Self-Supervised Representation Learning for Images

Иванов Данила,
Халматова Мадина.

Higher School of Economics
Faculty of Computer science

Types of Machine learning

	+	−
• Supervised	**the most accurate** type. simple learning process.	limited number **labelled** datasets.
• Unsupervised	use large collections of **unlabeled data.**	usually works much **less efficiently.**

Types of Machine learning

	+	−
• Supervised	**the most accurate** type. simple learning process.	limited number **labelled** datasets.
• Unsupervised	use large collections of **unlabeled data.**	usually works much **less efficiently.**
• Semi-Supervised	**bigger datasets** than SL, **higher accuracy** than UsL.	still need **human-labeled** data.

Types of Machine learning

	+	−
• Supervised	**the most accurate** type. simple learning process.	limited number **labelled** datasets.
• Unsupervised	use large collections of **unlabeled data.**	usually works much **less efficiently.**
• Semi-Supervised	**bigger datasets** than SL, **higher accuracy** than UsL.	still need **human-labeled** data.
• **Self-Supervised.**	?	?

What Self-Supervised Learning is?

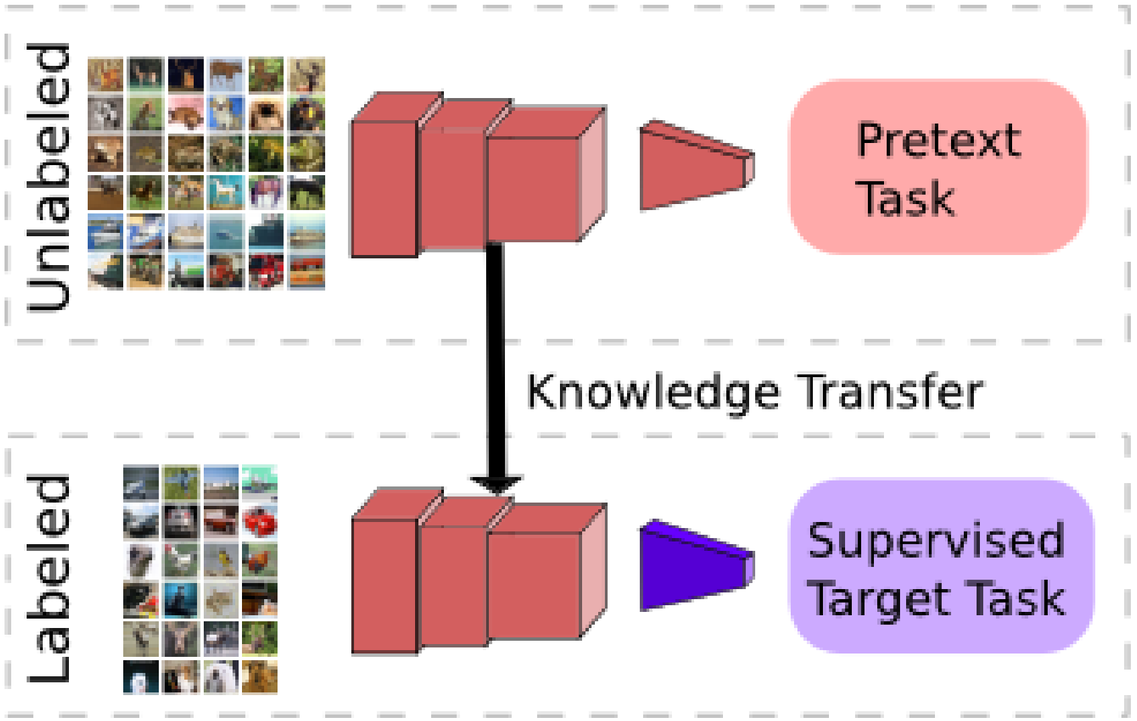
Goal: Downstream task - main supervised-learning task that utilizes a pre-trained model or component.

Motivation: Automatically label unlabeled data. Learn high-quality representation.

Idea: To solve downstream task we will solve **pretext tasks** as a supervised learning task and learn intermediate representation with the expectation that this representation can carry good semantic or structural meanings.

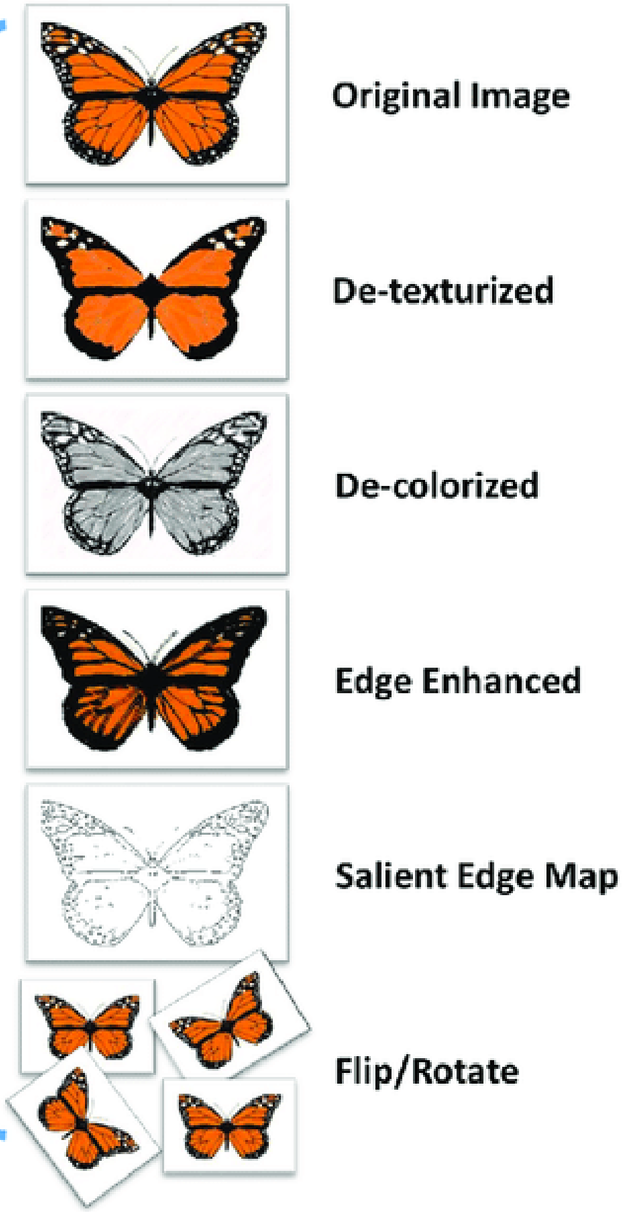
What Self-Supervised Learning is?

