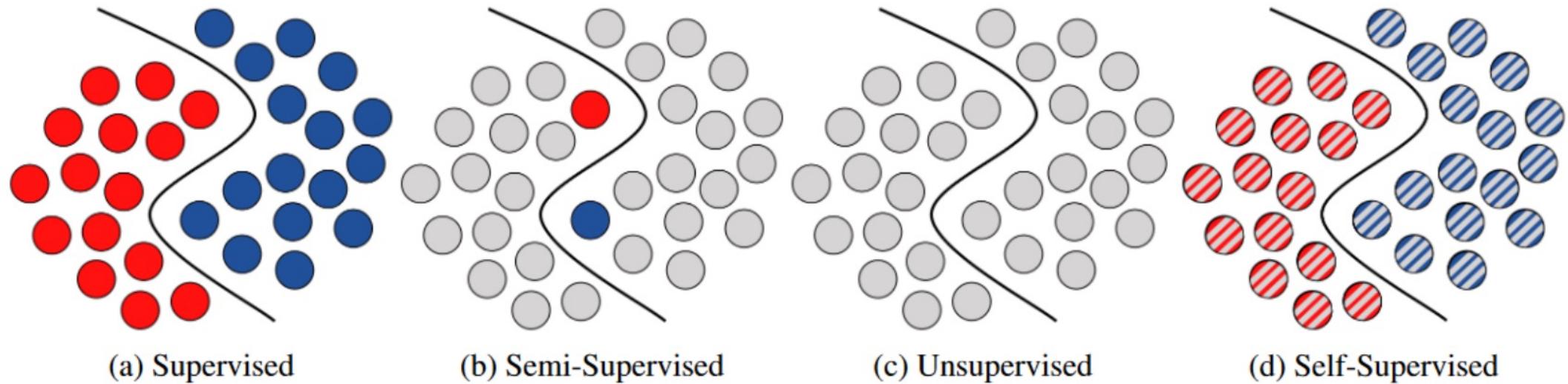


Self-supervised representation learning

Артем Исмагилов

Михаил Катунькин

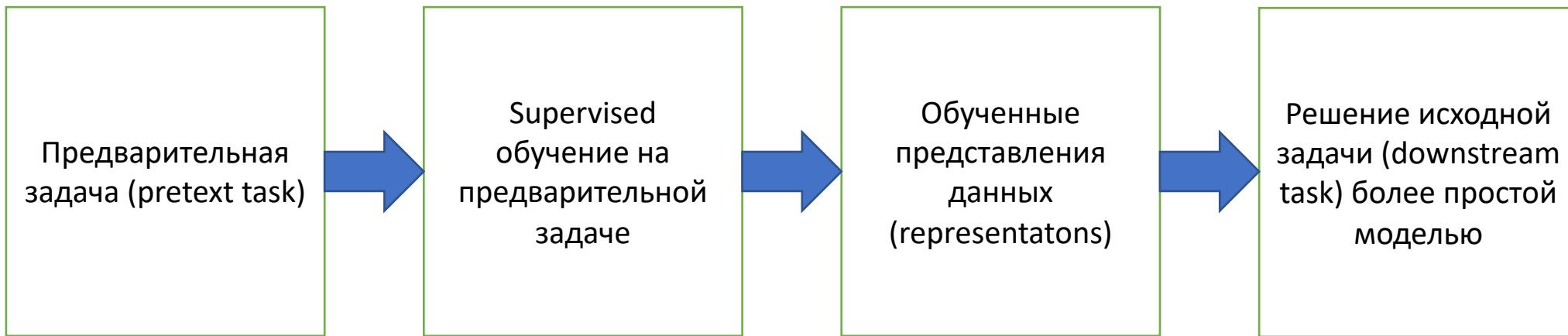
О чём речь?



- Supervised – по известным меткам учимся различать объекты
- Semi-Supervised – известны метки только для некоторых объектов
- Unsupervised – никакие метки неизвестны
- Self-Supervised – сами придумываем себе метки и учимся различать объекты

Зачем придумывать задачу самим?

Придумываем синтетическую задачу ради решения другой



Придумываем задачу и автоматически размечаем данные

Обучаем на полностью размеченных данных классическими методами

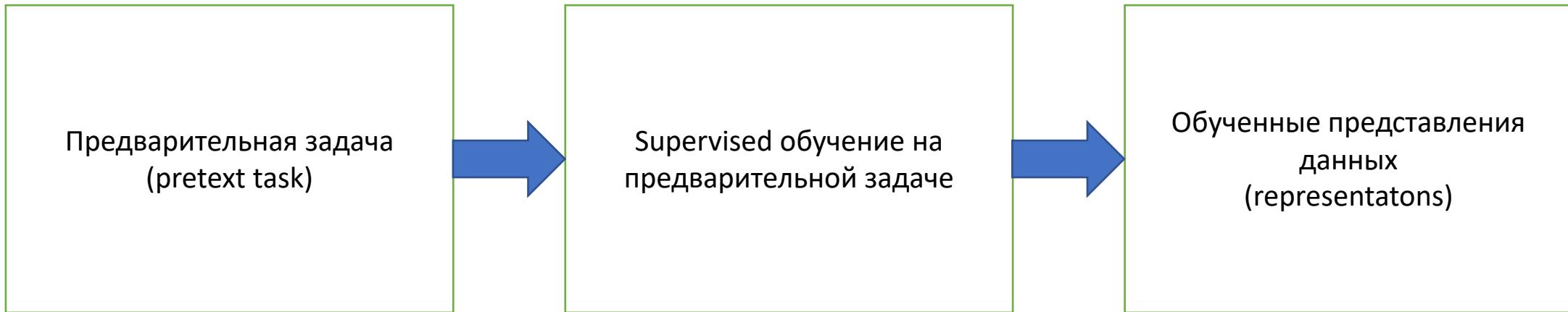
Получаем с помощью построенной модели представления для объектов

Используя полученные представления, обучаем более простую модель, решающую исходную задачу

Пример 0

Алгоритм word2vec – рассматривался на лекции Intro to DL

Задача: для каждого слова получить представление в виде вектора, отражающее смысл слова



Синтетическая задача: по контексту восстановить слово. Данные получаем из неразмеченного текста.

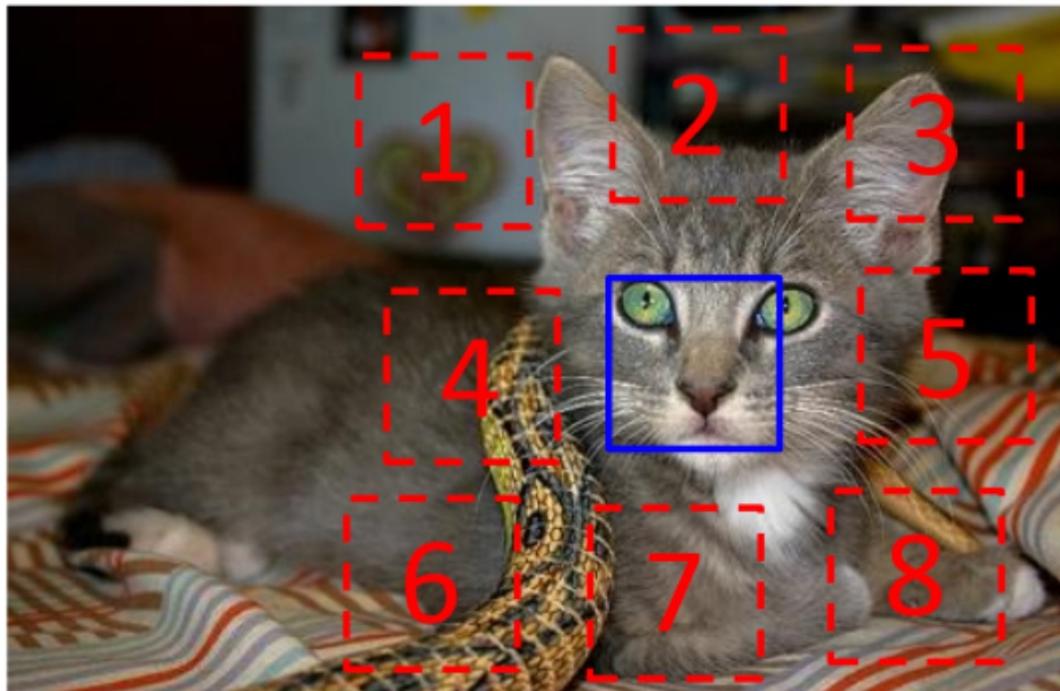
Обучаем нейронную сеть для решения синтетической задачи

Обученные представления данных (representations)

Верим, что близкие по смыслу слова получили близкие представления

Пример про компьютерное зрение

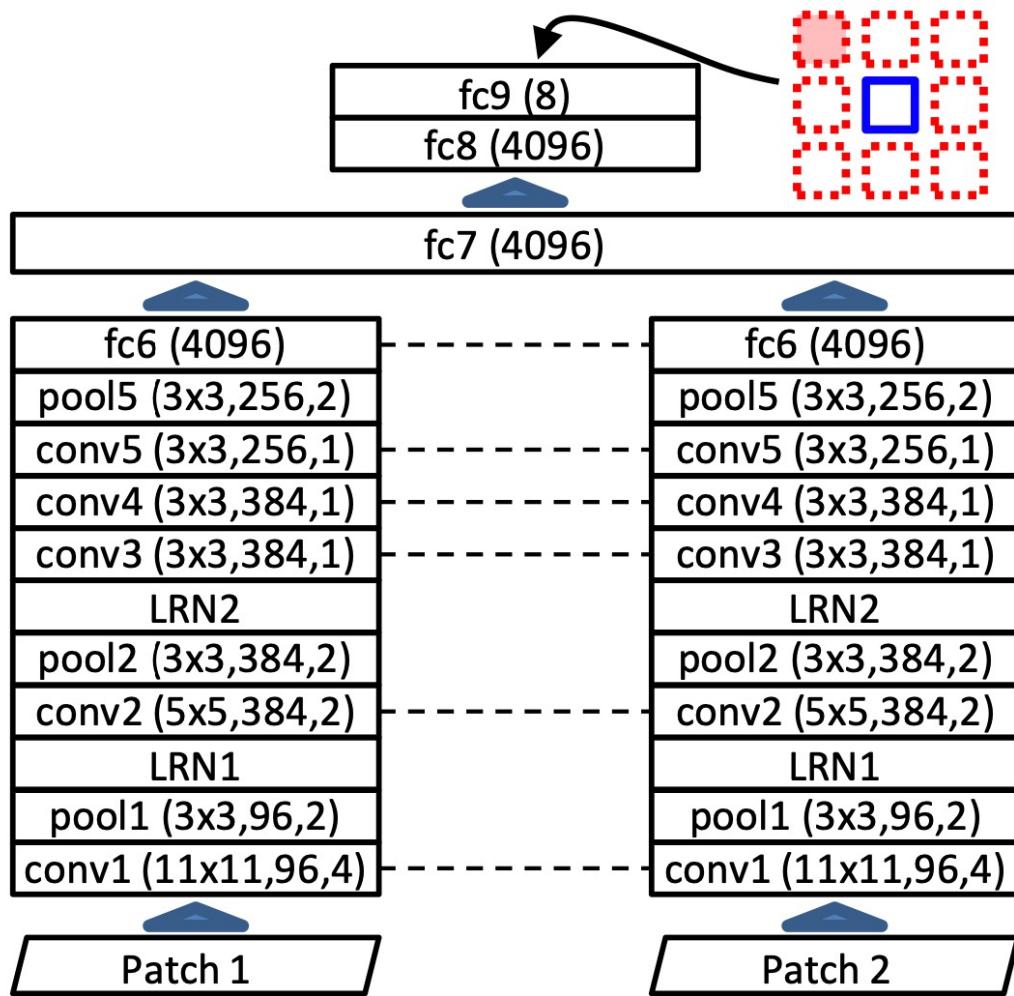
Предсказание пространственного контекста



