Neural Network Guided Evolution of L-system Plants

Xuhao (Eric) Chen

Submitted in partial fulfilment
of the requirements for the degree of

Master of Science

Faculty of Mathematics and Science
Department of Computer Science
Brock University, St. Catharines, Ontario

Abstract

A Lindenmayer system is a parallel rewriting system that generates graphic shapes using several rules. Genetic programming (GP) is an evolutionary algorithm that evolves expressions. A convolutional neural network(CNN) is a type of neural network which is useful for image recognition and classification. The goal of this thesis will be to generate different styles of L-system based 2D images of trees from scratch using genetic programming. The system will use a convolutional neural network to evaluate the trees and produce a fitness value for genetic programming. Different architectures of CNN are explored. We analyze the performance of the system and show the capabilities of the combination of CNN and GP. We show that a variety of interesting tree images can be automatically evolved. We also found that the success of the system highly depends on CNN training, as well as the form of the GP's L-system language representation.

Acknowledgements

The worldwide spreading COVID-19 has changed our lives in many different ways. Writing a thesis during a pandemic is not an easy task. Since everything is online, it becomes harder to keep in contact with others. But thankfully, many people still have been around, supporting me to resolve all issues during these special days.

I want to offer my most sincere thanks to my supervisor Prof. Brian Ross for all his support. He used his professional knowledge to provide me with priceless advice and make sure I was healthy during the pandemic, both physically and mentally. It is impossible for me to write this thesis without the help of him. I will never forget his kindness and helpfulness in my life.

I would like to say special thanks to Cale Fairchild for resolving all the CUDA server issues, which allowed me to train my network faster. Thanks to Illya Bakurov, Justin Maltese, and Tyler Cowan for their writings and tutorial of ECJ, which saved me lots of headaches. Thanks to the authors of ECJ, TensorFlow, CUDA, and Flask frameworks for their fantastic works that saved me a lot of time in programming.

I would like to thank Brock University for financial support. All works require a longer time to finish during the pandemic. With the university's additional funding, I could have extra time to wrap up my work without rushing.

Thanks to everyone in my family, supporting me physically, mentally, and financially. I would like to say special thanks to my dad, Jianxiong, my mom, Xurong. You believed in me, gave me hope, and taught me the skill I cannot learn from school. Even though you have been far away from me, I always know you are there.

I would love to express my special thanks to my significant other, Wenting, for accompanying me in Canada, especially during the pandemic. You have always been here for me these years, supporting me with your time, advice, and decisions, and always creating an atmosphere that makes me feel comfortable. I am glad that I met you, and I believe that we will have more amazing adventures in the future.

The pandemic might change many things, and some of them might be serious and irrevocable. But with all of you next to me, I still believe tomorrow will be better.

Eric Chen

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Artificial intelligence (AI) is growing rapidly during the recent decade. People use AI for risk control, disease analysis, automatic driving, etc. In art, AI has been applied to various applications. Evolutionary algorithms (EAs), which is a branch of AI, have proven to be creative for art [17, 48]. Bentley and Corne introduced several EAs for art, music, architecture and design [17]. Neufeld *et al.* proposed a multi-objective optimization method for genetic programming to evolve artistic image filters [48]. Baluja *et al.* proposed a system that generates images using genetic algorithms based on the learning of user preferences using an artificial neural network [16]. However, they also have some limitations. Lopez discussed computational creativity in art and music, and found computers can be limited by several pre-determined rules [43].

A Lindenmayer system (L-system) is a parallel rewriting system introduced by Aristid Lindenmayer [51]. L-systems are useful for generating graphical models of plant images using symbolic expressions. Each L-system expression contains two major components: the axiom and the production rules. The axiom is the beginning of the rewriting, and the production rules are the methods that are used to rewrite the axiom. Number of iterations defines how many times the system will rewrite. It will rewrite the symbols in the axiom in parallel in each iteration until the number of iterations has reached. Then the expression will be rendered using turtle graphics [57]. Since the L-system is very sensitive to the change on expression, it is almost impossible to know what the final rendering will look like just from the L-system expression.

One evolutionary algorithm that has been applied to L-system synthesis is genetic programming (GP) [35]. GP is a type of evolutionary algorithm that evolves programs to solve certain problems. Previous researchers have used GP and L-systems for creative art in several different ways. Congdon and Mazza proposed a system that using genetic algorithm and 3D L-system that requires human interactions. Bonfim and

Castro proposed a hybrid evolutionary algorithm with L-system evolving individuals from several pre-determined expressions [20]. Jacob developed an GP-based 3D L-system that using interpretation functions for fitness evaluation [32]. Bergen and Ross proposed a multi-objective GP to create aesthetic 3D models using 3D L-system [18].

A difficulty in using GP in some L-system applications, for example, for evolving trees and plants models, is finding an appropriate fitness function. One typical way to do it is to use an interactive GP system. This system will let a human give a score to each individual. However, humans are slow and inconsistent relative to computers. We need a more automated and reliable solution than a human to guide the system.

An artificial neural network (ANN) can be used as the fitness evaluator for GP. One type of ANN is the convolutional neural network (CNN). It adds convolutional layers to an ANN that can extract high level information from the input images. A CNN is a type of ANN widely used to recognize and classify images. It is usually trained by a large number of images to form an abstract model of the images. CNNs have been used in vision and art applications by several researchers. Gatys *et al.* proposed a CNN-based system that can separate the style and content from an image [28]. Simonyan and Zisserman introduced VGGNet that is designed for classifying real-world photos [54]. Maturana and Scherer proposed VoxNet that is used to classify voxel-based 3D objects [45]. Qi *et al.* proposed PointNet, which is similar to VoxNet, but using a point cloud as the input data, to classify 3D objects [52]. These researchers achieved impressive results and proved that CNN can be used in image processing and art.

In this thesis, we use GP as the evolutionary system, L-system as the renderer, and the CNN as the fitness evaluator. By combining the GP with an L-system, GP can generate different styles of plants automatically. Thus, a CNN can be used to evaluate the images generated by the L-system. In this case, the CNN will be acting as a fitness function in the GP system. We experimented with several CNN strategies and found that one strategy is better than the others. During the experiments, we also found different GP languages used will have impact on the results.

In this thesis, we propose a fully automated tree image generating system that requires no human interaction. We discussed the impact of using different network architectures and different fitness evaluations for GP. Our research shows the effective combination of using CNN with GP. We also show that GP can easily mislead the CNN to make unsatisfactory classifications.

The thesis is organised as follows:

- Chapter 2 is the background knowledge of the thesis.

- Chapter 3 discusses literature related to the thesis.
- Chapter 4 discusses our system design.
- Chapter 5 presents the experiment setup and results of our thesis.
- Chapter 6 discusses concluding remarks and future work.

Chapter 2

Background

2.1 Genetic Programming

In 1988, Koza proposed and patented a genetic algorithm (GA) for program evolution named genetic programming (GP) [34, 35]. Similar to a normal GA, a difference is that GP uses a tree-based structure as a chromosome to represent a program and uses genetic operations to evolve new programs.

Genetic Programming (GP), introduced by John Koza in 1988 [34], is one of the evolutionary-based artificial intelligence algorithms that simulate Darwinian evolution [35]. It uses a basic genetic algorithm (GA) to simulate evolution in order to find a solution to a particular problem. Computer programs that can potentially solve the problem are the chromosomes in GP. Each chromosome is also referred to as an individual. The set of individuals comprises the population. GP evolves new individuals using two genetic operations, crossover and mutation. Each evolution cycle is called a generation. At the end of each generation, GP evaluates every individual in the current generation and gives each a fitness score, using a fitness function. GP terminates when it meets a termination condition, either solution found or maximum generation reached. After termination, the best individual selected is the optimal solution given by GP. More details are discussed in the following sections.

2.1.1 Representation

Different from GA, GP uses a tree structure as a chromosome. There are two types of nodes used to construct a tree: functions and terminals. The function is a node that requires at least one node to be its child to operate. A terminal does not require any children. A simple example of a GP tree would be ADD(DIV(6, 3), X). The ADD

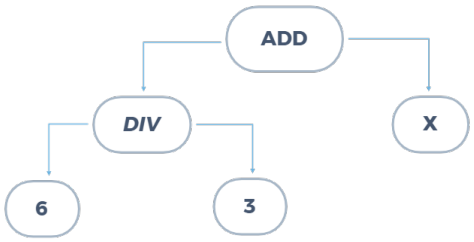


Figure 2.1: A simple example of GP tree structure.

and DIV in this tree are functions, and the numeric values, 6 and 3, and the variable X are terminals. Figure 2.1 shows the diagram of this tree. It represents Equation 2.1:

$$Result = (6 \div 3) + X \tag{2.1}$$

In GP, the functions and terminals should be pre-defined before execution. The set of functions and terminals is called the language in GP. The language is defined according to the problem to be solved. Usually, a size limit is applied to trees to prevent them from growing too large.

2.1.2 Initialization

Before evolution happens, individuals need to be randomly generated to form the first generation's population. The process of generating the first generation is called initialization. A random tree generator is used to perform this task. There are two main methods of initializing a tree: the grow method and the full method.

The grow method creates a tree that can be of any depth up to a pre-set maximum depth. This method randomly picks nodes from functions set and terminals set to construct the tree. If the pre-set maximum depth is reached, only the terminals will be used to construct the remaining tree.

The full method generates a tree having a pre-set depth every time. It only picks functions to construct the tree before reaching the maximum depth, after which the terminals will be picked to terminate the tree.

Comparing these two methods, the full method will generate trees with similar bushy shapes but different nodes in a tree. However, the trees generated from the grow method can have a variety of shapes and depths.

2.1.3 Fitness-proportional selection

After initializing a population, GP will select individuals to evolve the next generation. Individuals are selected via the Darwinian principle that fitter individuals are more likely to reproduce. There are multiple methods of selection. The tournament selection is one of the most popular selection methods. It randomly selects a set of a pre-defined number of individuals. The individual having the highest fitness in this set will be selected as the "winner" of the tournament. For crossover, two tournaments are done to find two parents. This method ensures the fitter individuals have a higher chance of being selected, but also gives a chance to individuals that are not too fit.

2.1.4 Reproduction

In order to simulate Darwinian evolution, there must be a method to generate the next generation. The GP will pick individuals from the current population using the selection method and perform reproduction using genetic operations. These genetic operations will recombine or mutate the selected individuals. The new individuals will be used to construct the next generation. The sexual recombination operation is called crossover, and the mutate operation is called mutation.

Crossover

Crossover combines two existing individuals as parents to create offspring. Two individuals selected using the selection method will be used to perform this operation. A random node or the entire sub-tree from this random node in each of these two trees will be swapped to generate the offspring. Figure 2.2 shows an example of crossover. The selected sub-trees of two parents in the red box are swapped to produce two offspring. Crossover is the primary genetic operation in GP that allows offspring to inherit features from their parents.

Mutation

The method of modifying the existing selected individuals is called mutation. Mutation will modify the selected individuals by randomly selecting a node or sub-tree, and replace it with a randomly generated sub-tree. Figure 2.3 shows an example of mutation. In nature, mutations can be good, bad or indifferent. Most mutations are

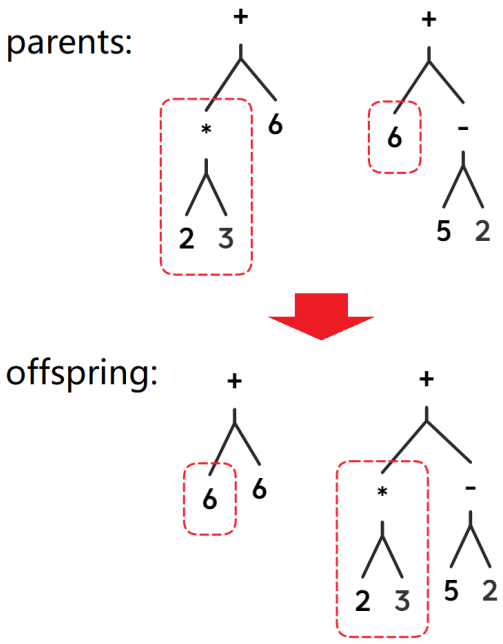


Figure 2.2: An example of GP crossover. The sub-trees in the red box are swapped in the offspring.

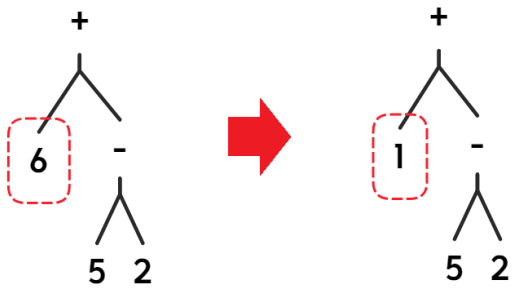


Figure 2.3: An example of GP mutation. The node selected in the red box is replaced with a newly random generated number 1.

neutral, but they can also be harmful [42]. Similarly, GP's mutation rate should not be too high, otherwise it tends to become a random search.

2.1.5 Evaluation

Since GP needs to evolve the population, there must be a way to measure how good the individuals are. The method of measuring individuals is called fitness evaluation. The evaluation process will give a fitness score to each individual based on a fitness function. Fitness function is a pre-defined method that evaluates individuals based on their performance on the problem at hand. It is problem-related, and each fitness function is usually only used in its designated problem. The fitness function can be an algorithm, a score given by a human, or a mathematical measurement. Fitness evaluation, in most cases, is the most time-consuming process in GP.

2.1.6 Automatic defined function

Automatic defined function(ADF) is introduced by Koza [36]. It is a sub-tree module that can be evolved separately from the main tree during the evolution. It allows GP to evolve more than one tree concurrently and permits modular program to be evolved. The ADF can use either same or different language from the main tree. GP with ADFs will have one or more dummy arguments to be replaced by the ADF trees during evaluation.

2.2 L-system

The Lindenmayer system (L-system) is a parallel rewriting system that was introduced by Lindenmayer [51]. It was designed to generate plants by computers using symbolic expressions. The famous turtle graphics [7] inspired L-systems. There are two major components in L-systems: the rewriting system, and the representing system. The rewriting system requires one axiom and one or more production rules as input and expands the input iteratively. The representing system acts like turtle graphics, to draw vector graphics by following the expanded commands from the rewriting system. The original system works in two-dimensional space. However, Lindenmayer expanded the system to three-dimensions to generate more realistic plants.

L-system translates a symbolic grammar to a graphical image. The rewriting system consists of an initial axiom and production rules. Both the initial axiom and rules are made of symbols that indicate draw lines, turns and stack operations. Equation 2.2 defines a general L-system, where V is a set of symbols, ω is a string containing the symbols from V that indicates the initial state of the system, and P is a set of production rules that can replace the symbols in the ω.

$$G = (V, \omega, P) \tag{2.2}$$

A pre-set value indicates the number of iterations to be performed. In each iteration, the rules will replace the symbols in axiom in parallel. Once the number of iterations has been met, the rewriting system stops and passes the produced string to the drawer.

The drawer is similar to turtle graphics that follows the symbols as commands to draw a graphical image. Before running the L-system, the angle of rotation of the turtle can be set by the user. Table 2.1 lists several commonly used symbols of L-system. Once the system has a "F" command, it will move the turtle forward with certain distance. The "+" and "-" commands are used to turn the turtle left or right. L-system also allows the stack operations. When the "[" command appears, the system will push the current location of the turtle into a stack. When the "]" command appears, the system will pop the most recent location in the stack and teleport the turtle to this location. The stack operations give the system ability of drawing branches.

