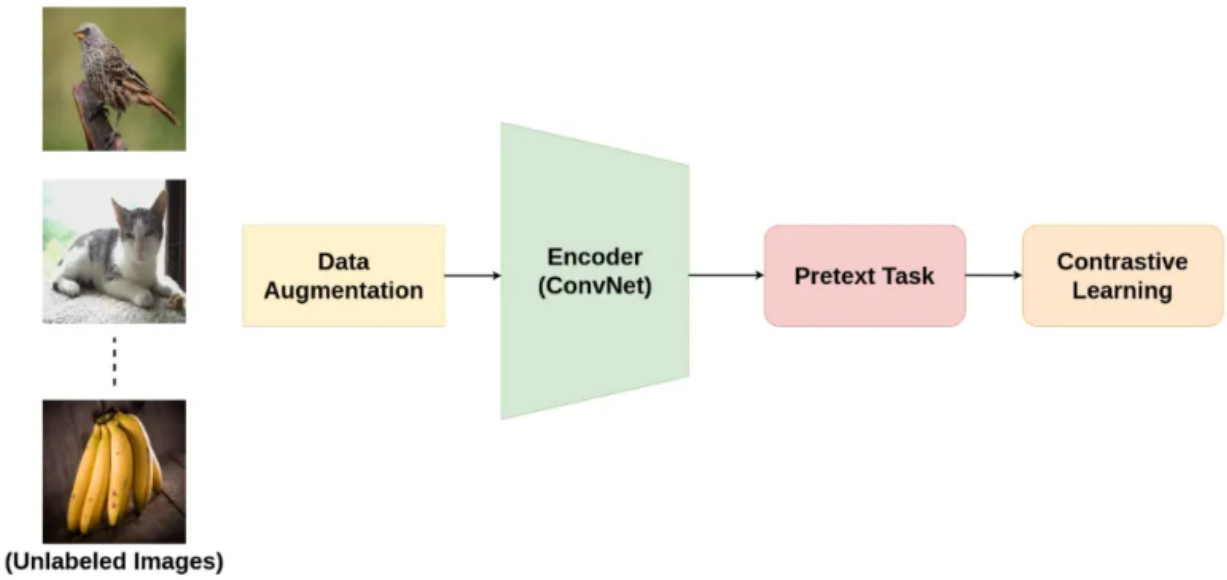
MOCOv1

背景介绍



对比学习的pipeline 来源：鲁班模锤

来源：b站 李沐 论文精读：MOCO

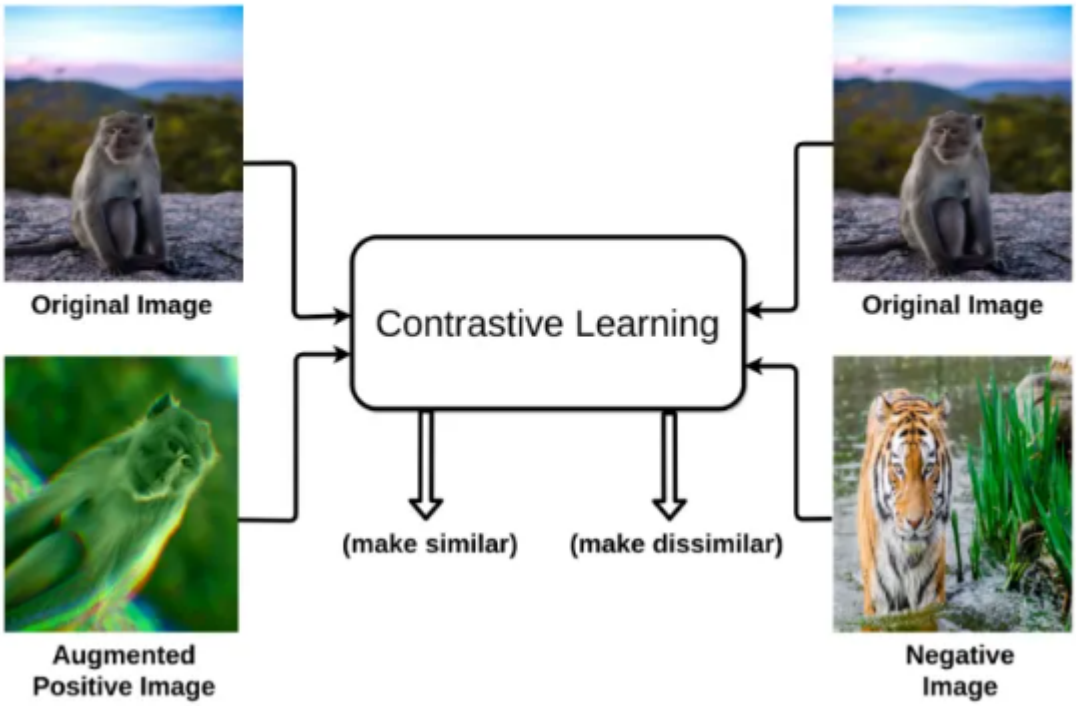
预训练

预训练的目的是训练一个强大的特征提取器，只需要简单的微调就可以轻松的迁移到下游任务中。主流的工作都是在ImageNet上做有监督的训练，而Moco采用无监督的方式进行训练，不需要标注数据集。

