

```
                outfile << endl;
                outfile.close();
                }
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_32FC1);
                sqrtImage(imgReg[input1Index%regImgSize],imgReg[outputIndex%regImgSize]);
        }
        else if( opcode == "absoluteImage"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_32FC1);
                absoluteImage(imgReg[input1Index%regImgSize],imgReg[outputIndex%regImgSize]);
        }
        else if( opcode == "scalingImage"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_32FC1);
                scalingImage(imgReg[input1Index%regImgSize],imgReg[outputIndex%regImgSize]);
        }
        else if( opcode == "findMean"){
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                numReg[outputIndex%regNumSize]=findMean(imgReg[input1Index%regImgSize]);
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regNumSize << " ";
                outfile << endl;
                outfile.close();
        }
        else if( opcode == "findVariance"){
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                numReg[outputIndex%regNumSize]=findVariance(imgReg[input1Index%regImgSize]);
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regNumSize << " ";
                outfile << endl;
                outfile.close();
                }
        }
        else if( opcode == "findSTD"){
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                numReg[outputIndex%regNumSize]=findSTD(imgReg[input1Index%regImgSize]);
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regNumSize << " ";
                outfile << endl;
                outfile.close();
                }
}
else if( opcode == "findSkewness"){
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                numReg[outputIndex%regNumSize]=findSkewness(imgReg[input1Index%regImgSize]);
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regNumSize << " ";
                outfile << endl;
                outfile.close();
```

```
                }
        }
        else if( opcode == "findKurtosis"){
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                numReg[outputIndex%regNumSize]=findKurtosis(imgReg[input1Index%regImgSize]);
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regNumSize << " ";
                outfile << endl;
                outfile.close();
        }
        else if( opcode == "findMax"){
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                numReg[outputIndex%regNumSize]=findMax(imgReg[input1Index%regImgSize]);
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regNumSize << " ";
                outfile << endl;
                outfile.close();
                }
        }
        else if( opcode == "findMin"){
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                numReg[outputIndex%regNumSize]=findMin(imgReg[input1Index%regImgSize]);
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regNumSize << " ";
                outfile << endl;
                outfile.close();
                }
        }
        else if( opcode == "findMedian"){
                optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
                numReg[outputIndex%regNumSize]=findMedian(imgReg[input1Index%regImgSize]);
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regNumSize << " ";
                outfile << endl;
                outfile.close();
        }
        else if( opcode == "findMode"){
                optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
                numReg[outputIndex%regNumSize]=findMode(imgReg[input1Index%regImgSize]);
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regNumSize << " ";
                outfile << endl;
                outfile.close();
                }
        }
        else if( opcode == "findRange"){
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                numReg[outputIndex%regNumSize]=findRange(imgReg[input1Index%regImgSize]);
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regNumSize << " ";
                outfile << endl;
                outfile.close();
                }
        }
        else if( opcode == "findEntropy"){
                optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
                numReg[outputIndex%regNumSize] = findEntropy(imgReg[input1Index%regImgSize]);
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regNumSize << " ";
                outfile << endl;
                outfile.close();
```

```
                }
        }
        else if( opcode == "addImage"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " input2: "<< setw(3)  << input2Index%regImgSize<< " ";
                outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[input2Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_32FC1);
        addImage(imgReg[input1Index%regImgSize],imgReg[input2Index%regImgSize],imgReg[outputIn
dex%regImgSize]);
        }
        else if( opcode == "subImage"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " input2: "<< setw(3)  << input2Index%regImgSize<< " ";
                outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[input2Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_32FC1);
        subImage(imgReg[input1Index%regImgSize],imgReg[input2Index%regImgSize],imgReg[outputIn
dex%regImgSize]);
        }
        else if( opcode == "mulImage"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " input2: "<< setw(3)  << input2Index%regImgSize<< " ";
                outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[input2Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_32FC1);
        mulImage(imgReg[input1Index%regImgSize],imgReg[input2Index%regImgSize],imgReg[outputIn
dex%regImgSize]);
        }
        else if( opcode == "divImage"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " input2: " << setw(3) << input2Index%regImgSize<< " ";
                outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[input2Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_32FC1);
        divImage(imgReg[input1Index%regImgSize],imgReg[input2Index%regImgSize],imgReg[outputIn
dex%regImgSize]);
        }
        else if( opcode == "lowpassImage"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                filterSize = optimizeFilterSize(numReg[input2Index%regNumSize]);
```

```
                optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
        lowpassImage(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize]);
        }
        else if( opcode == "lowpassGaussian"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                filterSize = optimizeFilterSize(numReg[input2Index%regNumSize]);
                optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
        lowpassGaussian(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize])
;
        }
        else if( opcode == "medianImage"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                filterSize = optimizeFilterSize(numReg[input2Index%regNumSize]);
                optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
    medianImage(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize]);
    }
    else if( opcode == "dilationImage"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                filterSize = optimizeFilterSize(numReg[input2Index%regNumSize]);
                optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
    dilationImage(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize]);
    }
    else if( opcode == "erosionImage"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3) << input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3) << outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                filterSize = optimizeFilterSize(numReg[input2Index%regNumSize]);
                optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
    erosionImage(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize]);
    }
    else if( opcode == "insertChannel"){
                filterSize = ((int)abs(numReg[input2Index%regNumSize]))%3;
                        if(flag == 0 ){
                outfile << "input1: "<< setw(3)<< input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3)<< outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_32FC3);
    insertChannel(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize]);
    }
    else if( opcode == "extractChannel"){
```

```
                filterSize = ((int)abs(numReg[input2Index%regNumSize]))%3;
                if(flag == 0 ){
                outfile << "input1: "<< setw(3)<< input1Index%regImgSize << " ";
                outfile << " output: "<< setw(3)<< outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC3);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
        extractChannel(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize]);
        }
        else if( opcode == "openImage"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3)<< input1Index%regImgSize << " ";
                outfile << " input2: "<< setw(3)<< input2Index%regNumSize << " ";
                outfile << " output: "<< setw(3)<< outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                filterSize = optimizeFilterSize(numReg[input2Index%regNumSize]);
                optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
        openImage(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize]);
        }
        else if( opcode == "closeImage"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3)<< input1Index%regImgSize << " ";
                outfile << " input2: "<< setw(3)<< input2Index%regNumSize << " ";
                outfile << " output: "<< setw(3)<< outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                filterSize = optimizeFilterSize(numReg[input2Index%regNumSize]);
                optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
        closeImage(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize]);
        }
        else if( opcode == "thresholding"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3)<< input1Index%regImgSize << " ";
                outfile << " input2: "<< setw(3)<< input2Index%regNumSize << " ";
                outfile << " output: "<< setw(3)<< outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
        thresholding(imgReg[input1Index%regImgSize],numReg[input2Index%regImgSize],imgReg[outp
utIndex%regImgSize]);
        }
        else if( opcode == "getHistogram"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3)<< input1Index%regImgSize << " ";
                outfile << " input2: "<< setw(3)<< input2Index%regNumSize << " ";
                outfile << " output: "<< setw(3)<< outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                filterSize = optimizeFilterSize(numReg[input2Index%regNumSize]);
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_32FC1);
        getHistogram(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize]);
        }
        else if( opcode == "getVariance"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3)<< input1Index%regImgSize << " ";
```

```
                outfile << " input2: "<< setw(3)<< input2Index%regNumSize << " ";
                outfile << " output: "<< setw(3)<< outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                filterSize = optimizeFilterSize(numReg[input2Index%regNumSize]);
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_32FC1);
        getVariance(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize]);
        }
        else if( opcode == "getSkewness"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3)<< input1Index%regImgSize << " ";
                outfile << " input2: "<< setw(3)<< input2Index%regNumSize << " ";
                outfile << " output: "<< setw(3)<< outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                filterSize = optimizeFilterSize(numReg[input2Index%regNumSize]);
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_32FC1);
        getSkewness(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize]);
        }
        else if( opcode == "getKurtosis"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3)<< input1Index%regImgSize << " ";
                outfile << " input2: "<< setw(3)<< input2Index%regNumSize << " ";
                outfile << " output: "<< setw(3)<< outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                filterSize = optimizeFilterSize(numReg[input2Index%regNumSize]);
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_32FC1);
        getKurtosis(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize]);
        }
        else if( opcode == "getMax"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3)<< input1Index%regImgSize << " ";
                outfile << " input2: "<< setw(3)<< input2Index%regNumSize << " ";
                outfile << " output: "<< setw(3)<< outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                filterSize = optimizeFilterSize(numReg[input2Index%regNumSize]);
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_32FC1);
                getMax(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize]);
        }
        else if( opcode == "getMin"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3)<< input1Index%regImgSize << " ";
                outfile << " input2: "<< setw(3)<< input2Index%regNumSize << " ";
                outfile << " output: "<< setw(3)<< outputIndex%regImgSize << " ";
                outfile << endl;
                outfile.close();
                }
                filterSize = optimizeFilterSize(numReg[input2Index%regNumSize]);
                optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                optimizeImgType(imgReg[outputIndex%regImgSize],CV_32FC1);
                getMin(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize]);
        }
        else if( opcode == "getMode"){
                if(flag == 0 ){
                outfile << "input1: "<< setw(3)<< input1Index%regImgSize << " ";
                outfile << " input2: "<< setw(3)<< input2Index%regNumSize << " ";
```

```
                    outfile << " output: "<< setw(3)<< outputIndex%regImgSize << " ";
                    outfile << endl;
                    outfile.close();
                    }
                    filterSize = optimizeFilterSize(numReg[input2Index%regNumSize]);
                    optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
                    optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
            getMode(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize]);
            }
            else if( opcode == "getRange"){
                    if(flag == 0 ){
                    outfile << "input1: "<< setw(3)<< input1Index%regImgSize << " ";
                    outfile << " input2: "<< setw(3)<< input2Index%regNumSize << " ";
                    outfile << " output: "<< setw(3)<< outputIndex%regImgSize << " ";
                    outfile << endl;
                    outfile.close();
                    }
                    filterSize = optimizeFilterSize(numReg[input2Index%regNumSize]);
                    optimizeImgType(imgReg[input1Index%regImgSize],CV_32FC1);
                    optimizeImgType(imgReg[outputIndex%regImgSize],CV_32FC1);
            getRange(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize]);
            }
            else if( opcode == "getEntropy"){
                    if(flag == 0 ){
                    outfile << "input1: "<< setw(3)<< input1Index%regImgSize << " ";
                    outfile << " input2: "<< setw(3)<< input2Index%regNumSize << " ";
                    outfile << " output: "<< setw(3)<< outputIndex%regImgSize << " ";
                    outfile << endl;
                    outfile.close();
                    }
                    filterSize = optimizeFilterSize(numReg[input2Index%regNumSize]);
                    optimizeImgType(imgReg[input1Index%regImgSize],CV_8UC1);
                    optimizeImgType(imgReg[outputIndex%regImgSize],CV_8UC1);
            getEntropy(imgReg[input1Index%regImgSize],filterSize,imgReg[outputIndex%regImgSize]);
            }
}
float Evaluator::calSegmentAcc(cv::Mat&in ,cv::Mat &gt)
{
        float count = 0 ;
        for(int i=0;i < in.rows;i++){
                for(int j=0;j< in.cols;j++){
                        if (gt.at<uchar>(i,j) == (in.at<uchar>(i,j))){
                        count += 1.0;
                        }
                }
        }
        return count/(in.rows*in.cols);
}
```

