



Факультет  
компьютерных наук

Образовательная  
программа ПМИ

# Optimizing neural networks

Подготовили студенты:

Пономарева Ольга  
Теняев Александр  
Алимханов Карим  
Бабаев Минходж  
Солодовников Михаил  
Некрасов Артём

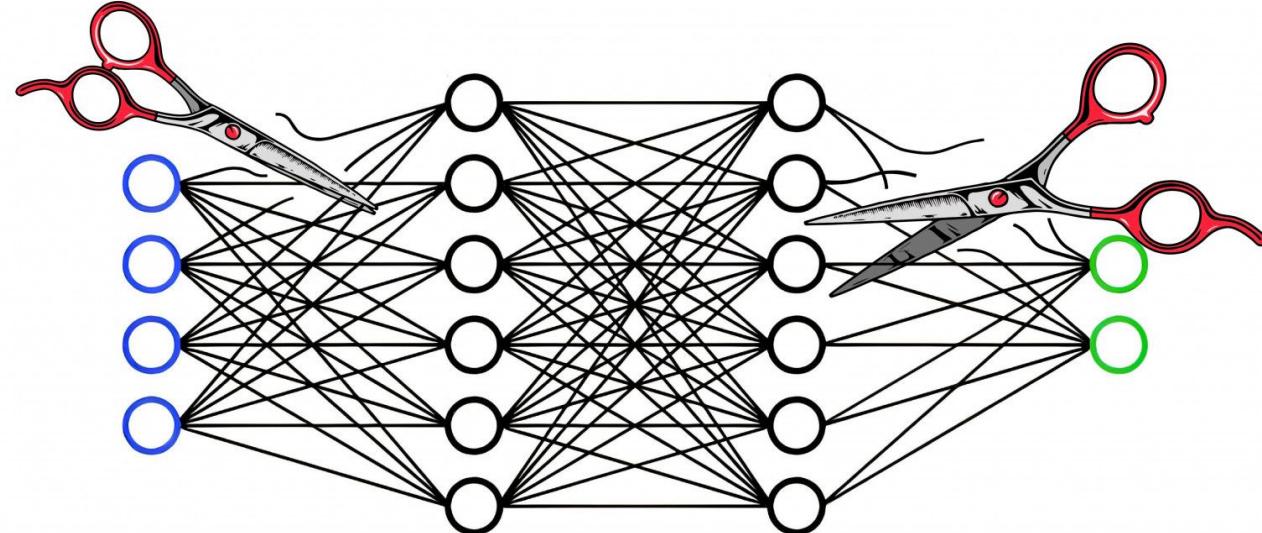


- 1. Общие методы**
- 2. Transformer**
- 3. Flash Attention**
- 4. Техники распределенного обучения**
- 5. Дополнительная статья**

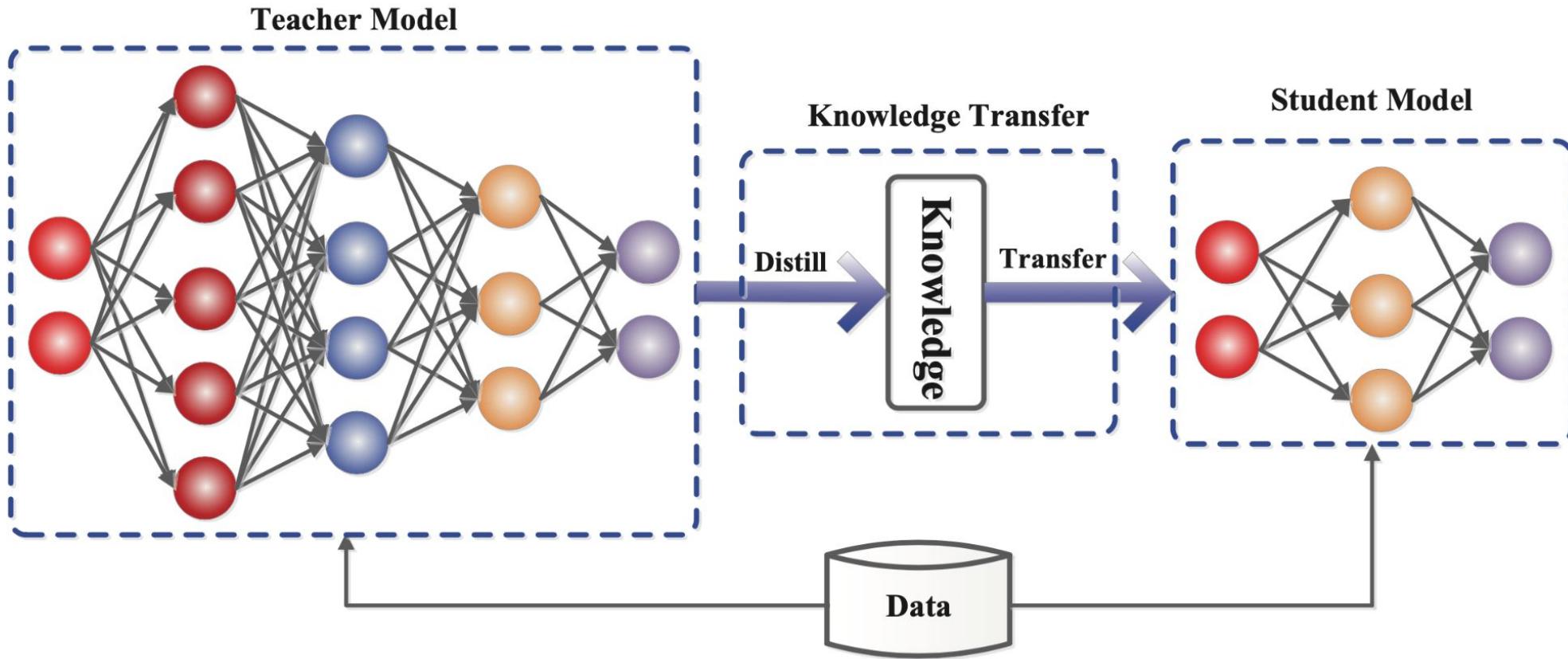


О Р Т У Н А

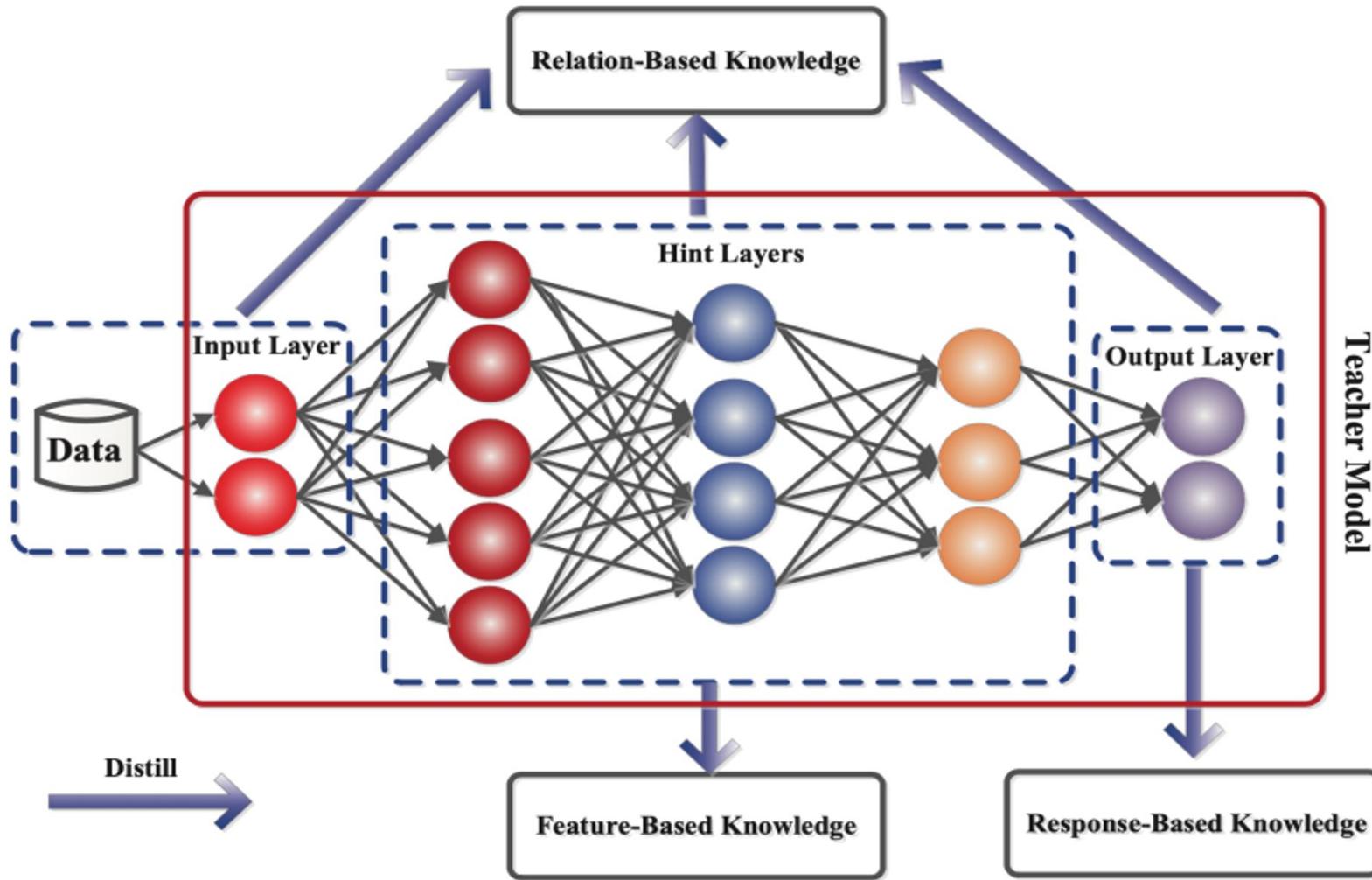
- Knowledge Distillation
- Optuna
- Pruning