- **Pretext task** - a synthetic task with supervised loss function.
  Labels for pretext tasks come with the data for free!
  - Rotation
  - Exemplar-CNN
  - Relative position
  - Jigsaw puzzle
  - Contrastive Learning



Augmentations:

augmentations



- **simple** enough to keep semantic meaning unchanged.
- **complicated** enough to avoid catching trivial signals.

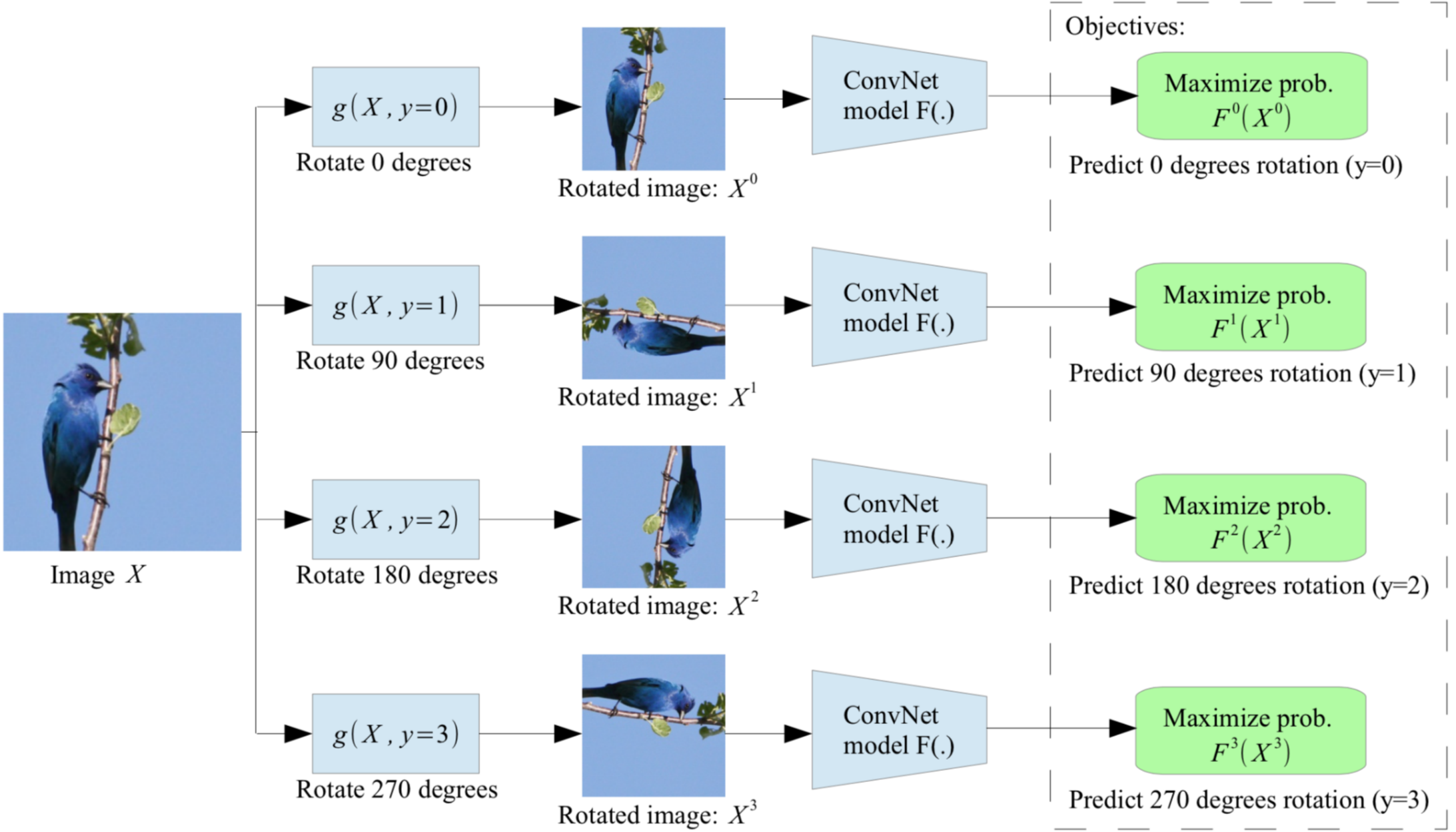
Rotation

The pretext task: Each input image is first rotated by a multiple of 90 at random, corresponding to [0, 90, 180, 270]. The model is trained to predict which rotation has been applied, thus a 4-class classification problem.

