

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра информационных технологий

КУРСОВАЯ РАБОТА

**СИМВОЛЬНАЯ РЕГРЕССИЯ ДЛЯ ОПИСАНИЯ КОНТУРНЫХ
ИЗОБРАЖЕНИЙ НА ПЛОСКОСТИ**

Работу выполнил _____ Т.Э. Айрапетов
(подпись)

Направление подготовки 02.03.03 Математическое обеспечение и
администрирование информационных систем курс 3

Направленность Технология программирования

Научный руководитель
д-р физ.-мат. наук _____ А.И. Миков
(подпись)

Нормоконтролер
канд. пед. наук, доц. _____ А.В. Харченко
(подпись)

Краснодар
2024

РЕФЕРАТ

Курсовая работа содержит 30 страниц, 21 рисунок, 5 источников.

СИМВОЛЬНАЯ РЕГРЕССИЯ, КОНТУРНЫЕ ИЗОБРАЖЕНИЯ,
ЭВОЛЮЦИОННОЕ ПРОГРАММИРОВАНИЕ, PYTHON.

Объектом исследования являются контурные изображения и применение методов символьной регрессии, что является значимой областью развития технологий обработки и анализа изображений.

Целью данной курсовой работы является исследование и анализ эффективности применения методов символьной регрессии, в частности, с использованием библиотеки PySR, в задаче описания контурных изображений на плоскости.

Работа направлена на изучение возможности описания контурных изображений с помощью символьной регрессии, а также особенностей работы методов символьной регрессии, оценку их производительности в разных случаях, выявление преимуществ и недостатков каждого метода. В основные задачи работы входило проведение экспериментов по разделению данных, полученных из контурных изображений, а также экспериментов с моделями символьной регрессии.

В результате выполнения курсовой работы были получены оценки качества применения методов символьной регрессии в разных случаях, а также оценки применимости наивного подхода к разделению данных контурных изображений. Сформированные выводы могут послужить практическим руководством для дальнейшего изучения подходов к эффективному разделению контурных изображений и настройки подходов к применению моделей символьной регрессии в зависимости от характеристик конкретной задачи и набора данных.

СОДЕРЖАНИЕ

Введение	4
1 Обзор предметной области	5
1.1 Контурные изображения	5
1.2 Сбор точек контурного изображения	6
1.3 Символьная регрессия.....	6
1.4 Применимость символьной регрессии в описании контурных изображений	10
2 Описание хода работы	12
2.1 Реализация алгоритмов разделения данных.....	12
2.2 Реализация алгоритмов символьной регрессии	18
Заключение	29
Список использованных источников	30

ВВЕДЕНИЕ

В современном мире обработка изображений является важной областью развития информационных технологий. В эту область входит множество различных задач, успешное решение которых требует постоянного совершенствования методов и алгоритмов обработки изображений.

Символьная регрессия представляет собой класс методов, направленных на построение математического выражения, наиболее точно описывающего входные данные. В большинстве случаев эти методы основаны на генетическом программировании, однако есть также и нейросетевые подходы.

Целью данной работы является изучение возможности использования методов символьной регрессии в задаче описания контурных изображений для дальнейшего анализа или воспроизведения.

В ходе исследования рассматриваются следующие аспекты: процесс выбора точек из контурного изображения, разделение точек на однородные или схожие области, настройка и использование методов символьной регрессии, а также оценка точности и надежности различных подходов.

Это исследование имеет важное теоретическое значение, поскольку затрагивает актуальность выбранной тематики и целесообразность выбора технологий символьной регрессии в задаче описания изображений, что позволит задать вектор развития будущих подходов в решении поставленной задачи.

1 Обзор предметной области

1.1 Контурные изображения

Контурные изображения, с точки зрения обработки изображений, являются особой категорией графических данных, в которых внимание акцентируется на границах объектов, обозначая их форму и структуру. В отличие от растровых изображений, где каждый пиксель представляет собой отдельный цвет или яркость, контурные изображения образуются набором точек, соединенных линиями или кривыми.

Контурные изображения играют важную роль в областях компьютерного зрения, медицинской диагностики, инженерии и других областях, где необходима высокоточный анализ и интерпретация графических данных. Например, контурные изображения используются для обнаружения и распознавания объектов на изображениях, в медицинской диагностике - для анализа медицинских снимков и выявления патологий, а в инженерии - для проектирования и моделирования объектов и систем. Пример контурного изображения представлен на рисунке 1.

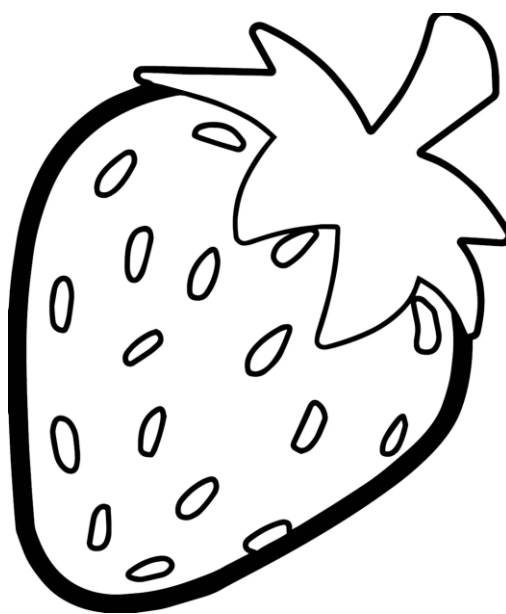


Рисунок 1 – Пример контурного изображения

1.2 Сбор точек контурного изображения

Если рассматривать контурные изображения с точки зрения компьютера, то это все тот же набор пикселей, однако для получения контуров могут использоваться алгоритмы компьютерного зрения, реализованные, например, в OpenCV. Тем не менее, несмотря на кажущуюся простоту контурных изображений, на них могут быть изображены и переплетены линии или объекты разных форм, что потенциально может усложнить дальнейший анализ полученных контуров. Это утверждение открывает одну из главных проблем использования символьной регрессии.

Для разделения полученных данных могут использоваться алгоритмы сегментации [1]. Большинство алгоритмов сегментации, такие как алгоритм водораздела или обнаружение краев, основаны на разнице цветов, однако в случае контурных изображений нам предоставлены лишь координаты распознанных точек соответствующего контура. Исходя из этого можно попробовать описать алгоритм для разделения контуров на более простые для аппроксимации части.

Также возможным вариантом для решения этой задачи может быть использование нейросети, которую необходимо будет обучить на распознавание и разделение контуров на относительно простые области.

В ходе выполнения работы будут рассмотрены наивные методы разделения данных (кластеризация, визуальное ручное разделение).

1.3 Символьная регрессия

Символьная регрессия – это метод аппроксимации, заключающийся в подборе модели из пространства математических выражений, которая наилучшим образом соответствует входным данным как с точки зрения точности, так и с точки зрения простоты. В отличие от традиционных методов

регрессии, символьная регрессия стремится найти не просто локальные соотношения в данных, но и глобальные функциональные зависимости [2].

Основная идея алгоритмов символьной регрессии – генетическое программирование. На основе генетического программирования алгоритмы символьной регрессии создают и эволюционируют математические выражения, представляющие собой символьные формулы или деревья выражений. Этот процесс основан на принципах биологической эволюции, таких как мутация, скрещивание и отбор, и позволяет создавать и улучшать модели, которые могут довольно точно описывать сложные функциональные зависимости в данных. На рисунке 2 представлена схема работы символьной регрессии.

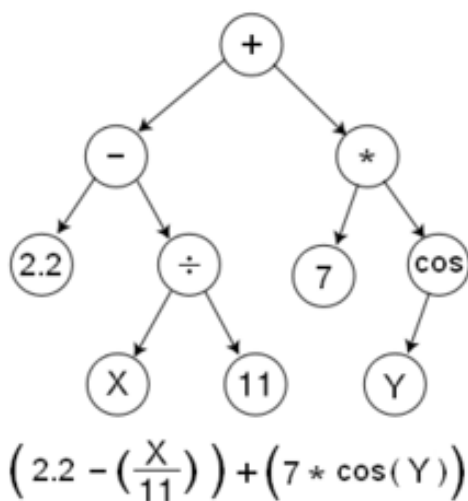


Рисунок 2 – Схема работы символьной регрессии

Одной из ключевых особенностей символьной регрессии является её способность к автоматическому отбору признаков и построению интерпретируемых моделей. В процессе эволюции символьные выражения могут включать только те переменные и операции, которые действительно важны для описания данных, что делает их более устойчивыми к переобучению и обеспечивает лучшую интерпретируемость результатов.