Figure 2.4 shows an example of an L-system expression. This expression starts with the axiom $F+$. In each iteration, symbol F will be replaced using rule $F = F+$, and symbol + will be replaced using rule $+ = FF$. After reaching the number of

Table 2.1: Commonly used L-system symbols

Symbol	Description
F	Move forward
+	Turn left
-	Turn right
[	Push commands into stack
]	Pop commands out of stack
X	Placeholder, can be replaced by rules
Y	Placeholder, can be replaced by rules

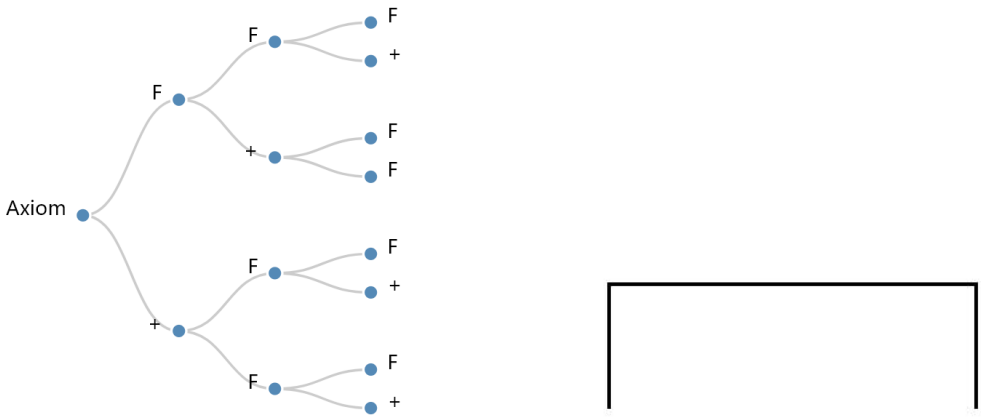


Figure 2.4: A simple example of L-System expression. Axiom:F+, Rules:F=F+, +=FF, Iteration=2, Angle=90°. Right image is the output of this example.

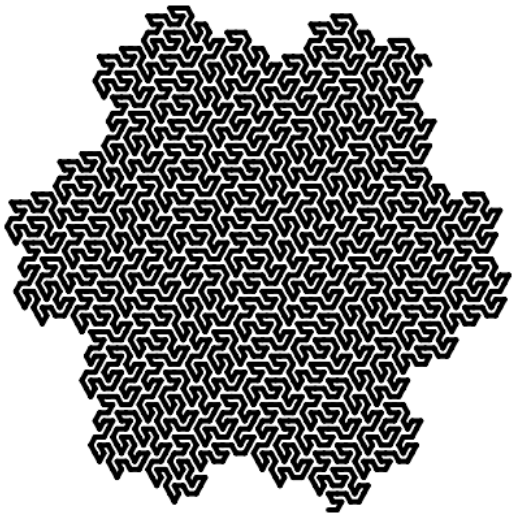
iterations, the rewriting process halts. The final produced string is $F + FFF + F+$.

The purpose of introducing the L-system is to simulate biological growth patterns. It can easily generate self-repeating images. Figure 2.5 shows several examples generated by L-systems. From these examples, we can find that an L-system is a general system that can generate different images. Within the range of all images that L-system can generate, a small portion of images is of special corner to this thesis: trees.

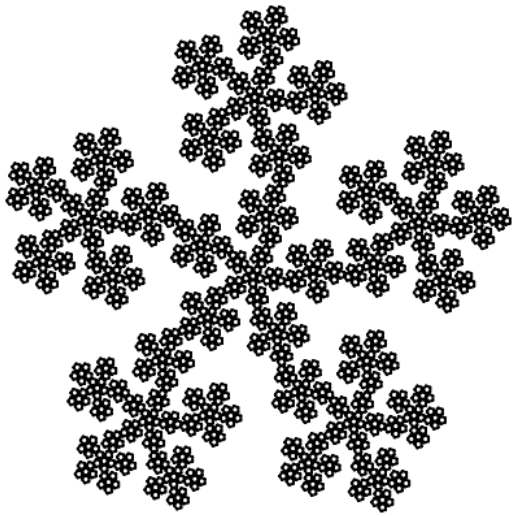
Figure 2.6 shows plants generated using L-system. From these examples, it is hard for a human to predict if results would be trees or not by just looking at the expressions.

2.3 Deep Learning

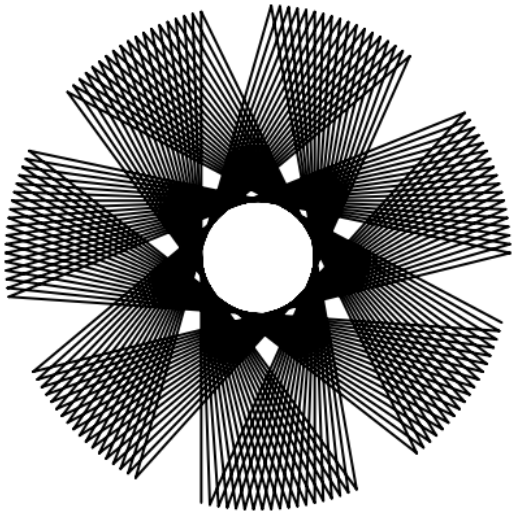
Deep learning is currently one of the hottest topics in the area of artificial intelligence. A basic technology of deep learning is the artificial neural network (ANN). It



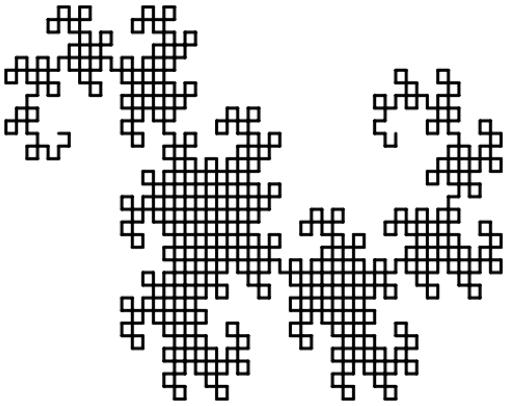
$n = 4, \delta = 60^\circ$
F
$F \to F - f - - f + F + + FF + f -$
$f \to + F - f f - - f - F + + F + f$



$n = 4, \delta = 72^\circ$
$F - F - F - F - F$
$F \to F - F - F + + F + F - F$



$n = 7, \delta = 77^\circ$
F
$F \to F + + F$



$n = 10, \delta = 90^\circ$
FX
$X \to X + YF +$
$Y \to - FX - Y$

Figure 2.5: Images generated using online L-system generator [5].

$n = 4, \delta = 22.5°$
F
$F \rightarrow FF - [-F + F + F] + [+F - F - F]$

$n = 5, \delta = 20°$
F
$F \rightarrow F[+F]F[-F][F]$

$n = 5, \delta = 22.5°$
X
$X \rightarrow F - [[X] + X] + F[+FX] - X$
$F \rightarrow FF$

Figure 2.6: Trees generated from L-systems. Images generated using online L-system generator [5]. The expressions are from Lindenmayer [51]. n is number of iterations. δ is the angle of rotation.

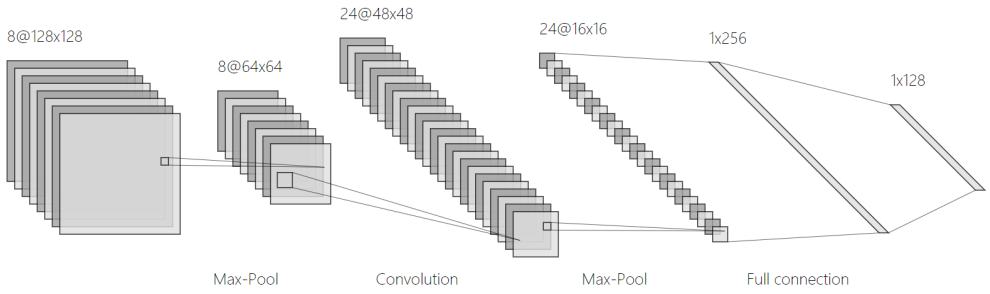


Figure 2.7: A typical CNN architecture. Image generated using *NN SVG* [9].

composes multiple processing layers to learn representations of data from the abstract level [38]. ANN uses the backpropagation algorithm to adjust the internal parameters automatically. By adding convolutional layers to the front of ANN, we can have a convolutional neural network (CNN). Figure 2.7 shows the architecture of a typical CNN model. Many researchers have proved that the CNN is capable of state-of-the-art results in image processing [37, 45, 53, 54]. The following sections discussed CNNs in more detail.

2.3.1 Convolutional layers

The main difference between a CNN and an ANN is the use of convolutional layers. Since images are made of millions of pixels, and each pixel comprises three channels,

(R,G,B), the amount of input data will be significantly large. In order to decrease the computational power required to process the images, one or more convolutional layers are added to the front of the fully connected layers. These convolutional layers can abstract the spatial and temporal dependencies from an image by using specialized filters within the layers. After the convolution process, the performance of the entire model will be better due to the reduced image size. Each convolutional layer has a filter that extracts the features from the original image. This filter is called the kernel. The size of the kernel and the weight of each cell on the kernel need to be pre-defined.

During the convolution process, the kernel shifts the original image to perform matrix multiplication between a matrix containing the kernel's weight and the portion of the image that the kernel is processing. The results will construct a new matrix containing high-level information from the original image. This new matrix is called a feature map. One network can have one or more convolutional layers. The more convolutional layers, the more abstract information is modelled.

2.3.2 Pooling

The primary purpose of pooling layers is to reduce the size of the convolved feature map. At the same time, it suppresses the image's noise to extract dominant features from the feature map. There two different methods of performing pooling: max pooling and average pooling. Max pooling takes the maximum value from the area of the image that is covered by the kernel. Average pooling calculates and returns the average of all the values covered by the kernel. In this case, the max pooling method has better performance on suppressing noise since it will throw away the insignificant information. Researchers analyzed the performance difference between these two methods and found the max pooling is better than the average method in most cases [22].

2.3.3 Flatten

Images are two-dimensional information, and the real-world objects are three-dimensional. Since the input layer of a fully connected neural network has only one dimension, a simple "flatten" strategy is required to perform the transformation. By flattening the two-dimensional data into one dimension, the flatten layer will perform a row-major scan through the image and append the row to the tail of its predecessor. The output of the Flatten layer will be a $n \times 1$ matrix, which n is the number of pixels in the

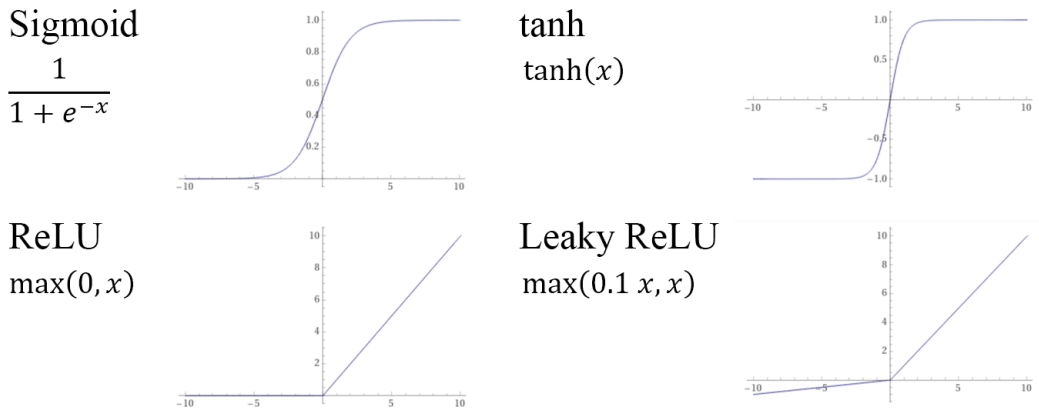


Figure 2.8: Activation functions that are commonly used. Graphs generated using Wolfram Alpha [13].

input image.

2.3.4 Fully connected layers

After convolutional, pooling, and flatten layers, the fully connected neural network is the next station for the data flow. The fully connected layer is an effective way of learning non-linear feature maps containing the high-level information generated from the convolutional layers. It uses the training data to construct a possibly non-linear function that will fit the data best.

The fully connected layers consist of multiple neurons. Each neuron uses a predefined function to make a decision. The function is called an activation function. Several weighted paths connect two neurons, and each neuron in the current layer has a connection to all neurons in the next layer. In some cases, one or more dropout layers will be added to the network to prevent over-fitting. By using a strategy called batch normalization in the network, the efficiency of the training might increase in some cases. The following sections discussed the activation function, dropout, and the batch normalization in detail.

Activation functions

The fully connected layers are made of neurons. The activation function is a mathematical function used in the neurons. Each neuron in the network is acting as a decision-maker, and cooperates with the others. The decisions made by the neurons are based on the activation function. There are several different activation functions possible [14, 41, 47]. Figure 2.8 shows several commonly used activation functions.

The ReLU function is widely used in the hidden layers. It formats the negative values to zero and keeps the positive value unchanged. Usually, a CNN uses more than one type of activation function, since different neurons have different tasks. The Softmax and Sigmoid functions are two common activation functions used in the output layer. This special type of activation function is also called a classification function. The sigmoid function is flattened at both sides along the input axis but has a slope in between. Softmax function extends the sigmoid to more than two categories. Binary classification problems commonly use Sigmoid as the classification function, and the multi-category classifications use Softmax.

Dropout

To train a neural network on a relatively small dataset will likely to over-fit. By adding a filter to ignore some information between two layers can effectively prevent over-fitting. A percentage of "ignorance" needs to be set when constructing the model. This percentage is called the dropout rate. It randomly picks neurons and prevents them from transferring their decisions to the next layer. The dropout rate limits the proportion of the selected neurons to be ignored.

Batch Normalization

Researchers first discussed batch normalization in deep learning in 2015 [31]. During the training process, the network parameters are changing to fit the data. Since the subsequent layer use the previous layer's output as its input, the parameter adjustment in the previous layer will affect the distribution of input in the current layer. This shift from the previous layer will be added to the shift made by current layer and passed to the next layer. This effect is called internal covariate shift [31]. The deeper the network, the more shift will appear at the output layer. Instead of using very small learning rate, the batch normalization can be used to solve this problem. It re-distributes the input data in current batch before each layer to avoid the internal covariate shift. Batch normalization is not a hard requirement for all neural networks. However, it can effectively utilize the sensitive area of the activation function to accelerate the training process.

Before applying the data to the model, the data need to be normalized or standardized. By normalizing the data, we will have a reasonable range of data. However, this regular normalization technique can only be applied to the data before it is fed into the neural network. When training neural network, gradient exploding and van-

ishing can lead the model to be unstable or unable to learn. Gradient exploding happens when the model accumulates large error gradients which causes very large updates to the weights, and which results in even larger error gradients. Once the accumulated error gradients become very large, the magnitude of weight updates will become very large as well. Gradient vanishing is similar but the accumulated error gradients become very small, and they hardly update the weights. They will cause the model updating the weight between neurons to be either too slow or too quick. This can happen in the hidden layer that regular normalization techniques cannot reach. Batch normalization can normalize the data in the hidden layer to avoid this from happening.

2.3.5 TensorFlow and Keras

In this section, we talk about several tools that are widely used by researchers and industries for machine learning nowadays.

TensorFlow

TensorFlow is an end-to-end open-source machine learning platform that is initially introduced by Google in 2015 [11]. It provides a complete set of tools for machine learning in several popular programming languages like Python, JavaScript, and Swift. It is capable of constructing various types of neural networks by using several lines of code to minimize programming effort. Since training neural networks requires a massive amount of computing power, TensorFlow also provides GPU acceleration as long as the environment required to execute commands on a GPU is installed on the computer. TensorFlow officially supports most modern NVIDIA graphics cards that use CUDA. Several open-source libraries add AMD graphics cards support to TensorFlow as well [4].

Keras

Keras is an open-source deep-learning framework in Python [4]. It provides a high-level neural network API capable of running on several different machine learning libraries such as TensorFlow [11], CNTK [8], and Theano [12]. The benefit of using Keras instead of these barebone libraries is that Keras simplifies the commands of constructing and running neural network models even further. Keras can automatically detect the GPU model and allow the user to use them by simply turning on a flag without programming extra configurations. Also, Keras adds support for AMD

GPUs to perform acceleration. It comes with clear and detailed documentation, works with TensorFlow perfectly, and allows researchers to focus more on the experiments, rather than struggling with programming and debugging.

Although Keras supports AMD GPUs, the graphics cards that support NVIDIA CUDA perform better with TensorFlow and Keras in most cases. For that reason, we decided to use NVIDIA GPU as part of our system. Because both TensorFlow and Keras are running on Python, we will use Python as the programming language for CNN in this thesis.

Chapter 3

Literature Review

This chapter reviews some previous works that are related to this thesis. The core components of our system are genetic programming, convolutional neural networks, and L-systems. We discuss applications of convolutional neural networks that are studied by researchers today. We also talk about relevant applications of genetic programming and L-systems. Since we used GPU acceleration in our system, the graphic card programming framework that we used is also discussed.