Rotation

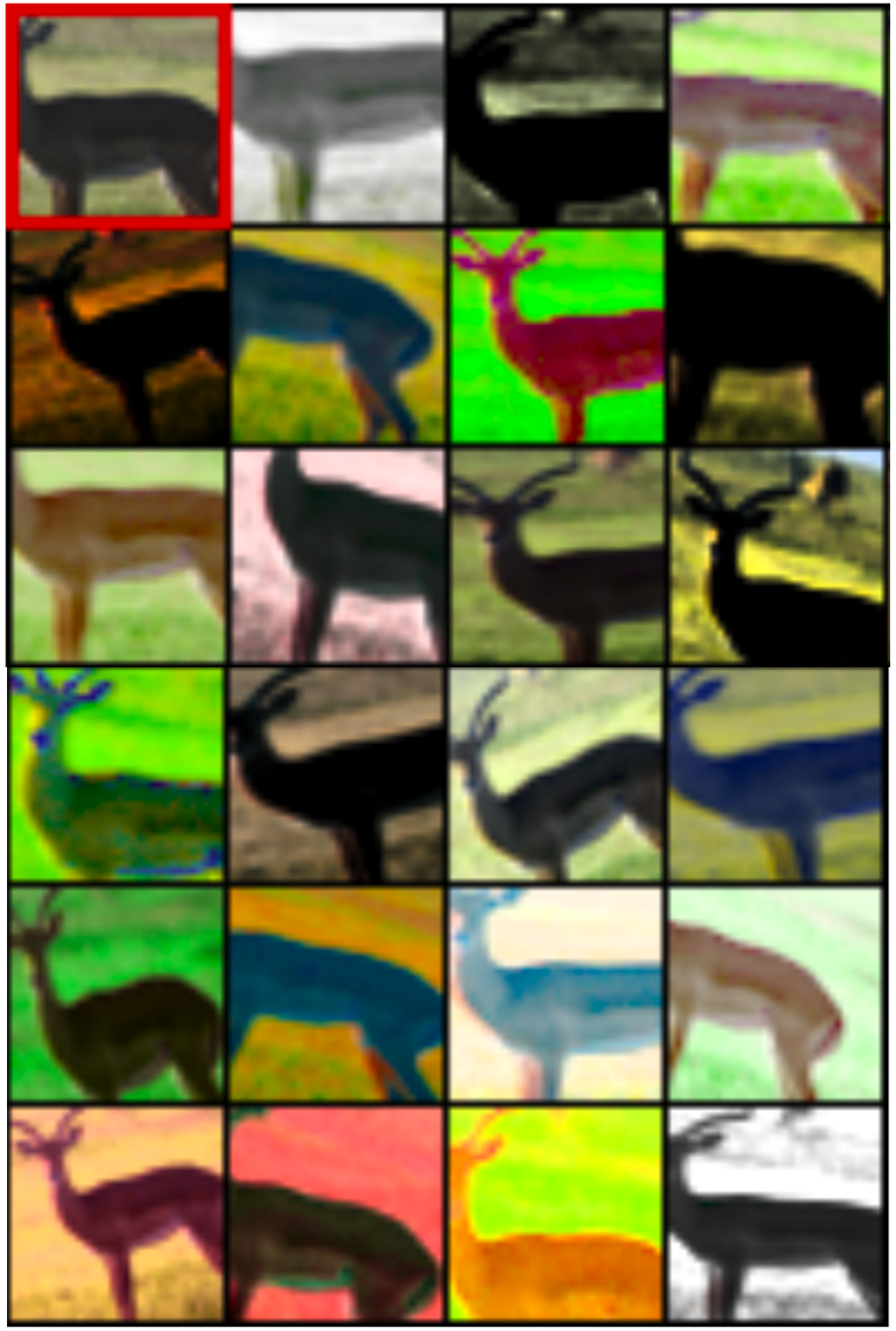
The pretext task: Each input image is first rotated by a multiple of 90 at random, corresponding to [0, 90, 180, 270]. The model is trained to predict which rotation has been applied, thus a 4-class classification problem.

The model has to learn to recognize **high level traits**, such as heads, noses, and eyes, and the relative positions of these parts, rather than local patterns.


Objectives:
Image X
$g(X, y=0)$
Rotate 0 degrees
Rotated image: X^0
ConvNet model F(.)
Maximize prob. $F^0(X^0)$
Predict 0 degrees rotation (y=0)
$g(X, y=1)$
Rotate 90 degrees
Rotated image: X^1
ConvNet model F(.)
Maximize prob. $F^1(X^1)$
Predict 90 degrees rotation (y=1)
$g(X, y=2)$
Rotate 180 degrees
Rotated image: X^2
ConvNet model F(.)
Maximize prob. $F^2(X^2)$
Predict 180 degrees rotation (y=2)
$g(X, y=3)$
Rotate 270 degrees
Rotated image: X^3
ConvNet model F(.)
Maximize prob. $F^3(X^3)$
Predict 270 degrees rotation (y=3)

Exemplar-CNN

augmentations



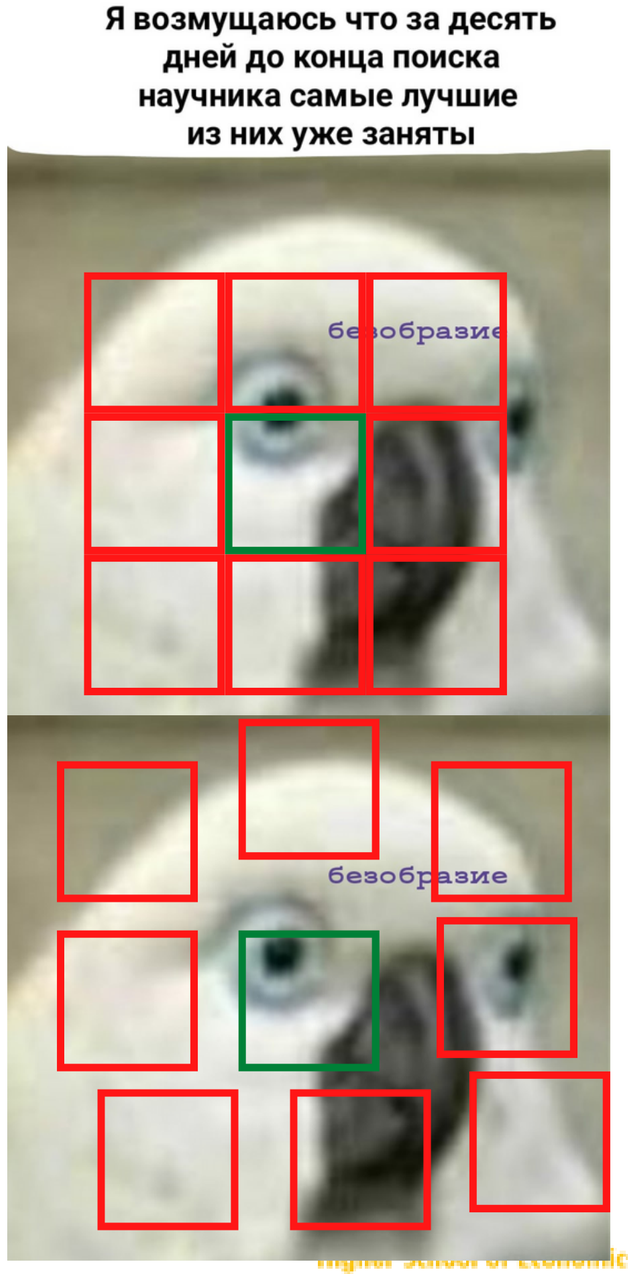
The pretext task is to discriminate between a set of surrogate classes.