Individual.h

```
#ifndef INDIVIDUAL_H
#define INDIVIDUAL_H

#include <iostream>
#include <vector>
#include "InstructionSet.h"

using namespace std;
class Individual{
private :
        vector<InstructionSet> ins;
        int maxSize;
        float fitness;
public :
        Individual(void);
        vector<string> flowchart;
        vector<string> getFlowChart(void);
        void setFlowChart(vector<string>);
        void initialInd(int);
        int getIndSize(void);
        float getFitness(void);
        void setFitness(float);
        InstructionSet getIns(int);
        void setIns(int,InstructionSet);
        void insertIns(int,InstructionSet);
        void addIns(InstructionSet);
        void deleteLastIns(void);
        void deleteIns(int);
};
#endif
```

Individual.cpp

```
#include "Individual.h"
Individual::Individual(){
        fitness = 0.0;
}
float Individual::getFitness(){
        return fitness;
}
vector<string> Individual::getFlowChart(){
        return flowchart;
}
void Individual::setFlowChart(vector<string> fchart){
        flowchart = fchart;
}
void Individual::setFitness(float val){
        fitness = val;
}
InstructionSet Individual::getIns(int index){
        return ins[index];
}
int Individual::getIndSize(){
        return ins.size();
}
void Individual::initialInd(int maxSize){
        if(!ins.empty()){
                ins.clear();
        }
        int size = Random::random(maxSize,1);
        InstructionSet in;
        for(int i = 0 ; i<size;i++){
                in.initialIn();
                ins.push_back(in);
        }
}
void Individual::setIns(int index,InstructionSet in){
        ins[index] = in;
}
void Individual::insertIns(int index,InstructionSet in){
}
void Individual::addIns(InstructionSet in){
        ins.push_back(in);
}
void Individual::deleteIns(int index){
}
void Individual::deleteLastIns(){
        ins.pop_back();
}
```

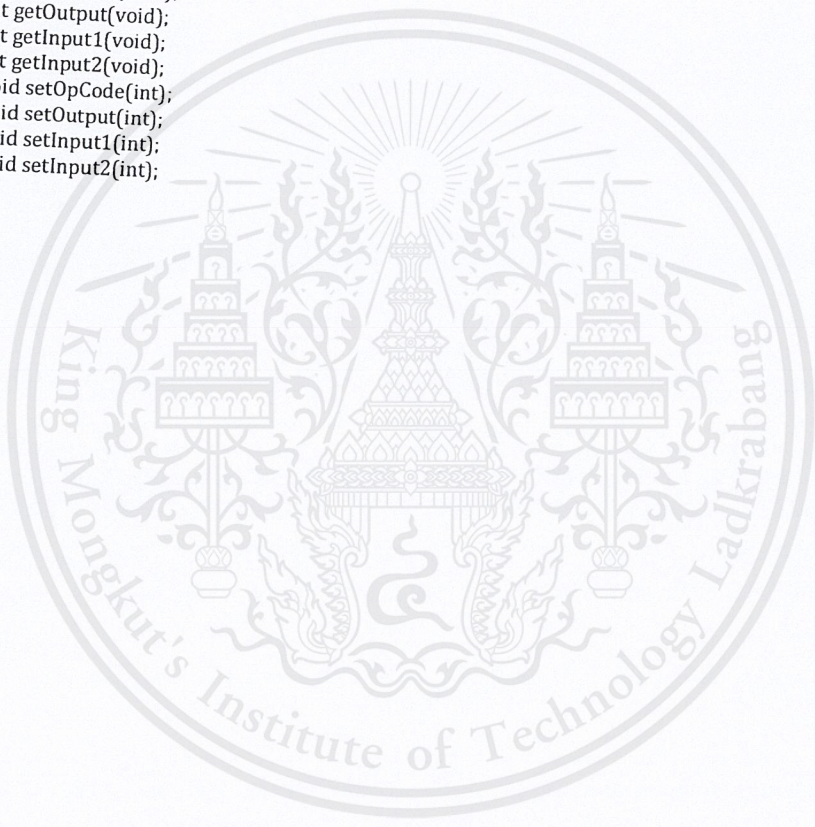
InstructionSet.h

```
#ifndef INSTRUCTIONSET_H
#define INSTRUCTIONSET_H

#include <iostream>
#include "Random.h"

using namespace std;

class InstructionSet{
private :
        int opCode;
        int output;
        int input1;
        int input2;
public :
        InstructionSet(void);
        void initialIn(void);
        int getOpCode(void);
        int getOutput(void);
        int getInput1(void);
        int getInput2(void);
        void setOpCode(int);
        void setOutput(int);
        void setInput1(int);
        void setInput2(int);
};
#endif
```



InstructionSet.cpp

```
#include "InstructionSet.h"

InstructionSet::InstructionSet(){
}
int InstructionSet::getOpCode(){
        return opCode;
}
int InstructionSet::getOutput(){
        return output;
}
int InstructionSet::getInput1(){
        return input1;
}
int InstructionSet::getInput2(){
        return input2;
}
void InstructionSet::setOpCode(int op){
        opCode = op;
}
void InstructionSet::initialIn(){
        opCode = Random::random(100);
        output = Random::random(100);
        input1 = Random::random(100);
        input2 = Random::random(100);
}
void InstructionSet::setOutput(int out){
        output = out;
}
void InstructionSet::setInput1(int in){
        input1 = in;
}
void InstructionSet::setInput2(int in){
        input2 = in;
}
```

OpenCVMethod.h

```
#ifndef OPENCV_METHODS_H
#define OPENCV_METHODS_H

#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/opencv.hpp"
#include <math.h>
#include <iostream>

using namespace cv;
using namespace std;

void highPass(Mat& in, Mat& out);
void highPass_laplacin(Mat& in, Mat& out);
void sobelImage(Mat& in,Mat& out);
void scharrImage(Mat& src,Mat& grad);
void cannyImage(Mat& in,Mat& out);
void negativeImage(Mat& in,Mat& out);
void adaptiveImage(Mat& in,Mat& out);
void histogramImage(Mat& in,Mat& out);
void convert2HSV(Mat& in,Mat& out);
void convert2XYZ(Mat& in,Mat& out);
void convert2YCrCb(Mat& in,Mat& out);
void convert2HLS(Mat& in,Mat& out);
void convert2Lab(Mat& in,Mat& out);
void sqrtImage(Mat& in, Mat& out);
void sqrtImage(Mat& in, Mat& out);
void absoluteImage(Mat& in, Mat& out);
float findMean(Mat& in);
float findVariance(Mat& in);
float findSTD(Mat& in);
float findSkewness(Mat& in);
float findKurtosis(Mat& in);
float findMax(Mat& img);
float findMin(Mat& img);
float findMedian(Mat& img);
float findMode(Mat& img);
float findRange(Mat& in);
float findEntropy(Mat& img);
void scalingImage(Mat& in, Mat& out);
void addImage(Mat& in,Mat& in2,Mat& out);
void subImage(Mat& in,Mat& in2,Mat& out);
void mulImage(Mat& in,Mat& in2,Mat& out);
void divImage(Mat& in,Mat& in2,Mat& out);
void lowpassImage(Mat& in,int val,Mat& out);
void lowpassGaussian(Mat& in,int val,Mat& out);
void medianImage(Mat& in,int val,Mat& out);
void dilationImage(Mat& in,int val,Mat& out);
void erosionImage(Mat& in,int val,Mat& out);
void openImage(Mat& in,int val,Mat& out);
void closeImage(Mat& in,int val,Mat& out);
void thresholding(Mat& in,int val,Mat& out);
void getHistogram(Mat& in,const int windows_size,Mat& out);
void getVariance(Mat& in,const int windows_size,Mat& out);
void getSkewness(Mat& in,const int windows_size,Mat& out);
void getKurtosis(Mat& in,const int windows_size,Mat& out);
void getMax(Mat& in,const int windows_size,Mat& out);
void getMin(Mat& in,const int windows_size,Mat& out);
void getMode(Mat& in,const int windows_size,Mat& out);
void getRange(Mat& in,const int windows_size,Mat& out);
void getEntropy(Mat& in,const int windows_size,Mat& out)
```

```
void insertChannel(Mat& in,int val,Mat& out);
void extractChannel(Mat& in,int val,Mat& out);
int countFunctionInput(string opcode);
void optimizeImgType(Mat& in,int type);
#endif
```

