Оптимизация архитектуры нейросети с контролем эксплуатационных характеристик на целевом устройстве

Фирсов Сергей, Бахтеев Олег
Интеллектуальный анализ данных
МФТИ

13 июня 2025 г.

Аннотация

В работе рассматривается задача выбора архитектуры модели глубокого обучения (NAS) с контролем характеристик на заранее неизвестном целевом устройстве. Предлагается метод, сочетающий точность модели и эффективность аппаратного исполнения. Рассматриваемый подход основан на дифференцируемом поиске архитектур (DARTS), в котором оптимизируются структурные параметры градиентными методами путём релаксации дискретной задачи к непрерывной.

В отличие от стандартных подходов, в рассматриваемом методе структурные параметры сети задаются как функции векторного параметра сложности. Компоненты вектора интерпретируются как штрафы за соответствующие операции, что позволяет более точно и детально контролировать структуру модели. Метод использует Gumbel-Softmax распределение в сочетании с гиперсетями для обучения параметров модели, что позволяет одновременно оптимизировать несколько архитектур на различных уровнях сложности. Процедура оптимизации за один этап обучения создает семейство архитектур, адаптированных к различным вычислительным ресурсам, что значительно сокращает время поиска и ресурсы. Проведённые эксперименты на датасетах CIFAR-10 и Fashion-MNIST демонстрируют, что предложенный подход позволяет эффективно искать архитектуры, обеспечивающие компромисс между точностью и задержкой на целевом оборудовании.

1 Введение

Выбор архитектуры для моделей глубокого обучения является важной задачей, которая напрямую влияет на эффективность и качество решения. Дифференцируемые методы NAS (Differentiable Architecture Search, DARTS) позволяют исследовать архитектурное пространство нейросетей с помощью градиентного спуска, сводя дискретный выбор операций к оптимизации логитов, определяющих архитектуру модели [7, 3]. Они обеспечивают высокую эффективность поиска, но предлагают двухуровневую оптимизацию и не позволяют контролировать аппаратные ограничения.

Среди методов, ориентированных на аппаратное обеспечение, выделим FBNet [9] и ProxylessNAS[2]. Оба метода напрямую интегрируют в функцию потерь аппаратно-зависимый критерий “время отклика” для адаптации архитектуры к целевому устройству. В ProxylessNAS архитектурные параметры переводятся в бинарные маски, позволяющие на каждом шаге выбирать и обновлять всего два случайных пути в суперсети — этот приём позволяет оптимизировать latency без прокси-задач с минимальным расходом памяти. В FBNet же строится полная суперсеть всех кандидатов, а штраф за задержку рассчитывается по заранее измеренной таблице, что обеспечивает точный учёт реального времени выполнения.

Ряд исследований концентрируется на one-shot подходах [1, 4], где единая суперсеть обучается одновременно для множества архитектурных вариантов, а затем из неё отбираются нужные под заданные ограничения. Эти методы демонстрируют гибкость производя семейство моделей после обучения и позволяют быстро адаптироваться к разным аппаратным задачам.

Отдельно отметим подходы с использованием гиперсетей для создания архитектур. В MODNAS [6] и DARTS-CC [10] обучаются дополнительные модели, которые по введённому параметру сложности генерируют архитектурные параметры, упрощая управление дилеммой точности и аппаратных метрик. Основное преимущество таких методов — получение целого Парето-фронта моделей в рамках единого процесса обучения, что позволяет быстро выбирать оптимальные архитектуры под разные ограничения без повторного NAS.

Основываясь на этих идеях, представляется подход использующий гиперсети для обучения с контролем характеристик на целевом устройстве. В отличие от большинства методов, предлагается возможность генерировать несколько архитектур в рамках одного одноуровневого процесса оптимизации [10, 6]. Описываемый подход внедряет оптимизацию с учетом задержки на устройстве[9, 2] и более гибкий выбор модели за счёт использования векторного параметра сложности вместо скалярного. Метод схематично проиллюстрирован на рисунке 1.