Relative position

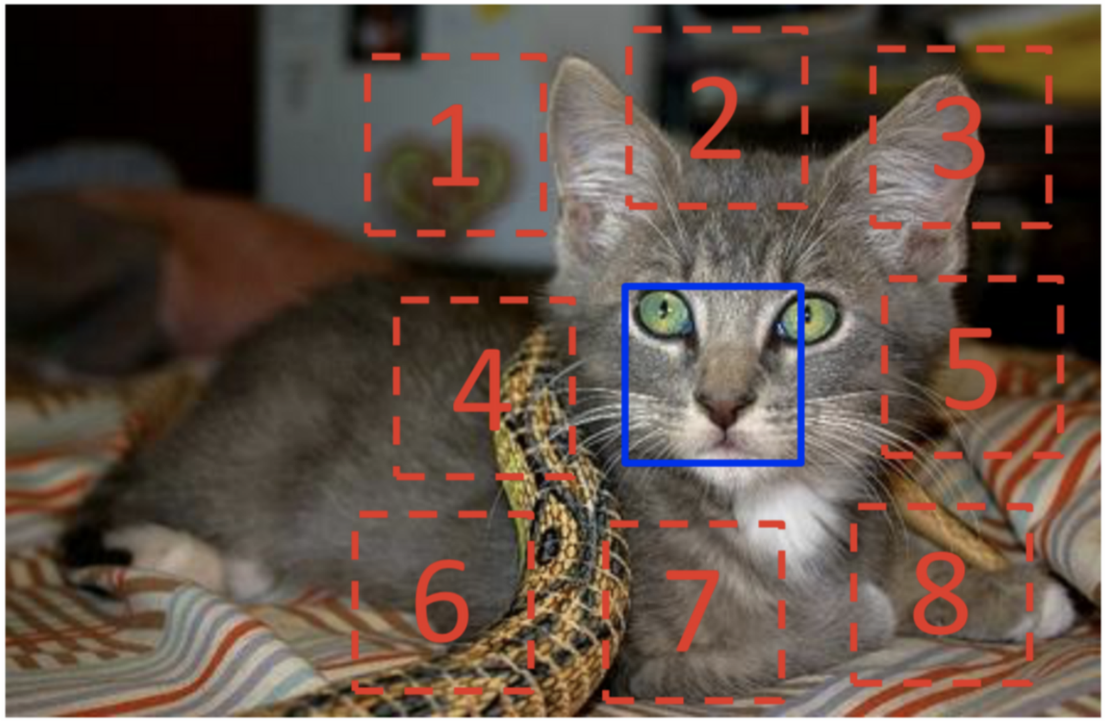
Preparation:

1. Randomly sample the first patch.
2. Considering that the first patch is placed in the middle of a 3x3 grid, and the second patch is sampled from its 8 neighboring locations around it.
3. Avoid the model only catching low-level trivial signals.

The pretext task is to predict which one of 8 neighboring locations the second patch is selected from, a classification problem over 8 classes.



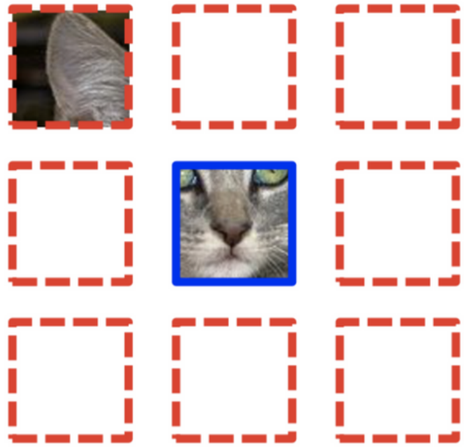
Higher School of Economics
Faculty of Computer science



X = ( , ); Y = 3

Example:





Question 1:

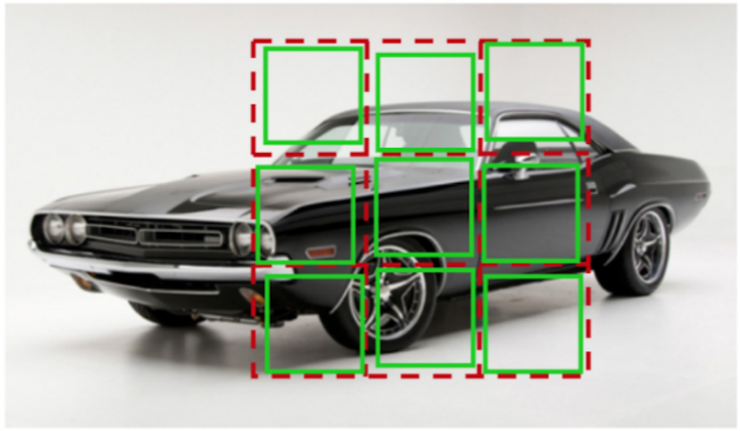
?

Question 2:

?