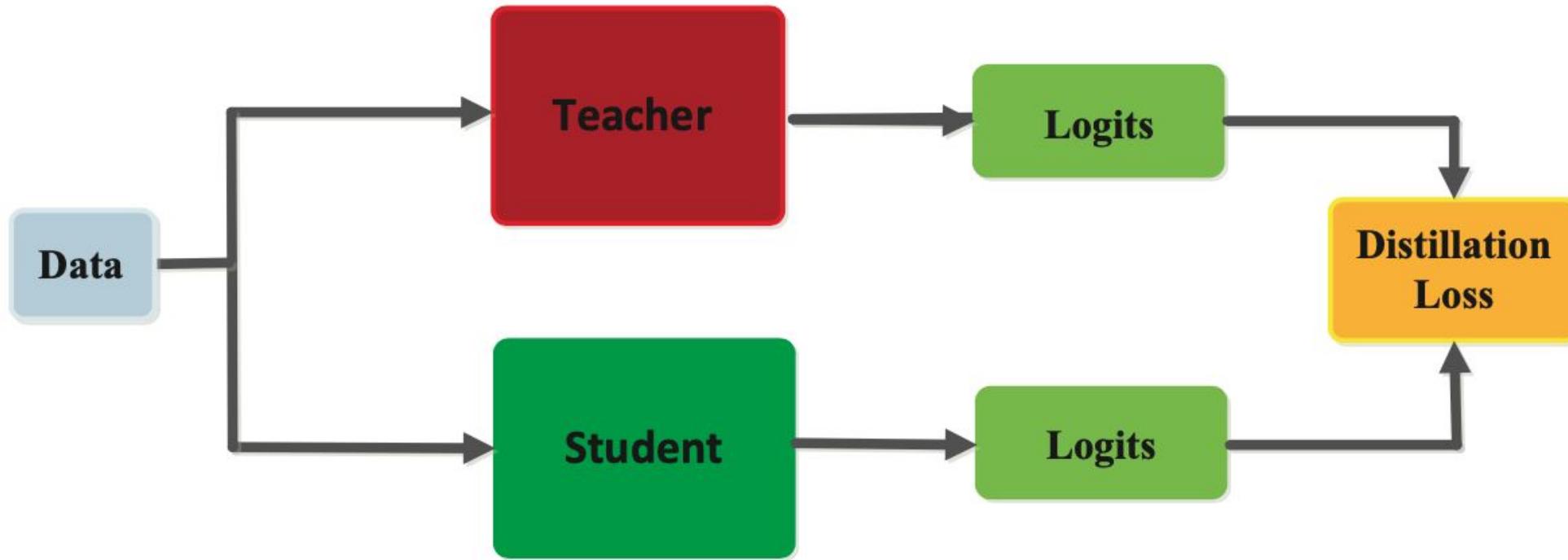
# Knowledge distillation



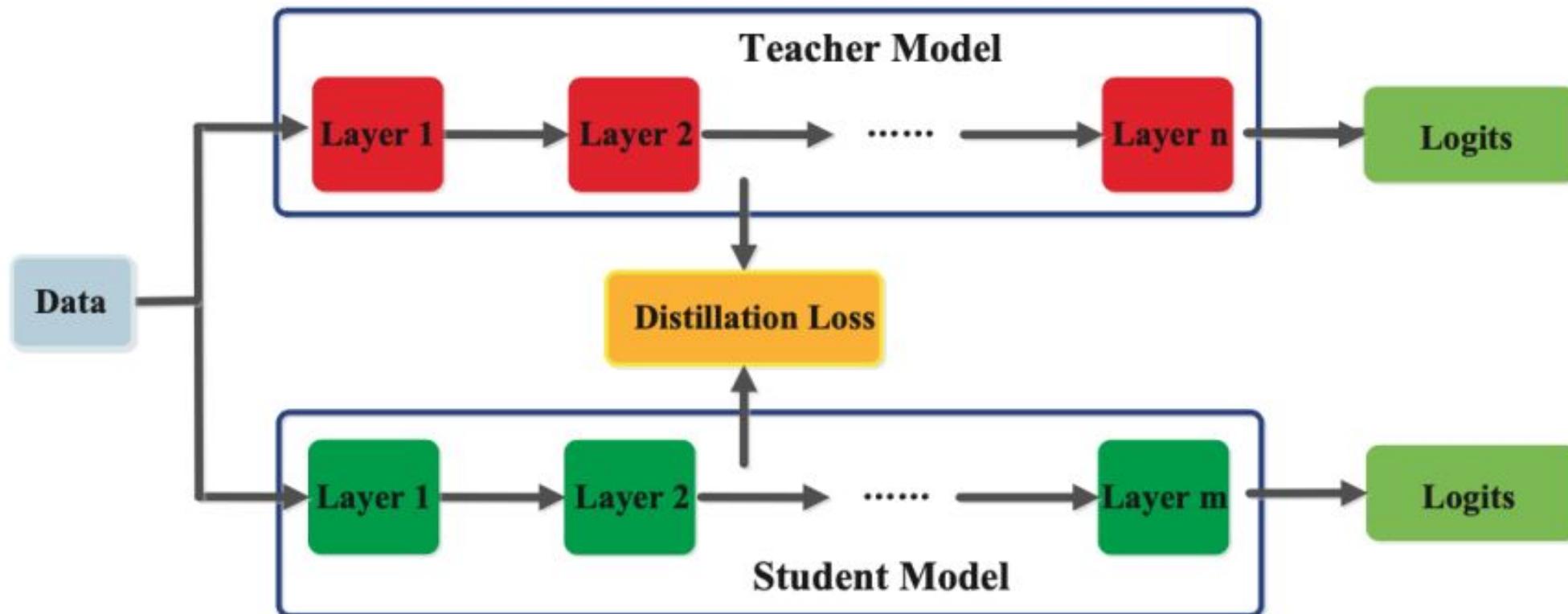
# Knowledge distillation



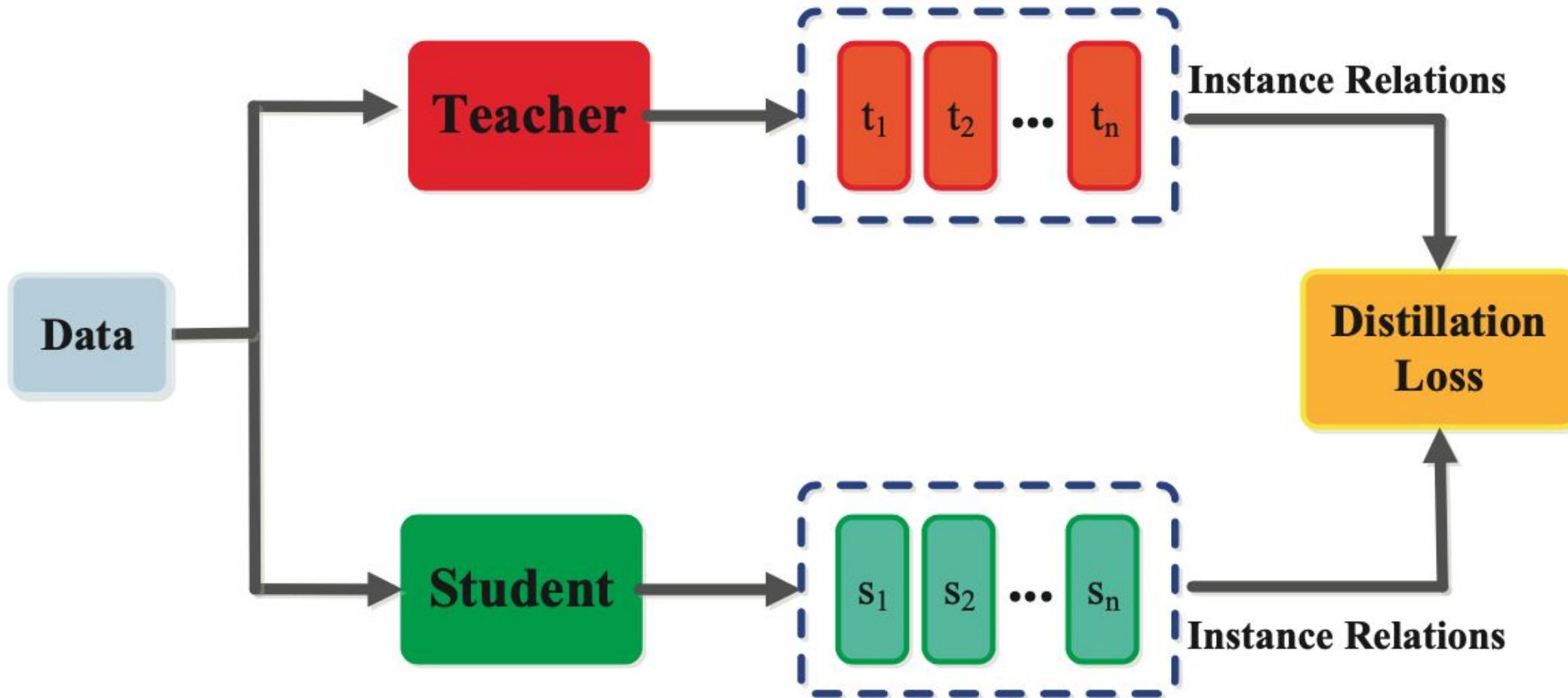
# Response-based knowledge



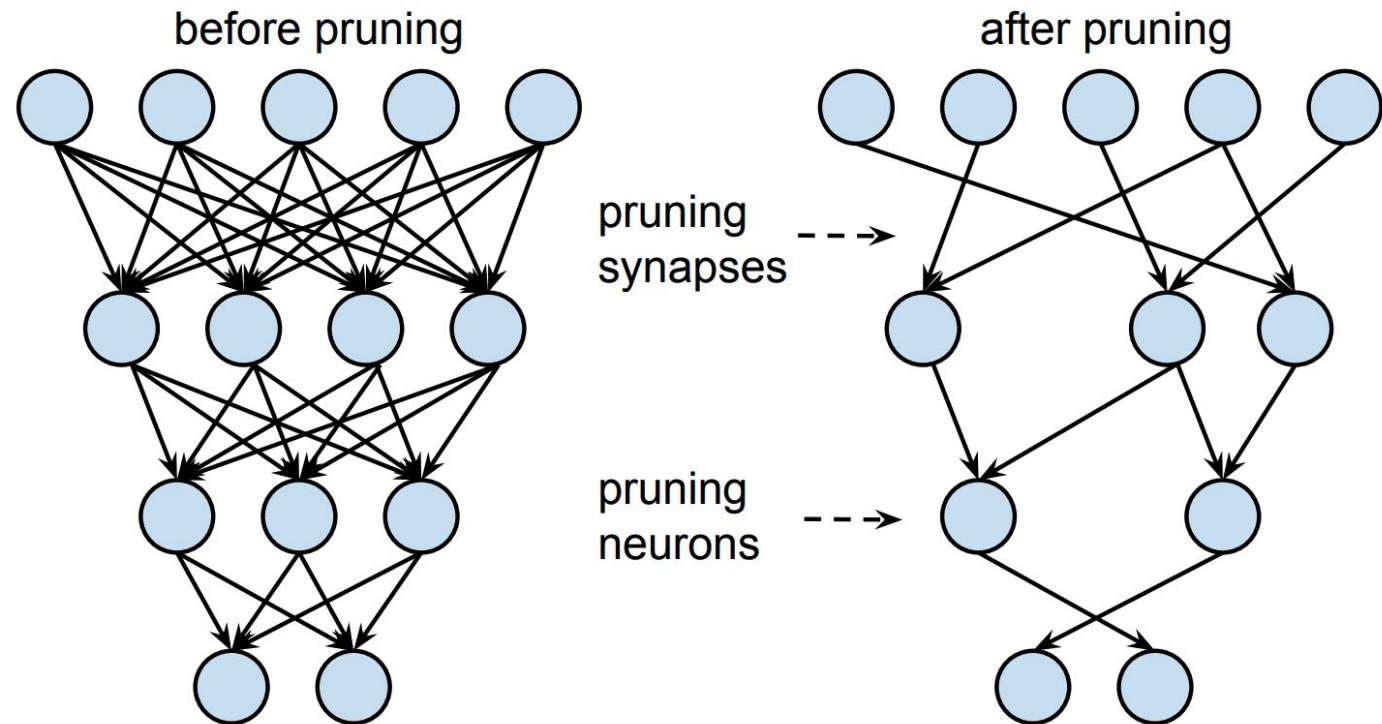
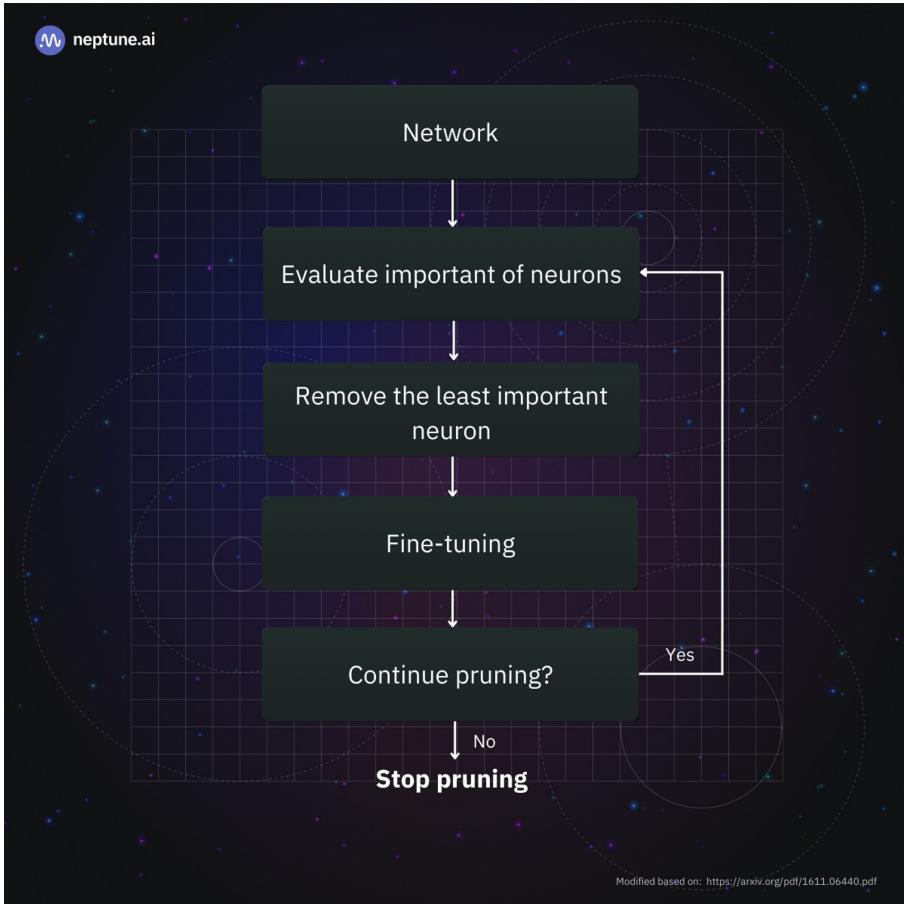
# Feature-based knowledge



# Relation-based knowledge

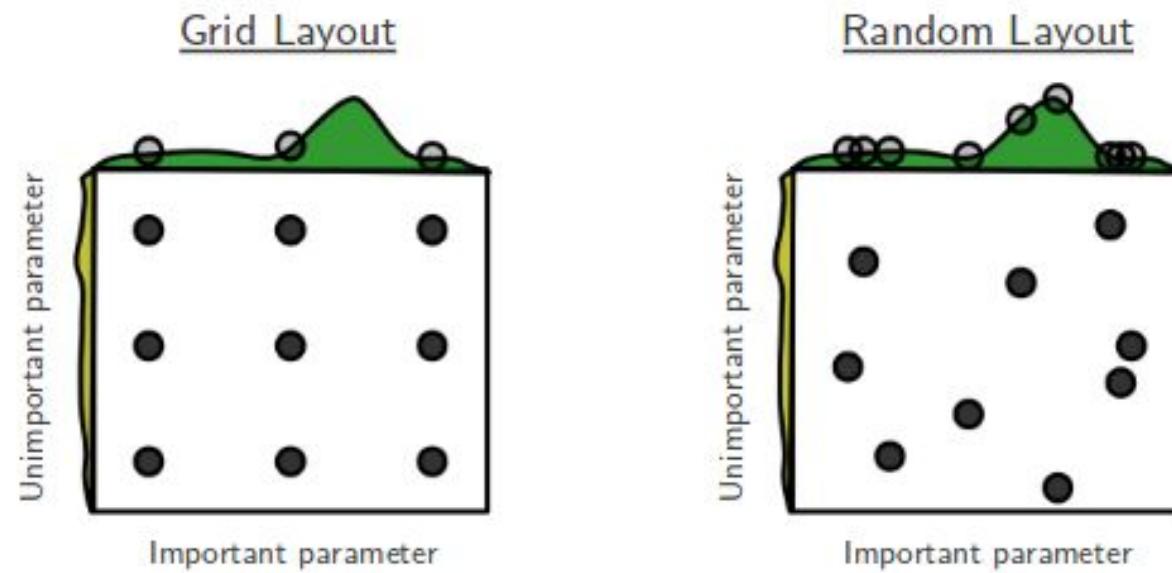


# Pruning



**Hyper-parameter search** is a part of almost every machine learning and deep learning project. When you select a candidate model, you make sure that it generalizes to your test data in the best way possible.

- Number of layers
- Units per layer
- Regularization strength
- Activation function
- Learning rate
- Optimizer parameters





```
def create_model(trial):
    model_type = trial.suggest_categorical('model_type', ['logistic-regression', 'decision-tree', 'svm'])

    if model_type == 'svm':
        kernel = trial.suggest_categorical('kernel', ['linear', 'poly', 'rbf', 'sigmoid'])
        regularization = trial.suggest_uniform('svm-regularization', 0.01, 10)
        degree = trial.suggest_discrete_uniform('degree', 1, 5, 1)
        model = SVC(kernel=kernel, C=regularization, degree=degree)

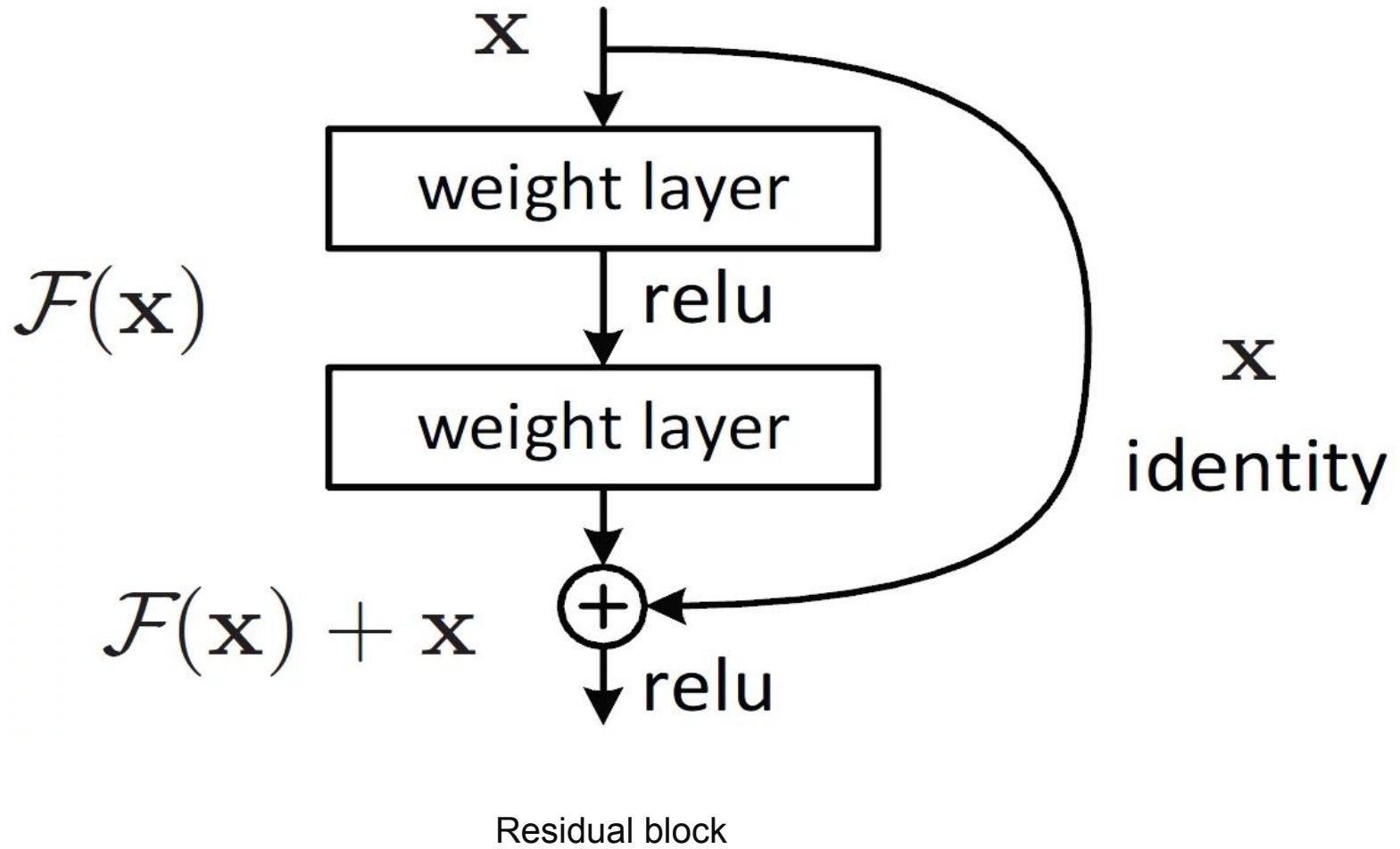
    if model_type == 'logistic-regression':
        penalty = trial.suggest_categorical('penalty', ['l2', 'l1'])
        if penalty == 'l1':
            solver = 'saga'
        else:
            solver = 'lbfgs'
        regularization = trial.suggest_uniform('logistic-regularization', 0.01, 10)
        model = LogisticRegression(penalty=penalty, C=regularization, solver=solver)

    if model_type == 'decision-tree':
        max_depth = trial.suggest_int('max_depth', 5, X_train.shape[1])
        min_samples_split = trial.suggest_int('min_samples_split', 2, 20)
        min_samples_leaf = trial.suggest_int('min_samples_leaf', 2, 20)
        model = DecisionTreeClassifier(
            max_depth=max_depth, min_samples_split=min_samples_split, min_samples_leaf=min_samples_leaf
        )
    return model

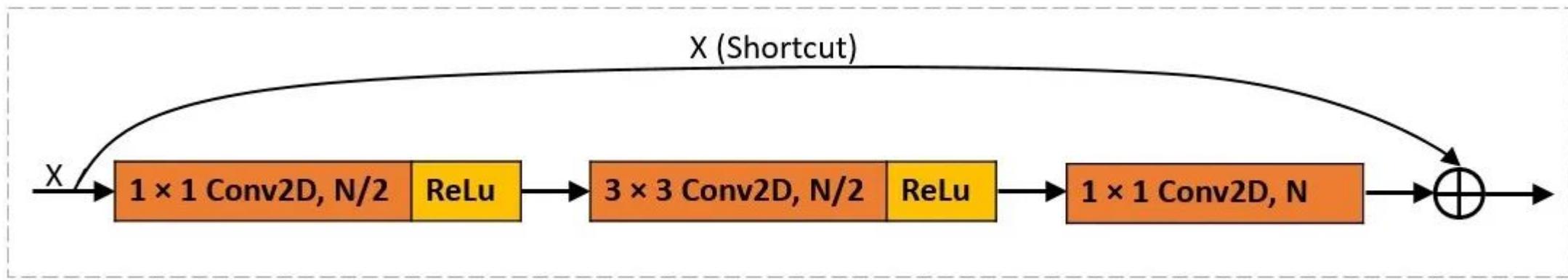
def objective(trial):
    model = create_model(trial)
    model.fit(X_train, y_train)
    return model_performance(model)

study = optuna.create_study(direction='maximize', study_name="starter-experiment")
study.optimize(objective, n_trials=300)
```

# Bottleneck

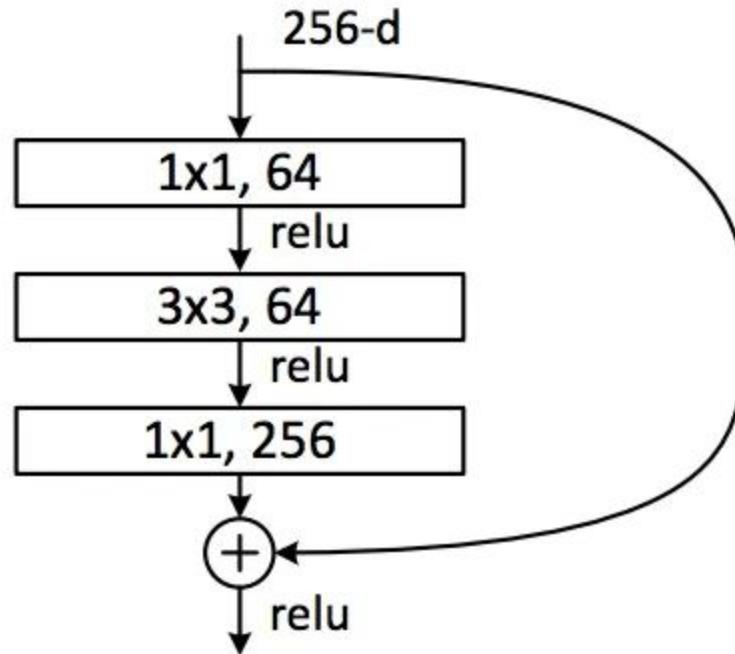
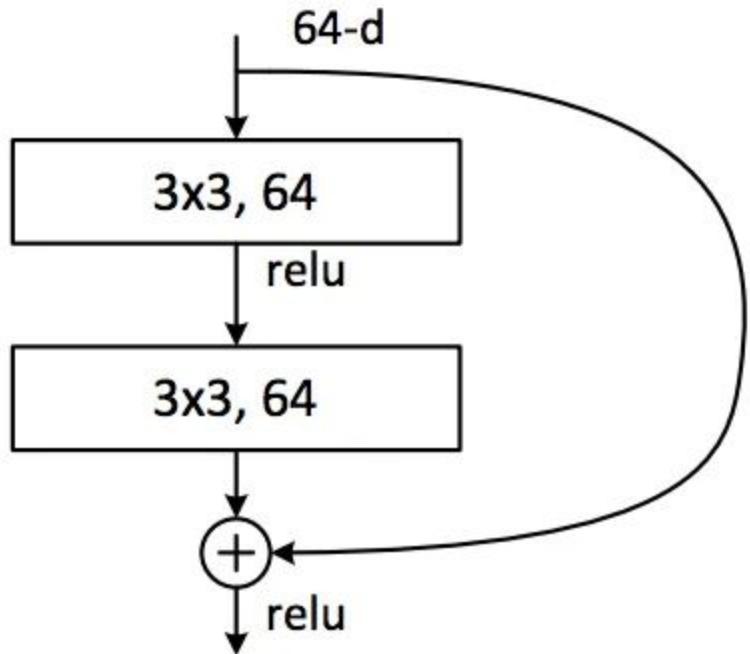


# Bottleneck



BottleNeck Block

# Bottleneck



Basic block on the left. BottleNeck on the right.