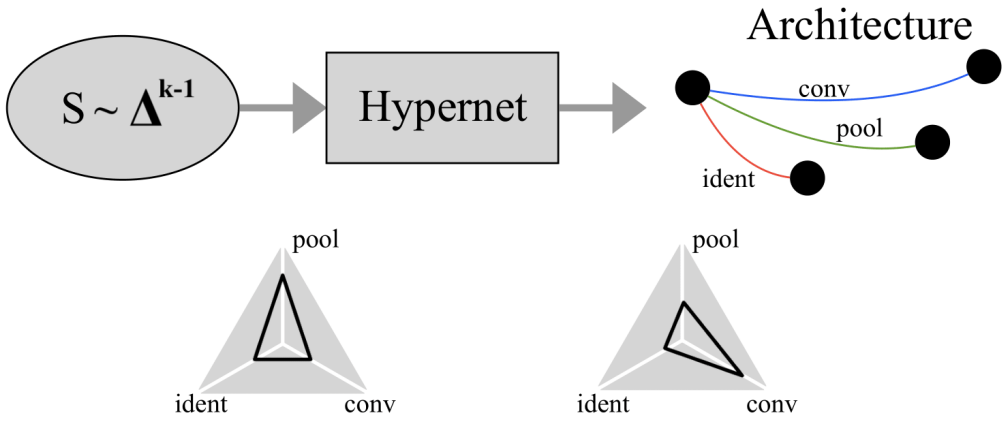


Рис. 1: Иллюстрация предлагаемого метода. Компоненты векторного параметра сложности $\boldsymbol{s}$ суть коэффициенты регуляризации по соответствующим операциям, их изменение даёт возможность получать различные архитектуры соответствующие желаемым требованиям к операциям.

В экспериментальной части приводится подтверждение работоспособности предлагаемого метода при решении задачи классификации на датасетах Fashion-MNIST и CIFAR-10. Результаты подтверждают получение широкого семейства моделей и возможность гибкого выбора с контролем желаемых характеристик на целевом оборудовании.

2 Постановка задачи

Ставится задача многоклассовой классификации. Пусть задана выборка $\mathfrak{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{\mathcal{N}}$, $\boldsymbol{x_i} \in \mathbf{X}$, $y_i \in \mathbf{Y}$, где $\mathcal{N}$ — количество примеров, $\mathbf{X}$ — множество признаков, а $\mathbf{Y}$ — множество меток. Пусть A — пространство поиска решений, подробное описание которого дано ниже. Предлагаемый метод ищет лучшие архитектуры из этого пространства оптимизируя как и потери задачи классификации, так и аппаратные требования на целевом устройстве. За один процесс обучения он получает семейство моделей зависящее от параметра сложности, что позволяет гибко выбирать модели подстраиваясь под целевое оборудование.

2.1 Предлагаемый метод

Метод опирается на DARTS: нейросеть представляется в виде последовательности повторяющихся блоков, называемых *клетками*. Каждая клетка имеет одну и ту же *архитектуру*, но различающиеся внутренние параметры. Формально каждая клетка описывается ориентированным ациклическим графом — со множеством вершин $\mathbf{V} = \{0, 1, \ldots, N\}$ и рёбер $\mathbf{E} = \big\{(i, j) \mid 0 \leq i < j \leq N\big\}$. Положим, что вершины $i = 0$ и $i = 1$ — это два входа в клетку, являющиеся результатами двух предыдущих клеток, а вершины $i = 2, \ldots, N$ — будем называть *промежуточные узлы*. На каждом ребре $(i, j) \in E$ выбирается *операция* из набора

$$\mathcal{O} = \big\{\mathbf{o}^{(1)}, \mathbf{o}^{(2)}, \ldots, \mathbf{o}^{(k)}\big\},$$

где $k = |\mathcal{O}|$ — число доступных операций. Для каждой операции обучаются соответствующие *параметры* $\boldsymbol{w}^{(i)} \in \mathbb{R}_i^p$, где размерность p_i зависит от типа операции.

