國立臺北科技大學

資訊工程系碩士班

碩士學位論文

Use Efficiency-based Genetic Programming to Create Loss Function for Image Classification Tasks

研究生：周子榆

指導教授：陳香君博士

中華民國一百一十四年五月

國立臺北科技大學

資訊工程系碩士班

碩士學位論文

Use Efficiency-based Genetic Programming to Create Loss Function for Image Classification Tasks

研究生：周子榆

指導教授：陳香君博士

中華民國一百一十四年五月

「學位論文口試委員會審定書」掃描檔

審定書填寫方式以系所規定為準，但檢附在電子論文內的掃描檔須具備以下條件：

1. 含指導教授、口試委員及系所主管的完整簽名。
2. 口試委員人數正確，碩士口試委員至少 3 人、博士口試委員至少 5 人。
3. 若此頁有論文題目，題目應和書背、封面、書名頁、摘要頁的題目相符。
4. 此頁有無浮水印皆可。

審定書不用頁碼

Abstract

Title: Use Efficiency-based Genetic Programming to Create Loss Function for Image Classification Tasks

Pages: (Fill it)

School: National Taipei University of Technology

Department: Computer Science and Information Engineering

Time: May, 2025

Degree: Master of Science

Researcher: Tzu-Yu Chou

Advisor: Shiang-Jiun Chen, Ph.D.

Keyword: Image Classification, Genetic Programming, Loss Function, Deep Learning

In the recent, with the rapid rise of deep learning, image classification models have quickly become one of the most popular and widely known models. When training such models, the steps are broadly divided into the following: preparing image datasets, dividing them into training, validation and testing sets as needed, training the model, evaluating the model, and repeating these steps until the ideal result is met or the computational resources are exhausted.

A crucial function during the process of training a model is called the loss function, which calculate the difference between the predicted values and the ground-truth values. The results of the loss function can significantly influence the effectiveness of the model's training because it simply decide the direction of the adjustment to the model. However, designing a loss function often requires the assistance of experts in the related field, leading to a resource-intensive design process. Recent research has proposed using Genetic Programming (GP) to generate loss functions to avoid the necessity of hiring numerous domain experts for assistance. Nevertheless, using GP typically results in decreased computational efficiency. This paper aims to improve the method of using GP to generate loss functions by modifying certain genetic operations and introducing the concept of tournament selection.

這邊要加結果

Acknowledgements

所有對於研究提供協助之人或機構，作者都可在誌謝中表達感謝之意。

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

After several decades of development, deep learning [1] technology has achieved significant breakthroughs in the last ten years, largely due to the remarkable improvements in GPU computational power. Consequently, the predictive and analytical capabilities of models have achieved substantial advancements. Numerous models have been launched by major companies and applied in practical scenarios, leading to significant changes in our daily lives.

When it comes to training deep learning model, we usually refer to the following steps: collecting relevant data and organizing it into datasets, partitioning the datasets into training and validation sets as needed, initializing model parameters, training the model, evaluating the performance of the model, adjusting the model parameters, and repeating the training until the target is achieved or computational resources are exhausted. During the evaluation of the model's performance, we use a function called the loss function [2]. Its purpose is to calculate the difference between the model's predicted results and the ground truth, which helps model adjust its parameter to better fit the target. Therefore, different loss functions can influence the direction of model adjustments, significantly impacting the final outcomes of the model.

However, the design of a loss function is often closely related to the propose of the model [3]. In other words, different models may require experts from specific fields to assist in designing the loss function to enhance the speed and effectiveness of model training. Nevertheless, recent research has shown that it's feasible to use genetic programming (GP) [4] to automatically generate loss functions. Due to the domain-independent nature of GP, it allows us to develop an algorithm that can automatically create the required loss function without needing specialized knowledge in that particular field. However, this method typically requires a significant amount of time and computational resources.

The operation mode of GP can be simply divided into the following steps: Representing the solution we hope to find (which may be a value or a function) in an encoded form, then forming these solutions into a population. After evaluating the whole population, we perform genetic operations (e.g., mutation or crossover) on the population's solutions, reevaluate them, and repeat the above steps until the target is achieved or computational resources are exhausted.

Although the computational power of GPUs today is far superior to that of the past, both training models and using GP to find solutions still require substantial computational resources and time to achieve a decent result. In this paper, an efficiency-oriented GP algorithm is proposed to reduce the computational resources and the time to train the model required for GP to search for the optimal algorithm while maintaining the same level of effectiveness. We aim to improve the genetic operation part of the existing GP framework by referencing the concept that offspring in a better living environment can usually receive better care. We modify the GP's genetic operation such that only a certain number of high-scoring populations can execute it. Here's what we want to do: Select a certain proportion of high-scoring offspring from the population, then randomly select a fixed number from these offspring to perform genetic operations. The randomized selection can avoid always using the same population for genetic operations, giving more excellent offspring the opportunity to improve.

這邊要加結果

The structure of this paper is as follows: Chapter 2 is related work. This chapter provides a detailed illustration to the development and history of GP, the detailed description of the loss function and how they collaborate with deep learning model, and how loss functions are randomly generated via GP. Chapter 3 is proposed algorithm and main methodology. In this chapter, we will explain the architecture of the algorithm implemented in this paper. After that, we will then describe the experimental environment setup and the initial parameter values. Finally, we will present the methods and procedures used in this paper. Chapter 4 is results analysis. This section will show the detail parameter settings among all compared peer-algorithm. After that, we will present the analysis of the results by organizing the experimental results into charts and figures. Finally, we will explain the outcomes and analyze why our algorithm can achieve the similar result while decreasing the usage of computational resources. Chapter 5 is conclusion and future work. In this chapter, we will provide the conclusion of this paper by outlining the contribution we have made so far. After that, we would like to discuss potential directions for future research or possible way to apply this algorithm in practical.

Chapter 2 Related Work

In the following sections, we will introduce the concept of metaheuristics [5, 6] . Then, we will discuss the importance of loss functions and their role in deep learning [1] models. Finally, we will explore image classification [7] and its real-life applications.

2.1 Metaheuristic

In the following section, we will discuss metaheuristic and some of the most famous metaheuristic algorithms include Evolving Algorithm (EA) [8], Genetic Algorithm (GA) [9], and Genetic Programming (GP) [4]. After that, we will introduce related research in the GP domain.

2.1.1 Overview of Metaheuristic

Metaheuristic [10] was first proposed by Glover. It can be regarded as a high-level procedure used to find solutions to optimization problems or other issues that conventional algorithms cannot solve. Within metaheuristics, an important concept is that these procedures must find a potentially optimal solution under reasonable computational costs or insufficient information. In this paper, we will primarily focus on nature-inspired metaheuristics [5]. In these methods, a common approach is to use a population-based strategy to find the optimal solution. Within a population, each individual typically represents a potential solution. We perform various operations on the individuals within the population to achieve our goal of approximating the optimal solution. A subset of specialized algorithms falls under population-based methods [11], commonly referred to as evolving algorithms (EAs). Commonly used methods within EAs include genetic algorithms (GA), genetic programming (GP) , evolutionary programming (EP) [12], and differential evolution (DE) [13]. The relationship between Metaheuristic, EA, GA, GP, EP and DE is illustrated in the figure 2.1. We will briefly discuss EAs and GAs, and then we will focus primarily on GPs in the following paragraph.