Таким образом, символьная регрессия представляет собой мощный и гибкий метод анализа данных, который находит широкое применение в

различных областях, требующих описания сложных функциональных зависимостей и создания интерпретируемых моделей.

Несмотря на все плюсы, было доказано, что символьная регрессия является NP-сложной проблемой в том смысле, что не всегда можно найти наилучшее математическое выражение, подходящее для данного набора данных за полиномиальное время. Тем не менее, если искомая функция не слишком сложная, можно точно решить задачу символической регрессии, сгенерировав все возможные функции (построенные из некоторого заранее определенного набора операторов) и вычислив их на рассматриваемом наборе данных.

Другим подходом решения задачи символьной регрессии является использование нейронных сетей. Одним из перспективных вариантов является использование архитектуры KAN (Kolmogorov-Arnold Network). Сама архитектура не нова, но её успешная реализация появилась лишь в этом году.

Согласно теореме Колмогорова-Арнольда, любая непрерывная функция n переменных может быть представлена в виде композиции одномерных функций. Исследователи заметили, что по аналогии с перцептроном мы можем на каждом слое построить матрицу обучаемых объектов. Просто в нашем случае это будут не параметры (числа), а функции.

Перемещение функций на ребра довольно сильно изменяет характеристики нейросети. Далее будут перечислены ключевые аспекты, отличающие KAN от перцептрона [3].

Точность. За счет того, что в KAN мы обучаем функции, а не числа, можно повысить точность сети без переобучения её с нуля. В многослойном перцептроне чтобы добиться лучшей точности, мы можем увеличивать количество слоев и нейронов, но это требует полноценного переобучения и работает не всегда. В KAN достаточно просто добавить больше точек в сетку аппроксимации. Это гарантирует лучший результат, и при этом не нужно переучивать нейросеть.

Интерпретируемость. KAN более интерпретируем, чем многослойный перцептрон. А ведь интерпретируемость – это одна из главных проблем современных нейросетей.

Аппроксимация. KAN лучше справляется с аппроксимацией сложных математических функций. В статье [4] показано, что KAN на порядок лучше решает дифференциальные уравнения и может (пере)открыть законы физики и математики.

Скорость обучения. У архитектуры есть бутылочное горлышко: KAN учится медленнее MLP примерно в 10 раз. Возможно, это станет серьезным камнем преткновения, а возможно инженеры быстро научатся оптимизировать эффективность таких сетей.

На рисунке 3 представлено сравнение архитектур многослойного перцептрона и KAN.

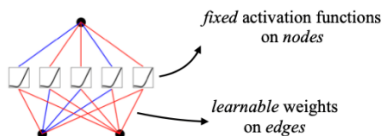
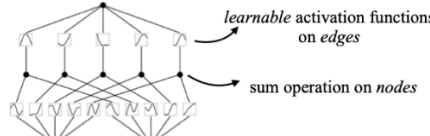
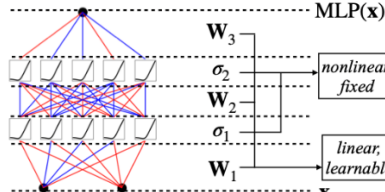
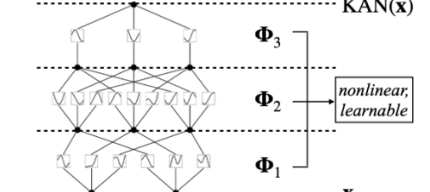
Model	Multi-Layer Perceptron (MLP)	Kolmogorov-Arnold Network (KAN)
Theorem	Universal Approximation Theorem	Kolmogorov-Arnold Representation Theorem
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(e)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^n \phi_{q,p}(x_p) \right)$
Model (Shallow)	(a) 	(b) 
Formula (Deep)	$\text{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\text{KAN}(\mathbf{x}) = (\Phi_3 \circ \Phi_2 \circ \Phi_1)(\mathbf{x})$
Model (Deep)	(c) 	(d) 

Рисунок 3 – Сравнение многослойного перцептрона и KAN

1.4 Применимость символьной регрессии в описании контурных изображений

Символьная регрессия представляет собой мощный метод аппроксимации функций, который может быть эффективно применен к анализу и описанию контурных изображений на плоскости. В данном контексте контурные изображения представляют собой наборы точек, образующих линии или кривые, задающие форму объектов на изображении.

Однако, применение символьной регрессии в описании контурных изображений на плоскости может столкнуться с несколькими значительными проблемами, которые необходимо учитывать при разработке соответствующих методов анализа.

Первая проблема заключается в необходимости четкого разделения контуров на простые области, подлежащие аппроксимации. В некоторых случаях контуры могут быть сложными и содержать различные элементы, такие как вогнутости, выпуклости или разрывы. Также контуры на изображении могут быть переплетены, что делает разделение еще более трудной задачей. Для успешного применения символьной регрессии необходимо предварительно выполнить этап сегментации контуров, который позволит разбить изображение на более простые и однородные области для последующего анализа.

Для разделения контуров на составные части можно сделать предположение об их природе, однако это представляется сложным без контекста. В зависимости от конкретного приложения и требований к анализу, контуры могут иметь различные характеристики и структуры, что затрудняет разработку универсального алгоритма разделения контуров на составные части. Это требует дополнительного исследования и разработки специализированных методов анализа контурных изображений для конкретных задач.

Для эффективного разделения контуров на составные части могут использоваться нейросети. Например, сверточные нейронные сети (CNN), демонстрируют высокую способность к извлечению признаков из визуальных данных и обучению сложных пространственных зависимостей. В контексте анализа контурных изображений, сверточные нейронные сети могут обучаться на размеченных данных, где контуры явно выделены, и затем применяться для сегментации контуров на новых изображениях. Тем не менее, разметка данных для обучения требует серьезного анализа и временных затрат.

Вторая проблема связана с вычислительными ресурсами, которые могут потребоваться для применения символьной регрессии. Поскольку этот метод может работать с высокой степенью точности и моделировать сложные зависимости между точками контура, он может потребовать значительных вычислительных мощностей и времени для обучения модели и выполнения прогнозов, особенно при работе с большими и сложными наборами данных.

Учитывая эти проблемы, необходимо тщательно анализировать и выбирать подходящие методы предобработки данных и параметры моделирования, чтобы эффективно применять символьную регрессию в анализе контурных изображений. Такой подход позволит минимизировать влияние указанных проблем и повысить качество и точность получаемых результатов.

В некоторых случаях, когда основная цель заключается в уменьшении размеров изображения или его представлении в более компактной форме, альтернативой символьной регрессии может стать векторизация изображений. Векторизация представляет изображение в виде набора векторов и точек, что позволяет сохранить его форму и структуру без потери качества. Этот подход может быть особенно полезен, когда контуры относительно просты и не требуют высокой степени аппроксимации.

2 Описание хода работы

В данном разделе описываются процессы проведения работы по разделению данных, обучению и оценке моделей символьной регрессии. Вся работа выполнена на языке Python с использованием различных библиотек, таких как: `opencv`, `scikit-learn`, `numpy`, `matplotlib`, `deap`, `pykan`. Также использовались вспомогательные библиотеки для создания пользовательского интерфейса.

`Numpy` используется для работы с многомерными массивами и матрицами, а также для выполнения математических операций над этими массивами. Она обеспечивает высокую производительность и удобные функции для работы с данными. `Matplotlib` используется для построения различного рода графиков. `OpenCV` (Open Source Computer Vision Library) – это библиотека компьютерного зрения и обработки изображений с открытым исходным кодом. Она предоставляет обширный набор функций и алгоритмов для работы с изображениями и видео, включая обнаружение объектов, распознавание лиц, сегментацию изображений, вычисление гистограмм, а также возможности для работы с камерами и видеопотоками. `scikit-learn` является мощной библиотекой для проведения экспериментов в области машинного обучения, которая включает в себя реализации различных алгоритмов. `DEAP` (Distributed Evolutionary Algorithms in Python) – это библиотека для реализации эволюционных алгоритмов. Она предоставляет удобные инструменты для создания генетических программ и других эволюционных методов оптимизации.

