

Автоматизация поиска архитектур искусственных нейронных сетей

А. А. Часовских¹, В. С. Половников², А. А. Хусаенов³,
Г. В. Боков⁴, А. Ю. Коновалов⁵, А. С. Дробышев⁶,
В. А. Бирюкова⁷

Представлен обзор работ о методах автоматизации генерирования архитектур искусственных нейронных сетей. Наиболее подробно излагается материал по использованию для этого больших

¹ *Часовских Анатолий Александрович* — доктор физико-математических наук, профессор каф. математической теории интеллектуальных систем мех.-мат. ф-та МГУ, e-mail: chasovskikh@mail.ru.

Chasovskikh Anatoly Alexandrovich — Doctor of Physical and Mathematical Sciences, Professor, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Mathematical Theory of Intellectual Systems.

² *Половников Владимир Сергеевич* — кандидат физико-математических наук, старший научный сотрудник каф. математической теории интеллектуальных систем мех.-мат. ф-та МГУ, e-mail: pvser@mail.ru

Polovnikov Vladimir Sergeevich — Candidate of Physical and Mathematical Sciences, Senior Researcher, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Mathematical Theory of Intellectual Systems.

³ *Хусаенов Артём Азатович* — аспирант каф. математической теории интеллектуальных систем мех.-мат. ф-та МГУ, e-mail: a.khusaenov@mail.ru

Khusaenov Artem Azatovich — Postgraduate, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Mathematical Theory of Intellectual Systems.

⁴ *Боков Григорий Владимирович* — кандидат физико-математических наук, доцент каф. математической теории интеллектуальных систем мех.-мат. ф-та МГУ, e-mail: bokovgrigoriy@gmail.com

Bokov Grigoriy Vladimirovich — Candidate of Physical and Mathematical Sciences, Associate Professor, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Mathematical Theory of Intellectual Systems.

⁵ *Коновалов Александр Юрьевич* — кандидат физико-математических наук, младший научный сотрудник лаб. математических проблем искусственного интеллекта мех.-мат. ф-та МГУ, e-mail: alexandr.konoval@gmail.com

Konovalov Aleksandr Yurievich — Candidate of Physical and Mathematical Sciences, Junior Researcher, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Laboratory of Mathematical Problems of Artificial Intelligence.

⁶ *Дробышев Александр Сергеевич* — аспирант каф. математической теории интеллектуальных систем мех.-мат. ф-та МГУ, e-mail: drobyshev.sanya@yandex.ru

Drobyshev Alexander Sergeevich — Postgraduate, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Mathematical Theory of Intellectual Systems.

⁷ *Бирюкова Вероника Андреевна* — аспирант каф. математической теории интеллектуальных систем мех.-мат. ф-та МГУ, e-mail: biryukovaveronika@mail.ru

Biryukova Veronika Andreevna — Postgraduate, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Mathematical Theory of Intellectual Systems.

языковых моделей. Для экспериментов использован ChatGPT & Midjourney, который доступен в мессенджере Telegram.

Ключевые слова: искусственная нейронная сеть, большая языковая модель, поиск архитектуры нейронной сети, генеративный предварительно обученный преобразователь.

1. Введение.

Поиск оптимальной архитектуры искусственной нейронной сети (ИНС) – важная задача глубокого обучения. Процедуры автоматизации этого процесса объединены общим термином Neural Architecture Search (NAS). Методика решения заключается в определении целевой функции, позволяющей оптимизировать точность функционирования нейронной сети путем варьирования ее характеристик при ограничении на количество итераций варьирования. Это типичный подход, при выполнении которого тестируются и отбраковываются тысячи архитектур нейронных сетей. Каждая итерация заключается в обучении и тестировании модели ИНС на выбранной базе данных.

Разработка хорошей обучаемой архитектуры ИНС – сложный процесс для экспертной разработки. NAS предполагает автоматическое нахождение архитектуры нейронной сети, которая будет иметь высокую эффективность после обучения. В обзоре [1] по NAS, вышедшем 2019 году, дана классификация разработанных к тому времени систем в трех измерениях: пространство поиска архитектур ИНС, стратегия поиска и стратегия оценки производительности.

Пространство архитектур, в котором осуществляется поиск нужной архитектуры ИНС, может быть существенно ограничено, если используются экспертные знания о характере решаемой задачи и типичных свойствах архитектур. С другой стороны, такие ограничения могут исключить эффективные не изученные варианты архитектур. Наиболее простые пространства архитектур – цепи, состоящие из слоев соединенных последовательно. Более сложный вариант – ИНС с ветвящейся архитектурой. Работа [2] посвящена стратегии построения архитектур из блоков. Оптимизация отдельных блоков, выбор их количества и порядка соединения позволяет существенно сократить время поиска нужной архитектуры. Построение сетей из однородных блоков продемонстрировало эффективность в LSTM и RNN. При таком подходе к поиску архитектур возникает микроуровень для оптимизации ячеек и макроуровень для построения сети из ячеек.

Стратегия поиска в пространстве архитектур, которое может быть бесконечным, реализует компромисс между временем, затрачиваемым на поиск и эффективностью полученного решения. К основным стра-

тегиям поиска в пространстве архитектур, согласно [1], следует отнести случайный поиск, байесовскую оптимизацию, эволюционные методы, обучение с подкреплением (Reinforcement Learning, RL) и градиентные методы. Использование NAS с байесовской оптимизацией позволило получить современные архитектуры машинного зрения [3] и ИНС, способные превзойти результаты распознавания экспертов [4]. Низкая сложность байесовских методов объясняет их широкую популярность [5]. Но используются они чаще для оптимизации гиперпараметров сетей, как более эффективные для оптимизации в пространствах низкой размерности.

Метод обучения с подкреплением, связан с обучением агента (как правило, другой нейронной сети) предлагать архитектуры нейронных сетей для решения данной задачи, используя показатель эффективности предложенной архитектуры после ее обучения в качестве вознаграждения [6]. Большинство известных алгоритмических подходов используют обучение с подкреплением или эволюционные алгоритмы.