# BotNet: Bottleneck Transformers

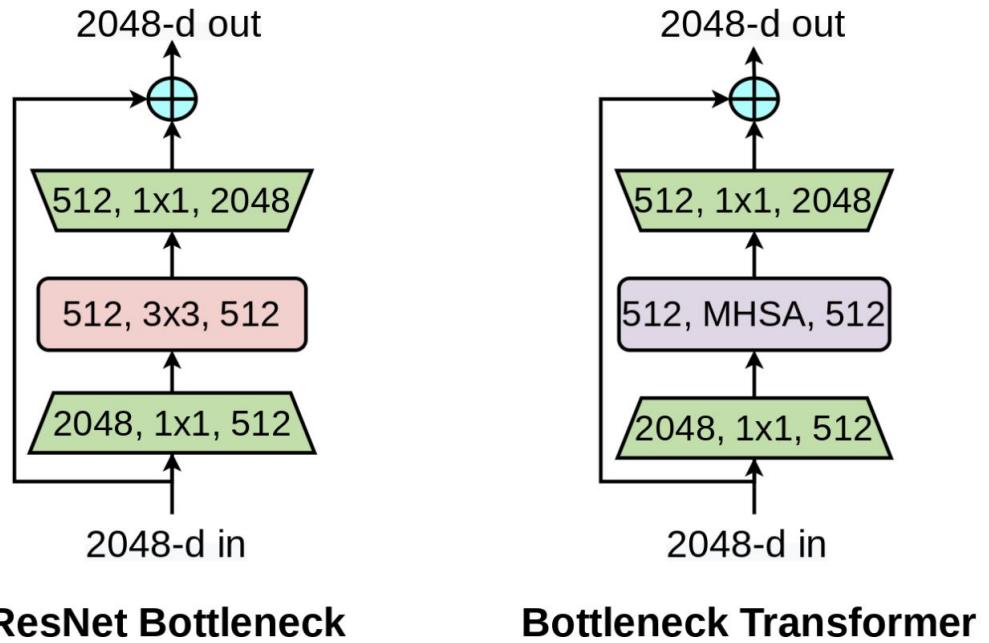
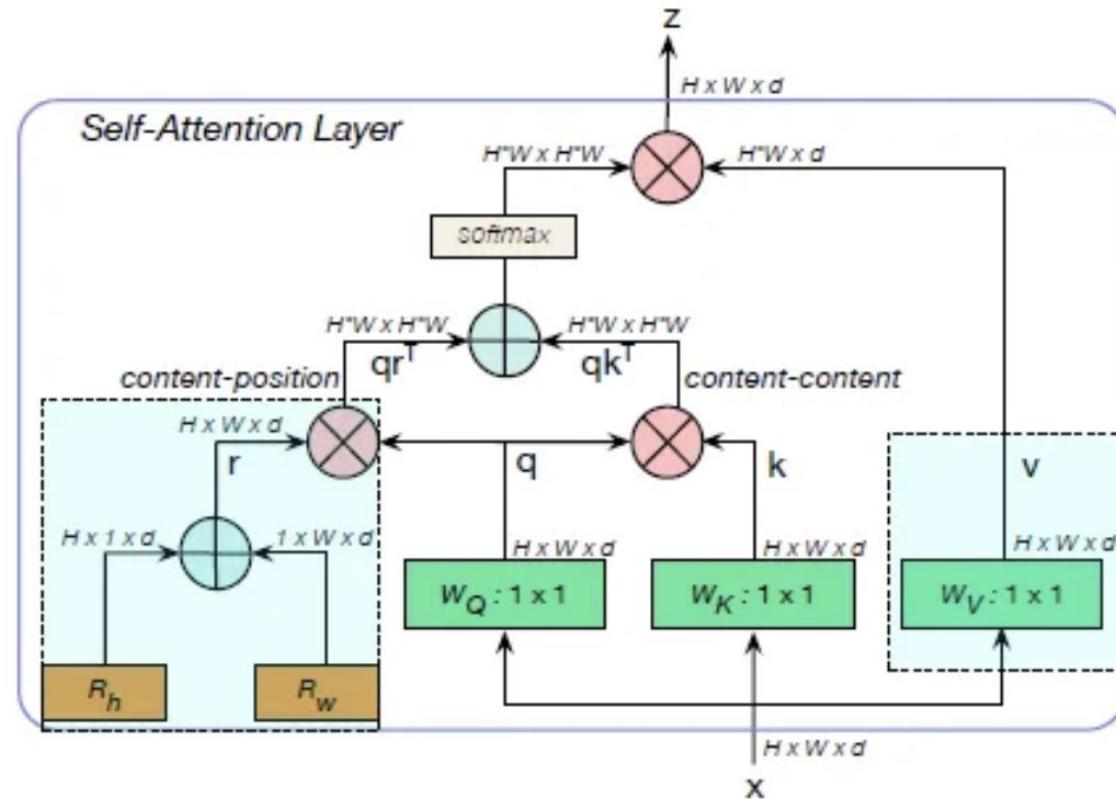


Figure 1: **Left:** A ResNet Bottleneck Block, **Right:** A Bottleneck Transformer (BoT) block. The only difference is the replacement of the spatial  $3 \times 3$  convolution layer with Multi-Head Self-Attention (MHSA). The structure of the self-attention layer is described in Figure 4.



Multi-Head Self-Attention (MHSA) layer used in the BoT block.

# BotNet: Bottleneck Transformers

stage	output	ResNet-50	BoTNet-50
c1	$512 \times 512$	$7 \times 7, 64$ , stride 2	$7 \times 7, 64$ , stride 2
c2	$256 \times 256$	$3 \times 3$ max pool, stride 2	$3 \times 3$ max pool, stride 2
		$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
c3	$128 \times 128$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
c4	$64 \times 64$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$
c5	$32 \times 32$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ \text{MHSA, 512} \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
# params.		$25.5 \times 10^6$	$20.8 \times 10^6$
M.Adds		$85.4 \times 10^9$	$102.98 \times 10^9$
TPU steptime		786.5 ms	1032.66 ms

Architecture of BoTNet-50 (BoT50).

Backbone	top-1 acc.	top-5 acc.
R50	77.7	93.9
BoT50	78.3 (+ 0.6)	94.2 (+ 0.3)

ImageNet results in an improved training setting.

# Training details

- Loss - standard NLL (cross-entropy) - lower is better:

$$NLL(y_{1:M}) = - \sum_{t=1}^M \log p(y_t | t_{t-1})$$

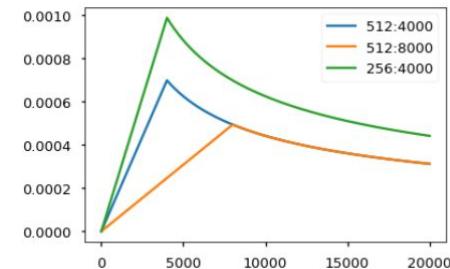
Instead perplexity is usually reported - higher is better:

$$\text{Perplexity}(y_{1:M}) = 2^{\frac{1}{M} NLL(y_{1:M})}$$

- Teacher forcing for decoder
- Adam optimizer
- Learning rate schedule with warm-up:

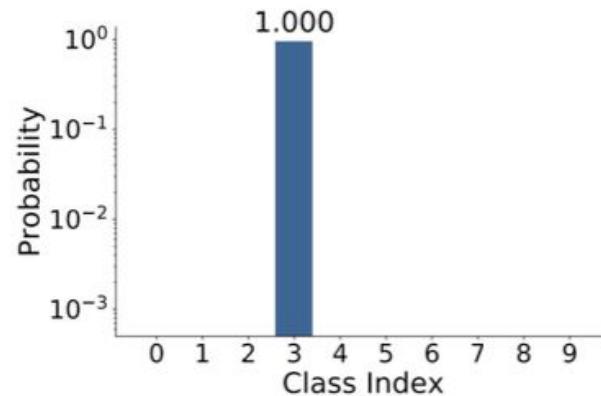
$$lr = d_{model}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup^{-1.5})$$

lr schedule

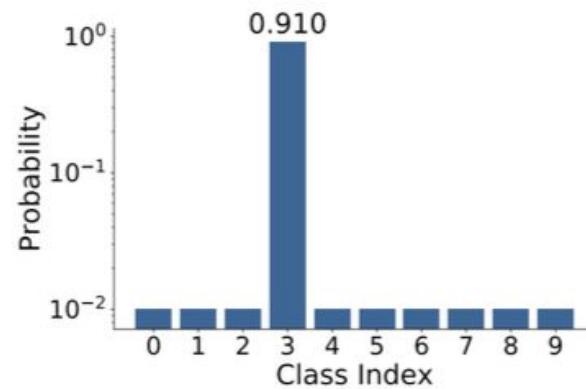


# Training details

## label smoothing



(a) Hard Label



(b) LS



# Breaking news

21

openai/**transformer-debugger**



7  
Contributors

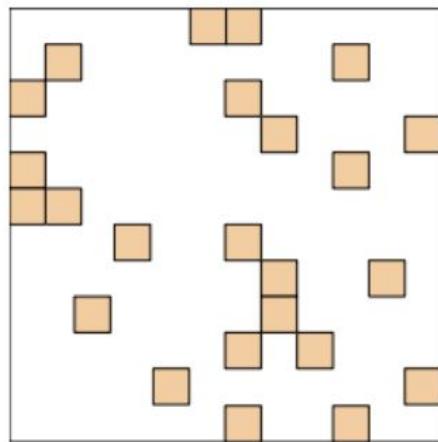
0  
Issues

3k  
Stars

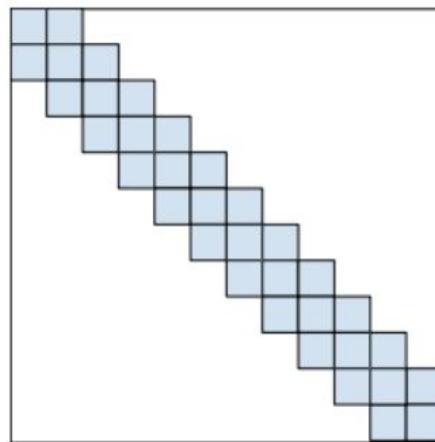
158  
Forks



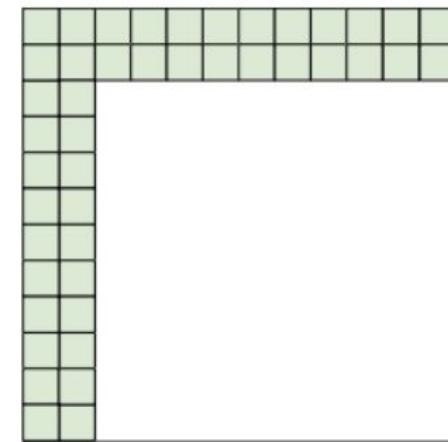
Варианты решения проблемы с расходом памяти и временем обучения



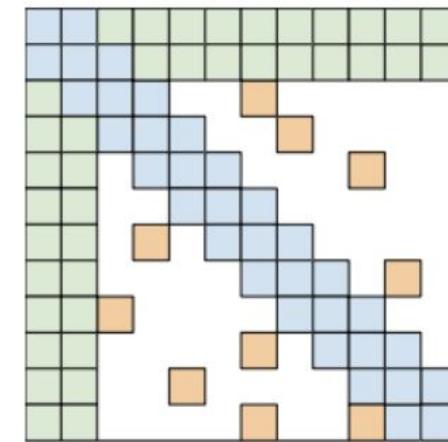
(a) Random attention



(b) Window attention



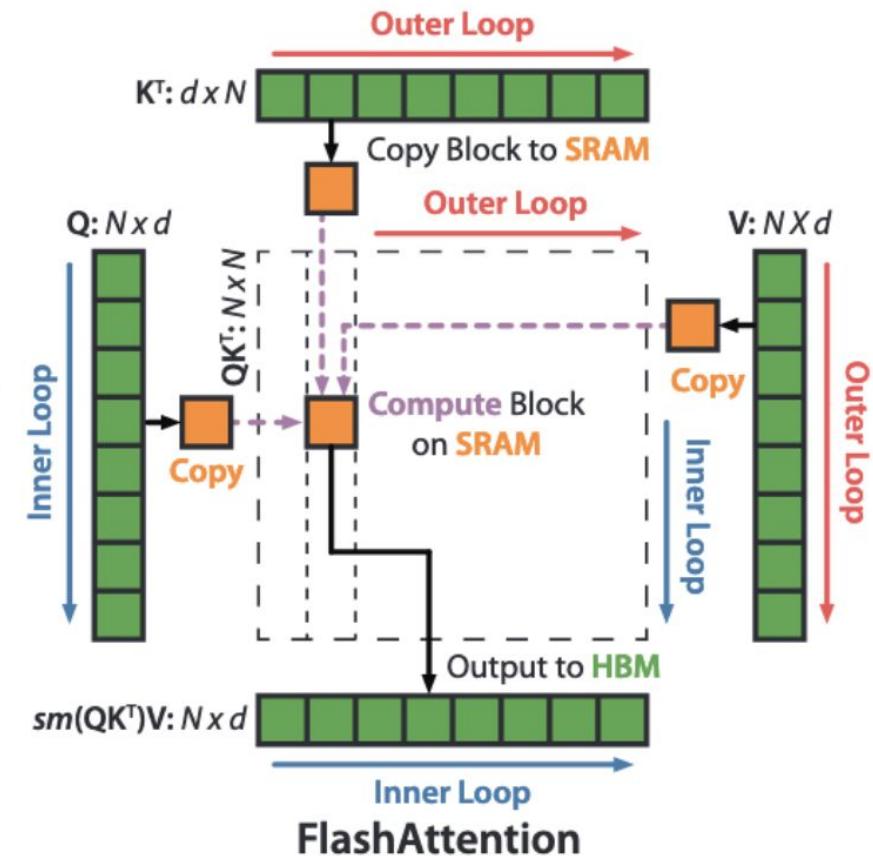
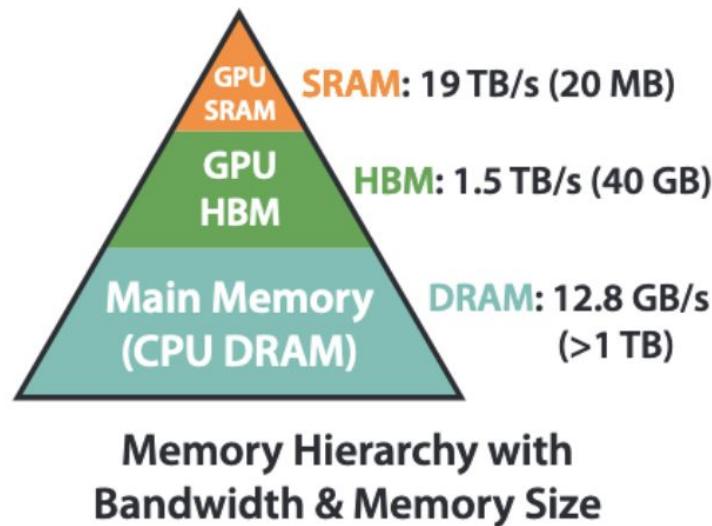
(c) Global Attention



(d) BIGBIRD

Различные вариации разреженного внимания и их смесь - BigBird

# Flash Attention



(Слева) Иерархия памяти при обучении на GPU.

(Справа) Иллюстрация поблочной процедуры вычисления внимания.