- Хотим получить хорошие представления для изображений
- Берем неразмеченные изображения
- Ставим задачу — по двум патчам предсказать их относительное положение на изображении
- Можем сгенерировать сколько угодно размеченных данных по набору изображений

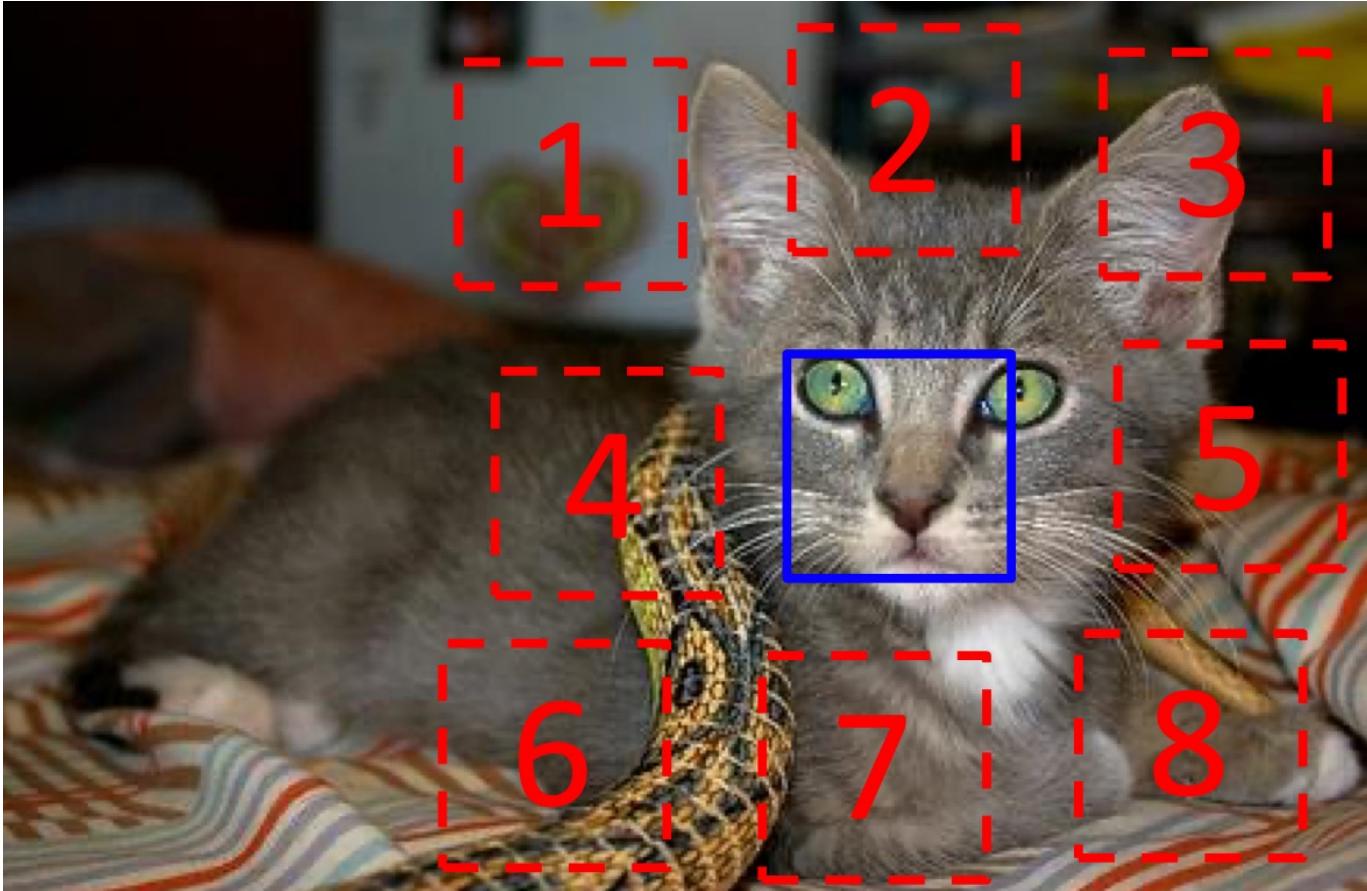
$$X = (\text{image}, \text{image}); Y = 3$$

Архитектура модели



- Сверточная нейронная сеть
- Блок со свертками, похожий на AlexNet
- ReLU-активация на всех слоях, кроме последнего, на последнем – Softmax
- Блоки для обоих патчей имеют одинаковые веса
- Каждый патч превращается в вектор размерности 4096
- Представление патча – выход слоя fc6

Аугментация данных – сдвиги



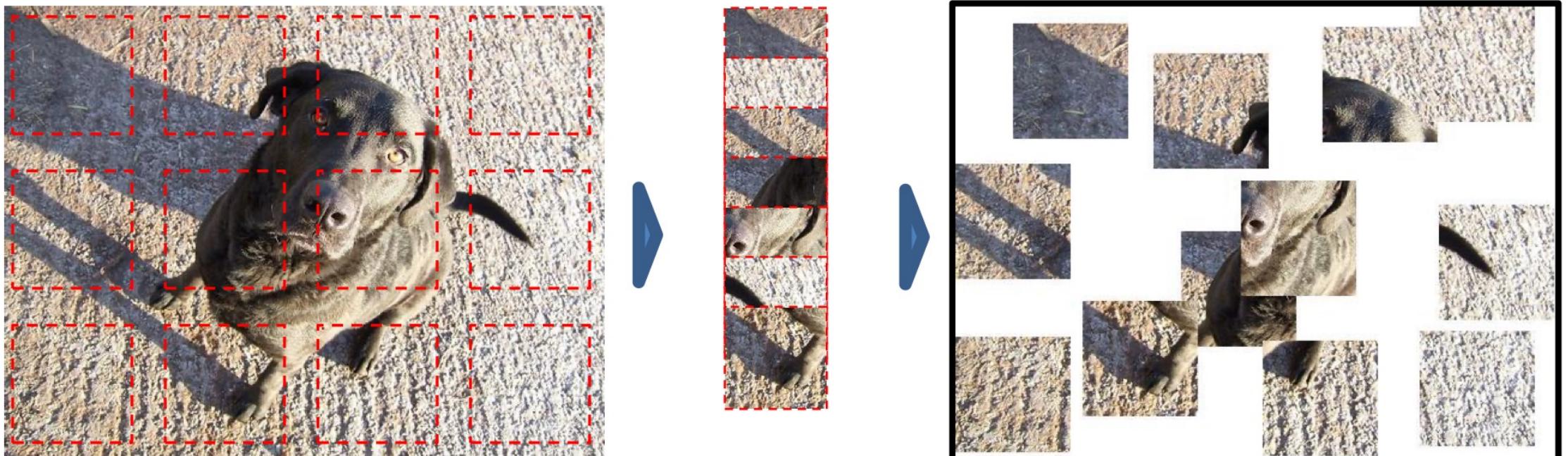
- Оставляем зазоры между патчами, чтобы избежать совмещения по краю патча
- Случайным образом сдвигаем патчи, чтобы избежать совмещения по длинным линиям на изображении

Аугментация данных – хроматическая аберрация

Проблема: оказывается, из-за искажений цвета в разных местах изображения, можно определять положение патчей на изображении, используя только цвета

Авторы статьи смогли обучить сеть, которая предсказывала абсолютное положение патча на картинке без использования какого-либо контекста

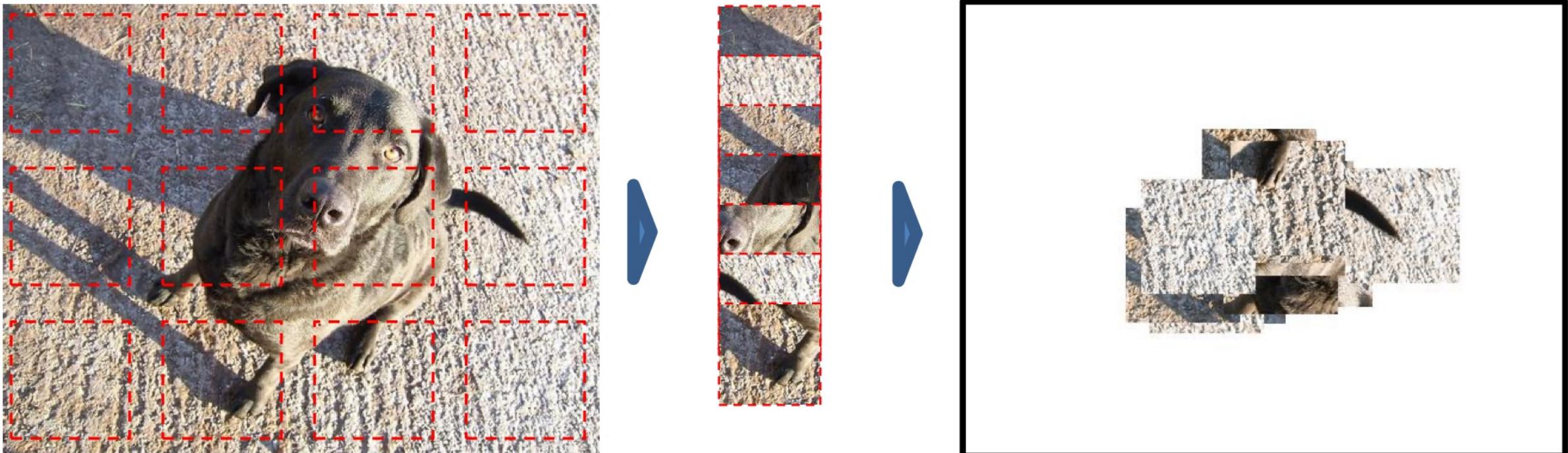
Этот эффект мешает модели запоминать реальную геометрию патчей