Можно считать, что эволюционный подход продолжает исследования по нейроэволюции, начатые еще в 1970-х годах. При использовании эволюционных методов нужную архитектуру ищут непосредственно в пространстве архитектур ИНС, без использования генератора [7], а эффективность обученной ИНС служит показателем приспособляемости. К недостаткам эволюционных алгоритмов следует отнести то, что они не используют большого накопленного опыта генерирования ИНС. Этим может объясняться большое количество перебираемых архитектур ИНС в процессе их работы.

Метод случайного поиска по дереву (Monte Carlo Tree Search) тоже использует поиск в пространстве возможных архитектур [8]. Во всех перечисленных случаях от эксперта требуется определение пространства допустимых архитектур.

Стратегия оценки производительности связана с сокращением времени на проверку эффективности разрабатываемого решения. Самый простой способ оценивания – обучить выбранную модель ИНС и протестировать ее, но при многократном выполнении это приводит к высоким затратам по времени. Следует выделить следующие подходы для сокращения общего времени обучения:

- сокращение количества эпох обучения, обучение на подмножестве данных, использование уменьшенных архитектур;
- экстраполяцию показателей эффективности после нескольких эпох обучения;
- использование весов предыдущей обученной модели для определения стартовых весов новой модели;

- обучение единственной модели, веса которой распределяются между различными архитектурами, которые являются подграфами, вообще говоря, пересекающимися, обученной модели.

Но наибольшая производительность при оценке эффективности предложенной архитектуры достигается, если использовать предсказание качества этой архитектуры, например, другой нейронной сетью [9].

Для того, чтобы не ограничивать проектировщика в выборе архитектур путем определения пространства архитектур, целесообразно использовать большие языковые модели (Large Language Models, LLM), которые обучаются на больших массивах данных, например, на открытых репозиториях из Github. Эффективность при генерации программного кода, большой объем обучающего материала по ранее разработанным эффективным архитектурам ИИС является важным аргументом в пользу использования LLM.

2. Большие языковые модели.

Широкому использованию LLM способствовало открытие трансформеров [10]. Большинство предшествовавших трансформерам системы обработки последовательной информации было разработано на основе сложных рекуррентных или сверточных архитектурах, которые включали кодировщик и декодер. Наиболее эффективные из этих моделей связывали кодировщик и декодер через механизм внимания. Трансформер, предложенный в работе [10], имел простую архитектуру кодировщик-декодер без рекуррентных и сверточных включений, основанную исключительно на механизме внимания в обеих частях архитектуры (кодировщик, декодер). Эксперименты на двух задачах перевода, с английского языка на немецкий и с английского на французский, показали лучшие результаты. При этом трансформеры поддаются хорошо распараллеливаются и требуют меньшего времени на обучение, чем предшествующие модели.

Открытие трансформеров [11]–[13] способствовало взрывному росту больших языковых моделей. Ориентированный на конкретную прикладную задачу подход, используемый для решения поставленной задачи, связанной с обработкой естественного языка (система вопрос-ответ, автоматический перевод, реферирование и др.), основан на процедуре обучения с учителем на специально разработанной обучающей базе данных. В [11] для решения языковых задач предложено использовать генеративного предварительно обученного преобразователя (generative pre-training transformer, GPT). Процесс обучения модели GPT можно выполнить в два этапа. На первом этапе выполняется обучение без учителя, на втором этапе выполняется тонкая настройка с учителем. Авторами показано,

что прикладные задачи могут быть решены с использованием предварительно обученной языковой модели GPT на неразмеченном большом корпусе языковых данных методом обучения с подкреплением при ее последующей дискриминационной тонкой настройке (discriminative fine-tuning) на данных, отражающих специфику прикладной задачи. Эффективность разработанного GPT по сравнению с существовавшими на тот момент решениями продемонстрирована на 9 из 12 изученных авторами языковых задач. В работе [12] предложена сеть GPT-2, содержащая 1,5 миллиарда параметров, обученная на большой языковой базе WebText, которая справляется с некоторыми прикладными задачами без дополнительного обучения на специально разработанных для этих задач обучающих языковых базах данных.

Использование трансформеров позволило не только создавать современные модели для основных задач обработки естественного языка, они привели к созданию моделей для широкого спектра других задач, таких как разработка кода видеоигр [14], [15]. EvoProming [16] – метод, использующий код LLM в качестве операторов мутации и скрещивания для реализации NAS в задачах классификации MNIST-1D [17] и тестирования алгоритмических рассуждений CLRS [18]. Поскольку эффективность модели можно повысить тривиальным образом, увеличивая количество параметров, большие модели дополнительно штрафуются из-за своего размера. Таким образом, предпочтение отдается малым моделям.

3. Поиск архитектуры ИНС с использованием метода LLMatic.

В работе [19] представлен метод автоматического поиска архитектур ИНС с использованием LLM и разработанного авторами генетического алгоритма. Метод использует характеристику сложности модели как показатель разнообразия, поиск высокоэффективных моделей выполняется при вариации их размера. Следует отметить, что LLM не имеет механизмов оптимизации решений. Для этого авторы работы [19] предлагают использовать генетический Quality-Diversity (QD) алгоритм. QD-метод [20] объединяет семейство эволюционных алгоритмов, которые в дополнение к оптимизации метрики приспособляемости находят разнообразные решения в соответствии с индивидуальными особенностями использования модели. Вместо того, чтобы сохранять популяцию наиболее эффективных особей, QD-методы, такие, как рассмотренный в [21] MAP-Elites, сохраняют популяции для заданных диапазонов «поведенческих характеристик».

QD-методы могут использоваться для управления роботами. Например, если оптимальная траектория окажется недоступной из-за препятствия или поломки, придётся использовать другие решения. Эти методы способствуют устойчивости функционирования решений, избеганию попадания популяций в процессе эволюции в локальный минимум.

В работе [19] предложено соединить возможности генерации кода LLM с разнообразием и устойчивостью генетического QD – алгоритма. Это позволило авторам разработать алгоритм LLMatic для автоматизации нахождения архитектуры ИНС. Если в других работах LLM управляет NAS напрямую через подсказки, то LLMatic использует QD-алгоритм для формирования разнообразных и надежных решений.