2.1 Реализация алгоритмов разделения данных

В ходе работы была попытка применить методы кластеризации к контурам с целью выделить однородные области. В качестве примера

использовалась спектральная кластеризация, реализованная в пакете `scikit-learn`. На рисунке 4 представлен листинг кода спектральной кластеризации.

```
from sklearn import cluster
spectral = cluster.SpectralClustering(eigen_solver="arpack", affinity="nearest_neighbors")
clusters = spectral.fit_predict(np.array(xs).reshape(-1,1), ys)
print(set(clusters))
plt.figure(figsize=(8, 6))
plt.scatter(xs, ys, c=clusters, cmap='viridis')
plt.xlabel('X')
plt.ylabel('Y')
plt.colorbar(label='Кластер')
plt.show()
```

Рисунок 4 – Листинг кода применения кластеризации к массиву точек

На рисунке 5 представлен пример использования кластеризации для контурного изображения сердца. Может показаться, что данные разделились довольно неплохо и мы потенциально сможем использовать данный подход в нашей задаче. Однако такое поведение возможно только при явном указании количества кластеров (в нашем случае 2), а также нельзя гарантировать что в общем случае (рисунок 6) данные могут быть разделены на действительно однородные области. Применение других методов кластеризации также не дало успешных результатов.

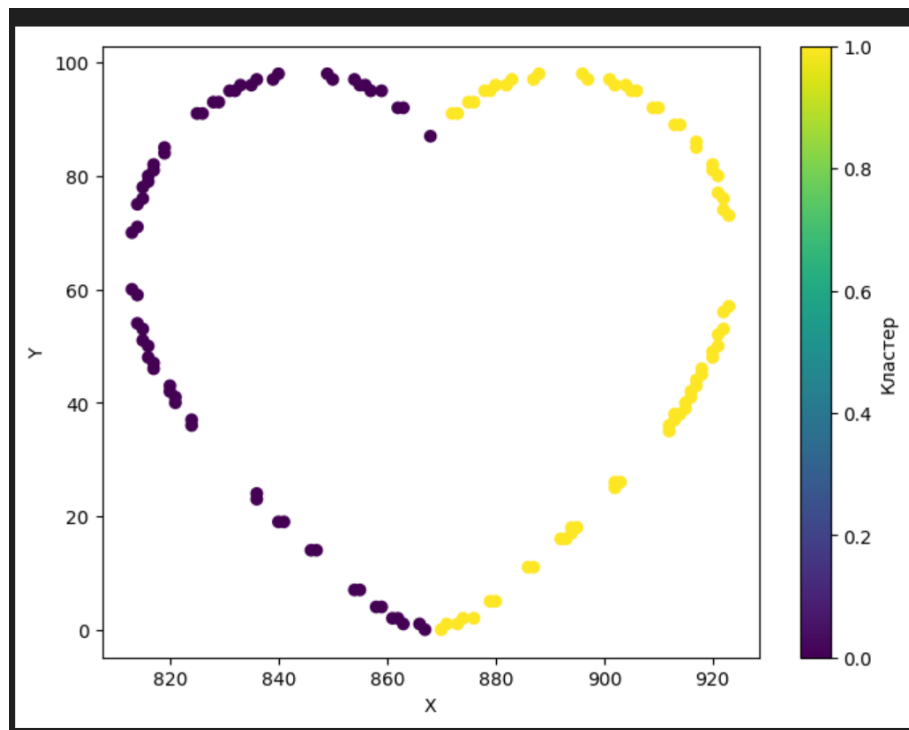


Рисунок 5 – Пример успешного применения кластеризации к данным

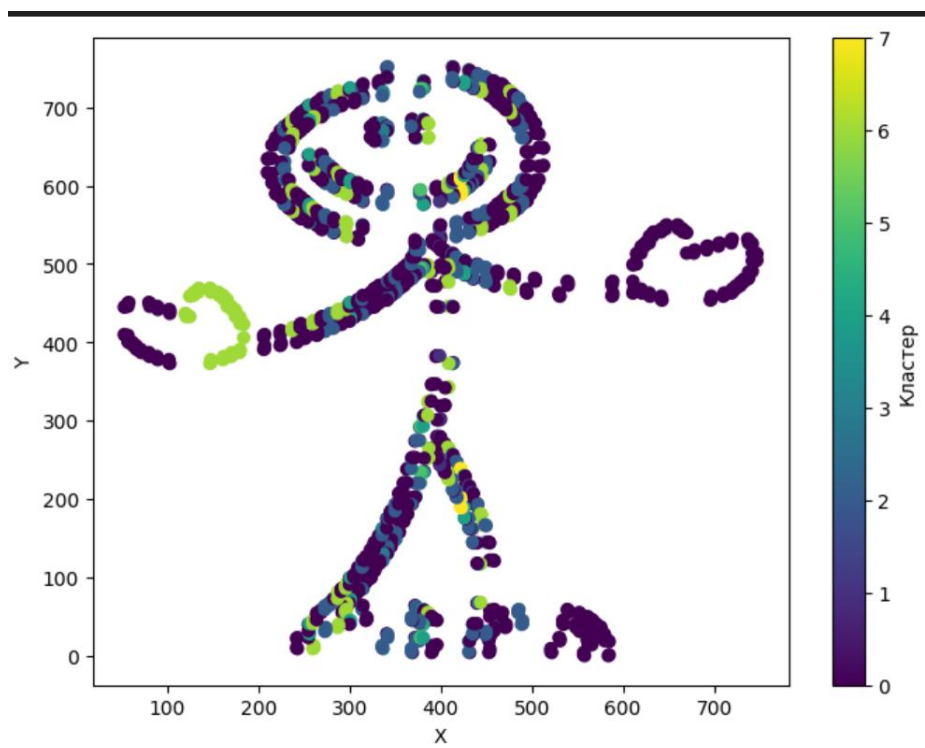


Рисунок 6 – Пример неудачного применения кластеризации к данным

Для изучения аспектов разделения двумерных данных, было разработано приложение на Tkinter. Цель приложения состояла в

автоматизации процесса сбора, анализа и разделения контурных изображений на более мелкие части, с целью дальнейшего прогнозирования их символического представления.

Основные шаги работы приложения:

- загрузка изображения: пользователь может загрузить черно-белое контурное изображение в формате PNG, JPEG и другие распространенные форматы;
- выделение контуров: после загрузки изображения приложение использует библиотеку OpenCV для выделения контуров на изображении (рисунок 7);
- отображение контуров и списка: выделенные контуры отображаются на изображении, а также предоставляется список контуров в виде элементов checkbox;
- выбор контуров: пользователь может выбирать контуры из списка, чтобы далее проводить с ними различные операции;
- усреднение значений по оси Y: реализована функция "Усреднение по y", которая изменяет каждый выбранный контур, оставляя для каждого значения x только одно значение y – усредненное среди всех значений, соответствующих текущему x;
- объединение контуров: пользователь может объединять выбранные контуры в один, что позволяет упростить дальнейший анализ и описание изображения;
- настройка порога: реализована возможность настройки порогового значения для выделения контуров, что позволяет пользователю управлять чувствительностью алгоритма выделения контуров;
- сохранение результатов: после проведения необходимых операций пользователь может сохранить результаты работы в текстовый файл для дальнейшего анализа или использования в других программах.

```

def load_image(self, path=''):
    if path:
        file_path=path
    else:
        file_path = filedialog.askopenfilename()
    self.path = file_path
    threshold_value = self.threshold_scale.get()
    if file_path:
        self.image = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
        self.original_image = cv2.imread(file_path, cv2.IMREAD_COLOR)
        _, threshold = cv2.threshold(self.image, threshold_value, 255, cv2.THRESH_BINARY)
        self.contours, _ = cv2.findContours(threshold, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        # self.show_image()
        self.create_contour_checkboxes()
        self.update_image()

```

Рисунок 7 – Листинг кода выделения контуров

Пример работы программы приведен на рисунке 8. Зеленым цветом обозначены распознанные и выбранные пользователем контуры. Уже на этом шаге можно отметить сложность дальнейшего разделения данных, так как даже если мы сможем отделить, например, голову от тела, дальнейшая аппроксимация головы вызывает вопросы. Также проблемой является то, что при анализе изображения библиотекой OpenCV, контурами считаются области на границе двух цветов (белого и черного), и в последствии контуры как бы «оборачивают» линии изображения, а не идут по нему. Эту проблему можно решить, например, применив методы векторизации.