3.1 Convolutional Neural Network

McCulloch and Pitts first introduced the concept of the artificial neural network in 1943 [46]. A neural network is made of neurons that have activation functions. For any logical expressions, there must be a network containing neurons that will have the same behaviour. One of the major branches of artificial neural networks is convolutional neural networks. In 1980, Fukushima introduced two fundamental ideas for CNNs: convolutional layers and downsampling layers [26]. A pooling layer is a popular approach used by both academic and industrial areas for the downsampling layer. It is specifically designed for computer vision. Today, researchers specialize various CNN models to fit different applications.

Krizhevsky *et al.* introduced a CNN model called AlexNet in 2012 [37]. There are about 60 million parameters and 650,000 neurons to construct the network. They trained the network to classify real-world photos into 1000 different categories. GPUs are also used as co-processors to accelerate their training process. This model is the state-of-the-art approach in the ImageNet LSVRC 2010 contest [6].

Another model used for classifying real-world photos is called VGGNet that is introduced by Simonyan and Zisserman in the ImageNet ILSVRC 2014 contest [54].

It is two years newer than AlexNet and performs better. They proposed six variations of the VGGNet, from 11 weight layers to 19 weight layers. Even the smallest version of the VGGNet contains 133 millions parameters, and the largest version is up to 144 million. The tremendous depth and width of the network requires much more computing power than AlexNet.

GoogLeNet, introduced by Szegedy *et al.* in 2014 [56], is a better approach to reduce computational requirements. The unique design of this model is the inception module. It gives the capability to the model to decide for itself which filters to use in the convolutional layers. The model will learn all these parameters when training. The inception module significantly increases the network's efficiency and allows researchers to create deeper and wider networks with similar computational resource consumption.

CNNs can also be used to classify 3D objects as well as 2D images. Maturana and Scherer proposed VoxNet to recognize 3D objects [45] and achieve state-of-the-art accuracy. They used voxels, which are the 3D version of pixels, to represent the 3D information, and a 3D CNN network structure to process the 3D data.

Qi *et al.* proposed a PointNet model to classify 3D objects based on point cloud data [52]. Different from the voxel, the point cloud is another type of data to represent three-dimensional objects. The point set is represented by a group of coordinates. They used two networks to achieve the task, the classification network and the segmentation network. The classification network extracts the global feature from the data and passes it into the segmentation network. The segmentation network is an extension that makes the decision on both local and global features.

Socher *et al.* introduced a model that combined the convolutional and recursive neural networks (CNNs and RNNs) to classify 3D objects [55]. The data they used is in RGB-D format, which adds a depth extension to the standard two-dimensional images. They generated the CNN filters using unsupervised training and applied them to a single convolutional layer. After that, the RNN will take the data flow and learn hierarchical features from the data.

However, the CNN is not the silver bullet for all kinds of problems. Some other systems can fool it easily. Nguyen *et al.* investigated several state-of-the-art neural networks and fooled them using the image generated by an evolutionary algorithm [49]. They used an evolutionary algorithm generating images to maximize the classification error of the neural network. They generated a group of images that are unrecognizable to human but 99.9% correct to the model. The evolutionary algorithm (EA) generated images that achieved high scores in many categories. This

target triggers the EA system to merge multiple features that are recognizable by the neural network into one image.

3.2 Genetic Programming and Lindenmayer Systems

By combining GP and L-systems, we have an evolutionary system that can automatically generate images. The evolved GP expressions will be L-system expressions. Automatic defined functions (ADFs) in ECJ provides GP a way to denote multiple evolved trees at the same time. Each individual will have at least two trees, the main tree and ADF. One of these trees, the main tree, will be used to evolve the axiom of the L-system. The ADF trees will be used to evolve the production rules. The number of production rules must be pre-defined so that the number of ADFs can be set before GP starts running.

Once the GP generates an individual, the system will translate the individual to an L-system axiom and a set of production rules. The main tree will be expanded to a string directly. Since every production rules must have a symbol that will be replaced, the root of ADF will be the replaced symbol of this rule, and the rest of the tree will be used to construct the replacement string. Then, the L-system will render the axiom and the set of production rules and produce an image. This image will be used in fitness evaluation to generate a fitness score for the individual.

Past research has combined the evolutionary algorithm and L-systems in both 2D and 3D. Congdon and Mazza proposes an interactive GA and 3D L-system [25]. The chromosome they used in the GA represents several parameters of a tree. It defines number of branches, length ratio, width ratio, branch taper, branch proximity, tree depth, branch angle delta, vertical change, parent influence, branch direction noise, branch number noise, subtree depth noise, and random seed. The renderer will use these parameters as reference and generate a tree using L-system. The GA parameters can be determined by the users or evolved by the GA process. The user is the fitness evaluator in their system. Each individual will be given a score by the user. Since their system has several restrictions to the trees and requires human to interact, their results turn to be very good but have similar style.

Bonfim and Castro proposed the FranksTree [20], which investigated the possibility of combining hybrid evolutionary algorithms and L-systems. Instead of evolving the L-system grammar, their work evolves the derived L-systems. Their system starts

at a population that is already known as trees and only cuts the branch in the interconnections of the trees during the crossover. The user will interact with the system by choosing which trees will be the parents and perform a crossover on them. No mutation is implemented in their system. Since the selection process is done by the user, there is no need to implement fitness score in their system. The most of their results can be recognized as trees. However, they have limited population size so that it can let the user to choose parents. Their results are combinations of several branches from the first generation.

Jacob proposed the Genetic L-system Programming (GLP) system [32] that evolves three-dimensional plants using genetic operations. The GP language used in the system is as L-system. They introduced a context-dependent L-system data type with interpretation functions. The fitness values for each individual are given by the interpretation functions. They show that the system has ability to create some complex structures in a 3D environment.

Bergen and Ross investigated the multi-objective evaluation approach to the GP and 3D L-system [18]. They used voxels as the basic unit of 3D object, and automated the process of aesthetic 3D modelling. They experimented with the summed rank and Pareto strategies of multi-objective fitness evaluation and found that each of them has its strengths.

3.3 Artificial Intelligence and the Arts

Since aesthetics and arts is highly subjective to individual's preferences, the fitness evaluations in past research are mainly interactive or mathematical. Galanter talked about the history of computational creativity in his book [27]. Most of the previous research in this area is based on psychological facts, such as the golden ratio, the rule of thirds, and the fractal dimensions. Lopez talked about computational creativity in both music and visual arts [43]. In the visual arts section, AARON, a robotic painting system, is discussed in this book. The purpose of the system is to make robots become more creative. However, the robot cannot break several rules that were pre-determined by the researchers, which limited the system's creativity. This brings up an issue that the computer system will encounter a hard time without the help of humans in the field of arts. Cohen-Or and Zhang talked about three types of computational creativity approaches: creative support tool, generative systems, and computer colleagues [24]. Computers are supporting humans in the first and third approaches. Only the second approach will be a fully automated system.

These days, many researchers are trying to replace the role of humans with trained AI during the recent boom of machine learning. Baluja *et al.* developed an artificial neural network (ANN) to learn user preferences and generate human pleasing images using GA [16]. They investigate the impact of the results caused by the size of ANN and different evolutionary algorithms. Their final results are acceptable, but the performance can be improved.

Blessing and Wen proposed a support vector machine (SVM) approach to predict the artist's name for example artworks [19]. They focused on seven artists' works and a total of 1400 images for the training dataset. Their results shows that machine learning is capable of classifying art works.

Cetinic *et al.* investigated the performance of CNN evaluating fine arts [23]. This research focused on three aspects of artworks: aesthetics evaluation, sentiment, and memorability. They used images from multiple sources, from modern photographs to traditional paintings. They did the experiments on different CNN models and picked the ones with the best results and compared them with the existing ratings from humans.

Gatys *et al.* proposed a CNN approach that separates the image content from style and applies a different style to the content [28]. Their work is based on the 19-layer VGG-Network but using only convolutional and pooling layers. They found that using average pooling improves the quality of the results more than the max pooling. Their research apply styles of given artworks to real-world photographs. Several companies and teams are using the methodology they proposed, such as Prisma Lab [10] and Deep Arts [1], allowing the users to change the styles of their photos.

The innovation engine is introduced by Nguyen *et al.* in 2015 [50]. They used deep learning to generate two-dimensional images that are recognizable by both humans and deep neural networks. Based on their work, Lehman *et al.* designed a system that can generate 3D objects without humans interaction [40]. The system demonstrates a novel method of combining evolutionary algorithms and deep learning. They rendered the 3D objects to 2D images and recognized the images using a deep neural network.

Bontrager *et al.* combined generative adversarial networks(GAN) with interactive evolutionary computation [21]. This is unsupervised learning that cooperated with the human interactive system. Their system can evolve original images under the supervision from human to match the target images. They trained the GAN on a specific target domain, and let the interactive evolutionary algorithm to control the input of the GAN.

Chapter 4

System Design

There are three major components in our system: the L-system, the CNN, and the GP. Figure 4.1 shows the overall design of the system. The details of each module are discussed in the sections that follow.

4.1 L-system

The L-system is the painter in our system. It takes evolved expressions from GP and produces the images for the CNN to evaluate and people to look at. The L-system command set is defined in Table 4.1. Our L-system is working on a two-dimensional canvas with a white background and a black pen stroke. We set a dynamic canvas size for the L-system so that the painting turtle stays in the canvas most of the time. Also, we leave a 20 pixel wide gap from the image borders to the drawing when adjusting the canvas size. However, the L-system is so general that it can draw a variety of images that might require a very large canvas. In these cases, this might cause the system to run out of memory when drawing and evaluating. Thus we set a maximum value for both the width and height of the canvas to be 4096×4096. Number of iteration will significantly slow down the execution of the L-system. Let number of symbols in an axiom be a, number of iteration be i, and average number of symbols in production rules be p. Assuming 25% of the symbols in an expression will be rewritten on average, the number of symbols in the final expression can be recursively defined as

$$\begin{aligned} f(0) &= a \\ f(i) &= \frac{3}{4} \times f(i-1) + \frac{1}{4} \times f(i-1) \times p \quad . \end{aligned} \tag{4.1}$$

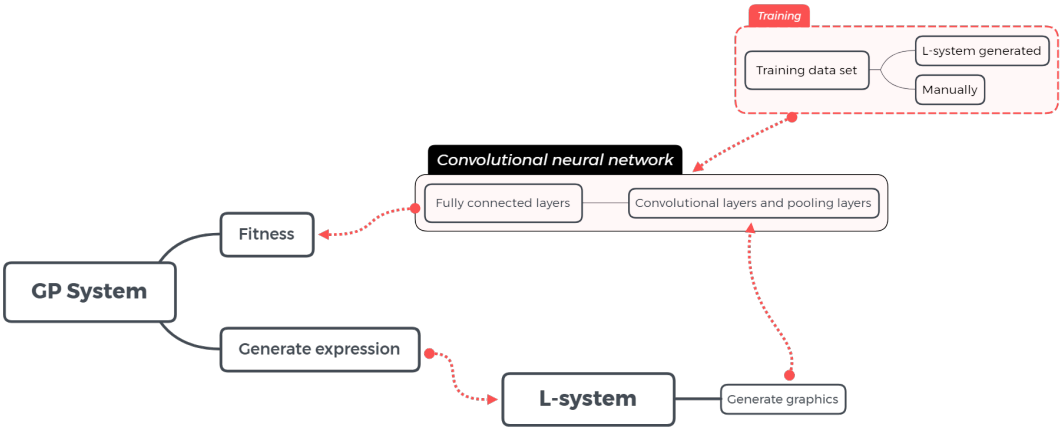


Figure 4.1: The structure of the system.

Let the number of symbols in an axiom be constant and number of iteration be n. We find the time complexity of a rewriting relative to the number of iteration to be $O(2^n)$.

Table 4.1: The commands of L-system.

Command	Usage
F	Move forward.
+	Turn left.
-	Turn right.
[	Push command into stack.
]	Pop out the commands in the stack.
X	Placeholder, can be replaced by rules.

To generate different looking trees within same species, we introduced a random branch cutting strategy. It decides if some branches need to be cut. The probability of cutting a branch varies from 0% to 50%, and it is defined by each branch individually. The cutting probability of the main trunk is 0%. A smaller branch has a higher probability of being cut. Each level of branches' cutting probability will increase 5%. Since the randomization is used to generate training data, all of these random features will be removed from the L-system when integrated with GP in Chapter 5.

There are various of L-system that can be modified before or during the execution. We tested several combinations and picked one that works best for us. The parameters are shown in Table 4.2.

Table 4.2: The parameters used in our L-system.

Parameters	Value	Usage
Turning angle	20°	Control the angle of rotation
Step length	15 pixels	Control the length of each step
Iteration depth	4	Control the depth of the iteration
Maximum width	4096 pixels	Maximum width allowance of the canvas
Maximum height	4096 pixels	Maximum height allowance of the canvas

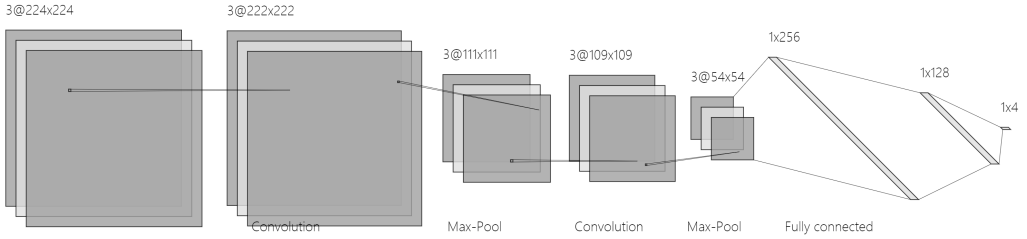


Figure 4.2: The architecture of our CNN model. Image generated from NN SVG [9].

4.2 Convolutional neural network

CNN is the fitness evaluation engine of the system. It replaces the position of human evaluation in human-interactive GP to identify the quality of trees produced by GP and L-systems. It takes the images generated from L-system and gives a fitness score for each image. The architecture of the CNN we used is shown in Figure 4.2. Our model is inspired by the VGG Net [54]. The scale of the problem that VGG Net solves is much larger than ours as it classifies 1000 image categories. Thus we reduced the size of the network and slightly modified the structure to fit our problem. Although not necessarily optimal, we picked one which works reasonably well. The detailed setup of the convolutional layers and the fully connected layers are discussed in the following sections.

4.2.1 Convolutional layers

Since we might add colours to our L-system in the future, we use all of the three channels as the input of the neural network. There are two convolutional layers, two max-pooling layers and a dropout layer in this set.

Since we also borrowed some idea from VGG16, we let the dimension of the input image is resized to 224×224, which is the same as input size of VGG16 [54]. We

also tried to resize the image to 32×32, which is the input size of LeNet-5 [39]. However, the LeNet-5 is used to recognize handwritten, which is much simpler than L-system trees. Resizing from 4096×4096 to 32×32 makes the images too blurry, which are even impossible for human to distinguish if they are trees or not due to the distortion. Then, we add our first convolutional layers to the system. The kernel size of this convolutional layer is 3×3. There is a total of 128 convolutional neurons in this layer using ReLU as the activation function. The next layer is the first max-pooling layer. It takes the output from the previous convolutional layer and using a pool size of 2×2 to get the output. After that, another convolutional layer takes the output from the max-pooling layer and uses a 3×3 kernel to convolve the data. In this layer, there are a total of 32 neurons, and the activation function is ReLU. The second max-pooling layer is following the second convolution layer. It is identical to the first max-pooling layer that having a 2×2 pool size. After the second max-pooling layer, we added a dropout layer, which has a 25% dropout rate before flattening the data and sending it to the fully-connected layers.

The initializer we used to randomly initialize the convolution filter is Glorot uniform initializer (also called Xavier uniform initializer) [29]. It samples a uniform distribution from -*limit* to *limit* where *limit* is defined in the Equation 4.2.

$$limit = \sqrt{\frac{6}{fan\_in + fan\_out}} \quad , \tag{4.2}$$

where *fan_in* is the number of input units and *fan_out* os the number of output units [4].