Алгоритм LLMatic протестирован авторами на базе MNIST. Показана возможность построения эффективной архитектуры за примерно 2000 поисковых запросов без использования предварительной информации о базе данных и без использования какой-либо известной заранее эффективно работающей на этой базе модели. При этом интерес для авторов представляет поиск не единственного решения, оптимизирующего заданную целевую функцию, а определение набора решений для использования в различных условиях. Например, различие в оптимальных архитектурах может возникать из-за разного объема оперативной памяти на клиентском устройстве. LLMatic начинает поиск с очень простой нейронной сети, разработанной в соответствии с работой Стенли [22], в которой предполагается, что нейроэволюция происходит успешнее, если начинается с простой сети. Эта сеть содержит всего один сверточный слой и один полносвязный слой. Для LLMatic используются метод двунаправленной совместной оптимизации, в котором два отдельных архива используются для хранения взаимодополняющих компонентов, которые можно объединить для решения задачи поиска наиболее «приспособляемой» популяции сетей. В первом архиве сохраняется нейронная сеть, для которой характеристиками поведения являются значения глубины и ширины, а также количество операций с плавающей точкой в секунду (FLOPS). FLOPS был выбран вместо количества параметров, поскольку FLOPS лучше коррелирует с фактическим временем, затраченным на обучение сети. Этот архив назван «архив сетей». Значением приспособляемости для этой сети является значение ошибки на тестовом множестве, достигнутой после обучения сети. Второй архив называется «архив подсказок». Он содержит подсказку и температуру для генерации кода, которые также являются поведенческими дескрипторами. Выбор подсказки и температуры зависит от того, добавлена ли сгенерированная сеть в архив сетей и насколько она обучаемая. Значение приспособляемости для подсказки зависит также от того, насколько добавленная сеть эффективнее сети предыдущего поколения.

Внешний цикл алгоритма LLMatic индексируется номером поколения. Количество поколений фиксировано и является параметром алгоритма. В течении одного поколения осуществляется проверка заполнения обоих архивов. Если архивы недостаточно заполнены, то заданное количество раз с использованием случайно выбранной подсказки, начального значения температуры, значения ошибки наилучшей из сгенерированных сетей операцией мутации исходной сети генерируется новая сеть, температура и новая подсказка, которой добавляются в соответствующие архивы. После заполнения обоих архивов до заданных объемов, в каждом поколении алгоритм LLMatic в течении заданного количества раундов выполняет операции мутации или скрещивания. На каждом раунде выполняется генерация новой сети, а во время операции мутации, кроме того, генерируется новая подсказка. По прошествии очередного поколения в результате тестирования новых сгенерированных сетей обновляется минимальная ошибка сети, и архивы пополняются сгенерированными сетями и подсказками. Архив сетей и архив подсказок используются для выполнения операций мутации и скрещивания.

4. Использование метода GPT-NAS

Идея метода автоматического генерирования моделей GPT-NAS состоит в том, что генерирующая модель, предварительно обученная на корпусе данных большого объема, может обобщить фундаментальные принципы построения нейронных архитектур. Таким образом, GPT-NAS использует модель GPT чтобы сформировать подходящие компоненты архитектуры с использованием базовых, а затем использует эволюционную стратегию для поиска оптимального решения. Такой подход может значительно сократить пространство поиска за счет введения в процесс поиска предварительных знаний. Обширные экспериментальные результаты, выполненные авторами работы [23], показывают, что результаты использования метод GPT-NAS значительно превосходят семь нейронных архитектур, разработанных вручную, и тринадцать архитектур, предоставляемых конкурирующими методами NAS. Кроме того, эти эксперименты также показывают, что предложенный алгоритм повышает производительность точной настройки нейронных архитектур примерно на 12 процентов по сравнению с алгоритмами, не использующими GPT, что дополнительно демонстрирует его эффективность при поиске нейронных архитектур.

GPT-NAS разработан с целью оптимизации поиска оптимальной архитектуры ИНС и пространства поиска. Архитектура нейронной сети состоит из операций: сверток, подвыборки, блоков (состоящих из нескольких слоев). Через OP_m обозначим множество всех операций, где m – ко-

личество операций. Перестановкой операций можно получить пространство поиска архитектур: $X = \{x_i | i = \overline{1, n}\}$, где через n обозначено количество архитектур. При этом, значение n , как правило, огромно. Например, при разработке архитектуры с 20 слоями пространство X состоит из m^{20} архитектур. Целью NAS является поиск оптимальной сети x^* , $x^* = f(X)$, в этом пространстве при использовании стратегии поиска f .

Для оптимизации поиска оптимальной сети стратегия поиска может включать предварительные знания, объем которых обозначается через G . Таким образом, задача формализуется так:

$$x^* = f(X, G).$$

Для эффективного включения G в стратегию поиска авторы работы [23] предложили использовать GPT, которая демонстрирует высокие результаты в решении задачи предсказания текста. Целью обучения модели GPT на крупномасштабном массиве ИНС является формирование у нее «общего понимания» нейросетевых архитектур и перенос его на конкретные задачи. Вся процедура поиска оптимальной архитектуры разделяется на 3 этапа: кодирование архитектуры ИНС, предварительное обучение и тонкая настройка модели GPT, поиск архитектуры ИНС с использованием обученной модели GPT.

Этап 1. Кодирование архитектуры ИНС. На этом этапе архитектура кодируется последовательностью символов, где каждый символ соответствует определенной операции (сверточный слой, полносвязный слой и т.д.) Эффективное кодирование архитектур нейронных сетей может помочь модели GPT усвоить фундаментальные законы архитектурной композиции. Поэтому необходимо разработать общую стратегию кодирования, подходящую для популярных архитектур ИНС. Основываясь на архитектурных закономерностях сверточных нейронных сетей (convolutional neural networks, CNN), авторы работы [23] разделяют слои на 4 категории: сверточные, подвыборки, полносвязные и другие. Код сверточного слоя должен включать его характеристики: размеры входа и выхода слоя, высоту и ширину ядра, величину шага (stride), ширину и способ обрамления (padding), величину расширения (dilation), количество групп каналов для обработки разными ядрами, возможность использования смещения (bias). Для кода подвыборки из ранее перечисленных характеристик следует исключить количество групп, но указать тип подвыборки (максимум или среднее). Полносвязный слой имеет только две характеристики: размеры входа и размеры выхода. К другим слоям относятся, например, слои активации, слои пакетной нормализации (batch normalization) и другие, для которых необходимо указать имя, размеры входа и выхода, а также значения используемых параметров. Таким образом, структура CNN кодируется текстовой строкой.

