



тема

# Neural Architecture Search (NAS)

спикер



Тимофей  
Науменко



R&D инженер  
@ Wanna Fashion

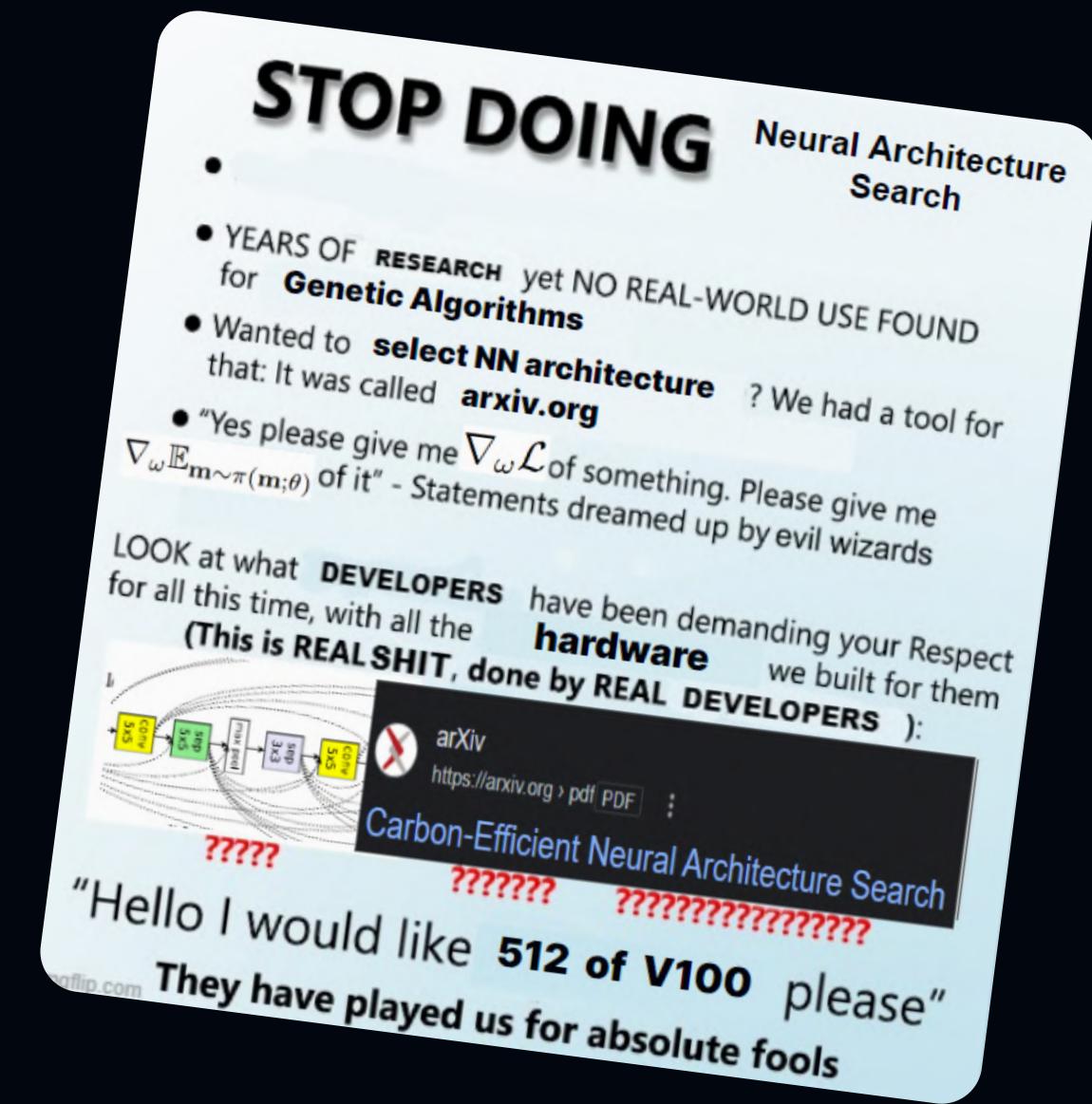
# Содержание

Постановка задачи NAS →



# Содержание

Постановка задачи NAS → Простой  
подход →



# Содержание

Постановка задачи NAS → Простой  
подход → **Weight sharing** →



# Содержание

Постановка задачи NAS → Простой подход → Weight sharing → **Zero-shot NAS** →



# Содержание

Постановка задачи NAS → Простой подход → Weight sharing → Zero-shot NAS → Реальное применение →



# Постановка задачи NAS

## Задача NAS

- определить оптимальную архитектуру НС для фиксированных задач и датасета

## Пример

- найти оптимальную архитектуру НС для классификации изображений из ImageNet

## Критерий

- классификация изображений

## Датасет

- ImageNet

## Критерий оптимальности

- Classification accuracy

Fox → → →



# Самое простое решение

## Решение #1

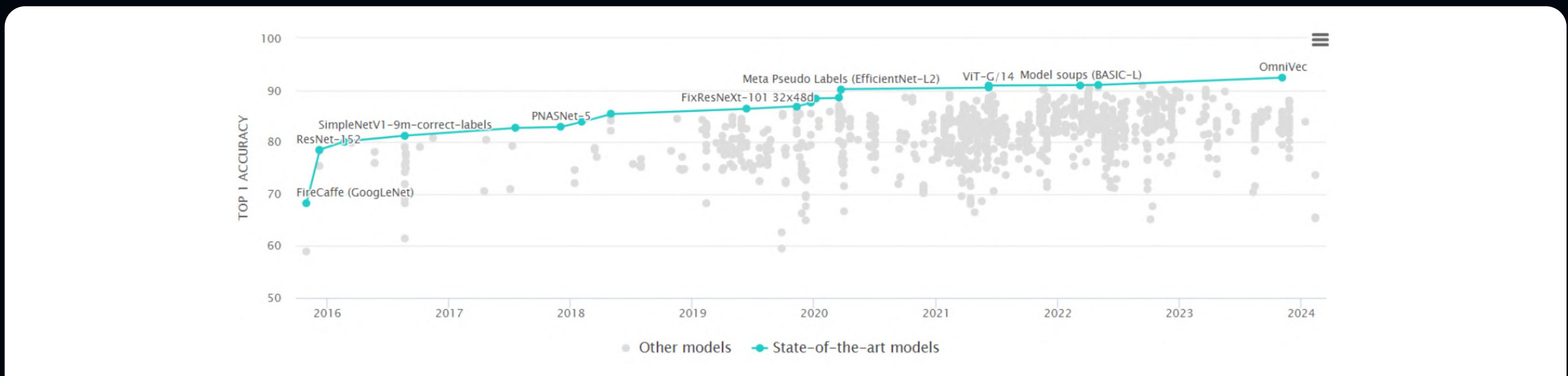
- Залезть на [paperswithcode.com](https://paperswithcode.com) ↗, выбрать лучшую модель с выложенными весами

## Проблема

- модель может вам не подходить по ряду причин (качество, производительность, не конвертируется под iOS, слишком много весит, ...).

## Проблема

- для вашей задачи или датасета может не быть модели на paperswithcode



## Самое распространённое решение

Решение #2

- Придумать архитектуру самостоятельно, используя идеи из статей, *human bias*. Реализовать алгоритм обучения и подобрать гиперпараметры

Проблемы

- Решение неоптимально, требуется много ручной и исследовательской работы

Хочется, чтобы всё было автоматизировано

```
model = Sequential()
model.add(Dense(784, input_shape=(784,), activation='relu'))
model.add(Dense(1000, activation='relu'))
model.add(Dense(1000, activation='relu'))
model.add(Dense(1000, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer=RMSprop(), loss='categorical_crossentropy', metrics=['accuracy'])
#model.fit(train_array_data, train_array_labels, batch_size=128, epochs=30, validation_dat
model.fit(train_array_data, train_array_labels, batch_size=128, epochs=30, validation_data
test_array_result = model.predict_classes(test_array_data , batch_size=128 , verbose=0 )
test_array_result = np.array(test_array_result)
|
test_array_ids = []
for i in range(112):
    test_array_ids.append(i)

test_array_ids = np.array(test_array_ids)

d = {'ids' : test_array_ids , 'result' : test_array_result}
df1 = pd.DataFrame.from_dict(d, orient='index')
df1 = df1.transpose()
df1.to_csv("sub.csv", index=False)
```

## Простейшая автоматизация: цикл `for`

Решение #3

→ **Автоматизация.** Написать алгоритм, который будет перебирать различные архитектуры, пока не найдём хорошую

Возьмём топовые архитектуры — MobileNeXt, EfficientNetV2, ViT, ResNeXt, SWIN, и другие. Напишем цикл `for` по всем, будем сидеть ждать, пока обучится.

Проблема

→ разным сетям нужны разные гиперпараметры



# Пространство поиска

## Проблема

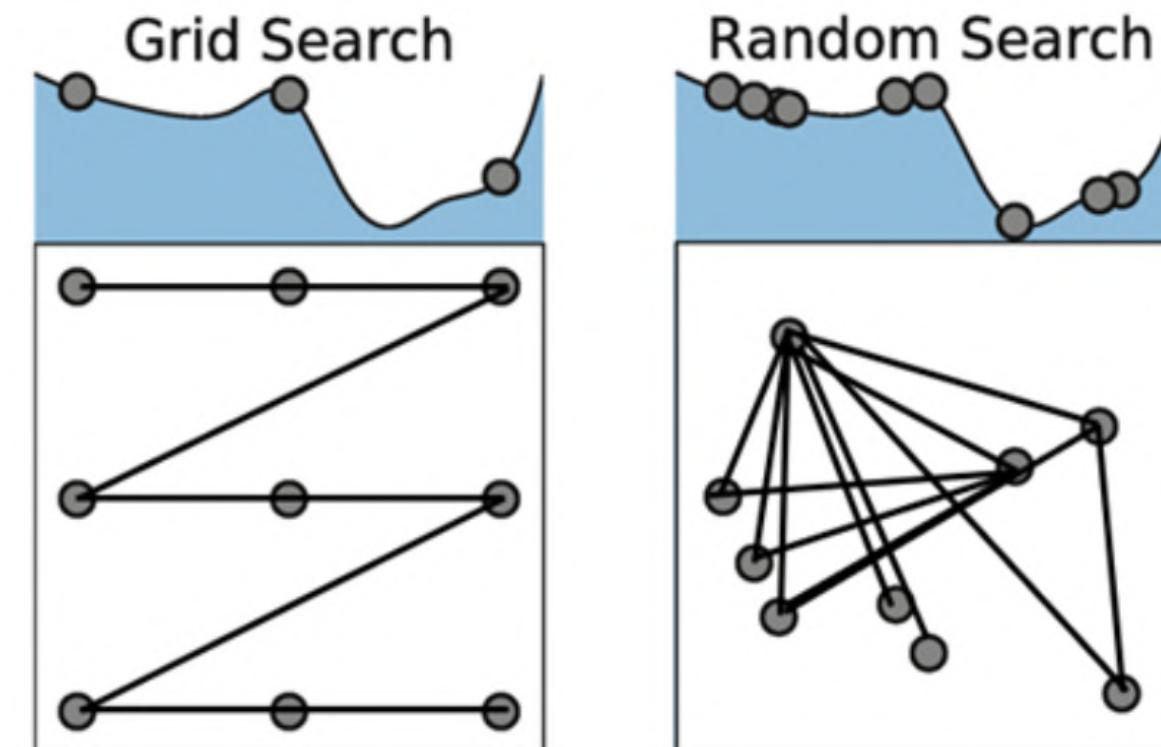
- надо как-то ограничить список нейросетей, чтобы можно было их все обучать с одинаковыми гиперпараметрами

## Предположение

- одни и те же гиперпараметры будут работать примерно одинаково для похожих архитектур

## Решение

- Создадим **пространство поиска**, в котором будет много однотипных архитектур

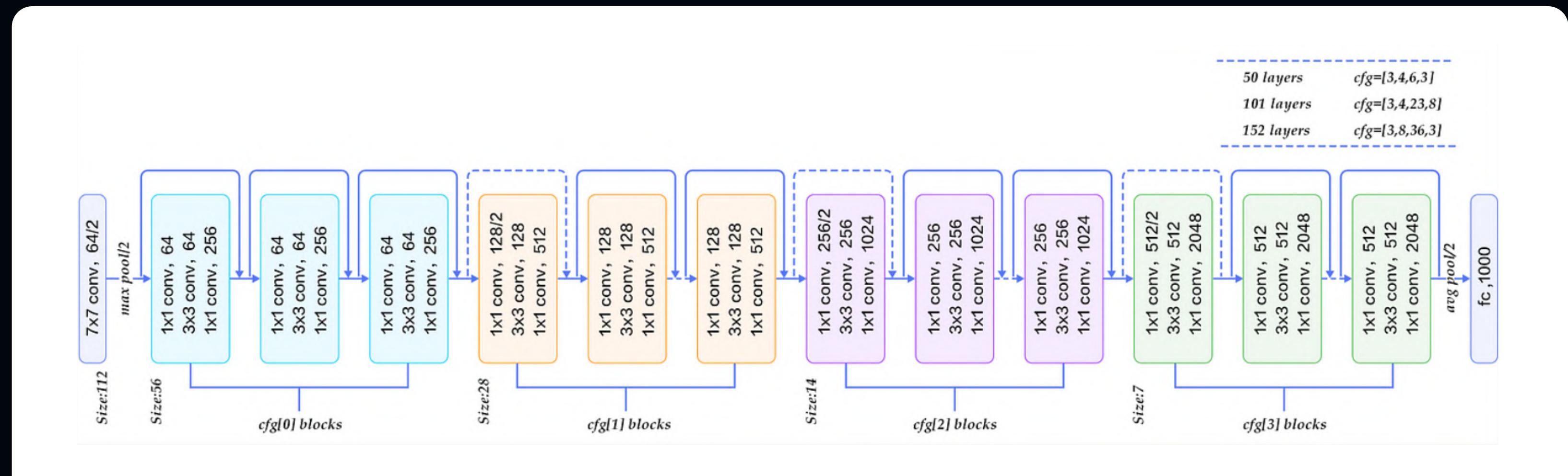


# Перебор по пространству поиска

## Решение #4

→ Написать алгоритм, который будет перебирать архитектуры из пространства поиска

Для простоты пространство поиска составим из ResNet-подобных архитектур. В каждом слое будем перебирать размер ядра ( $3 \times 3$ ,  $5 \times 5$ ,  $7 \times 7$ ), число каналов (16, 32, 64, ..., 1024). Число слоёв пусть будет как в ResNet-50.