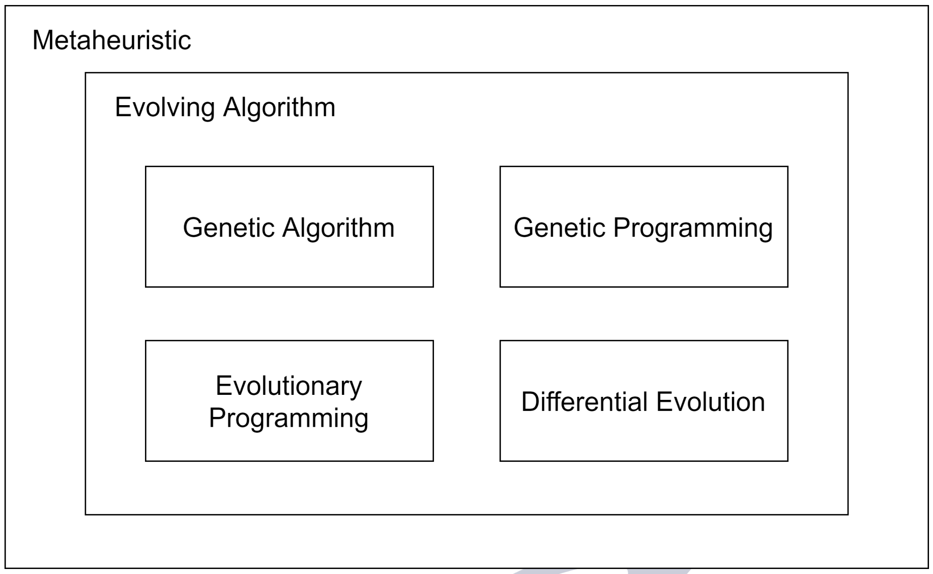


Figure 2.1 Relationship between metaheuristic, EA, GA, GP, EP, and DE

2.1.2 Evolving Algorithms

EAs can be described as algorithms inspired by the concept of ”survival of the fittest [14].” These algorithms often take cues from natural evolutionary mechanisms to create algorithms that mimic animal behavior in the search for optimal solutions. Common examples include Ant Colony Optimization (ACO) [15], GA, and Particle Swarm Optimization (PSO) [16]. The common EA procedure is shown in figure 2.2. In this type of algorithm, the typical approach is to initialize a population, which consists of several individuals. Each individual represents a potential optimal solution. After initializing the population, we usually set a number to represent how many generations we want this population to evolve. Before the start of each generation, a special function is used to calculate the fitness value of each individual, determining their level of excellence. Next, we perform selection, mutation, and crossover on the population. Selection involves choosing individuals based on their fitness to reproduce the next generation. Mutation generally refers to making random changes to an individual, while crossover combines the characteristics of two individuals to create the next generation. These steps are repeated until the predefined number of generations is reached, or the individuals fail to achieve the expected results, leading to a forced stop. Ultimately, the best individual in the population is obtained as the solution.

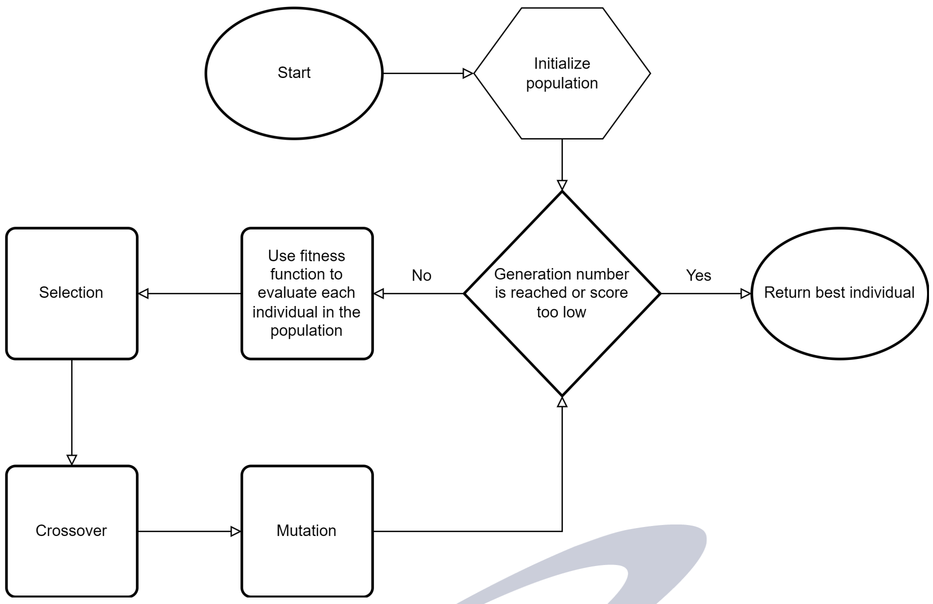

Start
Initialize population
Selection
Use fitness function to evaluate each individual in the population
No
Generation number is reached or score too low
Yes
Return best individual
Crossover
Mutation

Figure 2.2 Flowchart for EA

2.1.3 Genetic Algorithms & Genetic Programming

GA is considered a type of EA, utilizing the representation of each solution in the form of chromosomes to perform selection, mutation, and crossover. GP is regarded as a special case of GAs, but due to its versatility and practicality, it is seen as a reliable solution. Unlike GAs, GP typically uses a more robust encoding method to represent chromosomes. For instance, GAs often use strings to represent chromosomes, which can present potential issues that need to be addressed, such as the priority of operations for each symbol or the validity of the equation itself. As shown in the figure 2.3, if we want to express the function $(x + y) \div (5 * x)$ in GA, the chromosome representation of this function will be $x + y \div 5 * x$. Without the proper parentheses, it can easily cause confusion. In contrast, GP usually represents chromosomes as trees, a method that can employ predefined traversal techniques to avoid these issues. As illustrated in the figure 2.3, the same function represented by GA can be expressed through a tree structure in GP. By specifying the use of inorder traversal, we can obtain the same function. It is worth noting that in GAs and GP, we can leverage their inherent ability to evolve automatically to find a reasonable optimal solution without having an in-depth understanding of the knowledge domain related to the problem we aim to solve. In the following paragraphs, we will introduce related research.

In [17], Co-Reyes et al. proposed a meta-learning reinforcement learning algorithm. They

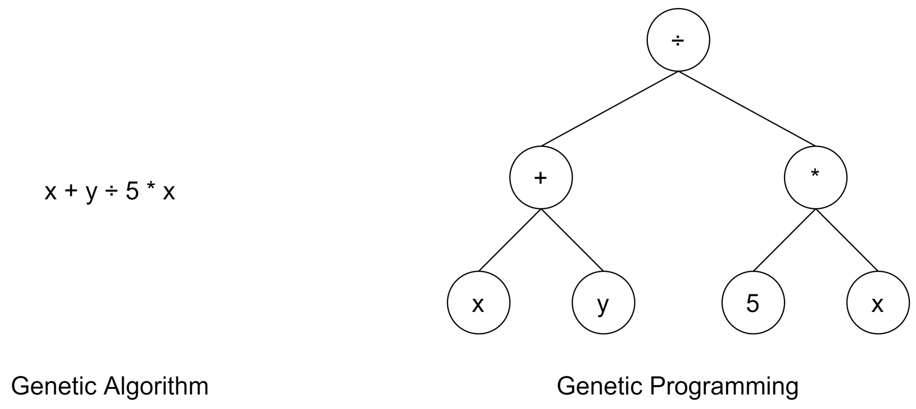


Figure 2.3 Different chromosome representation between GA and GP

used computational graphs to represent algorithms. By doing so, algorithms could be identified, calculated, and optimized through Reinforcement Learning (RL) [18]. Notably, in this study, algorithms were represented as directed acyclic graphs (DAG) [19] of nodes. Within the DAG , all nodes were classified into three categories: input nodes, parameter nodes, and operation nodes. Once the algorithms were represented as DAGs, they could be placed into RL for training and evaluation. In this proposed algorithm, they employed the concept of regularized evolution [20] to evolve a population formed by several randomized and known algorithms. The method was as follows: initialize the population with algorithms, evaluate each algorithm in the population and record their performance, then in a loop, repeatedly use a sample tournament [21] to select algorithms, perform mutations on the algorithms by the mutator they designed, and evaluate them again until the loop ends. During the evaluation process, they trained and assessed the algorithms using RL, continuously testing the performance of each algorithm in different training environments. They also utilized normalized training performance to avoid numerical biases caused by varying environments. Through this approach, the study successfully created two algorithms, named DQNClipped and DQNReg, which outperformed classical control tasks.

