Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)

КАФЕДРА «Системы обработки информации и управления» (ИУ5)

Домашнее задание по дисциплине «Методы машинного обучения» на тему: «Анализ современных методов машинного обучения и их применение для решения практических задач»

Студент группы ИУ5-21М _____ Р.В. Фонканц

Преподаватель _____ Ю.Е. Гапанюк

2022 г.

Содержание

Введение

Основанные на законах природы, генетические алгоритмы являются одними из самых удивительных методов решения задач поиска, оптимизации и обучения. Они могут привести к успеху там, где традиционные алгоритмы не способны дать адекватные результаты за приемлемое время.

Генетический алгоритм – это стохастический поисковый алгоритм, который итеративно трансформирует множество математических объектов (популяцию), представляющих собой кодированные решения некоторой задачи. [1] Фактически, генетические алгоритмы математически имитируют процессы эволюции и естественного отбора, которые лежат в основе эволюционной теории Чарльза Дарвина. Большую популярность генетические алгоритмы начали приобретать с 1975 года после выхода книги Джона Холланда «Адаптация в естественных и искусственных системах».

Генетические алгоритмы как одно из направлений машинного обучение не получили должного внимания и популярности среди ученых, аналитиков и инженеров по работе с данными. Предположительно, этот факт объясняется эвристической природой алгоритма, в виду чего решение четко поставленных задач с известными характеристиками может быть найдено быстрее с помощью специально разработанных методов и алгоритмов. Но главная задача генетических алгоритмов – это решение нестандартных, малоизвестных и слабоанализируемых задач, где они оказываются наиболее эффективными.

Постановка задачи машинного обучения

Данная работа является исследованием в области генетических алгоритмов. В ходе работы ставятся следующие задачи: исследование теоретических основ генетических алгоритмов, обзор библиотеки PyGAD для работы с генетическими алгоритмами на языке программирования Python и решение задачи воспроизведения изображения с помощью генетических алгоритмов. Для решения первой задачи используются учебно-методические пособия и книги Батищева Д.И., Неймарка Е.А., Буракова М.В., Вирсански Э. и других авторов, исследующих генетические алгоритмы. Для решения второй и третьей задач используются статьи *«PyGAD: An Intuitive Genetic Algorithm Python Library»* и *«Reproducing Images using a Genetic Algorithm with Python»*, написанные кандидатом технических наук Ахмедом Гадом.

Далее в теоретической части работы будет представлена основная информация из перечисленных научных трудов, а практическая часть будет посвящена решению задачи воспроизведения изображения посредством генетических алгоритмов с использованием библиотеки PyGAD.

Теоретическая часть

Теоретические основы генетических алгоритмов: принцип работы, общие понятия и ключевые особенности

Генетические алгоритмы возникли в результате наблюдения и попыток копирования естественных процессов, происходящих в мире живых организмов, в частности, эволюции, связанной с ней селекции (естественного отбора) популяции живых существ. Идея генетических алгоритмов была высказана в конце шестидесятых - начале семидесятых годов XX века. Она была основана на желании составить и реализовать в виде компьютерной программы алгоритм, который будет решать сложные задачи так, как это делает природа - путем эволюции. [2]

Генетические алгоритмы — это совокупности операций поиска, в основе которых лежат принципы самой природы: наследование и естественный отбор. Отсюда можно заключить, что результатом работы генетических алгоритмов являются наиболее сильные и приспособленные особи.

Стоит отметить, что генетические алгоритмы обладают рядом ключевых особенностей и свойств:

- Работа с закодированными формами параметров, влияющих на целевую функцию;
- На каждой итерации генетические алгоритмы осуществляют поиск решения в рамках некоторой популяции;
- Для работы генетическим алгоритмам достаточно только целевой информации (нет необходимости в какой-либо дополнительной информации или иных математических представлениях целевой функции);

- Правила выбора в генетических алгоритмах являются вероятностными, а не детерминированными.