# Перебор по пространству поиска

## Проблема

- для такого пространства цикл for не подойдет, там  $10^{77}$  архитектур

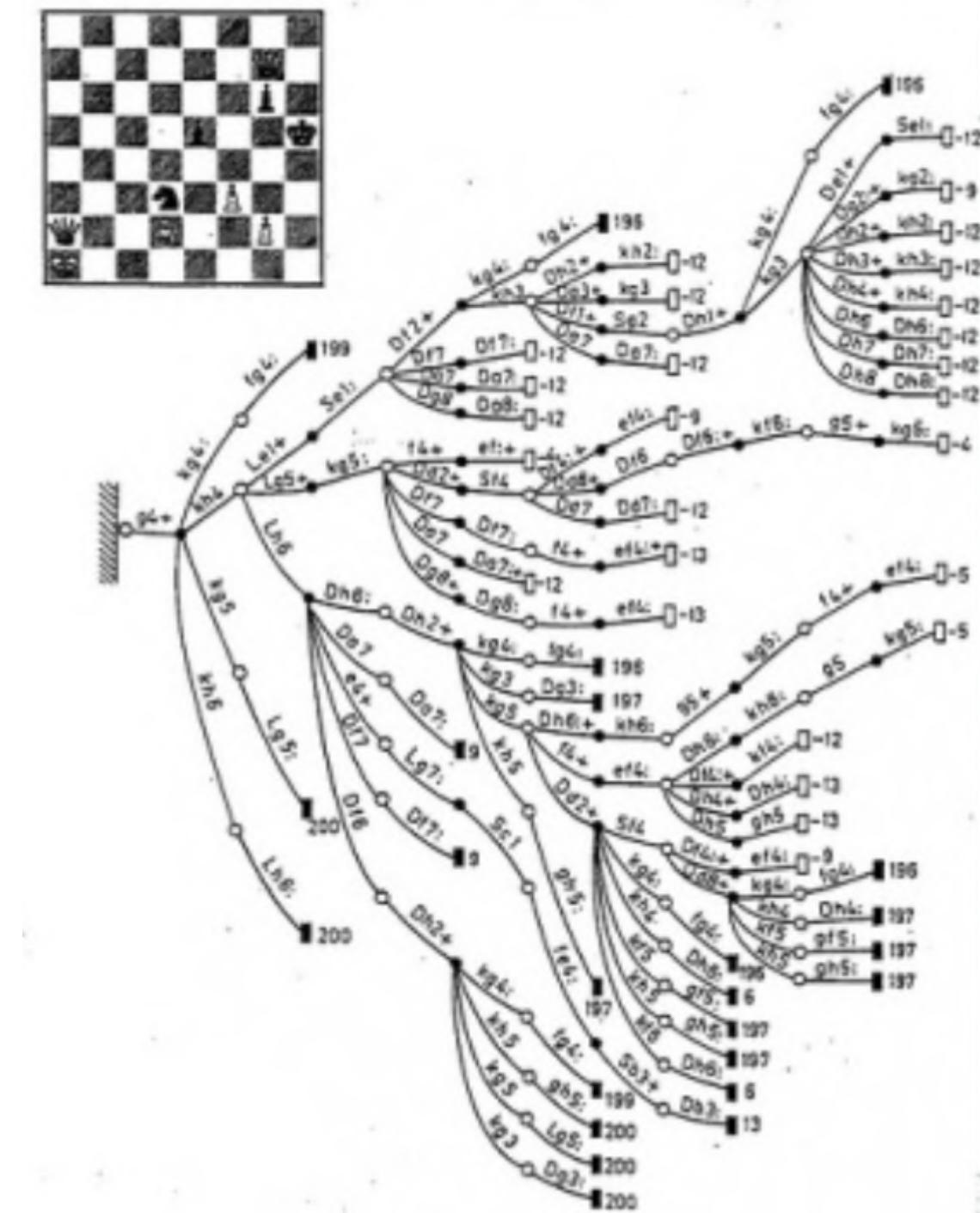
Как решать проблему большого пространства поиска?

### 1 Сократить пространство поиска

Первое решение сводится к тому, что пространство поиска сокращается настолько, что наш перебор можно назвать перебором гиперпараметров

### 2 Не перебирать все архитектуры

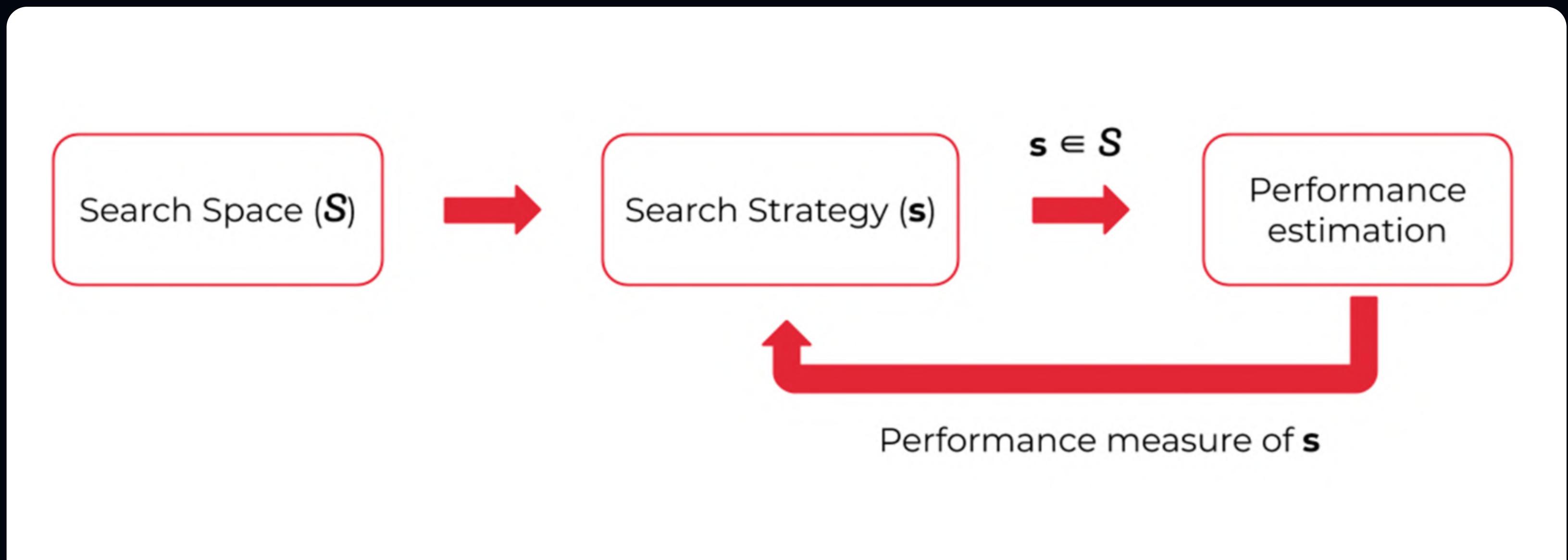
Второй вариант предлагает исследовать пространство поиска, выбирая только малую часть возможных архитектур. Именно это и предполагает NAS



## Простейший вариант NAS

Решение #5

→ **Neural Architecture Search.** Алгоритм, который семплирует различные архитектуры из пространства поиска в поисках оптимальной



# Простой подход

# Пространство поиска

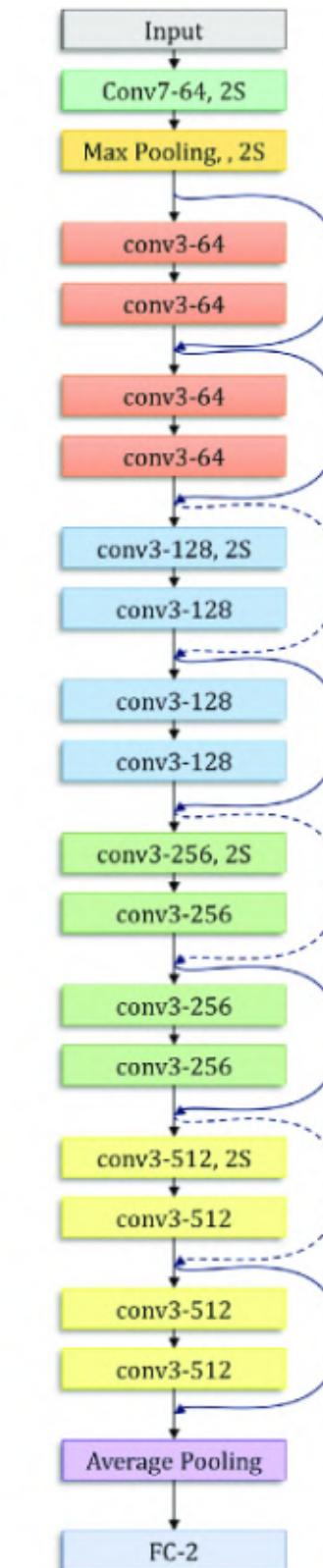
В качестве базовой модели возьмём ResNet-18. Будем подбирать параметры каждого блока

Две свёртки + skip connection

Число каналов между двумя свёртками в блоке:  $\times 0.5$ ,  $\times 1$ ,  $\times 2$  от числа каналов в базовой ResNet-18.

Размеры ядер в каждом блоке:  $3 \times 3$ ,  $5 \times 5$ ,  $7 \times 7$

Размер пространства поиска:  $9^8$



# Random Search

Для выбора архитектур воспользуемся методом **Random Search**.

В каждом блоке выберем случайный размер ядра и число каналов. Все архитектуры **равновероятны**.

Выберем 1000 случайных архитектур, обучим каждую и замерим accuracy.

```
def sample_architecture():
    config = []
    for block_index in range(8):
        kernel = random.choice([3, 5, 7])
        expansion_ratio = random.choice([0.5, 1, 2])
        config.append([kernel, expansion_ratio])
    return config

max_accuracy, best_arch = 0, None
for i in range(1000):
    arch_config = sample_architecture()
    accuracy = train_arch(arch_config)
    if accuracy > max_accuracy:
        max_accuracy = accuracy
        best_arch = arch_config

print(best_arch)
```

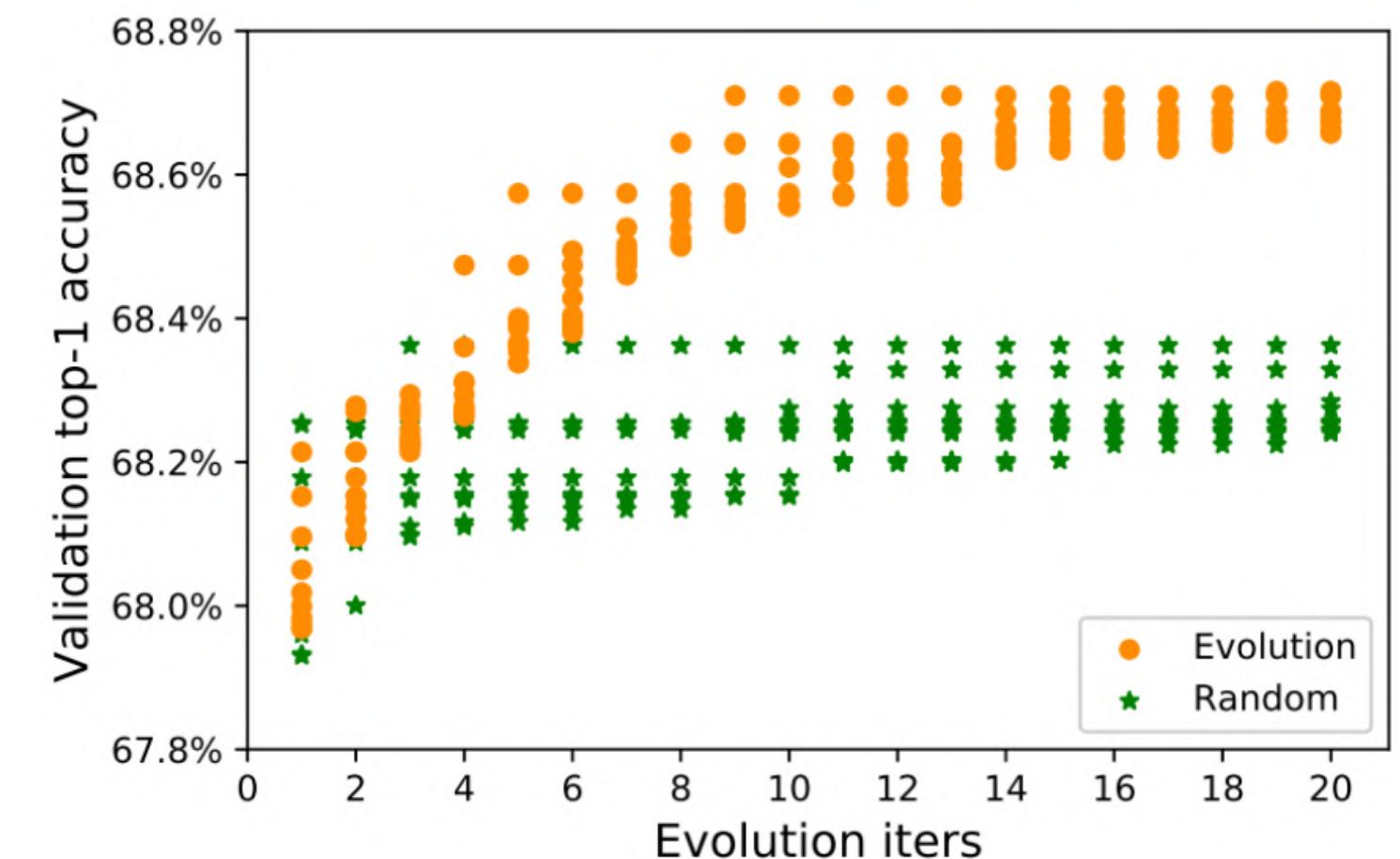
# Другие методы поиска

Идея

→ давайте использовать accuracy как сигнал для метода сэмплирования!