In [22], Akhmedova and Körber proposed a genetic programming-based method to find a better function for the image classification training task. They encoded the loss functions into trees, with each tree considered an individual. All individuals were aggregated into a population. In this study, the population was evolved across multiple generations. Before the start of each generation, all individuals were evaluated. Each individual had a probability of undergoing crossover with an individual from a special external archive to create a new individual; otherwise, it would perform the crossover with another individual. Following this, two probabilistic decisions were made. If successful, the new individual could perform subtree mutation or one-point mutation, respectively. At the end of each generation, the fitness value of all individuals was re-evaluated. A certain number of low-scoring individuals were eliminated to the special external archive mentioned earlier, maintaining the stability of the population's size. After these steps, a new generation began, continuing until a predefined number of generations was reached. By employing this method, this study successfully evolved an outstanding individual within the population, creating a function that could train an image classification model more effectively compared to Cross Entropy (CE) [23].

2.2 Loss Function

In this section, we will first introduce deep learning model and explain how does it work. Secondly, we will discuss the importance of loss function and present some related researches.

2.2.1 Deep Learning Model

In recent years, due to the significant increase in the computational speed of GPUs, training deep learning models to solve a wide variety of problems has become widely regarded as a feasible and popular solution. The process of training a deep learning model is often divided into several steps: collecting the data needed to train the model, and adding ground truth values to the data according to the model's requirements. Ground truth can be thought of as the ideal answers we hope to achieve after the model's computation. Once the data collection is complete, these data sets are collectively referred to as the dataset. Typically, the dataset is divided into training sets, validation sets, and testing sets. Data from the training set are used to train the model, the validation set is used to evaluate the model's effectiveness after each training loop, and the testing set is used to calculate the model's final score and determine whether the model successfully achieves its designed purpose. It is worth noting that the testing set data should not be seen during training and validation phases. After dividing the dataset, we decide on the model's architecture. Common architectures include Fully Connected Networks (FCN) [24] and Convolutional Neural

Networks (CNN) [25], etc. Then the training process can begin. During training, we compare the model's output with the ground truth of the training data and use a loss function to calculate the difference between the output and the ground truth. This guides the adjustment of the model's parameters. Therefore, the loss function significantly determines the effectiveness of model training, and this aspect will be explained further later. After adjusting the model, the next round of training begins, continuing until the training is deemed ineffective or a predetermined maximum number of iterations is reached.

2.2.2 Loss Function In Deep Learning Model

In section 2.2.1, we discussed the role of loss functions in deep learning models. Here, we would like to introduce some commonly used loss functions along with their advantages and disadvantages. Mean-Square Error (MSE) [26] is a loss function used for solving regression problems. It calculates the difference between the actual and predicted values, squares them, and then takes the average of these squared differences to obtain the final loss value. However, a notable disadvantage of MSE is that it amplifies the impact of outliers exponentially. Cross-Entropy (CE), unlike MSE, leverages probability concepts to help compute the error in classification tasks. In other words, CE measures the difference between the predicted probability and the true probability to calculate the error. CE can also be adapted to different types of classification tasks to better suit their requirements, with common variants including Binary CE Loss [27] and Multiclass CE Loss [28].

From the above paragraphs, it seems that using a single type of loss function to train all models is considered impractical. Therefore, researchers design different loss functions based on specific needs to calculate more appropriate loss values for the model. As the introduction above indicates, designing a loss function requires an in-depth understanding of the model and a clear grasp of how to define and calculate the loss value. Consequently, designing a loss function, similar to adjusting hyperparameters or model architecture, is considered as a challenging task that needs a deep understanding of the domain for effective design and adjustment.

In [29], Gonzalez and Miikkulainen proposed a meta-learning approach to create a loss function called Genetic Loss-function Optimization (GLO). Through their research, the authors experimentally found a loss function named de novo, created by GLO, outperforms the well-known CE loss in standard image classification tasks. Additionally, because GLO enables training to be completed in fewer steps, this method also enhances the speed and efficiency of the training process.

In [22], Akhmedova and Körber utilize GP to create a loss function that outperforms CE in image classification. The method described in the paper led to the creation of a loss function named Next Generation Loss (NGL). When trained with the Inception model, NGL outperforms CE on datasets such as CIFAR-10 [30], CIFAR-100 [31], and Fashion-MNIST [32]. Additionally, it demonstrates excellent performance on larger datasets like [33]. Furthermore, NGL is also effective in improving model performance in segmentation downstream tasks.

From the above paragraphs, we can observe that numerous studies indicate that in image classification tasks, if a well-designed loss function is developed, it has the potential to outperform the most famous and widely regarded as the most effective CE loss function. However, as mentioned earlier, designing a loss function is usually considered a resource-intensive task. Therefore, the two studies discussed above have begun to use genetic programming to enable computers to automatically design a loss function that can effectively collaborate with a model in a specific domain and significantly improve the model's performance.

2.3 Image Classification

Image classification is a technique in computer vision that enables computers to identify what object present in an image. This technology is often used with image localization [34]. Image localization determines the location of objects within an image, using bounding box [35] to enclose the areas where the objects are found. Additionally, object detection [36] is a similar technique to image classification and image localization. Unlike image classification and image localization, which focus on identifying one label per image, object detection can identify multiple labels in a single image. The difference between image classification, image localization and object detection is illustrated in the figure 2.4. We can see that image classification is capable of recognizing which label the entire image belongs to, whereas image localization identifies which specific regions of the image belong to which labels. Object detection is a more complex technique, which can identify multiple labels and also mark the object by using bounding boxes.

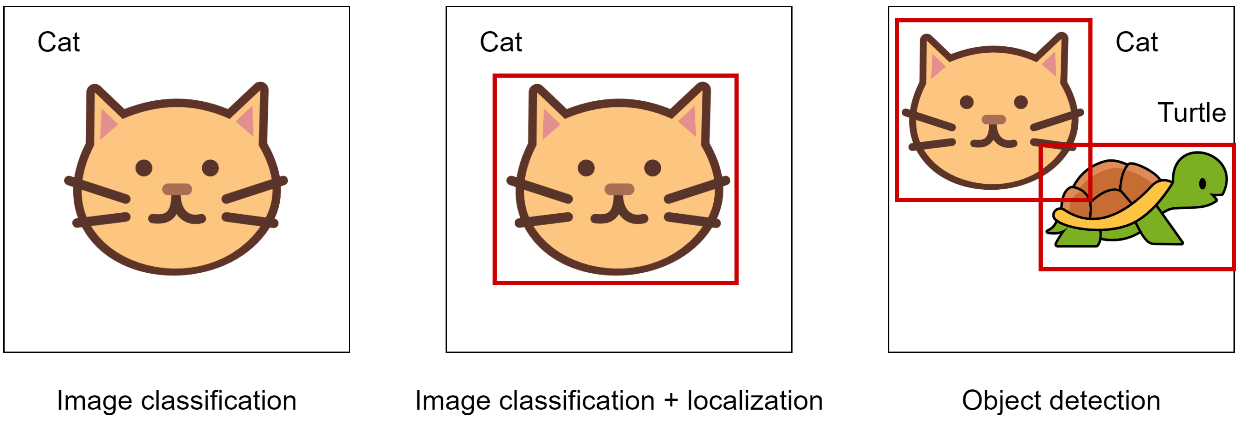


Figure 2.4 Difference between image classification, image localization and object detection