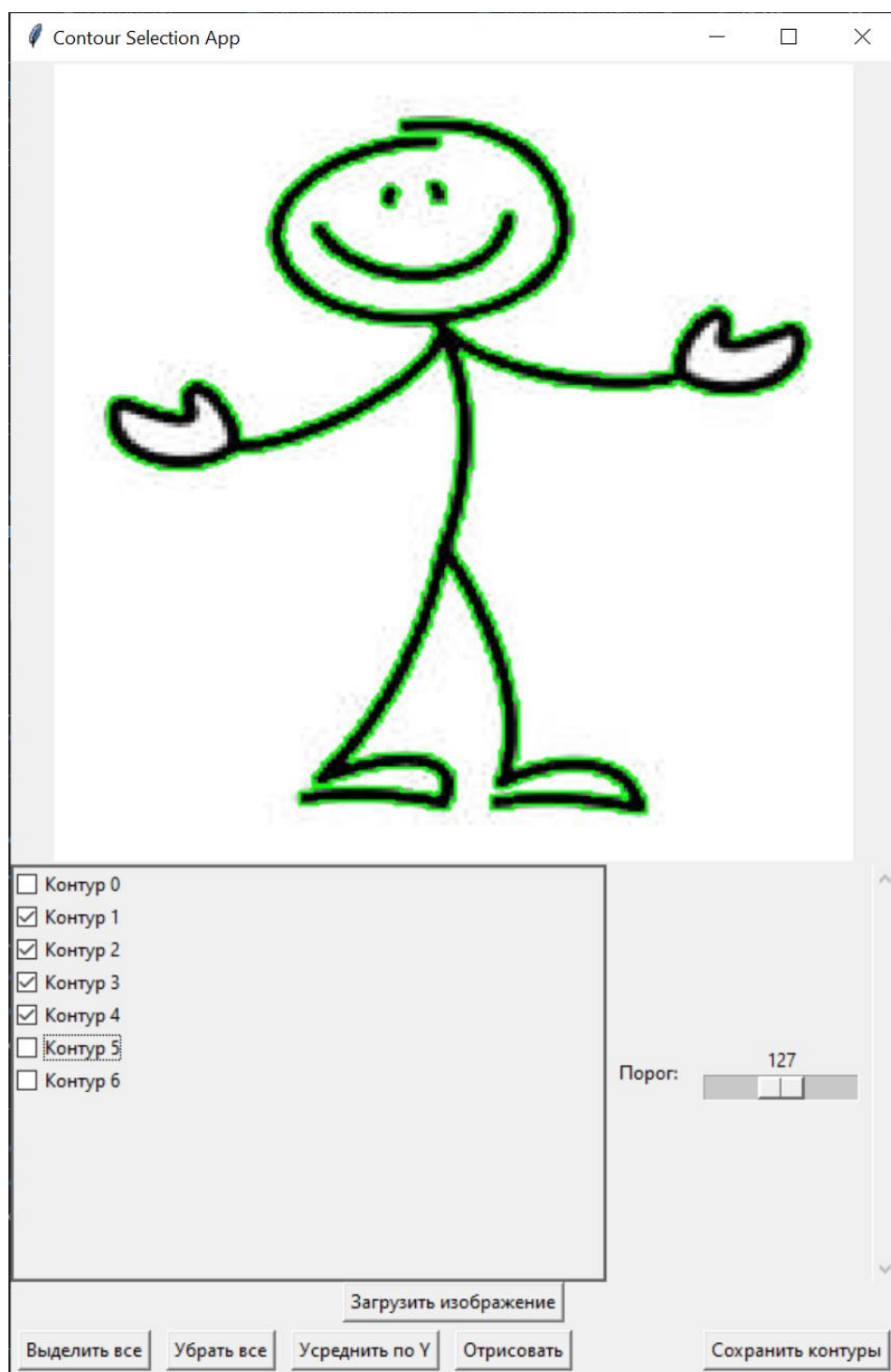


Рисунок 8 – Пример работы приложения для разделения контуров

Говоря о векторизации, сложно не отметить возможность замены символьной регрессии методами векторизации особенно в случаях, когда символьная регрессия применяется для эффективного масштабирования изображения в дальнейшем. Тем не менее, методы векторизации могут включать в себя методы машинного обучения, что может требовать

дополнительных исследований. Пример эффективности векторизации изображений можно увидеть на рисунке 9. В обычных случаях векторизация уменьшает размер примерно в 1.5 раза, увеличивая при этом интерпретируемость изображения и добавляя возможность масштабирования без потери качества.



Рисунок 9 – Пример эффективности векторизации

2.2 Реализация алгоритмов символьной регрессии

В этом разделе рассматриваются два подхода к построению модели символьной регрессии, а также используется библиотека с открытым исходным кодом PySR для сравнения результатов.

Первый подход. Использование библиотеки DEAP для построения алгоритма генетического программирования. Вначале создаются классы `FitnessMin` и `Individual` с помощью модуля `creator`. `FitnessMin` используется для минимизации значения целевой функции, а `Individual` представляет собой примитивное дерево (выражение).

Далее необходимо описать набор примитивов для создания выражений (деревьев). Это включает в себя арифметические операции, математические функции `sin`, `cos`, `tan` и другие, а также константу `rand`. На рисунке 10 представлен листинг кода перечисления примитивов.

Для работы алгоритма нужно зарегистрировать функции, используемые в генетическом программировании, такие как создание выражений, создание особей, создание популяции, компиляция и оценка.

Функцию оценивания необходимо передать модели как параметр. Опишем эту функцию как квадрат разности между фактическим значением переменной y и значением рассматриваемого выражения в точке x .

После этих шагов необходимо задать начальное значение популяции (набор индивидуальных решений), задать количество поколений оптимизации и передать все параметры в основной цикл эволюции (`algorithms.eaSimple`).

В каждой итерации происходит скрещивание (`mate`), мутация (`mutate`), отбор (`select`) и оценка (`evaluate`). Все это продолжается на протяжении `ngen` поколений.

```
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", gp.PrimitiveTree, fitness=creator.FitnessMin)

pset = gp.PrimitiveSet("MAIN", arity=1)
pset.addPrimitive(operator.add, arity=2)
pset.addPrimitive(operator.sub, arity=2)
pset.addPrimitive(operator.mul, arity=2)
pset.addPrimitive(protectedDiv, arity=2)
pset.addPrimitive(sqrt, arity=1)
pset.addPrimitive(np.sin, arity=1)
pset.addPrimitive(np.cos, arity=1)
pset.addPrimitive(np.tan, arity=1)
pset.addEphemeralConstant("rand", lambda: random.uniform(-1, 1))
pset.renameArguments(ARG0='x')
```

Рисунок 10 – Листинг кода перечисления примитивов

Рассмотрим некоторые примеры применения данного алгоритма для случайных зависимостей. На рисунке 11 представлен итог 100 поколений развития формул на абсолютно случайных данных. При отсутствии зависимости как таковой, алгоритм чаще всего уменьшает ошибку до какого-то предела и перестает улучшать формулу. Зачастую итоговая зависимость является константной и не несет в себе никакой пользы. Существует также

другая проблема – при большом количестве поколений формула разрастается до невероятных масштабов, вызывая ошибку у интерпретатора.