4.2.2 Fully connected layers

After the convolutional layers, the system holds a flattened feature map. The fully connected layers will take this feature map as the input. There are two hidden layers and one output layer in this set. To prevent over-fitting, we have two dropout layers. Also, we compiled two batch normalization layers into the model to stabilize and accelerate the training process.

The first fully connected layer consists of 256 neurons using the activation function ReLU. Following that, a dropout layer with a 50% dropout rate processes the data. Then, a batch normalization layer next to it will normalize the data. After that, the second fully connected layer that has 128 neurons with activation function ReLU is used. Then, the second dropout layer and batch normalization layer are added to

the model. These two layers have the same dimension as the first ones. In the end, another fully connected layer is added as the output layer. The number of neurons in this layer should be the same as the number of output categories. In our case, we have three different species and a non-tree category, so this ends up with four neurons in the output layer of our model.

4.3 Genetic programming system

The GP system is the evolution engine of the entire system. It evolves the L-system expressions and sends them to the L-system interpreter. The L-system generates the images using the GP expressions and sends the images to the CNN for fitness evaluation. We used ECJ 26 [2], a Java-based evolutionary library, to implement our GP.

The first thing that we need to decide is the language used by the GP. Since ECJ defines both functions and terminals as functions, where terminals are functions with no arguments. Since GP must cooperate with the L-system, they must have the same language so that they communicate flawlessly. The commands in Table 4.1 define the GP language. However, most of these commands do not have hard requirements for the number of descendants. We have to carefully re-define these commands in GP to fit the L-system's requirements.

Table 4.3: The language set used in the GP.

Function	# of arguments	Description
Forward (F)	2	Move forward.
Left (+)	2	Turn left.
Right (-)	2	Turn right.
ForwardLeaf (F)	0	Move forward without descendants.
LeftLeaf (+)	0	Turn left without descendants.
RightLeaf (-)	0	Turn right without descendants.
Branch ([ ])	1	Create a branch using stack.
X	0	Placeholder, can be replaced by rules.

The Table 4.3 shows the GP language. Most of these commands have two versions, with arguments and without arguments. That means most of these commands can be either functions or terminals. The GP constructs L-system expressions from the tree by performing an inorder traversal.

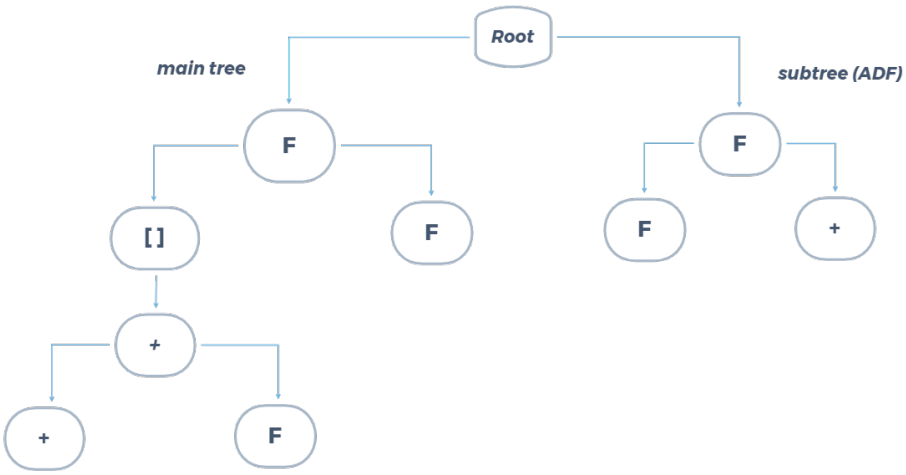


Figure 4.3: An example of an individual using our GP language. Left-hand side is the main tree, and right-hand side is the subtree.

Since ECJ can evolve multiple trees simultaneously, our GP has a subtree to construct the L-system production rules. We implemented the subtree using the automatically defined functions (ADFs) in ECJ. Usually, GP will replace the ADFs during execution. We use the ADF as a subtree to construct the production rules. The replacement procedure will be performed by the L-system instead. The ADF trees will construct the replacement body of a production rule by performing an inorder traversal as well. The root of the ADF tree will be the symbol in the production rule that is to be replaced.

Figure 4.3 shows an example of a GP individual, which consists of two trees. The tree on the left-hand side is the main tree. It will be used to construct the axiom of the L-system. After performing an inorder traversal on this tree, the result would be $[++F]FF$. The subtree on the right-hand side will execute an inorder traversal to construct a production rule. The production rule constructed based on the tree is $F = FF+$.

Although the L-system can have multiple production rules, we only used one production rule in our GP system to reduce the complexity of individuals. Using multiple rules might perform better in some cases, but it requires more computational resources and time to run the experiments. We did some trial experiments on multiple-rule expression. However, it frequently falls into an "out of memory exception" because of the limitations of the hardware we used. Thus, we rolled back to

the single rule system. Moreover, the single rule system we used provides acceptable results for our applications.

Chapter 5

Experiment Setup

In this chapter, we present the experiments for tree generation with 2D L-system. The environment we used to run these experiments has been discussed in Chapter 4.

5.1 Preparation and first experiment

The goal is to generate 2D trees belonging to one of three styles (or species). We will use the trained CNN to evaluate trees generated by evolved L-system expressions.

5.1.1 Generating training images

We first have to obtain the training images from our L-system. The two-dimensional L-system is written in Java. It has two major parts: the iterator and the interpreter. The iterator is the recursive rewriting system to expand the L-system expression during iterations, and the interpreter is the renderer of the image. It renders the final images based on the results from the iterator.

L-system iterator setup

Since the L-system expressions have an axiom and production rules, they need to be expanded to commands for the interpreter to execute. The purpose of the iterator is to expand the L-system expressions. The iterator requires a depth of iteration that indicates how many times the expression will be recursively rewritten. The more iterations the system runs, the more complex structures it can generate. Since we want our system to have more variations, we should not use a low number of iterations. However, if the number of iterations is too high, the expanded expression will be very long, and the final structure will be enormous. This will easily consume computing

resources and reach an "out of memory error". After several tests, we found that setting the number of iterations to 4 is ideal for obtaining both complex images and performance stability.

L-system interpreter setup

The interpreter uses the idea of the turtle graphics [57]. A turtle carrying a pen is on the canvas. The turtle will follow the commands to move on the canvas. Since it carries a pen, it will draw patterns when moving. Besides vanilla turtle graphics, the L-system interpreter adds stack operations to the basic command list. It uses a pair of squared brackets to refer to push and pop the location of the turtle. The stack operation gives the feature of creating branches. When the open squared bracket appears, the system will push the current turtle's location into a stack. When the squared bracket closed, it will pop out the location from the stack and teleport the turtle to this location and continue drawing.

To move the turtle using commands efficiently, the step length and the turning angle of the turtle need to be pre-defined. In our system, we set the step length to be 15 pixels and the turning angle to be 20°. There are two turning commands: left (+) and right (-). The left command simply turns the turtle counter-clockwise, and the right command turns the turtle clockwise. This means that every time a forward command appears, the turtle will move forward 15 pixels, and every time a turning command appears, the turtle will turn 20°. The turning direction depends on the specific turning command.

Random branching sub-system

To generate similar but different trees in our training and test data set, we introduced a random branching sub-system. This strategy adds a flag in the vanilla L-system to toggle the random feature. The random branching sub-system has two parts: the branch cutting module and the random variables module.

As discussed above, the stack operation gives the feature of branching to the system. The branch cutting module randomly cuts the branches, acting like people who trim tree branches. This operation changes the appearance of the trees but keeps them part of the original species. However, we should give less chance to the bigger branches to be cut. Otherwise, the L-system will likely generate several identical images with just trunks. The implementation of the strategy is to set a range for cutting probabilities. The main trunk will have the lowset cutting probability. The

probability will increase every time the opened squared brackets appear and decrease when the closed squared brackets appear. The minimum cutting probability is set to be 0%, which means it will not cut the main trunk. The highest probability of a branch being cut is 50%. The probability increase or decrease is 5% every step. The result is that this module will generate similar trees from the same expression, but missing some branches.

Having the branch cutting module gives the L-system the ability to generate a variety of trees from the same expression. However, the variation is still not enough for the training and test data set. Thus we introduced a random variables module to the sub-system. This module contains several sets of pre-defined random variables. It can attach to the forward, left and right commands by adding a prefix to it. A prefix of "(R0)" means the random variables set 0 will override this command's parameters. In our system, we have two sets of random variables. "R0" has a minimum value of 15 and a range of 10. "R1" has a minimum value of 2 and a range of 10. For example, the command "F" will move the turtle forward by 15 pixels. By adding a prefix to it, it becomes "(R0)F". This command will first generate a random value from 15 to 25 and use it to override the step length of just this forward command. If the prefix is added to a turning command, the value will override the turning angle instead. This module will change the length and direction of several branches. By combining the branch cutting module with this module, we can use one L-system expression to generate similar but different trees that will belong to the same species.

All the parameters discussed above are in the Table 5.1. After setting these parameters, we used three different expressions to generate three species of trees. The expressions we used are included in Table 5.2. Figure 5.1 shows several examples of each species. We generated 3000 images for each category. These images will be split into training and test sets.

5.1.2 First convolutional neural network training

After generating tree images, we split the data into three categories based on species. Then, we randomly picked 2800 images from each category to form the training set and 200 images to form the test set. For the training set, we flipped the images horizontally and vertically to avoid the directional effect which might lead CNN cannot recognize the same image after flipping it. The architecture of the CNN model is similar to the VGG16 [54]. The only difference is the last layer. Since we only have three categories, the output layer has only three neurons. The optimizer we used for this model is Adam

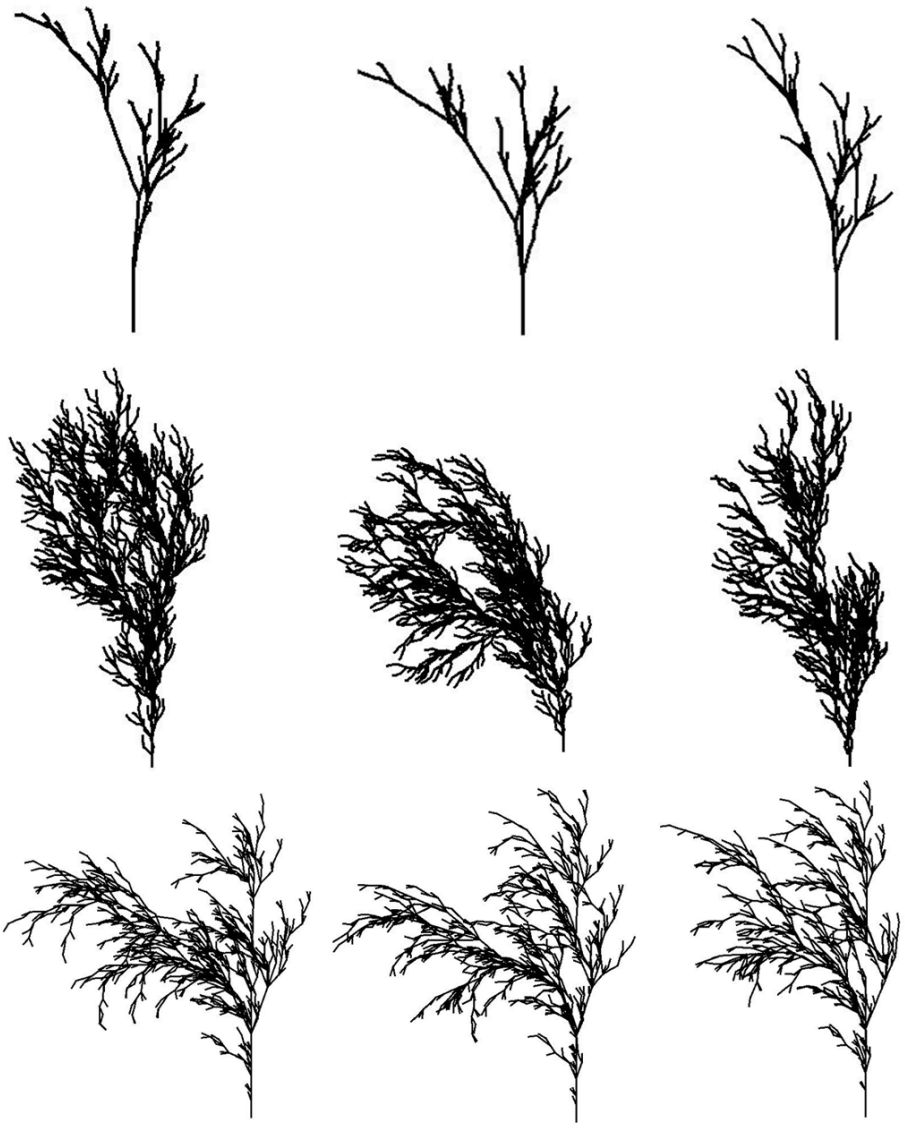


Figure 5.1: The examples of each species one per row. The first, second and third rows are generated by Expression 1, 2, and 3 in Table 5.2.

Table 5.1: The L-system parameters used to generate the training and test data set.

Number of iterations	4	
Turtle step length	15 pixels	
Turtle turning angle	20°	
Branch cutting probability	min	0%
	max	50%
Random variable set R0	min	15
	range	10
Random variable set R1	min	2
	range	10

Table 5.2: The L-system expressions used to generate three species of trees.

Axiom		Rules
Expression 1	X	X=F(R1)-(R0)[[X]+(R0)X]+(R0)F(R1)[+(R0)F(R1)X]-(R0)X
		F=F(R1)F(R1)
Expression 2	F	F=F(R1)F(R1)+(R0)[+(R0)F(R1)-(R0)F(R1)-(R0)F(R1)]-(R0)[-(R0)F(R1)+(R0)F(R1)+(R0)F(R1)]
Expression 3	X	F=FX[FX[+(R0)XF]]
		X=FF[+(R0)XZ+(R0)+(R0)X-(R0)F[+(R0)ZX]][-(R0)X+(R0)+(R0)F-(R0)X]
		Z=[+(R0)F-(R0)X-(R0)F][+(R0)+(R0)ZX]

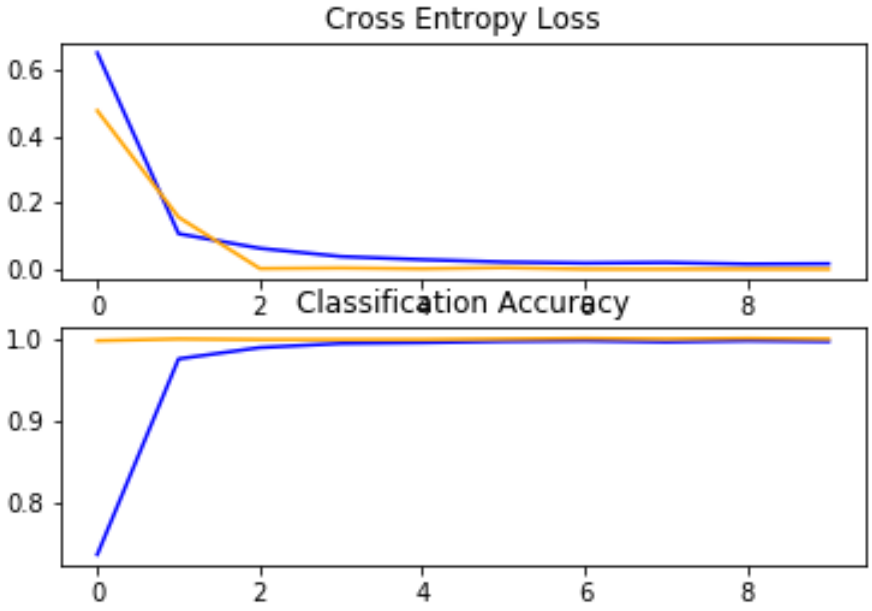


Figure 5.2: The loss and accuracy during the first training process. Blue lines indicate training data. Orange lines indicate test data.

optimization [33]. A higher learning rate might skip the best solution, and a lower learning rate will have a slower update rate. Since our images are not complicated compared to real-world photos, we set the learning rate to 0.0001 and run for more epochs. During the training process, these images will be fed to CNN in batches. Larger batch sizes requires more memory space. Due to hardware restrictions, our GPU memory can only hold 32 images in one batch to train. We saved the models after each epoch so that we can easily ignore the ones after overfitting points. We let our model run for 10 epochs.