OpenCVMethod.cpp

```
#include "OpenCVMethod.h"

void highPass(Mat& in, Mat& out)
{
        blur(in, out,cvSize(11,11),cvPoint(-1,-1));
}
void highPass_laplacin(Mat& in, Mat& out)
{
        blur(in, out,cvSize(17,17),cvPoint(-1,-1));
}
void sobelImage(Mat& in,Mat& out){
        int scale = 1;
        int delta = 0;
        int ddepth = CV_16S;
        Mat abs_grad_x, abs_grad_y,grad_x, grad_y;
        GaussianBlur(in, in, Size(3,3), 0, 0, BORDER_DEFAULT );
        Sobel( in,grad_x, ddepth, 1, 0, 3, scale, delta, BORDER_DEFAULT );
        convertScaleAbs(grad_x, abs_grad_x );
        Sobel( in,grad_y, ddepth, 0, 1, 3, scale, delta, BORDER_DEFAULT );
        convertScaleAbs( grad_y, abs_grad_y );
        addWeighted( abs_grad_x, 0.5, abs_grad_y, 0.5, 0, out );
}
void scharrImage(Mat& src,Mat& grad){
        int scale = 1;
        int delta = 0;
        int ddepth = CV_16S;
        GaussianBlur( src, src, Size(11,11), 0, 0, BORDER_DEFAULT );
        Mat src_gray = src;
        Mat grad_x, grad_y,Mat, abs_grad_x, abs_grad_y;
        Scharr( src_gray, grad_x, ddepth, 1, 0, scale, delta, BORDER_DEFAULT );
        convertScaleAbs( grad_x, abs_grad_x );
        Scharr( src_gray, grad_y, ddepth, 0, 1, scale, delta, BORDER_DEFAULT );
        convertScaleAbs( grad_y, abs_grad_y );
        addWeighted( abs_grad_x, 0.5, abs_grad_y, 0.5, 0, grad );
}
void cannyImage(Mat& in,Mat& out){
        Canny(in,out,10,100,3);
}
void threshold(Mat& in,Mat& out,int val){
        for (int i=0;i<in.rows;i++){
                for (int j=0;j<in.cols;j++){
                        if(in.at<uchar>(i,j)>val){
                                out.at<uchar>(i,j) = 255;
                        }
                        else{
                                out.at<uchar>(i,j) = 0;
                        }
                }
        }
}
void negativeImage(Mat& in,Mat& out){
        for(int i=0;i< in.rows;i++)
                for(int j=0;j< in.cols;j++)
                        out.at<float>(i,j) = 255 - (in.at<float>(i, j));
}
void adaptiveImage(Mat& in,Mat& out){
        adaptiveThreshold(in, out,255,CV_ADAPTIVE_THRESH_MEAN_C, CV_THRESH_BINARY,75,10);
}
void histogramImage(Mat& in,Mat& out){
        equalizeHist( in, out );
}
void convert2HSV(Mat& in,Mat& out){
        cvtColor(in,out,CV_RGB2HSV);
```

```
}
void convert2XYZ(Mat& in,Mat& out){
        cvtColor(in,out,CV_RGB2XYZ);
}
void convert2YCrCb(Mat& in,Mat& out){
        cvtColor(in,out,CV_RGB2YCrCb);
}
void convert2HLS(Mat& in,Mat& out){
        cvtColor(in,out,CV_RGB2HLS);
}
void convert2Lab(Mat& in,Mat& out){
        cvtColor(in,out,CV_RGB2Lab);
}
void sqrtImage(Mat& in, Mat& out) {
        for(int i=0;i< in.rows;i++)
                for(int j=0;j< in.cols;j++)
                        out.at<float>(i,j) = sqrt( in.at<float>(i,j) );
}
void absoluteImage(Mat& in, Mat& out) {
        for(int i=0;i< in.rows;i++)
                for(int j=0;j< in.cols;j++)
                        out.at<float>(i,j) = abs( in.at<float>(i,j) );
}
float findMean(Mat& in ) {
        float sum = 0;
        for(int i=0;i< in.rows;i++)
                for(int j=0;j< in.cols;j++)
                        sum += in.at<float>(i, j);
        return sum/( in.rows*in.cols );
}
float findVariance(Mat& in) {
        double var= 0;
        double mean = findMean(in);
        for(int i=0;i< in.rows;i++)
                for(int j=0;j< in.cols;j++)
                        var += in.at<float>(i, j)*in.at<float>(i, j);
        return ((float)(var/(in.rows*in.cols)-(mean*mean)));
}
float findSTD(Mat& in) {
        double var = findVariance( in );
        double std = var;
        return (float)sqrt(std);
}
float findSkewness(Mat& in) {
        double mean = findMean( in );
        double var;
        long sum=0;
        for(int i=0;i< in.rows;i++) {
                for(int j=0;j< in.cols;j++){
                        var = in.at<float>(i, j)-mean;
                        sum += (var*var*var);
                }
        }
        return abs((float)-sum/(in.rows*in.cols));
}
float findKurtosis(Mat& in) {
        int i,j;
        float d=0;
        long sum = 0;
        float mean = findMean( in );
        for(i=0;i< in.rows;i++)
                for(j=0;j< in.cols;j++)
                        {
                                d = in.at<float>(i, j)-mean;
                                d=d*d;
                                d=d*d;
```

```
                                        sum += (d);
                                }
                        return ((float)abs(sum/(in.rows*in.cols)));
        }
        float findMax(Mat& img) {
                float max = -10000000;
                for(int i=0;i< img.rows;i++)
                        for(int j=0;j< img.cols;j++)
                                if(img.at<float>(i,j) > max)
                                        max = img.at<float>(i,j);
                return max;
        }
        float findMin(Mat& img) {
                float min = 10000000;
                for(int i=0;i< img.rows;i++)
                        for(int j=0;j< img.cols;j++)
                                if(img.at<float>(i,j) < min)
                                        min = img.at<float>(i,j);
                return min;
        }
        float findMedian(Mat& img) {
                int m;
                int long hist[256];
                int long sum=0;
                const float med_position=(int)(0.5*(img.rows*img.cols));
                for(int i=0;i<256;i++)
                        hist[i]=0;
                for(int i=0;i< img.rows;i++)
                        for(int j=0;j< img.cols;j++)
                                hist[ (int)img.at<uchar>(i,j) ] += 1;
                for(m=0;m<256;m++)
                {
                        sum+=hist[m];
                        if(sum>med_position)
                                break;
                }
                return m;
        }
        float findMode(Mat& img) {
                int long hist[256];
                for(int i=0;i<256;i++)
                        hist[i]=0;
                for(int i=0;i< img.rows;i++)
                        for(int j=0;j< img.cols;j++)
                                hist[ (int)img.at<uchar>(i,j) ] += 1;
                int long max = hist[0];
                int max_index = 0;
                for(int i=1;i<256;i++)
                        if(hist[i] > max)
                        {
                                max = hist[i];
                                max_index = i;
                        }
                return (int)max_index;
}
float findRange(Mat& in) {
        float max = findMax( in );
        float min = findMin( in );
        const float range = max - min;
        return range;
}
float findEntropy(Mat& img) {
        double sum ;
        double hist[256];
        const unsigned long img_size = img.rows*img.cols
                for(int i=0;i<256;i++)
```

```
                hist[i]=0;
                for(int i=0;i< img.rows;i++)
                        for(int j=0;j< img.cols;j++)
                                hist[ (int)img.at<uchar>(i,j) ] += 1;
        sum = 0 ;
        for(int i=0;i<256;i++)
                {
                        hist[i] /= img_size;
                        if(hist[i]>0)
                                sum += hist[i]*cv::log(hist[i]);
                }
        return (int)abs(sum);
}
void scalingImage(Mat& in, Mat& out) {
        float max = findMax( in );
        float min = findMin( in );
        const float max_min = max - min;
        for(int i=0;i< in.rows;i++)
                for(int j=0;j< in.cols;j++)
                        out.at<float>(i,j) = (in.at<float>(i,j) - min)*255.0f/max_min;
}
void addImage(Mat& in,Mat& in2,Mat& out) {
        for(int i=0;i< in.rows;i++)
                for(int j=0;j< in.cols;j++)
                        out.at<float>(i,j) = (in.at<float>(i,j) + in2.at<float>(i,j));
}
void subImage(Mat& in,Mat& in2,Mat& out) {
        for(int i=0;i< in.rows;i++)
                for(int j=0;j< in.cols;j++)
                        out.at<float>(i,j) = abs((in.at<float>(i,j) - in2.at<float>(i,j)));
}
void mulImage(Mat& in,Mat& in2,Mat& out) {
        for(int i=0;i< in.rows;i++)
                for(int j=0;j< in.cols;j++)
                        out.at<float>(i,j) = (in.at<float>(i,j) * in2.at<float>(i,j));
}
void divImage(Mat& in,Mat& in2,Mat& out) {
        for(int i=0;i< in.rows;i++)
                for(int j=0;j< in.cols;j++)
                        out.at<float>(i,j) = (in.at<float>(i,j) / (in2.at<float>(i,j)+1));
}
void lowpassImage(Mat& in,int val,Mat& out) {
        blur(in,out,cv::Size(val,val));
}
void lowpassGaussian(Mat& in,int val,Mat& out) {
        GaussianBlur(in,out,cv::Size(val,val),1.5);
}
void medianImage(Mat& in,int val,Mat& out) {
        medianBlur(in,out,val);
}
void dilationImage(Mat& in,int val,Mat& out) {
        Mat element( val,val,CV_8U,Scalar(1));
        morphologyEx(in,out,cv::MORPH_DILATE,element);
}
void erosionImage(Mat& in,int val,Mat& out) {
        Mat element( val,val,CV_8U,Scalar(1));
        morphologyEx(in,out,cv::MORPH_ERODE,element);
}
void openImage(Mat& in,int val,Mat& out) {
        Mat element( val,val,CV_8U,Scalar(1));
        morphologyEx(in,out,cv::MORPH_OPEN,element);
}
void closeImage(Mat& in,int val,Mat& out) {
        Mat element( val,val,CV_8U,Scalar(1));
        morphologyEx(in,out,cv::MORPH_CLOSE,element);
}
```

```
void thresholding(Mat& in,int val,Mat& out) {
        threshold(in,out,val);
}
Mat img_equalHist( Mat in ){
        Mat out;
        equalizeHist ( in, out );
        return out;
}
Mat img_crop( Mat in, double width, double height ) {
        if ( in.channels ( ) != 1 )
                return in;
        Mat temp;
        for ( int x = 0; x < in.cols - width; x++ ) {
                for ( int y = 0; y < in.rows - height; y++ ) {
                        temp = in ( Rect ( x, y, width, height ) );
                        equalizeHist( temp, temp );
                        temp.copyTo ( in ( Rect ( x, y, width, height ) ) );
                }
        }
        return in;
}
Mat img_resize ( Mat in, double zin ) {
        Mat out;
        out.create ( in.cols / zin, in.rows / zin, CV_8UC(15) );
        resize ( in, out, Size ( in.cols / zin, in.rows / zin ), 0, 0, INTER_LINEAR );
        return out;
}
void getHistogram(Mat& in,const int windows_size,Mat& out) {
        int i,j,x,y;
        double sum = 0;
        float d;
        const int block_size_2 = (windows_size-1)/2;
        const int str_row = block_size_2 ;
        const int end_row = in.rows -(block_size_2);
        const int str_col = block_size_2;
        const int end_col = in.cols - block_size_2;
        in.convertTo( in, CV_8UC1 );
        in = img_crop(in,windows_size,windows_size);
        in.convertTo( in, CV_32FC1 );
        for (i=0;i<out.rows;i++)
                for (j=0;j<out.cols;j++)
                        out.at<float>(i,j)=0;
        for(i=str_row;i<end_row;i++)
                for(j=str_col;j<end_col;j++)
                {
                        out.at<float>(i,j)=in.at<float>(i,j);
                }
        }
void getVariance(Mat& in,const int windows_size,Mat& out) {
        int i,j,x,y;
        double var =0;
        double mean =0;
        double sum = 0;
        float d;
        const int block_size_2 = (windows_size-1)/2;
        const int str_row = block_size_2 ;
        const int end_row = in.rows -(block_size_2);
        const int str_col = block_size_2;
        const int end_col = in.cols - block_size_2;
        for (i=0;i<out.rows;i++)
                for (j=0;j<out.cols;j++)
                        out.at<float>(i,j)=0;
        for(i=str_row;i<end_row;i++)
                for(j=str_col;j<end_col;j++) {
                        sum = 0;
                        for(x=i-block_size_2;x<=i+block_size_2;x++)
```

```
                                        for(y=j-block_size_2;y<=j+block_size_2;y++) {
                                                sum += in.at<float>(x,y);
                                        }
                        mean = sum / (windows_size*windows_size);
                        }
                        for(i=str_row;i<end_row;i++)
                                for(j=str_col;j<end_col;j++){
                                        var = 0;
                                for(x=i-block_size_2;x<=i+block_size_2;x++)
                                        for(y=j-block_size_2;y<=j+block_size_2;y++) {
                                        d = in.at<float>(x,y)-mean;
                                                var += (d*d);
                                }
                                out.at<float>(i,j) =(float)var/(windows_size*windows_size);
                        }
}
void getSkewness(Mat& in,const int windows_size,Mat& out){
        int i,j,x,y;
        double sum=0;
        double mean =0;
        float d;
        const int block_size_2 = (windows_size-1)/2;
        const int str_row = block_size_2 ;
        const int end_row = in.rows -(block_size_2);
        const int str_col = block_size_2;
        const int end_col = in.cols - block_size_2;
        for (i=0;i<out.rows;i++)
                for (j=0;j<out.cols;j++)
                        out.at<float>(i,j)=0;
        for(i=str_row;i<end_row;i++)
                for(j=str_col;j<end_col;j++) {
                        mean =0;
                        sum = 0;
                        for(x=i-block_size_2;x<=i+block_size_2;x++)
                                for(y=j-block_size_2;y<=j+block_size_2;y++) {
                                        sum += in.at<float>(x,y);
                                }
                mean = sum / (windows_size*windows_size);
                }
                        for(i=str_row;i<end_row;i++)
                                for(j=str_col;j<end_col;j++) {
                                        for(x=i-block_size_2;x<=i+block_size_2;x++)