Борьба с хроматической аберрацией

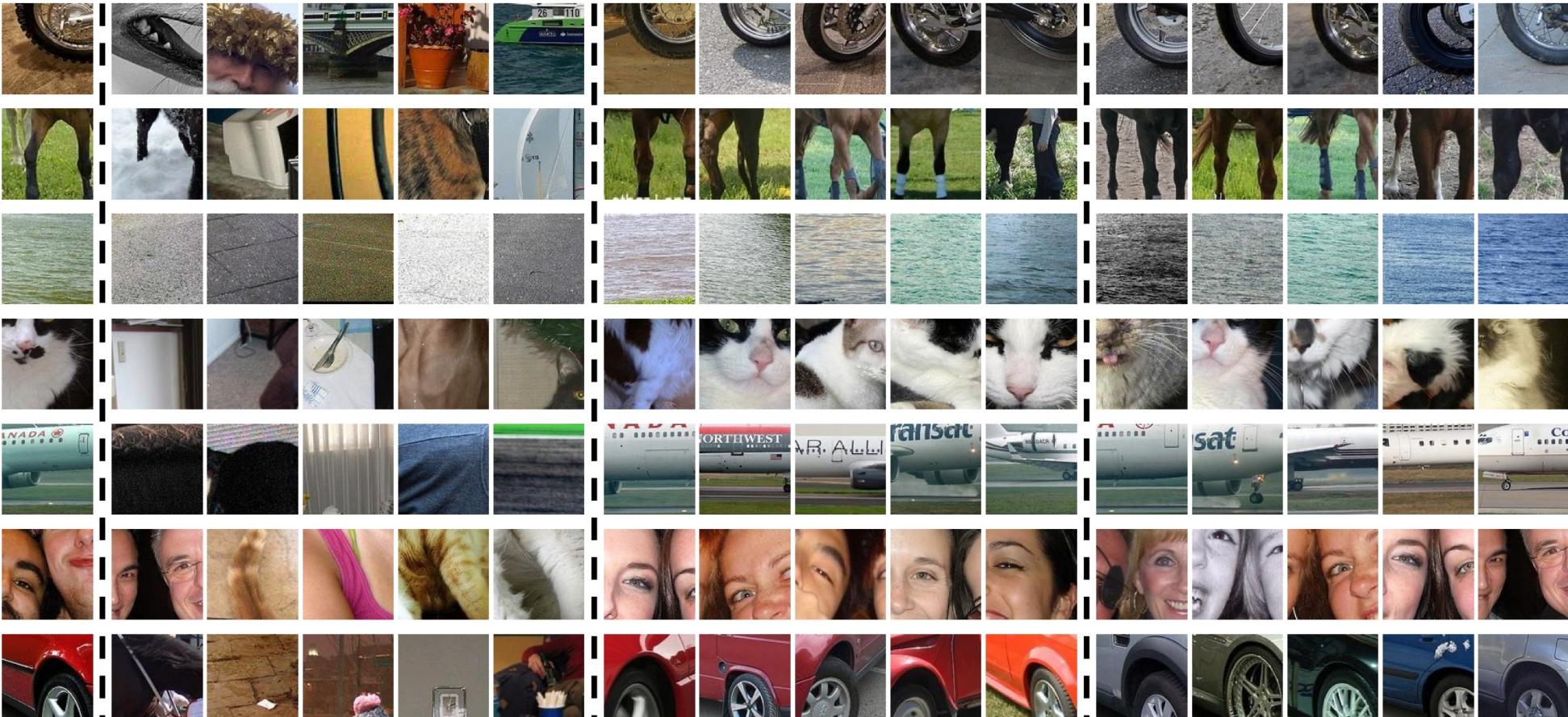
Заменяем случайные два из трех каналов на случайный шум

Теперь угадать положение патчей не удается



Какие представления получились?

Для каждого представления находим 5 ближайших соседей:



Вход

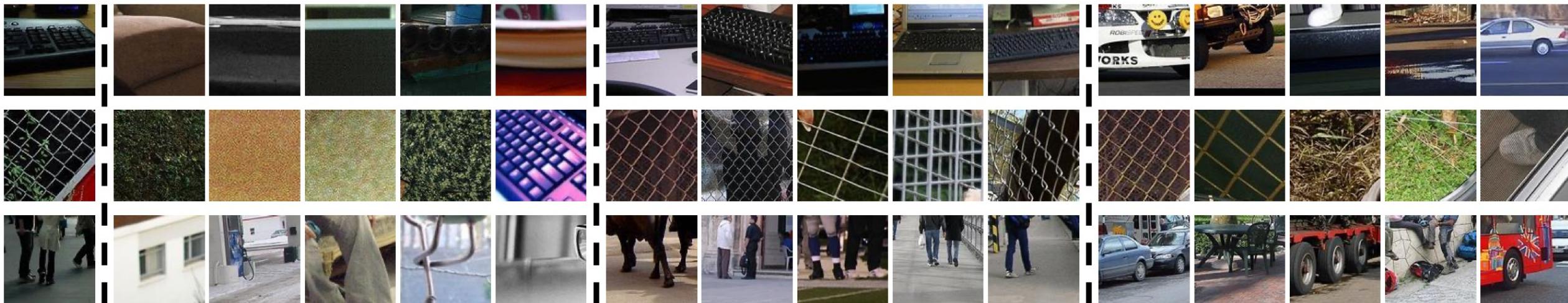
Случайно
инициализированная
сеть

AlexNet обученная на
данных ImageNet с
метками классов

Сеть из статьи,
обученная без меток
классов

Какие представления получились?

Однако, информация про классы в ImageNet все таки помогает с некоторыми примерами



Вход

Случайно
инициализированная
сеть

AlexNet обученная на
данных ImageNet с
метками классов

Сеть из статьи,
обученная без меток
классов

Применение к реальной задаче

Авторы статьи взяли датасет VOC-2007 — классификация объектов на изображениях, 20 классов и посмотрели на точность разных моделей

Наиболее интересные строки таблицы — Ours-rescale и Scratch-Ours

Ours-rescale — модель, которую мы рассмотрели, обученная на ImageNet без меток

Scratch-Ours — такая же модель, которую обучали сразу на VOC-2007 без выученных представлений

Видно, что выученные представления дали хороший прирост в точности

VOC-2007 Test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
DPM-v5[17]	33.2	60.3	10.2	16.1	27.3	54.3	58.2	23.0	20.0	24.1	26.7	12.7	58.1	48.2	43.2	12.0	21.1	36.1	46.0	43.5	33.7
[8] w/o context	52.6	52.6	19.2	25.4	18.7	47.3	56.9	42.1	16.6	41.4	41.9	27.7	47.9	51.5	29.9	20.0	41.1	36.4	48.6	53.2	38.5
Regionlets[58]	54.2	52.0	20.3	24.0	20.1	55.5	68.7	42.6	19.2	44.2	49.1	26.6	57.0	54.5	43.4	16.4	36.6	37.7	59.4	52.3	41.7
Scratch-R-CNN[2]	49.9	60.6	24.7	23.7	20.3	52.5	64.8	32.9	20.4	43.5	34.2	29.9	49.0	60.4	47.5	28.0	42.3	28.6	51.2	50.0	40.7
Scratch-Ours	52.6	60.5	23.8	24.3	18.1	50.6	65.9	29.2	19.5	43.5	35.2	27.6	46.5	59.4	46.5	25.6	42.4	23.5	50.0	50.6	39.8
Ours-projection	58.4	62.8	33.5	27.7	24.4	58.5	68.5	41.2	26.3	49.5	42.6	37.3	55.7	62.5	49.4	29.0	47.5	28.4	54.7	56.8	45.7
Ours-color-dropping	60.5	66.5	29.6	28.5	26.3	56.1	70.4	44.8	24.6	45.5	45.4	35.1	52.2	60.2	50.0	28.1	46.7	42.6	54.8	58.6	46.3
Ours-Yahoo100m	56.2	63.9	29.8	27.8	23.9	57.4	69.8	35.6	23.7	47.4	43.0	29.5	52.9	62.0	48.7	28.4	45.1	33.6	49.0	55.5	44.2
ImageNet-R-CNN[21]	64.2	69.7	50	41.9	32.0	62.6	71.0	60.7	32.7	58.5	46.5	56.1	60.6	66.8	54.2	31.5	52.8	48.9	57.9	64.7	54.2
K-means-rescale [31]	55.7	60.9	27.9	30.9	12.0	59.1	63.7	47.0	21.4	45.2	55.8	40.3	67.5	61.2	48.3	21.9	32.8	46.9	61.6	51.7	45.6
Ours-rescale [31]	61.9	63.3	35.8	32.6	17.2	68.0	67.9	54.8	29.6	52.4	62.9	51.3	67.1	64.3	50.5	24.4	43.7	54.9	67.1	52.7	51.1
ImageNet-rescale [31]	64.0	69.6	53.2	44.4	24.9	65.7	69.6	69.2	28.9	63.6	62.8	63.9	73.3	64.6	55.8	25.7	50.5	55.4	69.3	56.4	56.5
VGG-K-means-rescale	56.1	58.6	23.3	25.7	12.8	57.8	61.2	45.2	21.4	47.1	39.5	35.6	60.1	61.4	44.9	17.3	37.7	33.2	57.9	51.2	42.4
VGG-Ours-rescale	71.1	72.4	54.1	48.2	29.9	75.2	78.0	71.9	38.3	60.5	62.3	68.1	74.3	74.2	64.8	32.6	56.5	66.4	74.0	60.3	61.7
VGG-ImageNet-rescale	76.6	79.6	68.5	57.4	40.8	79.9	78.4	85.4	41.7	77.0	69.3	80.1	78.6	74.6	70.1	37.5	66.0	67.5	77.4	64.9	68.6