Сфера применения генетических алгоритмов — это в основном оптимизация многопараметрических функций. Прикладное же применение генетических алгоритмов весьма обширно. Они применяются при разработке программного обеспечения в системах искусственного интеллекта, оптимизации, искусственных нейронных сетях и в других отраслях знаний. Следует отметить, что с их помощью решаются задачи, для которых ранее использовались нейронные сети. В этом случае генетические алгоритмы выступают просто в роли независимого от нейронных сетей метода, предназначенного для решения той же самой задачи. Примером может служить задача коммивояжера, изначально решавшаяся при помощи сети Хопфилда. Генетические алгоритмы часто используются совместно с нейронными сетями. Они могут поддерживать нейронные сети или наоборот, либо оба метода взаимодействуют в рамках одной гибридной системы, предназначенной для решения конкретной задачи. Генетические алгоритмы так же применяются совместно с нечеткими системами. [2]

Генетические алгоритмы не гарантируют обнаружение глобального оптимума за полиномиальное время, однако они позволяют выбрать «достаточно хорошее» решение за меньшее время, чем другие известные детерминированные или эвристические алгоритмы поисковой оптимизации. [3]

В одной из своих статей К. Де Йонг довольно четко и емко изложил рациональность использования генетических алгоритмов: «Решающий аргумент при использовании генетических алгоритмов тесно связан с вопросом о том, какое пространство поиска будет исследовано. Если это пространство легко анализировать, и его структура позволяет использовать специализированные методы поиска, то использование генетических алгоритмов менее эффективно с точки зрения затрат вычислительных ресурсов. Если же пространство поиска не

поддается анализу и относительно слабо структурировано, и если возможен эффективный способ представления в генетических алгоритмах этого пространства, то они оказываются удивительно эффективным методом эвристического поиска в больших и сложных областях». [4]

В рамках работы с генетическими алгоритмами принято оперировать следующими понятиями: [5]

- Популяция – набор особей;
- Особь (хромосома) – потенциальное решение поставленной задачи;
- Отбор (селекция) – процесс выбор наилучших особей для размножения на основании функции приспособленности;
- Размножение – процесс создания потомков либо с помощью скрещивания особей, либо их полным копированием;
- Мутация – это изменение особей случайным образом для поддержания разнообразия популяции;
- Функция приспособленности (пригодности) – функция, определяющая степень близости особи к удовлетворительному решению поставленной задачи (т.е. функция, определяющая качество особей).

Основные принципы работы генетических алгоритмов можно описать следующим образом: [6]

1. Производится генерация начальной популяции, состоящей из n особей. Генерация начальной популяции может проводиться различными способами.
2. С помощью функции пригодности оценивается пригодность каждой особи в популяции.
3. Осуществляется селекция с целью поиска наиболее пригодных особей.
4. Проводится размножение наиболее приспособленных особей. Это может быть как скрещивание двух особей, так и их полное копирование.

5. К произведенным потомкам применяются определенные заранее мутации.
6. Шаги 2-5 повторяются до тех пор, пока не будет достигнут критерий окончания. В качестве критерия окончания может рассматриваться два условия: схождение популяции или достижение максимально заданного количества итераций (поколений). [7]

Блок-схема работы описанного генетического алгоритма представлена на рисунке 1.