                                                for(y=j-block_size_2;y<=j+block_size_2;y++)
                                        d = in.at<float>(x,y)-mean;
                                        sum += (d*d*d);
                        out.at<float>(i,j)=abs(sum/(windows_size*windows_size));
        }
}
void getKurtosis(Mat& in,const int windows_size,Mat& out) {
        int i,j,x,y,n;
        float  var;
        long sum=0;
        double mean = findMean(in);
        const int block_size_2 = (windows_size-1)/2;
        const int str_row = block_size_2 ;
        const int end_row = in.rows -(block_size_2);
        const int str_col = block_size_2;
        const int end_col = in.cols - block_size_2;
        for (i=0;i<out.rows;i++)
                for (j=0;j<out.cols;j++)
                        out.at<float>(i,j)=0;
        for(i=str_row;i<end_row;i++)
                for(j=str_col;j<end_col;j++)
                {
                        for(x=i-block_size_2;x<=i+block_size_2;x++)
```

```
                                        for(y=j-block_size_2;y<=j+block_size_2;y++) {
                                                var = in.at<float>(x,y)-mean;
                        }                       sum += (var*var*var*var);
                                out.at<float>(i,j)=(float)sum/(in.rows*in.cols);
                }
        }
        void getMax(Mat& in,const int windows_size,Mat& out){
                int i,j,x,y;
                float max;
                const int block_size_2 = (windows_size-1)/2;
                const int str_row = block_size_2 ;
                const int end_row = in.rows -(block_size_2);
                const int str_col = block_size_2;
                const int end_col = in.cols - block_size_2;
                const int fram1 = (str_row*str_col) ;
                const int fram2 =(end_row*end_col);
                for (i=0;i<out.rows;i++)
                        for (j=0;j<out.cols;j++)
                                out.at<float>(i,j)=0;
                for(i=str_row;i<end_row;i++)
                        for(j=str_col;j<end_col;j++) {
                        max=0;
                        for(x=i-block_size_2;x<=i+block_size_2;x++) {
                                for(y=j-block_size_2;y<=j+block_size_2;y++) {
                                        if(in.at<float>(x,y) > max)
                                                max = in.at<float>(x,y);
                                }
                        }
                        out.at<float>(i,j)=max;
        }
}
void getMin(Mat& in,const int windows_size,Mat& out) {
        int i,j,x,y,min;
        const int block_size_2 = (windows_size-1)/2;
        const int str_row = block_size_2 ;
        const int end_row = in.rows -(block_size_2);
        const int str_col = block_size_2;
        const int end_col = in.cols - block_size_2;
        for (i=0;i<out.rows;i++)
                for (j=0;j<out.cols;j++)
                        out.at<float>(i,j)=0;
        for(i=str_row;i<end_row;i++)
                for(j=str_col;j<end_col;j++)
                {
                        min=255;
                        for(x=i-block_size_2;x<=i+block_size_2;x++) {
                                for(y=j-block_size_2;y<=j+block_size_2;y++) {
                                        if(in.at<float>(x,y) < min)
                                                min = in.at<float>(x,y);
                                }
                        }
                        out.at<float>(i,j)=min;
                }
}
void getMode(Mat& in,const int windows_size,Mat& out){
        int i,j,x,y,n;
        int long hist[256];
        int max_index = 0;
        const int block_size_2 = (windows_size-1)/2;
        const int str_row = block_size_2 ;
        const int end_row = in.rows -(block_size_2);
        const int str_col = block_size_2;
        const int end_col = in.cols - block_size_2;
        for (i=0;i<out.rows;i++)
                for (j=0;j<out.cols;j++)
```

```
                                        out.at<uchar>(i,j)=0;
                for(i=str_row;i<end_row;i++)
                        for(j=str_col;j<end_col;j++)
                        {
                                for(n=0;n<256;n++)
                                        hist[n]=0;
                                for(x=i-block_size_2;x<=i+block_size_2;x++)
                                        for(y=j-block_size_2;y<=j+block_size_2;y++)
                                                hist[ (int)in.at<uchar>(x,y) ] += 1;
                                int long max = hist[0];
                                        for(n=0;n<256;n++)
                                                if(hist[n] > max)
                                                {
                                                        max = hist[n];
                                                        max_index =n;
                                                }
                                        out.at<uchar>(i,j) = max_index;
                        }
        }
        void getRange(Mat& in,const int windows_size,Mat& out) {
                int i,j,x,y;
                float max,min ;
                const int block_size_2 = (windows_size-1)/2;
                const int str_row = block_size_2 ;
                const int end_row = in.rows -(block_size_2);
                const int str_col = block_size_2;
                const int end_col = in.cols - block_size_2;
                for (i=0;i<out.rows;i++)
                        for (j=0;j<out.cols;j++)
                                        out.at<float>(i,j)=0;
                for(i=str_row;i<end_row;i++)
                        for(j=str_col;j<end_col;j++)
                        {
                                min=255;
                                max=0;
                                for(x=i-block_size_2;x<=i+block_size_2;x++) {
                                        for(y=j-block_size_2;y<=j+block_size_2;y++){
                                                if(in.at<float>(x,y) < min)
                                                        min = in.at<float>(x,y);
                                                if(in.at<float>(x,y) > max)
                                                        max = in.at<float>(x,y);
                                        }
                                }
                                out.at<float>(i,j)=max-min;
                }
        }
        void insertChannel(Mat& in,int val,Mat& out){
                if( &in == &out ) {
                        if(out.channels() == 1){
                                cvtColor( out, out, CV_GRAY2BGR );
                        }
                        return;
                }
                if( out.channels() == 1 ) {
                        cvtColor( out, out, CV_GRAY2BGR );
                }
                if( in.channels() == 3 ) {
                        cvtColor( in, in, CV_BGR2GRAY );
                }
                for( int i = 0; i < in.rows; ++i )
                        for( int j = 0; j < in.cols; ++j ) {
                                out.at<Vec3f>(i,j)[val] = in.at<float>(i,j);
                        }
}
void extractChannel(Mat& in,int val,Mat& out){
```

```
        if( in.channels() == 1 ) {
                out = in;
                return;
        }
        vector<Mat> channels(3);
        cv::split(in,channels);
        if (val == 0){
                out = channels[0];
        }
        else if(val == 1){
                out = channels[1];
        }
        else if(val == 2){
                out = channels[2];
        }
}
void getEntropy(Mat& in,const int windows_size,Mat& out) {
        int i,j,x,y,n;
        double sum ;
        double hist[256];
        const unsigned long num_size_in_block = windows_size*windows_size;
        const int block_size_2 = (windows_size-1)/2;
        const int str_row = block_size_2 ;
        const int end_row = in.rows -(block_size_2);
        const int str_col = block_size_2;
        const int end_col = in.cols - block_size_2;
        for (i=0;i<out.rows;i++)
                for (j=0;j<out.cols;j++)
                        out.at<uchar>(i,j)=0;
        for(i=str_row;i<end_row;i++)
                for(j=str_col;j<end_col;j++)
                {
                        for(n=0;n<256;n++)
                                hist[n]=0;
                        for(x=i-block_size_2;x<=i+block_size_2;x++)
                                for(y=j-block_size_2;y<=j+block_size_2;y++)
                                        hist[(int)in.at<uchar>(x,y)] += 1;
                        sum = 0 ;
                        for(n=0;n<256;n++)
                        {
                                hist[n]/=num_size_in_block;
                                if(hist[n]>0)
                                        sum+=hist[n]*cv::log(hist[n]);
                        }
                        out.at<uchar>(i,j)=(uchar)-sum;
        }
}
int countFunctionInput(string opcode){
        if( opcode == "highPass"){
                return 1;
        }
        else if( opcode == "highPass_laplacin"){
                return 1;
        }
        else if( opcode == "sobelImage"){
                return 1;
        }
        else if( opcode == "scharrImage"){
                return 1;
        }
        else if( opcode == "cannyImage"){
                return 1;
        }
        else if( opcode == "negativeImage"){
                return 1;
        }
```

```
        else if( opcode == "adaptiveImage"){
                return 1;
        }
        else if( opcode == "histogramImage"){
                return 1;
        }
        else if( opcode == "convert2HSV"){
                return 1;
        }
        else if( opcode == "convert2XYZ"){
                return 1;
        }
        else if( opcode == "convert2YCrCb"){
                return 1;
        }
        else if( opcode == "convert2HLS"){
                return 1;
        }
        else if( opcode == "convert2Lab"){
                return 1;
        }
        else if( opcode == "sqrtImage"){
                return 1;
        }
        else if( opcode == "absoluteImage"){
                return 1;
        }
        else if( opcode == "scalingImage"){
                return 1;
        }
        else if( opcode == "findMean"){
                return -1;
        }
        else if( opcode == "findVariance"){
                return -1;
        }
        else if( opcode == "findSTD"){
                return -1;
        }
        else if( opcode == "findSkewness"){
                return -1;
        }
        else if( opcode == "findKurtosis"){
                return -1;
        }
        else if( opcode == "findMax"){
                return -1;
        }
        else if( opcode == "findMin"){
                return -1;
        }
        else if( opcode == "findMedian"){
                return -1;
        }
        else if( opcode == "findMode"){
                return -1;
        }

        else if( opcode == "findRange"){
                return -1;
        }
        else if( opcode == "findEntropy"){
                return -1;
        }
        else if( opcode == "addImage"){
                return 2;
```

```
        }
        else if( opcode == "subImage"){
                return 2;
        }
        else if( opcode == "mulImage"){
                return 2;
        }
        else if( opcode == "divImage"){
                return 2;
        }
        else if( opcode == "lowpassImage"){
                return 0;
        }
        else if( opcode == "lowpassGaussian"){
                return 0;
        }
        else if( opcode == "medianImage"){
                return 0;
        }
        else if( opcode == "medianImage"){
                return 0;
        }
        else if( opcode == "dilationImage"){
                return 0;
        }
        else if( opcode == "erosionImage"){
                return 0;
        }
        else if( opcode == "openImage"){
                return 0;
        }
        else if( opcode == "closeImage"){
                return 0;
        }
        else if( opcode == "thresholding"){
                return 0;
        }
        else if( opcode == "insertChannel"){
                return 0;
        }
        else if( opcode == "extractChannel"){
                return 0;
        }
        else if( opcode == "getHistogram"){
                return 0;
        }
        else if( opcode == "getVariance"){
                return 0;
        }
        else if( opcode == "getSkewness"){
                return 0;
        }
        else if( opcode == "getKurtosis"){
                return 0;
        }
        else if( opcode == "getMax"){
                return 0;
        }
        else if( opcode == "getMin"){
                return 0;
        }
        else if( opcode == "getMode"){
                return 0;
        }
        else if( opcode == "getRange"){
                return 0;
```

```
            }
            else if( opcode == "getEntropy"){
                return 0;
            }
    }
    void optimizeImgType(Mat& in,int type){
        if(in.type() != type && type == CV_8UC1){
            if(in.type() != CV_32FC1){
                cvtColor(in,in,CV_RGB2GRAY);
            }
            in.convertTo(in,type);
        }
        else if(in.type() != type && type == CV_8UC3){
            if(in.type() !=CV_32FC3){
                cvtColor(in,in,CV_GRAY2RGB);
            }
            in.convertTo(in,type);
        }
        else if(in.type() != type && type == CV_32FC1){
            if(in.type() != CV_8UC1){
                cvtColor(in,in,CV_RGB2GRAY);
            }
            in.convertTo(in,type);
        }
        else if(in.type() != type && type == CV_32FC3){
            if(in.type() !=CV_8UC3){
                cvtColor(in,in,CV_GRAY2RGB);
            }
            in.convertTo(in,type);
        }
}
```