Получаем такие алгоритмы, как:

- Bayesian optimization
- Evolutionary algorithms (GA, MOGA, NSGA-III, ...)
- Reinforcement learning (REINFORCE, PPO, ...)



Простой подход

20

# В чём проблема?

# Классический NAS

## Проблема

→ Надо обучить очень много нейросетей!

## Возможные варианты решения

→ Взять меньше данных / Обучать меньше эпох (early stopping, Successive Halving, Hyperband) / Экстраполировать кривую обучения

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	#ops	Search Method
DenseNet-BC (Huang et al., 2017)	3.46	25.6	–	–	manual
NASNet-A + cutout (Zoph et al., 2018)	2.65	3.3	2000	13	RL
NASNet-A + cutout (Zoph et al., 2018) <sup>†</sup>	2.83	3.1	2000	13	RL
BlockQNN (Zhong et al., 2018)	3.54	39.8	96	8	RL
AmoebaNet-A (Real et al., 2018)	$3.34 \pm 0.06$	3.2	3150	19	evolution
AmoebaNet-A + cutout (Real et al., 2018) <sup>†</sup>	3.12	3.1	3150	19	evolution
AmoebaNet-B + cutout (Real et al., 2018)	$2.55 \pm 0.05$	2.8	3150	19	evolution
Hierarchical evolution (Liu et al., 2018b)	$3.75 \pm 0.12$	15.7	300	6	evolution
PNAS (Liu et al., 2018a)	$3.41 \pm 0.09$	3.2	225	8	SMBO

# Weight sharing

# Основная идея

Проблема

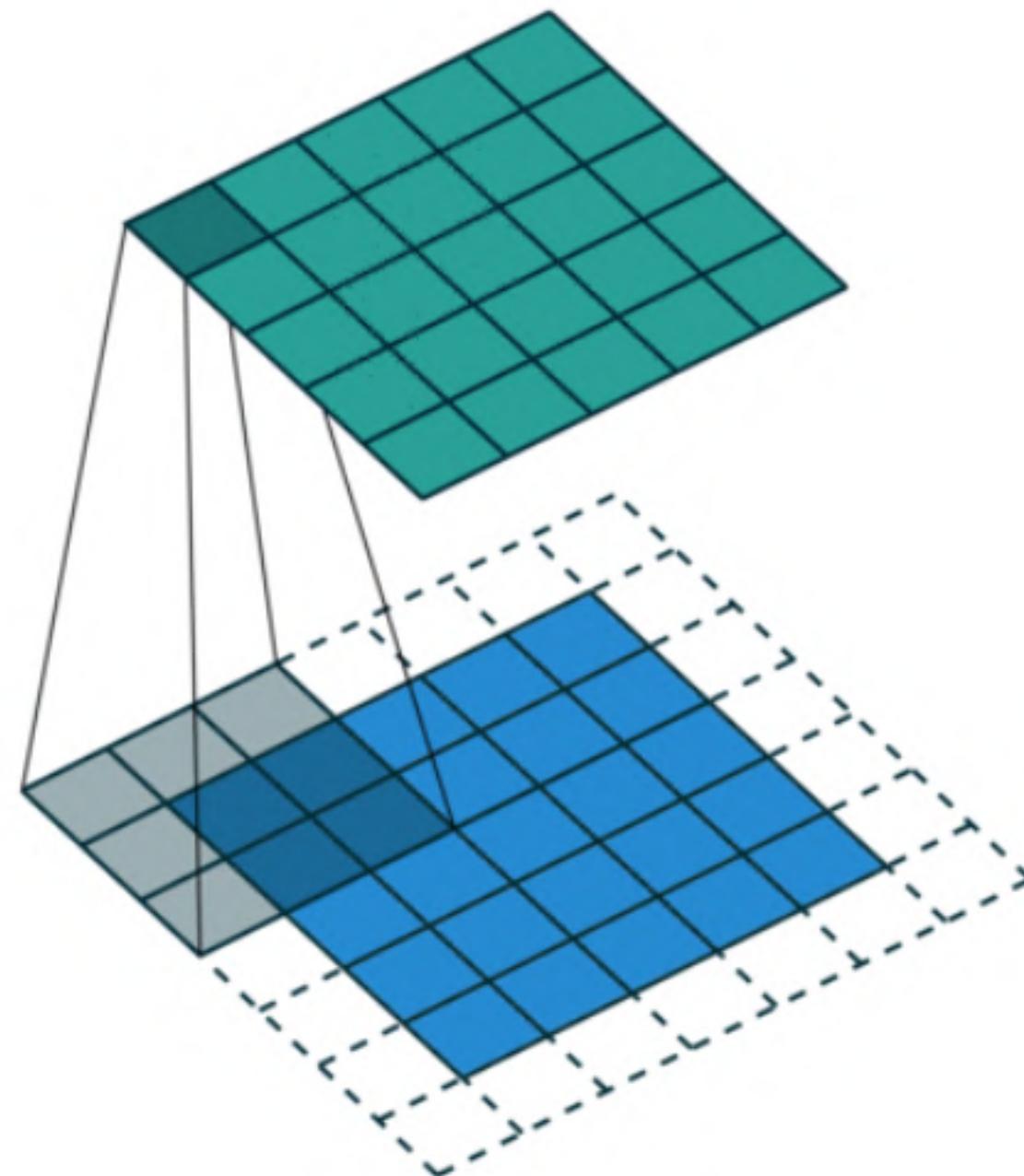
- надо обучить очень много нейросетей!

Идея 🌱

- Давайте попробуем использовать одни и те же веса в разных архитектурах

Реализация

- ???

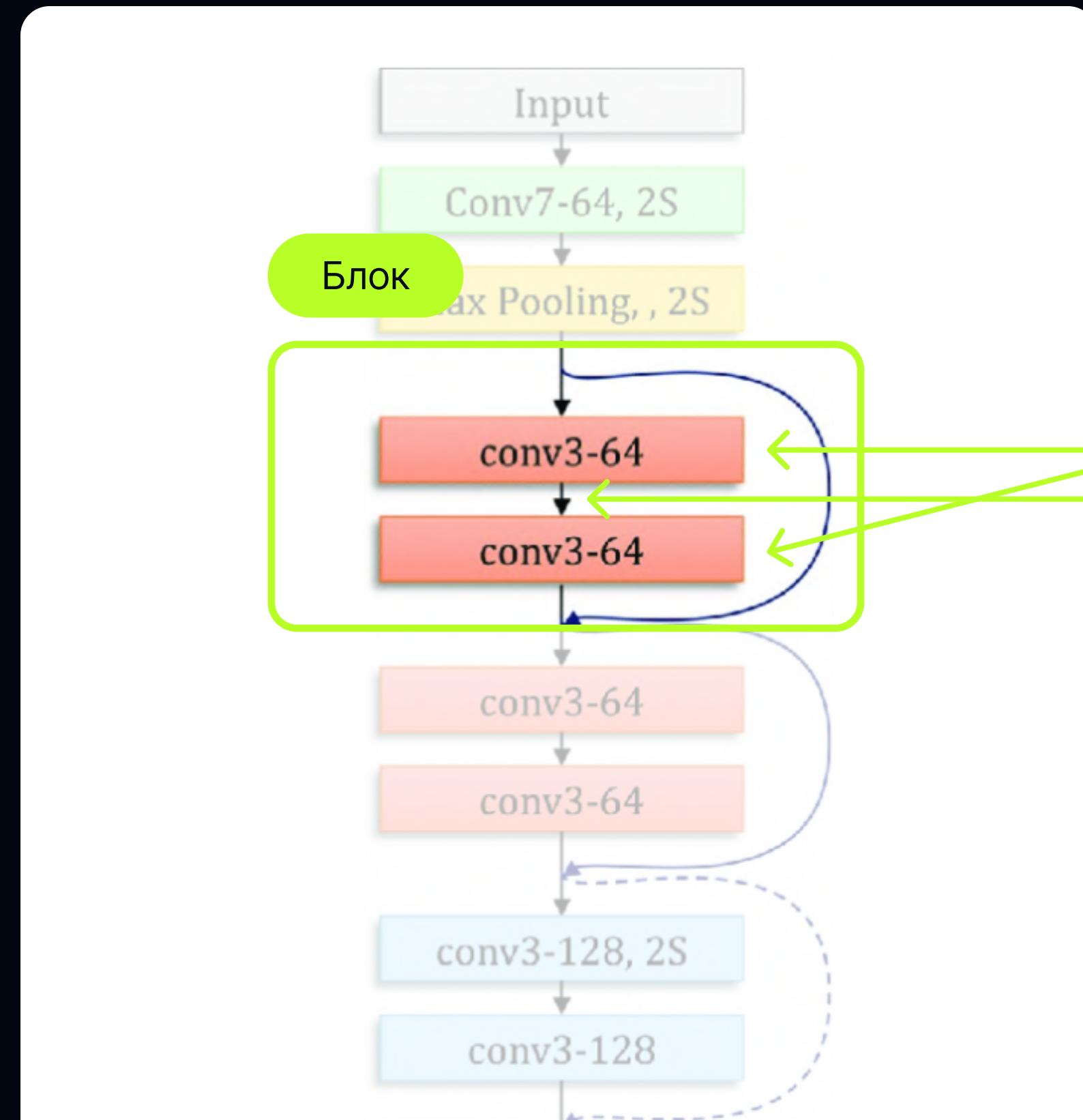


# Превращаем поиск в сборку LEGO

Вспоминаем про наше пространство поиска.

В каждом блоке доступно 9 различных опций: { $(\times 0.5, 3 \times 3)$ ,  $(\times 1, 3 \times 3)$ , ...,  $(\times 2, 7 \times 7)$ }.

В ResNet-18 восемь блоков. Итого мы имеем 72 кубика, из которых можно собирать архитектуры. Теперь бы их обучить так, чтобы они подходили друг к другу...



Размер ядра:

$3 \times 3, 5 \times 5, 7 \times 7$

Число каналов:

$\times 0.5, \times 1, \times 2$

# Как обучать?

# Пространство поиска

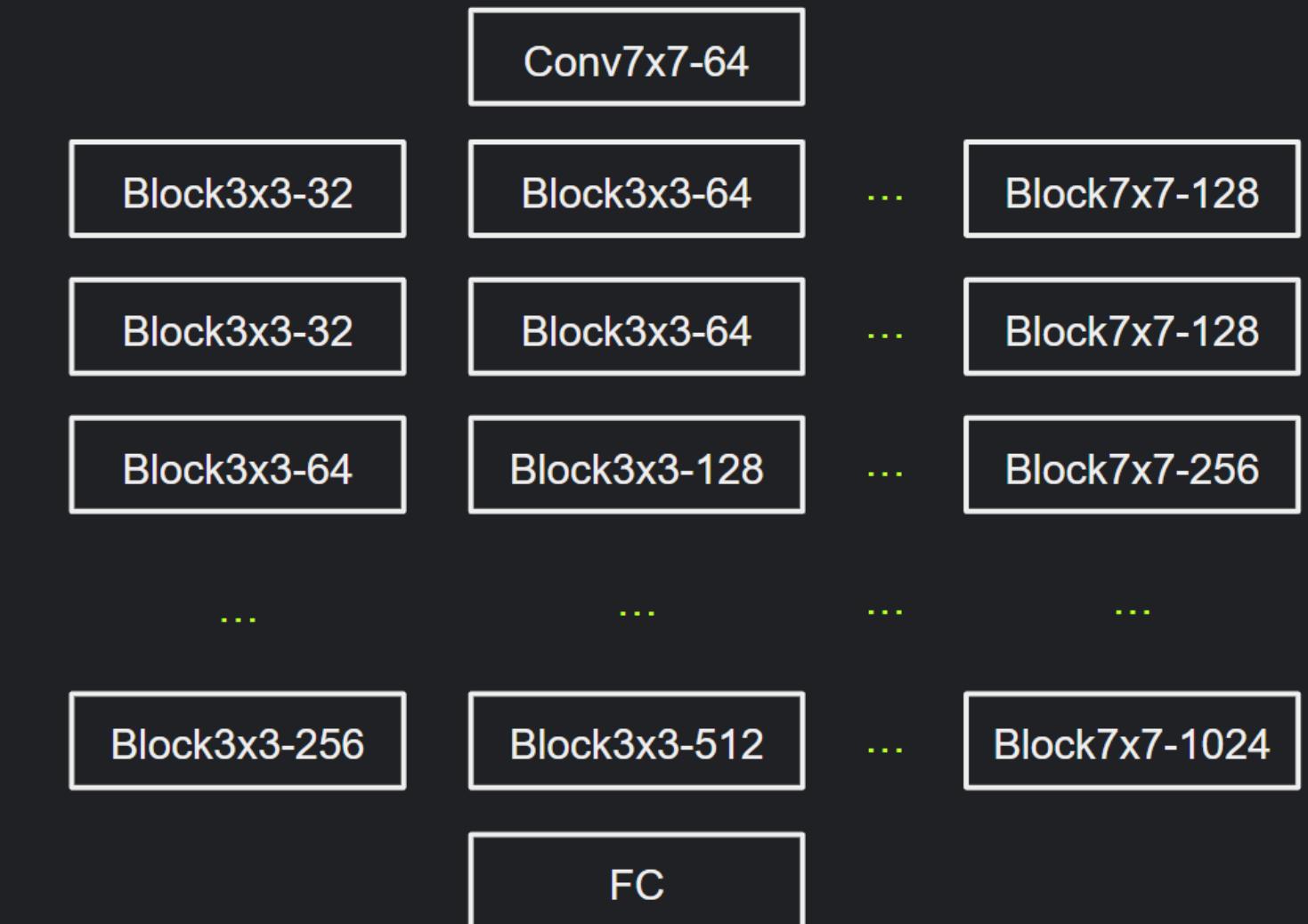
Идея

- Давайте хранить веса 72 блоков, а также веса первой свёртки и FC слоя на классификацию

Идея

- Будем выбирать по одной случайной архитектуре, собирать её веса из набора справа и обучать. Повторим до сходимости