In image classification, the most common types are binary classification [37], multiclass classification [38] and multilabel classification [39]. As the name implies, in binary classification, there are only two possible outcomes when identifying an image. For instance, if the model's labels are cat and dog, the prediction can only be either a cat or a dog. Conversely, multiclass classification means that the image can belong to multiple possible classes instead of two classes. Using the previous example, the prediction in multiclass classification could be a cat, dog, elephant, snake, or other different animals, rather than choosing between just two labels. It is important to note that the final prediction in both binary and multiclass classification will result in only one label. If an image needs to have multiple labels, a different type of image classification model is required, such as a multilabel model.

The process of training an image classification model is generally similar to what is described in section 2.2.1 on Deep Learning. In image classification, commonly used datasets include: ImageNet [40], CIFAR-10, CIFAR-100, and MNIST [41]. It's worth mentioning that we often perform preprocessing on images. Common preprocessing techniques include image cropping, image resizing, and image normalization. Image cropping is a technique that removes the unnecessary parts of the image. Through image cropping, the irrelevant parts of the image will be removed. As shown in the figure 2.5, the pink area in the original picture is removed to prevent model from learning the useless information. Image resizing ensures that all images are of the same size, which helps improve computational speed. However, there is a potential risk of degrading the model's performance by using image resizing [42]. Hence, the size of the resized images is typically determined based on actual requirements. Normalizing images also helps the model learn from each image more effectively. After preprocessing, we usually use pretrained models instead of designing a new model architecture from scratch. This approach enhances computational efficiency and results. Common image classification models used include InceptionV3 [43], ResNet [44], and others.

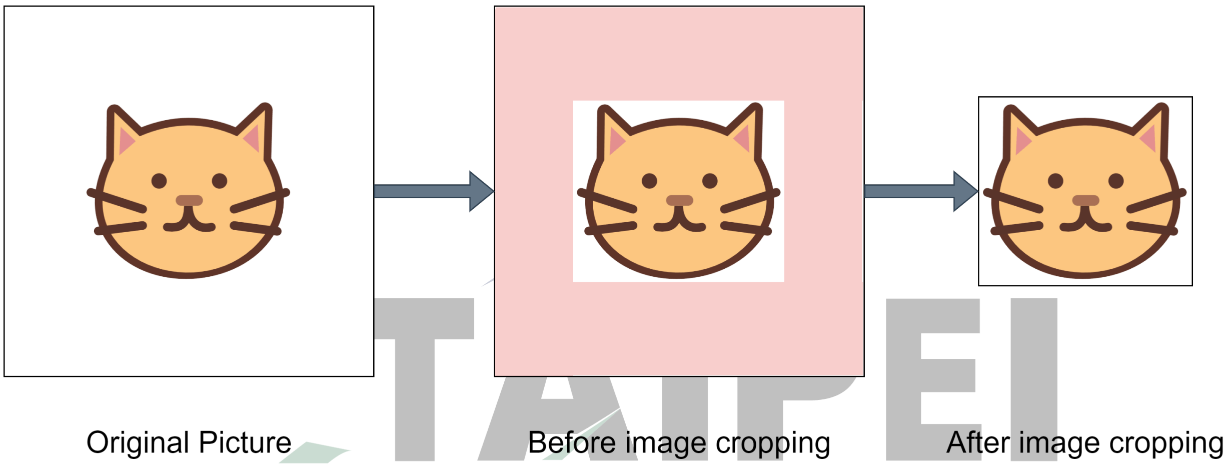


Figure 2.5 The process of image cropping

In the real world, image classification can be applied in many areas. For example, it can be used in medical imaging to identify tumors in X-rays. It can also be employed in autonomous vehicles to quickly recognize objects surrounding the vehicle. Facial recognition is another application of image classification, used for company access control and as a method to unlock personal devices such as phones or computers.

Chapter 3 Proposed Algorithm

In this chapter, we will introduce the architecture and flowchart of our algorithm. Next, we will provide an in-depth explanation of our proposed algorithm, including the underlying concepts and the rationale behind our approach. Subsequently, we will present a detailed description of our software and hardware specifications, as well as the setup procedures. Finally, we will execute the experiment and describe how we implemented our concept and successfully completed the entire experiment.

3.1 Main Method

Figure 3.1 illustrates our software stack. On our existing hardware, we use Windows 11 as our operating system. In addition, we use Python 3.9.18 as the programming language for this research. Based on Python 3.9.18, the main packages we use are TensorFlow, NumPy, and Scikit-learn.

In previous research [22], we observed that genetic operations are performed on the entire population, requiring the evaluation of the entire new population after each complete set of genetic operations. However, the evaluation process is time-consuming. Therefore, in this study, we will employ the concept of a sample tournament to select certain individuals rather than the entire population for genetic operations. By doing so, we only need to evaluate the newly generated individuals. Figure 3.2 illustrates the conceptual flowchart of this study.

As illustrated in the figure 3.2, we initialize a population consisting of P individuals before commencing the experiment. Once initialized, we evaluate all individuals, store their scores, and identify the best score for subsequent analysis and research. The process then enters a loop, continuing until the generation number reaches the specified value. Initially in this loop, we use a sample tournament to randomly select N individuals from the population, followed by choosing the top X individuals for genetic operations (crossover and mutation). The offspring of these X individuals are then evaluated and added to the population. We update the best-performing individual and remove the poorly performing individuals until the population size returns to P. Upon