Population.h

```
#ifndef POPULATION_H
#define POPULATION_H

#include <iostream>
#include <vector>
#include "Individual.h"
using namespace std;
class Population{
private :
        vector<Individual> inds;
        int popSize;
        int maxIndivSize;
public :
        Population(void);
        Population(int,int);
        void addInd(Individual);
        void initialPop(void);
        void setPopsize(int);
        int getPopSize(void);
        int getMaxIndivSize(void);
        void setMaxIndivSize(int);
        Individual getInd(int);
        void setInd(int,Individual);
        int getCurrentPopSize();
        void clearPop();
};
#endif
```

Population.cpp

```
#include "Population.h"

Population::Population(){}
Population::Population(int populationSize,int maxIndividualSize){
        popSize = populationSize;
        maxIndivSize = maxIndividualSize;
}
void Population::initialPop(){
        Individual ind;
        if(!inds.empty()){
                inds.clear();
        }
        for(int i = 0 ; i< popSize ;i++){
                ind.initialInd(maxIndivSize);
                inds.push_back(ind);
        }
}
void Population::addInd(Individual indiv){
        inds.push_back(indiv);
}
void Population::clearPop(){
        if(!inds.empty()){
                inds.clear();
        }
}
int Population::getPopSize(){
        return popSize;
}
Individual Population::getInd(int index){
        return inds[index];
}
int Population::getCurrentPopSize(){
        return inds.size();
}
void Population::setInd(int index,Individual ind){
        inds[index] = ind;
}
void Population::setMaxIndivSize(int size){
        maxIndivSize =size;
}
int Population::getMaxIndivSize(){
        return maxIndivSize;
}
void Population::setPopsize(int size){
        popSize = size;
}
```

Random.h

```
#ifndef RANDOM_H
#define RANDOM_H

#include <iostream>
#include <stdlib.h>

using namespace std;

class Random{
public :
        static int random(int,int = 0);
        static bool dOn(int rate);
};
#endif
```

Random.cpp

```
#include "Random.h"

int Random::random(int range,int start ){
        if (range == 0){
                return 0;
        }
        return ( rand() % range ) + start;
}
bool Random::dOn(int rate){
        bool result;
        result = false;
        if(rate > random(100,0)){
                result = true;
        }
        return result;
}
```

Reproducer.h

```
#ifndef REPRODUCER_H
#define REPRODUCER_H

#include "InstructionSet.h"
#include "Individual.h"
#include "Population.h"
#include "Random.h"
#include <vector>

using namespace std;

class Reproducer{
private :
        int tournamentSize;
        Individual selection(Population);
public :
        Reproducer(void);
        void setTournamentSize(int);
        Population reproduce(Population);
        Individual mutation(Individual);
        vector<Individual> crossover(Individual,Individual,int);
};
#endif
```

Reproducer.cpp

```
#include "Reproducer.h"
#include <iostream>
#include <stdlib.h>

Reproducer::Reproducer(){
        tournamentSize = 5;
}
void Reproducer::setTournamentSize(int tourSize){
        tournamentSize = tourSize;
}
Population Reproducer::reproduce(Population pop){
        int selectedMedthod;
        Population temp;
        temp.setMaxIndivSize(pop.getMaxIndivSize());
        temp.setPopsize(pop.getPopSize());
        vector<Individual> crossoveredInd;
        while(temp.getCurrentPopSize()!=pop.getCurrentPopSize()){
                selectedMedthod = Random::random(3,1);
                if (selectedMedthod==1){
                        temp.addInd(mutation(selection(pop)));
                }
                else {
                        crossoveredInd =
crossover(selection(pop),selection(pop),pop.getMaxIndivSize());
                        if( temp.getCurrentPopSize()!=pop.getCurrentPopSize() ){
                                temp.addInd(crossoveredInd[0]);
                        }
                        if(temp.getCurrentPopSize()!=pop.getCurrentPopSize()){
                        temp.addInd(crossoveredInd[1]);
                        }
                }
        }
        return temp;
}
Individual Reproducer::selection(Population pop){
        Individual highestFitnessInd;
        Individual temp;
        highestFitnessInd = pop.getInd(Random::random(pop.getPopSize()));
        for(int i = 0; i < tournamentSize-1;i++){
                temp = pop.getInd(Random::random(pop.getPopSize()));
                if (highestFitnessInd.getFitness()<temp.getFitness()){
                        highestFitnessInd = temp;
                }
        }
        return highestFitnessInd;
}
vector<Individual> Reproducer::crossover(Individual ind1,Individual ind2,int maxIndivSize){
        vector<Individual> parent;
        vector<Individual> child;
        Individual temp;
        int cutPoint1,cutPoint2;
        parent.push_back(ind1);
        parent.push_back(ind2);
        child.push_back(temp);
        child.push_back(temp);
        do{
                cutPoint1 = Random::random(parent[0].getIndSize());
                cutPoint2 = Random::random(parent[1].getIndSize());
        }
        while((cutPoint1 == 0 && cutPoint2 == parent[1].getIndSize()) || (cutPoint2 == 0 && cutPoint1 ==
parent[0].getIndSize()) ||
                cutPoint2 + (parent[0].getIndSize()-cutPoint1) > maxIndivSize || cutPoint1 +
(parent[1].getIndSize()-cutPoint2) > maxIndivSize );
```

```
        for(int i = 0 ; i < cutPoint1;i++){
                        child[0].addIns(parent[0].getIns(i));
        }
        for(int j = cutPoint2 ;j < parent[1].getIndSize(); j++){
                        child[0].addIns(parent[1].getIns(j));
        }
        for(int i = 0 ; i < cutPoint2;i++){
                        child[1].addIns(parent[1].getIns(i));
        }

        for(int j = cutPoint1 ;j < parent[0].getIndSize(); j++){

                        child[1].addIns(parent[0].getIns(j));
        }
        return child;
}
Individual Reproducer::mutation(Individual indiv){
        Individual temp;
        temp = indiv;
        int mutateMethod;
        int possi;
        InstructionSet keep;
        for(int i = 0 ; i<temp.getIndSize();i++){
        possi = Random::random(5);
                if(possi == 0){
                        mutateMethod = Random::random(4);
                        keep = temp.getIns(i);
                        if(mutateMethod == 0){
                                keep.setOpCode(keep.getOpCode()+Random::random(100));
                        }
                        else if(mutateMethod == 1){
                                keep.setOutput(keep.getOutput()+Random::random(100));
                        }
                        else if(mutateMethod == 2){
                                keep.setInput1(keep.getInput1()+Random::random(100));
                        }
                        else if(mutateMethod == 3){
                                keep.setInput2(keep.getInput2()+Random::random(100));
                        }
                        temp.setIns(i,keep);
                }
        }
        return temp;
}
```

TypeConverter.h

```
#include <sstream>
#include <string>
#include <iostream>
#include <vector>

using namespace std;

vector<string> split(string);
string integerToString(int);
int stringToInteger(string);
```

TypeConverter.cpp

```
#include "TypeConverter.h"
string integerToString(int val){
int Number = val;
string Result;
ostringstream convert;
convert << Number;
Result = convert.str();
return Result;
}
int stringToInteger(string val){
string str = val ;
int numb;
istringstream ( str ) >> numb;
return numb;
}
vector<string> split(string input){
        vector<string> value;
        string keep;
        string strkeep;
        stringstream str(input) ;
        while(!str.eof()){
                getline(str, keep ,',');
                stringstream ss(keep);
                ss >> strkeep;
                value.push_back(strkeep);
        }
        return value;
}
```

GUI_mainMenu.h

```
#ifndef MAIN_PAGE
#define MAIN_PAGE

#include <vector>
#include "ImageVariable.hpp"
using namespace std;

class MyThread : public QThread{
        Q_OBJECT
public:
        explicit MyThread(ImageVariable* CV_Variable);
        void run();
private:
        ImageVariable* CV_Variable;
signals:
        void complete();
        void flowchartChanged();
        void logfileChanged(QString);
        void graphChanged();
        void outputChanged(QStringList);
};
#endif
```