Jigsaw puzzle

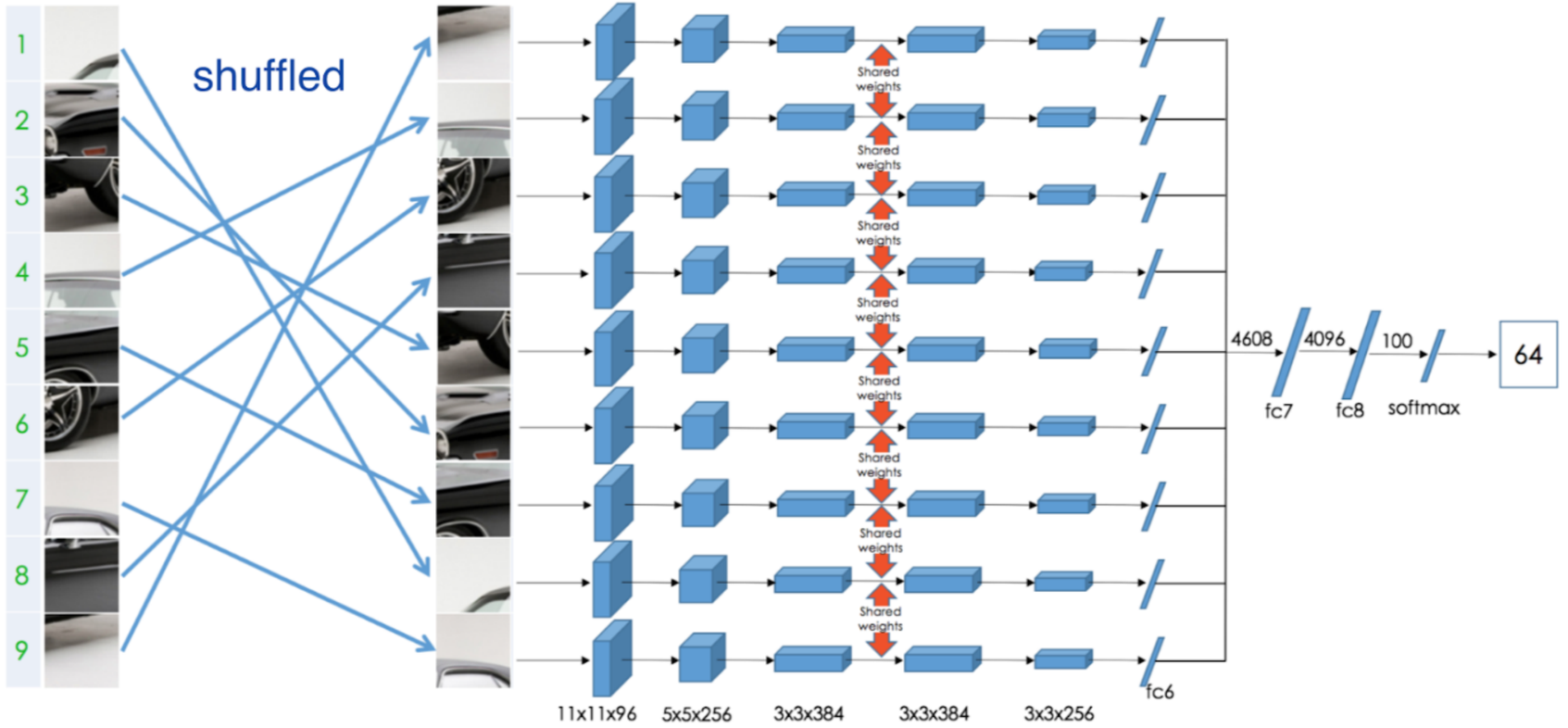
The pretext task: The model is trained to place 9 shuffled patches back to the original locations.