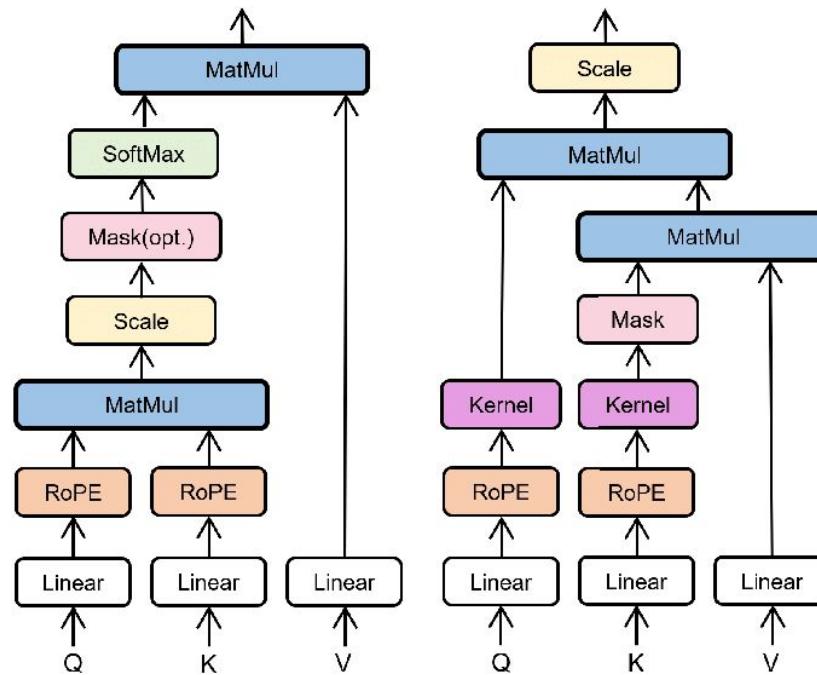
---

**Algorithm 0** Standard Attention Implementation

---

**Require:** Matrices  $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$  in HBM.

- 1: Load  $\mathbf{Q}, \mathbf{K}$  by blocks from HBM, compute  $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$ , write  $\mathbf{S}$  to HBM.
  - 2: Read  $\mathbf{S}$  from HBM, compute  $\mathbf{P} = \text{softmax}(\mathbf{S})$ , write  $\mathbf{P}$  to HBM.
  - 3: Load  $\mathbf{P}$  and  $\mathbf{V}$  by blocks from HBM, compute  $\mathbf{O} = \mathbf{PV}$ , write  $\mathbf{O}$  to HBM.
  - 4: Return  $\mathbf{O}$ .
- 



**Algorithm 1** FLASHATTENTION

**Require:** Matrices  $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$  in HBM, on-chip SRAM of size  $M$ .

- 1: Set block sizes  $B_c = \lceil \frac{M}{4d} \rceil$ ,  $B_r = \min(\lceil \frac{M}{4d} \rceil, d)$ .
- 2: Initialize  $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}$ ,  $\ell = (0)_N \in \mathbb{R}^N$ ,  $m = (-\infty)_N \in \mathbb{R}^N$  in HBM.
- 3: Divide  $\mathbf{Q}$  into  $T_r = \lceil \frac{N}{B_r} \rceil$  blocks  $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$  of size  $B_r \times d$  each, and divide  $\mathbf{K}, \mathbf{V}$  in to  $T_c = \lceil \frac{N}{B_c} \rceil$  blocks  $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$  and  $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$ , of size  $B_c \times d$  each.
- 4: Divide  $\mathbf{O}$  into  $T_r$  blocks  $\mathbf{O}_i, \dots, \mathbf{O}_{T_r}$  of size  $B_r \times d$  each, divide  $\ell$  into  $T_r$  blocks  $\ell_i, \dots, \ell_{T_r}$  of size  $B_r$  each, divide  $m$  into  $T_r$  blocks  $m_1, \dots, m_{T_r}$  of size  $B_r$  each.
- 5: **for**  $1 \leq j \leq T_c$  **do**
- 6:   Load  $\mathbf{K}_j, \mathbf{V}_j$  from HBM to on-chip SRAM.
- 7:   **for**  $1 \leq i \leq T_r$  **do**
- 8:     Load  $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$  from HBM to on-chip SRAM.
- 9:     On chip, compute  $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$ .
- 10:    On chip, compute  $\tilde{m}_{ij} = \text{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$ ,  $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$  (pointwise),  $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$ .
- 11:    On chip, compute  $m_i^{\text{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$ ,  $\ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$ .
- 12:    Write  $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1} (\text{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$  to HBM.
- 13:    Write  $\ell_i \leftarrow \ell_i^{\text{new}}$ ,  $m_i \leftarrow m_i^{\text{new}}$  to HBM.
- 14:   **end for**
- 15: **end for**
- 16: Return  $\mathbf{O}$ .



# Flash Attention

26

$$m(x) := \max_i x_i, \quad f(x) := [e^{x_1 - m(x)} \dots e^{x_B - m(x)}], \quad \ell(x) := \sum_i f(x)_i, \quad \text{softmax}(x) := \frac{f(x)}{\ell(x)}.$$

Как вычисляется softmax

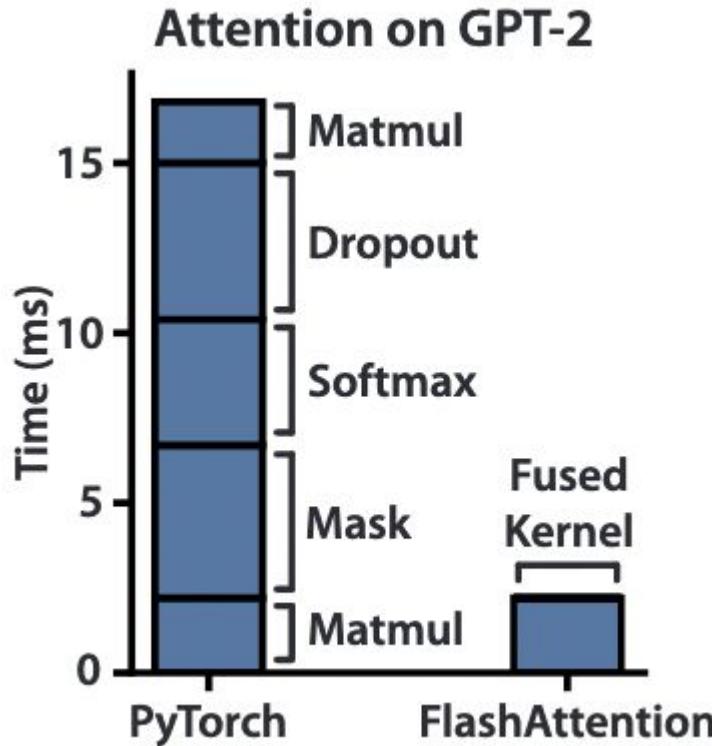
For vectors  $x^{(1)}, x^{(2)} \in \mathbb{R}^B$ , we can decompose the softmax of the concatenated  $x = [x^{(1)} \ x^{(2)}] \in \mathbb{R}^{2B}$  as:

$$m(x) = m([x^{(1)} \ x^{(2)}]) = \max(m(x^{(1)}), m(x^{(2)})), \quad f(x) = [e^{m(x^{(1)}) - m(x)} f(x^{(1)}) \ e^{m(x^{(2)}) - m(x)} f(x^{(2)})],$$

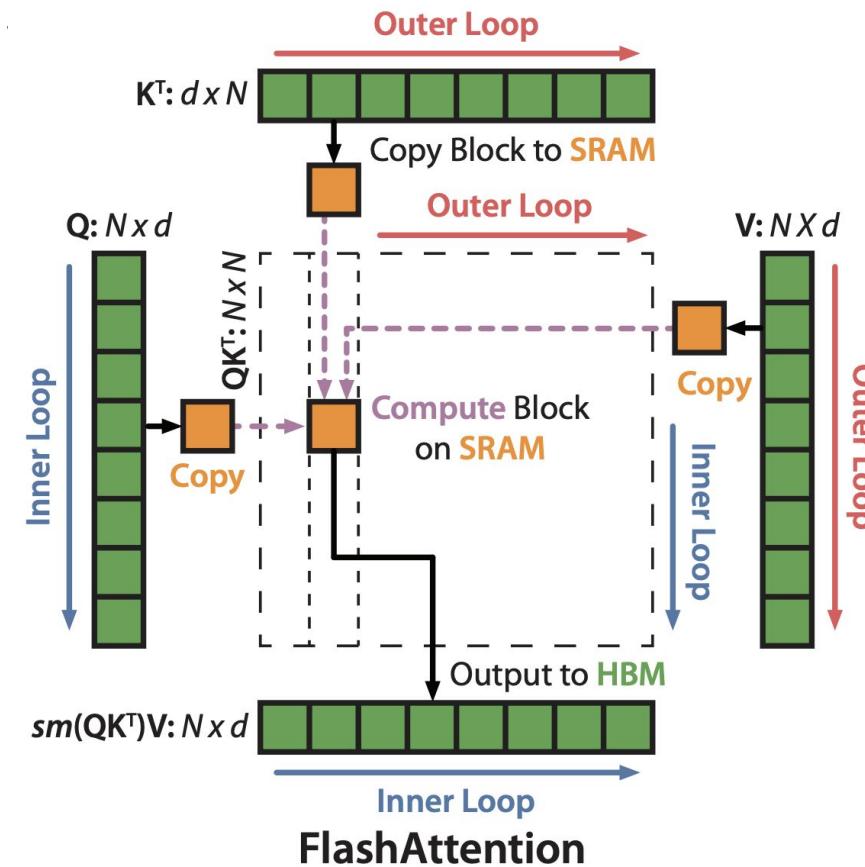
$$\ell(x) = \ell([x^{(1)} \ x^{(2)}]) = e^{m(x^{(1)}) - m(x)} \ell(x^{(1)}) + e^{m(x^{(2)}) - m(x)} \ell(x^{(2)}), \quad \text{softmax}(x) = \frac{f(x)}{\ell(x)}.$$

Слияние softmax из разных блоков

# Flash Attention



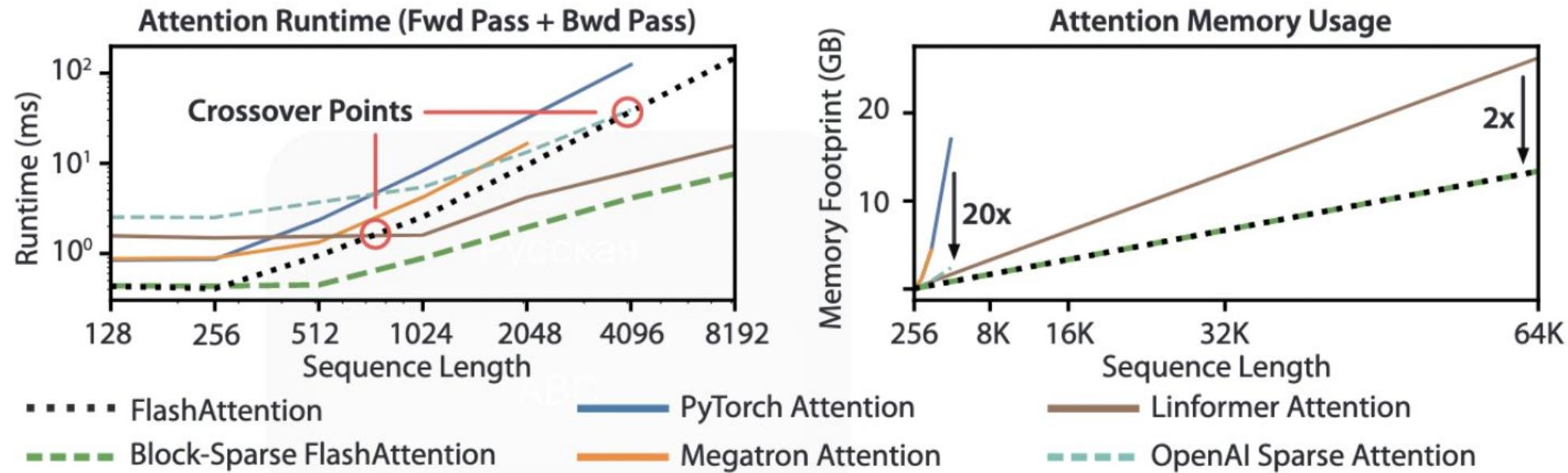
Attention в PyTorch и предложенный Flash Attention



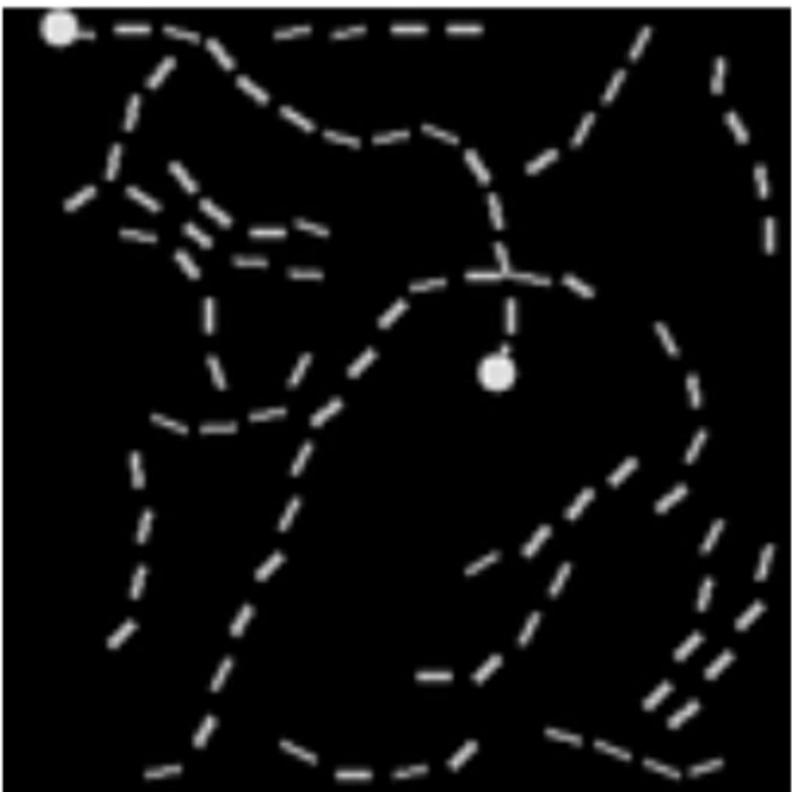
$$\mathbf{S} = \mathbf{Q}\mathbf{K}^\top \in \mathbb{R}^{N \times N}, \quad \mathbf{P} = \text{softmax}(\mathbf{S} \odot \mathbf{1}_{\tilde{M}}) \in \mathbb{R}^{N \times N}, \quad \mathbf{O} = \mathbf{P}\mathbf{V} \in \mathbb{R}^{N \times d},$$

Вычисление блочно-разреженного внимания. Матрица произведений между  $\mathbf{Q}$  и  $\mathbf{K}$  маскируется некоторой заданной заранее маской.

# Flash Attention



(Слева) Время прямого и обратного прохода на задаче моделирования длинных последовательности (Справа)  
Зависимость расхода памяти от длины последовательности.



(a) A positive example.

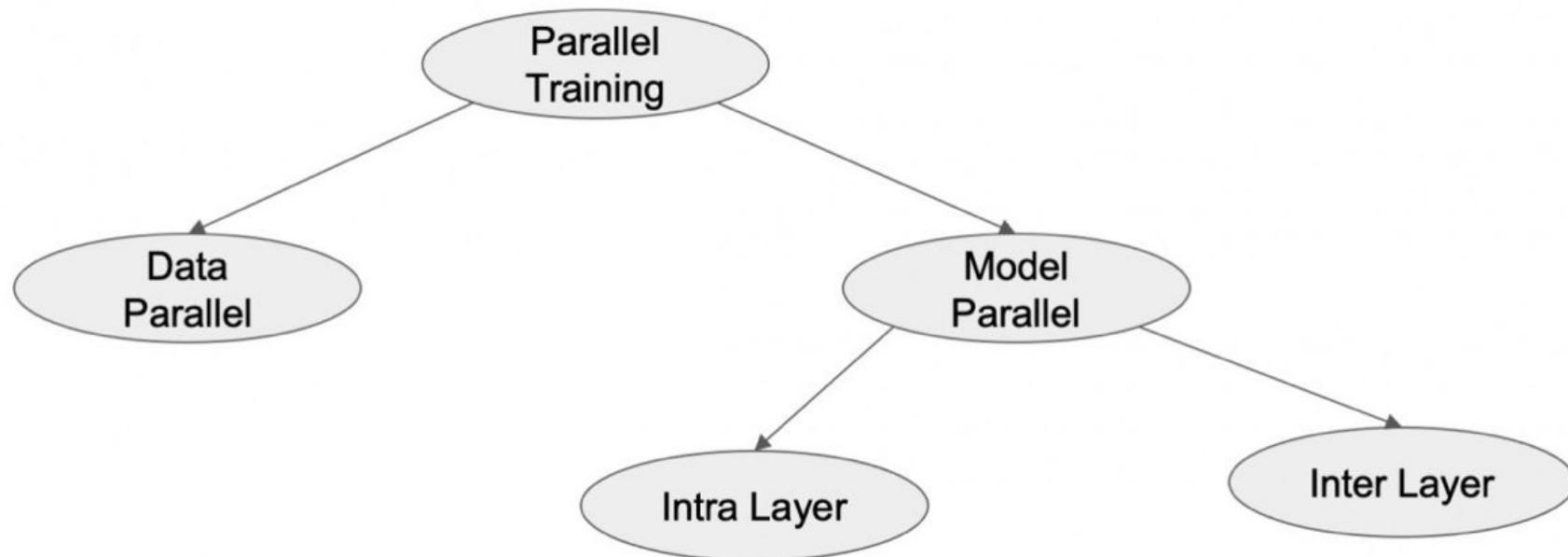
Пример из Path-X

Model	Path-X	Path-256
Transformer	X	X
Linformer [81]	X	X
Linear Attention [48]	X	X
Performer [11]	X	X
Local Attention [77]	X	X
Reformer [49]	X	X
SMYRF [18]	X	X
FLASHATTENTION	<b>61.4</b>	X
Block-sparse FLASHATTENTION	56.0	<b>63.1</b>

Все подходы до Flash Attention не справлялись с задачей Path-X/256

# Разбор техник распределенного обучения

## Parallelism Taxonomy





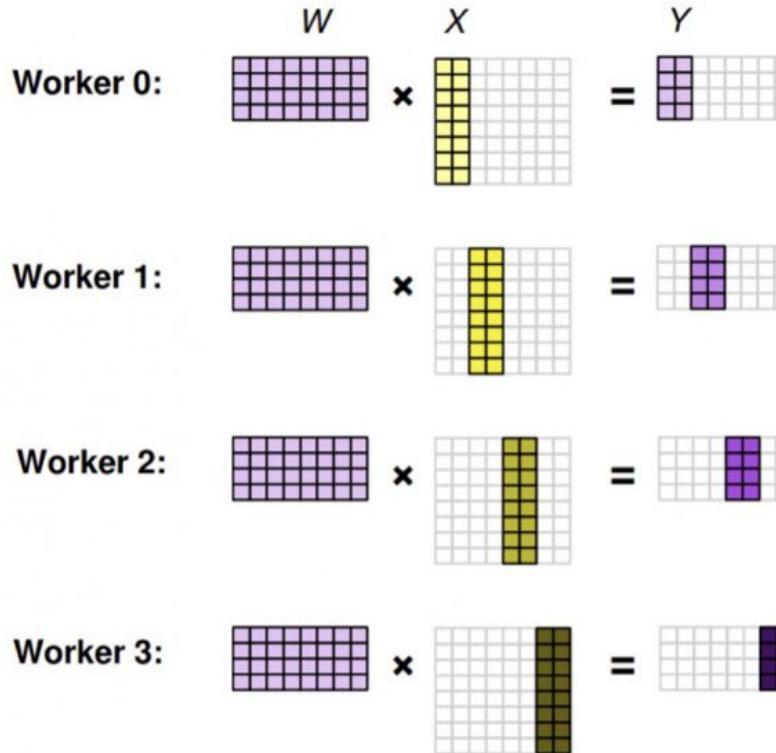
# Разбор техник распределенного обучения