Этап 2. Предварительное обучение и тонкая настройка модели GPT. Предварительное обучение GPT выполняется на большом массиве данных, как правило, без учителя для достижения максимального правдоподобия. Тонкая настройка осуществляется с учителем на меньшем объеме данных. Но, учитывая, что на обоих этапах GPT-NAS обучается на архитектурах нейронных сетей, то на каждом этапе используется обучение с учителем. Целью обучения GPT-NAS является предсказание структурной информации о следующем слое по известной информации о предыдущих слоях. Таким образом, минимизируется следующая функция потерь:

$$\begin{cases} \mathfrak{F} = \sum_t^T \mathcal{L}(\hat{l}_t, l_t | l_{<t}, C), \\ C = f(\theta, S), \end{cases}$$

где через C обозначена архитектура ИНС, θ – набор значений ее параметров, S – соответствующая структура слоев. В формуле для целевой функции F через T обозначено количество слоев в C , L обозначает функцию потерь, \hat{l}_t и l_t – обозначают, соответственно, предсказанную и истинную структуру слоя с номером t , и $l_{<t} = (l_{t-k}, \dots, l_{t-1})$, k – размер окна данных.

Этап 3. Поиск архитектуры ИНС. Этот этап выполняется после предварительного обучения и тонкой настройки модели GPT. Этап состоит из двух шагов. Сначала архитектура отбирается, обучается и оценивается с использованием генетического алгоритма в качестве стратегии поиска. На втором шаге выбранные архитектуры реорганизуются с использованием GPT. На первом шаге каждой итерации инициализируются несколько архитектур. При этом инициализируемые архитектуры могут иметь разную длину. Затем оценивается эффективность каждой из этих архитектур. После этого выполняются эволюционные операции над получаемыми архитектурами, включая скрещивание, мутацию и отбор. Результатом первого шага является сеть с наилучшей эффективностью, полученная после выполнения заданного числа итераций. Второй шаг итерации. Соседние слои нейронной сети группируются в блоки. Блочная структура используется для ускорения процедуры селекции оптимальной архитектуры. Реконструкция сети на втором шаге итерации выполняется путем замены некоторого блока. Пусть, например, ИНС перед выполнением этого шага представлена последовательностью блоков в порядке их следования от входа сети к ее выходу: B_1, B_2, \dots, B_m . На этом шаге случайно выбирается некоторый слой оптимизируемой нейронной сети. Пусть этот слой содержится в блоке B_t . Из архитектуры удаляется этот блок B_t . Далее, с использованием модели GPT предсказывается слой l наиболее подходящий для последовательности слоев, образуемой блоками B_1, B_2, \dots, B_{t-1} . Предварительно авторами работы

[23] сформирован перечень из 12 стандартных блоков в привязке к стандартным архитектурам ИНС (таблица 1), которые они используют для замены блока B_t .

Таблица 1. Соответствие стандартных архитектур и блоков, содержащихся в них

| Номер стандартной архитектуры ИНС | Стандартная архитектура ИНС | Наименование блока |
|-----------------------------------|-----------------------------|---|
| 1 | EfficientNet | ConvNormActivation SqueezeExcitation |
| 2 | GoogleNet | Inception Avgpool |
| 3 | RegNet | ResBottleneckBlock Stem |
| 4 | ResNet | Bottleneck BasicBlock |
| 5 | ResNext | Bottleneck |
| 6 | ShuffleNet | InvertedResidual |
| 7 | Wide-ResNet | Bottleneck |
| 8 | other | Maxpool BatchNormal Relu Conv |

С использованием предварительно обученной полносвязной ИНС классификации для слоя l выбирается наиболее подходящий блок B'_t , который вставляется на место B_t в последовательности B_1, B_2, \dots, B_m . Многократное повторение рассмотренной двухшаговой оптимизации позволяет достичь требуемого результата.

Разработанная авторами [23] система GPT-NAS сравнивалась с 20 современными конкурентами, среди которых были как алгоритмы, разработанные вручную, так и полученные с использованием NAS, на трех популярных наборах данных: CIFAR10, CIFAR100 и ImageNet-1K. Анализ экспериментальных результатов показал, что GPT-NAS достигает самых современных результатов, чем обосновывается положительное влияние GPT для разработки алгоритма NAS.

Вот пример архитектуры нейронной сети для бинарной классификации (лягушки или собаки) с использованием библиотеки Keras:

```
python
from keras.models import Sequential
from keras.layers import Dense, Flatten

# Создание модели
model = Sequential()

# Добавление слоев
model.add(Flatten(input_shape=(32, 32))) # Преобразование входных изображений в
одномерный вектор
model.add(Dense(64, activation='relu')) # Полносвязный слой с 64 нейронами и функцией
активации ReLU
model.add(Dense(1, activation='sigmoid')) # Выходной слой с 1 нейроном и функцией
активации sigmoid

# Компиляция модели
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

В данном примере используется последовательная модель Sequential, которая позволяет добавлять слои последовательно. Сначала добавляется слой Flatten, который преобразует входные изображения размером 32x32 в одномерный вектор. Затем добавляется полносвязный слой Dense с 64 нейронами и функцией активации ReLU. На выходе используется выходной слой Dense с 1 нейроном и функцией активации sigmoid, которая применяется для бинарной классификации.

Функция ошибки установлена как бинарная перекрестная энтропия (binary_crossentropy), а оптимизатор - Adam (adam). Метрика точности (accuracy) также указана для оценки производительности модели.

Рис. 1. Ответ чат-бота на запрос 1.

5. Эксперименты в Telegram с ChatGPT & Midjourney / AI bot

Запрос 1. Разработай в библиотеке Keras языка Python архитектуру нейронной сети бинарной классификации (лягушки или собаки). На вход сети подаются изображения размера 32 на 32. Сеть имеет 1 выход. Функция ошибки - бинарная перекрестная энтропия.

Ответ чат-бота представлен на рисунке 1.

Запрос 2 (уточнение). Добавь обучение на классах "лягушки" и "собаки" этой сети на базе CIFAR10.

Ответ чат-бота представлен на рисунках 2 и 3.

Результат проверки. Чат-ботом создана сеть бинарной классификации. В один из классов выделены лягушки, в другой объединены все остальные 9 классов базы CIFAR10, включая класс собак. Исправлять этот недочет не будем, а обучим модель и протестируем полученный

1. Загрузите данные CIFAR10 и разделите их на обучающую и тестовую выборки:

```
Python
from keras.datasets import cifar10
# Загрузка данных CIFAR-10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
# Нормализация данных
x_train = x_train / 255.0
x_test = x_test / 255.0
# Преобразование меток в бинарные классы (0 - лягушка, 1 - собака)
y_train = (y_train == 6).astype(int)
y_test = (y_test == 6).astype(int)
```

2. Создайте модель и добавьте слои:

```
python
from keras.models import Sequential
from keras.layers import Dense, Flatten
# Создание модели
model = Sequential()
# Добавление слоев
model.add(Flatten(input_shape=(32, 32, 3))) # Преобразование входных изображений в
одномерный вектор
model.add(Dense(64, activation='relu')) # Полносвязный слой с 64 нейронами и функцией
активации ReLU
model.add(Dense(1, activation='sigmoid')) # Выходной слой с 1 нейроном и функцией
активации sigmoid
```

Рис. 2. Ответ чат-бота на запрос 2. Часть 1.

классификатор. Точность классификации на тестовом массиве после отработки скрипта составила 0.91.

Теперь проверим, как чат-бот отреагирует на более полный запрос, который включает запрос 1 и запрос 2.

Запрос 3. Напиши мне в библиотеке Keras языка Python программу разработки бинарного классификатора (лягушки или собаки) с использованием нейронной сети. На вход сети подаются изображения размера 32 на 32. Сеть имеет 1 выход. Функция ошибки - бинарная перекрестная энтропия. В программу включи возможность обучения и тестирования на классах "лягушки" и "собаки" этой сети на базе zifar10.

Ответ чат-бота представлен на рисунках 4 и 5.

Результат проверки. Чат-бот предложил существенно более сложную архитектуру нейронной сети, сохранив прежнее деление на 2 класса. Точ-

3. Компилируйте модель с функцией ошибки `binary_crossentropy`, оптимизатором `adam` и метрикой `accuracy`:

```
python
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

4. Обучите модель на обучающей выборке:

```
python
model.fit(x_train, y_train, batch_size=32, epochs=10, validation_data=(x_test, y_test))
```

5. Оцените производительность модели на тестовой выборке:

```
python
loss, accuracy = model.evaluate(x_test, y_test)
print('Test loss:', loss)
print('Test accuracy:', accuracy)
```

Вы можете настроить параметры обучения, такие как размер пакета (`batch_size`) и количество эпох (`epochs`), в зависимости от ваших потребностей.

Вы можете дальше настраивать эту архитектуру и обучать модель на своих данных.

Рис. 3. Ответ чат-бота на запрос 2. Часть 2.

```
python
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
# Загрузка данных CIFAR-10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
# Преобразование меток в бинарные значения (0 для лягушек, 1 для собак)
y_train = (y_train == 6).astype(int)
y_test = (y_test == 6).astype(int)
# Нормализация данных
x_train = x_train / 255.0
x_test = x_test / 255.0
```

Рис. 4. Ответ чат-бота на запрос 3. Часть 1.

ность классификации новой сети на тестовой базе CIFAR10 оказалась существенно выше полученной в первом эксперименте и составила 0.95.

Попробуем добиться от сети корректного выделения изображений собак и лягушек.

Запрос 4. Напиши мне в библиотеке Keras языка Python программу разработки бинарного классификатора (лягушки или собаки) с исполь-

```

# Нормализация данных
x_train = x_train / 255.0
x_test = x_test / 255.0
# Создание модели нейронной сети
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Компиляция модели
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Обучение модели
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
# Оценка производительности модели на тестовых данных
_, accuracy = model.evaluate(x_test, y_test)
print('Accuracy: %.2f' % (accuracy * 100))

```

Обратите внимание, что вам потребуется установить TensorFlow и Keras для запуска этого кода. Также обратите внимание, что это простой пример, и вам может потребоваться настроить модель и параметры обучения в соответствии с вашими потребностями и требованиями.

Рис. 5. Ответ чат-бота на запрос 3. Часть 2.

зованием нейронной сети. На вход сети подаются изображения размера 32 на 32 из базы CIFAR10. В обучающей и тестовой базах оставь только изображения лягушек и собак, остальные изображения отсей. Сеть имеет 1 выход. Функция ошибки - бинарная перекрестная энтропия. В программу включи возможность обучения и тестирования на классах "лягушки" и "собаки" этой сети на базе CIFAR10.

Ответ чат-бота представлен на рисунках 6 и 7.

Результат проверки. Чат-бот попытался выделить изображения лягушек и собак из обучающей и тестовой баз. При этом полученный код можно было использовать после некоторой доработки. Чат-ботом в этом эксперименте предложена упрощенная модель нейронной сети, ее точность на тестовом наборе составила всего 0.76.

Попытаемся повысить точность классификации. Для этого используем архитектуру, предложенную в ответе на запрос 3. В результате обу-

Ниже представлен пример программы на языке Python с использованием библиотеки Keras для разработки бинарного классификатора с использованием нейронной сети:

```
python
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.utils import to_categorical
# Загрузка данных CIFAR-10
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
# Отфильтровать изображения, оставив только классы "лягушка" и "собака"
classes = [6, 5] # Лягушка - класс 6, Собака - класс 5
train_filter = np.isin(y_train, classes)
test_filter = np.isin(y_test, classes)
X_train = X_train[train_filter]
y_train = y_train[train_filter]
X_test = X_test[test_filter]
y_test = y_test[test_filter]
# Преобразование меток в формат one-hot encoding
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

Рис. 6. Ответ чат-бота на запрос 4. Часть 1.

чения и тестирования получаем точность на тестовой базе 0.87. Можно ли увеличить этот показатель, используя чат-бот?

Запрос 5. У меня есть скрипт, позволяющий выполнять классификацию по двум классам (лягушки, собаки) на базе CIFAR10 с точностью 0.87. Привожу этот скрипт [здесь вставлен скрипт, содержащийся на рисунках 6 и 7]. Можно ли изменить архитектуру модели так, чтобы повысить точность классификации?

Ответ чат-бота. Да, можно изменить архитектуру модели для повышения точности классификации. Например, вы можете добавить дополнительные сверточные слои или полносвязные слои, увеличить количество нейронов в слоях, использовать различные функции активации, изменить параметры обучения, добавить регуляризацию или использовать другие методы оптимизации. Важно экспериментировать с различными архитектурами и параметрами модели, чтобы найти наилучшую комбинацию для данной задачи классификации.