Обозначим через $\boldsymbol{x}^{(i)}$ вектор признаков в вершине i. Тогда выходы промежуточных вершин j (для $2 \leq j \leq N$) вычисляются как сумма по всем входящим рёбрам:

$$\boldsymbol{x}^{(j)} \;=\; \sum_{\substack{(i,j)\in E\\ i<j}} \mathbf{g}^{(i,j)}\big(\boldsymbol{x}^{(i)}\big).$$

Здесь $\mathbf{g}^{(i,j)}(\cdot)$ — *смешанная операция*. Данное выражение представляет *релаксацию дискретной задачи* выбора архитектуры модели глубокой сети, в которой предполагается выбор строго одной операции из смешанных, на *непрерывную*, в которой возможно обучение сразу по нескольким вариантам. За счёт хранения на ребре (i,j) *логитов*

$$\boldsymbol{\alpha}^{(i,j)} = \big(\alpha_1^{(i,j)}, \ldots, \alpha_k^{(i,j)}\big) \in \mathbb{R}^k,$$

вводим операцию взвешенного объединения всех рёбер между вершинами i, j

$$\widehat{\mathbf{g}}^{(i,j)}(\boldsymbol{x}) \;=\; \sum_{m=1}^{k} \gamma_m^{(i,j)}\, \mathbf{o}^{(m)}(\boldsymbol{x}),$$

где *вектор весов* $\boldsymbol{\gamma}^{(i,j)}$ предлагается получать из Gumbel-Softmax распределения

$$\boldsymbol{\gamma}^{(i,j)} = \big(\gamma_1^{(i,j)}, \ldots, \gamma_k^{(i,j)}\big) \in \Delta^{\,k-1} \quad , \quad \boldsymbol{\gamma}^{(i,j)} \;\sim\; \mathrm{GumbelSoftmax}\big(\boldsymbol{\alpha}^{(i,j)},\, t\big),$$

где t — температура распределения, $t \to 0$.

Введя такую смешанную операцию, задача выбора дискретной архитектуры становится непрерывной, и веса $\boldsymbol{\gamma}^{(i,j)}$ оптимизируются вместе с параметрами $\boldsymbol{w}$ самих операций $\mathbf{o}^{(m)}$ с помощью градиентного спуска.

2.2 Гиперсеть для контроля сложности

Введём *вектор сложности*

$$\boldsymbol{s} \in \Delta^{k-1}, \quad \Delta^{k-1} \;=\; \Big\{\, \boldsymbol{s} \in \mathbb{R}^k \mid \sum_{m=1}^{k} s_m = 1,\; s_m \geq 0 \Big\}.$$

где $k = |\mathcal{O}|$. Этот вектор порождается из равномерного распределения на *симплексе* размерности $k-1$. Компоненты s_m интерпретируются как коэффициенты регуляризации для соответствующих примитивных операций $\mathbf{o}^{(m)}$: чем выше s_m, тем больший штраф за использование операции m.

Теперь введём $\boldsymbol{u_a}$ — *гиперсеть* с параметрами $\boldsymbol{a}$, которая на вход принимает вектор сложности $\boldsymbol{s}$ и выдаёт *логиты* $\boldsymbol{\alpha}^{(i,j)}$ для всех $(i,j) \in E$:

$$\boldsymbol{\alpha}^{(i,j)} \;=\; \boldsymbol{u}_{\boldsymbol{a}}^{(i,j)}\big(\boldsymbol{s}\big), \quad \boldsymbol{\alpha}^{(i,j)} \in \mathbb{R}^k.$$

Гиперсеть строит логиты $\boldsymbol{\alpha}^{(i,j)}$ для каждого ребра (i,j) как взвешенную комбинацию заранее выученных логитов опорных архитектур $\boldsymbol{\alpha}_{\mathrm{ref},r}^{(i,j)}$, полученных в процессе оптимизации гиперсети. По вектору сложности $\boldsymbol{s}$ вычисляются веса интерполяции

$$w_r(\boldsymbol{s}) = \frac{\exp\big(-\|\boldsymbol{s} - \boldsymbol{s}_{\mathrm{ref},r}\|^2\big)}{\sum\limits_{l=1}^{m} \exp\big(-\|\boldsymbol{s} - \boldsymbol{s}_{\mathrm{ref},l}\|^2\big)}, \quad r = 1, \ldots, m,$$