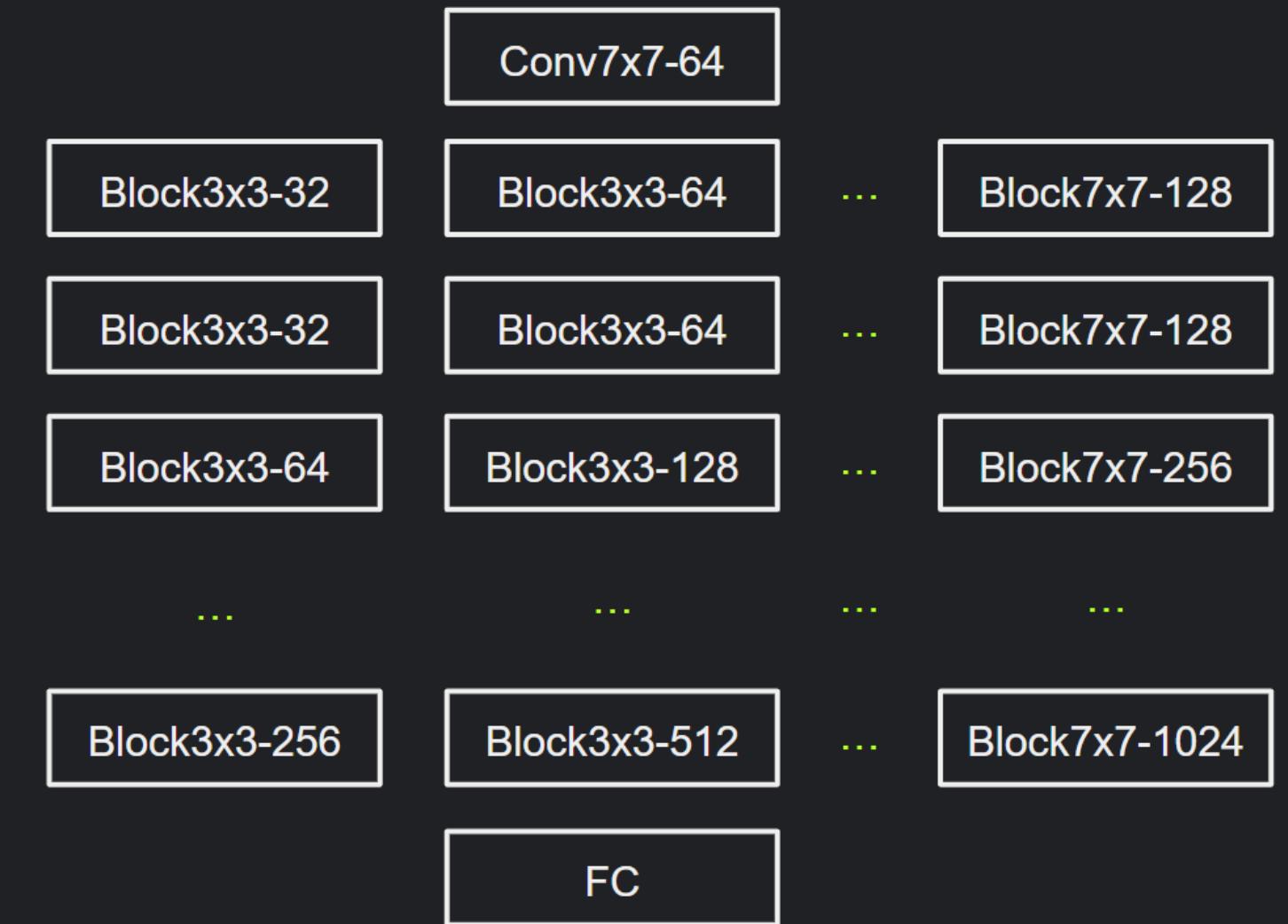
В результате одни и те же веса будут использоваться при обучении разных архитектур, отсюда и название — weight sharing



## Как искать?

Ну, допустим, расширили веса. Теперь можем замерять точности и лоссы для каждой архитектуры. А как найти лучшую? Есть две опции:

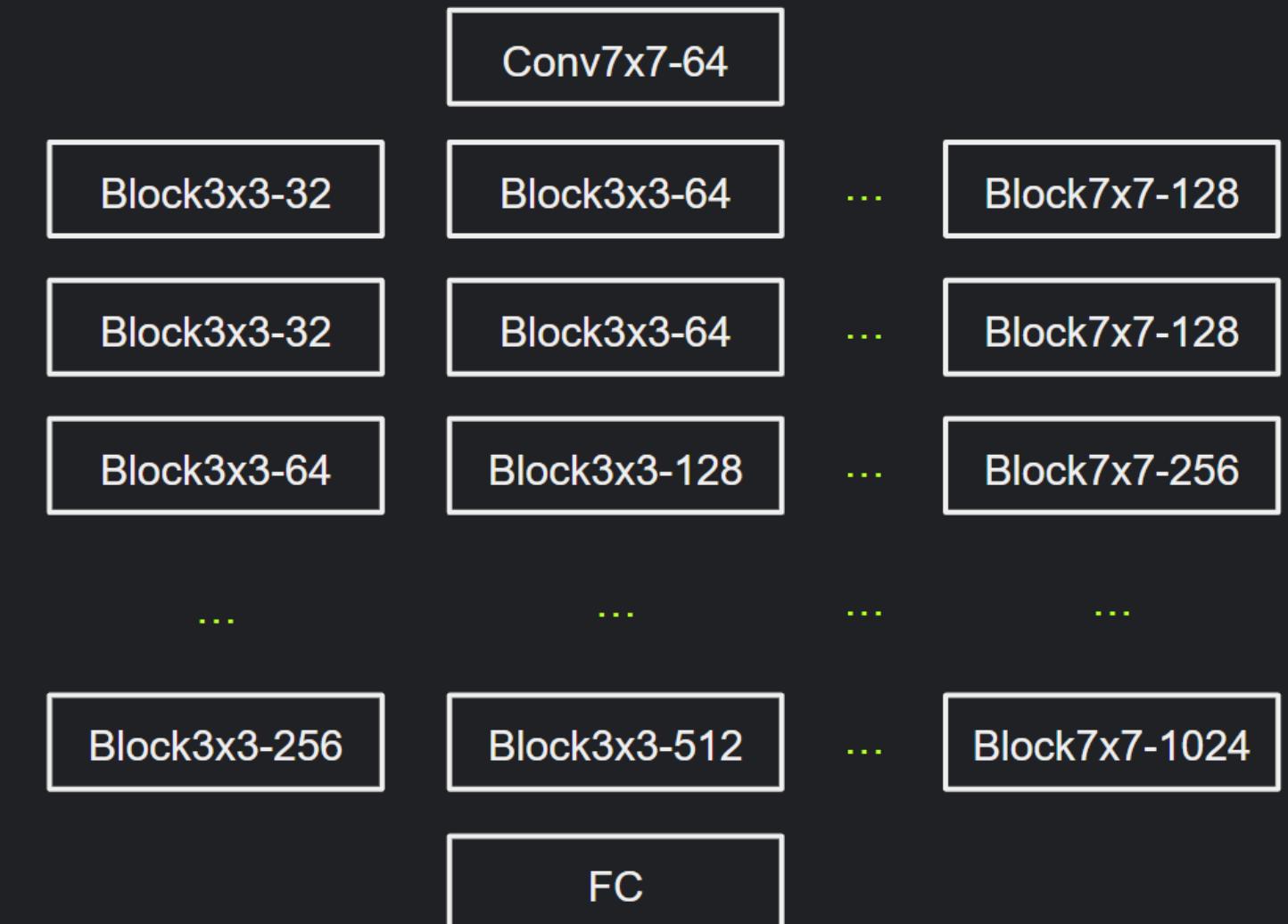
- 1 Ждём когда сойдётся — а там уже random search, RL или генетические алгоритмы.
- 2 В процессе обучения можно выбирать не случайную архитектуру, а с помощью RL. При этом надо замерять точность модели на валидации и скормливать её как reward для RL.



## Блочное пространство поиска

Заметим, что у нас есть выбор из нескольких операций для каждого слоя финальной сети.

Пространство поиска, в котором финальная архитектура получается с помощью выбора одной из операций в каждой позиции — называется **блочным**.



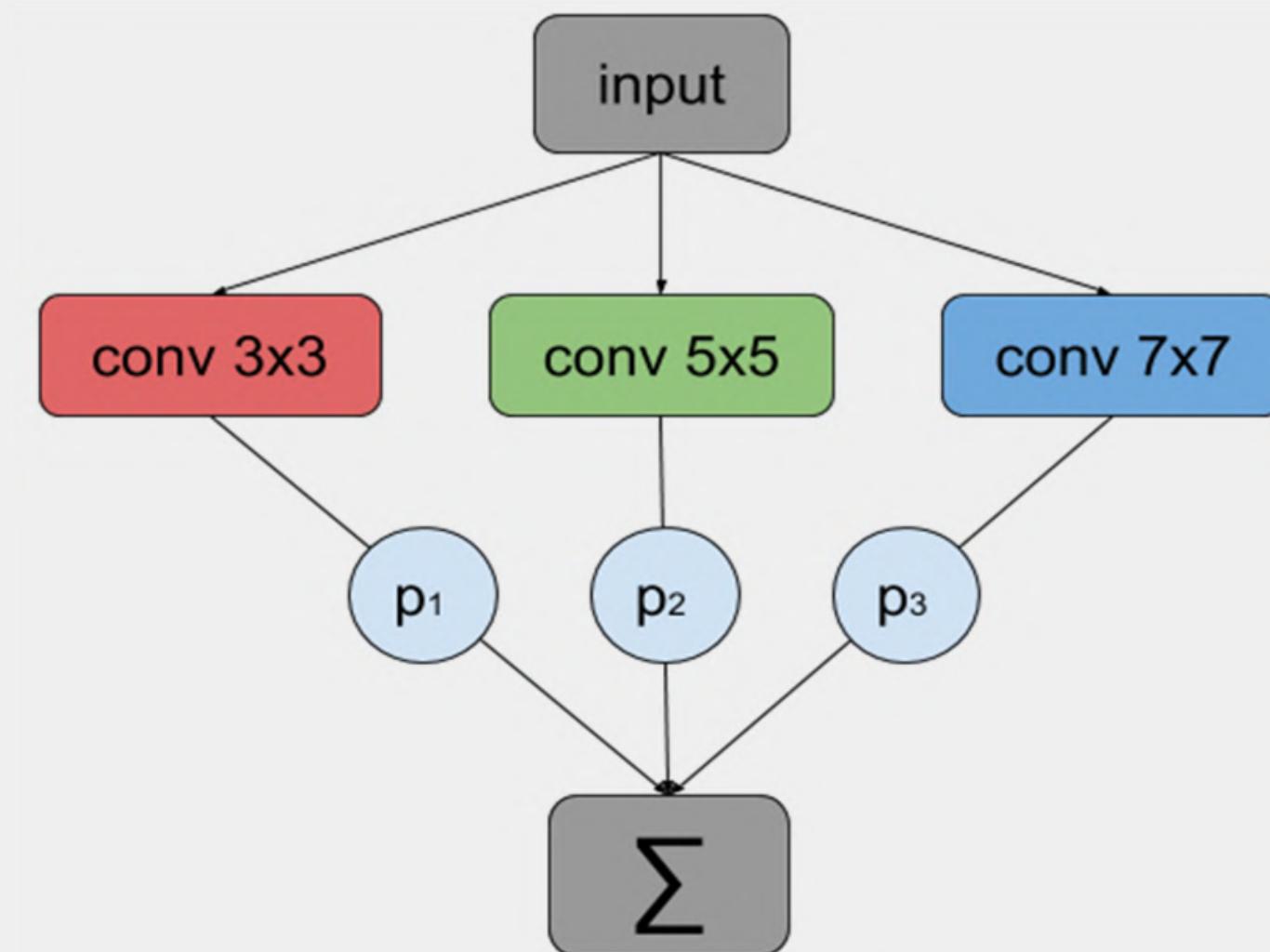
# Основная идея

## Идея

- Возьмем все операции в каждом блоке, сложим их с какими-то весами, а потом будем эти веса обучать. В конце выберем операцию с самым большим весом

## Аналогия

- Механизм внимания (attention), самые полезные операции будут иметь больший вес



# Процедура обучения и поиска

Обучение весов  
и обучение  
вероятностей:

$$\min_{\alpha} \quad \mathcal{L}_{\text{val}}(w^*(\alpha), \alpha) \quad \text{s.t.} \quad w^*(\alpha) = \operatorname{argmin}_w \mathcal{L}_{\text{train}}(w, \alpha)$$