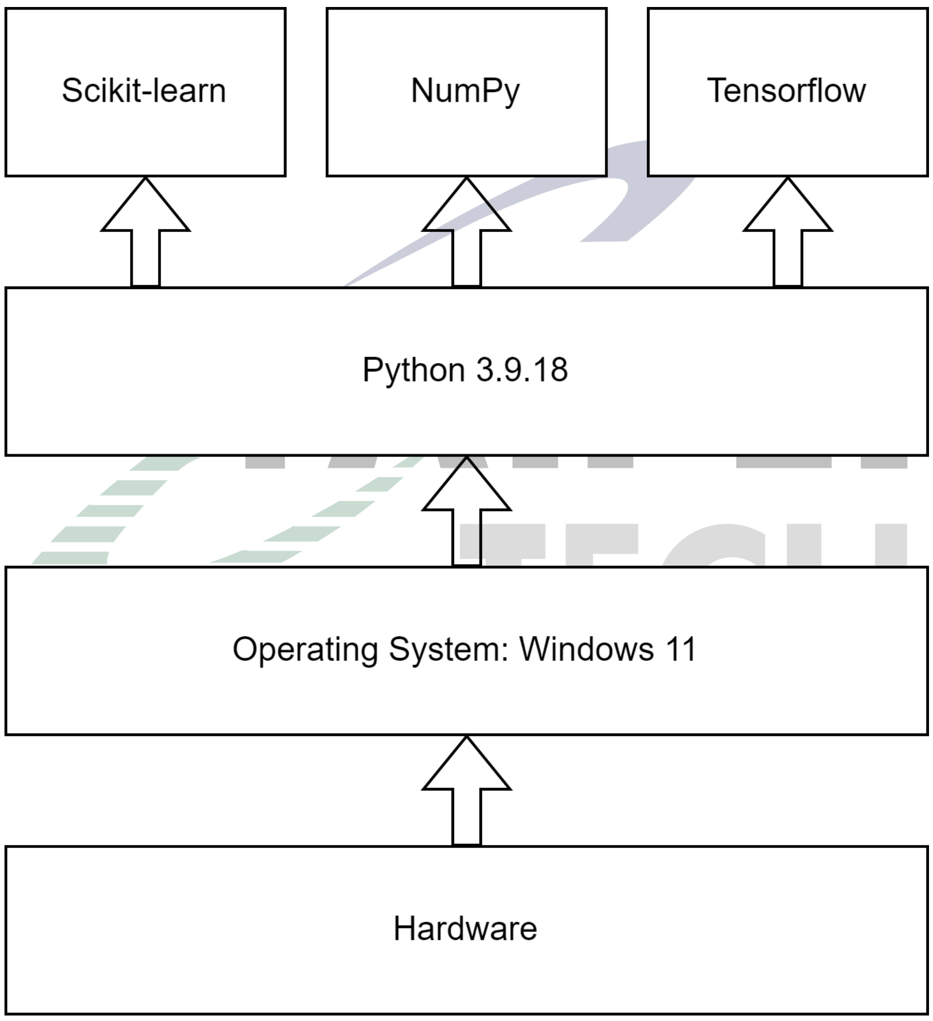


Figure 3.1 Software stack for Efficiency-based GP

concluding the loop, we generate the best-performing loss function and conclude the experiment.

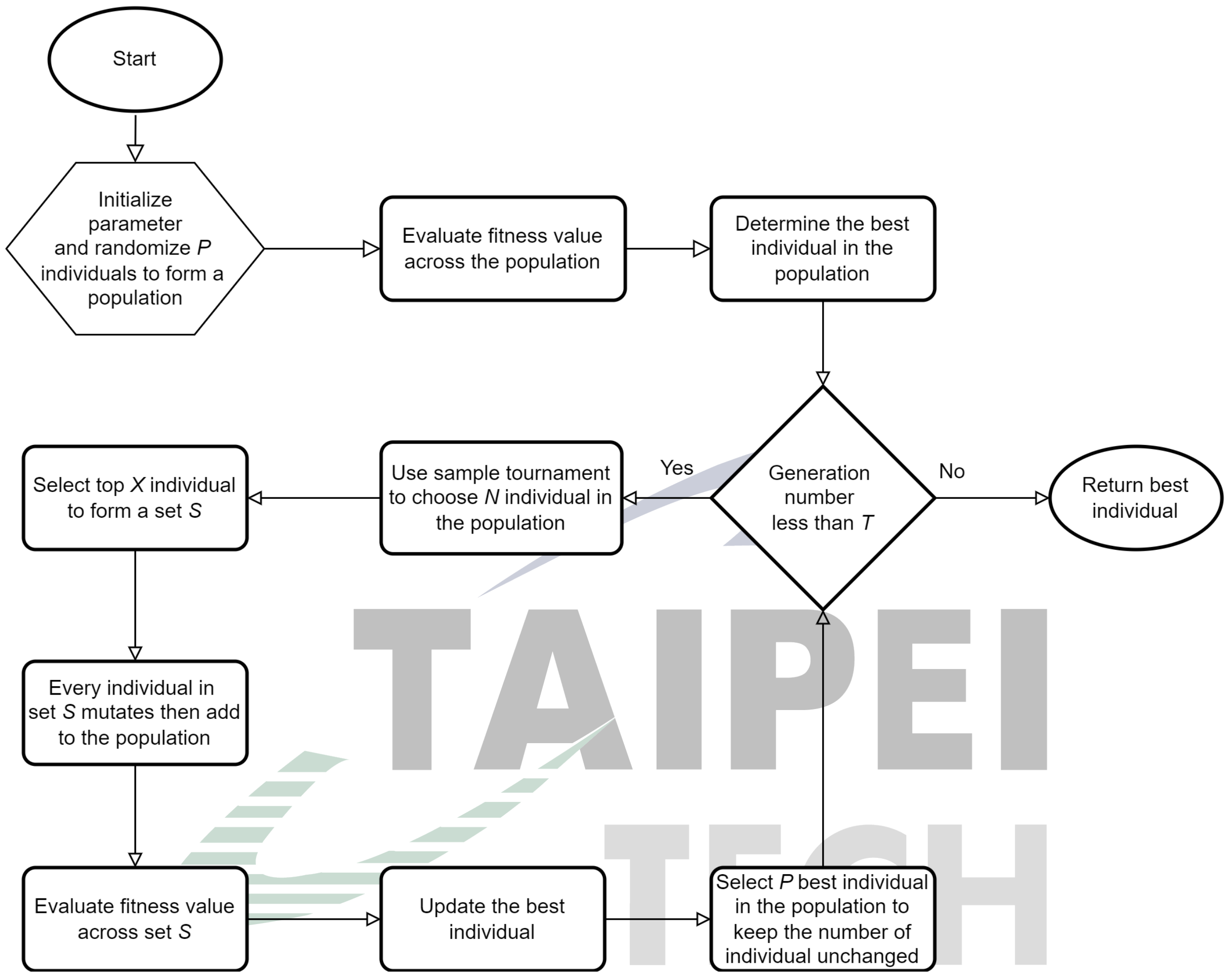

Start
Initialize parameter and randomize *P* individuals to form a population
Evaluate fitness value across the population
Determine the best individual in the population
Generation number less than *T*
Yes
No
Return best individual
Use sample tournament to choose *N* individual in the population
Select top *X* individual to form a set *S*
Every individual in set *S* mutates then add to the population
Evaluate fitness value across set *S*
Update the best individual
Select *P* best individual in the population to keep the number of individual unchanged

Figure 3.2 FlowChart for Efficiency-based GP

3.2 Implementation Details

To implement our experiment, the hardware and software requirements are first defined in Section 3.2.1. Next, the environment setup steps are described in Section 3.2.2. Finally, the implementation steps are explained in Section 3.2.3.

3.2.1 Requirements

Table 3.1 outlines our software requirements. Our code execution environment is Windows 11. We utilize Python version 3.9.18 and TensorFlow version 2.6.0. The specific packages used within Python and TensorFlow are detailed in Table 3.2.

Table 3.1 Software requirements

Software Type	Software Name	Version
Operating System	Windows	11
Programming Language	Python	3.9.18

Table 3.2 Python package requirements

Package Name/Version	Imported from	License
tensorflow/2.6.0	N/A	Apache License 2.0
numpy/1.20.3	N/A	modified BSD license
cudnn/8.2.1	N/A	N/A
os	N/A	N/A
copy	N/A	N/A
time	N/A	N/A
math	N/A	N/A
random	N/A	N/A
datetime	datetime	N/A
cmp_to_key	functools	N/A
Sequential	tensorflow.keras.models	N/A
Dense	tensorflow.keras.layers	N/A
Flatten	tensorflow.keras.layers	N/A
Dropout	tensorflow.keras.layers	N/A
UpSampling2D	tensorflow.keras.layers	N/A
BatchNormalization	tensorflow.keras.layers	N/A
InceptionV3	tensorflow.keras.applications.inception_v3	N/A
preprocess_input	tensorflow.keras.applications.inception_v3	N/A
ReduceLROnPlateau	tensorflow.keras.callbacks	N/A
to_categorical	tensorflow.keras.utils	N/A
train_test_split/1.5.2	sklearn.model_selection	N/A
ImageDataGenerator	tensorflow.keras.preprocessing.image	N/A
np_config	tensorflow.python.ops.numpy_ops	N/A

Table 3.3 presents our hardware requirements. The computer we used to execute our code is equipped with a 13th Gen Intel(R) Core(TM) i7-13700 CPU, 32GB of memory operating at a frequency of DDR5-5600MHz, and an NVIDIA GeForce RTX 4060 GPU to meet our computational needs.