где $\boldsymbol{s}_{\mathrm{ref},r} \in \Delta^{k-1}$ — r-я опорная точка, а m — их общее число. Затем итоговые логиты для ребра (i,j) вычисляются как

$$\boldsymbol{\alpha}^{(i,j)}(\boldsymbol{s}) \;=\; \sum_{r=1}^{m} w_r(\boldsymbol{s})\; \boldsymbol{\alpha}_{\mathrm{ref},r}^{(i,j)}.$$

Таким образом гиперсеть реализует piecewise-интерполяцию в пространстве сложности, плавно меняя $\boldsymbol{\alpha}^{(i,j)}$ в зависимости от $\boldsymbol{s}$.

2.3 Регуляризация предпочтения операций в архитектуре модели

Как описано выше, предлагается использовать вектор сложности $\boldsymbol{s}$ как регуляризацию для примитивных операций. Определим функцию затрат Cost архитектуры γ на основе $\boldsymbol{s}$:

$$Cost\big(\gamma;\, \boldsymbol{s}\big) \;=\; \sum_{(i,j)\in E} \sum_{m=1}^{k} \gamma_m^{(i,j)}\; s_m.$$

Здесь $\gamma_m^{(i,j)}$ — вес m-й операции на ребре (i,j), а s_m — штраф для этой операции. Таким образом, чем больше s_m, тем менее невыгодно использовать примитивную операцию $\boldsymbol{o}_m$.

Объединив кросс-энтропийную функцию потерь $\mathcal{L}_{\text{task}}(\boldsymbol{w}, \gamma)$ и штраф Cost , получаем функцию потерь:

$$\mathcal{L}_{\text{full}}\big(\boldsymbol{w}, \boldsymbol{a}\big) \;=\; \mathbb{E}_{\boldsymbol{s}\sim U(\Delta^{\,k-1})}\Big[\, \mathcal{L}_{\text{task}}\big(\boldsymbol{w}, \gamma(\boldsymbol{s})\big) \;+\; \kappa\, Cost\big(\gamma(\boldsymbol{s});\, \boldsymbol{s}\big)\Big],$$

, где $\kappa > 0$ гиперпараметр, управляющий важностью штрафа.

Стохастическое порождение $\gamma \sim GS$ вносит высокую дисперсию градиентов, особенно при малой температуре t. Так как математическое ожидание GS-вектора совпадает с обычным Softmax $\hat{\gamma} = \mathbb{E}[\gamma] = Softmax(\alpha/t)$, можно безопасно подменить γ на $\hat{\gamma}$ в Cost части функции потерь, где важна стабильность. Формальную корректность такой подмены устанавливает следующая теорема.

Theorem 1 (Фирсов, 2025)**.** *Пусть для каждого набора весов $\boldsymbol{w} \in \mathbb{R}^n$ и любого входа $\boldsymbol{x} \in \mathcal{X}$ функция*

$$f(\boldsymbol{\gamma}) \;=\; \mathcal{L}_{\text{task}}\big(\boldsymbol{w}, \boldsymbol{\gamma}\big) \;+\; \kappa\, \langle \boldsymbol{\gamma}, \boldsymbol{s} \rangle, \qquad \boldsymbol{s} \in \Delta^{k-1},$$

непрерывна и ограничена на симплексе $\Delta^{k-1} = \big\{\boldsymbol{\gamma} \in \mathbb{R}^k \mid \sum_{i=1}^k \gamma_i = 1,\ \gamma_i \geq 0\big\}$. Положим

$$\boldsymbol{\gamma}(t) \sim \text{GumbelSoftmax}(\boldsymbol{\alpha}, t), \qquad \bar{\boldsymbol{\gamma}}(t) = \text{Softmax}\big(\boldsymbol{\alpha}/t\big) = \mathbb{E}\big[\boldsymbol{\gamma}(t)\big].$$

Тогда

$$\lim_{t\to 0^+} \mathbb{E}\big[f\big(\boldsymbol{\gamma}(t)\big)\big] \quad = \quad f\big(\bar{\boldsymbol{\gamma}}(0)\big).$$

Доказательство. Основано на доказательстве Th.1 из [10].