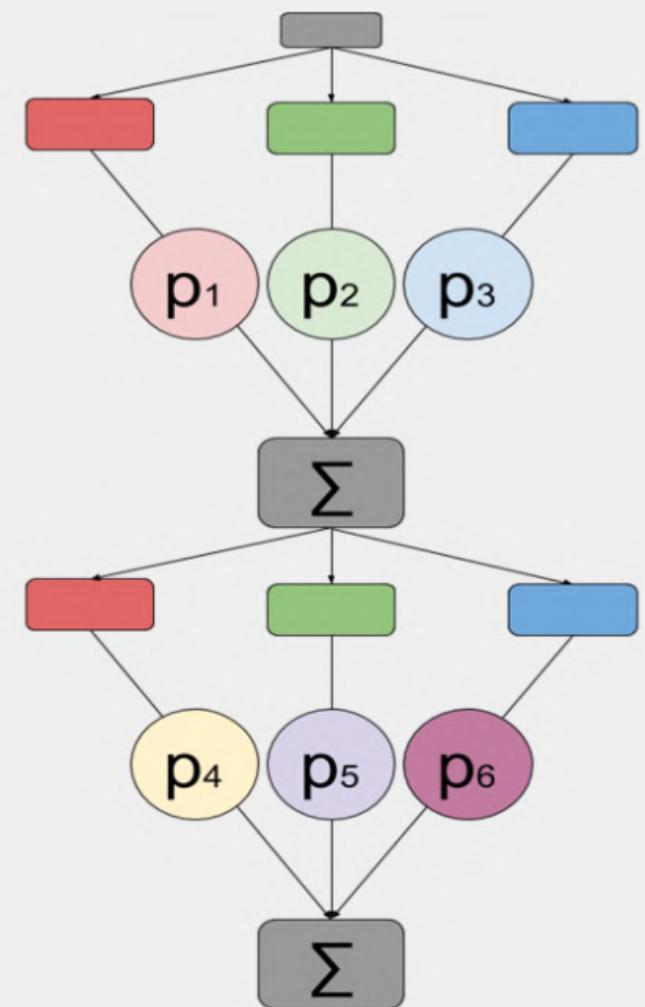
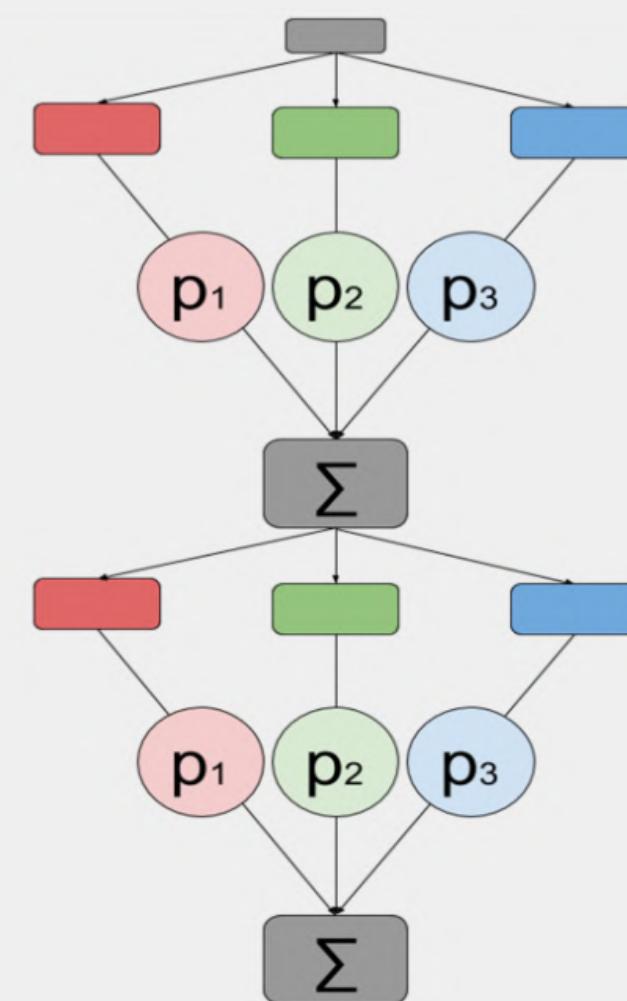


# Пространства поиска

Есть две опции:

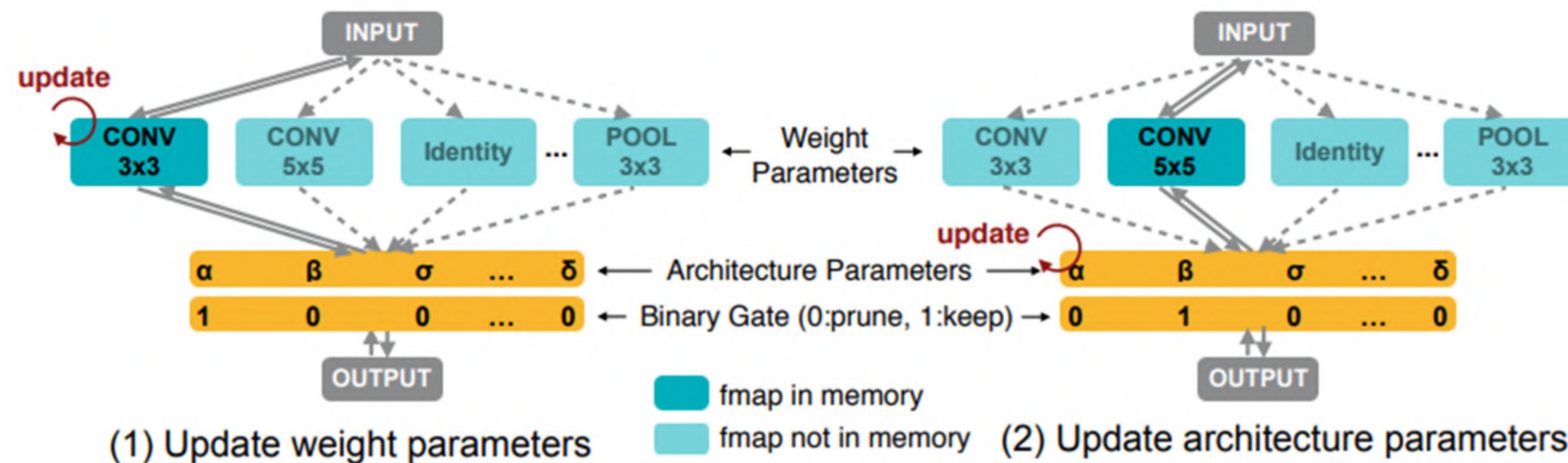
- ① **Micro-search.** Ищем один блок, который повторяется во всей сети
- ② **Macro-search.** Все блоки независимые

На примере слева — в первой архитектуре всего два независимых параметра ( $p_1, p_2$ ), во второй — четыре ( $p_1, p_2, p_4, p_5$ ).



# ProxylessNAS

Основная идея



## Сравнение с DARTS

$$m_O^{DARTS} = \sum_{j=1}^N p_j o_j(x) = \sum_{j=1}^N \frac{\exp(\alpha_j)}{\sum_k \exp(\alpha_k)} o_j(x)$$

$$m_O^{Proxyless} = \sum_{j=1}^N g_j(p) o_j(x) = \begin{cases} o_1(x) \text{ with proba } p_1 \\ o_2(x) \text{ with proba } p_2 \\ \dots \\ o_N(x) \text{ with proba } p_N \end{cases}$$

# Сравнение с DARTS

	DARTS	ProxylessNAS
Пространство поиска	Micro-search	Macro-search
Операции (кандидаты)	DWS $3 \times 3$ , DWS $5 \times 5$ , maxpool, avgpool, ...	MBConv [ $3 \times 3$ , $5 \times 5$ , $7 \times 7$ ] x [3,6]
Процедура обучения	Один forward — обновление всей суперсети	Один forward — обновление только одного пути
Процедура поиска	Обновляем все веса разом во всей сети	Обновляем только по 2 веса в каждом блоке

Наконец-то принимаем  
во внимание latency

$$E[\text{latency}_i] = \sum_{j=0}^i p_j^i L(o_j^i)$$

$$\text{Loss} = \text{Loss}_{CE} + \lambda_1 \|w\|_2^2 + \lambda_2 \mathbb{E}[\text{latency}]$$

# SPOS

## Основная идея

---

### Идея

- Не обновлять итеративно вероятности операций и их веса, а разделить процедуру NAS на две стадии: *pretrain* (претрейн) и *search* (поиск)
  - На стадии претрэйна на каждый батч равновероятно семплируется одна архитектура из пространства поиска
  - При детекции отдельных слов (символов) не делать это вплотную к словам, а оставить немного места вокруг
-

Таким образом, все операции  
выбираются **одинаковое число раз**,  
у всех **одинаковый** шанс показать  
себя...

**ВЕДЬ ТАК?**

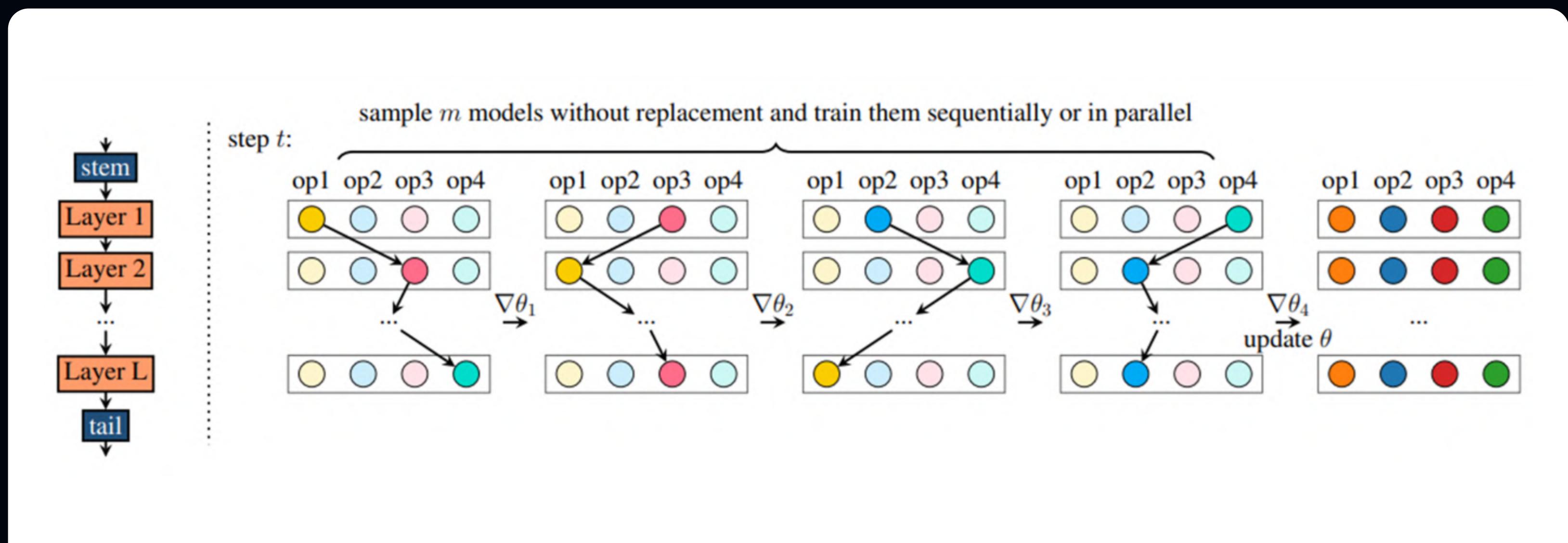
# FairNAS

## Основная идея

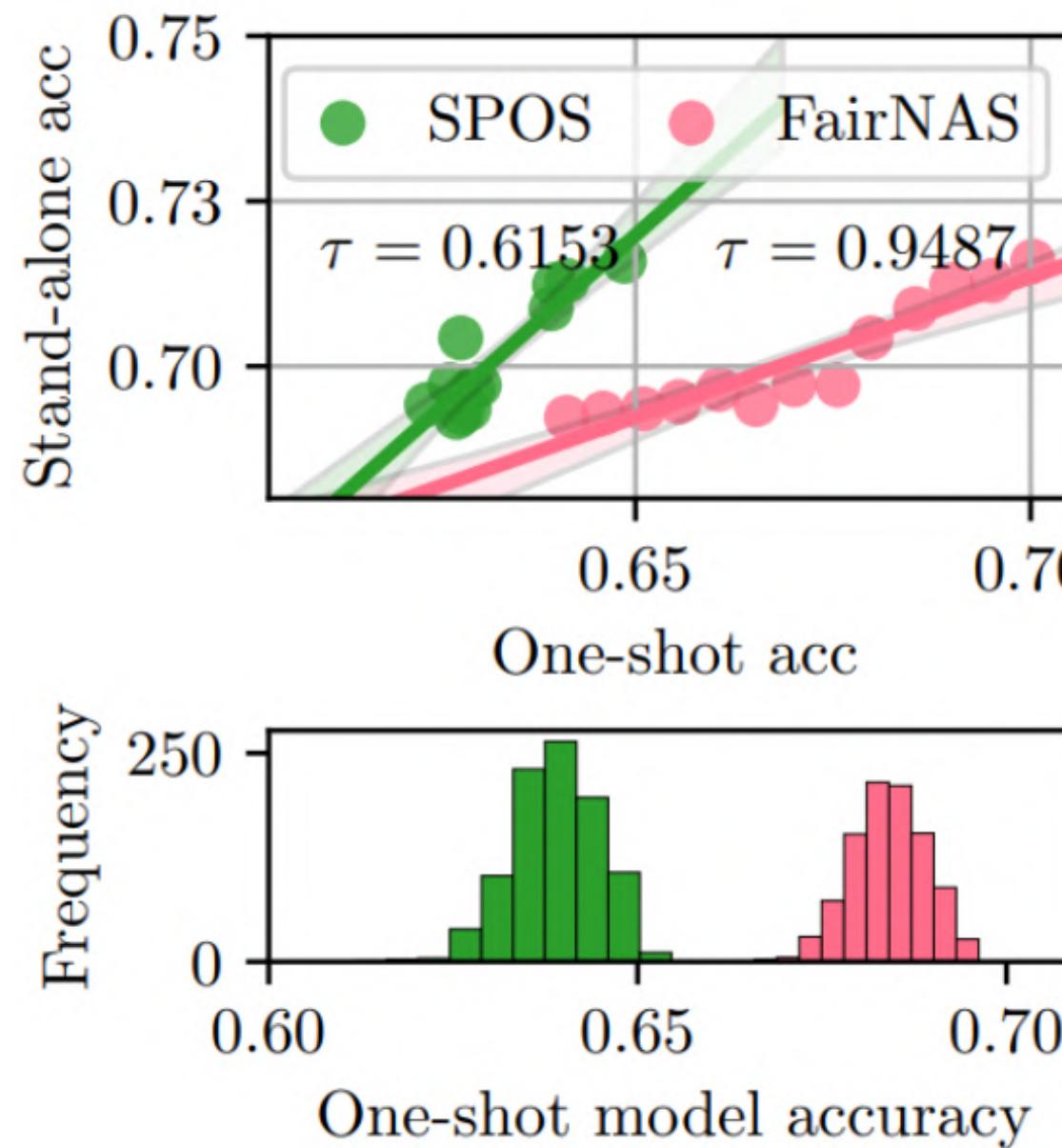
- 1 Не все слои при случайном семплировании выбираются одинаково часто
- 2 Значит не все операции учатся «одинаково»
- 3 Маленькие изменения могут привести к значительному ухудшению «качества» работы операции
- 4 Будут такие операции, которые хуже остальных, только потому что они выбрались меньше. При этом, выберись они больше, то вероятность семплирования у них могла быть выше

# Честное семплирование

- 1 Берем батч изображений
- 2 Строим всевозможные сети при помощи перестановок
- 3 Вычисляем градиенты на этом батче для каждой сетки
- 4 Накапливаем градиенты
- 5 Обновляем параметры суперсети



# Корреляции при честном сэмплировании



NAS Methods	Type	EF	SF
SMASH [2]	Hypernet	✗	✗
One-Shot [1]	Supernet	✗	✗
DARTS [26]	Gradient-based	✗	✗
FBNet [46]	Gradient-based	✗	✗
ProxylessNAS [4]	Gradient-based/RL	✗	✗
Single Path One-Shot [12]	Supernet+EA	✗	✗
Single-Path NAS [39]	Gradient-based	✓	✗
FairNAS (Ours)	Fair Supernet+EA	✓	✓

Так, а какие подводные камни?