```

# Преобразование изображений в формат, подходящий для обучения
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255
# Создание модели нейронной сети
model = Sequential()
model.add(Flatten(input_shape=(32, 32, 3)))
model.add(Dense(256, activation='relu'))
model.add(Dense(2, activation='softmax'))
# Компиляция модели
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Обучение модели
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=32)
# Тестирование модели
_, accuracy = model.evaluate(X_test, y_test)
print(f"Точность классификатора: {accuracy}")

```

В данном примере используется нейронная сеть с одним скрытым слоем, состоящим из 256 нейронов. Функция активации для скрытого слоя - ReLU, а для выходного слоя - softmax. В качестве функции ошибки используется бинарная перекрестная энтропия. Для обучения модели используется оптимизатор Adam.

Программа загружает данные CIFAR10 и фильтрует изображения, оставляя только классы "лягушка" и "собака". Затем данные преобразуются в формат one-hot encoding и нормализуются. Модель нейронной сети создается и компилируется с заданными параметрами. После этого модель обучается на обучающем наборе данных и тестируется на тестовом наборе данных. В конце выводится точность классификатора на тестовом наборе данных.

Рис. 7. Ответ чат-бота на запрос 4. Часть 2.

Наш комментарий. Бот понял вопрос буквально и ограничился лишь рекомендациями, скрипт дорабатывать не стал.

Попробуем нагрузить его задачей детектирования.

Запрос 6. Предложи программу на Python с использованием архитектуры детектора YOLOv3 из модуля gluoncv, предобученного на базе COCO. Предусмотри обработку детектором случайно выбранного из проверочной части базы COCO изображения. Результат обработки (изображение и рамки) представь с использованием модуля matplotlib.

Ответ чат-бота представлен на рисунке 8.

Наша адаптация предложенного чат-ботом решения. Как и следовало ожидать, в этом случае пришлось выполнить существенную доработку кода. Его работоспособный вариант, использующий загруженные из COCO данные, после нашей доработки представлен на рисунках 9 и 10.

Конечно! Вот пример программы на Python, которая использует архитектуру детектора YOLOv3 из модуля gluoncv и предобученную на базе COCO модель. Она обрабатывает случайно выбранное изображение из проверочной части базы COCO и выводит результаты с использованием модуля matplotlib:

```
python
import matplotlib.pyplot as plt
from gluoncv import model_zoo, data, utils
# Загрузка предобученной модели YOLOv3
net = model_zoo.get_model('yolo3_darknet53_coco', pretrained=True)
# Загрузка данных из проверочной части базы COCO
test_dataset = data.COCODetection(splits=['instances_val2017'])
image, label = test_dataset[0]
# Обработка изображения с помощью модели YOLOv3
x, _ = data.transforms.presets.yolo.transform_test(image)
class_IDs, scores, bounding_boxes = net(x)
# Вывод изображения и рамок с помощью модуля matplotlib
ax = utils.viz.plot_bbox(image, bounding_boxes[0], scores[0], class_IDs[0],
class_names=net.classes)
plt.show()
```

Эта программа загружает предобученную модель YOLOv3 (yolo3_darknet53_coco) из модуля gluoncv и случайно выбирает одно изображение из проверочной части базы COCO. Затем она обрабатывает это изображение с помощью модели YOLOv3 и выводит результаты в виде изображения и рамок с использованием модуля matplotlib.

Обратите внимание, что для запуска этой программы вам понадобится установить модули gluoncv и matplotlib. Вы можете установить их с помощью pip:

```
pip install gluoncv matplotlib
```

Также убедитесь, что у вас есть доступ к базе COCO для загрузки данных.

Рис. 8. Ответ чат-бота на запрос 6.

Как видно, при доработке не решалась задача внесения минимальных исправлений для обеспечения работоспособности кода, а в код были внесены еще некоторые дополнения для анализа результатов работы сети.

Результат выполнения доработанного кода приведен на рисунке 11.

Выводы. Предложенные чат-ботом модули могут быть использованы как первичные грубые приближения для решения поставленной задачи, нуждающиеся в дальнейшей доработке экспертом. Ответы чат-бота не учитывают специфики версионности программных модулей, имеющегося у пользователя, могут ошибаться или неточно воспринимать запрос.

```

import matplotlib.pyplot as plt
from gluoncv import model_zoo, data, utils
import mxnet as mx
import time
import random
from pycocotools.coco import COCO
from pathlib import Path
from skimage import io
def test_model(model,img):
    # preprocessing
    mx_img = mx.nd.array(img)
    x,original_img = data.transforms.presets.rcnn.transform_test(mx_img)
    # time measurement
    start = time.time()
    box_ids, scores, bboxes = model(x)
    t = time.time() - start
    # image with model detections boxes, scores and ids
    ax = utils.viz.plot_bbox(original_img, bboxes[0], scores[0], box_ids[0],
class_names=model.classes)
    return ax,t

```

Рис. 9. Доработанный код. Часть 1.