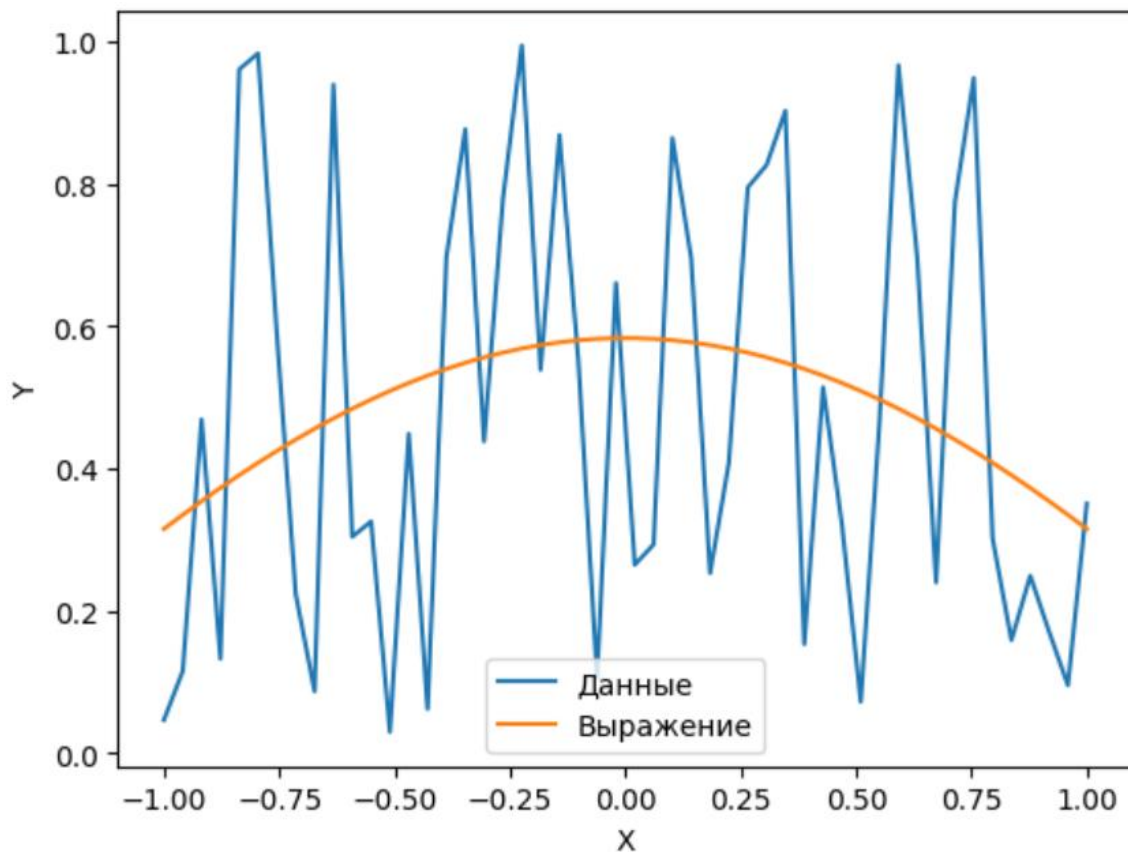


Рисунок 11 – Пример запуска генетического алгоритма на случайных данных

На рисунке 12 представлен результат выполнения алгоритма для данных, удовлетворяющих зависимости $y = x \cdot \sin(5 \cdot x)$. Можно заметить, что итоговое выражение неплохо удовлетворяет заданным условиям, однако если посмотреть на само выражение, то станет понятно, что это скорее случайность чем закономерный результат (итоговое выражение: $x \cdot (-x^{**3} - x \cdot \sqrt{x^{**2}} + x) \cdot \sqrt{\text{protectedDiv}(0.9925505057717932, x \cdot \sqrt{\text{protectedDiv}(0.9925505057717932, x)})} \cdot (-x^{**3} - x \cdot \sqrt{x^{**2}} + x)$)

И тут надо отметить, что все это время мы задавали рисунки на маленьких промежутках – от -1 до 1 по x и всего 100 точек. Теперь попробуем

увеличить размер нашей выборки до 1000 при границах от -10 до 10. На рисунке 13 изображен результат вычислений при всего 50 поколениях.

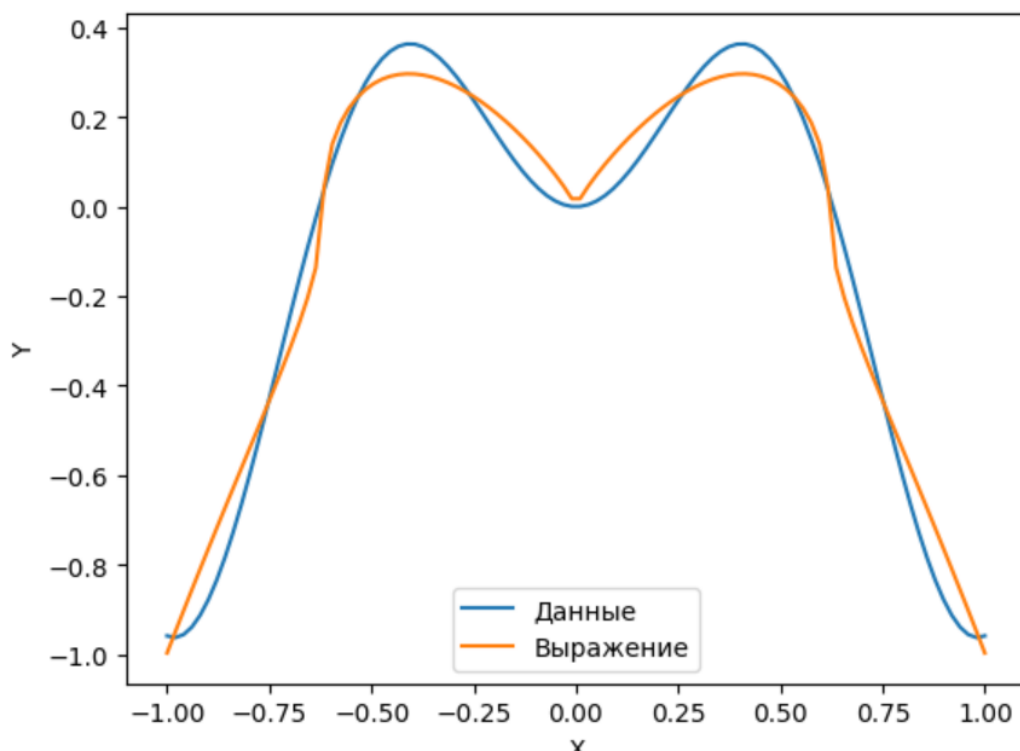


Рисунок 12 – Пример запуска алгоритма на функции $x \cdot \sin(5 \cdot x)$

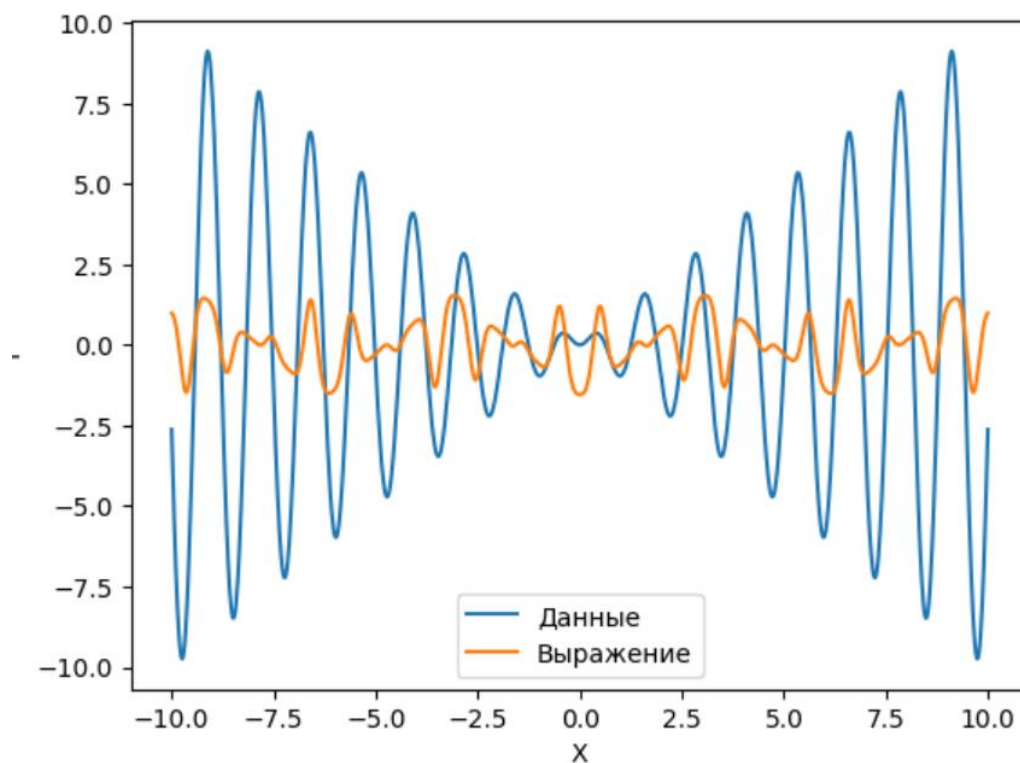


Рисунок 13 – Пример запуска алгоритма при увеличении выборки и границ

Отсюда можно сделать вывод: залог хорошей аппроксимации — это передача большого набора данных. Также стоит отметить, что при увеличении выборки, также увеличивается и время выполнения, но результат оправдывает ожидания — в данном случае выражение получилось компактнее $\cos(\text{protectedDiv}(\cos(\text{protectedDiv}(x, 0.3233209543714832)), -0.33259622893160423)) * \tan(\cos(x))$. Для оптимизации времени выполнения можно более детально задавать параметры модели при каждом испытании, если у нас есть информация о природе нашей выборки.