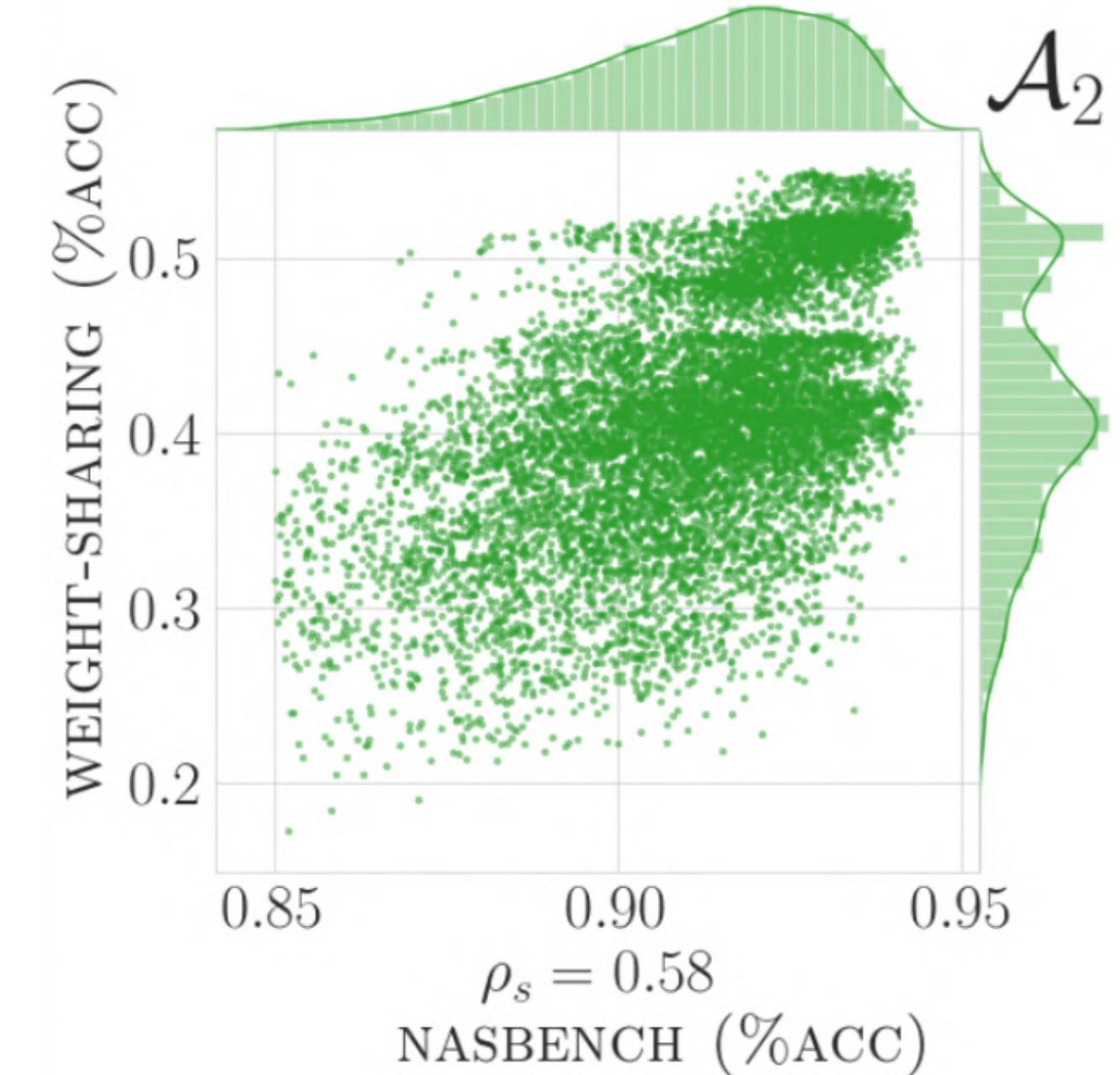
# Корреляции

## Проблема

→ А кто сказал, что используя одни и те же веса в разных нейросетях, мы получим такую же точность, что и при обучении «с нуля» ?

К сожалению, проблема выше присуща всем методам NAS которые используют weight sharing.

[arxiv.org/abs/2002.04289 ↗](https://arxiv.org/abs/2002.04289)



# Zero-shot NAS

# Основная идея

Проблема

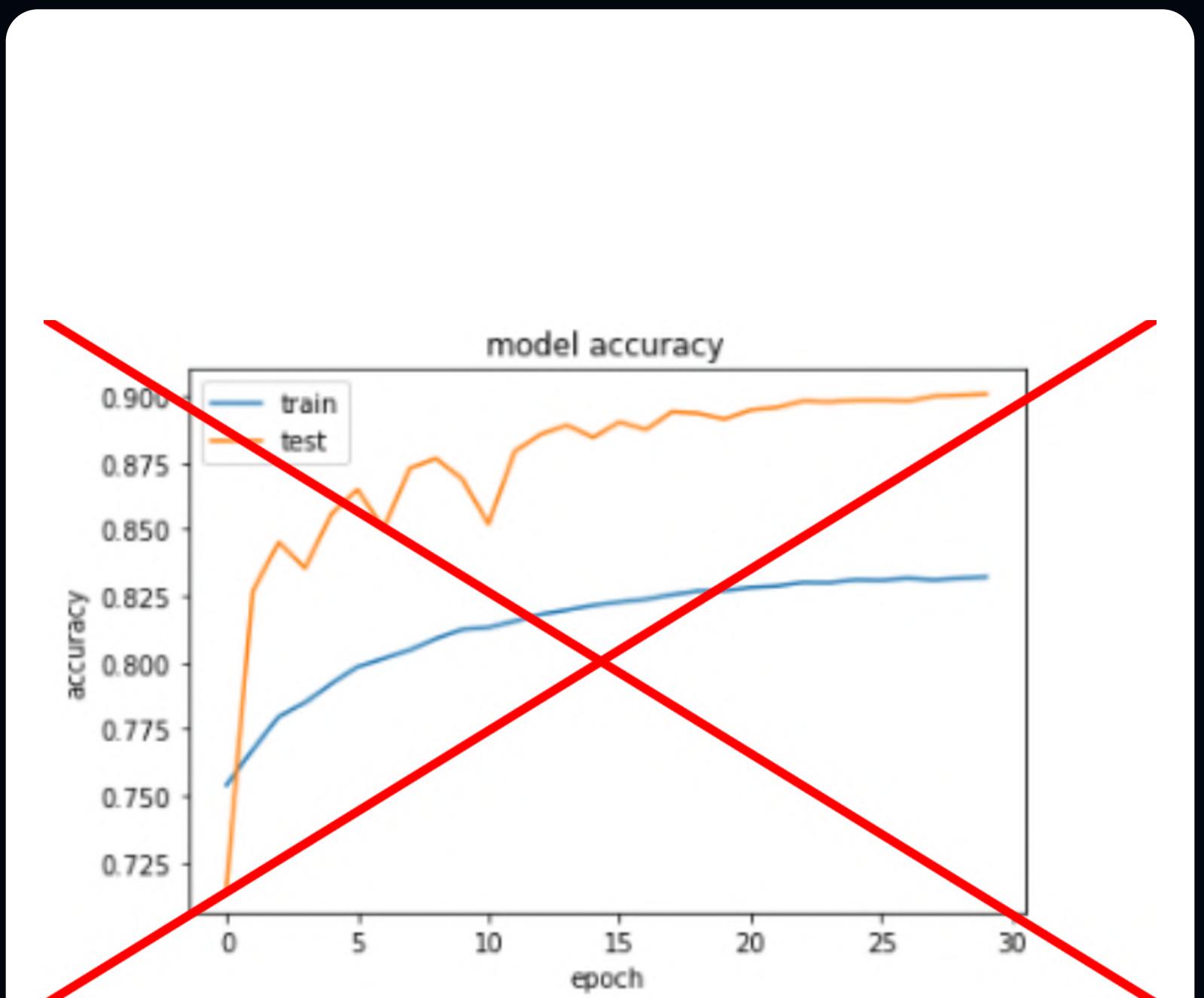
- Weight sharing подходы учатся дольше обучения одной сети. Нельзя ли как-то побыстрее?

Идея

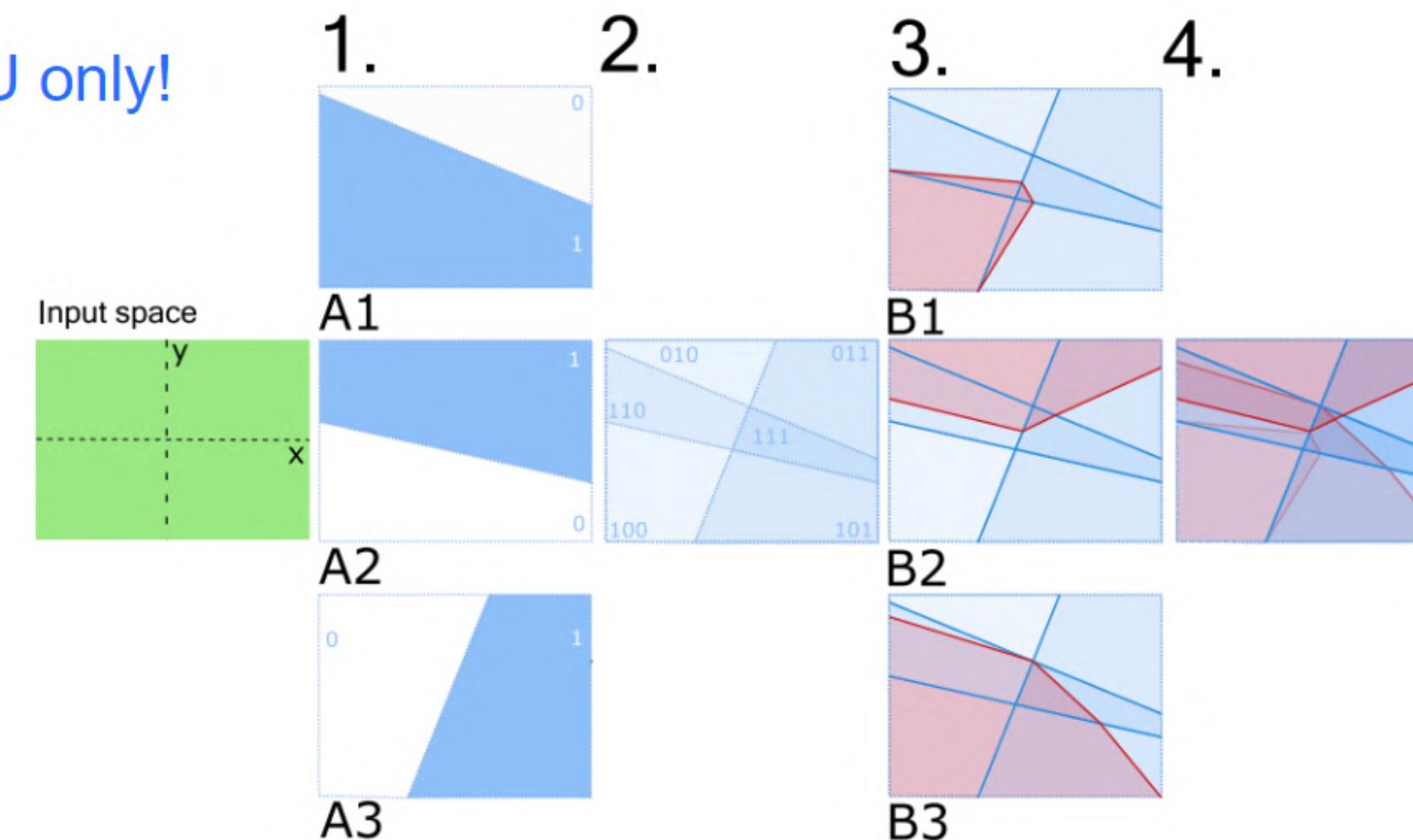
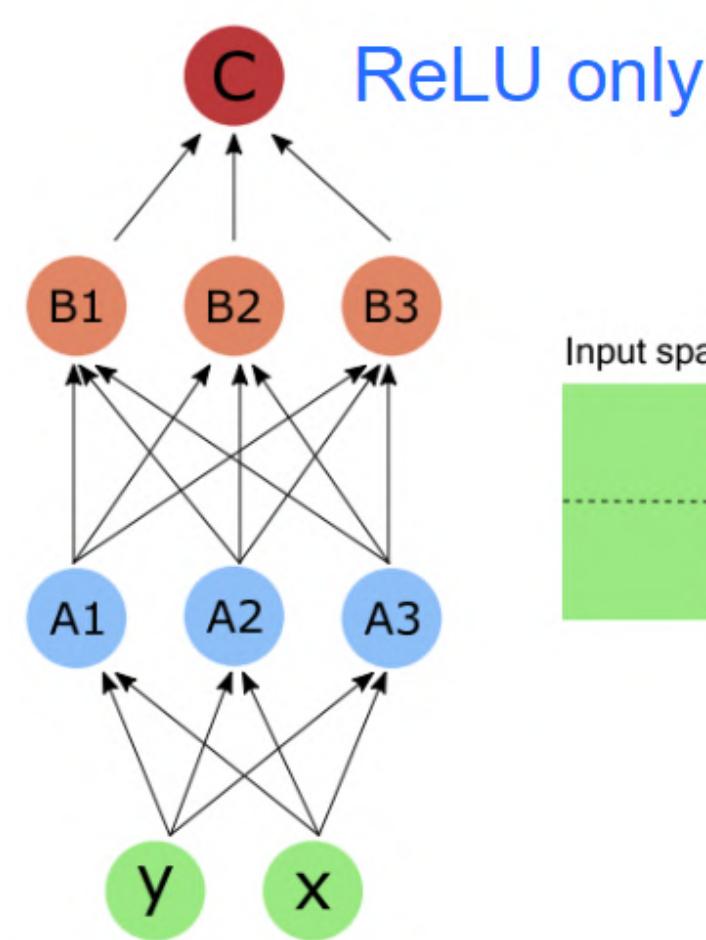
- Давайте попробуем оценить необученную архитектуру НС по каким-нибудь её характеристикам

Реализация

- ???