31

## Data Parallel

- Each worker:
  - Has a copy of the entire neural network model
  - Responsible for compute of a portion of data (training minibatch)
- Forward pass:
  - Computes output activations for its portion of minibatch
  - No communication is needed
- Backward pass:
  - Computes activation gradients for its portion of minibatch
  - Computes contribution to the weight gradient based on its portion of minibatch
    - All workers' contributions must be summed before weight update
- Weight update:
  - Each worker updates its copy of the model with combined gradients
  - Variants: distributed optimizer

## Data Parallel: Forward Pass



- No communication needed
  - Own portion of output becomes own portion of input for next layer
- Backward activation-gradient compute is essentially the same

## Data Parallel: Backward Pass

Worker 0:

$$\begin{matrix} dY \\ \times \\ \text{---} \end{matrix} \quad \begin{matrix} X^\top \\ \times \\ \text{---} \end{matrix} = \begin{matrix} dW \end{matrix}$$

- Each worker computes a different weight gradient ( $dW$ )
  - Based only on its own unique portion of data
- Weight gradients will have to be communicated so that after update each worker has the same exact weights

Worker 1:

$$\begin{matrix} \text{---} \\ \times \\ \text{---} \end{matrix} \quad \begin{matrix} X^\top \\ \times \\ \text{---} \end{matrix} = \begin{matrix} \text{---} \end{matrix}$$

Worker 2:

$$\begin{matrix} \text{---} \\ \times \\ \text{---} \end{matrix} \quad \begin{matrix} X^\top \\ \times \\ \text{---} \end{matrix} = \begin{matrix} \text{---} \end{matrix}$$

Worker 3:

$$\begin{matrix} \text{---} \\ \times \\ \text{---} \end{matrix} \quad \begin{matrix} X^\top \\ \times \\ \text{---} \end{matrix} = \begin{matrix} \text{---} \end{matrix}$$

## Data Parallel: Communication

- Allreduce:
  - Sum all the workers' gradients
  - Distribute the sum to all the workers
- After Allreduce each worker has the same “global” gradient
  - Can execute a weight update on its own model -> all workers will have the same model
- Any exposed communication is overhead, thus:
  - Use efficient communication (hw and sw), overlap communication, etc.



# Разбор техник распределенного обучения

35

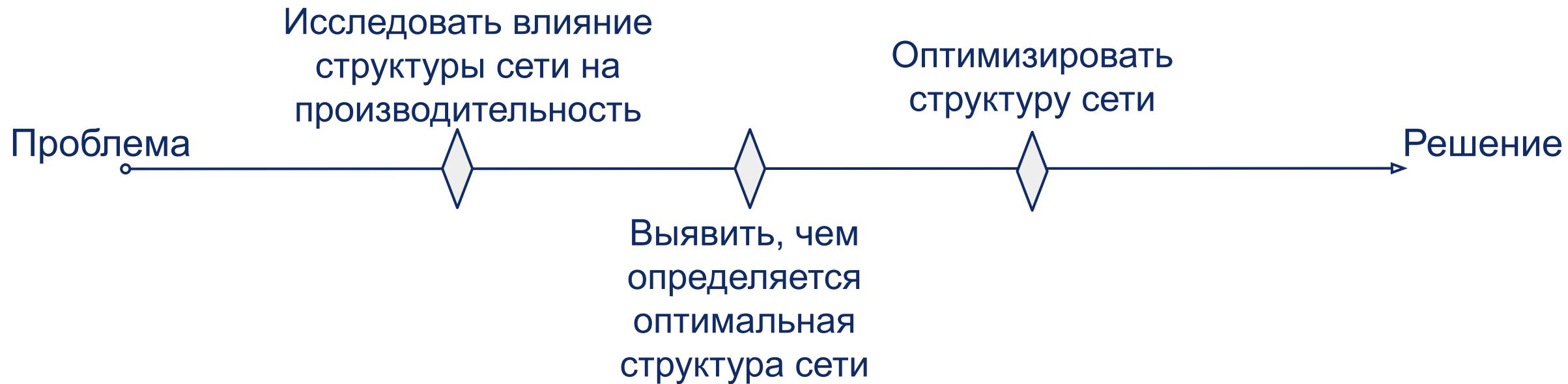
## Allreduce Implementation Choices

- Each of  $N$  workers is responsible for:
  - Summing  $1/N$  gradients collected from  $(N - 1)$  peers
  - Distributing the sums to the  $(N - 1)$  peers
- “Ring” reduction
  - For any topology that contains a 1D torus (ring)
  - Each worker communicates with 2 neighbors
  - $2(N - 1)$  steps, worker sends/receives  $1/N$  of all bytes
    - Each step requires a synchronization  $\rightarrow 2(N - 1)$  syncs total
- “One-shot” reduction:
  - For fully-connected topologies (switches)
  - Each worker communicates with  $(N - 1)$  neighbors
  - 2 steps, each with  $(N - 1)$  substeps
    - One step per synchronization  $\rightarrow 2$  syncs total
  - Allows the use of arithmetic in switches (Mellanox SHARP)
    - Reduces memory accesses and math by the worker

# Структурный анализ и оптимизация сверточных нейронных сетей с небольшим размером выборки

Проблема: В прикладных задачах доступность обучающих выборок ограничена - проблема “малых данных”

## План работы





# Рост глубоких нейронных сетей

37

8-слойная AlexNet

19-слойная VGG

22-слойная GoogleNet

152-слойная ResNet

Обобщение идеи: более глубокие сети работают лучше  
Тенденция развития нейронных сетей движется в направлении ещё более глубоких и более сложных структур.

Мало данных

Недостаточное обучение  
или  
переобучение

Типичные стратегии, используемые при обучении больших данных, могут оказаться неприменимыми в задачах малых данных



### использование предварительно обученных сетей

Однако в областях, где используются медицинские изображения, формат входных пикселей (например, данные, отличные от RGB, такие как ультразвуковые/МРТ-изображения в оттенках серого) может полностью отличаться от обычных фотографических изображений, где используются предварительно обученные сети из других областей изображений.

Также может произойти «отрицательный перенос» в тех случаях, когда существует большой разрыв между набором данных для предварительного обучения и набором данных для обучения, что приводит к падению производительности по сравнению с обучением с нуля

Поскольку мы сталкиваемся с необходимостью применять нейронные сети к выборкам все меньшего и меньшего размера, вопрос заключается в том, может ли подмножество более крупной выборки данных иметь ту же оптимальную сеть, что и весь набор данных. Например, при меньшем размере обучающих данных очень глубокая сеть может плохо обобщать.



## Генерация сетевых структур

39

Использовалась почти жесткая граница размерности VC для любой многослойной нейронной сети с кусочно-линейной функцией. Для ограничения размера сети использовалась абсолютная верхняя граница. Она выражается как:

$$VC\text{Dim}(\text{sgn}(F)) = O(W * L * \log W)$$

$W$  - общее количество весов в сети (включая смещение)

$L$  - количество слоев в сети

$\text{sgn}()$  - функция знака

$F$  - множество вещественнозначных функций, вычисляемых или предоставляемых сетью.

Размерность изображения на любом слое равна ( $n \times n$ )

Все операции с окнами/фильтрами (объединение и свертка) имеют одинаковую прокладку: добавление нулей к краям в соответствии с размером фильтра ( $k$ )



Для любого сверточного слоя общее кол-во весов вычислялось по формуле:

$$W = k * k * \text{Входные каналы} * \text{Выходные каналы} + \text{Выходные каналы}$$

Для любого полносвязного слоя общее количество весов вычислялось как:

$$W = \text{Входной размер} * \text{Выходной размер} + \text{Выходной размер}$$

Max-pooling слои не имеют весов и, следовательно, не вносят вклад в VC-dimension (однако они изменяют размерность входа для следующих слоев и, следовательно, играют роль в влиянии на VC-dimension всей структуры сети).

Далее суммируются все значения весов всех слоев, а затем применяется формула для VC-dimension.



# Перестановка размеров слоев

41

Цель подхода: определить, какие сетевые структуры могут быть эффективными для обучения на различных наборах данных, учитывая их размеры и характер.

- Для упорядочивания различных возможных слоев сети используется древовидная структура данных
- В каждом узле дерева рассчитывается VC-размерность структуры, включая соответствующий конечный выходной слой
- После определения узла дерева, который удовлетворяет условию размера VC, этот узел помещается в дерево, и рекурсивно вызывается функция построения

Фильтры сверточных  
слоев

5x5  
7x7

Окна пулинга

2x2  
4x4

Уровень размерности  
канала

10



После первой итерации выбирается 5 наиболее эффективных нейронных сетей

Далее для каждой сети подбирается “идеальный” размер каждого слоя

Эти сети имеют одинаковую структурную конфигурацию

При этом размеры слоев удваивается при каждом шаге

Для разных подмножеств данных количество возможных вариантов размеров слоев будет отличаться. Это позволяет более детально исследовать влияние размеров слоев на производительность сети.

# Метод обучения и проверки, наборы данных - MNIST

43



Был использован метод ADAM с mini-batch = 100 и 10, кросс-энтропия softmax с логарифмической целевой функцией

Скорость обучения поддерживается постоянной на рекомендуемом значении 0,001 для всех эпох обучения

Снижение веса не использовалось

Всего использовалось 8, 4 и 6 итераций градиентного спуска

Внедрен протокол ранней остановки, чтобы избежать переобучения

Для демонстрационных целей было разрешено три варианта размеров сети: 32, 64 и 128. Было сгенерировано 9261 сеть для подмножества размера 100, 3267 сетей для подмножества размера 500 и 13729 сетей для подмножества размера 1000

# Метод обучения и проверки, наборы данных - CIFAR-10



Был использован метод ADAM с mini-batch = 100 и 10, кросс-энтропия softmax с логарифмической целевой функцией

Скорость обучения поддерживается постоянной на рекомендуемом значении 0,001 для всех эпох обучения

Снижение веса не использовалось  
Всего использовалось 40 итераций градиентного спуска за эпоху

Внедрен протокол ранней остановки, чтобы избежать переобучения

Для демонстрационных целей было разрешено три варианта размеров сети: 32, 64, 128 и 256. Было сгенерировано в общей сложности 1172 различных комбинации измерений слоев

# Метод обучения и проверки, наборы данных - mitosis



Был использован метод ADAM с mini-batch = 100 и 10, кросс-энтропия с логарифмической целевой функцией

Скорость обучения поддерживается постоянной на рекомендуемом значении 0,001 для всех эпох обучения

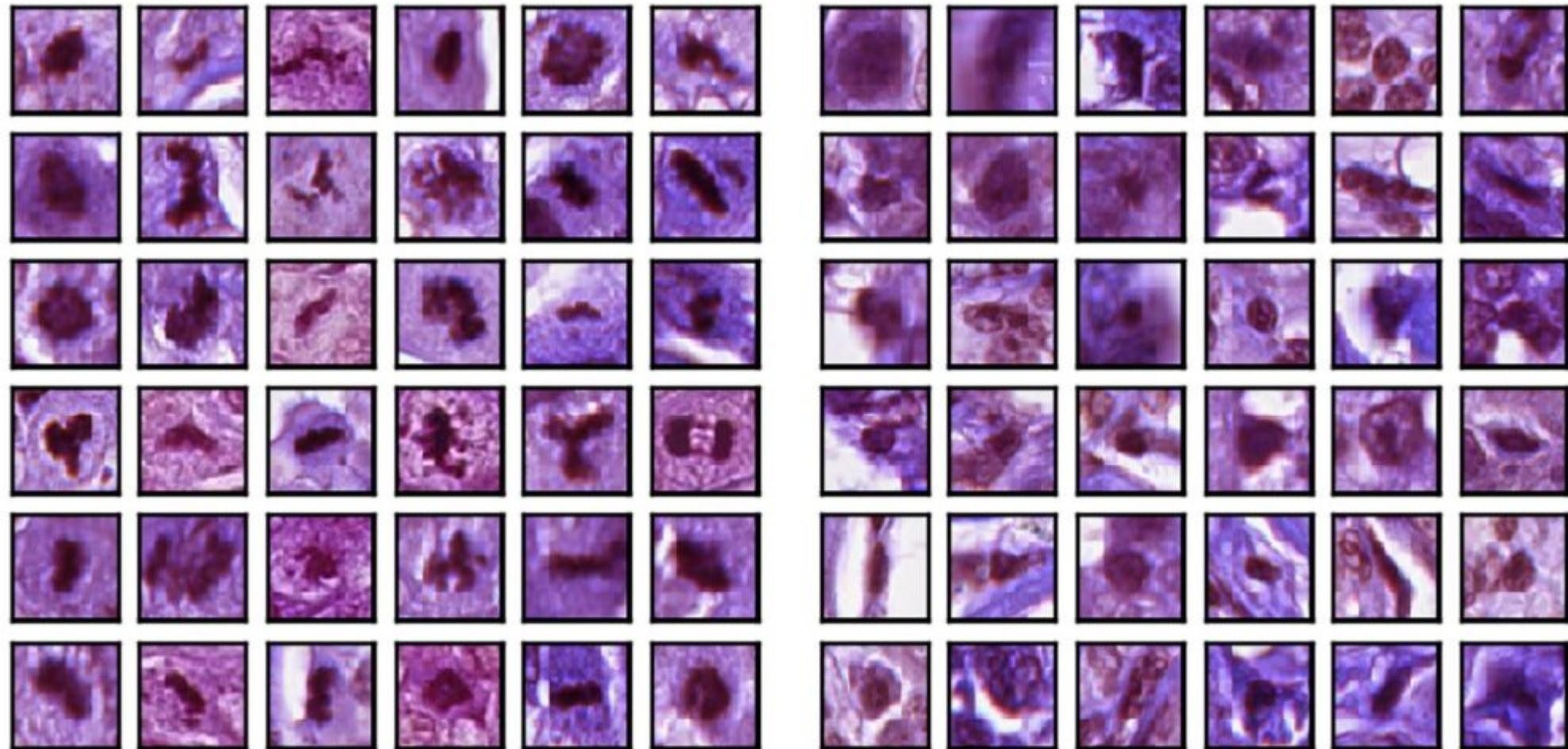
Снижение веса не использовалось

Всего использовалось 25 итераций градиентного спуска за эпоху

Внедрен протокол ранней остановки, чтобы избежать переобучения

Для демонстрационных целей было разрешено три варианта размеров сети: 32 и 64. Сгенерировали 576 различных возможных сетей.