Permutation Set

index	permutation
64	9,4,6,8,3,2,5,1,7

Reorder patches according to the selected permutation



Higher School of Economics
Faculty of Computer science

Contrastive Representation Learning

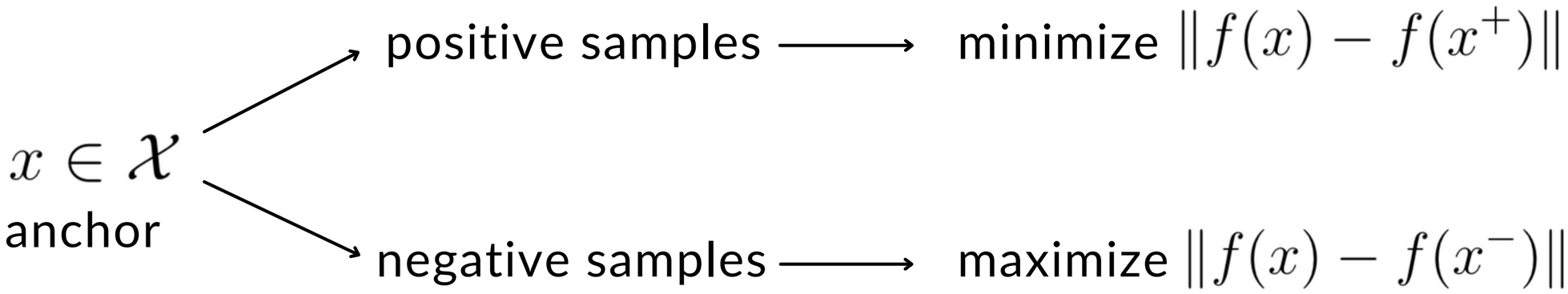
Contrastive Representation Learning

Goal: to learn an embedding space - $f : \mathcal{X} \to \mathbb{R}^d$

Contrastive Representation Learning

Goal: to learn an embedding space - $f : \mathcal{X} \to \mathbb{R}^d$

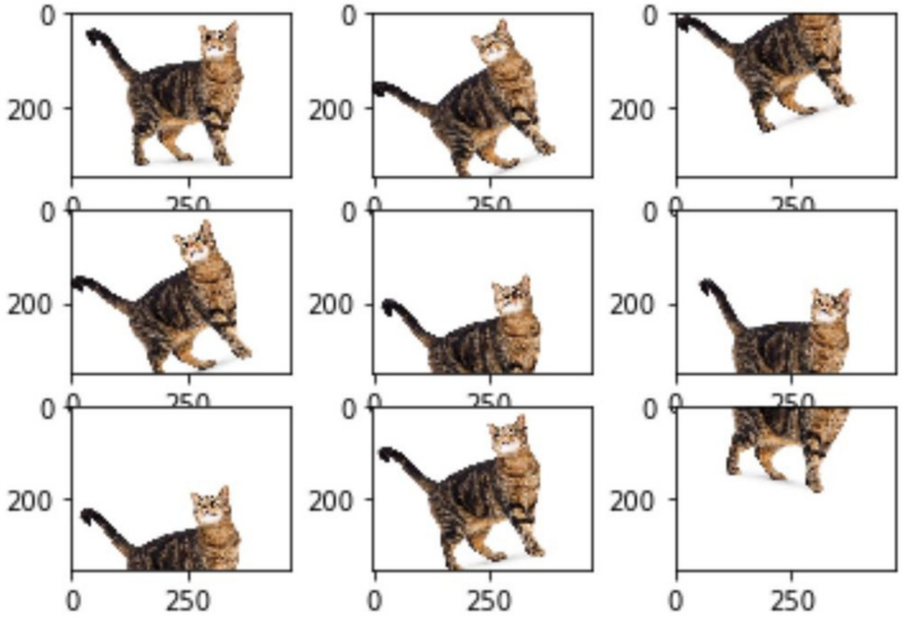
Idea: $\mathcal{X}$ - unlabeled data



Positive samples

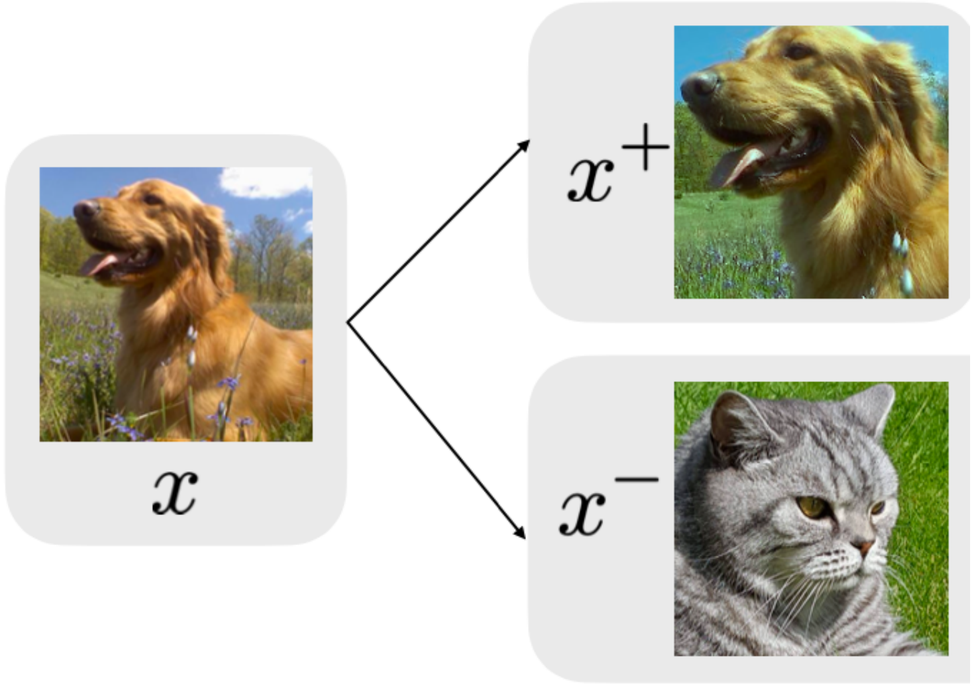
Augmentations:

- **simple** & **complicated**



Negative samples

Hard negatives: defferent labels, but similar embeddings



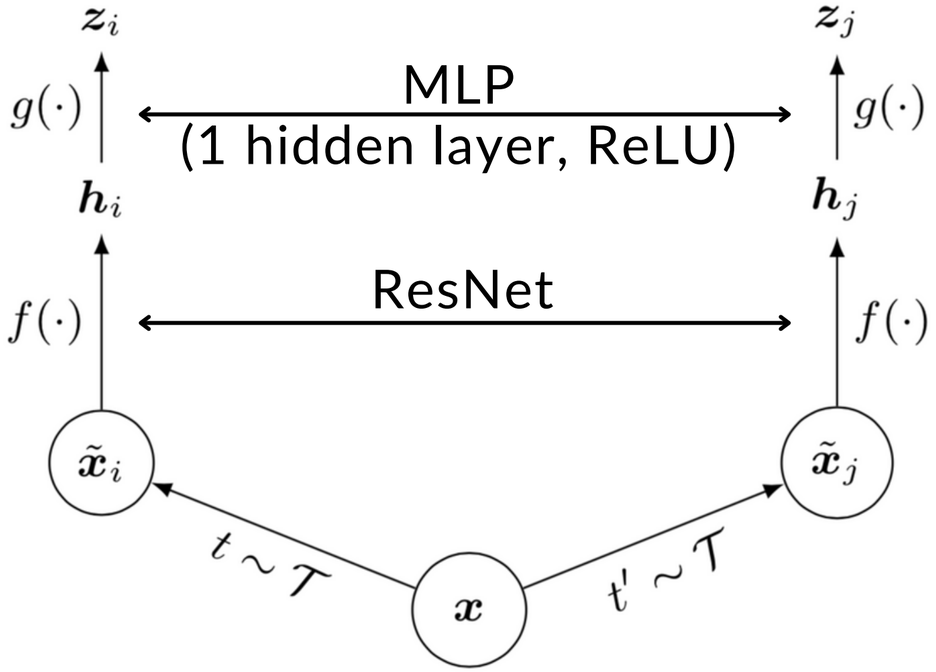
Hard negatives mining is hard when data is unlabeled.