Применение к реальной задаче

Авторы статьи взяли датасет NYUv2 — набор цветных изображений и карт глубин и пытались предсказывать угол нормали к поверхности в точке

Модель, обученная с использованием представлений, выученных на ImageNet без меток получила точность значительно лучше, чем модель, которая обучалась сразу для этой задачи

Это показывает, что выученные представления отражают геометрию объектов

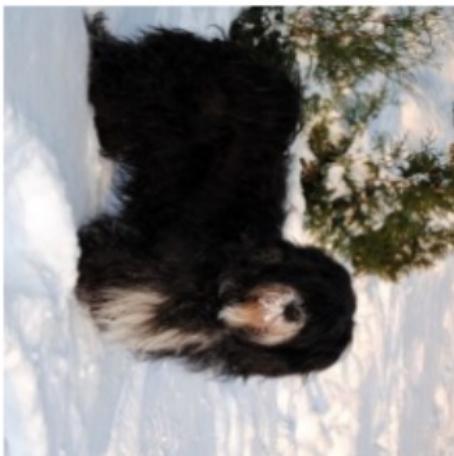
	Lower Better		Higher Better		
	Mean	Median	11.25°	22.5°	30°
Scratch	38.6	26.5	33.1	46.8	52.5
Unsup. Tracking [57]	34.2	21.9	35.7	50.6	57.0
Ours	33.2	21.3	36.0	51.2	57.8
ImageNet Labels	33.3	20.8	36.7	51.7	58.1

Какие еще задачи можно взять?

Предсказание поворота



90° rotation



270° rotation



180° rotation



0° rotation

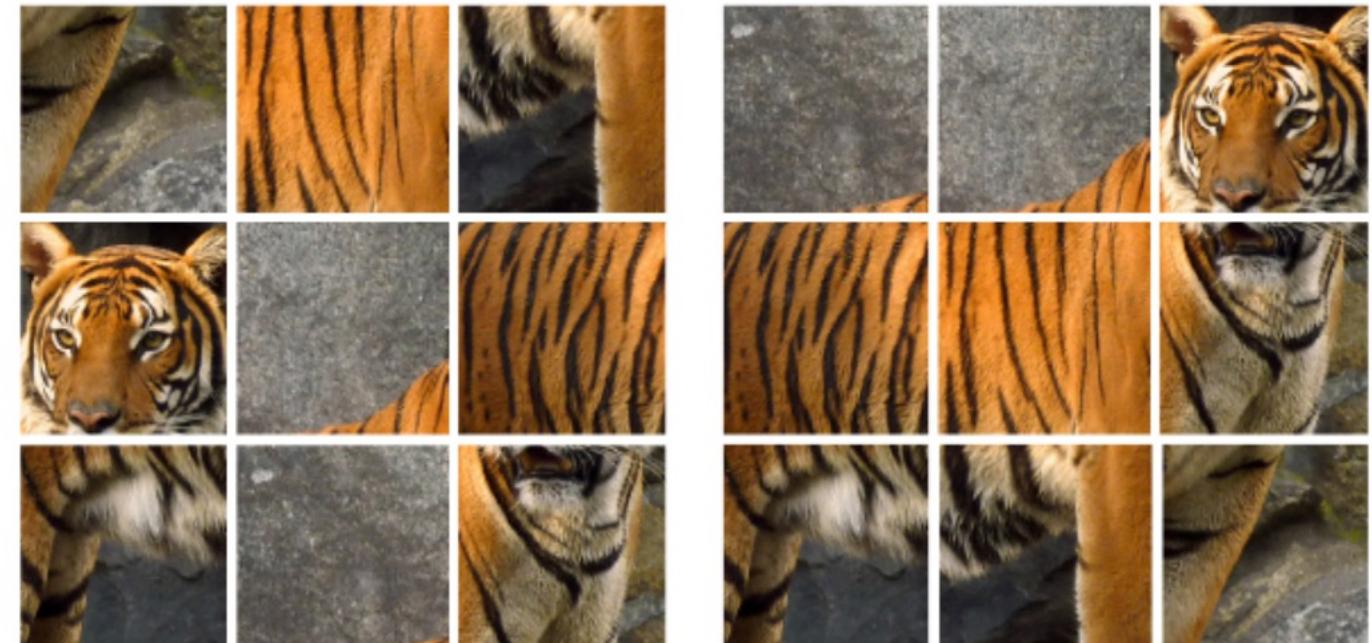
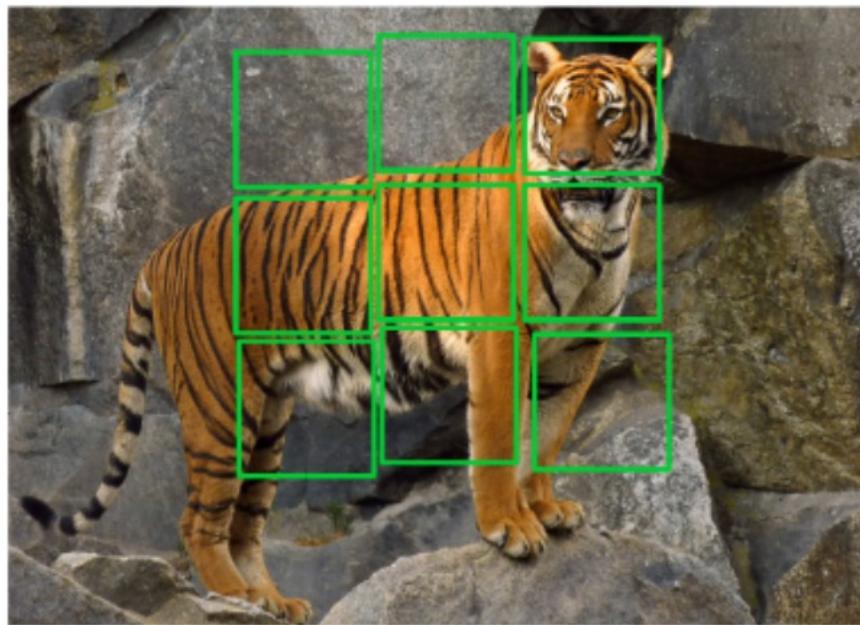


270° rotation

- Берем неразмеченные изображения и случайным образом поворачиваем
- Запоминаем, как повернули и обучаем нейросеть, определять один из четырех поворотов

Какие еще задачи можно взять?

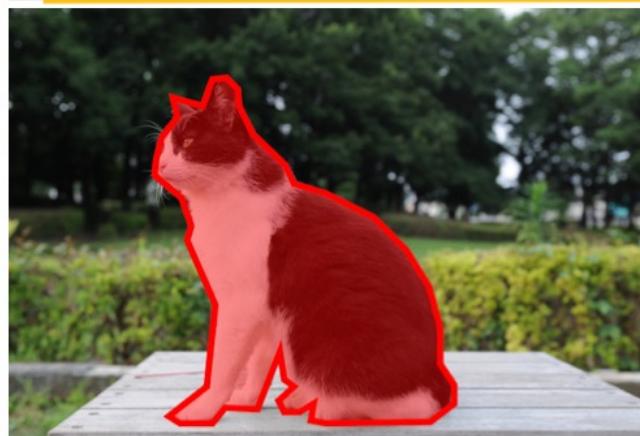
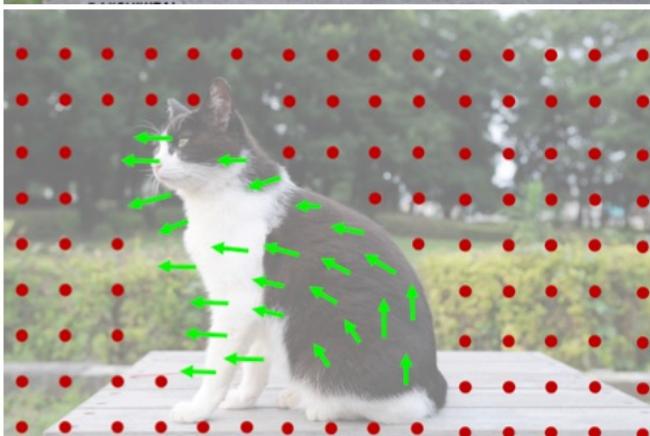
Сборка паззла



- Выбираем несколько перестановок девяти объектов
- Переставляем патчи с изображения случайной перестановкой
- Обучаем нейросеть предсказывать, какую именно перестановку применили

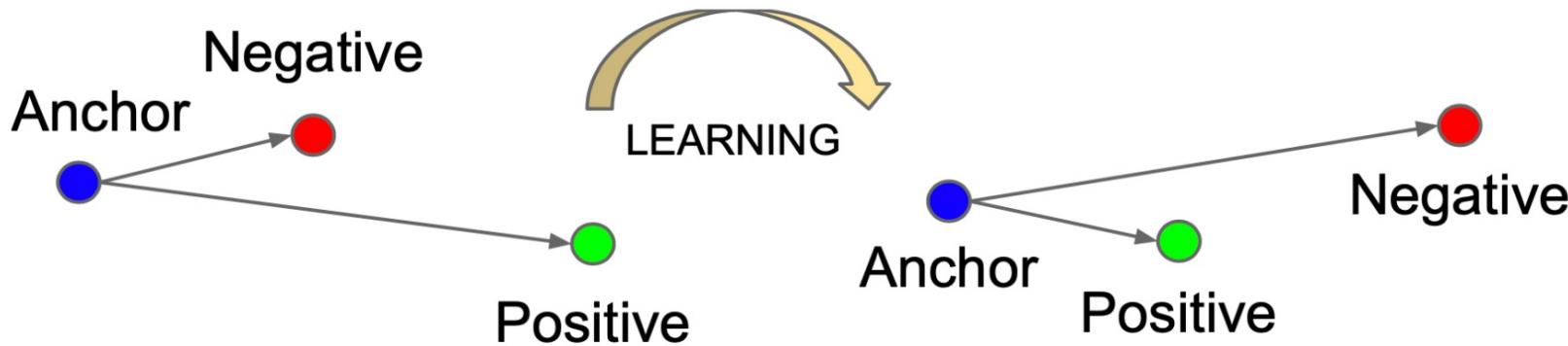
Какие еще задачи можно взять?