# Линейные регионы в ReLU сетях



# Паттерны активаций

## Идея

- Репрезентативная сила НС тем больше, чем более разнообразные паттерны активаций для разных входных данных

## Реализация

- Подадим в необученную нейросеть один минибатч данных и будем смотреть на активации отдельных нейронов

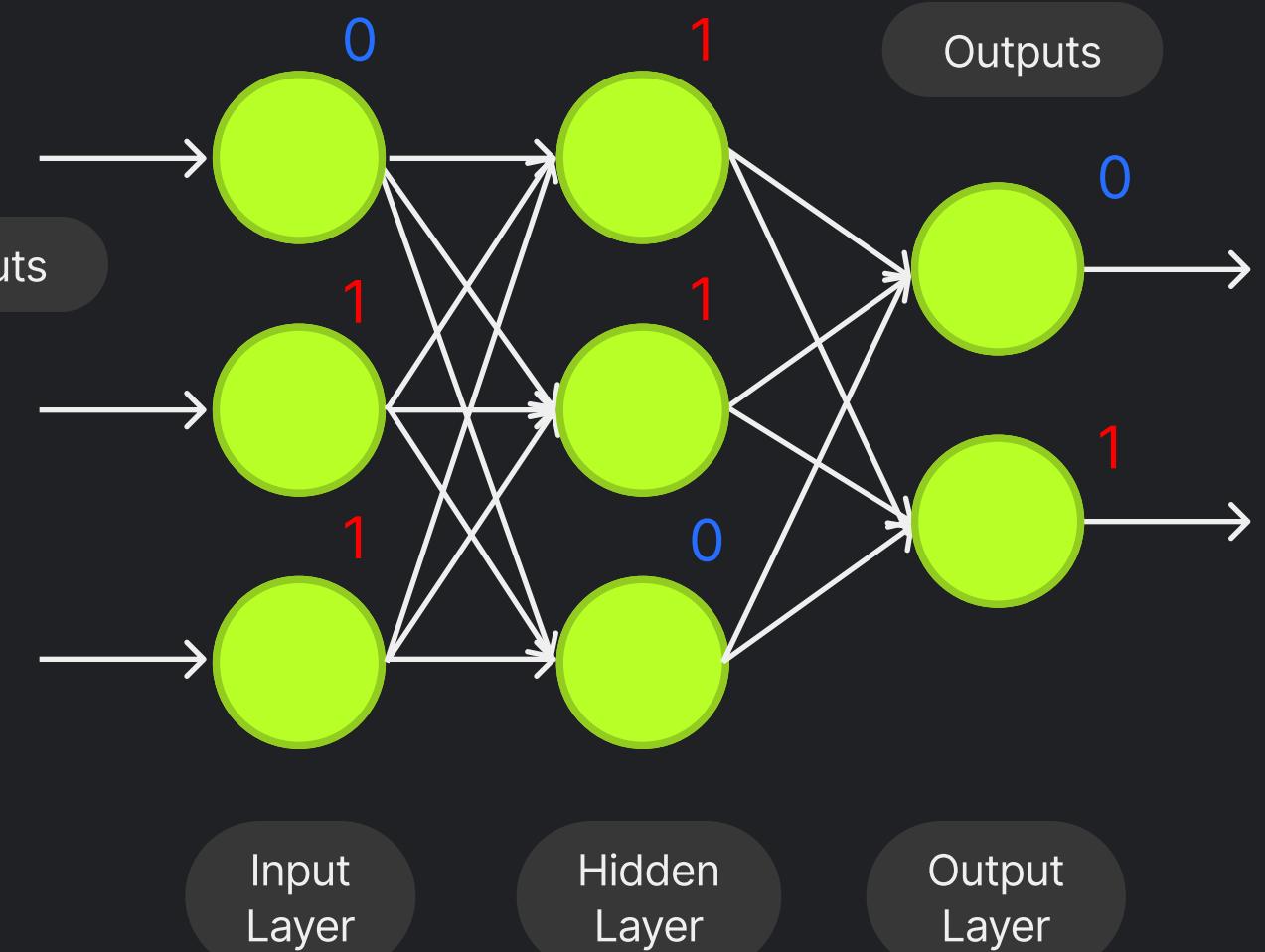
Расстояние между двумя паттернами – расстояние Хэмминга:

$$c_i = \underline{01111001}$$

$$c_j = \underline{11101101}$$

$$d_H = \text{sum}(\underline{10010100}) = 3$$

Паттерн ( $c_i$ ):  $\underline{01111001}$



# Метрика и результаты

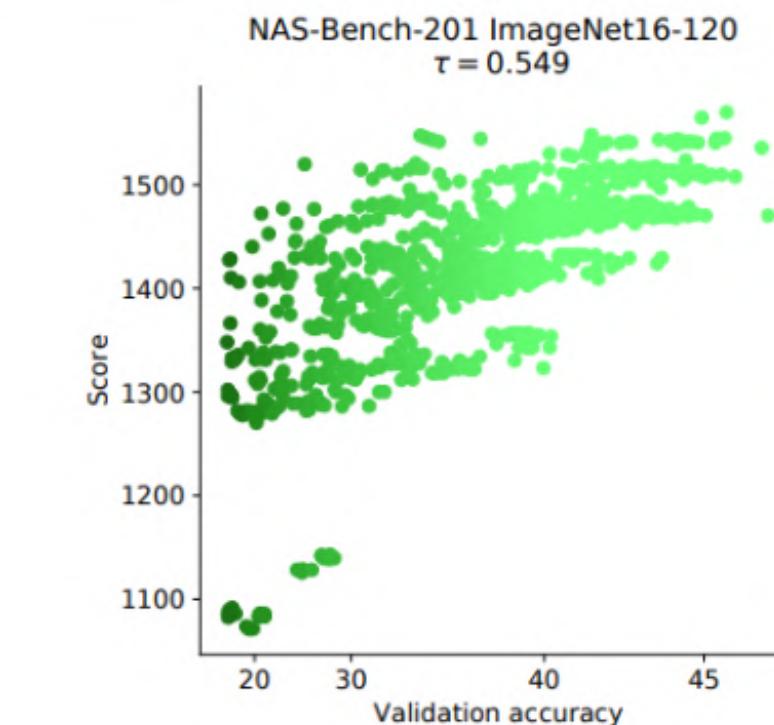
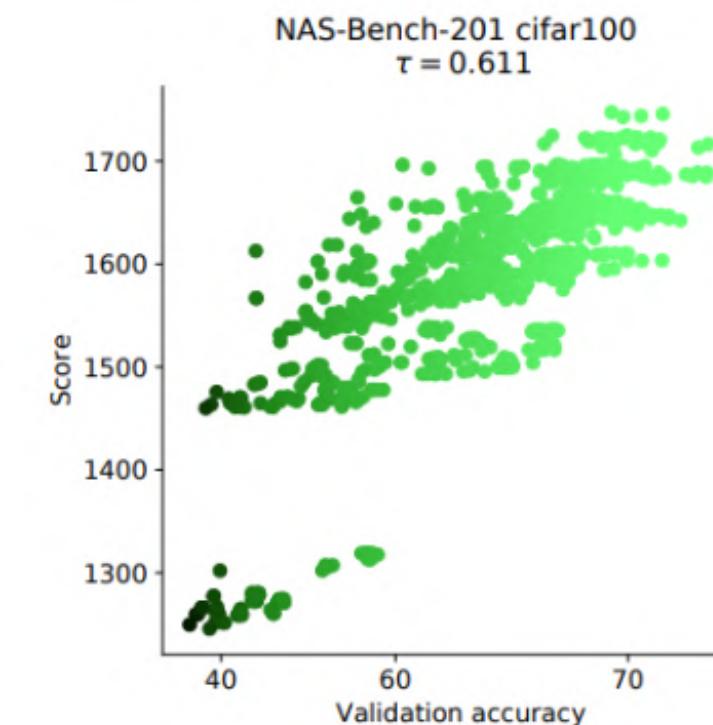
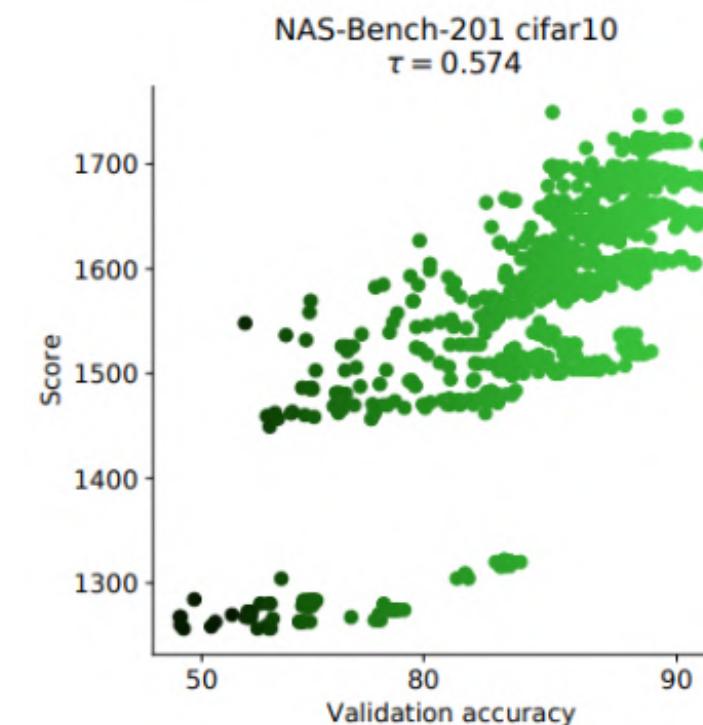
$N_A$  – длина паттерна

$N$  – число примеров в минибатче.

Метрика репрезентативности:

$$s = \log|\mathbf{K}_H|$$

$$\mathbf{K}_H = \begin{pmatrix} N_A - d_H(\mathbf{c}_1, \mathbf{c}_1) & \cdots & N_A - d_H(\mathbf{c}_1, \mathbf{c}_N) \\ \vdots & \ddots & \vdots \\ N_A - d_H(\mathbf{c}_N, \mathbf{c}_1) & \cdots & N_A - d_H(\mathbf{c}_N, \mathbf{c}_N) \end{pmatrix}$$



# Что такое NAS-Bench

---

## Проблема

→ В разных статьях используются разные пространства поиска, разные датасеты и гиперпараметры. Как сравнивать методы поиска между собой?

---

## Решение

---

→ Пусть кто-то создаст одно пространство поиска для всех и обучит все модели в нём

---

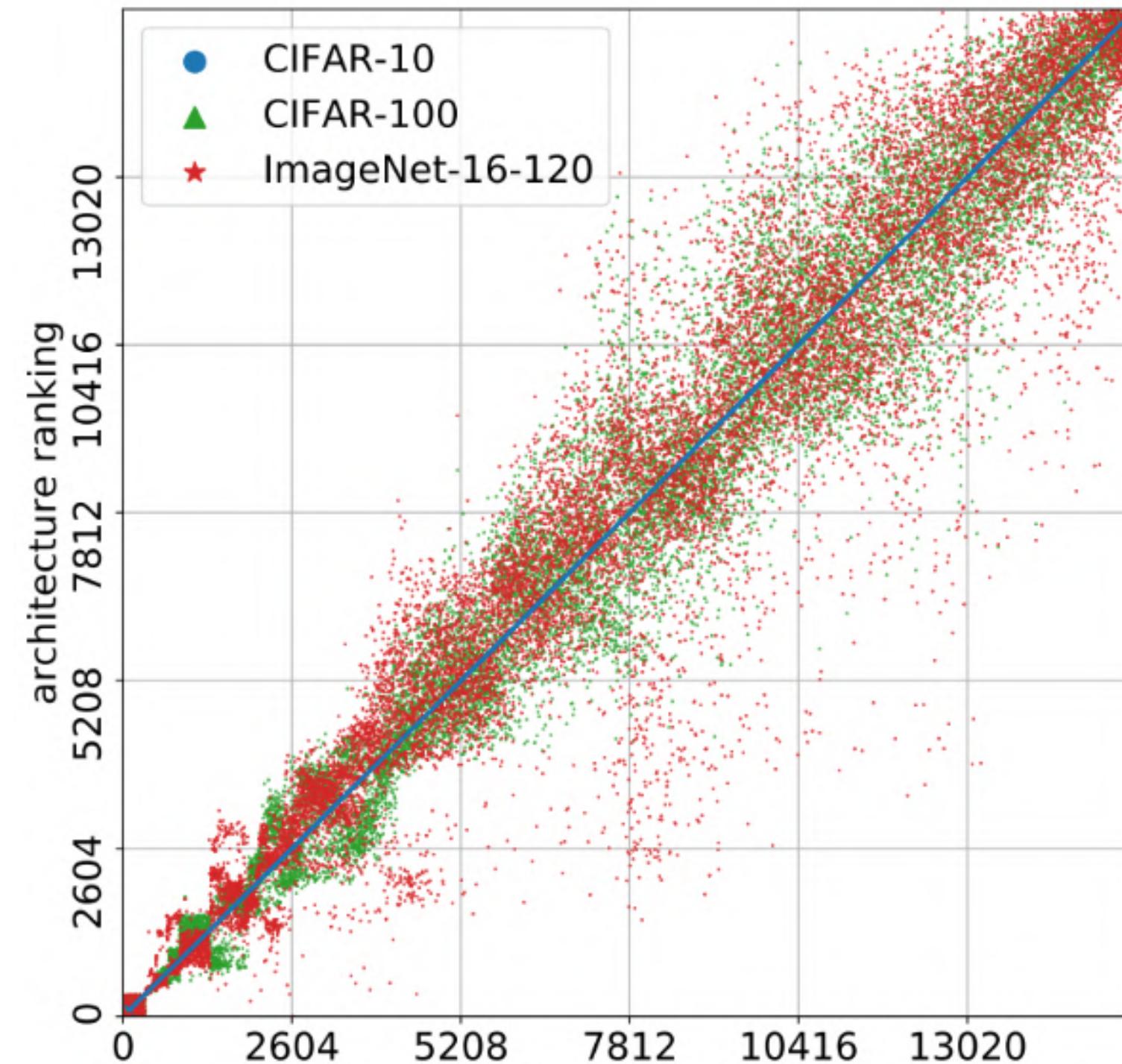
→ ???