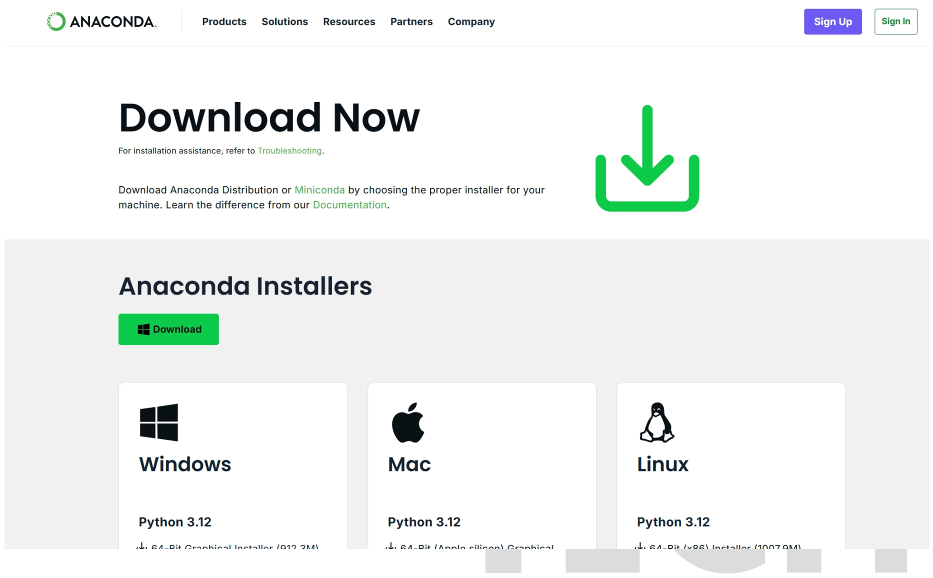
3.2.2 Environment Setup

The following section will describe how to properly set up the environment we used to conduct our experiment. First, we used Anaconda as our package management tool for the implemen-

Table 3.3 Hardware requirements

Hardware Type	Name
CPU	13th Gen Intel(R) Core(TM) i7-13700
Memory	DDR5-5600MHz 32 GB
Graphic Card	NVIDIA GeForce RTX 4060

tation. Therefore, you must first visit the Anaconda official website (`https://www.anaconda.com/download`) to download the software. As shown in figure 3.3, click the Download button to complete the download.



Download Now

For installation assistance, refer to Troubleshooting.

Download Anaconda Distribution or Miniconda by choosing the proper installer for your machine. Learn the difference from our Documentation.

Anaconda Installers

Windows — Python 3.12

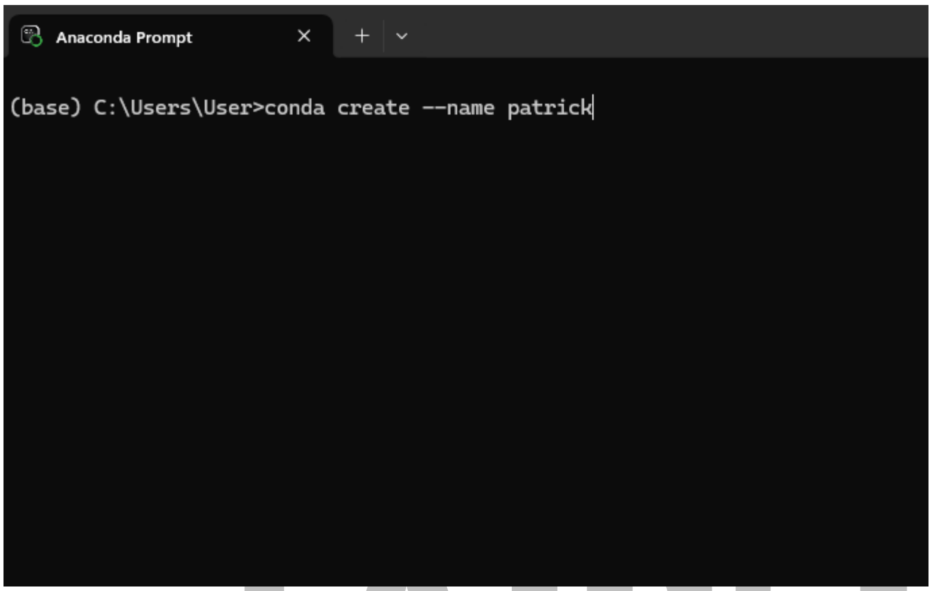
Mac — Python 3.12

Linux — Python 3.12

Figure 3.3 Anaconda official website

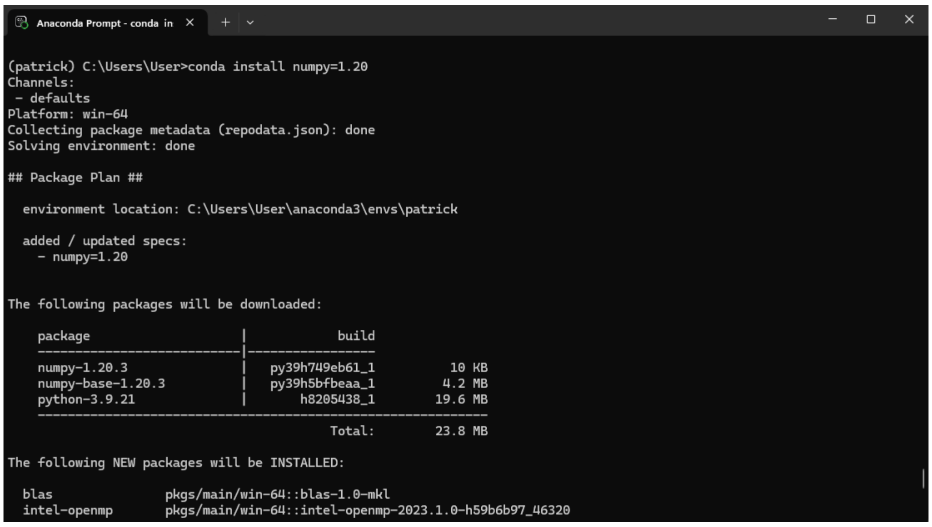
After installation, we need to create a new environment to ensure the independence of all the packages. In Anaconda prompt, enter `conda create --name environment-name` to create a new environment, as shown in figure 3.4. Next, enter `conda activate environment-name` to activate the environment just like the first prompt entered in figure 3.6.

Since directly installing TensorFlow may result in the installation of incompatible versions of NumPy, we first use the command shown in figure 3.5 to download version 1.20.3 of NumPy: `conda install numpy=1.20`. Afterward, we can install the GPU version of TensorFlow by entering the following command in the Anaconda prompt: `conda install tensorflow-gpu`, as shown in figure 3.6. Finally, we need to download Scikit-learn to split the dataset during the model training process. As shown in figure 3.7, enter the following command to complete the final installation step: `conda install scikit-learn`.