# Метод обучения и проверки, наборы данных - mitosis

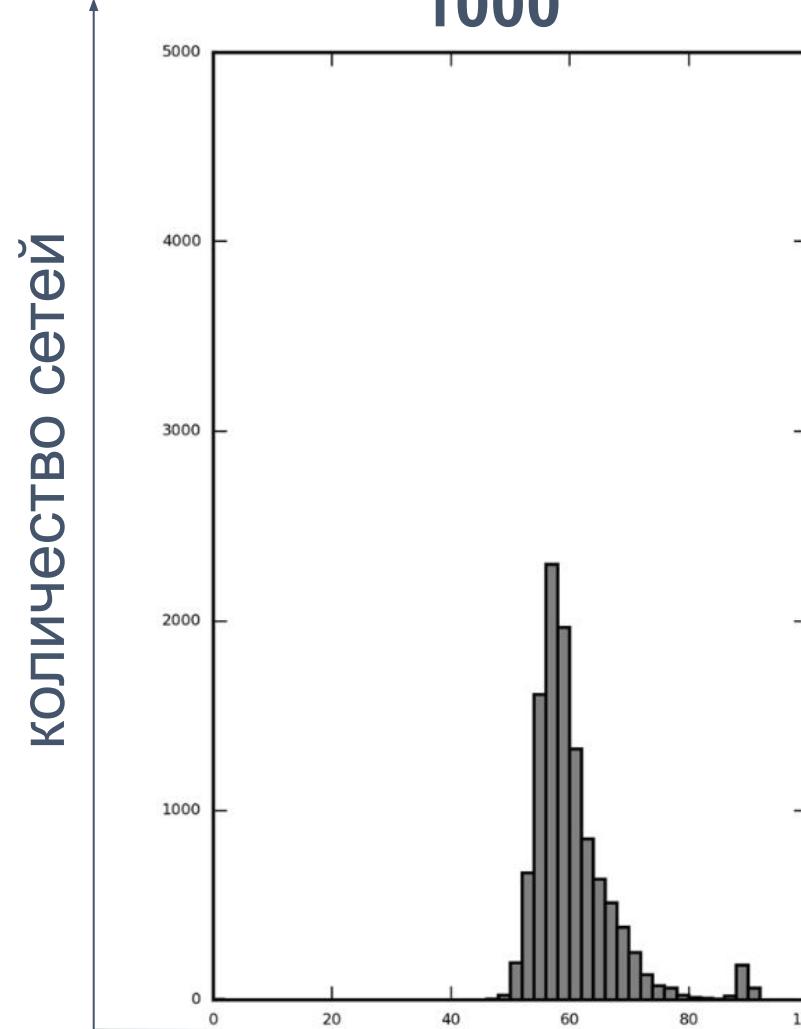




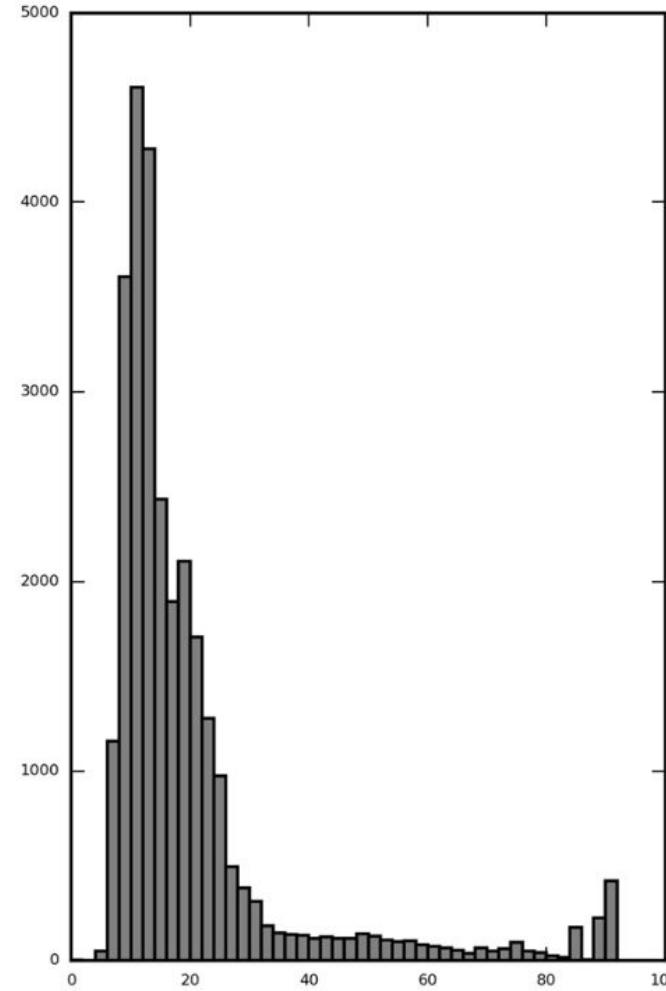
# Результаты

47

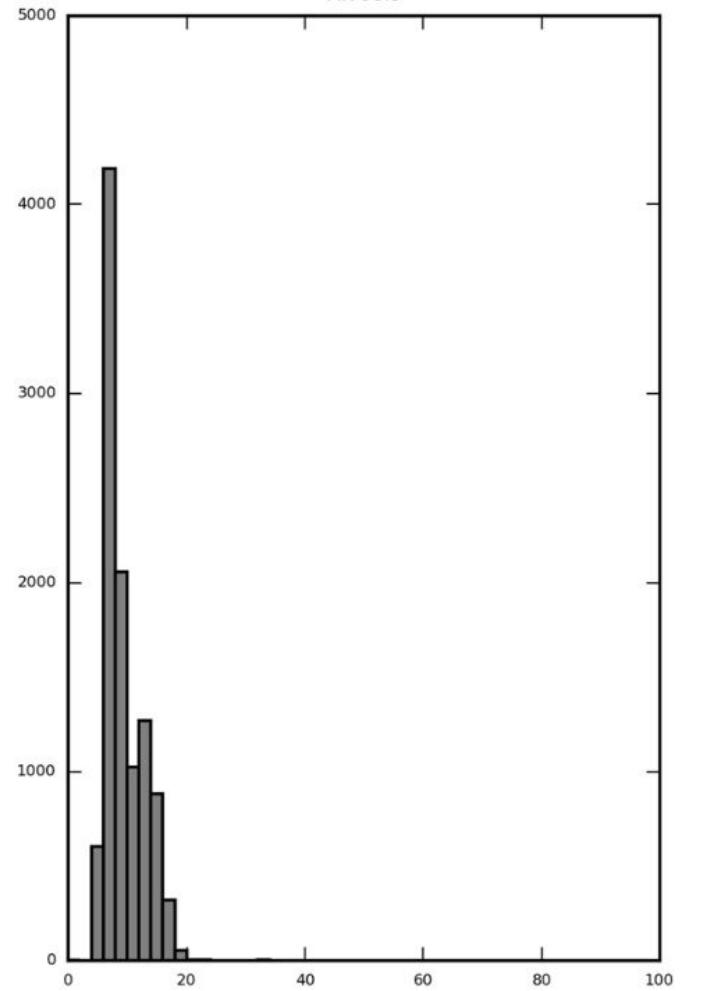
cifar-10  
1000



mnist  
1000



mitosis  
3300

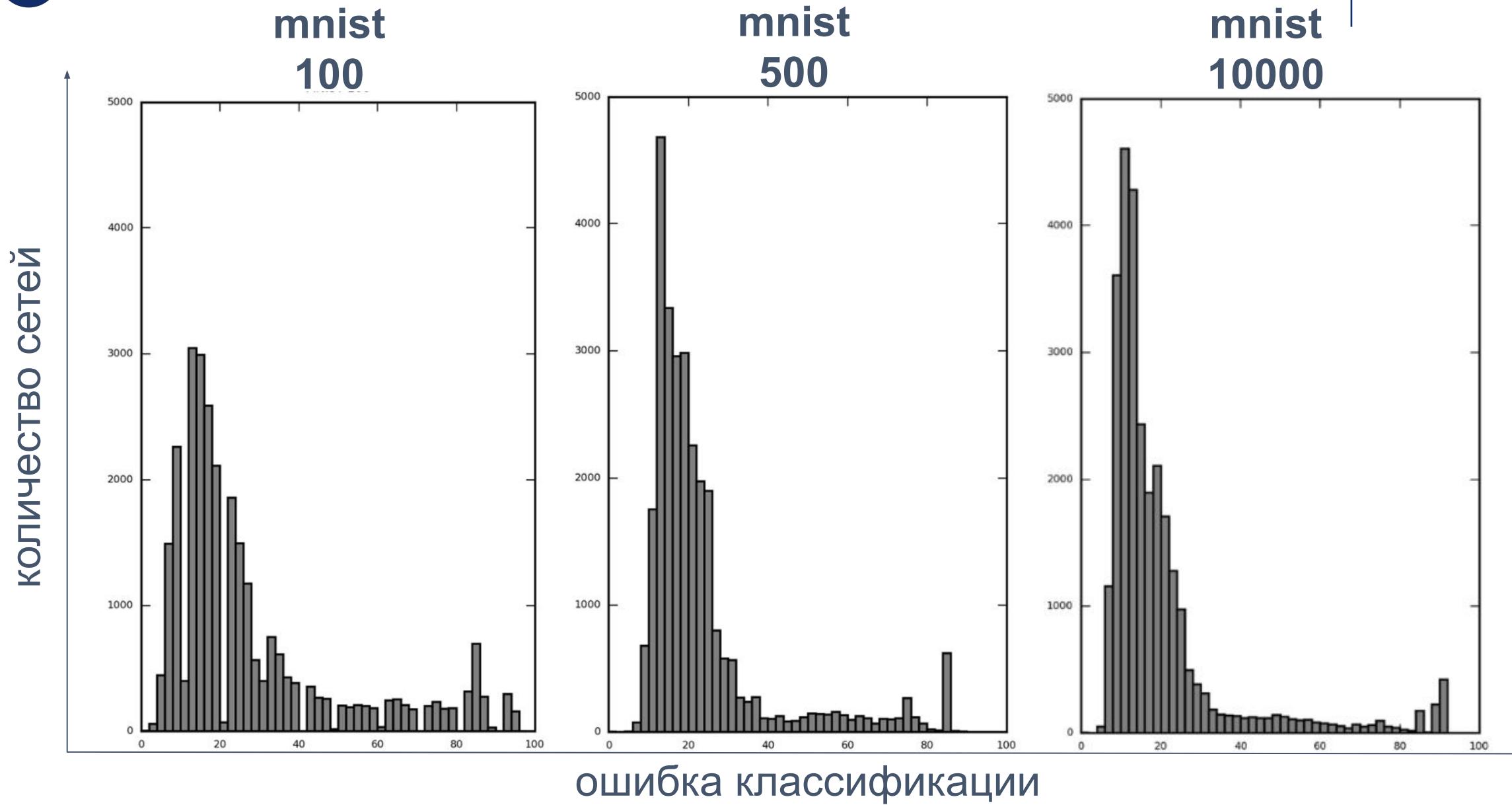


ошибка классификации

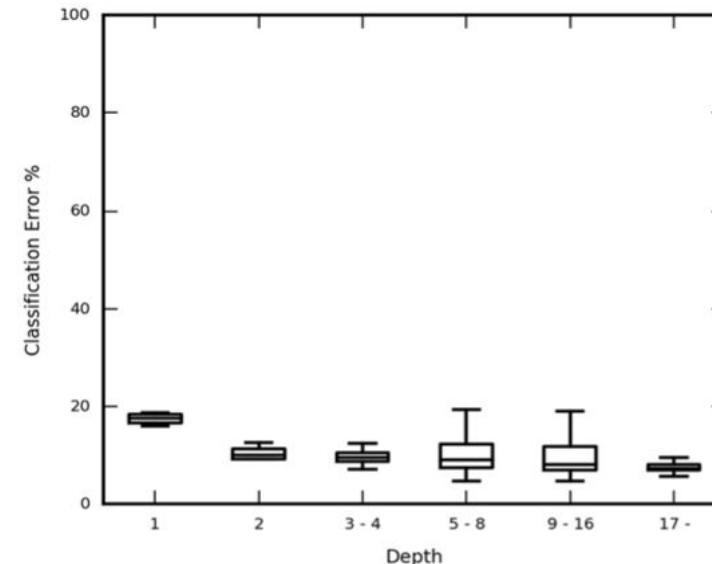
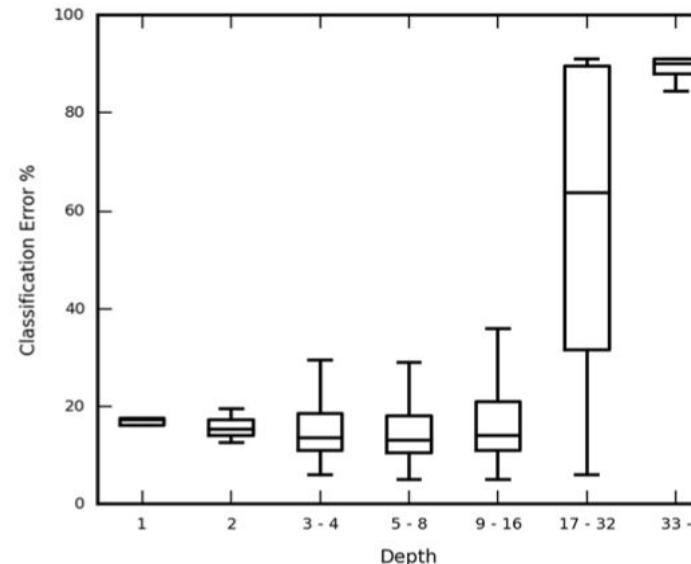
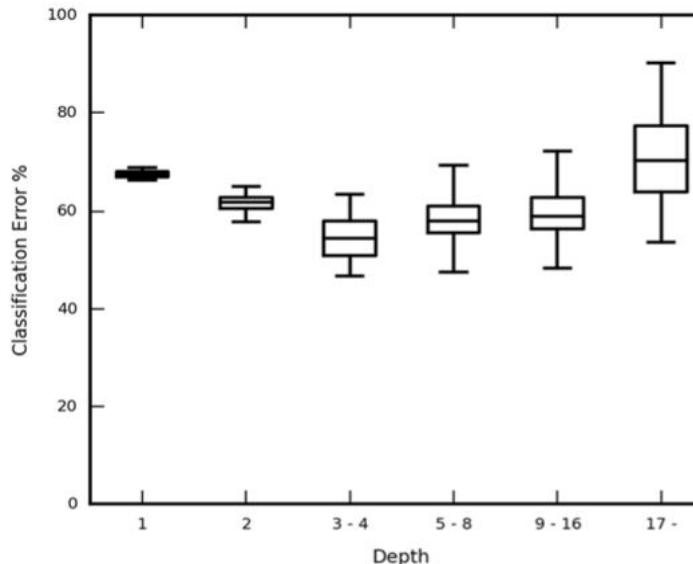
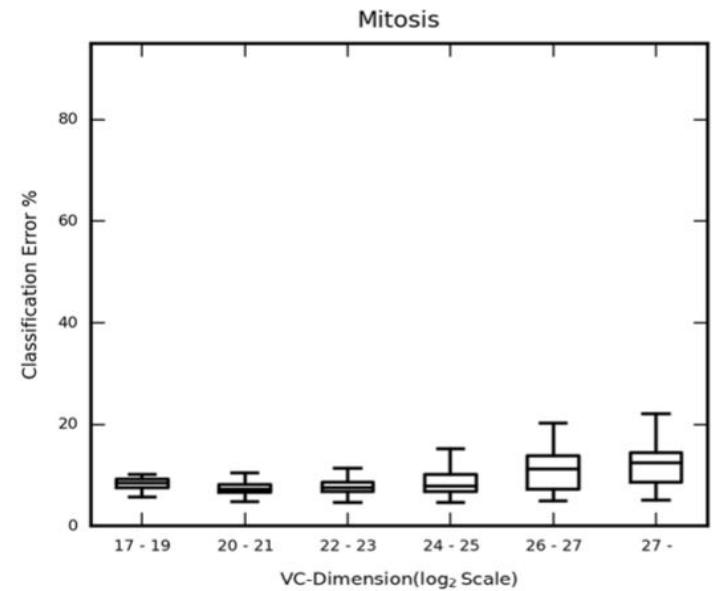
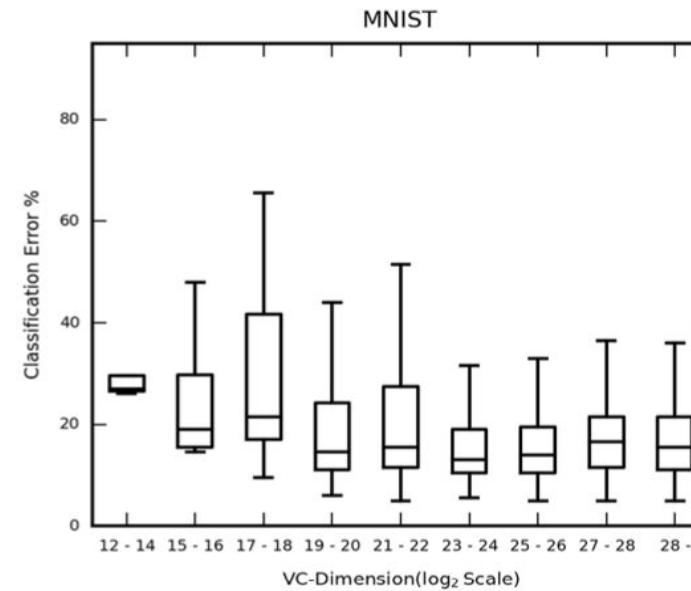
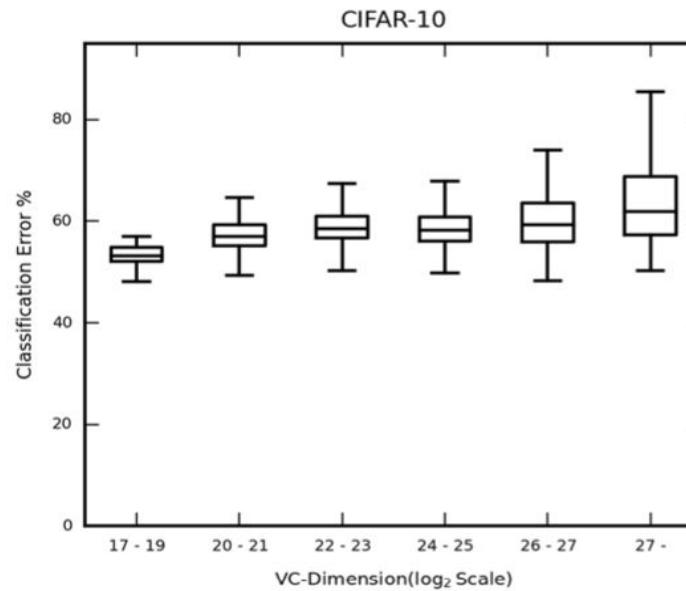