GUI_mainMenu.cpp

```
#include "GUI_mainMenu.h"
#include <stdlib.h>
#include <time.h>
#include "CReproducer.h"
#include "CPopulation.h"
#include "CIndividual.h"
#include "Evaluator.h"
#include "TypeConverter.h"
#include "Reproducer.h"
#include "Evaluator.h"
#include "Individual.h"

MyThread::MyThread(ImageVariable* CV_Variable) {
        this->CV_Variable = CV_Variable;
}
void MyThread::run(){
        emit graphChanged();
        srand((unsigned int)time(NULL));
        CIndividual cind,bestSoFar;
        CPopulation cpop;
        Individual feature;
        Evaluator evaluator;
        string imgPath;
        vector<string> imgPaths;
        QStringList QimgPaths;
        vector<cv::Mat> imgBuffer;
        int flag = 0;
        for(int op = 0; op < CV_Variable->cv_primitiveOperations.size();op++){
                evaluator.addOpcode(CV_Variable->cv_primitiveOperations[op].toStdString());
        }
        for(int i = 0 ; i < CV_Variable->cv_trainingSetPBImagePart.size() ; i++ ){
                evaluator.addTrainingSample(CV_Variable-
>cv_trainingSetPBImagePart[i].toStdString());
                evaluator.addTrainingGTSample(CV_Variable-
>cv_trainingSetGTImagePart[i].toStdString());
        }
        for(int i = 0 ; i < CV_Variable->cv_testSetPBImagePart.size() ; i++) {
                evaluator.addTestSample(CV_Variable->cv_testSetPBImagePart[i].toStdString());
                evaluator.addTestGTSample(CV_Variable->cv_testSetGTImagePart[i].toStdString());
        }
        evaluator.setRegSize(CV_Variable->cv_numberOfImageRegisters.toInt(),CV_Variable-
>cv_numberOfNumericalRegisters.toInt());
        evaluator.setImgBuffer();
        cpop.setCPopsize(CV_Variable->cv_populationSize.toInt());
        cpop.setNumberOfFeature(CV_Variable->cv_numberOfFeatures.toInt());
        cpop.setMaxFeatureSize(CV_Variable->cv_maxLength.toInt());
        cpop.initialCPop();
        CReproducer reproducer;
        int numberOfEvoledPrograms = 0;
        float desiredFitnessVal = CV_Variable->cv_fitnessValue.toInt()/100;
        int maxGen = CV_Variable->cv_numberOfGeneration.toInt();
        reproducer.setTournamentSize(CV_Variable->cv_tournamentSize.toInt());
        float fitVal;
        evaluator.setTrainClass();
        bestSoFar.setFitness(0);
        float averageFitVal;
        float bestSoFarFitVal;
        vector<string> test;
        float result = 0;
        vector<QString> flowchart;
        vector<vector<QString>> flowcharts;
        evaluator.countFeature = 0;
        evaluator.countIndiv = 0;
```

```
evaluator.countGen = 0;
for(int i = 0 ; i<maxGen ; i++){
        fitVal = 0 ;
        evaluator.countGen += 1;
        evaluator.countIndiv = 0;
        for(int j = 0 ;j<cpop.getCPopSize() ; j++){
                evaluator.countIndiv += 1;
                cind = cpop.getCIndiv(j);
                numberOfEvoledPrograms += 1 ;
                evaluator.setData(cind);
                result = evaluator.processIndinCInd();
                cind.setFitness(result);
                cpop.setCInd(j,cind);
                fitVal += cind.getFitness();
                if(cind.getFitness()>bestSoFar.getFitness()||bestSoFar.getFitness()==0){
                        bestSoFar = cind;
                        imgBuffer = evaluator.getImgBuffer();
                        if(!imgPaths.empty()){
                                imgPaths.clear();
                        }
                        if(flag == 0){
                                for (int imgCount = 0 ; imgCount <
imgBuffer.size();imgCount++){

        imwrite("C:/Users/Chaowat/Desktop/Test_2/Best/Result"+integerToString(imgCount+1)+".bmp
",imgBuffer[imgCount]);
                                        imgPath =
"C:/Users/Chaowat/Desktop/Test_2/Best/Result"+integerToString(imgCount+1)+".bmp";
                                        imgPaths.push_back(imgPath);
                                }
                                flag = 1;
                        }
                        else if(flag == 1){
                                for (int imgCount = 0 ; imgCount <
imgBuffer.size();imgCount++){
        imwrite("C:/Users/Chaowat/Desktop/Test_2/Best/Results"+integerToString(imgCount+1)+".bm
p",imgBuffer[imgCount]);
                                        imgPath =
"C:/Users/Chaowat/Desktop/Test_2/Best/Results"+integerToString(imgCount+1)+".bmp";
                                        imgPaths.push_back(imgPath);
                                }
                                flag = 0;
                        }
                        if(!QimgPaths.empty()){
                                QimgPaths.clear();
                        }
                        for(int count = 0 ; count < imgBuffer.size();count++){
                                QimgPaths.push_back(QString(imgPaths[count].c_str()));
                        }
                        emit outputChanged(QimgPaths);
                        if(!flowcharts.empty()){
                                flowcharts.clear();
                        }
                        for(int k = 0 ; k < cind.getNumOfFeature() ; k++){
                                if(!flowchart.empty()){
                                        flowchart.clear();
                                }
                                for(int m = 0 ; m <
bestSoFar.getFlowChartFromFeature(k).size();m++){
        flowchart.push_back(QString(bestSoFar.getFlowChartFromFeature(k)[m].c_str()));
                                }
                                flowcharts.push_back(flowchart);
                        }
                        CV_Variable->flowChartProtocol = flowcharts;
                        emit flowchartChanged();
                }
```

```
                }
                bestSoFarFitVal = bestSoFar.getFitness();
                averageFitVal = fitVal/cpop.getCPopSize();
                CV_Variable->bestSoFarFitnessVar.push_back(bestSoFarFitVal);
                CV_Variable->avarageFitnessVar.push_back(averageFitVal);

                emit graphChanged();
                cpop = reproducer.reproduce(cpop);
                cpop.setNumberOfFeature(CV_Variable->cv_numberOfFeatures.toInt());
        }
        emit complete();
}
```

GUI_Chart.h

```
#ifndef CHART
#define CHART

#include <QtWidgets/QWidget>
#include "qwidget.h"
#include "ImageVariable.hpp"
#include <vector>

using namespace std;

class Chart : public QWidget
{
  Q_OBJECT
public:
        Chart(ImageVariable* CV_Variable, QWidget *parent = 0);
        void drawChart(void);
private:
        ImageVariable* CV_Variable;
        void drawAxis();
        void drawPointChart(int generationNum,float fitnessNum);
        void drawBestSoFarFitnessLine(vector<float> bestSoFarFitnessVar,float maxFitness);
        void drawAverageFitnessLine(vector<float> averageFitnessVar,float maxFitness);
        void showExample();
        float findMaxFitness();
protected:
  void paintEvent(QPaintEvent *event);
};
#endif
```

GUI_Chart.cpp

```
#include "GUI_Chart.h"
#include "qpainter.h"
#include "qstring.h"
#include "qwidget.h"
#include <vector>
#include <stdio.h>
#include <math.h>

using namespace std;

Chart::Chart(ImageVariable* CV_Variable, QWidget *parent)
	: QWidget(parent)
{
	this->CV_Variable = CV_Variable;
}
void Chart::paintEvent(QPaintEvent *e)
{
	drawChart();
	update();
}
void Chart::drawChart(){
	float maxFitness=findMaxFitness();
	showExample();
	drawAxis();
	drawPointChart(CV_Variable->bestSoFarFitnessVar.size(),maxFitness);
	drawBestSoFarFitnessLine(CV_Variable->bestSoFarFitnessVar,maxFitness);
	drawAverageFitnessLine(CV_Variable->avarageFitnessVar,maxFitness);
}
float Chart::findMaxFitness()
{
	float maxFitness=0 ;
	for(float i=0;i<CV_Variable->bestSoFarFitnessVar.size();i++){
		if(maxFitness < CV_Variable->bestSoFarFitnessVar[i]){
			maxFitness = CV_Variable->bestSoFarFitnessVar[i];
		}
	}
	for(float i=0;i<CV_Variable->avarageFitnessVar.size();i++){
		if(maxFitness < CV_Variable->avarageFitnessVar[i]){
			maxFitness = CV_Variable->avarageFitnessVar[i];
		}
	}
	return maxFitness;
}
void Chart::showExample(){
	QPainter box_bestSoFar(this);
	QPixmap pix;
	QRectF target(550, 20, 80, 50);
	QRectF source(0.0, 0.0, 0, 0);
	pix.load("C:/Users/Chaowat/Documents/Visual Studio
2010/Projects/NewImageRecognition/NewImageRecognition/System Images/Graph.jpg");
	box_bestSoFar.drawPixmap(target,pix,source);
}
void Chart::drawAxis()
{
	QPainter x_axis_line(this);
	QPainter y_axis_line(this);
	y_axis_line.setPen(QPen(Qt::black,2,Qt::SolidLine,Qt::SquareCap,Qt::MiterJoin));
	y_axis_line.drawText(QPoint(25,15),"Fitness");
	y_axis_line.drawLine(40,250,40,20);
	y_axis_line.drawLine(40,20,45,30);
	y_axis_line.drawLine(40,20,35,30);
	x_axis_line.setPen(QPen(Qt::black,2,Qt::SolidLine,Qt::SquareCap,Qt::MiterJoin));
	x_axis_line.drawText(QPoint(555,253),"Genneration");
```

```
        x_axis_line.drawLine(40,250,550,250);
        x_axis_line.drawLine(550,250,540,245);
        x_axis_line.drawLine(550,250,540,255);
}
void Chart::drawPointChart(int generationNum,float fitnessNum){
        QPainter point(this);
        float generation_num = generationNum;
        float fitness_num = fitnessNum;
        float text_genneration;
        float point_genneration = 500/generation_num;
        QString str_genneration;
        point.setPen(QPen(Qt::red,5,Qt::SolidLine,Qt::SquareCap,Qt::MiterJoin));
        for(float i=0,j=0;i<=generation_num;i++){
                if(i==0){
                        point.drawPoint(40,250);
                }
                else{
                        if(generation_num>=10){
                                text_genneration = (generation_num/10)*i;
                                point.drawPoint(40+50+j,250);
                                point.drawText(QPoint((40+50+j)-
5,265),str_genneration.number((int)text_genneration));
                                j=50+j;
                                if(i==10){
                                        break;
                                }
                        }
                        else{
                                point.drawPoint(40+point_genneration+j,250);
                                point.drawText(QPoint((50+point_genneration+j)-
5,265),str_genneration.number(i));
                                j=point_genneration+j;
                        }
                }
        }
        QString str_fitness;
        float point_fitness = 220/fitness_num;
        float text_fitness;
        float cal_fitness2;
        QPainter y_axis(this);
        y_axis.setPen(QPen(Qt::red,5,Qt::SolidLine,Qt::SquareCap,Qt::MiterJoin));
        if((fitness_num<=1)){
                if(fitnessNum>0){
                        y_axis.drawPoint(40,30);
y_axis.drawText(QPoint(10,30),str_genneration.number(fitnessNum));
                }
                cal_fitness2 = fitnessNum/5;
                for(float i=0,j=0;i<=4;i++){
                        if(fitnessNum>0){
                                y_axis.drawPoint(40,295-44-j);
                                y_axis.drawText(QPoint(10,(295-50-
j)+5),str_fitness.number(cal_fitness2*i));
                        }
                        j=44+j;
                }
        }
        for(float i=0,j=0;i<=fitness_num;i++){
                if(i==0){
                        y_axis.drawPoint(40,250);
                        y_axis.drawText(QPoint(10,250),"0");
                }
                else{
                        if(fitness_num>5){
                                text_fitness = (fitness_num/5)*i;
                                y_axis.drawPoint(40,250-44-j);
```

```
                                        y_axis.drawText(QPoint(10,(250-50-
 j)+5),str_fitness.number(text_fitness));
                                        j=44+j;
                                        if(i==5){
                                                break;
                                        }
                                }
                                else if(fitness_num<=5){
                                        y_axis.drawPoint(40,250-point_fitness-j);
                                        y_axis.drawText(QPoint(10,(250-point_fitness-
 j)),str_fitness.number(i));
                                        j=point_fitness+j;
                                }
                        }
                }
}
void Chart::drawBestSoFarFitnessLine(vector<float> bestSoFarFitnessVar,float maxFitness)
{
        QPainter bestSoFar_Line(this);
        bestSoFar_Line.setPen(QPen(Qt::blue,1,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        int generation = bestSoFarFitnessVar.size();
        vector <float>sum_y;
        vector <float>sum_x;
        for(int i=0;i<generation;i++){
                float cal_x_1 = 0;
                float cal_x_2 = 0;
                cal_x_1=((i+1)/1.0/generation)*100.00;
                cal_x_2=cal_x_1*5;
                sum_x.push_back(cal_x_2+40);
        }
        for(int i=0;i<generation;i++){
                float cal_1 = 0;
                float cal_2 = 0;
                cal_1=(bestSoFarFitnessVar[i]/maxFitness)*100.00;
                cal_2=cal_1*2.2;
                sum_y.push_back(220-cal_2+30);
        }
        for(int i=0;i<generation;i++){
                if(i==0){
                        bestSoFar_Line.drawLine(40,250,sum_x[i],sum_y[i]);
                }
                else{
                        bestSoFar_Line.drawLine(sum_x[i-1],sum_y[i-1],sum_x[i],sum_y[i]);
                }
        }
}
void Chart::drawAverageFitnessLine(vector<float> avarageFitnessVar,float maxFitness)
{
        QPainter average_Line(this);
        average_Line.setPen(QPen(Qt::red,1,Qt::SolidLine,Qt::SquareCap,Qt::MiterJoin));
        int generation = avarageFitnessVar.size();
        vector <float>sum_y;
        vector <float>sum_x;
        for(int i=0;i<generation;i++){
                float cal_x_1 = 0;
                float cal_x_2 = 0;
                cal_x_1=((i+1)/1.0/generation)*100.00;
                cal_x_2=cal_x_1*5;
                sum_x.push_back(cal_x_2+40);
        }
        for(int i=0;i<generation;i++){
                float cal_1 = 0;
                float cal_2 = 0;
                cal_1=(avarageFitnessVar[i]/maxFitness)*100.00;
                cal_2=cal_1*2.2;
                sum_y.push_back(220-cal_2+30);
```

```
        }
        for(int i=0;i<generation;i++){
                if(i==0){
                        average_Line.drawLine(40,250,sum_x[i],sum_y[i]);
                }
                else{
                        average_Line.drawLine(sum_x[i-1],sum_y[i-1],sum_x[i],sum_y[i]);
                }
        }
}
```