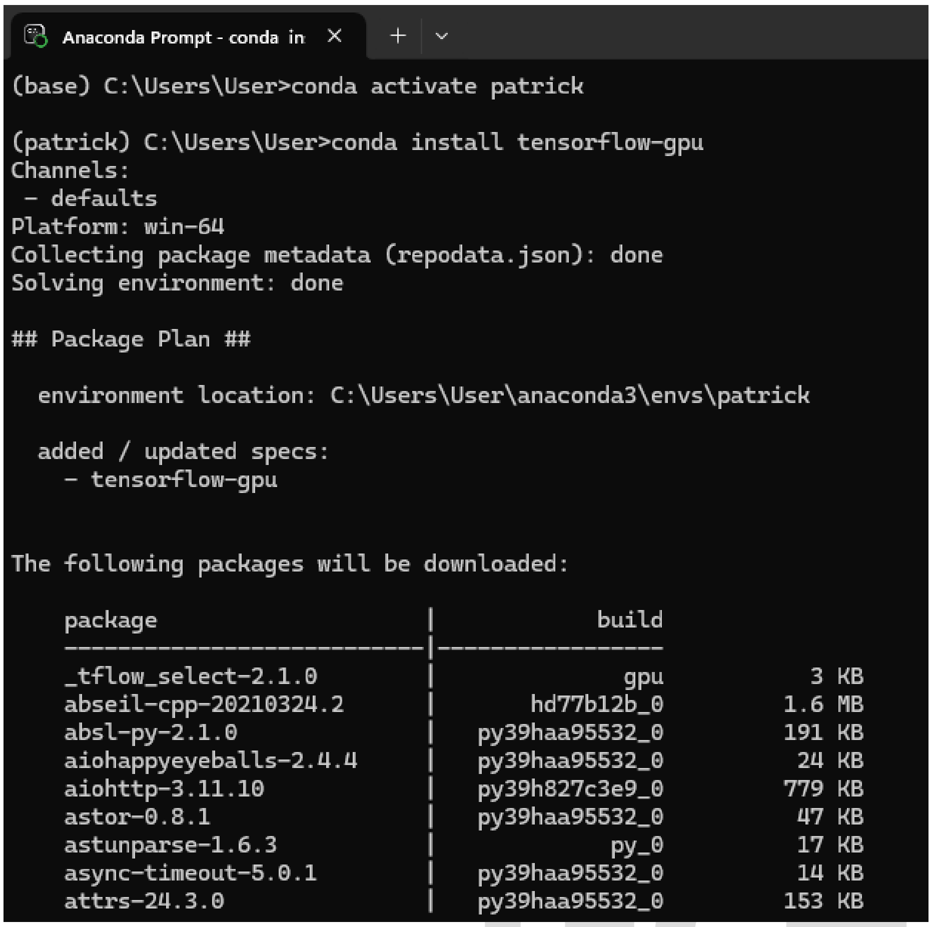
```
(base) C:\Users\User>conda create --name patrick
```

Figure 3.4 Conda prompt for creating new environment



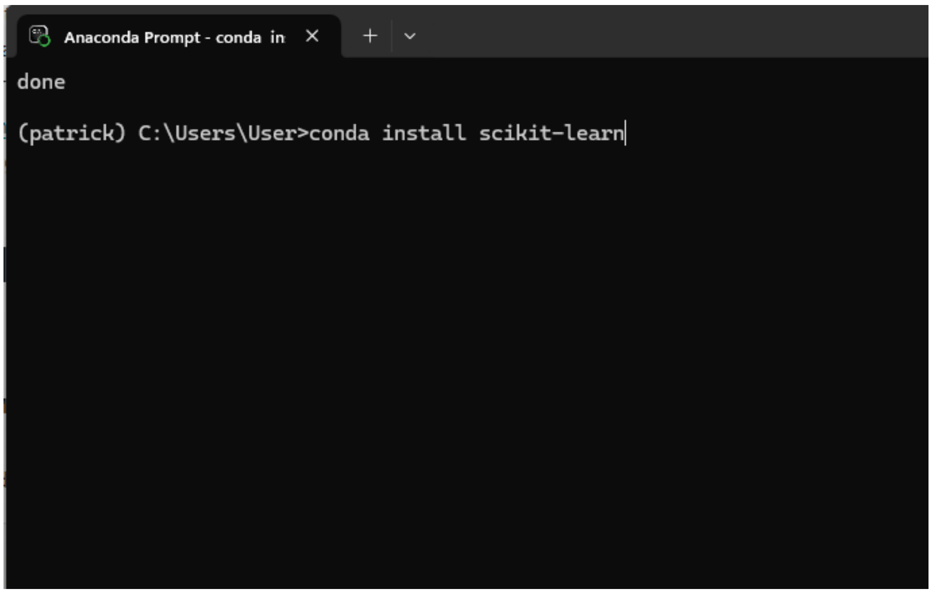
```
(patrick) C:\Users\User>conda install numpy=1.20
Channels:
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\User\anaconda3\envs\patrick

  added / updated specs:
    - numpy=1.20


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    numpy-1.20.3               |   py39h749eb61_1          10 KB
    numpy-base-1.20.3          |   py39h5bfbeaa_1         4.2 MB
    python-3.9.21              |       h8205438_1        19.6 MB
    ------------------------------------------------------------
                                           Total:        23.8 MB

The following NEW packages will be INSTALLED:

  blas               pkgs/main/win-64::blas-1.0-mkl
  intel-openmp       pkgs/main/win-64::intel-openmp-2023.1.0-h59b6b97_46320
```

Figure 3.5 Install numpy in conda environment



```
(base) C:\Users\User>conda activate patrick

(patrick) C:\Users\User>conda install tensorflow-gpu
Channels:
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\User\anaconda3\envs\patrick

  added / updated specs:
    - tensorflow-gpu


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    _tflow_select-2.1.0        |              gpu           3 KB
    abseil-cpp-20210324.2      |       hd77b12b_0         1.6 MB
    absl-py-2.1.0              |   py39haa95532_0         191 KB
    aiohappyeyeballs-2.4.4     |   py39haa95532_0          24 KB
    aiohttp-3.11.10            |   py39h827c3e9_0         779 KB
    astor-0.8.1                |   py39haa95532_0          47 KB
    astunparse-1.6.3           |             py_0          17 KB
    async-timeout-5.0.1        |   py39haa95532_0          14 KB
    attrs-24.3.0               |   py39haa95532_0         153 KB
```

Figure 3.6 Install tensorflow in conda environment



```
done

(patrick) C:\Users\User>conda install scikit-learn
```

Figure 3.7 Install scikit-learn in conda environment

3.2.3 Implementation Steps

Based on the architecture we introduced on section 3.1, the pseudo code of this algorithm is presented as below.

Algorithm 1 Efficiency-based GP to generate loss function

```
Initialize: $P = 8$, $T = 12$, $G = P * 3/4$, $N = G/2$, $C_r = 0.5$, $M_{ST} = 0.3$, $M_N = 0.1$
Randomly initialize population which include P trees
Evaluate GP fitness function $F$ for each individual in the population
Determine the best individual
while generation number < $T$ do
    $S$ = Sample tournament G $\sim$ Uniform (P)
    Select top $N$ trees from $G$ to form a set $S$
    for Individual in $S$ do
        if $rand_1 < 0.5$ then
            random_individual = Randomly select a individual in $S$ except Individual
        else
            random_individual = Randomly generated tree
        end if
        generated_child = Crossover(Individual, random_individual)
        if $rand_2 < M_{ST}$ then
            Apply subtree mutation to the generated_child
        end if
        if $rand_3 < M_N$ then
            Apply one-point mutation to the generated_child
        end if
    end for
    Evaluated GP fitness function $F$ for each generated child individual
    Update the best individual
    Select $P$ best trees from population to form a new population containing $P$ trees
end while
```

Chapter 4 Result & Analysis

Chapter 5 Conclusion & Future Work

5.1 Conclusion

5.2 Future Work

References

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[2] Katarzyna Janocha and Wojciech Marian Czarnecki. *On Loss Functions for Deep Neural Networks in Classification*. 2017. arXiv: `1702.05659 [cs.LG]`. URL: `https://arxiv.org/abs/1702.05659`.