1. Свойство GS. По свойству Gumbel-Softmax[5, 8], при $t \to 0$ $\boldsymbol{\gamma}(t) \xrightarrow[\text{a.s.}]{t\to 0^+} \tilde{\boldsymbol{\gamma}}$, где $\tilde{\gamma}$ — one-hot вектор с единицей в $\arg\max_j \boldsymbol{\alpha}_j$.

2. Ограниченность f. f непрерывна на компакте Δ^{k-1}, значит ограничена — существует $M < \infty$: $|f(\boldsymbol{\gamma})| \leq M$ для всех γ.

3. Теорема Манна–Вальда. Из сходимости $\boldsymbol{\gamma}(t) \to \tilde{\boldsymbol{\gamma}}$ и непрерывности f следует $f\big(\boldsymbol{\gamma}(t)\big) \xrightarrow[\text{a.s.}]{} f(\tilde{\boldsymbol{\gamma}})$.

4. Доминированная сходимость. Поскольку f ограничена, применима теорема о доминированной сходимости:

$$\lim_{t\to 0^+} \mathbb{E}\, f\big(\boldsymbol{\gamma}(t)\big) \;=\; \mathbb{E}\, f\big(\tilde{\boldsymbol{\gamma}}\big) \;=\; f(\tilde{\boldsymbol{\gamma}}).$$

5. Заменяем $\tilde{\boldsymbol{\gamma}}$ на матожидание. $\tilde{\boldsymbol{\gamma}}$ детерминирован (a.s.), а $\tilde{\boldsymbol{\gamma}} = \bar{\boldsymbol{\gamma}}(0)$, откуда $f(\tilde{\boldsymbol{\gamma}}) = f\big(\bar{\boldsymbol{\gamma}}(0)\big)$. □

3 Оптимизационная задача

С учётом Теоремы 1 используем GS-сэмпл только внутри task-лосса, а в штрафе — детерминированный Softmax. Таким образом мы получили *одноуровневую* задачу оптимизации:

$$\mathbb{E}_{\boldsymbol{s}\sim U(\Delta^{k-1})} \Big[\; \mathbb{E}_{\boldsymbol{\gamma}\sim \text{GS}(\boldsymbol{\alpha}(\boldsymbol{s}),t)} \mathcal{L}_{\text{task}}(\boldsymbol{w}, \boldsymbol{\gamma}) \;+\; \kappa\, Cost\big(\bar{\boldsymbol{\gamma}}(\boldsymbol{s}); \boldsymbol{s}\big) \;\Big] \to \min_{\boldsymbol{w}, \boldsymbol{a}},$$

где $\boldsymbol{w}$ — все параметры внутри примитивных операций $\mathbf{o}^{(m)}$; $\boldsymbol{a}$ — параметры гиперсети $\boldsymbol{u}_{\boldsymbol{a}}$, которая генерирует логиты $\boldsymbol{\alpha}^{(i,j)}$ для всех $(i,j) \in E$ при подстановке $\boldsymbol{s}$; $\boldsymbol{s} = (\boldsymbol{s}\_1, \ldots, \boldsymbol{s}\_k) \in \Delta^{k-1}$ — вектор сложности, порождённый из равномерного распределения на симплексе Δ; $\mathcal{L}_{\text{task}}$ — кросс-энтропия по обучающей выборке; $\boldsymbol{\gamma}^{(i,j)}(\boldsymbol{s})$ — веса операций на ребре (i,j), получаемые через Gumbel-Softmax от логитов $\boldsymbol{u}_{\boldsymbol{a}}^{(i,j)}(\boldsymbol{s})$; $\boldsymbol{\gamma}$ — вектор состоящий из $\boldsymbol{\gamma}^{(i,j)}$; $\bar{\boldsymbol{\gamma}} = \mathbb{E}[\boldsymbol{\gamma}(t)]$; $Cost$ — штраф за использование дорогих операций; κ — гиперпараметр, управляющий важностью штрафа.