```

yolo = model_zoo.get_model('yolo3_darknet53_coco', pretrained=True)
DATA_PATH = "D:/COCO/coco2017/val2017/"
annFile = Path("D:/COCO/coco2017/annotations/instances_val2017.json")
coco = COCO(annFile)
imgIds = coco.getImgIds()
img,anns = get_rand_img()
ax3,t3 = test_model(yolo,img)
ax3.set_title(f"YOLOv3 \n time taken {round(t3,4)}")
plt.show()

```

Рис. 10. Доработанный код. Часть 2.

6. Заключение

В этой работе выполнен обзор существующих подходов для решения задачи автоматизированного поиска архитектур нейронных сетей, удовлетворяющих запросу пользователя. Последние наиболее эффективные и перспективные решения связаны с применением больших языковых моделей, для обучения которых использовались значительные массивы данных. При этом применение больших языковых моделей для генерирования адекватных архитектур требует существенной алгоритмической доработки с применением, например, принципов генетического отбора и стратегии обучения с подкреплением.

Помимо обзора, выполнены некоторые эксперименты с ChatGPT & Midjourney доступного в Telegram для выяснения возможности использо-

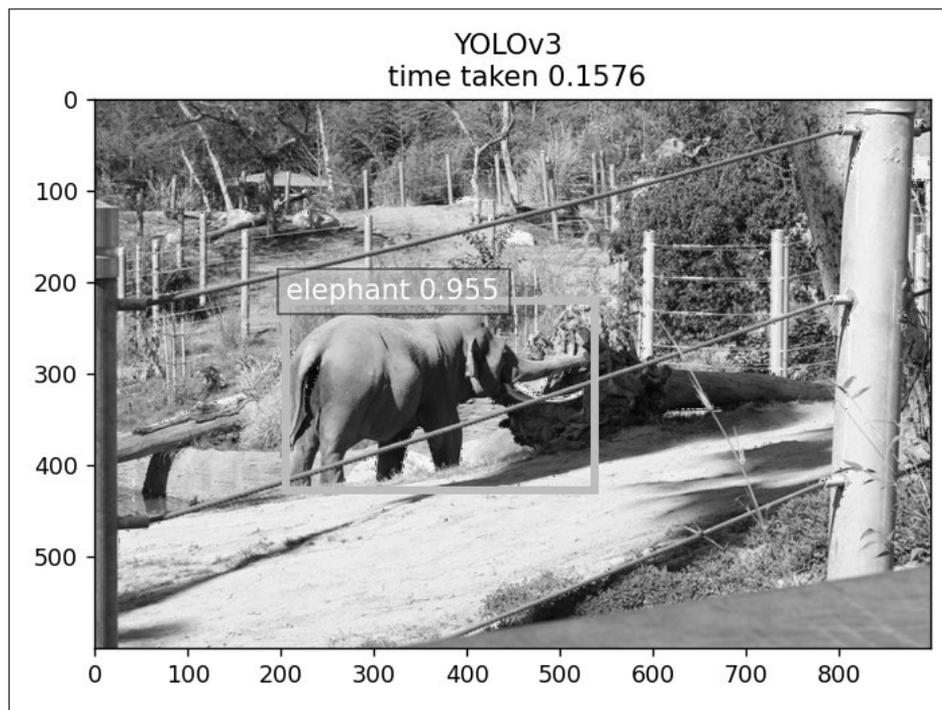


Рис. 11. Результат работы доработанного кода.

вания этого инструмента при разработке ИНС классификации и детектирования. Как показали результаты экспериментов, пока что от этого инструмента не следует ожидать окончательного точного решения, но его предложения можно использовать в качестве некоторой основы для последующей доработки.

Исследование выполнено при поддержке Министерства науки и высшего образования Российской Федерации (грант № 075-15-2020-801).

Список литературы

- [1] Elsken T., Metzen J. H., Hutter F., “Neural architecture search: A survey”, *The Journal of Machine Learning Research*, **20**:1 (2019), 1997–2017.
- [2] Zoph B., Le Q.V., “Neural architecture search with reinforcement learning”, *International Conference on Learning Representations*, 2017.
- [3] Bergstra J., Yamins D., Cox D. D., “Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures”, *ICML*, 2013.

- [4] Mendoza H., Klein A., Feurer M., Springenberg J., Hutter F., “Towards Automatically-Tuned Neural Networks”, *International Conference on Machine Learning, AutoML Workshop*, 2016.
- [5] Kandasamy K., Neiswanger W., Schneider J. , Poczos B., Xing E. P., “Neural architecture search with Bayesian optimization and optimal transport”, *ArXiv: 1802.07191*, 2018.
- [6] Jaafra Y., Laurent J. L., Deruyver A., Naceur M. S., “Reinforcement learning for neural architecture search: A review”, *Image and Vision Computing*, **89** (2019), 57–66.
- [7] Liu Y., Sun Y., Xue B., Zhang M., Yen G. G., Tan K. C., “A survey on evolutionary neural architecture search”, *IEEE transactions on neural networks and learning systems*, 2021.
- [8] M. Wistuba, “Finding competitive network architectures within a day using UCT”, *ArXiv: 1712.07420*, 2017.
- [9] Letunovskiy A., Korviakov V., Polovnikov V., Kargapolitseva A., Mazurenko I., Xiong Y., “ISyNet: Convolutional Neural Networks design for AI accelerator”, *ArXiv: 2109.01932*, 2021.
- [10] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I., “Attention is all you need”, *ArXiv: 1706.03762v7*, 2018.
- [11] Radford A., Narasimhan K., Salimans T., Sutskever I., “Improving language understanding by generative pre-training, <https://gwern.net/doc/www/s3-us-west-2.amazonaws.com/d73fdc5ffa8627bce44dcda2fc012da638ffb158.pdf>”, 2018.
- [12] Radford A., Wu J., Child R., Luan D., Amodei D., Sutskever I., “Language models are unsupervised multitask learners”, *OpenAI blog*, **1:8** (2019).
- [13] Brown T., Mann B., Ryder N., Subbiah M., Kaplan J. D., Dhariwal P., Neelakantan A., Shyam P., Sastry G., Askell A. et al., “Language models are few-shot learners”, *Advances in neural information processing systems*, **33** (2020), 1877–1901.
- [14] Chen M., Tworek J., Jun H., Yuan Q., Pinto H. P. d. O., Kaplan J., Edwards H., Burda Y., Joseph N., Brockman G. et al., “Evaluating large language models trained on code”, *ArXiv: 2107.03374*, 2021.

- [15] Todd G., Earle S., Nasir M. U., Green M. C., Togelius J., “Level generation through large language models”, *Proceedings of the 18th International Conference on the Foundations of Digital Games*, 2023, 1–8.
- [16] Chen A., Dohan D. M., and So D. R., “Evoprompting: Language models for code-level neural architecture search”, *ArXiv: 2302.14838*, 2023.
- [17] Greydanus S., “Scaling down deep learning”, *ArXiv: 2011.14439*, 2020.
- [18] Veličković P., Badia A. P., Budden D., Pascanu R., Banino A., Dashevskiy M., Hadsell R., Blundell C., “The CLRS algorithmic reasoning benchmark”, *International Conference on Machine Learning. PMLR*, 2022, 84–102.
- [19] Caiyang Yu, Xianggen Liu, Wentao Feng, Chenwei Tang, Jiancheng Lv., “LLMatic: Neural Architecture Search via Large Language Models and Quality-Diversity Optimization”, *ArXiv: 2306.01102v1*, 2023.