[3] Lorenzo Rosasco et al. "Are Loss Functions All the Same?" In: *Neural Computation* 16.5 (2004), pp. 1063–1076. DOI: `10.1162/089976604773135104`.

[4] Michael O' Neill. "Riccardo Poli, William B. Langdon, Nicholas F. McPhee: A Field Guide to Genetic Programming". In: *Genetic Programming and Evolvable Machines* 10.2 (2009), pp. 229–230.

[5] Xin-She Yang. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[6] Xin-She Yang. "Metaheuristic optimization". In: *Scholarpedia* 6.8 (2011), p. 11472.

[7] Waseem Rawat and Zenghui Wang. "Deep convolutional neural networks for image classification: A comprehensive review". In: *Neural computation* 29.9 (2017), pp. 2352–2449.

[8] Heinz Mühlenbein, Martina Gorges-Schleuter, and Ottmar Krämer. "Evolution algorithms in combinatorial optimization". In: *Parallel computing* 7.1 (1988), pp. 65–85.

[9] Manoj Kumar et al. "Genetic algorithm: Review and application". In: *Available at SSRN 3529843* (2010).

[10] Fred Glover. "Future paths for integer programming and links to artificial intelligence". In: *Computers & operations research* 13.5 (1986), pp. 533–549.

[11] Wikipedia contributors. *Metaheuristic — Wikipedia, The Free Encyclopedia*. [Online; accessed 15-December-2024]. 2024. URL: `https://en.wikipedia.org/w/index.php?title=Metaheuristic&oldid=1255593755`.

[12] Xin Yao, Yong Liu, and Guangming Lin. "Evolutionary programming made faster". In: *IEEE Transactions on Evolutionary computation* 3.2 (1999), pp. 82–102.

[13] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. "Differential evolution: A survey of the state-of-the-art". In: *IEEE transactions on evolutionary computation* 15.1 (2010), pp. 4–31.

[14] Diane B Paul. "The selection of the "Survival of the Fittest" ". In: *Journal of the History of Biology* 21 (1988), pp. 411–424.

[15] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. "Ant colony optimization". In: *IEEE computational intelligence magazine* 1.4 (2006), pp. 28–39.

[16] James Kennedy and Russell Eberhart. “Particle swarm optimization”. In: *Proceedings of ICNN’95-international conference on neural networks*. Vol. 4. ieee. 1995, pp. 1942–1948.

[17] John D. Co-Reyes et al. *Evolving Reinforcement Learning Algorithms*. 2022. arXiv: 2101.03958 [cs.LG]. URL: https://arxiv.org/abs/2101.03958.

[18] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. “Reinforcement learning: A survey”. In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[19] Veronika Thost and Jie Chen. *Directed Acyclic Graph Neural Networks*. 2021. arXiv: 2101.07965 [cs.LG]. URL: https://arxiv.org/abs/2101.07965.

[20] Esteban Real et al. “Regularized evolution for image classifier architecture search”. In: *Proceedings of the aaai conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 4780–4789.

[21] David E Goldberg and Kalyanmoy Deb. “A comparative analysis of selection schemes used in genetic algorithms”. In: *Foundations of genetic algorithms*. Vol. 1. Elsevier, 1991, pp. 69–93.

[22] Shakhnaz Akhmedova and Nils Körber. *Next Generation Loss Function for Image Classification*. 2024. arXiv: 2404.12948 [cs.CV]. URL: https://arxiv.org/abs/2404.12948.

[23] Zhilu Zhang and Mert Sabuncu. “Generalized cross entropy loss for training deep neural networks with noisy labels”. In: *Advances in neural information processing systems* 31 (2018).

[24] Michael Iliadis, Leonidas Spinoulas, and Aggelos K Katsaggelos. “Deep fully-connected networks for video compressive sensing”. In: *Digital Signal Processing* 72 (2018), pp. 9–18.

[25] Jianxin Wu. “Introduction to convolutional neural networks”. In: *National Key Lab for Novel Software Technology. Nanjing University. China* 5.23 (2017), p. 495.

[26] Zhou Wang and Alan C Bovik. “Mean squared error: Love it or leave it? A new look at signal fidelity measures”. In: *IEEE signal processing magazine* 26.1 (2009), pp. 98–117.

[27] Usha Ruby and Vamsidhar Yendapalli. “Binary cross entropy with deep learning technique for image classification”. In: *Int. J. Adv. Trends Comput. Sci. Eng* 9.10 (2020).

[28] Alexis Plaquet and Hervé Bredin. “Powerset multi-class cross entropy loss for neural speaker diarization”. In: *arXiv preprint arXiv:2310.13025* (2023).

[29] Santiago Gonzalez and Risto Miikkulainen. *Improved Training Speed, Accuracy, and Data Utilization Through Loss Function Optimization*. 2020. arXiv: 1905.11528 [cs.LG]. URL: https://arxiv.org/abs/1905.11528.

[30] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. “CIFAR-10 (Canadian Institute for Advanced Research)”. In: (). URL: http://www.cs.toronto.edu/~kriz/cifar.html.

[31] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. “CIFAR-100 (Canadian Institute for Advanced Research)”. In: (). URL: http://www.cs.toronto.edu/~kriz/cifar.html.

[32] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. 2017. arXiv: 1708.07747 [cs.LG]. URL: https://arxiv.org/abs/1708.07747.

[33] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[34] Torsten Sattler, Bastian Leibe, and Leif Kobbelt. “Fast image-based localization using direct 2d-to-3d matching”. In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 667–674.

[35] Yihui He et al. *Bounding Box Regression with Uncertainty for Accurate Object Detection*. 2019. arXiv: 1809.08545 [cs.CV]. URL: https://arxiv.org/abs/1809.08545.

[36] Zhong-Qiu Zhao et al. “Object detection with deep learning: A review”. In: *IEEE transactions on neural networks and learning systems* 30.11 (2019), pp. 3212–3232.

[37] Bohan Zhuang et al. “Structured binary neural networks for accurate image classification and semantic segmentation”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 413–422.

[38] Venkatesh N Murthy et al. “Deep decision network for multi-class image classification”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2240–2248.

[39] Grigorios Tsoumakas and Ioannis Katakis. “Multi-label classification: An overview”. In: *Data Warehousing and Mining: Concepts, Methodologies, Tools, and Applications* (2008), pp. 64–74.

[40] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[41] Li Deng. “The mnist database of handwritten digit images for machine learning research”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[42] Sergio Saponara and Abdussalam Elhanashi. “Impact of Image Resizing on Deep Learning Detectors for Training Time and Model Performance”. In: Jan. 2022, pp. 10–17. ISBN: 978-3-030-95497-0. DOI: 10.1007/978-3-030-95498-7_2.

[43] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. 2015. arXiv: 1512.00567 [cs.CV]. URL: https://arxiv.org/abs/1512.00567.

[44] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.