Выделение объекта по движению на видео



- Сравниваем последовательные кадры видео, считаем optical flow для каждого пикселя
- Пытаемся предсказывать, сдвинется ли пиксель в следующем кадре
- Таким образом учимся выделять движущиеся объекты на картинке

Contrastive learning



- Сравнительное обучение
- Подается опорный объект, похожий и непохожий на него
- Модель учится, чтобы вектора представлений похожих элементов были похожи, а разных элементов – различались
- Похожи изображения из одного класса или аугментированные патчи из одного изображения
- Не похожи – из разных классов или просто разные изображения

Аугментация

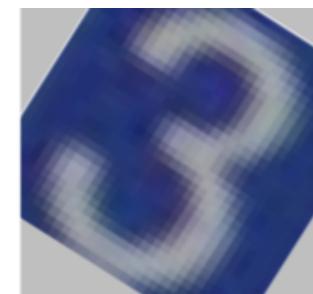
- Обрезание
- Изменение размера
- Поворот
- Дисторсия цвета
- Гауссовское размытие



a)



b)



c)



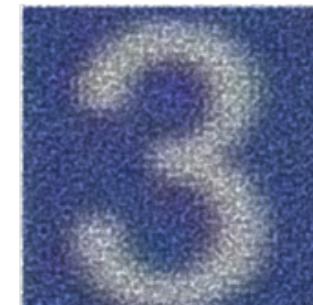
d)



e)



f)



g)

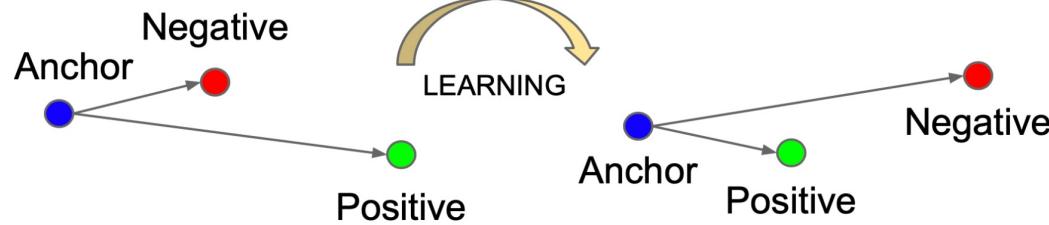


h)

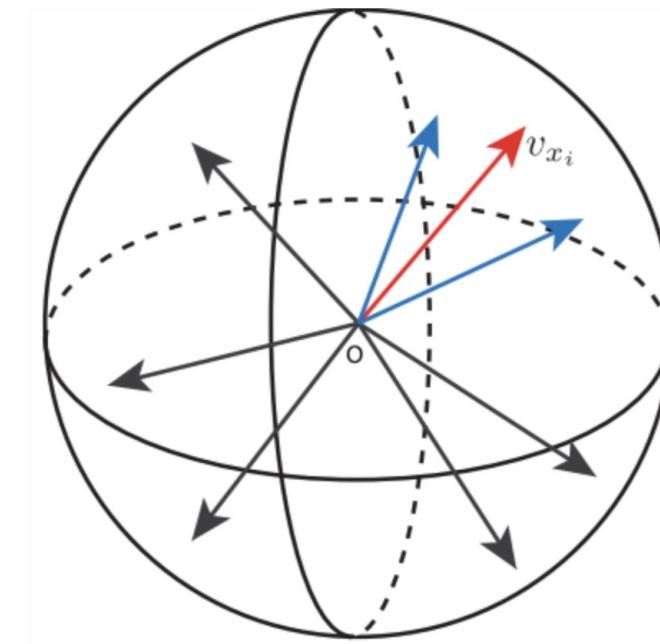
ФУНКЦИИ ПОТЕРЬ

Что значит объекты похожи/различаются?

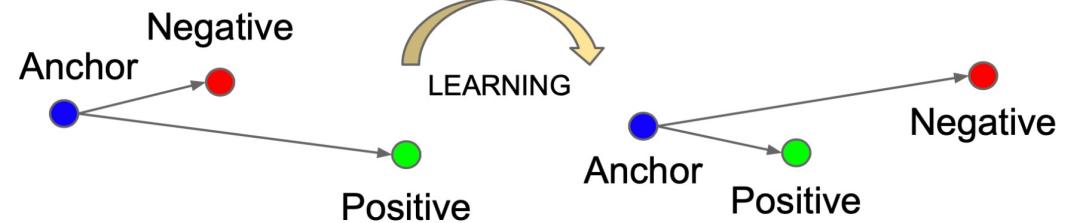
+ близко, - далеко



+ коллинеарен, - ортогонален



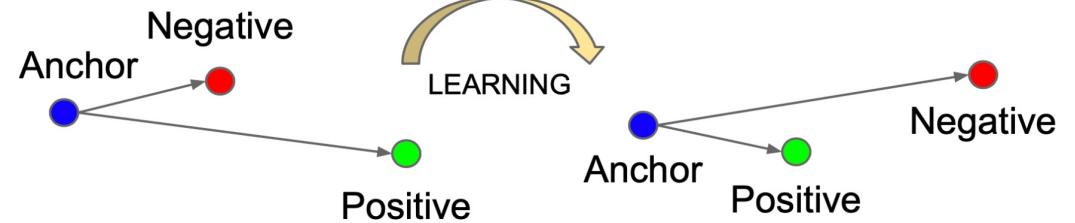
Contrastive loss



$$\mathcal{L}_{\text{cont}}(\mathbf{x}_i, \mathbf{x}_j, \theta) = \mathbb{1}[y_i = y_j] \|f_\theta(\mathbf{x}_i) - f_\theta(\mathbf{x}_j)\|_2^2 + \mathbb{1}[y_i \neq y_j] \max(0, \epsilon - \|f_\theta(\mathbf{x}_i) - f_\theta(\mathbf{x}_j)\|_2)^2$$

- Если из одного класса – минимизируем расстояние
- Если из разных – расстояние хотя бы ϵ

Triplet loss



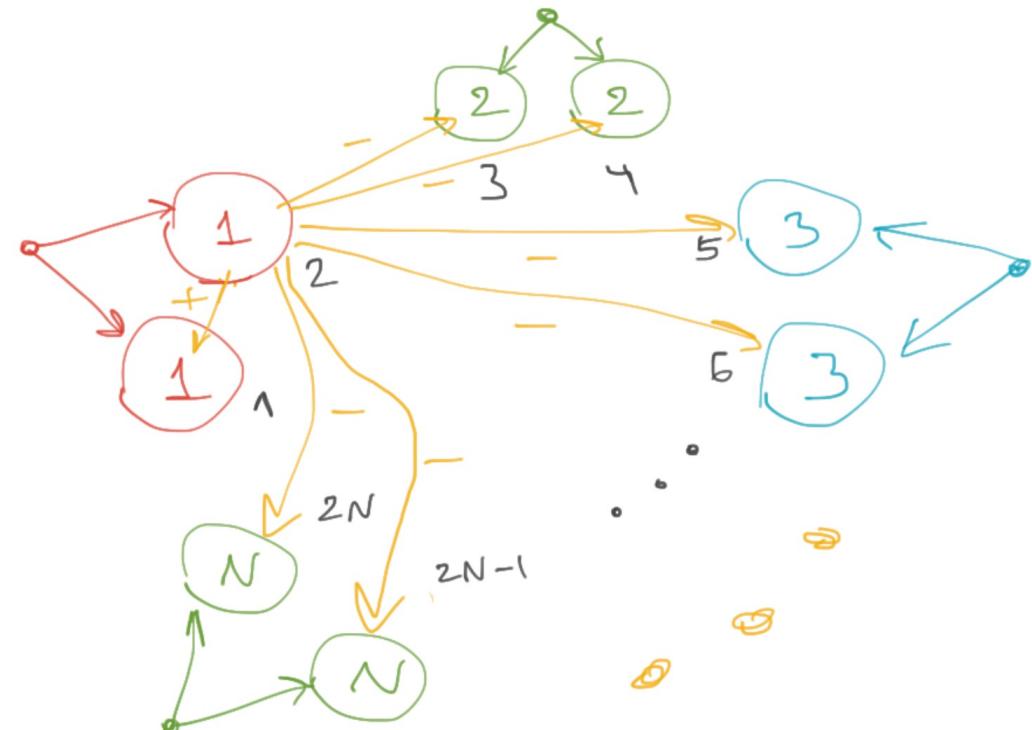
$$\mathcal{L}_{\text{triplet}}(\mathbf{x}, \mathbf{x}^+, \mathbf{x}^-) = \sum_{\mathbf{x} \in \mathcal{X}} \max \left(0, \|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}^+)\|_2^2 - \|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}^-)\|_2^2 + \epsilon \right)$$