4 Вычислительные эксперименты

Эксперименты поставлены для подтверждения работоспособности метода и иллюстрации его возможностей. Для этих целей используются выборки Fashion-MNIST и CIFAR-10, где каждая из них разделена на равные части для обучения и тестирования.

- Эксперимент 1. Данные — Fashion-MNIST. Эпохи — 10. Список операций: 3×3 convolution, 3×3 max-pooling и identity (передача данных без изменений, далее *пропуск*).
- Эксперимент 2. Данные — Fashion-MNIST. Эпохи — 20. Список операций: 3×3 convolution, 5×5 convolution , 3×3 max-pooling, 3×3 avg-pooling и identity.
- Эксперимент 3. Данные — CIFAR-10. Эпохи — 50. Список операций: 3×3 convolution, 3×3 max-pooling и identity.

Каждая модель представляла собой нейронную сеть с фиксированными первым свёрточным *(головой)* и последним полносвязным слоем *(хвостом)*. Между головой и хвостом располагается n клеток по m вершин. Во всех экспериментах рассматриваемых в этой работе описываются конфигурации с $n = 3, m = 4$. Параметры $\boldsymbol{w}$ и параметры гиперсети $\boldsymbol{a}$ тренировались с помощью оптимизаторов Adam. Температура Gumbel–Softmax линейно уменьшалась от t=1.0 до t=0.2 по эпохам. Все эксперименты проводились на одной видеокарте NVIDIA RTX 3080 for notebooks, процесс поиска целиком занимал от двух до четырёх часов в зависимости от конфигурации.

Получение результатов. После обучения получено семейство нейронных сетей, из которого для анализа в каждом эксперименте выделялись четыре архитектуры, соответствующие приведённым ниже векторам сложности $\boldsymbol{s}$. Представленные архитектуры отражают равномерный штраф за операции (A), увеличенный за свёртки (B), увеличенный за пулинг (C) и увеличенный за пропуск (D). Для случая трёх доступных операций, эксперименты 1 и 3, вектора соответствуют штрафам (pool, ident, conv):

$$\text{(A) } (0.33, 0.33, 0.33), \quad \text{(B) } (0.15, 0.15, 0.70), \quad \text{(C) } (0.70, 0.15, 0.15), \quad \text{(D) } (0.15, 0.70, 0.15).$$

Для случая пяти доступных операций, эксперимент 2, вектора соответствуют штрафам (pool1, pool2, ident, conv1, conv2):

$$\text{(A) } (0.2, 0.2, 0.2, 0.2, 0.2), \ \text{(B) } (0.077, 0.077, 0.077, 0.38, 0.38),$$

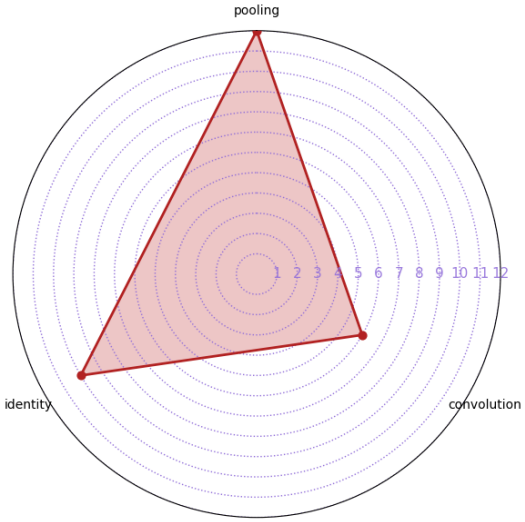
$$\text{(C) } (0.38, 0.38, 0.077, 0.077, 0.077), \ \text{(D) } (0.11, 0.11, 0.55, 0.11, 0.11)$$