---

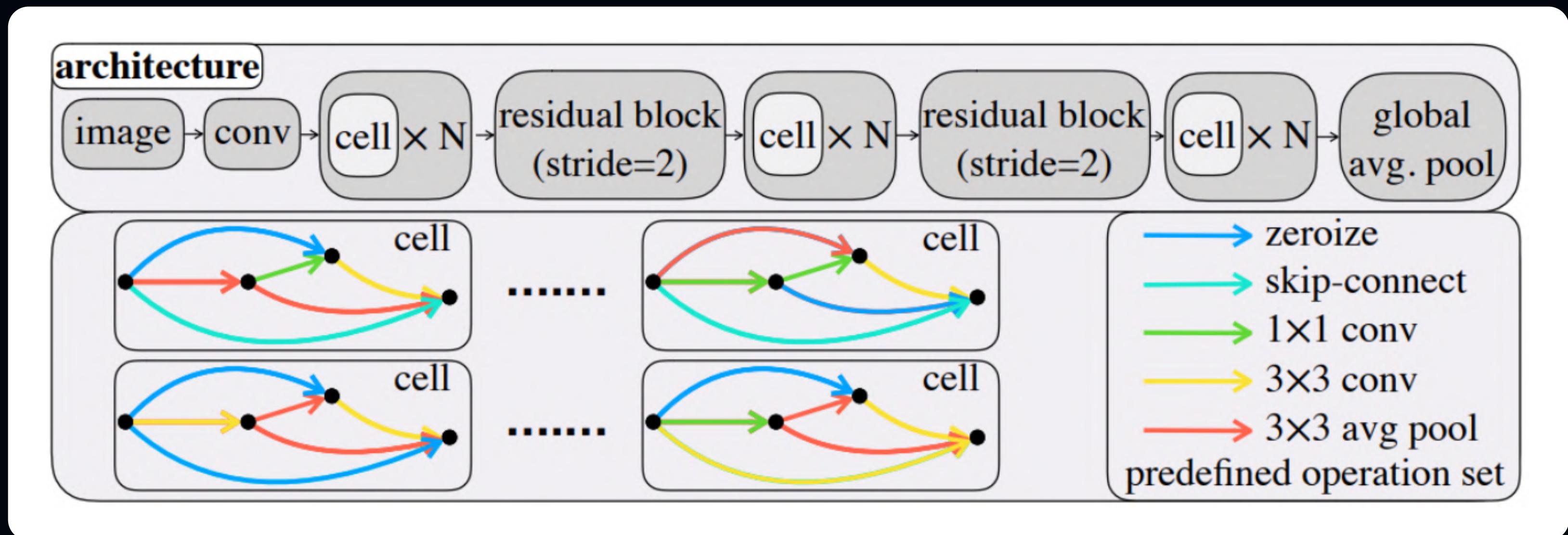
# Предисловие: NAS-Bench

NAS-Bench-201

Каждая точка — одна  
обученная с нуля  
модель. 15625 штук



# Пространство поиска в NAS-Bench-201



## Другие метрики

→ `grad_norm`:  $\text{sum}(p.\text{grad}.\text{norm}() \text{ for } p \text{ in parameters})$

→ `snip`:  $\text{sum}((p.\text{grad} * p).\text{abs}().\text{sum}() \text{ for } p \text{ in parameters})$

→ `fisher`:  $\text{sum}((a.\text{grad} * a).\text{pow}(2).\text{sum}() \text{ for } a \text{ in activations})$

→ `grasp`

→ `synflow`

→ `jacob_cov`

Table 1: Spearman  $\rho$  of zero-cost proxies on NAS-Bench-201.

Dataset	grad_norm	<u>snip</u>	grasp	fisher	<u>synflow</u>	<u>jacob_cov</u>	vote
CIFAR-10	0.58	0.58	0.48	0.36	0.74	0.73	0.82
CIFAR-100	0.64	0.63	0.54	0.39	0.76	0.71	0.83
ImageNet16-120	0.58	0.58	0.56	0.33	0.75	0.71	0.82

У многих методов хорошие  
корреляции. Но...

Есть ли метрики лучше?

# FLOPs & Params

Число операций и число параметров — лучшие предикторы точности.



# Применение

Если хочется что-нибудь попроще — можно поискать на [arxiv.org](https://arxiv.org/) ↗ современные Zero-Shot метрики. Неплохим кандидатом будет SWAP

## Личные рекомендации

### 1. Пространство поиска

зависит от baseline архитектуры, которую вы используете. Лучше всего перебирать небольшие модификации операций в вашей сети (4-10 операций).

Например, для ViT можно перебрать размерности query, key, value и число голов, а для MobileNetV2 – число каналов и размер ядра depthwise свёртки в бottлнеке

### 2. Метод поиска

random search (easy), DARTS (medium) или ProxylessNAS / NSGA-III (hard)

### 3. Оценка моделей

SPOS (easy) или FairNAS (harder)

# Ответы на вопросы



# Список литературы

## Обзор

---

1. Neural Architecture Search: Insights from 1000 Papers [arxiv.org/abs/2301.08727](https://arxiv.org/abs/2301.08727) ↗
- 

## Бенчмарки

---

1. NAS-Bench-101: Towards Reproducible Neural Architecture Search [arxiv.org/abs/1902.09635](https://arxiv.org/abs/1902.09635) ↗
  2. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search [arxiv.org/abs/2001.00326](https://arxiv.org/abs/2001.00326) ↗
-

## Weight sharing

---

1. To Share or Not To Share: An Extensive Appraisal of Weight-Sharing [arxiv.org/abs/2002.04289](https://arxiv.org/abs/2002.04289) ↗
  2. Efficient Neural Architecture Search via Parameter Sharing [arxiv.org/abs/1802.03268](https://arxiv.org/abs/1802.03268) ↗
  3. Single Path One-Shot Neural Architecture Search with Uniform Sampling [arxiv.org/abs/1904.00420](https://arxiv.org/abs/1904.00420) ↗
  4. FairNAS: Rethinking Evaluation Fairness of Weight Sharing Neural Architecture Search [arxiv.org/abs/1907.01845](https://arxiv.org/abs/1907.01845) ↗
  5. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [arxiv.org/abs/1812.00332](https://arxiv.org/abs/1812.00332) ↗
  6. Once-For-All: Train One Network and Specialize it for Efficient Deployment [arxiv.org/abs/1908.09791](https://arxiv.org/abs/1908.09791) ↗
  7. BigNAS: Scaling Up Neural Architecture Search with Big Single-Stage Models [arxiv.org/abs/2003.11142](https://arxiv.org/abs/2003.11142) ↗
-

## Zero-shot NAS

---

1. Neural Architecture Search without Training [arxiv.org/abs/2006.04647](https://arxiv.org/abs/2006.04647) ↗
  2. Zero-Cost Proxies for Lightweight NAS [arxiv.org/abs/2101.08134](https://arxiv.org/abs/2101.08134) ↗
  3. Evaluating Efficient Performance Estimators of Neural Architectures [arxiv.org/abs/2008.03064](https://arxiv.org/abs/2008.03064) ↗
  4. SWAP-NAS: Sample-Wise Activation Patterns for Ultra-fast NAS [arxiv.org/abs/2403.04161](https://arxiv.org/abs/2403.04161) ↗
-

# Дополнительно

# Как можно совместить pruning и NAS?

## Связь с пруингом

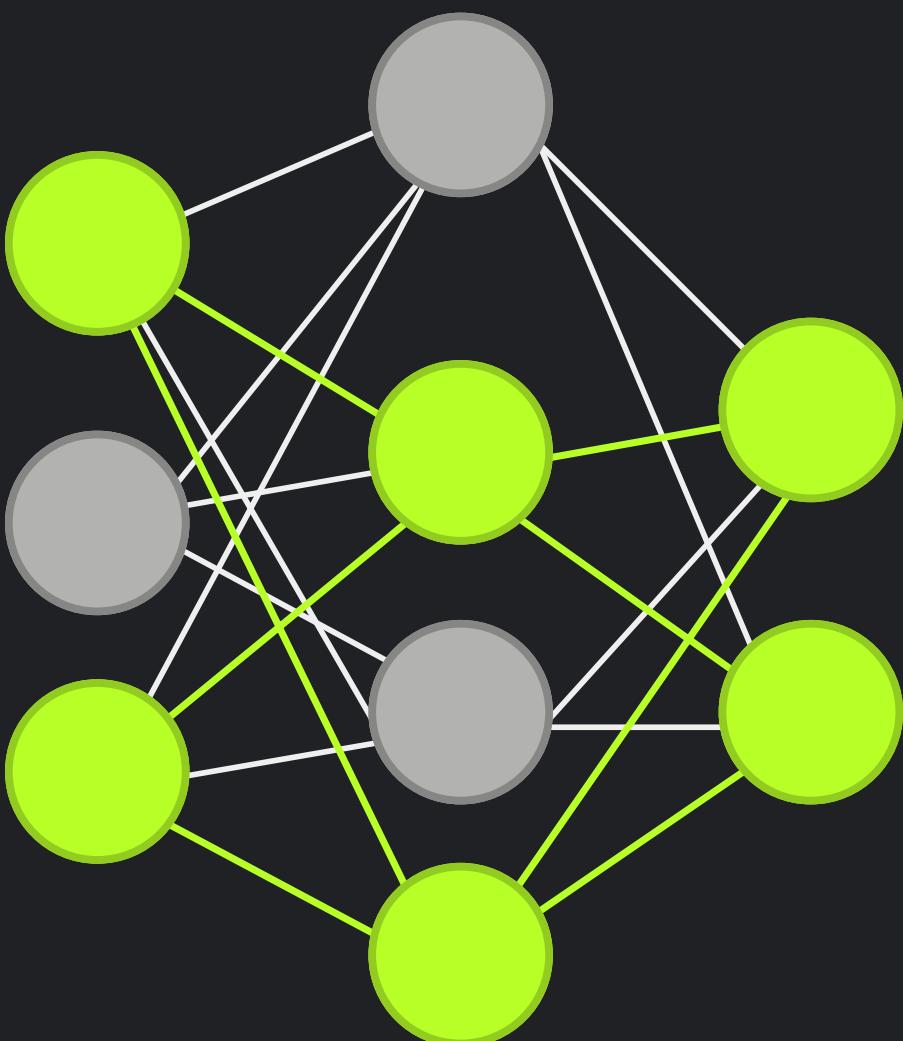
- 1 Давайте не будем перебирать размеры ядер
- 2 Обучим самую большую архитектуру ( $x2$ ) из пространства поиска
- 3 Любую другую архитектуру из пространства поиска можно получить с помощью пруинга

---

Проблема

→ Лучшая точность будет у оригинальной  $x2$  модели

---



	TNB101_MICRO-AUTOENC	TNB101_MACRO-OBJECT	TNB101_MACRO-JIGSAW	NB101-CF10	TNB101_MACRO-AUTOENC	TNB101_MACRO-SCENE	NB301-CF10	TNB101_MICRO-JIGSAW	TNB101_MACRO-OBJECT	NB201_MICRO-SCENE	NB201-IMGNT	NB201-CF10	NB201-CF100
plain	0.07	-0.19	-0.30	-0.32	-0.11	-0.19	-0.33	0.35	0.34	0.23	-0.24	-0.25	-0.19
grasp	-0.12	-0.64	-0.27	0.27	-0.02	-0.43	0.34	-0.13	-0.23	-0.27	0.54	0.51	0.53
num_lr	0.00	0.00	0.00	0.00	0.00	0.00	-0.02	0.00	0.00	0.00	0.07	0.07	0.07
fisher	-0.58	-0.30	-0.26	-0.28	-0.18	-0.12	-0.27	0.32	0.46	0.67	0.47	0.49	0.53
grad_norm	-0.32	-0.55	-0.27	-0.24	0.32	-0.34	-0.03	0.37	0.40	0.66	0.55	0.57	0.62
snip	-0.26	-0.37	-0.20	-0.19	0.20	-0.14	-0.04	0.43	0.47	0.71	0.55	0.58	0.62
epe_nas	0.00	-0.01	0.02	-0.00	0.00	-0.01	0.00	0.16	0.40	0.52	0.33	0.70	0.58
l2_norm	0.05	0.09	0.15	0.51	-0.19	0.29	0.46	0.35	0.33	0.53	0.68	0.68	0.71
zen	0.14	0.10	0.24	0.60	-0.01	0.28	0.44	0.52	0.55	0.72	0.38	0.33	0.34
jacov	0.18	0.07	0.19	-0.29	0.46	0.19	-0.05	0.56	0.51	0.75	0.70	0.74	0.70
params	-0.00	0.17	0.17	0.38	-0.18	0.33	0.46	0.44	0.46	0.63	0.69	0.72	0.73
synflow	0.00	0.12	0.34	0.31	0.00	0.28	0.18	0.48	0.49	0.72	0.75	0.73	0.76
zico	0.13	0.04	0.08	0.37	0.07	0.23	0.53	0.52	0.50	0.70	0.77	0.76	0.79
flops	-0.01	0.79	0.64	0.37	0.76	0.85	0.43	0.45	0.46	0.65	0.67	0.69	0.71
nwot	0.03	0.84	0.76	0.32	0.66	0.89	0.48	0.43	0.40	0.60	0.77	0.77	0.80
<b>swap</b>	-0.09	0.86	0.77	0.47	0.68	0.83	0.58	0.43	0.41	0.57	0.76	0.79	0.81
<b>reg_swap</b>	-0.00	0.86	0.77	0.75	0.68	0.83	0.63	0.47	0.48	0.67	0.87	0.88	0.90

# Классификация методов поиска

Оптимизационная задача:  $q = f(s)$ , где  $s$  — архитектура из пространства поиска,  $f$  — функция оценки качества (чёрный ящик),  $q$  — скалярная величина, которую нужно максимизировать.

## Black box optimization

---

Методы оптимизации вида «чёрный ящик» обычно не используют никакие свойства функции  $f$ .

## Gradient-based search

---

Данные методы используют градиентный спуск для оптимизации параметров сэмплирования архитектур из пространства поиска:  $s \sim p(\theta)$ . Таким образом используется дифференцируемость  $f$