- Отрицательный объект дальше положительного хотя бы на ϵ

SimCLR

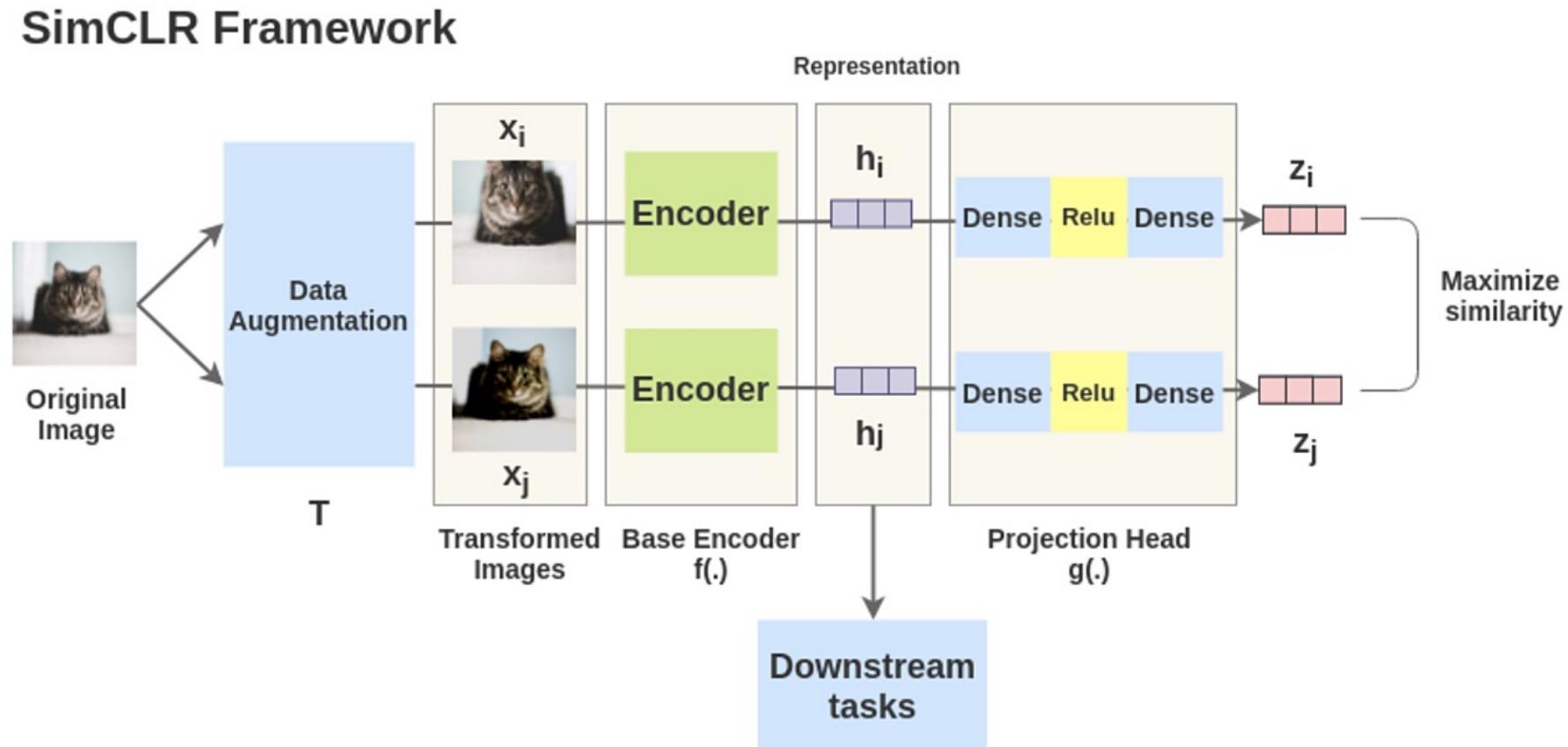
A Simple Framework for Contrastive Learning of Visual Representations

1. Приходит новый батч (N элементов)
2. Для каждого объекта создаются две аугментированные копии
3. Получаются их представления
4. Для каждого объекта есть 2 положительных представления и $2N-2$ отрицательных
5. Считается ошибка (учитывает схожесть положительных представлений и отличие отрицательных)
6. Обновляются веса



SimCLR

Как получаются представления?

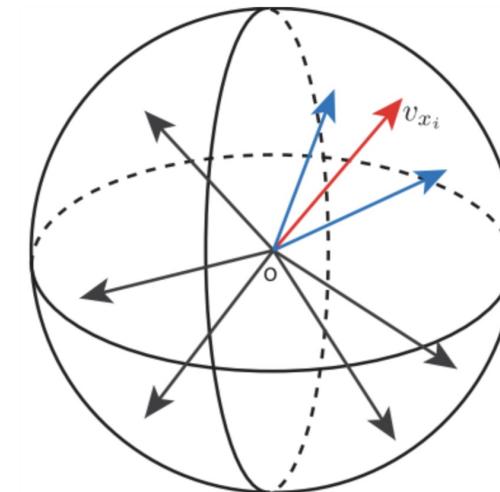
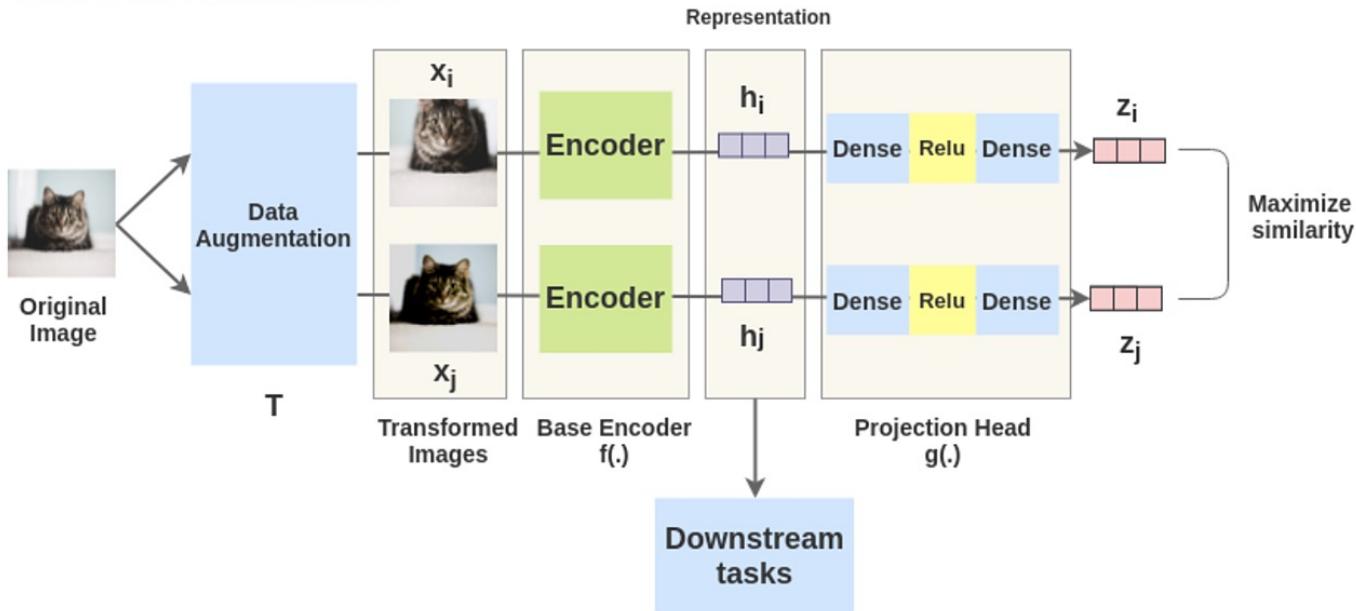


SimCLR

Схожесть представлений –
косинусное расстояние

$$\text{sim}(z_i, z_j) = \cos(\theta) = \frac{z_i \cdot z_j}{\|z_i\| \|z_j\|}$$

SimCLR Framework



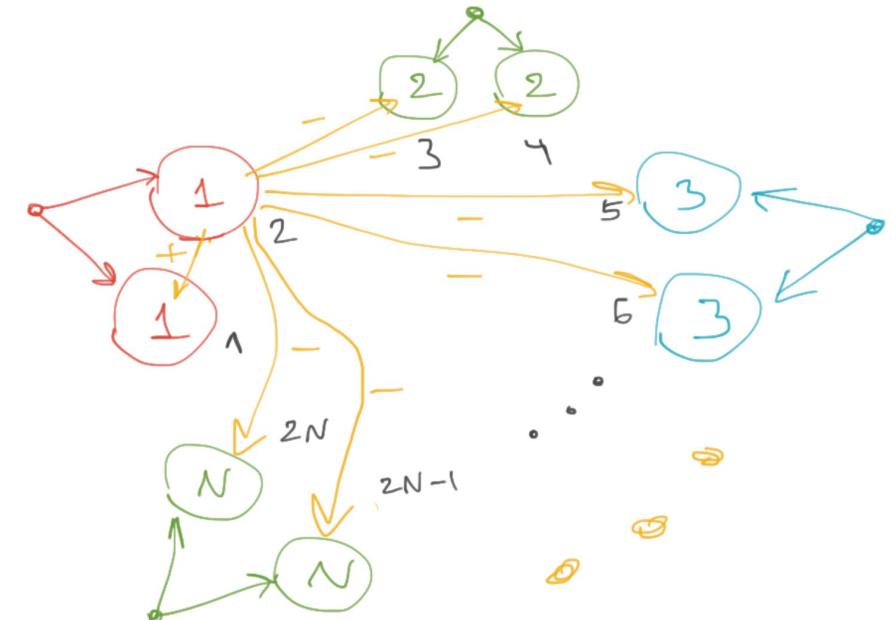
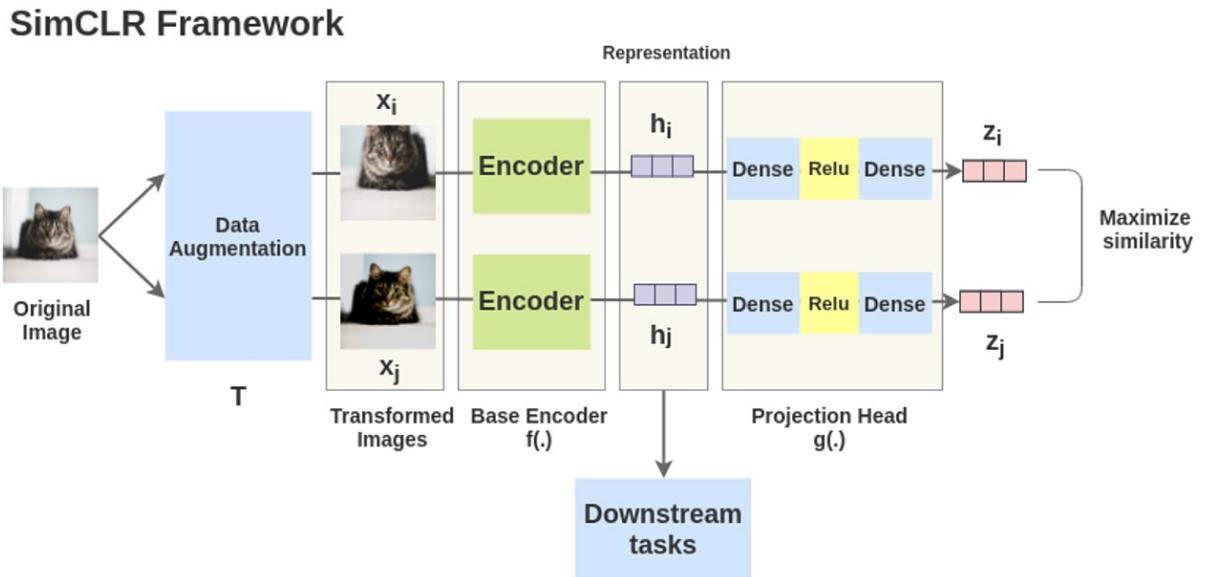
SimCLR