Второй подход. Использование библиотеки `rukan`, реализующей подход KAN. Код, реализуемый в данной библиотеке, был написан относительно недавно (в конце апреля 2024 года), поэтому не все может работать стабильно, однако уже сейчас данный подход находит применение во многих областях. Из плюсов можно выделить высокую скорость и качество работы почти без настройки. На рисунке 14 представлен весь код создания модели.

```
from kan import *

# формируем KAN: 2D входы, 1D выходы, 5 скрытых нейронов,
# кубические сплайны и сетка на 5 точках.
model = KAN(width=[1,2,1], grid=5, k=3, seed=0)

f = lambda x: x**2
dataset = create_dataset(f, ranges=[-1, 1], n_var=1)
dataset['train_input'].shape, dataset['train_label'].shape
66] ✓ 0.0s

.. (torch.Size([1000, 1]), torch.Size([1000, 1]))

> v
model.train(dataset, opt="LBFGS", steps=10, lamb=0.01, lamb_entropy=10.)
68] ✓ 2.6s

.. train loss: 4.07e-02 | test loss: 4.19e-02 | reg: 4.60e+00 : 100%|█| 10/10 [00:02.
.. {'train loss': [array(0.07094129),
```

Рисунок 14 – Код создания и обучения модели KAN

Первый пример: обычная парабола (Рисунок 15).

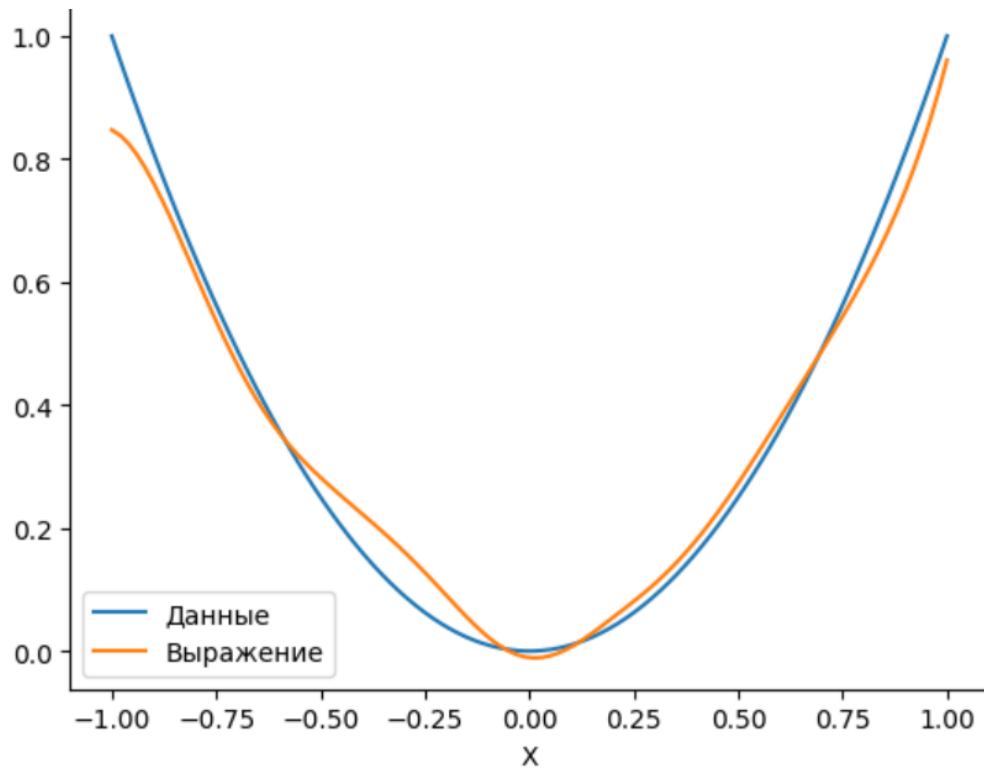


Рисунок 15 – Пример работы KAN на 10 итерациях

Посмотрев на график, можно сказать, что KAN неплохо аппроксимирует исходную зависимость, так как ошибка составляет всего $4.07e-02$. Увеличение числа итераций не дает большой прирост к качеству аппроксимации, равно, как и увеличение обучающей выборки. Однако, если задать границы не $[-1, 1]$, а $[-10, 10]$, ошибка уменьшится.

Рассмотрим теперь способность модели на данных, соответствующих зависимости $y = x \cdot \sin(5 \cdot x)$. На рисунке 16 отображен результат обучения.

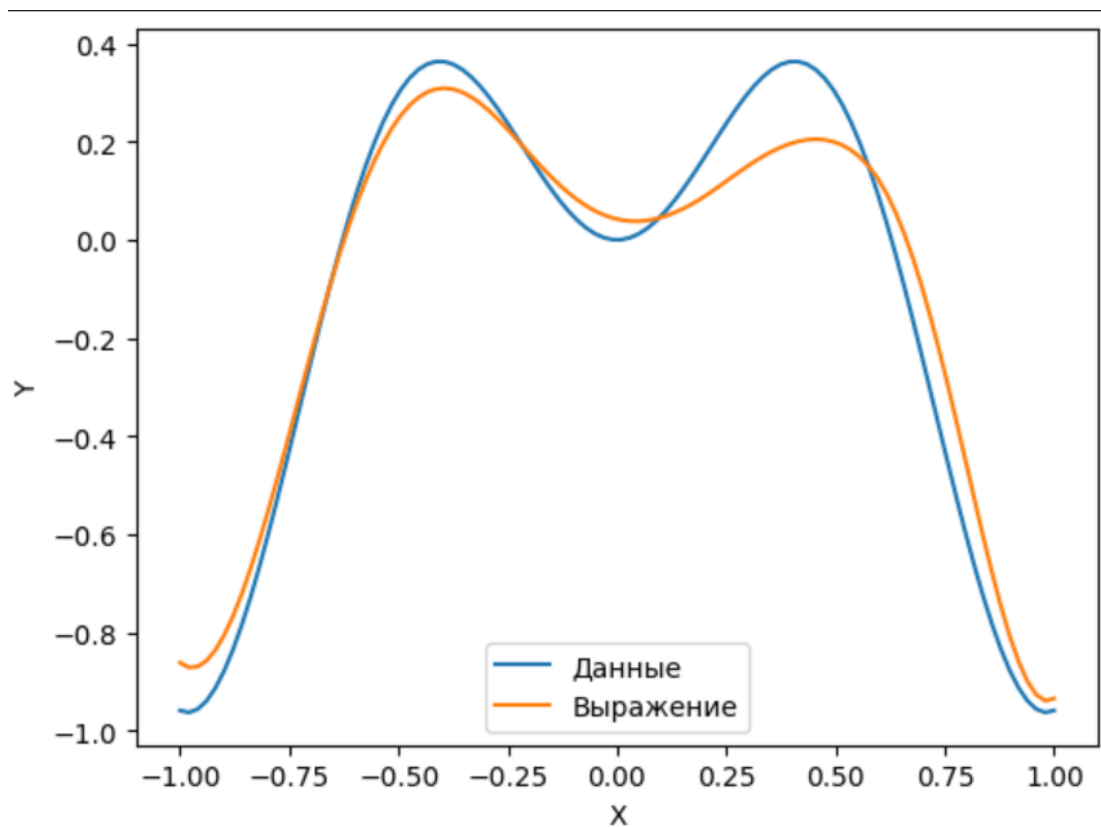


Рисунок 16 – Пример обучения модели KAN на более сложной зависимости

Как и в случае с первым примером, зададим более длинный промежуток точек и увеличим количество эпох. По итогам обучения модель показала не очень хороший результат в аппроксимации исходной зависимости при разных параметрах обучения. Тем не менее, при увеличении количества скрытых нейронов (до 10), модель смогла лучше соответствовать исходным данным (рисунок 17). Это не очень хорошо, так как, не зная сложность исходной зависимости, для достижения лучшей аппроксимации придется подбирать параметры сети.