# Результаты

48



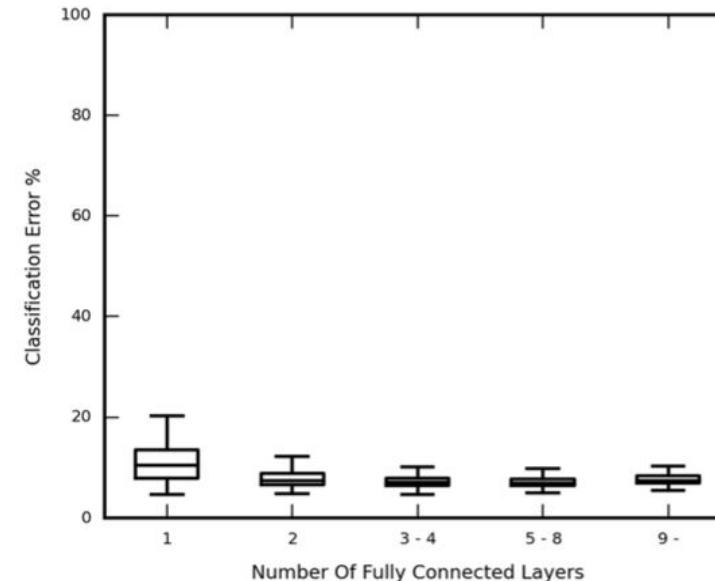
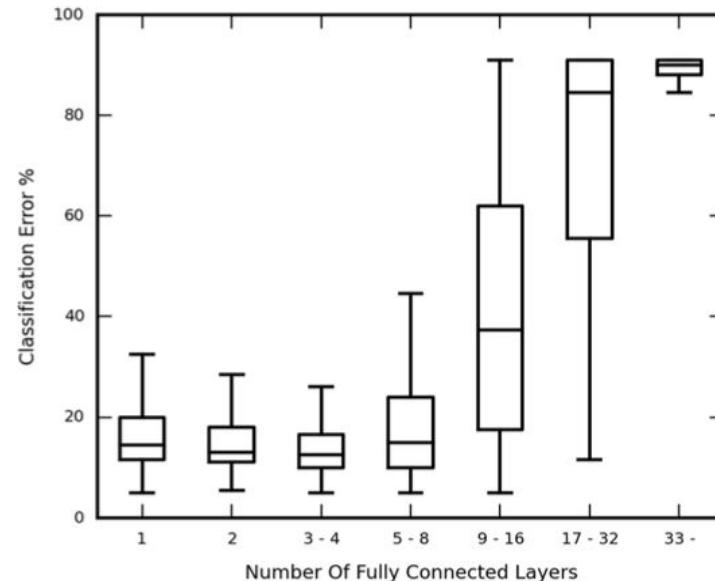
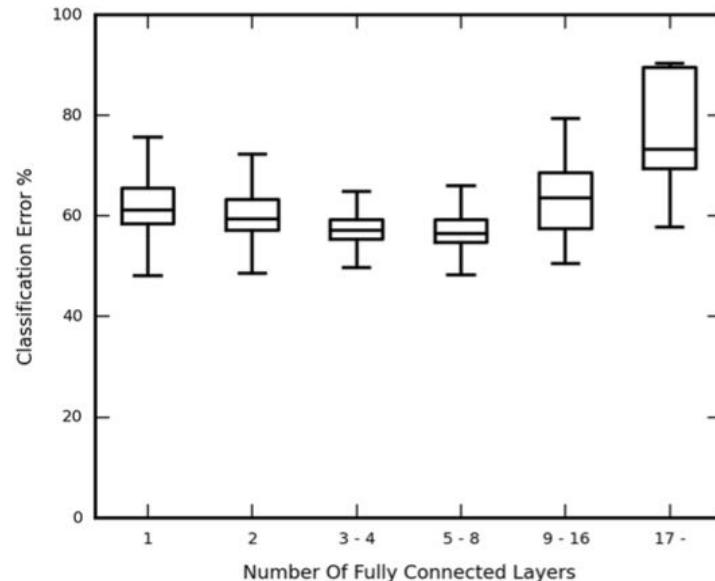
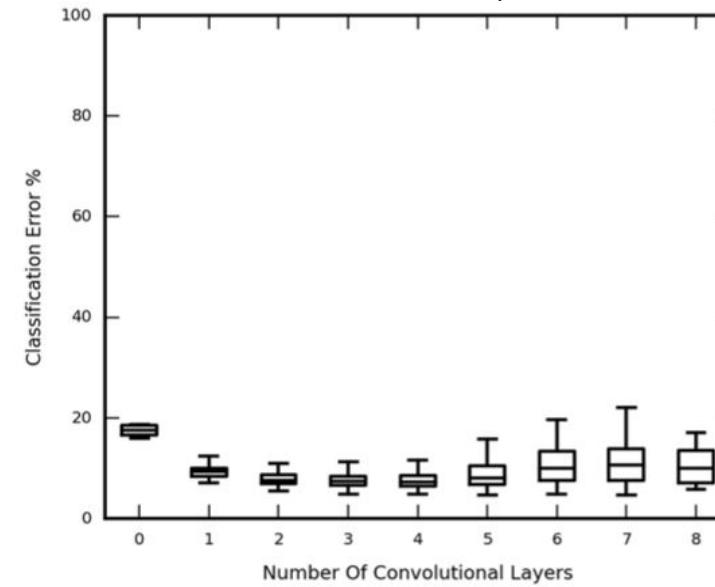
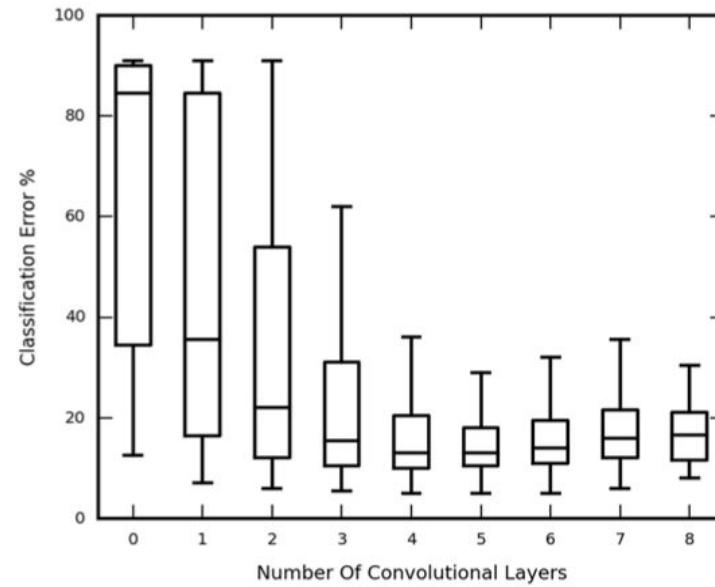
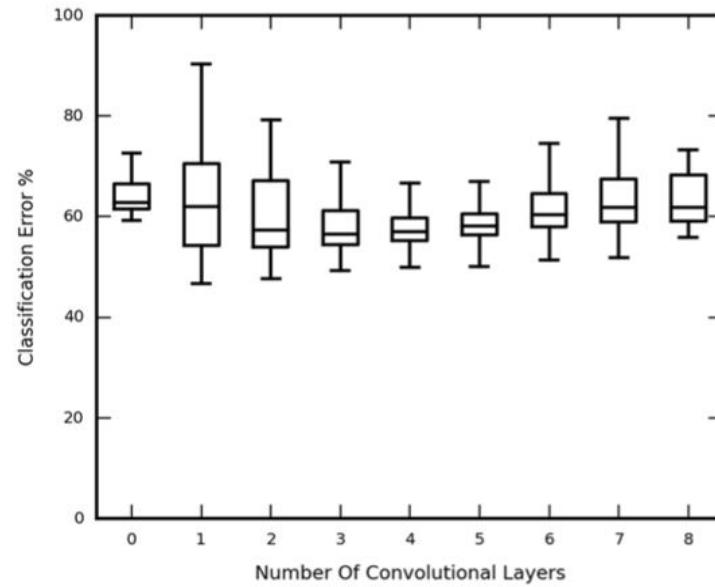
# Результаты



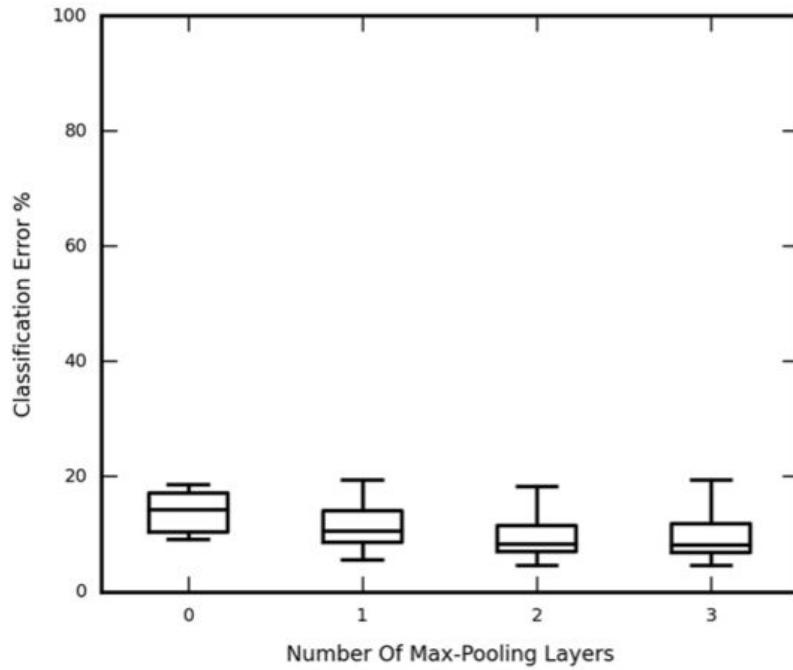
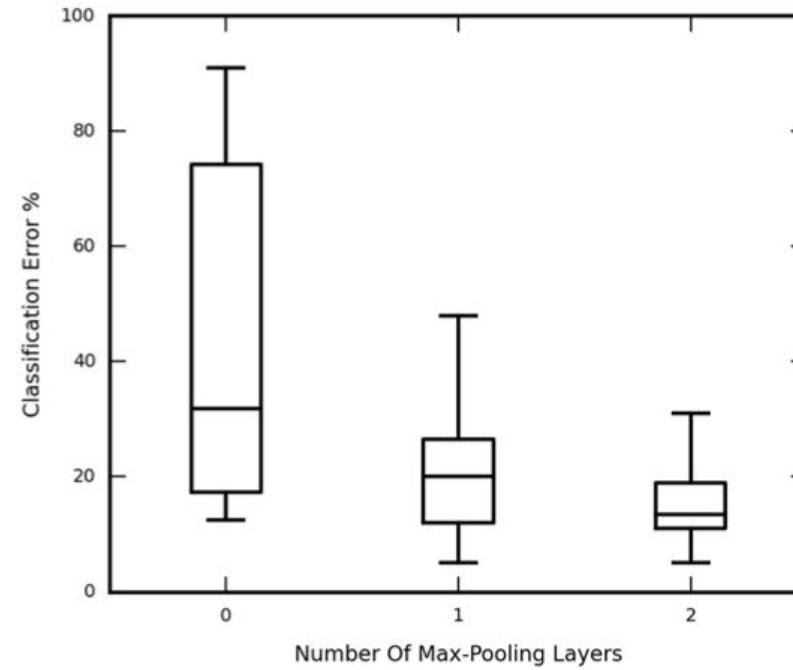
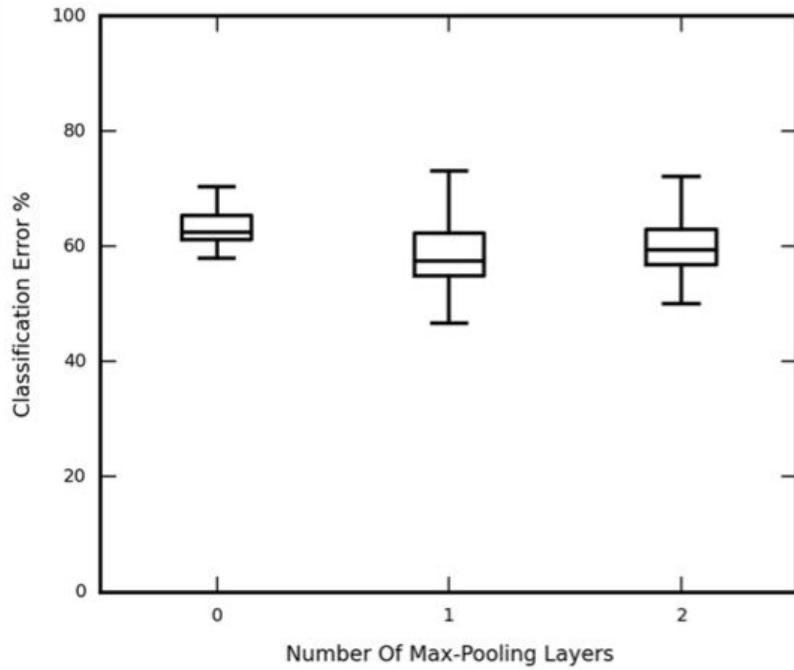


# Результаты

50



# Результаты



В целом, исследование подчеркивает важность адаптации сетевой структуры к характеру данных и подтверждает, что стандартные подходы к построению сети могут не всегда быть оптимальными для конкретных задач.

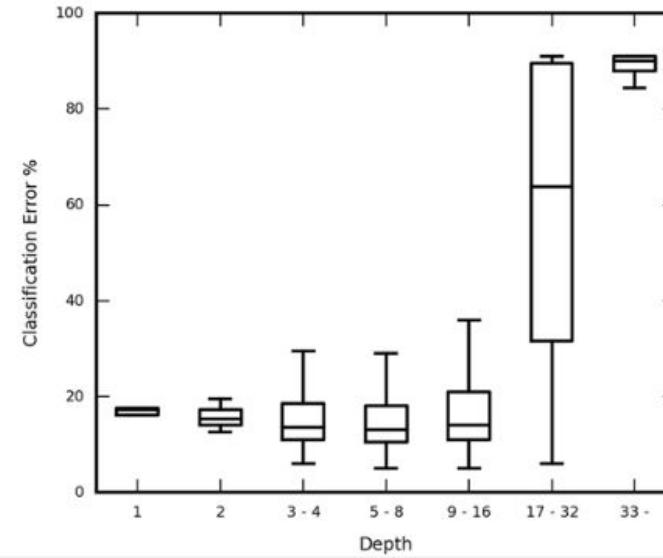
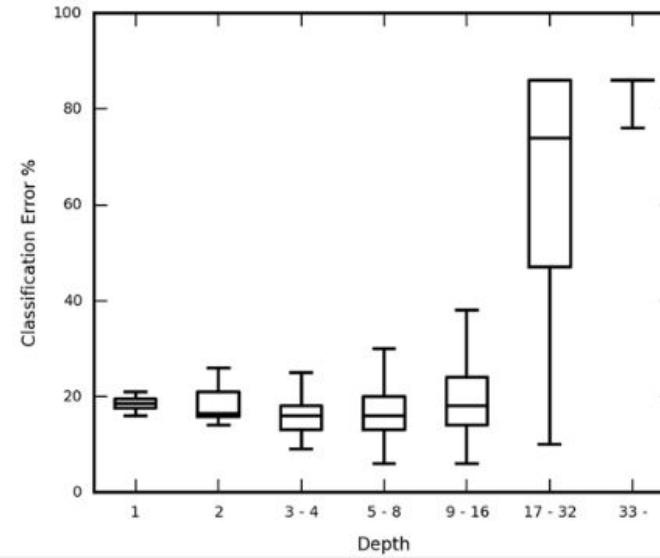
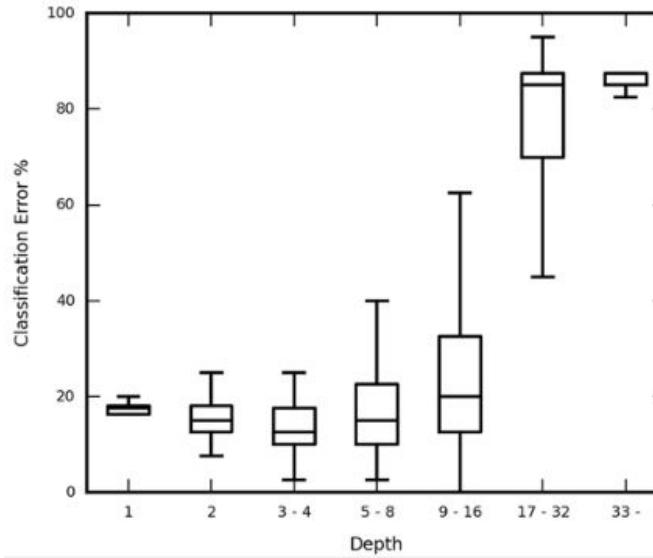
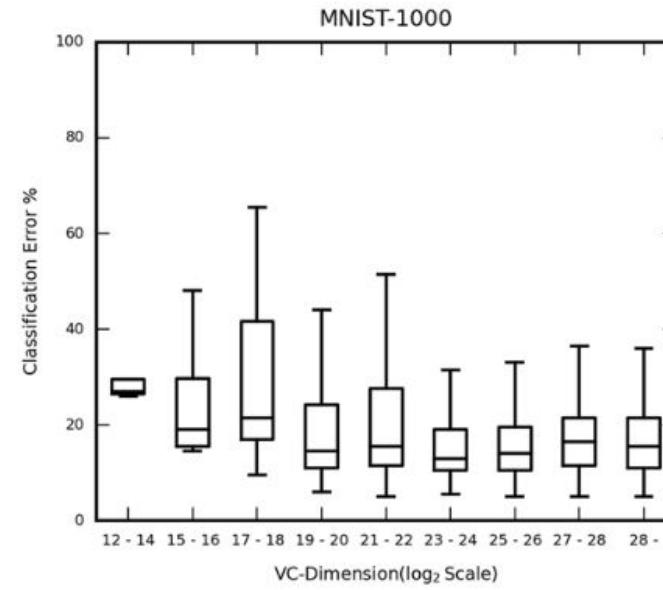
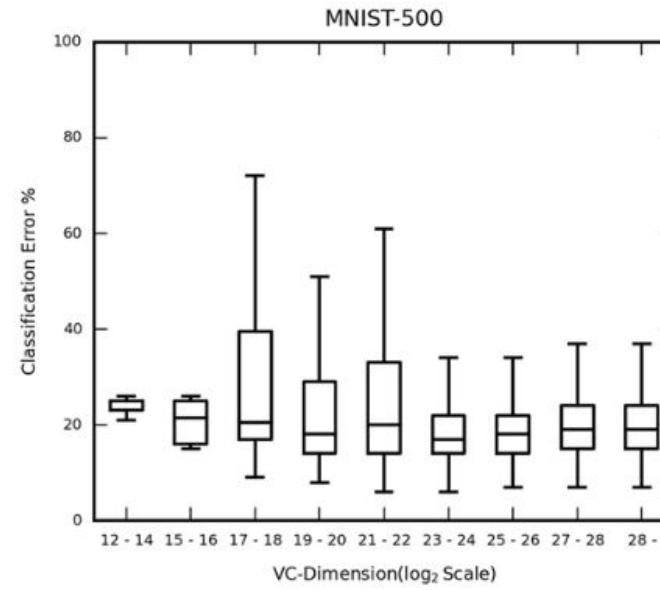
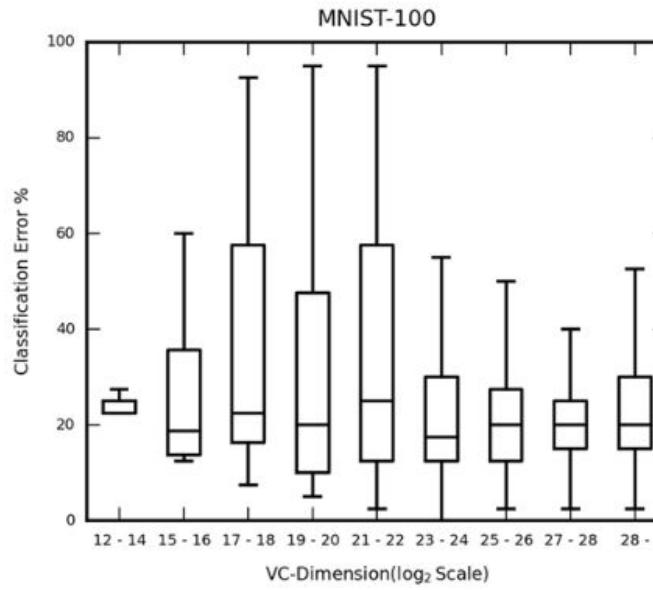