GUI_FlowChart.h

```
#ifndef FLOW_CHART
#define FLOW_CHART
#include <QtWidgets/QMainWindow>
#include <qtcore>
#include "qwidget.h"
#include "qstring.h"
#include "ImageVariable.hpp"
#include <vector>
#include <stdexcept>

using namespace std;
class FlowChart : public QWidget
{
  Q_OBJECT
public:
FlowChart( ImageVariable* CV_Variable, int hahaha=0, QWidget *parent = 0);
        ~FlowChart();
        vector<QString> cut_protocol(QString temp);
        vector<QString> protocol_with_start;
        void find_protocol(void);
        void draw_symbols(void);
        void find_x_y(void);
        void draw_line(int x_start,int y_start,int x_end,int y_end,int color);
        void draw_arrow(int x_point,int y_point,int angle,int color);
        void cut_text(vector<QString> flowchart_Text);
        void cal_line(void);
        void draw_path(int beg_x_pos, int beg_y_pos, int end_x_pos, int end_y_pos,int color);
        vector<int> level_x;
        vector<int> level_y;
        vector<QString> flowchart_Text;
        vector<QString> flowchart_Text_Cut;
        vector<int> flowchartDepth;
        vector<QString> flowchart_Depth;
        vector<QString> flowchart_Input_1;
        vector<QString> flowchart_Input_2;
        int maxDepth;
        QString find_max();
protected:
  void paintEvent(QPaintEvent *event);
private:
        ImageVariable* CV_Variable;
        int hahaha;
};
#endif
```