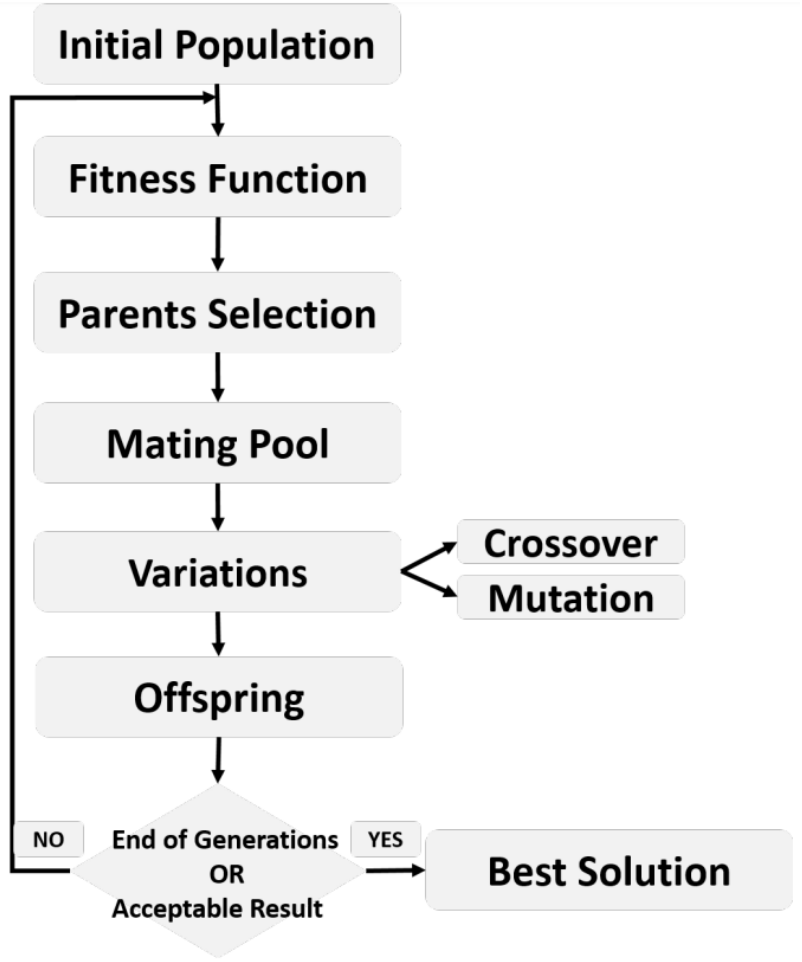

Рисунок 1 «Блок-схема работы генетического алгоритма»

Обзор библиотеки PyGAD

PyGAD — это библиотека Python с открытым исходным кодом для решения задач оптимизации с использованием генетических алгоритмов. Библиотека была опубликована в апреле 2020 года в PyPI, а проект на GitHub имеет более 525 звезд. С более чем 185 тысячами установок за 1 год PyGAD является самой быстрорастущей библиотекой по сравнению с другими библиотеками для генетических алгоритмов. [8]

Название PyGAD состоит из 3 частей:

1) ***Py*** означает, что это библиотека Python. Это общее соглашение об именовании для библиотек Python.

2) ***GA*** расшифровывается как генетический алгоритм.

3) ***D*** для десятичного числа, поскольку библиотека изначально поддерживала только десятичный генетический алгоритм. Теперь она поддерживает как десятичный, так и двоичный генетический алгоритм.

PyGAD — это интуитивно понятная библиотека для решения задач оптимизации с использованием генетических алгоритмов. Она разработана с двумя основными целями: [8]

1) Сделать все максимально простым для пользователей с наименьшими знаниями.

2) Предоставление пользователю контроля над всем возможным.

Простота достигается за счет использования описательных имен для классов, методов, атрибутов и параметров. Это в дополнение к упрощению построения генетического алгоритма и заданию широкого спектра простых для

понимания параметров конфигурации. Существует меньше классов, методов или функций, которые нужно вызывать для решения проблемы, по сравнению с другими библиотеками.

Пользователям не нужно создавать цикл эволюции в любой ситуации, поскольку PyGAD поддерживает эластичный жизненный цикл, который может быть изменен. Это включает, но не ограничивается этим, включение или отключение операторов мутации или кроссовера и переопределение их для создания новых операторов в исследовательских целях.

В PyGAD включено 7 основных модулей:

- pygad.pygad: Это основной модуль, который строит все в генетическом алгоритме. Этот модуль неявно импортируется при импорте самой библиотеки;
- pygad.nn: Создает полностью связанные нейронные сети (FCNN) с нуля, используя только NumPy;
- pygad.gann: Использует модуль pygad для обучения сетей, построенных с использованием модуля nn;
- pygad.cnn: Аналогично модулю nn, но строит сверточные нейронные сети (CNN);
- pygad.gacnn: Аналогично модулю gann, но обучает CNNs;
- pygad.kerasga: Обучает модели Keras с использованием модуля pygad;
- pygad.torchga: Обучает модели PyTorch с использованием модуля pygad.

Учитывая, что pygad.pygad является наиболее важным модулем в библиотеке, ему уделяется наибольшее внимание. PyGAD имеет подробную документацию, которая охватывает все его функции с примерами. Кроме того, исходный код различных проектов, созданных с использованием PyGAD, объясняется в руководствах пользователя. [8] PyGAD приобрел популярность в последние месяцы, и некоторые из его статей и руководств на английском языке

переведены на разные языки, такие как корейский, турецкий, венгерский, китайский и русский.

При использовании библиотеки PyGAD выполняются три основных шага:

1. Создается фитнес-функции или функция приспособленности, определяющая качество особей в популяции;
2. Также создается экземпляр класса pygad.GA с соответствующими конфигурационными параметрами;
3. Вызывается метод run() для запуска процесса эволюции.