But we can try to implicitly introduce them, for example via **large batch size.**

SimCLR

Idea:

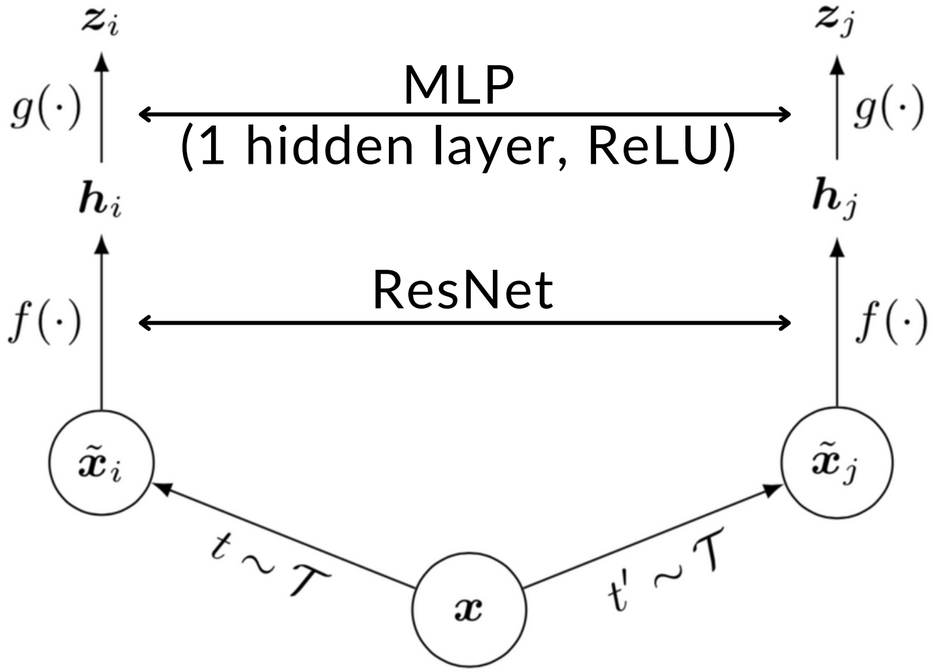
1. Batch of size N -> batch of size 2N
2. Representations of augmented images
3. Projections of representations
4. Cosine similarity: $\text{sim}(u, v) = \frac{u^T v}{\|u\| \cdot \|v\|}$



SimCLR

Idea:

1. Batch of size N -> batch of size 2N
2. Representations of augmented images
3. Projections of representations
4. Cosine similarity: $\text{sim}(u, v) = \frac{u^T v}{\|u\| \cdot \|v\|}$



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

SimCLR

Computing embedding for a large amount of negative samples in every batch is **expensive**.

⇓

Memory bank - store the representation in memory to trade off data staleness for cheaper compute

MoCo (Momentum Contrast)

$\{k_0, \ldots, k_K\}$ - dictionary of encoded keys (for previous batches)

f_q - honest representation, θ_q - parameters

f_k - key encoder, θ_k - parameters

slowly progressing! $\Rightarrow$ maintain key's consistency

Higher School of Economics
Faculty of Computer science

MoCo (Momentum Contrast)

$\{k_0, \ldots, k_K\}$ - dictionary of encoded keys (for previous batches)

f_q - honest representation, θ_q - parameters

f_k - key encoder, θ_k - parameters

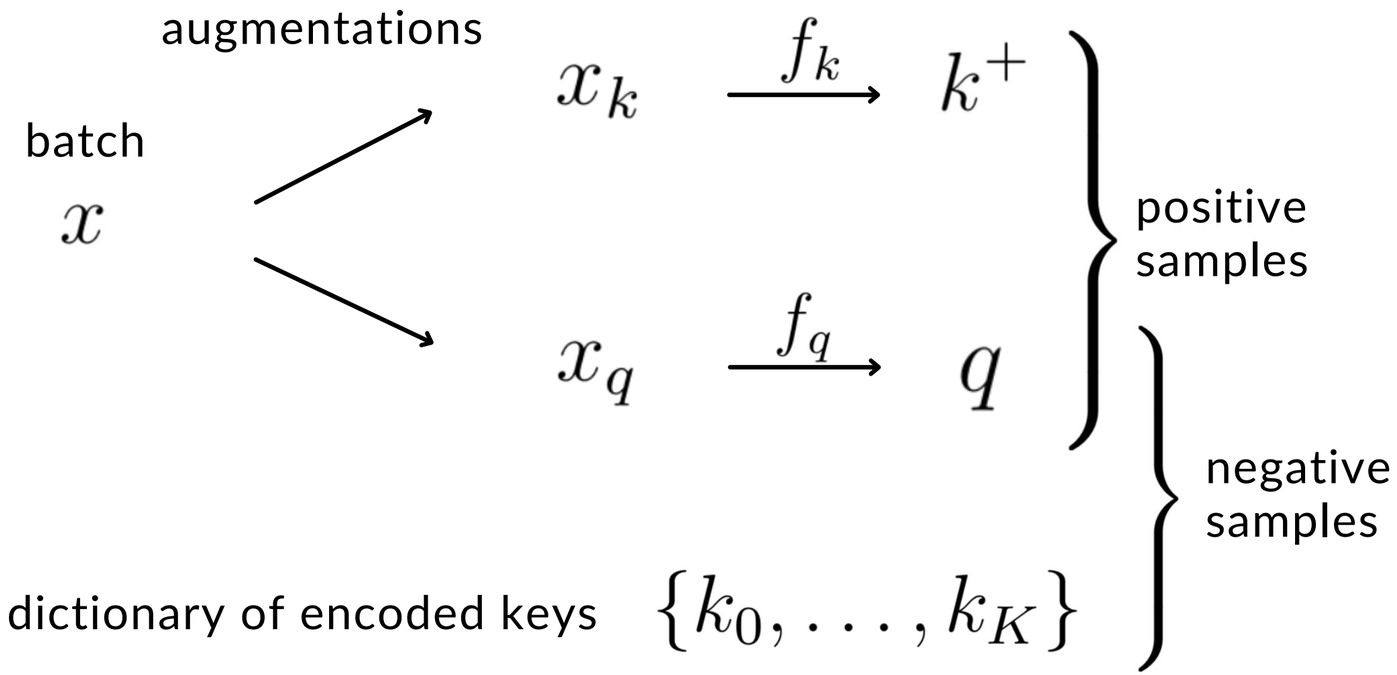
slowly progressing! $\Rightarrow$ maintain key's consistency