4.1 Анализ результатов

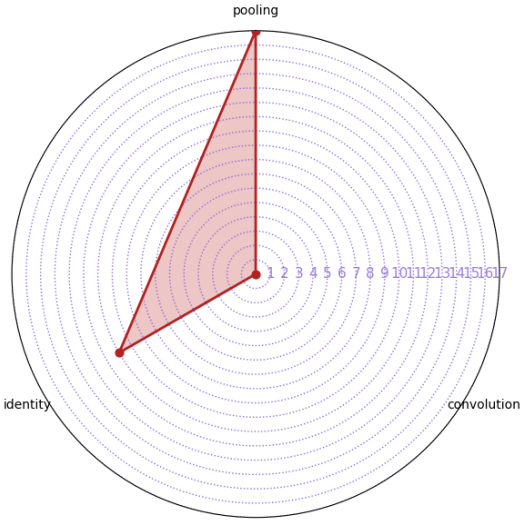
В данном разделе приводятся результаты трёх объявленных экспериментов и их анализ. Результаты представлены рисунком 2, на котором визуализировано распределение количества операций в архитектурах эксперимента 1 в зависимости от вектора сложности $\boldsymbol{s}$, и тремя таблицами со сравнениями метрик качества, количества параметров и количества слоёв в каждой архитектуре экспериментов 1,2,3 в зависимости от вектора сложности.

	(A) uniform	(B) conv. penalty	(C) pool. penalty	(D) ident. penalty
Test accuracy [%]	82.5	79.2	**85.0**	81.4
Parameter count	38 304	**5 120**	**143 488**	12 320
# Pooling layers	12	17	**0**	18
# Convolution layers	6	**0**	13	9
# Identity connections	10	11	15	**1**

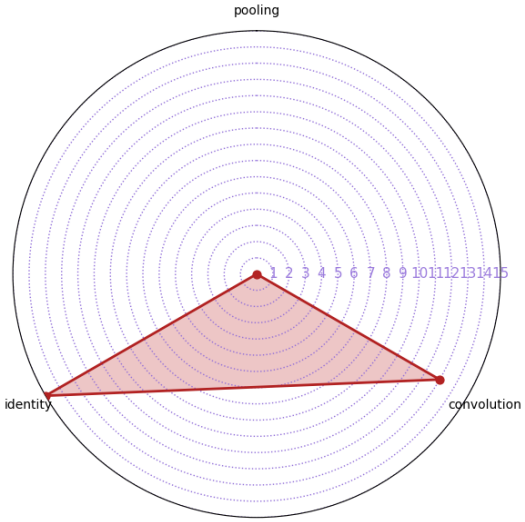
Таблица 1: Сравнение количества слоёв в архитектурах эксперимента 1.



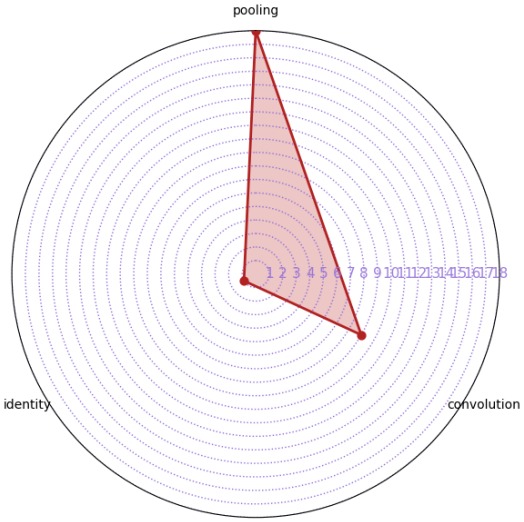
(A) равномерный штраф



(B) увеличенный за свёртки



(C) увеличенный за пулинг



(D) увеличенный за пропуск

Рис. 2: Статистика по операциям для четырёх архитектур эксперимента 1: **(A)** равномерный штраф, **(B)** увеличенный за свёртки, **(C)** увеличенный за пулинг, **(D)** увеличенный за пропуск.