PyGAD имеет жизненный цикл, который помогает пользователю отслеживать и контролировать все различные этапы процесса эволюции. На рисунке 2 показан жизненный цикл PyGAD. Обратите внимание, что боковые блоки относятся к операциям, выполняемым между одним состоянием и другим.

Сначала проверяются переданные параметры в конструктор класса pygad.GA. Затем атрибуты экземпляра инициализируются в соответствии с переданными параметрами. Некоторые из этих атрибутов включают в себя популяцию, которая представляет собой массив NumPy, содержащий все решения данной популяции.

Как только мы успешно создали экземпляр класса pygad.GA, пришло время запустить жизненный цикл PyGAD, вызвав метод run(). Для 7 состояний в жизненном цикле PyGAD существуют следующие 7 callback-функций:

1. on_start(): Вызывается один раз после вызова метода run();
2. on_fitness(): Вызывается после вычисления пригодности популяции в каждом поколении;
3. on_parents(): Вызывается в каждом поколении после выбора родителей;
4. on_crossover(): Для каждого поколения вызывается после применения операции скрещивания;

5. on_mutation(): Для каждого поколения вызывается после применения мутаций;
6. on_generation(): Вызывается в конце каждого поколения;
7. on_stop(): Вызывается один раз после остановки метода run().

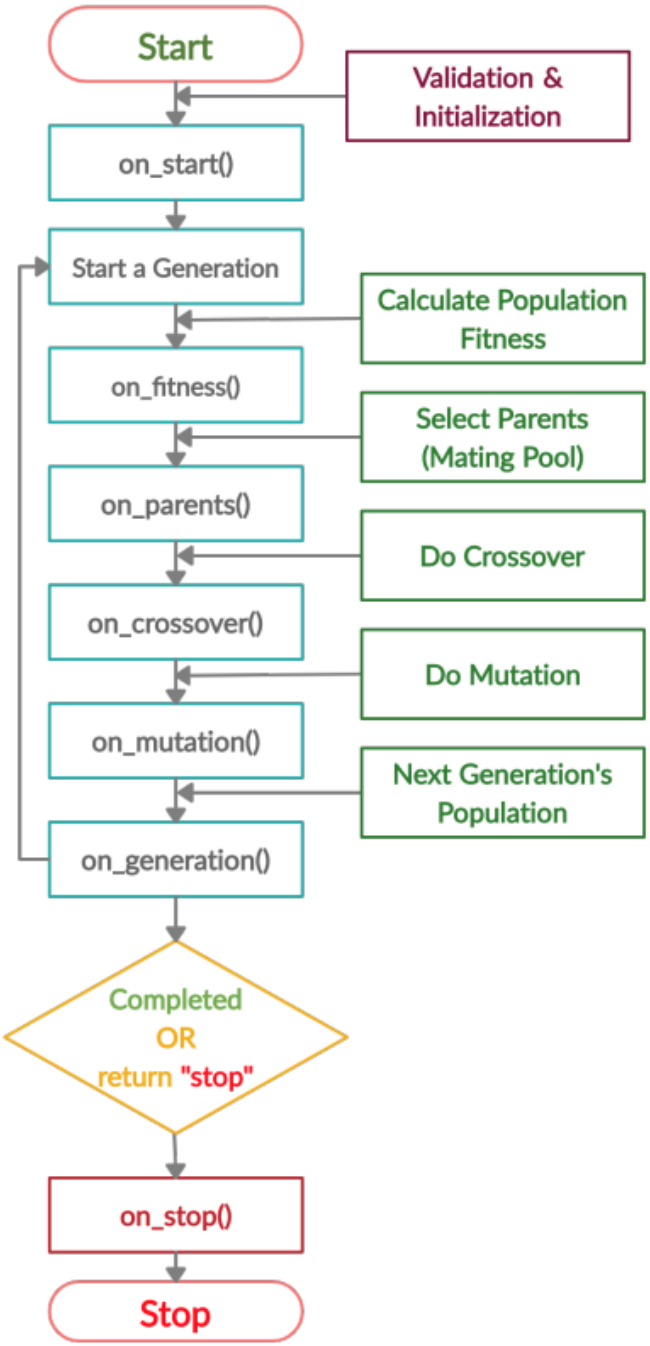

Рисунок 2 «Жизненный цикл PyGAD»

Пользователь может назначить функцию Python с соответствующими параметрами любой из этих callback-функций. Делая это, пользователь может отслеживать и контролировать каждый процесс от начала до конца эволюции. Например, пользователь может получать информацию о популяции, пригодности, выбранных родителях, результатах скрещивания или мутации, а также о наилучшем из найденных решений, и многое другое. Пользователь также может выполнять некоторые задачи предварительной обработки или последующей обработки в функциях on_start() и on_stop() соответственно.