Figure 5.2 shows the cross-entropy loss and classification accuracy during this training. Both loss and accuracy converged instantly after three epochs. The final test loss is 7.59×10^{-5}, and the final test accuracy is 99.9286%.

To understand how the CNN model works, we picked six images, one of each species from the training and test data set and made several modifications. The original images, shown in Figure 5.3, are black strokes on a white background. The examples of modified images are shown in Figure 5.4. We modified the colours of the strokes and background to be white, grey, and black. Then we let the trained model classify these modified images. Since there is no overfitting point observed from the Figure 5.2, we picked the last model to do this experiment. The result is shown in

Image 1 Image 2 Image 3

Image 4 Image 5 Image 6

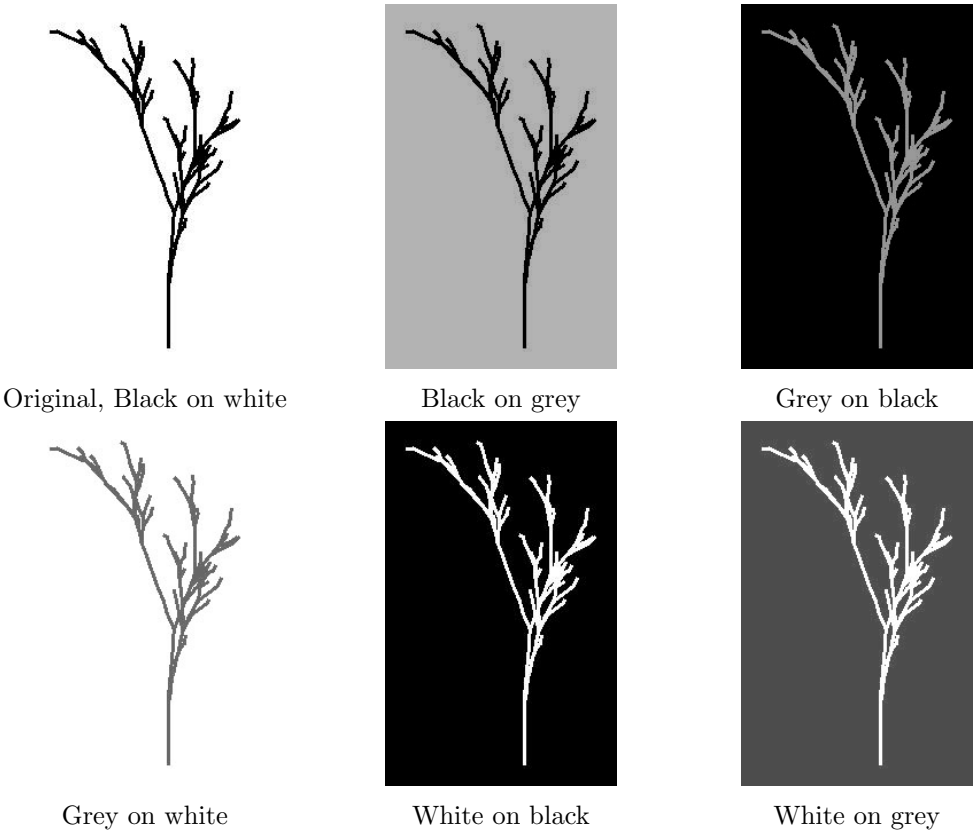
Figure 5.3: Original images before modification. Images in first row are from training set. Images in second row are from test set. Image 1, 4 belong to Species 1. Image 2, 5 belong to Species 2. Image 3, 6 belong to Species 3.

Table 5.3. From these results, it seems like the model classifies the images based on the density of black. Since the second species has the highest density in training, and the first species has the least density, the model classifies almost all the black background images to the second species. This model also made mistakes in grey and white pairs. Since Species 3 has the density between the first and second species, the model marks all the grey and white pairs to Species 3.

However, we created these "error" images on purpose to verify the behaviours of our CNN model. The system that integrates the L-system language in GP will not create images like these images. Even if model is struggling with these modified images, it might not have significant impact on our final system.

Table 5.3: The experiment results of modified colour of strokes and background.

Image	Original image belong to	Stroke	Background	Actual	Predicted
1	train	black	white	1	2
2	train	black	white	2	2
3	train	black	white	3	3
4	test	black	white	1	2
5	test	black	white	2	2
6	test	black	white	3	3
1	train	black	grey	1	2
2	train	black	grey	2	2
3	train	black	grey	3	3
4	test	black	grey	1	2
5	test	black	grey	2	2
6	test	black	grey	3	2
1	train	white	black	1	2
2	train	white	black	2	2
3	train	white	black	3	3
4	test	white	black	1	2
5	test	white	black	2	2
6	test	white	black	3	2
1	train	white	grey	1	3
2	train	white	grey	2	3
3	train	white	grey	3	3
4	test	white	grey	1	3
5	test	white	grey	2	3
6	test	white	grey	3	3
1	train	grey	black	1	2
2	train	grey	black	2	2
3	train	grey	black	3	2
4	test	grey	black	1	2
5	test	grey	black	2	2
6	test	grey	black	3	2
1	train	grey	white	1	3
2	train	grey	white	2	3
3	train	grey	white	3	3
4	test	grey	white	1	3
5	test	grey	white	2	3
6	test	grey	white	3	3



Original, Black on white | Black on grey | Grey on black

Grey on white | White on black | White on grey

Figure 5.4: The examples of modified images using Image 1 from Figure 5.3 as original image.

5.1.3 Experiment: Integrating CNN into GP

Although the experiment above has shown that the first trained model might not be ideal, we still want to know how it performs with the entire system. Since our CNN is built on TensorFlow using Python, and the GP system we used is Java-based, we built a local Python server using Flask framework [3] to host an API receiving images and returning predicted results. The fitness function of GP will perform a POST request to the Python server by sending the image rendered from the L-system and receive the predicted result. However, the ECJ requires standardized fitness such that 0 is the best individual, whereas the CNN returns a value from 0 to 1 where 1 is the best. Equation 5.1 shows the formula we used. This equation will convert the value to the range of 0 to 100, where 0 is the best.

$$\textit{Standardized fitness} = (1.0 - \textit{Prediction from CNN}) * 100 \tag{5.1}$$

In GP, various parameters can be adjusted. We inherited the parameters file from the ECJ's embedded Koza-style generational GP system [44]. We adjusted these parameters to fit our application. The GP parameters are in Table 5.4. Undoubtedly, although there might be a better settings than ours, these work well and can produce acceptable results. We have discussed the GP language in Chapter 4. Table 4.3 shows the language we used for this experiment. The setup for the L-system remains unchanged from Section 5.1.1 except toggling off the random branching. To generate different categories, we simply use different neurons' values from the output layer of CNN model. For each category, we let the system generate 100 images. Figure 5.8 shows several examples of the output using this setup. All the images are generated within the first generation. This means our GP is just acting as a random expression generator in this experiment.

After obtaining the results, we can see there are some tree-like patterns, but they are a small portion of the results. From our observations, the percentage of tree-like patterns in Species 1 is 0%, in Species 2 is 1%, in Species 3 is 5%. The total trees generated from the system using this setup is 2%. The rest of the images in the Species 1 are mostly blank or extremely simple, like the last images in the first row shown in Figure 5.8. For the images in Species 2, some of them are extremely simple similar to Species 1, but most of the images are very complex. The images in Species 3 have more variations than the previous two. They tend to form several tree-like patterns. Since the L-system is a very comprehensive, general system, it is capable of generating many different patterns, and a small portion of these patterns are visually

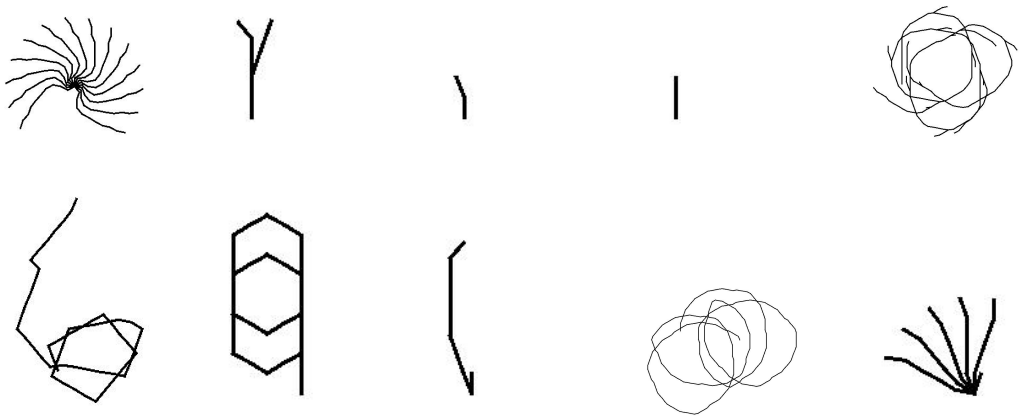


Image generated for Species 1.

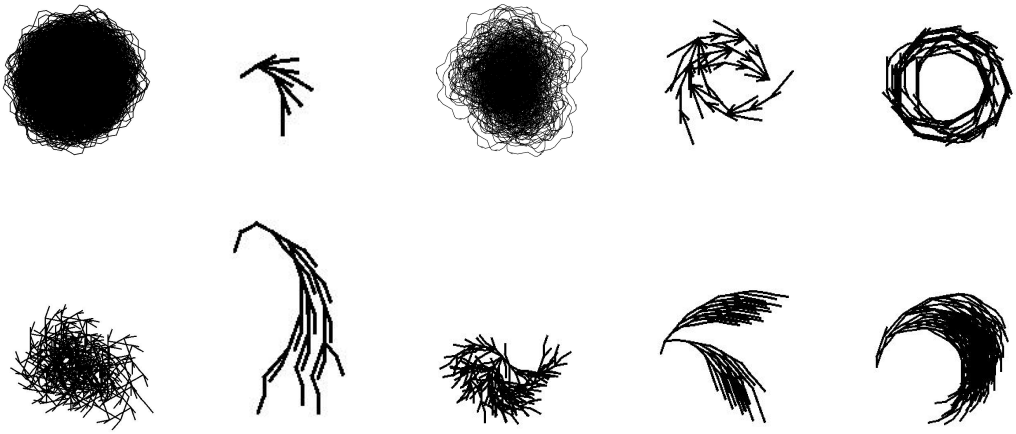


Image generated for Species 2.

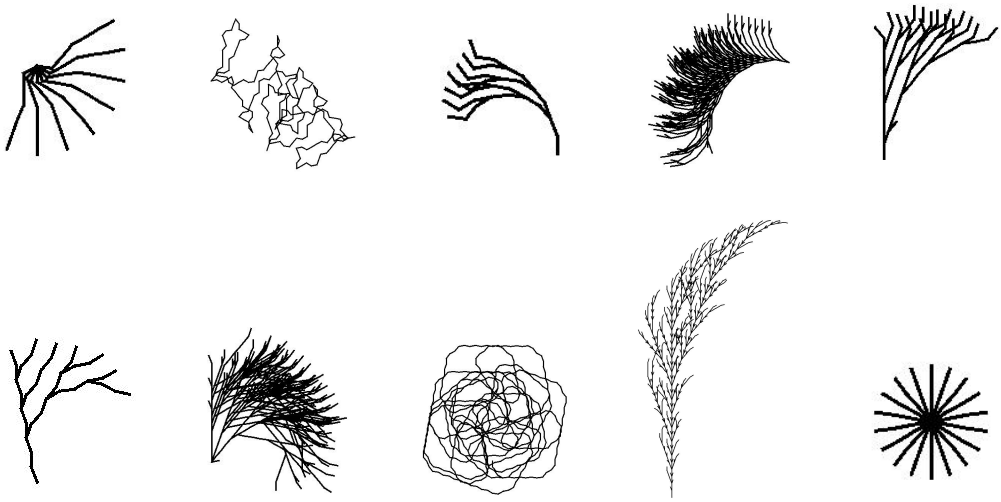


Image generated for Species 3.

Figure 5.8: Images generated from the system using first trained CNN model based on VGG16.

Table 5.4: The parameters used in the GP.

Parameter	Value
Population size	1024
Maximum generation	50
Maximum tree size	17
Crossover rate	90%
Mutation rate	10%
Tournament size	3
Elitism size	5
Tree initialization method	half & half
Initialization grow probability	0.5
Initial tree depth	2 - 5

interpretable as trees. Besides that, the GP explores a huge search space. Thus it might find many images that easily fool the CNN. We also tried adding more data to the training set. Although we increased the training set to 3000 images for each category, there was no significant improvement on the results. We also found that there is no direct relationship between the test accuracy of the CNN model when training and the acceptance rate when generating trees.

After having some ideas of how the VGG16 network works, we decided to modify the network to fit our application.

5.2 Experiment 1: Modified CNN model

We used our own CNN architecture to build the system in the second experiment. Figure 4.2 shows the architecture of the network. We compiled the architecture described in Table 5.5. Since the appearance of our tree images are very similar to the images used in handwritten recognition, we use ideas from LeNet-5 [39] which is a CNN application to recognize documents.

In Table 5.5, Layer 1 and 3 are used to extract the features from the input images. Layer 2 and 4 are used to subsample the data from the convolution to reduce the size of feature maps. Layer 5 is the first dropout to prevent the overfitting after the convolutional layers. Layer 6 simply flatten the output from the convolutional layers into sequential, so that it can be processed by the following fully connected layers. Layer 7, 10, 13 are the fully connected layers. We added dropout and batch normalization layers between every two fully connected layers to prevent the overfitting and stabilize

Table 5.5: Our re-designed CNN architecture.

	Layer	**Output Shape**	**# of neurons**	**Activation function**	**Note**
1	Conv2D	(222, 222)	128	ReLU	kernel size: (3, 3)
2	MaxPooling2D	(111, 111)	128		pool size: (2, 2)
3	Conv2D	(109, 109)	32	ReLU	kernel size: (3, 3)
4	MaxPooling2D	(54, 54)	32		pool size: (2, 2)
5	Dropout	(54, 54)	32		dropout rate: 0.25
6	Flatten	(93312)			
7	Dense	(256)	256	ReLU	
8	Dropout	(256)	256		dropout rate: 0.5
9	BatchNorm	(256)	256		
10	Dense	(128)	128	ReLU	
11	Dropout	(128)	128		dropout rate: 0.5
12	BatchNorm	(128)	128		
13	Dense	(3)	3	Softmax	loss: categorical cross entropy

the gradient.

The rest of the system remains the same as the first experiment we did in Section 5.1. Figure 5.12 shows some results from this setup. This model has similar performance to the previous experiment. After visual inspection, 1% of the results in Species 1, 3% of the results in Species 2, and 2% of the results in Species 3 can be considered to look like trees. This ends up with the total acceptance rate to be 2%.

In this experiment, the GP always find the optimal individuals in the Generation 0 since the CNN gives high scores to the these images. The GP does not evolve the individuals and works like a random search.

Since there are several spiral-like images, for example, the second image of the Species 1 in Figure 5.12, the non-tree images have some similarities as well. We decided to take a step back to teach the CNN to distinguish if a pattern is a tree or not.

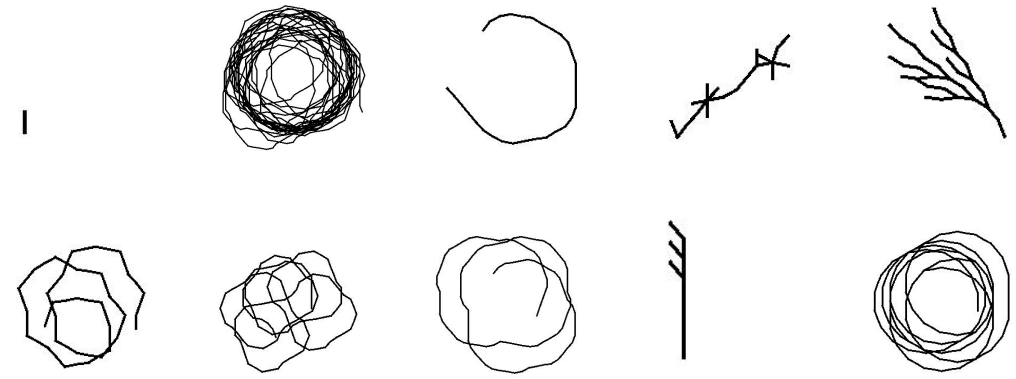


Image generated for Species 1.