	(A) uniform	(B) conv. penalty	(C) pool. penalty	(D) ident. penalty
Test accuracy [%]	85.62	82.37	**85.63**	85.60
Parameter count	93 888	**25 280**	**137 632**	72 928
# Pooling layers	16	22	**4**	22
# Convolution layers	8	**1**	13	6
# Identity connections	4	5	11	**0**

Таблица 2: Сравнение количества слоёв в архитектурах эксперимента 2.

	(A) uniform	(B) conv. penalty	(C) pool. penalty	(D) ident. penalty
Test accuracy [%]	71.02	52.37	**73.77**	68.43
Parameter count	113 984	10 240	**159 456**	**8 800**
# Pooling layers	4	12	**0**	17
# Convolution layers	18	**0**	9	11
# Identity connections	6	16	19	**0**

Таблица 3: Сравнение количества слоёв в архитектурах эксперимента 3.

Результаты экспериментов показывают, что при равномерном штрафе используются все операции и получается среднее, среди рассмотренных вариантов, качество и количество параметров. При увеличенном штрафе за свёртки падает качество модели и минимизируется использование соответствующих операций, при этом количество параметров уменьшается, что можно использовать при соответствующих эксплуатационных ограничениях. При увеличенном штрафе за пулинг уменьшается количество операций пулинга. За счёт этого возрастает количество свёрток и соответственно увеличивается количество параметров, а также получается наилучшее качество. При увеличенном штрафе за пропуск исчезают операции identity.

Общие выводы по экспериментам:

- **Контроль операций**. Результаты экспериментов подтверждают возможность гибкого контроля над типами операций: при использовании вектора сложности с увеличенным штрафом за операцию — она перестаёт использоваться в модели.
- **Управляемая сложность.** Метод позволяет изменяя вектор $\boldsymbol{s}$ перемещаться вдоль парето-фронта точность-сложность, получая модели с количеством параметров от 5 до 150 тысяч.
- **Эксплуатационные ограничения.** Проассоциировав операции со временем их выполнения на устройстве, метод позволяет подобрать архитектуру для уменьшения ожидаемого времени выполнения модели на целевом устройстве, за счёт уменьшения количества долго выполняющихся операций. Аналогично можно поступить при необходимости минимизировать не время, а энергетические затраты или любые другие эксплуатационные ограничения.
- **Семейство моделей.** Все рассматриваемые варианты внутри каждого отдельного эксперимента порождены одной обученной моделью, таким образом метод представляет целое семейство готовых моделей и механизм гибкого выбора из него.
- **Согласованность результатов.** Все рассмотренные зависимости сохраняются на разных наборах и списках операций, что подтверждает универсальность метода.

5 Заключение

Предложен метод для выбора архитектуры модели глубокого обучения с контролем характеристик на целевом устройстве сочетающий точность модели и эффективность аппаратного исполнения. Преимущества метода в том, что за один процесс обучения порождается семейство архитектур, а введённый вектор сложности позволяет интуитивно контролировать операции при выборе моделей. Такой подход позволяет получать модели под конкретные аппаратные ограничения без дополнительного обучения под эти требования. В экспериментальной части подтверждена работоспособность метода на разных наборах данных и при различных наборах допустимых операций.

В рамках дальнейшего развития метода планируется добавление многокритериального учета сложности и проведение полноформатного сопоставления результатов с классическими NAS-подходами.

Список литературы

[1] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations (ICLR)*, 2020.

[2] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware, 2019.

[3] X. Chen and ... Snas: Stochastic neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

[4] Jin Dong, Yiming Yang, and ... Bignas: Scaling up neural architecture search with big single stage models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3173–3182, 2021.

[5] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel–softmax. In *International Conference on Learning Representations (ICLR)*, 2017.

[6] Xin Li and ... Modnas: Modality adaptive one-shot neural architecture search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[7] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

[8] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations (ICLR)*, 2017.

[9] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[10] Konstantin Yakovlev, Olga Grebenkova, Oleg Bakhteev, and Vadim Strijov. Neural architecture search with structure complexity control. In *Recent Trends in Analysis of Images, Social Networks and Texts*, pages 207–219, 2022.