В исследовательских целях и для поддержки операторов, которые не поддерживаются PyGAD, пользователь может определить пользовательские операторы скрещивания и мутации в callback-функциях.

Функция обратного вызова on_generation() может быть использована для добавления некоторых условий и изменения процесса выполнения. Например, возврат строки «stop» немедленно останавливает метод run() ранее, прежде чем завершатся все поколения. [8]

PyGAD — это интуитивно понятная библиотека, которая позволяет легко оптимизировать задачи всего за 3 шага. Данная библиотека поддерживает широкий диапазон параметров и атрибутов для обеспечения высокой степени настройки генетического алгоритма. К ним можно отнести: определение пространства значений для каждого гена, настройку типа данных каждого гена, обучение моделей Keras и PyTorch, отклонение дубликатов и многое другое. Библиотека PyGAD имеет жизненный цикл, позволяющий отслеживать процесс эволюции от начала и до конца. PyGAD поддерживает простой интерфейс для пользователей с небольшим опытом работы с Python, позволяющий решать проблемы с помощью нескольких строк кода и даже быстрее, чем другие библиотеки.

Практическая часть

В практической части данной работы опишем и повторим решение задачи воспроизведения изображения с помощью генетического алгоритма, реализованное в проекте под названием *GARI*.

GARI (Генетический алгоритм воспроизведения изображений) — это проект на Python, который использует библиотеку PyGAD для воспроизведения изображений с использованием генетического алгоритма. GARI воспроизводит одно изображение с использованием генетического алгоритма (GA) путем изменения значений пикселей. Этот проект работает как с цветными, так и с серыми изображениями. Для реализации генетического алгоритма используется библиотека PyGAD. [9]

Для воспроизведения изображения необходимо выполнить следующие действия:

- Загрузить изображение для воспроизведения;
- Подготовить функцию пригодности;
- Создать экземпляр класса pygad.GA с соответствующими параметрами;
- Запустить работу генетического алгоритма;
- Вывести графики с результатами работы программы и рассчитать некоторые статистические данные.

Сначала загрузим изображение для воспроизведения, представленное на рисунке 3. Для этого воспользуемся следующим кодом:

```
import imageio
import numpy

target_im = imageio.imread('kitty.jpg')
target_im = numpy.asarray(target_im/255, dtype=numpy.float)
```

Рисунок 3 «Изображение, подлежащее воспроизведению»

На основе представления хромосом, используемых в примере, значения пикселей могут находиться в диапазоне 0-255, 0-1, либо в любых иных. Стоит обратить внимание, что диапазон значений пикселей влияет на другие параметры, такие как диапазон, из которого выбираются случайные значения во время мутации, а также диапазон значений, используемых в исходной популяции. [9]

Следующий код создает функцию, которая будет использоваться в качестве фитнес-функции для вычисления значения пригодности для каждого решения в популяции:

```
target_chromosome = gari.img2chromosome(target_im)

def fitness_fun(solution, solution_idx):
    fitness = numpy.sum(numpy.abs(target_chromosome-solution))

    # Negating the fitness value to make it increasing rather than decr
easing.
    fitness = numpy.sum(target_chromosome) - fitness
    return fitness
```

Эта функция должна быть функцией максимизации, которая принимает два параметра: решение и его индекс. Функция возвращает величину пригодности особей.