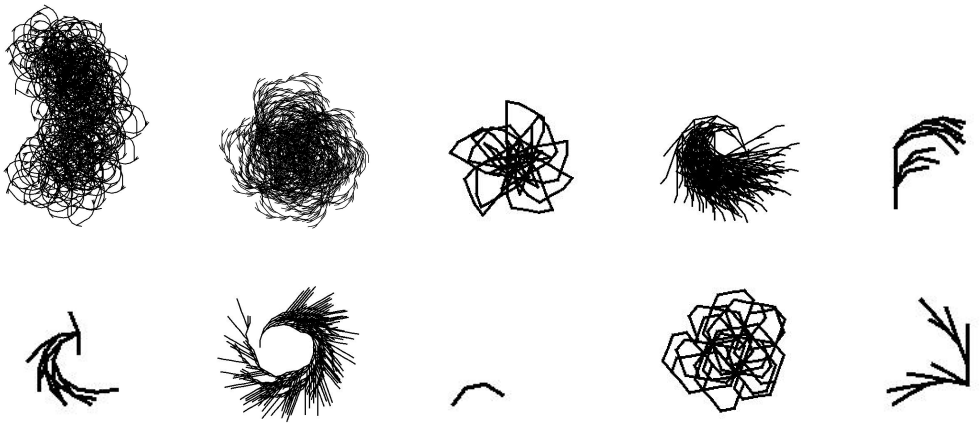


Image generated for Species 2.

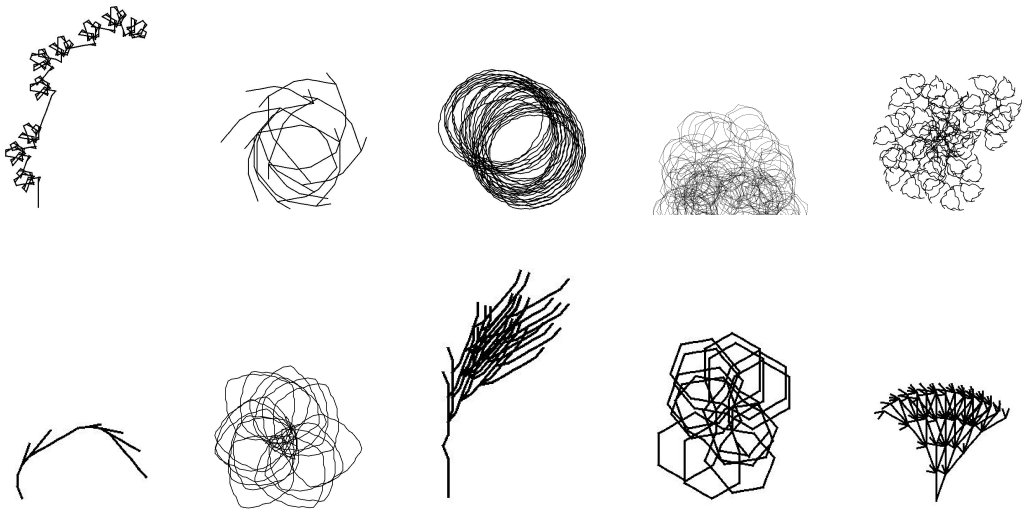


Image generated for Species 3.

Figure 5.12: Images generated from the system in the second experiment.

Table 5.6: Tree vs. non-tree CNN architecture. Red text is the difference from Table 5.5.

	Layer	Output Shape	# of neurons	Activation function	Note
1	Conv2D	(222, 222)	128	ReLU	kernel size: (3, 3)
2	MaxPooling2D	(111, 111)	128		pool size: (2, 2)
3	Conv2D	(109, 109)	32	ReLU	kernel size: (3, 3)
4	MaxPooling2D	(54, 54)	32		pool size: (2, 2)
5	Dropout	(54, 54)	32		dropout rate: 0.25
6	Flatten	(93312)			
7	Dense	(256)	256	ReLU	
8	Dropout	(256)	256		dropout rate: 0.5
9	BatchNorm	(256)	256		
10	Dense	(128)	128	ReLU	
11	Dropout	(128)	128		dropout rate: 0.5
12	BatchNorm	(128)	128		
13	Dense	(2)	2	Sigmoid	loss: binary cross entropy

5.3 Experiment 2: Tree vs non-tree model

5.3.1 CNN structure and data pre-processing

Since there are still too many non-trees, so we want to differentiate trees vs. non-tree. Thus we modified the network to fit the binary classification. Basically, we changed the number of neurons in the output layer to 2, the activation function of the output layer to sigmoid, and the loss function to binary cross entropy. Table 5.6 shows the architecture of the tree vs. non-tree model. The only modification we made is Layer 13.

For the data set, we chose 1000 tree images which are randomly picked from Species 1, 2, and 3. For the non-tree images, we first filtered out the tree-like patterns from the output of the previous images. Then we randomly picked 1000 non-tree images from the remaining.

We let the training versus test ratio to be 9:1. We randomly picked 900 images from both tree and non-tree categories to form the training set, and 100 images from these two categories to form the test set.

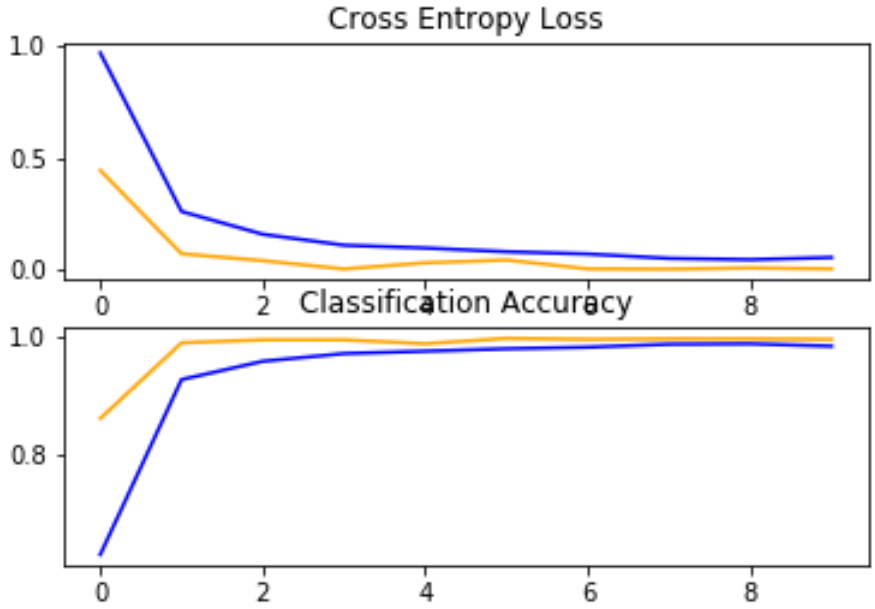


Figure 5.13: The loss and accuracy during the tree vs. non-tree model training process. Blue lines indicate training data. Orange lines indicate test data.

5.3.2 Train and test standalone tree vs non-tree model

We set the number of epoch to be 10. We got the loss and accuracy plot in Figure 5.13. From the plot, we found that it converged at Epoch 3. To prevent overfitting, we used the model after the Epoch 3 to perform testing.

The single tree vs. non-tree is not intended to be a fitness evaluator. But we were curious to use it as such. We replaced our Species CNN evaluator with the standalone tree vs.non-tree evaluator and let the GP generate just trees. After 100 runs, some of the results are shown in Figure 5.14. We can see that the single tree vs. non-tree model is inadequate as a fitness evaluator for evolving tree-like images, although we still can find some tree-like structures in these results. Instead, we decided to use the standalone tree vs. non-tree model to support our Species model during the evaluation.

5.3.3 Experiment 3: Dual CNN model

Now we have two models that are running simultaneously, we must modify the fitness function of the GP to fit this setup. Now GP will receive 2 values from 2 CNN models for each individual. Since the tree vs. non-tree model will send a probability to GP,

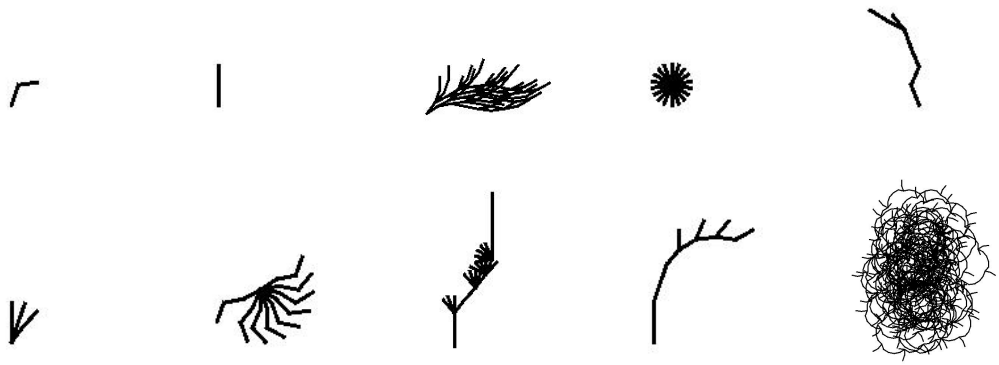


Figure 5.14: Images generated from the system using standalone tree vs. non-tree model.

and higher values means this image has a higher possibility to be a tree, we simply multiply this probability to the probability from the model we used in Section 5.2. Equation 5.2 shows how the new fitness is calculated.

$$\textit{Raw fitness} = \textit{Probability to be certain species} \times \textit{Probability to be tree} \tag{5.2}$$

From this equation, in order to get a higher fitness, both of these two probabilities must be high. Similar to Equation 5.1, we have to adjust the raw fitness to standardized fitness so that the ECJ can accept it:

$$\textit{Standardized fitness} = (1.0 - \textit{Raw fitness}) * 100 \tag{5.3}$$

Due to running two CNN models and GP at the same time, the first change of the system we found is the execution time to generate one image takes longer. It took around 4 to 5 minutes to generate one image on average. Because of that, we ran the system 20 times for each species to reduce the execution time and still keep the results statistically valid. Figure 5.18 shows the part of the results of this setup.

From these output images, we found that this setup works well when generating trees for Species 3. 34% of the output images for Species 3 can be considered as trees. However, this model did not perform very good for Species 1 and 2. The tree-like pattern we got for Species 1 is 2%, and Species 2 is 4%. The total acceptance rate for this setup is 13%. But again, these images are all generated from Generation 0.

To verify that we made the decision to choose the model after Epoch 3 is reasonable, we picked the model after Epoch 9 and did the same test again. The total acceptance rate dramatically drops to 4%. This result proves that the model is over

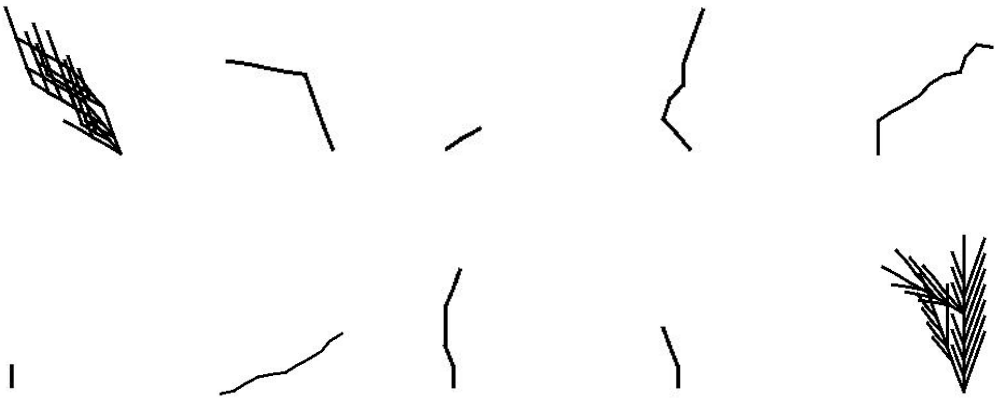


Image generated for Species 1.

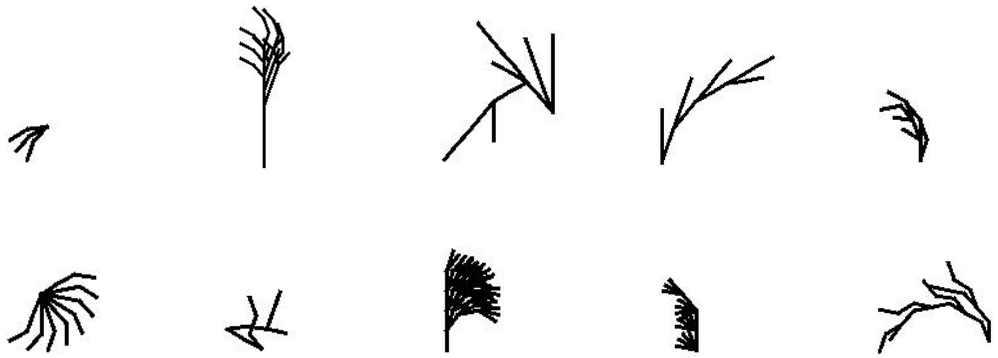


Image generated for Species 2.

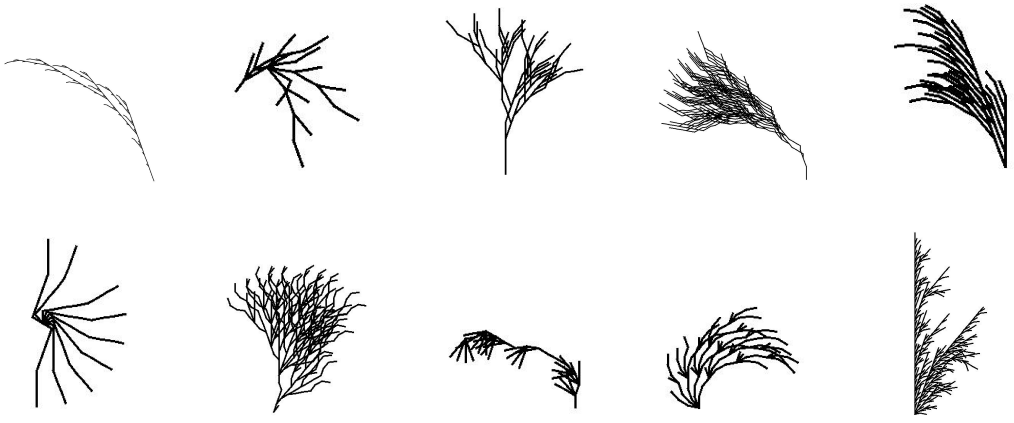


Image generated for Species 3.

Figure 5.18: Images generated from the system using both tree vs. non-tree model and Species model.

Table 5.7: Combined tree vs. non-tree model with species model. Red text is the difference from Table 5.5.

	Layer	**Output Shape**	**# of neurons**	**Activation function**	**Note**
1	Conv2D	(222, 222)	128	ReLU	kernel size: (3, 3)
2	MaxPooling2D	(111, 111)	128		pool size: (2, 2)
3	Conv2D	(109, 109)	32	ReLU	kernel size: (3, 3)
4	MaxPooling2D	(54, 54)	32		pool size: (2, 2)
5	Dropout	(54, 54)	32		dropout rate: 0.25
6	Flatten	(93312)			
7	Dense	(256)	256	ReLU	
8	Dropout	(256)	256		dropout rate: 0.5
9	BatchNorm	(256)	256		
10	Dense	(128)	128	ReLU	
11	Dropout	(128)	128		dropout rate: 0.5
12	BatchNorm	(128)	128		
13	Dense	(4)	4	Softmax	loss: categorical cross entropy

trained at Epoch 9, and the use of model after Epoch 3 is reasonable.

5.4 Experiment 4: Combined tree/non-tree model and species model

The system setup we used in Section 5.3.2 shows some improvement on one species, but the rest are still not good. Since we trained these two models separately, they evaluate individuals alone and cannot share information. This might cause the inefficiencies during both evaluating and training. So we decided to merge two CNN models into one. Due to the architecture we used for standalone tree vs. non-tree model is highly similar to the architecture we used in Section 5.2, they can be easily merged together by just changing the output layer. Table 5.7 shows the new architecture of our CNN. The difference from Table 5.5 is marked as red. There are total of 4 categories in this model: 3 species and non-tree.