- [20] Pugh J. K., Soros L. B., Stanley K. O., “Quality diversity: A new frontier for evolutionary computation”, *Frontiers in Robotics and AI*, 2016.
- [21] Mouret J.-B., Clune J., “Illuminating search spaces by mapping elites”, *ArXiv: 1504.04909*, 2015.
- [22] Stanley K. O., Miikkulainen R., “Evolving neural networks through augmenting topologies”, *Evolutionary computation*, **10**:2 (2002), 99–127.
- [23] Caiyang Yu, Xianggen Liu, Wentao Feng, Chenwei Tang, “GPT-NAS: evolutionary neural architecture search with generative pre-trained model”, *ArXiv: 2305.05351v2*, 2023.

Automation of artificial neural network architectures search
Chasovskikh A.A., Polovnikov V.S., Khusaenov A.A., Bokov G.V., Konovalov A.Yu, Drobyshev A.S., Biryukova V.A.

A review of works on methods for automating generation of artificial neural network architectures is presented. The most detailed material is presented on the use of large language models for this. For the experiments, we used ChatGPT & Midjourney, which is available in Telegram Messenger.

Keywords: artificial neural network, large language models, neural architecture search, generative pre-training transformer (GPT).

References

- [1] Elsken T., Metzen J. H., Hutter F., “Neural architecture search: A survey”, *The Journal of Machine Learning Research*, **20**:1 (2019), 1997–2017.
- [2] Zoph B., Le Q.V., “Neural architecture search with reinforcement learning”, *International Conference on Learning Representations*, 2017.
- [3] Bergstra J., Yamins D., Cox D. D., “Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures”, *ICML*, 2013.
- [4] Mendoza H., Klein A., Feurer M., Springenberg J., Hutter F., “Towards Automatically-Tuned Neural Networks”, *International Conference on Machine Learning, AutoML Workshop*, 2016.
- [5] Kandasamy K., Neiswanger W., Schneider J. , Poczos B., Xing E. P., “Neural architecture search with Bayesian optimization and optimal transport”, *ArXiv: 1802.07191*, 2018.
- [6] Jaafra Y., Laurent J. L., Deruyver A., Naceur M. S., “Reinforcement learning for neural architecture search: A review”, *Image and Vision Computing*, **89** (2019), 57–66.
- [7] Liu Y., Sun Y., Xue B., Zhang M., Yen G. G., Tan K. C., “A survey on evolutionary neural architecture search”, *IEEE transactions on neural networks and learning systems*, 2021.
- [8] M. Wistuba, “Finding competitive network architectures within a day using UCT”, *ArXiv: 1712.07420*, 2017.
- [9] Letunovskiy A., Korviakov V., Polovnikov V., Kargapolitseva A., Mazurenko I., Xiong Y., “ISyNet: Convolutional Neural Networks design for AI accelerator”, *ArXiv: 2109.01932*, 2021.
- [10] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I., “Attention is all you need”, *ArXiv: 1706.03762v7*, 2018.
- [11] Radford A., Narasimhan K., Salimans T., Sutskever I., “Improving language understanding by generative pre-training”, <https://gwern.net/doc/www/s3-us-west-2.amazonaws.com/d73fdc5ffa8627bce44dcda2fc012da638ffb158.pdf>, 2018.

- [12] Radford A., Wu J., Child R., Luan D., Amodei D., Sutskever I., “Language models are unsupervised multitask learners”, *OpenAI blog*, **1**:8 (2019).
- [13] Brown T., Mann B., Ryder N., Subbiah M., Kaplan J. D., Dhariwal P., Neelakantan A., Shyam P., Sastry G., Askell A. et al., “Language models are few-shot learners”, *Advances in neural information processing systems*, **33** (2020), 1877–1901.
- [14] Chen M., Tworek J., Jun H., Yuan Q., Pinto H. P. d. O., Kaplan J., Edwards H., Burda Y., Joseph N., Brockman G. et al., “Evaluating large language models trained on code”, *ArXiv: 2107.03374*, 2021.
- [15] Todd G., Earle S., Nasir M. U., Green M. C., Togelius J., “Level generation through large language models”, *Proceedings of the 18th International Conference on the Foundations of Digital Games*, 2023, 1–8.
- [16] Chen A., Dohan D. M., and So D. R., “Evoprompting: Language models for code-level neural architecture search”, *ArXiv: 2302.14838*, 2023.
- [17] Greydanus S., “Scaling down deep learning”, *ArXiv: 2011.14439*, 2020.
- [18] Veličković P., Badia A. P., Budden D., Pascanu R., Banino A., Dashevskiy M., Hadsell R., Blundell C., “The CLRS algorithmic reasoning benchmark”, *International Conference on Machine Learning. PMLR*, 2022, 84–102.
- [19] Caiyang Yu, Xianggen Liu, Wentao Feng, Chenwei Tang, Jiancheng Lv., “LLMatic: Neural Architecture Search via Large Language Models and Quality-Diversity Optimization”, *ArXiv: 2306.01102v1*, 2023.
- [20] Pugh J. K., Soros L. B., Stanley K. O., “Quality diversity: A new frontier for evolutionary computation”, *Frontiers in Robotics and AI*, 2016.
- [21] Mouret J.-B., Clune J., “Illuminating search spaces by mapping elites”, *ArXiv: 1504.04909*, 2015.
- [22] Stanley K. O., Miikkulainen R., “Evolving neural networks through augmenting topologies”, *Evolutionary computation*, **10**:2 (2002), 99–127.
- [23] Caiyang Yu, Xianggen Liu, Wentao Feng, Chenwei Tang, “GPT-NAS: evolutionary neural architecture search with generative pre-trained model”, *ArXiv: 2305.05351v2*, 2023.