Значение пригодности вычисляется с использованием суммы абсолютной разницы между значениями генов в исходной и воспроизведенной хромосомах. Функция gari.img2chromosome() вызывается перед функцией пригодности для представления изображения в виде вектора, поскольку генетический алгоритм может работать только с одномерными хромосомами. [9]

Очень важно использовать случайную мутацию и установить для параметра mutation_by_replacement значение True. В зависимости от диапазона значений пикселей, следует изменять значения, присвоенные параметрам infinite_range_low, limit_range_high, random_mutation_min_val и random_mutation_max_val. Если значения пикселей изображения находятся в диапазоне от 0 до 255, то необходимо установить init_range_low и random_mutation_min_val равными 0, но изменить init_range_high и random_mutation_max_val на 255. Также можно поэкспериментировать с другими параметрами для достижения лучших результатов. Ниже представлен код определения параметров нашего генетического алгоритма:

```
import pygad

ga_instance = pygad.GA(num_generations=20000,
                       num_parents_mating=10,
                       fitness_func=fitness_fun,
                       sol_per_pop=20,
                       num_genes=target_im.size,
                       init_range_low=0.0,
                       init_range_high=1.0,
                       mutation_percent_genes=0.01,
                       mutation_type="random",
                       mutation_by_replacement=True,
                       random_mutation_min_val=0.0,
                       random_mutation_max_val=1.0)
```

Для запуска генетического алгоритма необходимо выполнить функцию ga_instance.plot_result(). После того, как алгоритм закончит свою работу, с помощью функции ga_instance.plot_result() получим график изменения значения функции пригодности от поколения, представленный на рисунке 4.

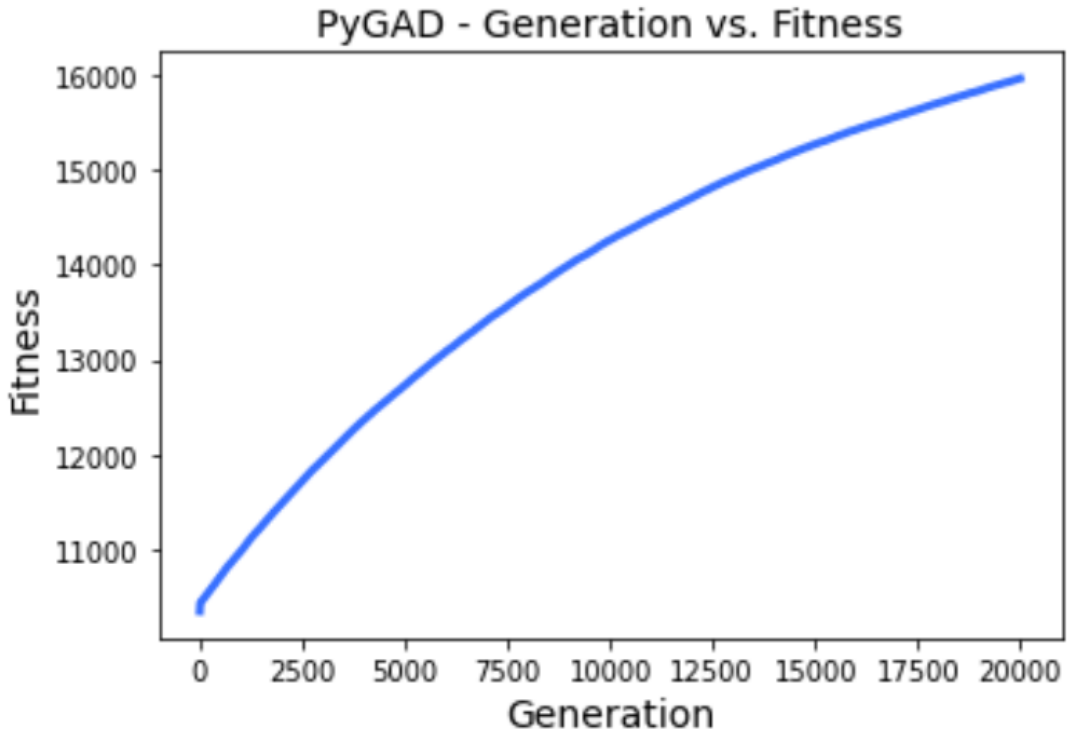

Рисунок 4 «График изменения значения функции пригодности от поколения»