Функция потерь – смесь
SoftMax и CrossEntropy

$$\text{sim}(z_i, z_j) = \cos(\theta) = \frac{z_i \cdot z_j}{|z_i||z_j|}$$

$$\mathcal{L}_{\text{SimCLR}}^{(i,j)} = -\log \frac{\exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)}$$

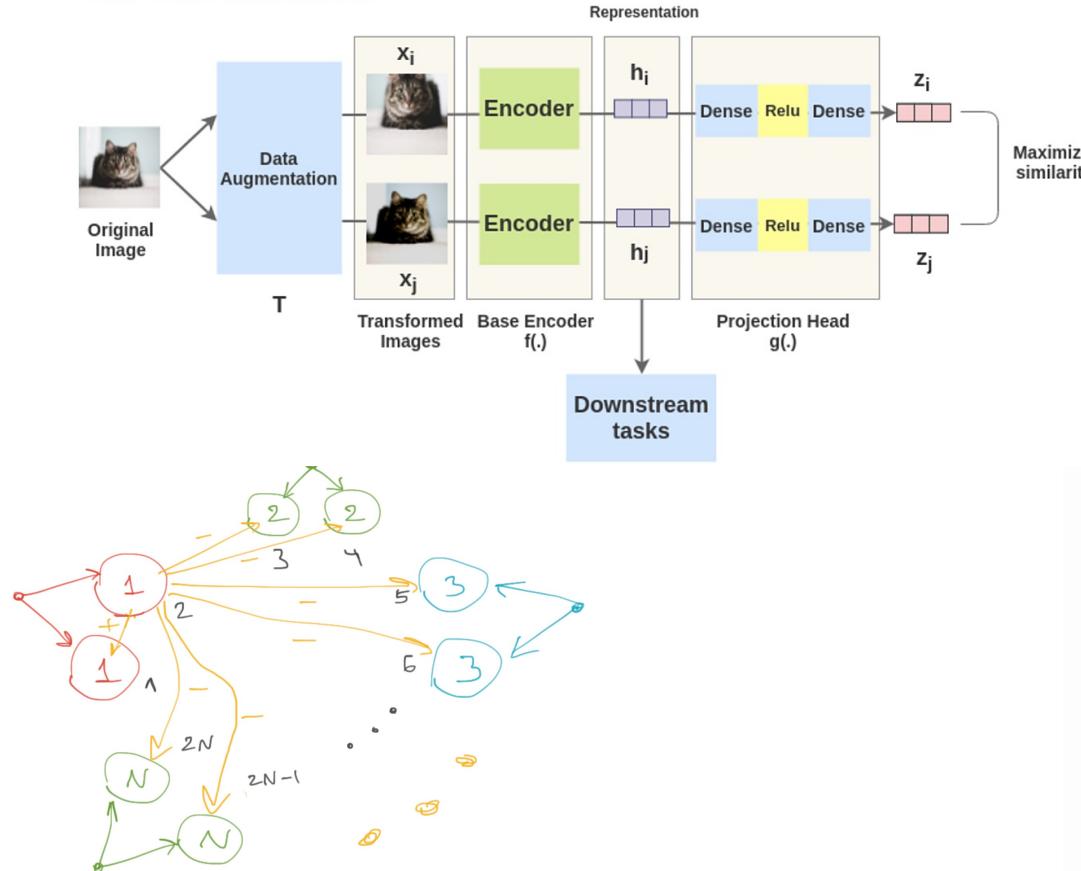
$$\mathcal{L}_{\text{SimCLR}} = \frac{1}{2N} \sum_{k=1}^N \left(\mathcal{L}_{\text{SimCLR}}^{(2k-1, 2k)} + \mathcal{L}_{\text{SimCLR}}^{(2k, 2k-1)} \right)$$



SimCLR

АЛГОРИТМ

SimCLR Framework

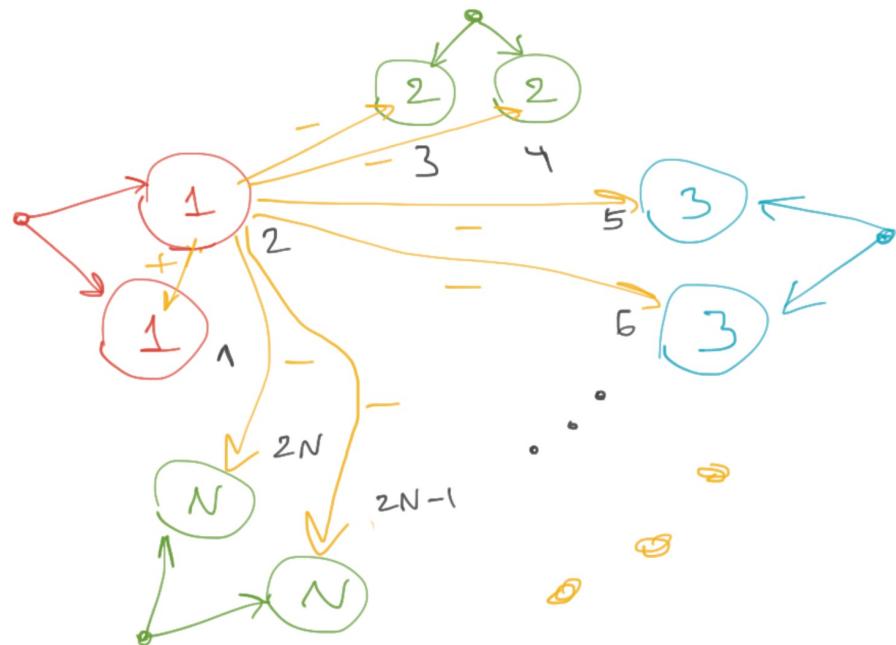


Algorithm 1 SimCLR's main learning algorithm.

```
input: batch size  $N$ , constant  $\tau$ , structure of  $f, g, \mathcal{T}$ .
for sampled minibatch  $\{x_k\}_{k=1}^N$  do
    for all  $k \in \{1, \dots, N\}$  do
        draw two augmentation functions  $t \sim \mathcal{T}, t' \sim \mathcal{T}$ 
        # the first augmentation
         $\tilde{x}_{2k-1} = t(x_k)$ 
         $\mathbf{h}_{2k-1} = f(\tilde{x}_{2k-1})$  # representation
         $\mathbf{z}_{2k-1} = g(\mathbf{h}_{2k-1})$  # projection
        # the second augmentation
         $\tilde{x}_{2k} = t'(x_k)$ 
         $\mathbf{h}_{2k} = f(\tilde{x}_{2k})$  # representation
         $\mathbf{z}_{2k} = g(\mathbf{h}_{2k})$  # projection
    end for
    for all  $i \in \{1, \dots, 2N\}$  and  $j \in \{1, \dots, 2N\}$  do
         $s_{i,j} = \mathbf{z}_i^\top \mathbf{z}_j / (\|\mathbf{z}_i\| \|\mathbf{z}_j\|)$  # pairwise similarity
    end for
    define  $\ell(i, j)$  as  $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$ 
     $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^N [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$ 
    update networks  $f$  and  $g$  to minimize  $\mathcal{L}$ 
end for
return encoder network  $f(\cdot)$ , and throw away  $g(\cdot)$ 
```

SimCLR

Проблема – нужно много отрицательных примеров (большие батчи)



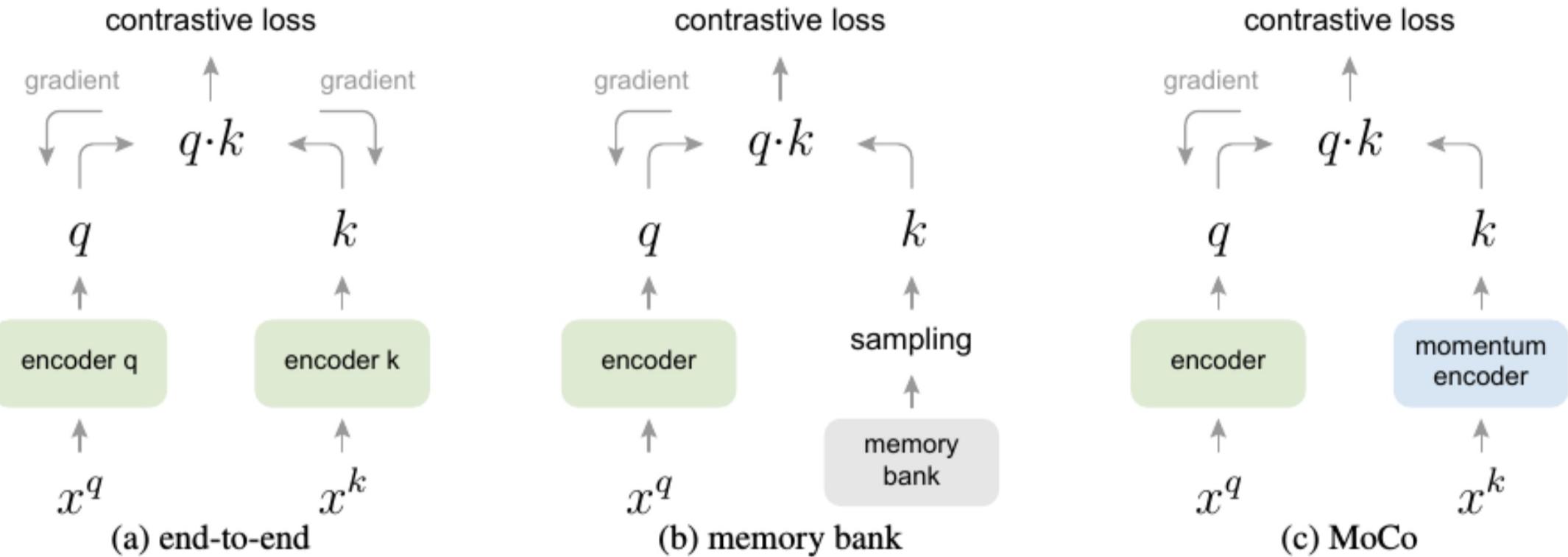
$$\text{sim}(z_i, z_j) = \cos(\theta) = \frac{z_i \cdot z_j}{|z_i||z_j|}$$

$$\mathcal{L}_{\text{SimCLR}}^{(i,j)} = -\log \frac{\exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)}$$

$$\mathcal{L}_{\text{SimCLR}} = \frac{1}{2N} \sum_{k=1}^N \left(\mathcal{L}_{\text{SimCLR}}^{(2k-1, 2k)} + \mathcal{L}_{\text{SimCLR}}^{(2k, 2k-1)} \right)$$