There are 3000 tree images for each species but only 2441 non-tree images available. To prevent training bias, the amount of the data in each category must be the same. The strategy we used to balance the data is called over sampling. It randomly

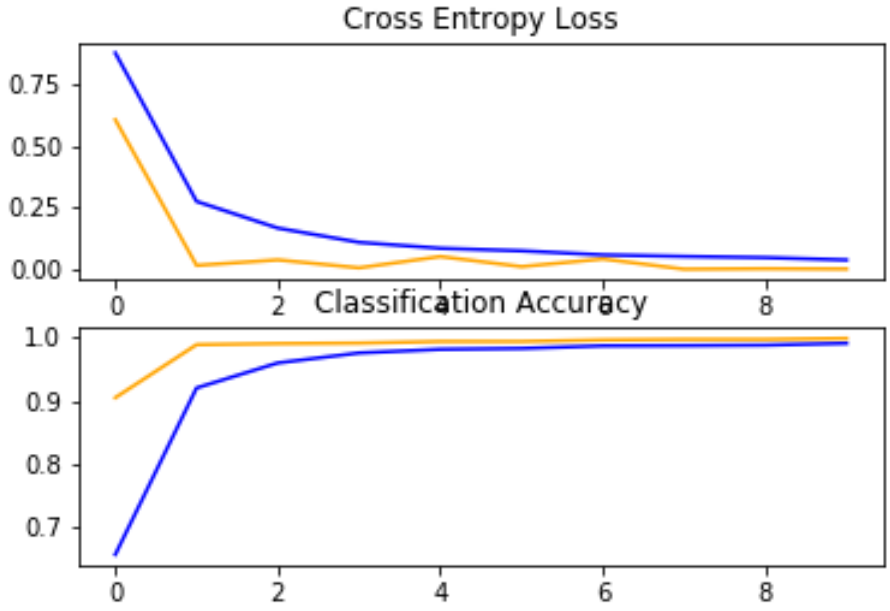


Figure 5.19: The loss and accuracy during the combined model training process. Blue lines indicate training data. Orange lines indicate test data.

picks several existing data and makes copies until the total amount of the data reach the target. At the end, we have 3000 images for each category. The training and test ratio is kept to 9:1, so there are 2700 images in training set and 300 images in test set for each category.

After training the CNN model, we plot out the loss and accuracy graph in Figure 5.19. The final test accuracy is 99.9167%, and the test loss is 3.789×10^{-4}. For GP fitness evaluation, we combined two CNN models to one, thus we changed the fitness function back to the one we used in Section 5.1.

Figure 5.23 shows part of the results from this setup. According to the results, this model performs well for all 3 species. 60% of the output for Species 1, 40% of the output for Species 2, and 50% of the output for Species 3 can be considered as trees. This comes up with the total acceptance rate of 50%. This is currently the best setup among all the experiments we did. Table 5.8 shows some L-system expressions that can be rendered to some images shown in Figure 5.23.

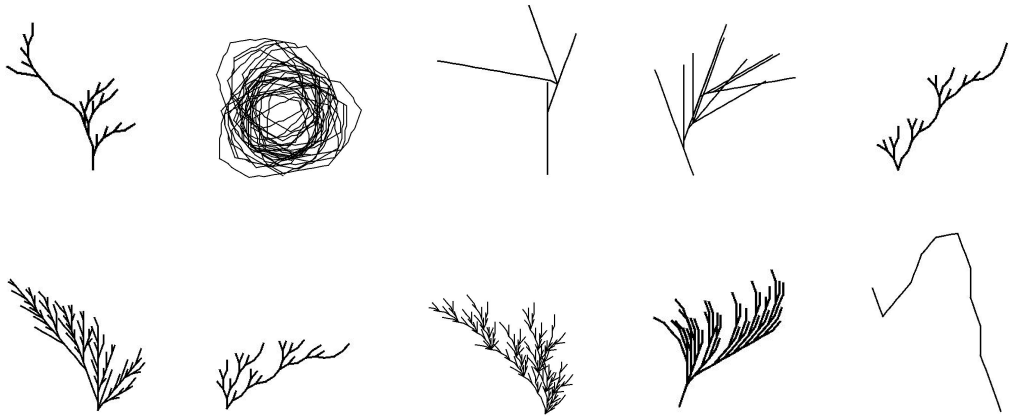


Image generated for Species 1.

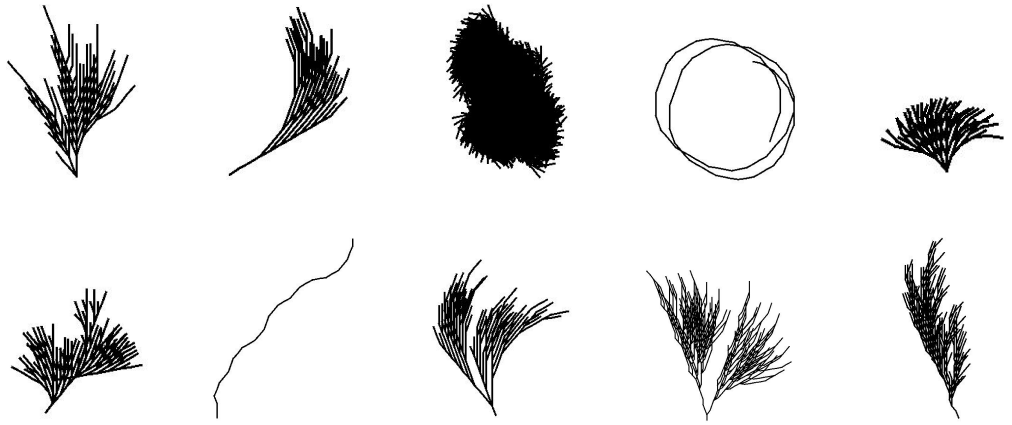


Image generated for Species 2.

Image generated for Species 3.

Figure 5.23: Images generated from the system using combined tree vs. non-tree and species model.

Table 5.8: The L-system expressions generated from experiment 4. These expressions are related to the first row of each species in Figure 5.23.

Tree	Species	Expression
1	1	FXXF++-FX-+FXF+F+FF+--F+--++-F+XX+[-]F[-]F[+]-F--
		X=[-F--[+]++XXXX+F]
2	1	++XFXX+X--+--FFFXXF+-F+F[-]-+++
		-=+FF-F++-F+X-FXF
3	1	XX-X+-++[XX-]++F+XX
		X=FX[XF-]
4	1	+XFFF+-+X-XX+X-FX--+XX-+F-FF-+-F[-F+++X+]+[F]++-+ -F-FFF--F-FX+XX+-XF-X-+XX++-
		F=[[[F]F--X]]F[FXXFFX+]-[-]X[+]+[X+-]
5	1	X-F
		X=[+FX]X-XF+[+]
6	2	[F]X++[FFF]FXF-
		+=X+[X-FX+X-F-++XXF-+F-[+]+F]
7	2	---FX-XX+X+-+X-XF
		F=XF[[X]FXF+FF]
8	2	[X]FXXFX-FXFF+++[-]+XX+FXXFFX-X
		X=[X]+X+---+X+-+FX[F]+XX+X[X++]
9	2	-+F
		+=F+-+FF+
10	2	-X+-X+[-]++
		+=X+[[FXX]X[XF-X++XX-]-F-[F]+-XF]
11	3	FX[-]FF+-
		F=XF-XFF++-X-X+X+X+--+[-XFF---F+]
12	3	+-X+X--FFF+
		X=[FFX]X+X++X
13	3	-+---X+-F-+F[X]X[[[X]]]
		X=[XXFF[F]XXFF-++FX[-XFF+X+]]
14	3	+XF-X
		-=[XF+XXFX+[F]-+-+]-[XFF+FF-]X[+]F[+]+F
15	3	X--
		X=[[F]FF+X--+F+-FFFXF+XFFFX--FFXX[+XX]+[-]]

5.5 Experiment 5: Extended model

Once we have the combined model, the non-tree data set has a lot of variations. Some of the non-tree patterns are very simple, some are very complex, and some are in between. We decided to explore further of the CNN model to see if we can improve the results. We split the non-tree category into two categories: simple non-tree and complex non-tree. This turns the total number of categories to 5: 2 non-tree and 3 species. Thus, we have to modify the output layer of CNN to contain 5 neurons.

Since the data "in-between" are very few, we simply ignore this small portion of the data. We manually split the remaining non-tree patterns into simple non-tree and complex non-tree. After that, there are a total 1623 images in simple non-tree category and 755 images in complex non-tree category. However, there are 3000 images for each species, and so we have to use the same strategy used in Section 5.4 to balance the data. However, over sampling does not work well when the major part of the data needs to be filled. So we decided to set the target to 1200 images for each category. If there are more than 1200 images in the category, these images will be randomly picked from the data set. If there are less than 1200 images in the category, over sampling will be used to fill images to the target amount.

The training and test ratio is kept to 9:1, and so there are 1080 images in the training set and 120 images in the test set for each category. After training the model, we plot out the loss and accuracy graph in Figure 5.24. According to the plot, the cross entropy loss on test set is fluctuating during training.

We kept the rest of the system the same as in Section 5.4. After executing 20 times for each species, part of the results are shown in Figure 5.28. The acceptance rate for Species 1 is 0%, Species 2 is 15%, and Species 3 is 5%. Total acceptance rate is 7%.

To prove that this result is not caused by the limited amount of training data, we set the target to 1500 images for each category and re-train the CNN model. After running the entire system again, the result remains similar. From these results, we can conclude that this strategy was not beneficial.

5.6 Discussion

After performing several experiments, we composed the results into Table 5.9. Experiment 6, which is the very first experiment, is discussed in Section 5.1.3. We also include the average generation producing the final results for each experiment. For

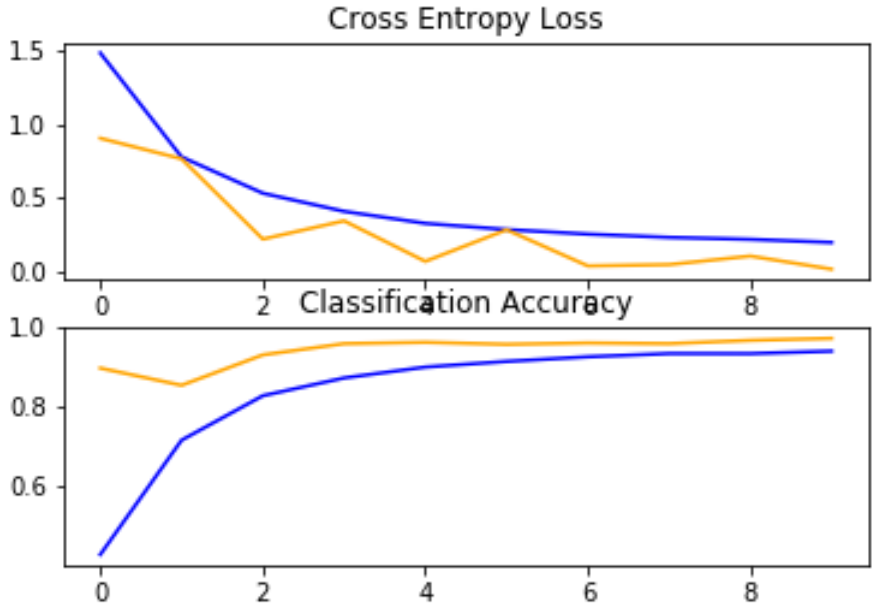


Figure 5.24: The loss and accuracy during the extended combined model training process. Blue lines indicate training data. Orange lines indicate test data.

experiment 1, 2, 3 and 6, we did 100 runs for each species. For experiment 4 and 5, we performed 20 runs for each species due to the slower execution speed.

According to the results, experiment 4 obtains the best acceptance rate in all Species. From our evaluation of image quality, the 4-output CNN performs best among all strategies. Although the standalone tree/non-tree model does not work well in experiment 2, experiment 3 shows the tree/non-tree model is useful when working with the species model. However, comparing experiment 4 with experiment 3, we found that the independent Species model and tree/non-tree model are mutually exclusive. By giving the tree/non-tree portion ability to share information with the species portion, the combination of both becomes more accurate.

The last 2 experiments in Table 5.9 did not perform well. As explained earlier, our initial idea to do the experiment 5 is due to the pattern of the false-positive images generated from experiment 4. These images can generally falls into 2 categories: simple non-tree and complex non-tree. We thought it might help the CNN if we split the non-tree category into these 2 new categories. However, this strategy is worse than the 4-output one.

Since the experiment 4 achieves the best results, we calculated the average best fitness of these 20 runs and plotted its average fitness for each species in Figure 5.29.

Image generated for Species 1.

Image generated for Species 2.

Image generated for Species 3.

Figure 5.28: Images generated from the system using extended combined tree vs. non-tree and species model.

Table 5.9: Summary of the results from all the experiments.

	CNN strategy	Test loss	Test accuracy	Acceptance rate(%)				Avg. generation solutions
				Species			Avg.	
				1	2	3		
1	Species	0.0001	0.9929	1	3	2	2	0.0
2	Tree/non-tree	0.0139	0.9811	-	-	-	2	0.0
3	Dual CNN	-	-	2	4	34	13	0.0
4	4-output	0.0020	0.9983	60	40	50	50	24.47
5	5-output	0.2402	0.9733	0	15	5	7	8.88
6	VGG16	0.0002	0.9979	0	1	5	2	0.0

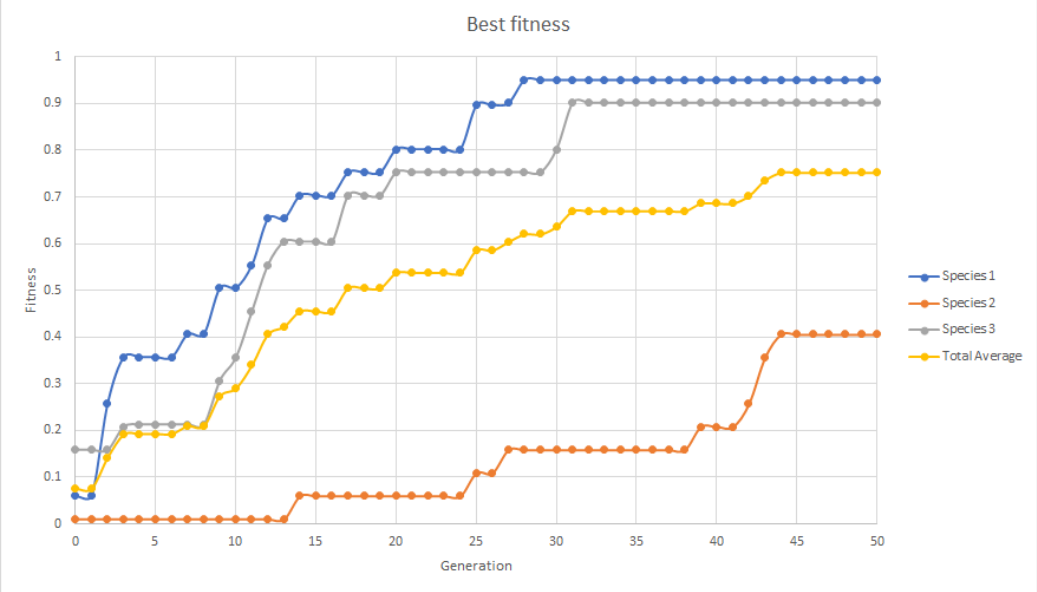


Figure 5.29: Average best fitness plot for 20 runs of each species in experiment 4.

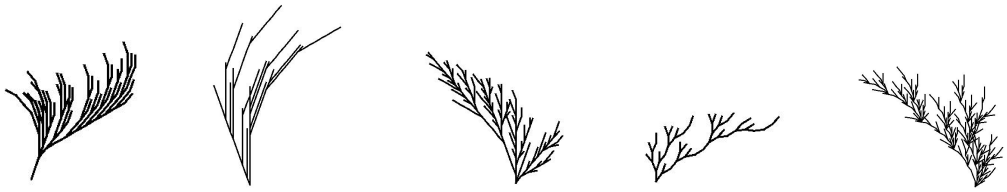


Image generated for Species 1.