С помощью кода ниже, выведем некоторую статистику работы нашего генетического алгоритма и лучшую полученную особь после 20000 поколений. Эти данные представлены на рисунке 5.

```
import matplotlib.pyplot

    # Returning the details of the best solution.
    solution, solution_fitness, solution_idx = ga_instance.best_solution()
    print("Fitness value of the best solution = {solution_fitness}".format(
    solution_fitness=solution_fitness))
    print("Index of the best solution : {solution_idx}".format(solution_idx
    =solution_idx))

    if ga_instance.best_solution_generation != -1:
```

```
    print("Best fitness value reached after {best_solution_generation}
generations.".format(best_solution_generation=ga_instance.best_solution
_generation))

result = chromosome2img(solution, target_im.shape)
matplotlib.pyplot.imshow(result)
matplotlib.pyplot.title("PyGAD & GARI for Reproducing Images")
matplotlib.pyplot.show()
```

```
Fitness value of the best solution = 15961.145244368026
Index of the best solution : 0
Best fitness value reached after 19996 generations.
```

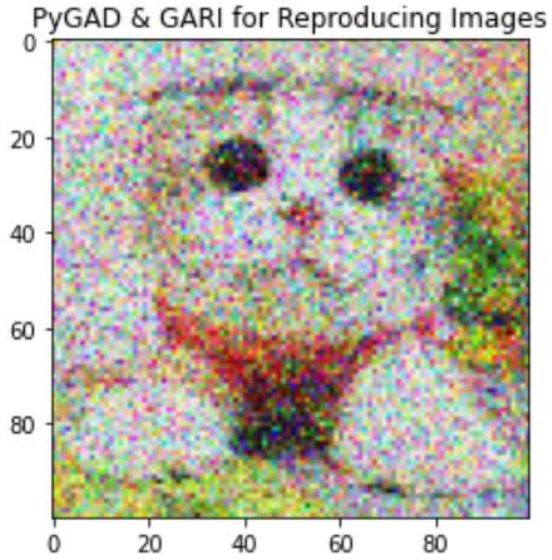

Рисунок 5 «Результаты работы генетического алгоритма»

Как видно, наилучшая особь была получена на 19996-м поколении. Учитывая особенности генетических алгоритмов, с помощью увеличения количества поколений можно улучшить данный результат.

Выводы

В данной работе было проведено исследование в области генетических алгоритмов и решен ряд поставленных задач, а именно: исследованы теоретические основы генетических алгоритмов, проведен обзор библиотеки PyGAD для работы с генетическими алгоритмами на языке программирования Python и решена задача воспроизведения изображения с помощью генетических алгоритмов.

Теоретическая часть работы содержит ключевую информацию из рассмотренного перечня научных трудов по генетическим алгоритмам, которая может быть полезна при изучении данной области. Положительные результаты практической части подтверждают предположения о том, что генетические алгоритмы могут быть успешно применены для решения задачи воспроизведения изображений.

Список использованных источников

1. Бураков М. В. Генетический алгоритм: теория и практика: учеб. пособие / М. В. Бураков. – СПб. : ГУАП, 2008. – 164 с.
2. Батищев Д.И., Неймарк Е.А., Старостин Н.В. Применение генетических алгоритмов к решению задач дискретной оптимизации. Учебно-методический материал по программе повышения квалификации «Информационные технологии и компьютерное моделирование в прикладной математике». Нижний Новгород, 2007, 85 с.
3. Генетический алгоритм как метод оптимизации [Электронный ресурс]. – Электрон. дан. – URL: https://knowledge.allbest.ru/programming/2c0b65635a3ad69a4d43a89521316c26_0.html
4. De Jong, K.A. Introduction to the second special issue on genetic algorithms. Machine Learning, 5(4), 351-353.
5. Вирсански Э. Генетические алгоритмы на Python / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 286 с.: ил.
6. Панченко, Т. В. Генетические алгоритмы [Текст] : учебно-методическое пособие / под ред. Ю. Ю. Тарасевича. — Астрахань : Издательский дом «Астраханский университет», 2007. — 87 [3] с.
7. Журавлев С.Ю., Терсков В.А. Применение генетических алгоритмов при оптимизации функционирования сложных механических систем // Вестник КрасГАУ. 2008. №4. С. 215-220.
8. PyGAD: An Intuitive Genetic Algorithm Python Library [Электронный ресурс]. – Электрон. дан. – URL: https://paperswithcode.com/paper/pygad-an-intuitive-genetic-algorithm-python
9. Reproducing Images using a Genetic Algorithm with Python [Электронный ресурс]. – Электрон. дан. – URL: https://heartbeat.comet.ml/reproducing-images-using-a-genetic-algorithm-with-python-91fc701ff84