Updating encoders:

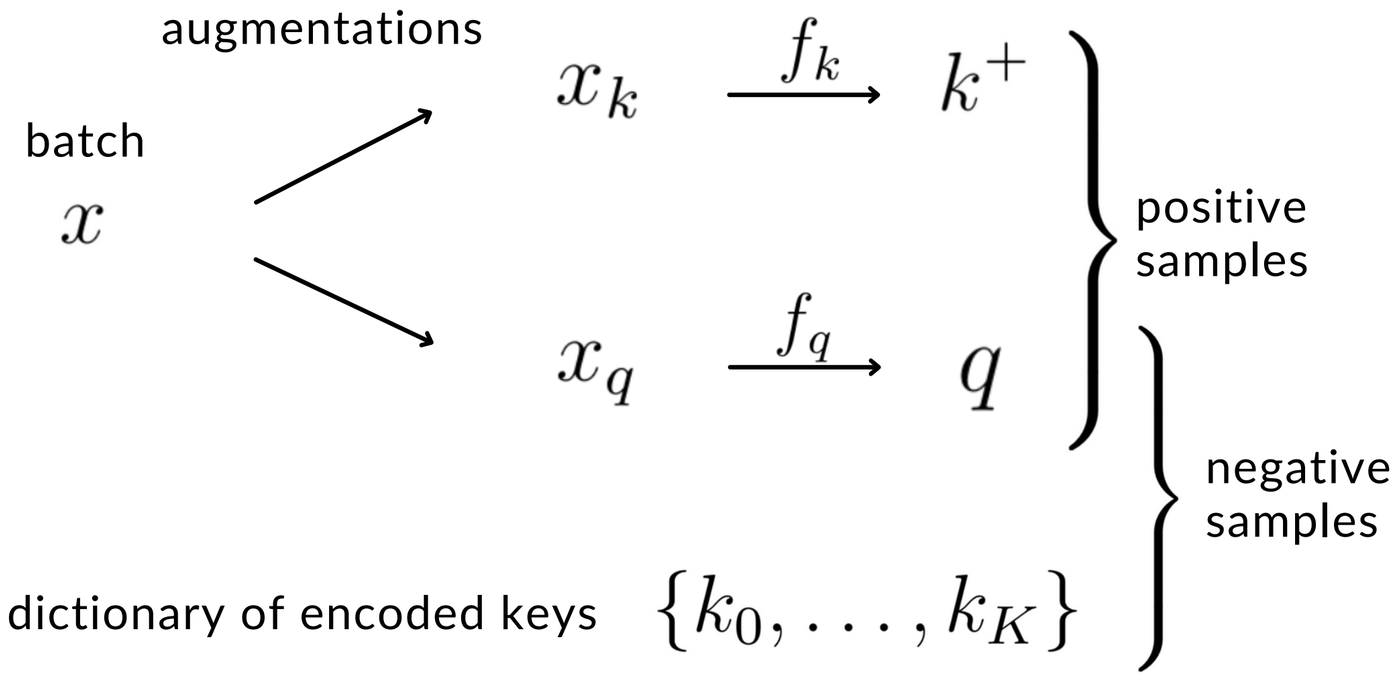
f_q - gradient

f_k - momentum: $\theta_k \leftarrow \underbrace{m\theta_k}_{m \approx 0.999} + (1-m)\theta_q$

MoCo


augmentations
batch
x
x_k
f_k
k^+
x_q
f_q
q
positive samples
negative samples
dictionary of encoded keys
$\{k_0, \ldots, k_K\}$

MoCo



Loss:

$$\mathcal{L}_q = -\log \frac{\exp(q \cdot k_+ / \tau)}{\sum_{i=0}^{K} \exp(q \cdot k_i / \tau)}$$

Updating dictionary:

dequeue the earliset batch k_0 $\longleftarrow$ $\{k_0, \ldots, k_K\}$ $\longleftarrow$ k^+ enqueue current batch

Higher School of Economics
Faculty of Computer science

MoCo v2

MoCo doesn't require a large batch size in order to have enough negative samples!
Can it become even better?

Key ideas of SimCLR:

1. Heavy data augmentation (+ Gaussian blur)
2. Projection layer

Apply them to MoCo $\Rightarrow$ **MoCo v2**

SimCLR vs MoCo

case	pre-train epochs	pre-train batch	ImageNet acc.
MoCo v1	200	256	60.6
SimCLR	200	256	61.9
SimCLR	200	8192	66.6
MoCo v2	200	256	**67.5**

- Large batches are not necessary
- Even better performance

What makes CRL special?

Self-supervised learning:
Learns representation by learning features, invariant to augmentations

Contrastive learning:
Learns representation by grouping similar objects together and detecting common features

Overview

- Self-Supervised Learning doesn't need human-labeled dataset.
- Don’t care about the final performance of pretext task.
- Care about the learned representation with high level traits.
- **Pretext tasks**: Rotation, Exemplar-CNN, Relative position, Jigsaw puzzle, Contrastive Learning and others!

- CLR aims at learning an embedding space
- Key factors: data augmentation and negative samples
- Large batch size (SimCLR), memory bank (MoCo)
- MoCo + key ideas of SimCLR = MoCo v2 -> better performance!

Higher School of Economics
Faculty of Computer science

Appendix

SimCLR

Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of f, g, $\mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # 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 $g(\cdot)$

Higher School of Economics
Faculty of Computer science

MoCo

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.

Higher School of Economics
Faculty of Computer science