GUI_FlowChart.cpp

```
#include <QtGui>
#include <QtWidgets/QMainWindow>
#include "GUI_FlowChart.h"

vector<vector<QString>> flowchartProtocol;
vector<int> amount;

using namespace std;

FlowChart::FlowChart(ImageVariable* CV_Variable, int hahaha,QWidget *parent)
  : QWidget(parent)
{
        this->CV_Variable = CV_Variable;
        this->hahaha = hahaha;
}
FlowChart::~FlowChart(){
}
void FlowChart::paintEvent(QPaintEvent *e)
{
        flowchart_Text.clear();
        flowchart_Depth.clear();
        flowchart_Input_1.clear();
        flowchart_Input_2.clear();
        level_x.clear();
        level_y.clear();
        flowchart_Text_Cut.clear();
        flowchartDepth.clear();
        protocol_with_start.clear();
        flowchart_Text.push_back("Start");
        flowchart_Depth.push_back("0");
        flowchart_Input_1.push_back("-");
        flowchart_Input_2.push_back("-");
        find_protocol();
        find_x_y();
        cut_text(flowchart_Text);
        draw_symbols();
        cal_line();
}
void FlowChart::find_x_y()
{
        int temp = 0;
        level_y.clear();
        level_x.clear();
        for (int i=0; i<flowchart_Depth.size(); ++i)
                level_y.push_back(flowchart_Depth[i].toInt());
        for (int i=0; i<level_y.size(); ++i){
                temp = 0;
                for (int j=i-1; j>=0; --j){
                        if (level_y[j] == level_y[i]){
                                temp = level_x[j] + 1;
                                goto out;
                        }
                }
out:;
                level_x.push_back(temp);
        }
}
void FlowChart::draw_symbols(){
        QPainter drawSymbols;
        drawSymbols.begin(this);
        drawSymbols.setPen(QPen(Qt::black,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        QFont font("Helvetica",18,QFont::Normal);
        drawSymbols.setFont(font);
```

```
            for(int i=0;i<level_x.size();i++){
                    int count_text = flowchart_Text_Cut[i].size();
                    if(i==0){
            drawSymbols.drawRoundedRect(10+(250*level_x[i]),10+(90*level_y[i]),200,45,15,15);
                            drawSymbols.drawText(QPoint(10+((200/count_text-
  1)+(250*level_x[i])),10+(25+(90*level_y[i]))),flowchart_Text_Cut[i]);
                    }
                    else if(i==level_x.size()-1){
            drawSymbols.drawRoundedRect(10+(250*level_x[i]),10+(90*level_y[i]),200,45,15,15);
                            drawSymbols.drawText(QPoint(10+((200/count_text-
  1)+(250*level_x[i])),10+(25+(90*level_y[i]))),flowchart_Text_Cut[i]);
                    }
                    else{
            drawSymbols.drawRoundedRect(10+(250*level_x[i]),10+(90*level_y[i]),200,45,0,0);
                            drawSymbols.drawText(QPoint(10+((200/count_text-
  1)+(250*level_x[i])),10+(25+(90*level_y[i]))),flowchart_Text_Cut[i]);
                    }
            }
}
void FlowChart::draw_line(int x_start,int y_start,int x_end,int y_end,int color){
            QPainter drawFlowChartLine;
            drawFlowChartLine.begin(this);
            if(color==1){
            drawFlowChartLine.setPen(QPen(Qt::black,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
            }
            else if(color==2){
                    drawFlowChartLine.setPen(QPen(Qt::cyan,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
            }
            else if(color==3){
            drawFlowChartLine.setPen(QPen(Qt::darkCyan,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
            }
            else if(color==4){
                    drawFlowChartLine.setPen(QPen(Qt::red,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
            }
            else if(color==5){
            drawFlowChartLine.setPen(QPen(Qt::darkRed,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
            }
            else if(color==6){
            drawFlowChartLine.setPen(QPen(Qt::magenta,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
            }
            else if(color==7){
            drawFlowChartLine.setPen(QPen(Qt::darkMagenta,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
            }
            else if(color==8){
            drawFlowChartLine.setPen(QPen(Qt::green,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
            }
            else if(color==9){
            drawFlowChartLine.setPen(QPen(Qt::darkGreen,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
            }
            else if(color==10){
            drawFlowChartLine.setPen(QPen(Qt::yellow,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
            }
            else if(color==11){
            drawFlowChartLine.setPen(QPen(Qt::darkYellow,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
            }
            else if(color==12){
                    drawFlowChartLine.setPen(QPen(Qt::blue,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
            }
            else if(color==13){
            drawFlowChartLine.setPen(QPen(Qt::darkBlue,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
            }
            else if(color==14){
                    drawFlowChartLine.setPen(QPen(Qt::gray,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
            }
            else if(color==15){
            drawFlowChartLine.setPen(QPen(Qt::darkGray,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
```

```
        }
        else if(color==16){
        drawFlowChartLine.setPen(QPen(Qt::white,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        }
        drawFlowChartLine.drawLine(x_start,y_start,x_end,y_end);
}
void FlowChart::draw_arrow(int x_point,int y_point,int angle,int color){
        QPainter drawArrow(this);
        if(color==1){
                drawArrow.setPen(QPen(Qt::black,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        }
        else if(color==2){
                drawArrow.setPen(QPen(Qt::cyan,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        }
        else if(color==3){
                drawArrow.setPen(QPen(Qt::darkCyan,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        }
        else if(color==4){
                drawArrow.setPen(QPen(Qt::red,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        }
        else if(color==5){
                drawArrow.setPen(QPen(Qt::darkRed,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        }
        else if(color==6){
                drawArrow.setPen(QPen(Qt::magenta,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        }
        else if(color==7){
        drawArrow.setPen(QPen(Qt::darkMagenta,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        }
        else if(color==8){
                drawArrow.setPen(QPen(Qt::green,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        }
        else if(color==9){
                drawArrow.setPen(QPen(Qt::darkGreen,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        }
        else if(color==10){
                drawArrow.setPen(QPen(Qt::yellow,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        }
        else if(color==11){
                drawArrow.setPen(QPen(Qt::darkYellow,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        }
        else if(color==12){
                drawArrow.setPen(QPen(Qt::blue,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        }
        else if(color==13){
                drawArrow.setPen(QPen(Qt::darkBlue,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        }
        else if(color==14){
                drawArrow.setPen(QPen(Qt::gray,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        }
        else if(color==15){
                drawArrow.setPen(QPen(Qt::darkGray,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        }
        else if(color==16){
                drawArrow.setPen(QPen(Qt::white,3,Qt::SolidLine,Qt::RoundCap,Qt::RoundJoin));
        }
        QPolygon polygon(3);
        if(angle==2){
                polygon.putPoints(0, 3, x_point,y_point, x_point+5,y_point-5, x_point+5,y_point+5);
        }
        else if(angle==1){
                polygon.putPoints(0, 3, x_point,y_point, x_point-5,y_point+5, x_point+5,y_point+5);
        }
        else if(angle==3){
                polygon.putPoints(0, 3, x_point,y_point, x_point-5,y_point-5, x_point+5,y_point-5);
        }
```

```
        else if(angle==0){
                polygon.putPoints(0, 3, x_point,y_point, x_point-5,y_point+5, x_point-5,y_point-5);
        }
        drawArrow.drawPolygon(polygon);
}
void FlowChart::cut_text(vector<QString> flowchart_Text){
        flowchart_Text_Cut.push_back("Start");
        for(int i=0;i<flowchart_Text.size();i++){
                if(flowchart_Text[i]=="highPass"){
                        flowchart_Text_Cut.push_back("HighPass");
                }
                if(flowchart_Text[i]=="highPass_laplacin"){
                        flowchart_Text_Cut.push_back("Laplacian");
                }
                if(flowchart_Text[i]=="sobelImage"){
                        flowchart_Text_Cut.push_back("Sobel");
                }
                if(flowchart_Text[i]=="scharrImage"){
                        flowchart_Text_Cut.push_back("Scharr");
                }
                if(flowchart_Text[i]=="cannyImage"){
                        flowchart_Text_Cut.push_back("Canny");
                }
                if(flowchart_Text[i]=="negativeImage"){
                        flowchart_Text_Cut.push_back("Img negative");
                }
                if(flowchart_Text[i]=="adaptiveImage"){
                        flowchart_Text_Cut.push_back("Adaptive Th");
                }
                if(flowchart_Text[i]=="histogramImage"){
                        flowchart_Text_Cut.push_back("Histogram");
                }
                if(flowchart_Text[i]=="convert2HSV"){
                        flowchart_Text_Cut.push_back("HSV model");
                }
                if(flowchart_Text[i]=="convert2XYZ"){
                        flowchart_Text_Cut.push_back("XYZ model");
                }
                if(flowchart_Text[i]=="convert2YCrCb"){
                        flowchart_Text_Cut.push_back("YCrCb model");
                }
                if(flowchart_Text[i]=="convert2HLS"){
                        flowchart_Text_Cut.push_back("HLS model");
                }
                if(flowchart_Text[i]=="convert2Lab"){
                        flowchart_Text_Cut.push_back("Lab model");
                }
                if(flowchart_Text[i]=="sqrtImage"){
                        flowchart_Text_Cut.push_back("Img square root");
                }
                if(flowchart_Text[i]=="absoluteImage"){
                        flowchart_Text_Cut.push_back("Img absolute");
                }
                if(flowchart_Text[i]=="scalingImage"){
                        flowchart_Text_Cut.push_back("Img scaling");
                }
                if(flowchart_Text[i]=="findMean"){
                        flowchart_Text_Cut.push_back("Global mean");
                }
                if(flowchart_Text[i]=="findSTD"){
                        flowchart_Text_Cut.push_back("Global SD");
                }
                if(flowchart_Text[i]=="findVariance"){
                        flowchart_Text_Cut.push_back("Global variance");
                }
                if(flowchart_Text[i]=="findSkewness"){
```

```
                flowchart_Text_Cut.push_back("Global skewness");
        }
        if(flowchart_Text[i]=="findKurtosis"){
                flowchart_Text_Cut.push_back("Global kurtosis");
        }
        if(flowchart_Text[i]=="findMax"){
                flowchart_Text_Cut.push_back("Global max");
        }
        if(flowchart_Text[i]=="findMin"){
                flowchart_Text_Cut.push_back("Global min");
        }
        if(flowchart_Text[i]=="findMedian"){
                flowchart_Text_Cut.push_back("Global median");
        }
        if(flowchart_Text[i]=="findMode"){
                flowchart_Text_Cut.push_back("Global mode");
        }
        if(flowchart_Text[i]=="findRange"){
                flowchart_Text_Cut.push_back("Global range");
        }
        if(flowchart_Text[i]=="findEntropy"){
                flowchart_Text_Cut.push_back("Global entropy");
        }
        if(flowchart_Text[i]=="addImage"){
                flowchart_Text_Cut.push_back("Img addition");
        }
        if(flowchart_Text[i]=="subImage"){
                flowchart_Text_Cut.push_back("Img subtraction");
        }
        if(flowchart_Text[i]=="mulImage"){
                flowchart_Text_Cut.push_back("Img multiplication");
        }
        if(flowchart_Text[i]=="divImage"){
                flowchart_Text_Cut.push_back("Img division");
        }
        if(flowchart_Text[i]=="lowpassImage"){
                flowchart_Text_Cut.push_back("Lowpass");
        }
        if(flowchart_Text[i]=="lowpassGaussian"){
                flowchart_Text_Cut.push_back("Gaussian");
        }
        if(flowchart_Text[i]=="medianImage"){
                flowchart_Text_Cut.push_back("Median");
        }
        if(flowchart_Text[i]=="dilationImage"){
                flowchart_Text_Cut.push_back("Dilation");
        }
        if(flowchart_Text[i]=="erosionImage"){
                flowchart_Text_Cut.push_back("Erosion");
        }
        if(flowchart_Text[i]=="openImage"){
                flowchart_Text_Cut.push_back("Opening");
        }
        if(flowchart_Text[i]=="closeImage"){
                flowchart_Text_Cut.push_back("Closing");
        }
        if(flowchart_Text[i]=="thresholding"){
                flowchart_Text_Cut.push_back("Thresholding");
        }
        if(flowchart_Text[i]=="getHistogram"){
                flowchart_Text_Cut.push_back("Local histogram");
        }
        if(flowchart_Text[i]=="getVariance"){
                flowchart_Text_Cut.push_back("Local variance");
        }
        if(flowchart_Text[i]=="getSkewness"){
```

```
                        flowchart_Text_Cut.push_back("Local skewness");
                    }
                    if(flowchart_Text[i]=="getKurtosis"){
                        flowchart_Text_Cut.push_back("Local kurtosis");
                    }
                    if(flowchart_Text[i]=="getMax"){
                        flowchart_Text_Cut.push_back("Local Max");
                    }
                    if(flowchart_Text[i]=="getMin"){
                        flowchart_Text_Cut.push_back("Local Min");
                    }
                    if(flowchart_Text[i]=="getMode"){
                        flowchart_Text_Cut.push_back("Local Mode");
                    }
                    if(flowchart_Text[i]=="getRange"){
                        flowchart_Text_Cut.push_back("Local Range");
                    }
                    if(flowchart_Text[i]=="getEntropy"){
                        flowchart_Text_Cut.push_back("Local Entropy");
                    }
                    if(flowchart_Text[i]=="insertChannel"){
                        flowchart_Text_Cut.push_back("Insert Channel");
                    }
                    if(flowchart_Text[i]=="extractChannel"){
                        flowchart_Text_Cut.push_back("Extract Channel");
                    }
                }
                flowchart_Text_Cut.push_back("Stop");
        }
        void FlowChart::find_protocol()
        {
                vector<QString> keep_protocol;
                for(int i=0;i<CV_Variable->flowChartProtocol[hahaha].size();i++){
                        keep_protocol=cut_protocol(CV_Variable->flowChartProtocol[hahaha][i]);
                        flowchart_Text.push_back(keep_protocol[0]);
                        flowchart_Depth.push_back(keep_protocol[1]);
                        flowchart_Input_1.push_back(keep_protocol[2]);
                        flowchart_Input_2.push_back(keep_protocol[3]);
                }
                flowchart_Text.push_back("Stop");
                int a=find_max().toInt()+1;
                ostringstream outstr;
                outstr << a;
                flowchart_Depth.push_back(QString(outstr.str().c_str()));
                a = CV_Variable->flowChartProtocol[hahaha].size();
                outstr.str("");
                outstr << a;
                flowchart_Input_1.push_back(QString(outstr.str().c_str()));
                flowchart_Input_2.push_back(QString("-"));
        }
        void FlowChart::cal_line(){
                int beg_x_pos,beg_y_pos,end_x_pos,end_y_pos;
                for (int i=1; i<flowchart_Text.size(); ++i){
                        beg_x_pos = level_x[flowchart_Input_1[i].toInt()];
                        beg_y_pos = level_y[flowchart_Input_1[i].toInt()];
                        end_x_pos = level_x[i];
                        end_y_pos = level_y[i];
                        draw_path(beg_x_pos, beg_y_pos, end_x_pos, end_y_pos,i);
                        if (flowchart_Input_2[i] == QString("-") || flowchart_Input_2[i] == flowchart_Input_1[i])
                                continue;
                        beg_x_pos = level_x[flowchart_Input_2[i].toInt()];
                        beg_y_pos = level_y[flowchart_Input_2[i].toInt()];
                        draw_path(beg_x_pos, beg_y_pos, end_x_pos, end_y_pos,i);
                }
        }
        QString FlowChart::find_max(){
```

```
            int max=0;
            for(int i=0;i<flowchart_Depth.size();i++){
                if(flowchart_Depth[i].toInt()>flowchart_Depth[max].toInt()){
                    max = i;
                }
            }
            return flowchart_Depth[max];
    }
    vector<QString> FlowChart::cut_protocol(QString temp){
            string input = temp.toStdString();
            istringstream instr(input);
            string text;
            getline(instr,text,',');
            string dep;
            getline(instr,dep,',');
            string input1;
            getline(instr,input1,',');
            string input2;
            getline(instr,input2);
            vector<QString> ans;
            ans.push_back(QString(text.c_str()));
            ans.push_back(QString(dep.c_str()));
            ans.push_back(QString(input1.c_str()));
            ans.push_back(QString(input2.c_str()));
            return ans;
}
void FlowChart::draw_path(int beg_x_pos, int beg_y_pos, int end_x_pos, int end_y_pos,int color){
        double beg_x_co,beg_y_co,end_x_co,end_y_co;
        double angle_x1_co,angle_y1_co,angle_x2_co,angle_y2_co;
        double angle_x3_co,angle_y3_co,angle_x4_co,angle_y4_co;
        beg_x_co = 110 + beg_x_pos * 245;
        end_x_co = 110 + end_x_pos * 245;
        beg_y_co = beg_y_pos * 90 + 55;
        end_y_co = end_y_pos * 90 + 10;
        angle_x1_co = beg_x_co;
        angle_y1_co = beg_y_co + 22.5;
        angle_x4_co = end_x_co;
        angle_y4_co = end_y_co - 22.5;
        if (end_y_pos - beg_y_pos != 1){
            if (beg_x_pos <= end_x_pos)
                angle_x2_co = angle_x1_co + 122.5;
            else
                angle_x2_co = angle_x1_co - 122.5;
            angle_x3_co = angle_x2_co;
        }
        else{
            angle_x2_co = angle_x1_co;
            angle_x3_co = angle_x4_co;
        }
        angle_y2_co = angle_y1_co;
        angle_y3_co = angle_y4_co;
        draw_line(beg_x_co,beg_y_co,angle_x1_co,angle_y1_co,color);
        draw_line(angle_x1_co,angle_y1_co,angle_x2_co,angle_y2_co,color);
        draw_line(angle_x2_co,angle_y2_co,angle_x3_co,angle_y3_co,color);
        draw_line(angle_x3_co,angle_y3_co,angle_x4_co,angle_y4_co,color);
        draw_line(angle_x4_co,angle_y4_co,end_x_co,end_y_co,color);
        draw_arrow(end_x_co,end_y_co,3,color);
}
```