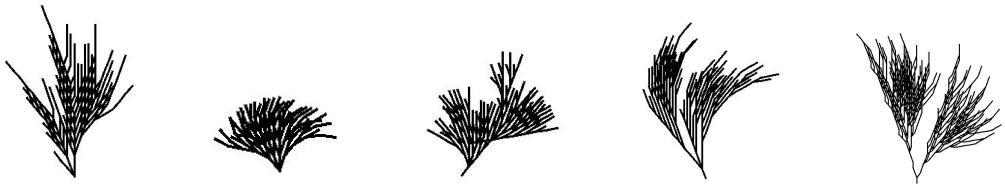


Image generated for Species 2.

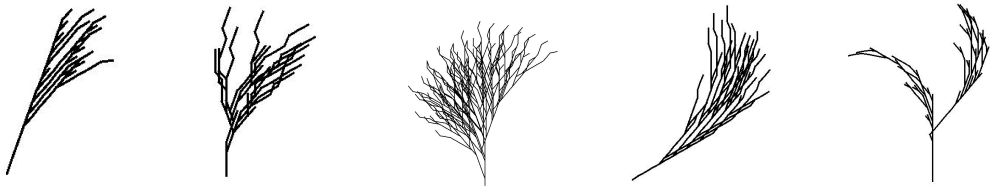


Image generated for Species 3.

Figure 5.33: Some of successful results chosen from the experiment 4 in Table 5.9

Comparing this graph with the data in Table 5.9, we found the species that have a higher acceptance rate tends to have higher fitness score. Figure 5.33 shows some of our successful results generated by experiment 4 in Table 5.9. Most of the results can be categorized into their own Species. In Table 5.8, we listed the expressions corresponding to the first row of each species in Figure 5.23. Due to the sensitivity of L-system expressions, there is no obvious similarity among these expressions or the expressions we used to generate the target images, even they belongs to the same species. With a given expression, it is impossible that human can distinguish whether it is tree or not without rendering it.

We extended the size of the data set and retrained the networks for each setup as well, but the results become even worse than the networks trained with small data set. From our understanding, this effect might be caused by the variations of our data. Adding more data will increase the variation within each categories. Due to the scale of our network, the high variation among images in the same category can cause the model to be unstable and make poor predictions.

In summary, our system can be considered as capable of generating trees from scratch by the guide of CNN. As the statistics shows in Table 5.9, experiment 4 achieves 50% acceptance rate, and still has divergent creativity.

5.6.1 Comparison between species

As discussed earlier, some of our training images are shown in Figure 5.1. The branches of species 1 trees are very sparse and well-separated, while the branches of species 2 are more thick and dense. And the species 3 are in between, which will have more overlapping branches than species 1 but less than species 2. Comparing the results we obtained from experiment 4 with these training images, these characteristics of each species still exist.

5.6.2 Other observations during the experiments

When testing our system, we encountered a bug (X-bug) that encouraged the system to generate more promising results in some experiments. That bug did not allow the placeholder X to be rewritten properly, and it had no effect in expressions. In other words, the buggy L-system language was less expressive than it should have been. Figure 5.37 shows the results when experimenting the 4-output model with the X-bug.

From the results, we found that this bug pushed the total acceptance rate to

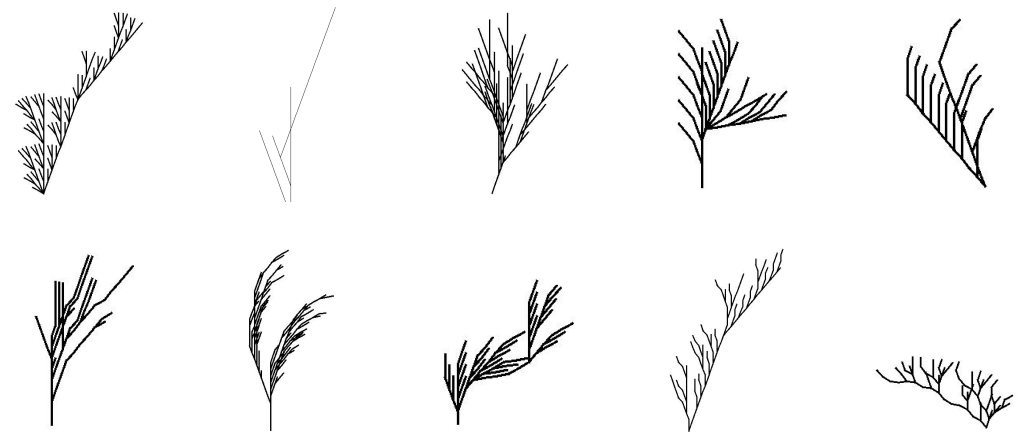


Image generated for Species 1.

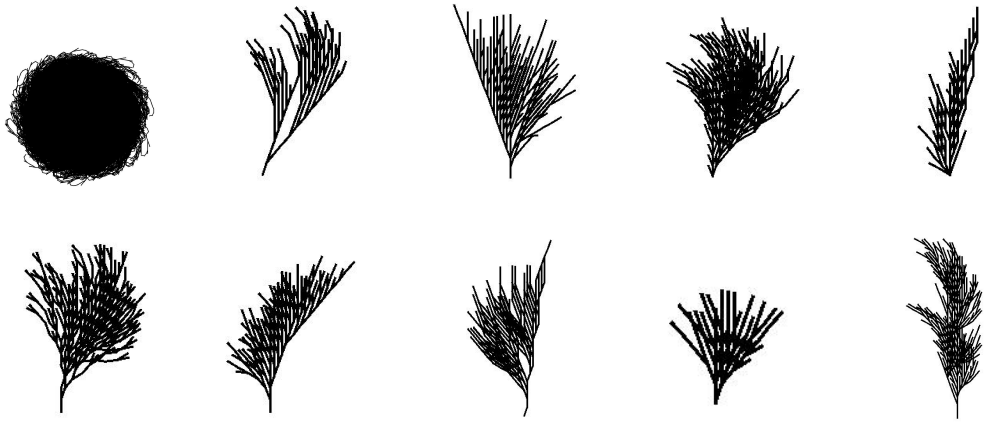


Image generated for Species 2.

Image generated for Species 3.

Figure 5.37: Images generated from the system with the X-bug using 4-output CNN model.

58.33%. This is even better than the results we got from the system with the bug fixed. We do not have a solid reason about what caused this phenomenon, but we thought the X-bug gives the system chance to create bloat. In the DNA of living creatures, there are also a lot of genes that are not translated into protein sequence [15]. The bloat created by the X-bug is like these unexpressed genes in the creatures' DNA. Since there is no other way that our system can create bloat, it might help the evolutionary system in some ways. However, this requires more investigation to figure out.

Chapter 6

Conclusion

This research explores the possibility of using a CNN as a guide for GP when evolving L-system rendered images. The wide potential of L-systems gives GP enough space for creating different variety of images. Our approach automates the process of generating desired trees images. There are several research that inspired us. There is also a lot of improvements possible for future work.

As we discussed in Section 3.2, there are several researchers combined evolutionary algorithms and L-system. Congdon *et al.* introduced the GenTree which is using interactive genetic algorithm. Since the system requires human to be part of it, they obtained good results. However, they focus on making 3D polygonal trees, and their system is specifically designed to build trees. Since our use of a general L-system creates a larger search space for GP, it can cause more challenges, but the trees generated from our system are more diverse.

FranksTree [20] used an L-system like ours. Instead of generating trees from scratch, their system requires humans to pick several trees to be the initial generation and then applies the genetic operations to these known trees. Thus, their results combine the features from the original generation and highly related to these original trees picked by the users. This lets the system evolve tree images right away, thanks to human interaction. Our system must use CNNs to find tree images. However, some of their results are not convincing biological trees. For example, one result from their system connect three trees in series, which makes it looks like there is a tree growing from a leaf of another tree.

Genetic L-system Programming [32] uses a pattern matching function for the evaluation process. Since this is a very early research, the complexity of results from the system is minimal.

Bergen and Ross [18] built a L-system that use GP to evolve 3D models. Since

their system is not designed for generating trees, but rather aesthetic 3D models, their results are not trees. They did use a general purpose 3D L-system, however, and tree might be evolved on occasion.

We achieves a tree acceptance rate of 50% shown in Table 5.9, but keep the ability of the system to generating diverse plants automatically. Considering the generality of the L-system, trees are a very small portion of the entire search space. Comparing with the previous researches, our system can be considered as a success based on the acceptance rate and the diversity of the results.

6.1 Future work

Since most of our work focuses on the CNN training, we did not explore on GP design as extensively. Future improvements can be to design a more appropriate GP language, using some heuristic randomization to generate the first generation, or simply adjusting the parameters of GP to find a better setup. Our "buggy" L-system was beneficial for species generator, and so evolution is very sensitive to the nature of the L-system language used.

Besides that, our CNN has a lot of room to improve as well. Different architecture will have different benefit to the system. Changing the number of convolutional layers or the number of neurons in each layer will change the behaviour of the network. Besides that, using different optimizer or different activation functions will also affect the output of the network. Future researchers can design a better structure of CNN that can fit this application better.

For the L-system, it can be extended to 3D by introducing pitch operations. However, it requires to modify the GP language and CNN structures as well. We tried to use only 3D tree/non-tree model in the new system, but the results are not promising. We thought it might require the Species model as well to perform better. This needs more study.

Since CNN is known as translation-invariant, it can only capture local features. Future studies can focus on adding components to capture global features. Having this extra information might help the system to evaluate individuals better.

Another possible improvement can be considering unsupervised learning or reinforcement learning. Since CNN is known as a supervised learning algorithm, the training of CNN usually requires a considerable amount of labeled data. Labeling data can easily consume a huge amount of time. Unsupervised learning does not require labeling data. And reinforcement learning will use reward system and follow the

trial-and-error basis to learn making decision. An example of reinforcement learning is generative adversarial network (GAN) [30]. Future researchers can design a GAN to replace the CNN in our system.

Bibliography

[1] Deep art. https://deepart.io/. Accessed: 2020-06-21.

[2] Ecj 26. https://cs.gmu.edu/ eclab/projects/ecj/. Accessed: 2020-04-16.

[3] Flask. https://palletsprojects.com/p/flask/. Accessed:2020-12-18.

[4] Keras: The python deep learning library. https://keras.io/. Accessed: 2020-04-18.

[5] L-system generator. https://onlinemathtools.com/l-system-generator. Accessed: 2020-04-17.

[6] Large scale visual recognition challenge 2010 (ilsvrc2010). http://image-net.org/challenges/LSVRC/2010/index. Accessed: 2020-08-18.

[7] Logo foundation. https://el.media.mit.edu/logo-foundation/index.html. Accessed: 2020-06-18.

[8] The microsoft cognitive toolkit. https://docs.microsoft.com/en-ca/cognitive-toolkit/. Accessed: 2020-08-17.

[9] Nn svg. https://alexlenail.me/NN-SVG/LeNet.html. Accessed: 2020-08-17.

[10] Prisma lab. https://prisma-ai.com/. Accessed: 2020-06-21.

[11] Tensorflow. https://www.tensorflow.org. Accessed: 2020-04-18.

[12] theano. http://deeplearning.net/software/theano/. Accessed: 2020-08-17.

[13] Wolfram alpha: Computational intelligence. https://www.wolframalpha.com/. Accessed: 2020-12-18.

[14] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.

[15] P. Andolfatto. Adaptive evolution of non-coding dna in drosophila. *Nature*, 437(7062):1149–1152, 2005.

[16] S. Baluja, D. Pomerleau, and T. Jochem. Towards automated artificial evolution for computer-generated images. *Connection Science*, 6(2-3):325–354, 1994.

[17] P. J. Bentley and D. Corne. *Creative Evolutionary Systems*. Morgan Kaufmann, 2002.

[18] S. Bergen and B. J. Ross. Aesthetic 3d model evolution. *Genetic Programming and Evolvable Machines*, 14(3):339–367, 2013.

[19] A. Blessing and K. Wen. Using machine learning for identification of art paintings. Technical report, Stanford University, 2010.

[20] D. M. Bonfim and L. N. de Castro. Frankstree: A genetic programming approach to evolve derived bracketed l-systems. In *Advances in Natural Computation*, pages 1275–1278, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[21] P. Bontrager, W. Lin, J. Togelius, and S. Risi. Deep interactive evolution. *CoRR*, abs/1801.08230, 2018.

[22] Y. Boureau, J. Ponce, and Y. LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, Haifa, Israel, June 2010. Omnipress.

[23] E. Cetinic, T. Lipic, and S. Grgic. A deep learning perspective on beauty, sentiment, and remembrance of art. *IEEE Access*, 7:73694–73710, 2019.

[24] D. Cohen-Or and H. Zhang. From inspired modeling to creative modeling. *The Visual Computer*, 32(1):7–14, 2016.

[25] C. B. Congdon and R. H. Mazza. Gentree: An interactive genetic algorithms system for designing 3d polygonal tree models. In *Genetic and Evolutionary Computation — GECCO 2003*, pages 2034–2045. Springer Berlin Heidelberg, 2003.

[26] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[27] P. Galanter. Computational aesthetic evaluation: Past and future. In J. McCormack and M. d'Inverno, editors, *Computers and Creativity*, pages 255–293. Springer Berlin Heidelberg, 2012.

[28] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[29] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, volume 9, pages 249–256, Sardinia, Italy, May 2010. JMLR Workshop and Conference Proceedings.

[30] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *arXiv preprint arXiv:1406.2661*, 2014.

[31] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[32] C. Jacob. Genetic l-system programming. In *Parallel Problem Solving from Nature — PPSN III*, pages 333–343. Springer Berlin Heidelberg, 1994.

[33] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[34] J. R. Koza. Non-linear genetic algorithms for solving problems. United States Patent 4935877, 19 June 1990.

[35] J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, 1992.

[36] J. R. Koza and R. Poli. Genetic programming. In Edmund K. Burke and Graham Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pages 127–164. Springer, Boston, MA, 2005.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[38] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[39] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[40] J. Lehman, S. Risi, and J. Clune. Creative generation of 3d objects with deep learning and innovation engines. In *Proceedings of the 7th International Conference on Computational Creativity*, pages 180–187, Paris, France, June 2016.

[41] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[42] L. Loewe and W. G. Hill. The population genetics of mutations: good, bad and indifferent. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 365(1544):1153–1167, 2010.

[43] R. López de Mántaras. *Artificial intelligence and the arts: Toward computational creativity.* Fundación BBVA, 2016. https://www.bbvaopenmind.com/en/articles/artificial-intelligence-and-the-arts-toward-computational-creativity/.

[44] S. Luke. The ecj owner's manual. *A user manual for the ECJ Evolutionary Computation Library*, 2010. https://cs.gmu.edu/ eclab/projects/ecj/manual.pdf.

[45] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928, Hamburg, Germany, 2015. IEEE.

[46] W. S. Mcculloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[47] V. Nair and G. E Hinton. Rectified linear units improve restricted boltzmann machines. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of 27th International Conference on Machine Learning*, page 807–814, Haifa, Israel, 2010.

[48] C. Neufeld, B. J. Ross, and W. Ralph. The evolution of artistic filters. In J. Romero and P. Machado, editors, *The art of artificial evolution*, pages 335–356. Springer, 2008.

[49] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436, Boston, MA, USA, June 2015. IEEE.

[50] A. M. Nguyen, J. Yosinski, and J. Clune. Innovation engines: Automated creativity and improved stochastic optimization via deep learning. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, page 959–966, New York, NY, USA, 2015. Association for Computing Machinery.

[51] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, Berlin, Heidelberg, 1996.

[52] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, Honolulu, HI, USA, July 2017. IEEE.

[53] P. Y. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *7th International Conference on Document Analysis and Recognition*, volume 3, Edinburgh, Scottland, 2003. https://doi.org/10.1109/ICDAR.2003.1227801.

[54] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[55] R. Socher, B. Huval, B. Bath, C. D. Manning, and A. Y. Ng. Convolutional-recursive deep learning for 3d object classification. In *Advances in neural information processing systems*, pages 656–664, 2012.

[56] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, Boston, MA, USA, 2015. IEEE.

[57] D. Thornburg. Friends of the turtle. *Compute!*, (34):148, March 1983.