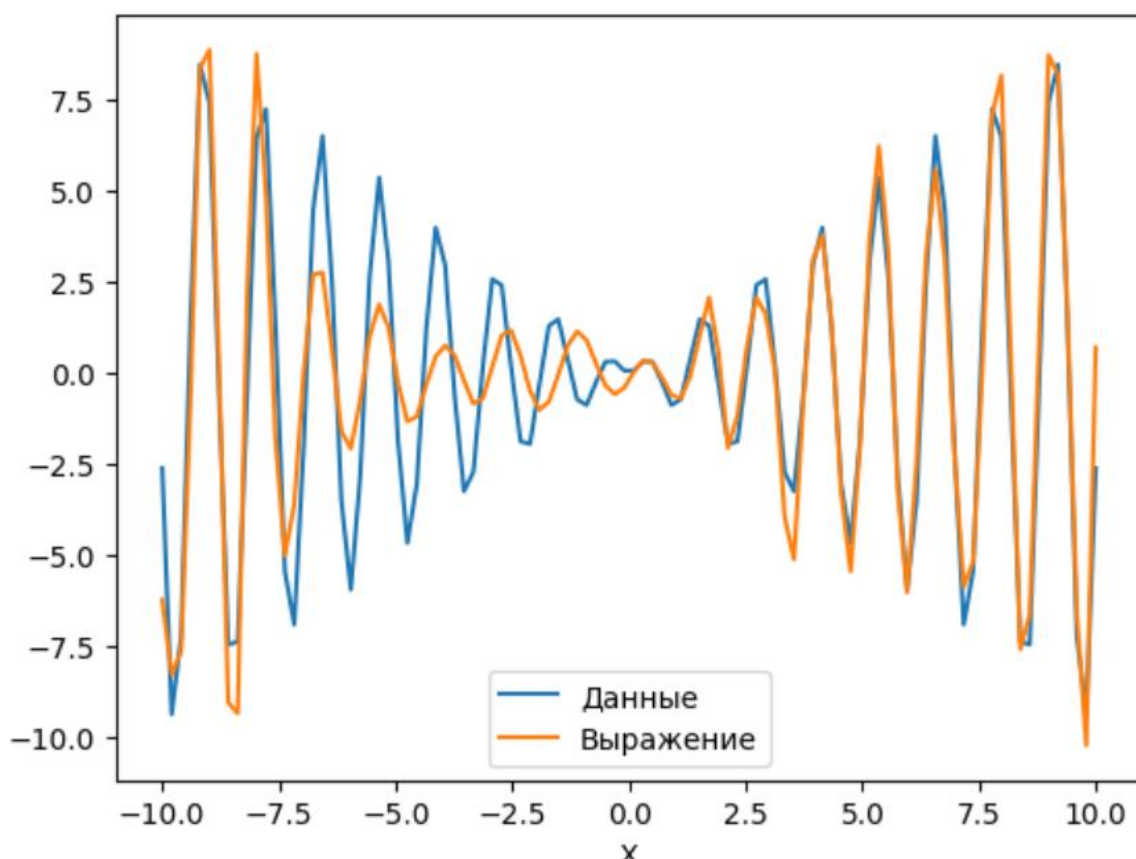


Рисунок 17 – Модель KAN на увеличенном промежутке с изменением параметров сети

В применении данного подхода есть свои недостатки – почти всегда мы не сможем получить исходную формулу зависимости. Это задает ограничение на применимость, если в нашей задаче требуется не просто перенять зависимость на текущем наборе точек, а обобщить зависимость для дальнейших исследований. Пример того, насколько неточным может быть модель вне диапазона значений выборки можно увидеть на рисунке 18.

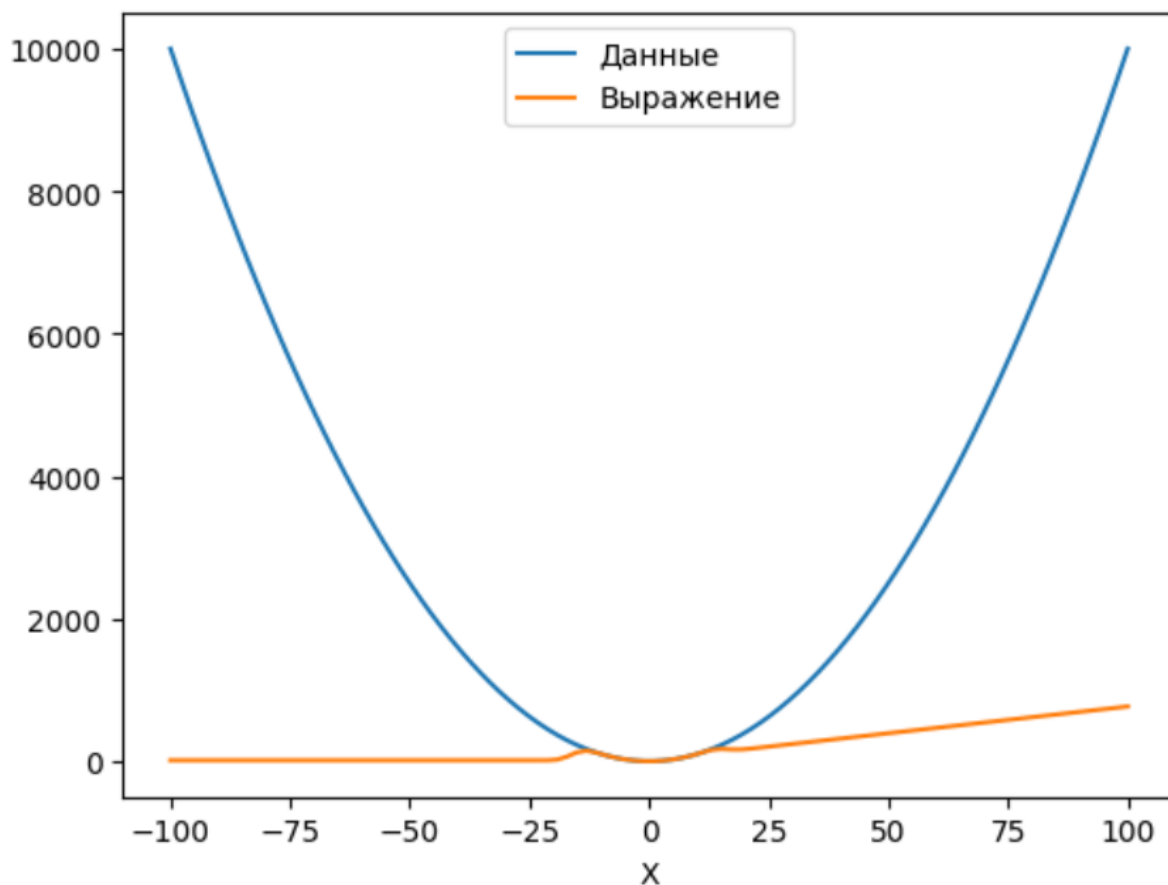


Рисунок 18 – Неточность зависимости вне диапазона выборки

На рисунке 18 можно заметить, что в диапазоне примерно от -10 до 10 (на котором мы обучали нейросеть), KAN довольно неплохо соответствует искомой параболе, однако вне этого диапазона модель плохо себя проявляет.