Идея

Будем брать представления из старых батчей как отрицательные

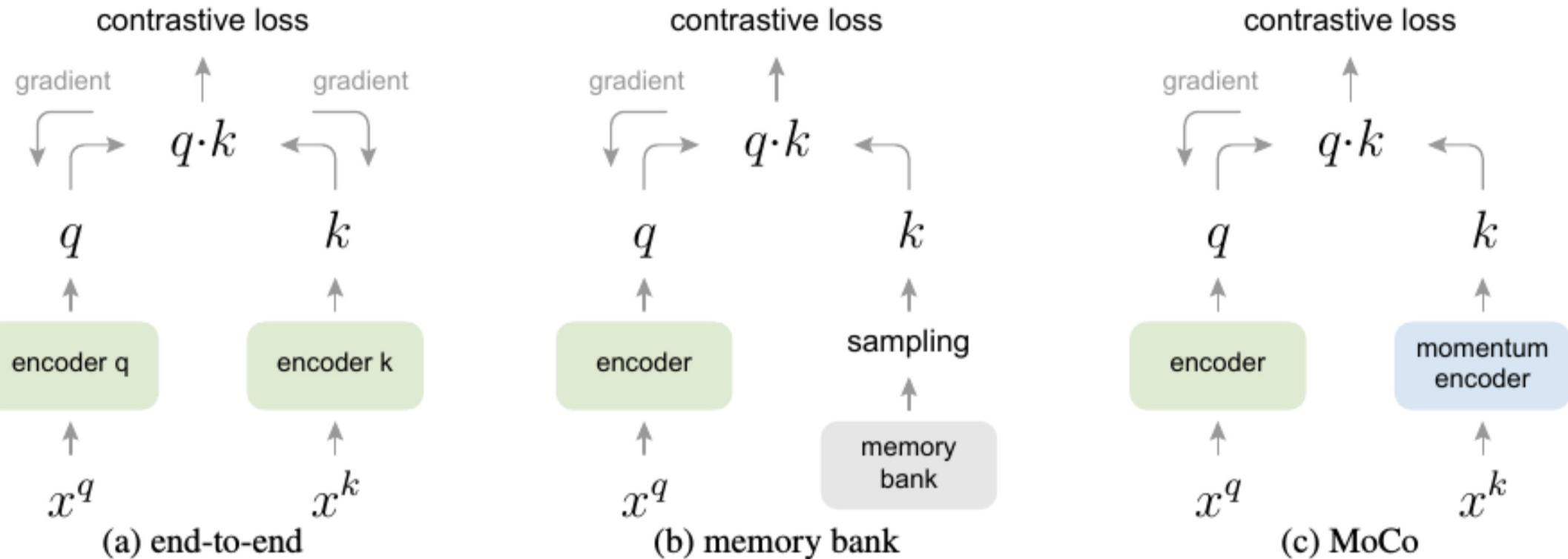


Проблема – с каждым шагом модель резко меняется

(представления не консистентны)

Идея

Ввести момент для весов энкодера ключей

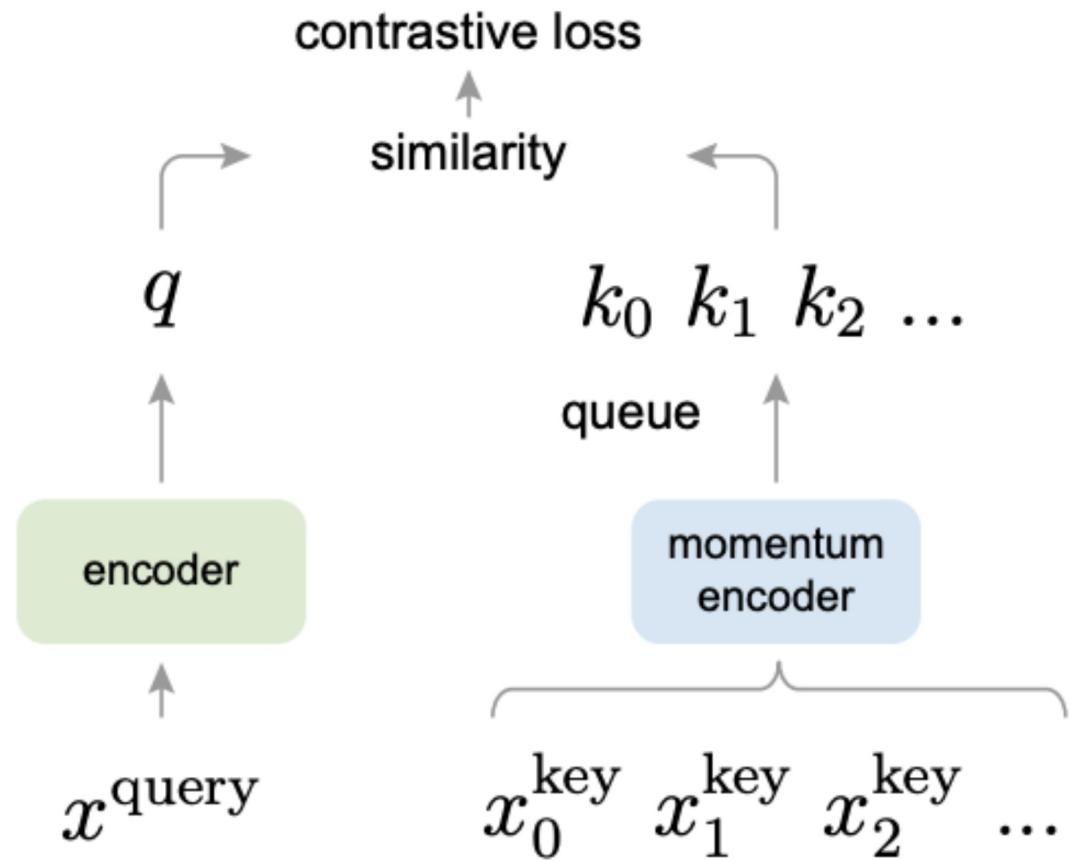


Момент – порядка 0,99. Веса очень плавно изменяются.

MoCo

Momentum Contrast

1. Приходит батч
2. Для каждого объекта создаются две аугментированные копии (x^k и x^q)
3. x^q пропускается через обычный энкодер
4. x^k пропускается через momentum encoder
5. $f_k(x^k)$ берется в качестве положительных объектов
6. Старые значения из очереди берутся как отрицательные
7. Считываются потери, обновляются веса энкодера
8. Веса энкодера обновляют веса momentum encoder
9. $f_k(x^k)$ добавляются в очередь

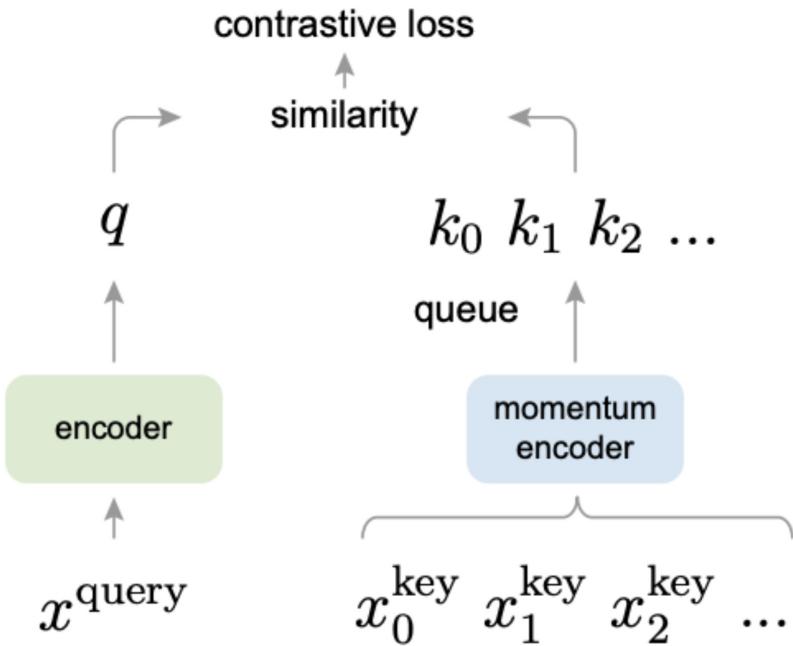


$$\mathcal{L}_{\text{MoCo}} = -\log \frac{\exp(\mathbf{q} \cdot \mathbf{k}^+/\tau)}{\sum_{i=1}^N \exp(\mathbf{q} \cdot \mathbf{k}_i/\tau)}$$

$$\theta_k \leftarrow m\theta_k + (1 - m)\theta_q$$

MoCo

Алгоритм



Algorithm 1 Pseudocode of MoCo in a PyTorch-like style.

```
# f_q, f_k: encoder networks for query and key
# queue: dictionary as a queue of K keys (CxK)
# m: momentum
# t: temperature

f_k.params = f_q.params # initialize
for x in loader: # load a minibatch x with N samples
    x_q = aug(x) # a randomly augmented version
    x_k = aug(x) # another randomly augmented version

    q = f_q.forward(x_q) # queries: NxC
    k = f_k.forward(x_k) # keys: NxC
    k = k.detach() # no gradient to keys

    # positive logits: Nx1
    l_pos = bmm(q.view(N,1,C), k.view(N,C,1))

    # negative logits: NxK
    l_neg = mm(q.view(N,C), queue.view(C,K))

    # logits: Nx(1+K)
    logits = cat([l_pos, l_neg], dim=1)

    # contrastive loss, Eqn.(1)
    labels = zeros(N) # positives are the 0-th
    loss = CrossEntropyLoss(logits/t, labels)

    # SGD update: query network
    loss.backward()
    update(f_q.params)

    # momentum update: key network
    f_k.params = m*f_k.params+(1-m)*f_q.params

    # update dictionary
    enqueue(queue, k) # enqueue the current minibatch
    dequeue(queue) # dequeue the earliest minibatch
```

bmm: batch matrix multiplication; mm: matrix multiplication; cat: concatenation.