# Оптимальная структура сети

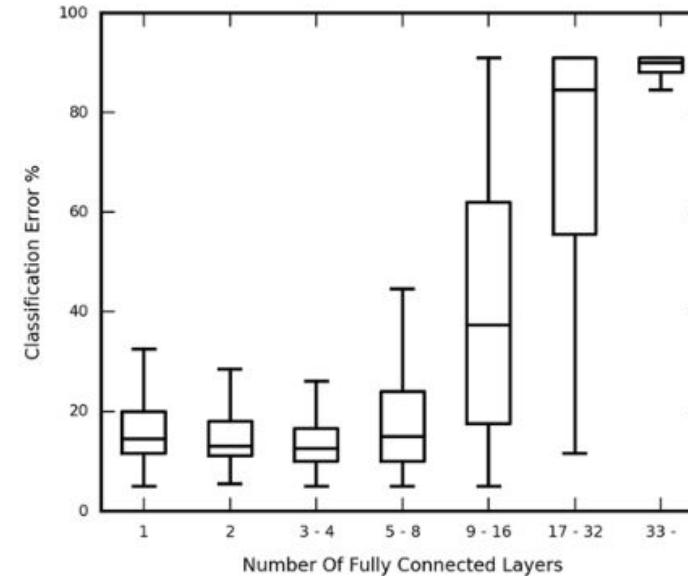
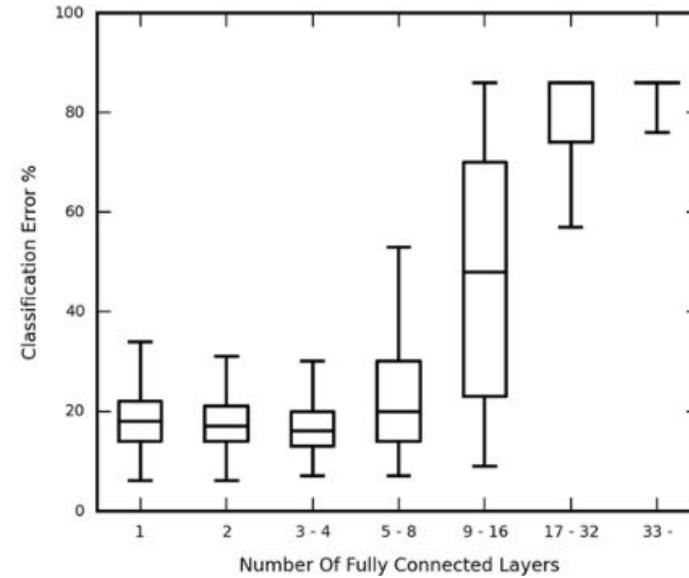
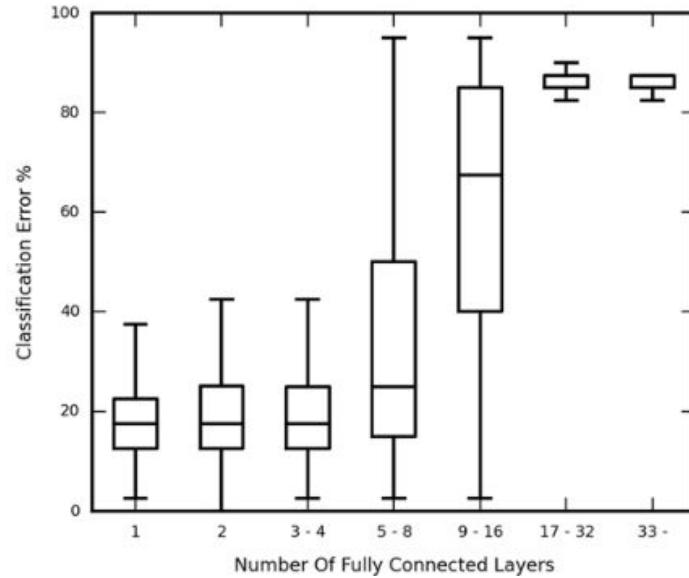
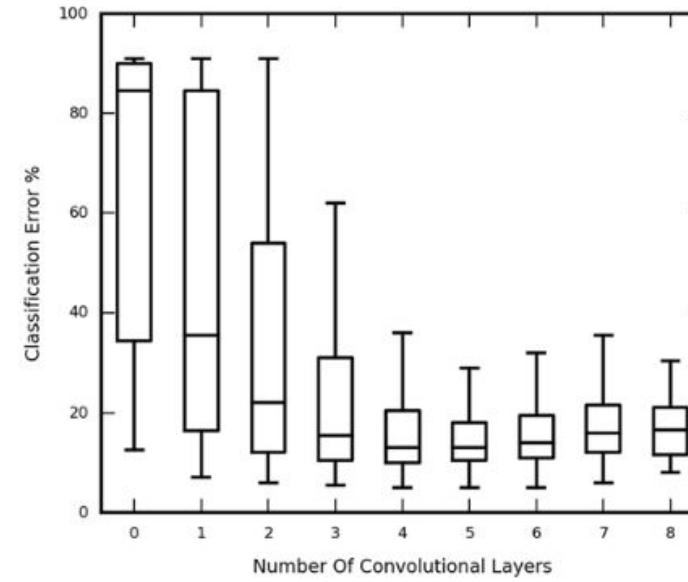
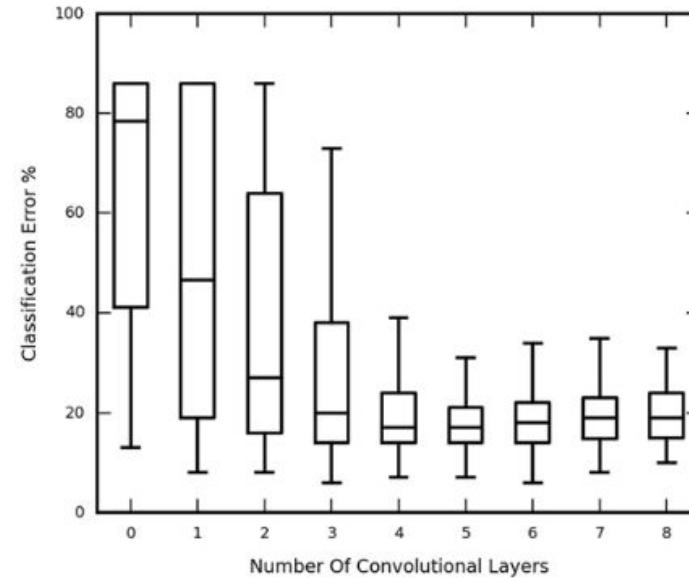
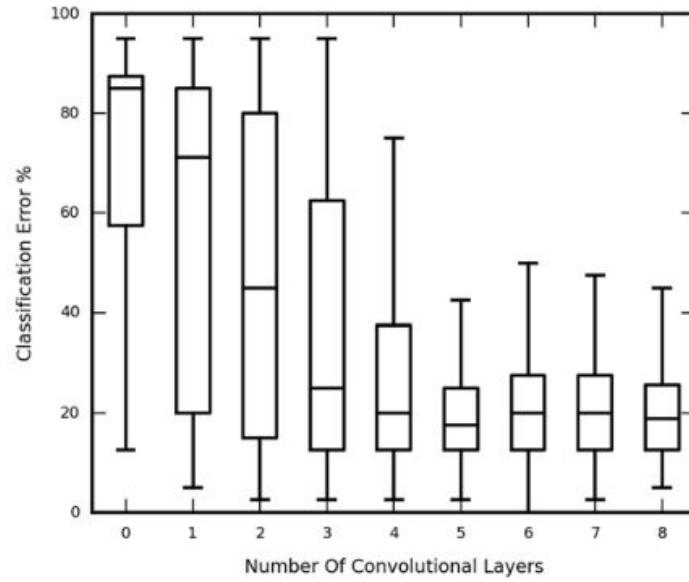
52

- Оптимальная сетевая структура зависит от данных: Наиболее эффективные сети для каждого набора данных сильно отличаются друг от друга, что подчеркивает, что оптимальная сетевая структура управляет данными.
- Противоречие принципу "чем глубже, тем лучше": Результаты показывают, что принцип "чем глубже, тем лучше" не всегда справедлив для традиционных сверточных нейронных сетей, особенно для небольших наборов данных.
- Определение оптимальной структуры зависит от характера данных: Результаты показывают высокую согласованность в оптимальных сетевых структурах для разных размеров выборки набора данных MNIST, что указывает на то, что оптимальные сетевые структуры определяются в первую очередь характером данных.
- Роль размера выборки: Несмотря на различия в размерах выборки, оптимальная структура для каждой выборки имеет определенные тенденции, что может указывать на то, что размер выборки может играть роль в определении оптимального количества слоев.
- Важность архитектурного поиска: Эмпирический архитектурный поиск может итеративно найти подходящую классификационную сеть для данного набора данных. Однако причина, по которой не существует обобщённой стратегии проектирования сети, неизвестна и, скорее всего, основана на данных.

# Влияние размера слоя на ошибку классификации

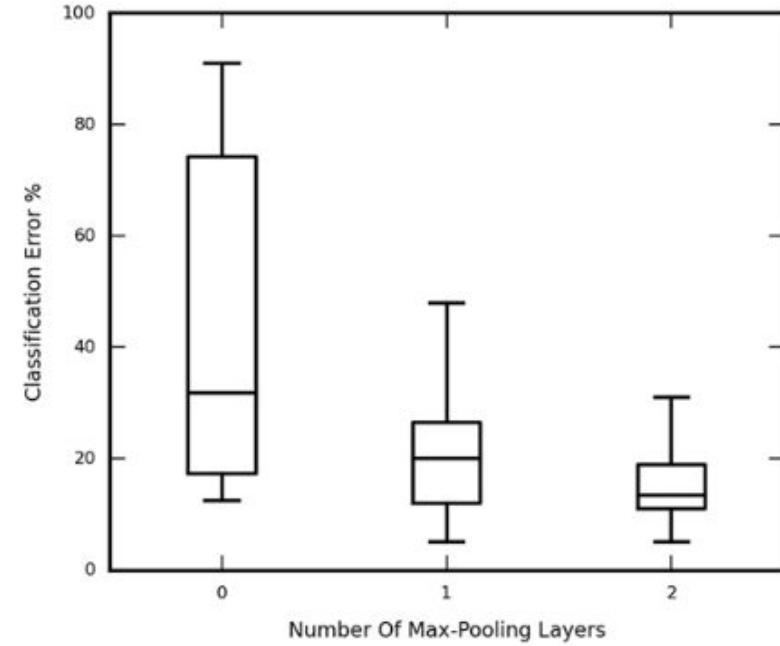
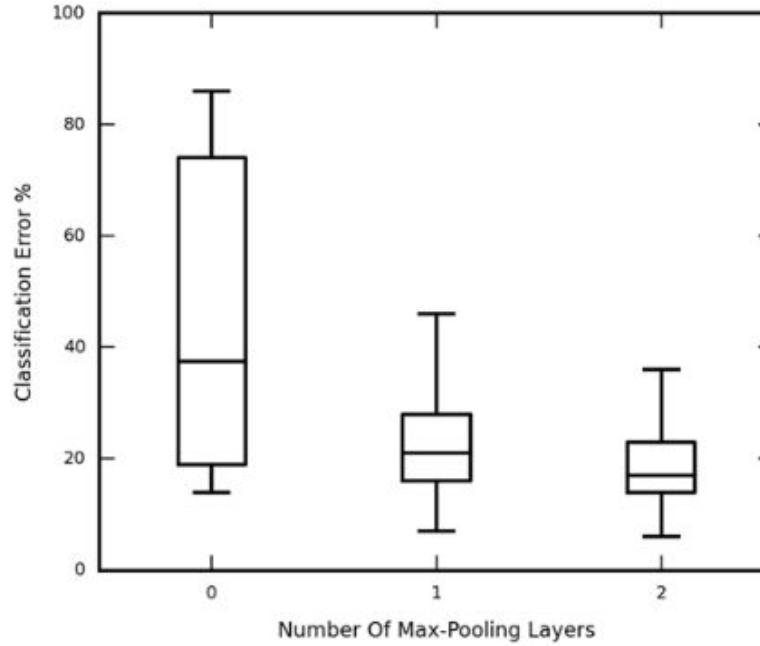
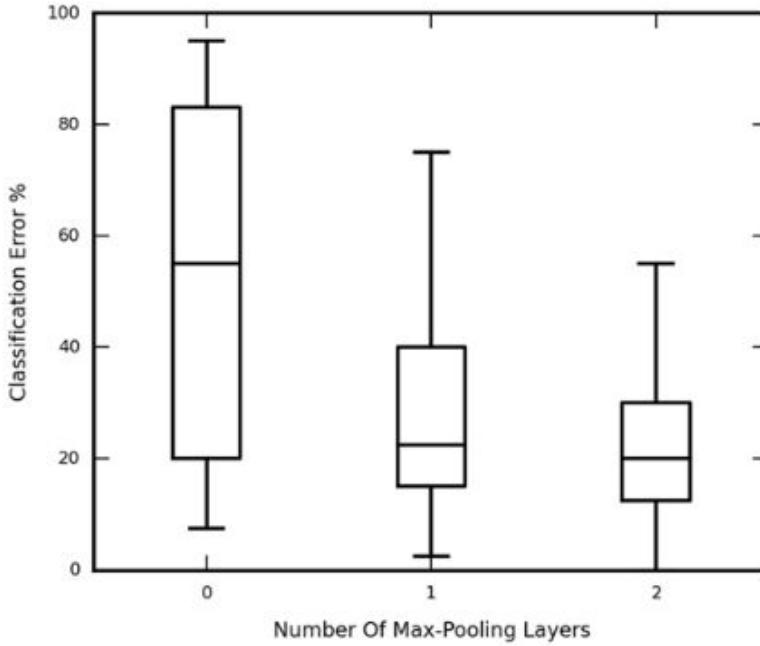


# Влияние размера слоя на ошибку классификации



# Влияние размера слоя на ошибку классификации

55



Основные наблюдения:

- Узкий диапазон размеров слоев
- Влияние размеров слоев на производительность
- Влияние размера выборки



## Результаты этапа выбора структуры

Dataset	Sample-Size	Optimal Structure	Cross-Validation Accuracy (%)	Classification Accuracy from for the Full Test Set (%)
MNIST	100	$MP2 \rightarrow Conv5 \rightarrow Conv7 \rightarrow MP2 \rightarrow Conv5 \rightarrow Conv7 \rightarrow Conv7 \rightarrow Conv7 \rightarrow Full \rightarrow Full$	100.00%*	60.16%
	500	$Conv7 \rightarrow MP4 \rightarrow Conv7 \rightarrow Conv5 \rightarrow Full \rightarrow Full \rightarrow Full \rightarrow Full \rightarrow Full$	93.00%	85.34%
	1000	$Conv5 \rightarrow MP2 \rightarrow Conv5 \rightarrow MP2 \rightarrow Conv5 \rightarrow Conv7 \rightarrow Full \rightarrow Full \rightarrow Full \rightarrow Full$	93.00%	92.68%
CIFAR-10	5000	$Conv5 \rightarrow MP4 \rightarrow Full \rightarrow Full \rightarrow Full$	52.20%	50.46%
Mitosis	3300	$Conv5 \rightarrow MP4 \rightarrow Conv7 \rightarrow Conv7 \rightarrow Conv7 \rightarrow MP2 \rightarrow Full \rightarrow Full \rightarrow Full \rightarrow Full$	92.63%	92.93%

## Результаты шага перестановки размеров слоя

Dataset	Sample-Size	Optimal Permutation	Cross-Validation Accuracy (%)	Classification Accuracy from for the Full Test Set (%)
MNIST	100	$Conv5,64 \rightarrow MP4,64 \rightarrow Conv5,128 \rightarrow Conv5,64 \rightarrow Conv5,64 \rightarrow Conv7,32 \rightarrow Conv5,128 \rightarrow Full,32 \rightarrow Full,32 \rightarrow Full,10$	100.00%*	68.33%
	500	$Conv7,64 \rightarrow MP4,64 \rightarrow Conv7,128 \rightarrow Conv5,64 \rightarrow Full,128 \rightarrow Full,128 \rightarrow Full,128 \rightarrow Full,32 \rightarrow Full,10$	93.00%	91.33%
	1000	$Conv5,32 \rightarrow MP2,32 \rightarrow Conv5,32 \rightarrow MP2,32 \rightarrow Conv5,128 \rightarrow Conv7,32 \rightarrow Full,64 \rightarrow Full,64 \rightarrow Full,32 \rightarrow Full,10$	94.50%	93.66%
CIFAR-10	5000	$Conv5,64 \rightarrow MP4,64 \rightarrow Full,64 \rightarrow Full,32 \rightarrow Full,10$	53.50%	55.18%
Mitosis	3300	$Conv5,32 \rightarrow MP2,32 \rightarrow Conv7,32 \rightarrow MP2,32 \rightarrow Conv5,64 \rightarrow Conv5,32 \rightarrow MP2,32 \rightarrow Conv7,32 \rightarrow Full,64 \rightarrow Full,32 \rightarrow Full,2$	95.87%	94.10%



В большинстве случаев (CIFAR-10 и MNIST) производительность снижалась с увеличением глубины сети. Улучшение по сравнению со средней производительностью случайной структуры сети в ошибке классификации составило 27,99% (MNIST, 100 образцов), 16,45% (MNIST, 500 образцов) и 13,11% (MNIST, 1000 образцов).

- Важность размера данных: играет меньшую роль
- Влияние глубины сети: в большинстве случаев производительность снижается с увеличением глубины
- Оптимизация размеров слоев: в некоторых случаях (MNIST 100 и mitosis) влияние ширины может быть значительным