Теперь можно сравнить эти два подхода с более продвинутым подходом к символьной регрессии. Библиотека PySR [5] была разработана для демократизации и популяризации символьной регрессии в науке и построена на высокопроизводительном распределенном бэкенде, гибком алгоритме поиска и интерфейсах с несколькими пакетами глубокого обучения. Внутренний алгоритм поиска PySR представляет собой многопопуляционный эволюционный алгоритм, который состоит из уникального цикла evolve-simplify-optimize, предназначенного для оптимизации неизвестных скалярных констант во вновь найденных эмпирических выражениях. Бэкэнд PySR – это

чрезвычайно оптимизированная библиотека Julia SymbolicRegression.jl4 которая может быть использована непосредственно из Julia.

Код создания и тренировки модели PySR представлен на рисунке 19.

```
default_pysr_params = dict(  
    | populations=30,  
    | model_selection="best",  
    | )  
✓ 0.0s
```

```
model = PySRRegressor(  
    | niterations=1,  
    | binary_operators=["+", "*", "-", "/"],  
    | unary_operators=["cos", "exp", "sin", "tan", "abs", "log"],  
    | **default_pysr_params,  
    | )  
  
model.fit(X, y)  
✓ 0.4s
```

Рисунок 19 – Код создания и обучения модели PySR

При обучении на выборке, так как модель хорошо отлажена и оптимизирована, она смогла найти искомую зависимость всего за 1 поколение. На графике на рисунке 20 видна только одна линия, а ошибка во время подбора выражения составила 5.189122e-16, что можно приравнять к нулю, а значит модель полностью нашла исходную зависимость.

В следующем примере использовалась зависимость $y = x \cdot \sin(5 \cdot x)$ и в этом случае модели понадобилось больше поколений (10) для аппроксимации (рисунок 21). Тем не менее, стабильно хороший результат требует либо большего числа генераций, либо большего числа попыток, так как алгоритм основан на случайном переборе.

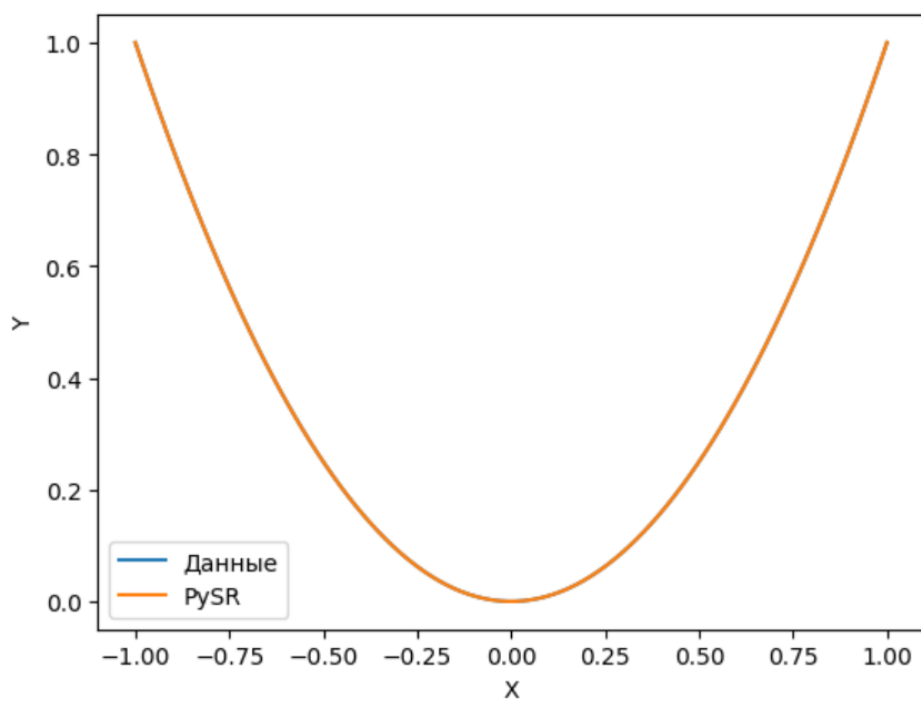


Рисунок 20 – PySR на наборе x^2

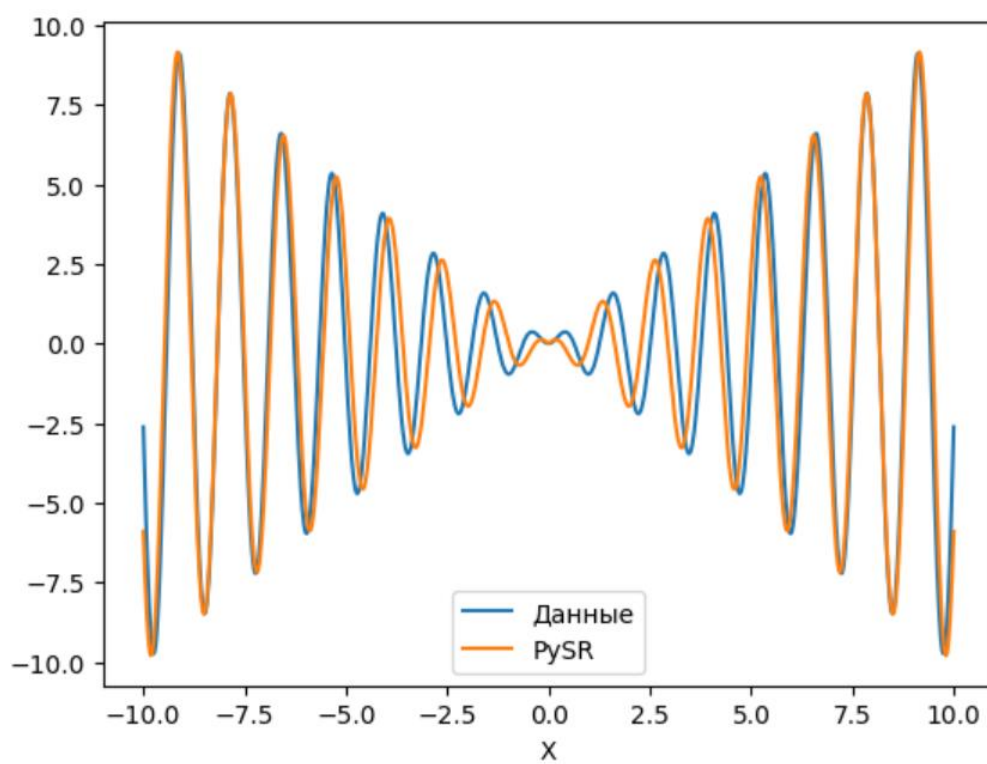


Рисунок 21 – PySR на наборе $x \cdot \sin(5 \cdot x)$

ЗАКЛЮЧЕНИЕ

За время выполнения работы были изучены аспекты задачи символьной регрессии для описания контурных изображений на плоскости. Проведенные исследования позволили выявить требования, предъявляемые к потенциально разрабатываемому решению, и проанализировать различные подходы к решению поставленной задачи.

Полученные результаты говорят о том, что выбор подхода к символьной регрессии должен соответствовать предметной области, в которой будет применяться решение. Если классические методы генетического программирования могут прийти до известного простого решения за довольно быстрый промежуток времени, то использование подхода KAN должно сопровождаться четкой надобностью в аппроксимации на данном промежутке.

Стоит отметить, что применение символьной регрессии непосредственно для представления контурных изображений в виде набора формул, требует более продуманного подхода, такого как использование нейросетей для сегментации контурных изображений.

Также можно подчеркнуть, что данная работа не только позволила лучше понять принципы обработки изображений и символьной регрессии, но и задала вектор развития потенциальным подходам в решении данной задачи. Полученные знания и опыт послужат отличным фундаментом для дальнейших исследований и разработок в области символьной регрессии контурных изображений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Stockman G. Computer Vision / G. Stockman, L. G. Shapiro – 2001 – 608 с. – ISBN:978-0-13-030796-5
2. Symbolic regression // Wikipedia URL: https://en.wikipedia.org/wiki/Symbolic_regression (дата обращения: 12.05.2024).
3. Kolmogorov-Arnold Networks // Data Secrets URL: <https://datasecrets.ru/articles/9> (дата обращения: 12.05.2024).
4. Liu Z., Wang Y., Vaidya S., Ruehle F., Halverson J., Soljačić M., Y. Hou T., Tegmark M. KAN: Kolmogorov-Arnold Networks – 2024 <https://doi.org/10.48550/arXiv.2404.19756>
5. Cranmer M. Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl – 2023 <https://doi.org/10.48550/arXiv.2305.01582>