对比学习

无监督和有监督的区别就在于没有groud truth，无监督通过代理任务，利用代码去为每一个图片生成一个groud truth。

对比学习的核心思想是学习数据的潜在表示，并使用这些表示来衡量不同数据点之间的相似性。通过训练模型来预测数据点之间的相似性，可以学习到数据的内在特征和结构。



对比学习是一种无监督的训练方式，首先要设计**代理任务**，规定哪些是正样本和负样本。例如：

• Instance discrimination：将每张图作为一个类，基于它生成的图也视为该类（即正样本），其他图片都是负样本。（MOCO用的就是这个代理任务）

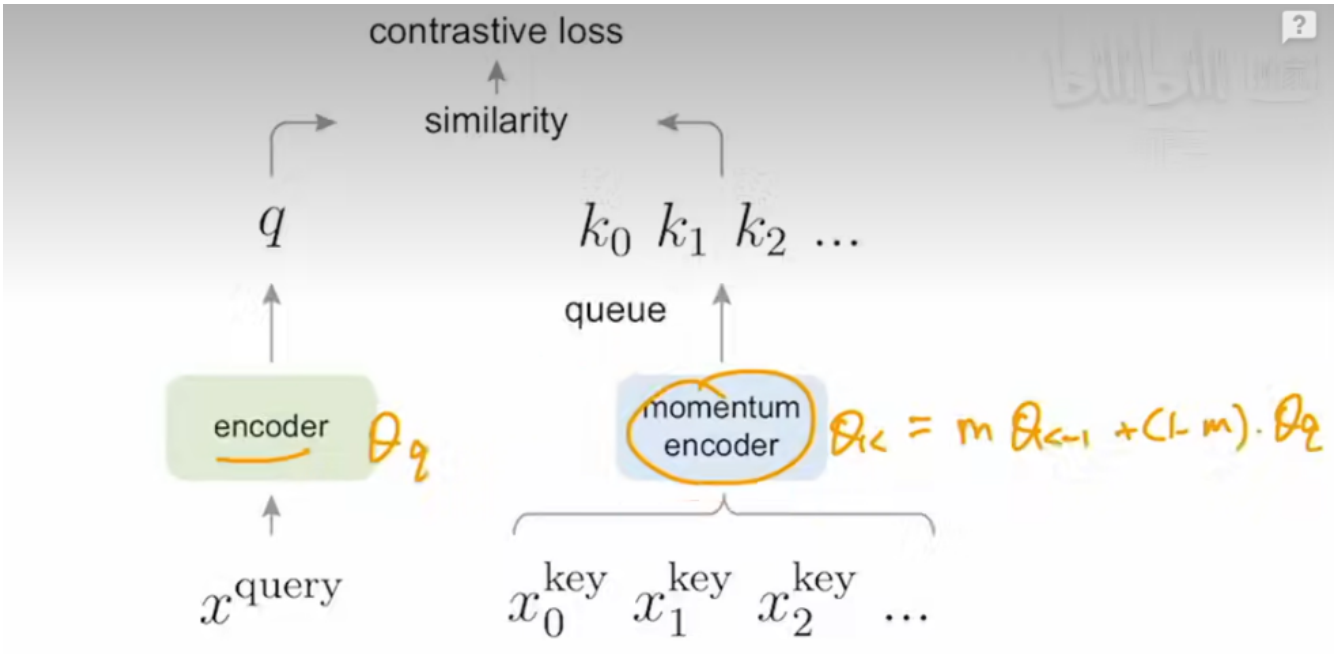
• 视频：正样本：同一个视频里的任意两帧，负样本：其他视频里的所有帧

• SimCSE：正样本：同样的句子给模型做两次forward，每次forward使用不同的dropout， 负样本：其他所有句子的特征。

然后基于一个编码器模型将图片转为特征，然后利用对比学习目标函数（如NCE loss），训练得到正负样本特征差异明显的的编码器。这样的编码器能最大程度抽取出图片的特征。

训练流程

1. 从momery bank中取出一个mini-batch的图片（下图中只用一张图片，实际有N张）。
2. 对图片进行2次数据增强，对应x^{query}和x_0^{key}。利用Instance discrimination代理任务，规定x^{query}只有一个正样本x_0^{key},字典中的图片都是负样本。
3. 对于x^{query}，使用encoderθ_q生成特征q，对于x_0^{key}和字典中的图片，利用momentum encoderθ_k产生一系列的k。
4. 由于dictionary采用队列结构（先进先出），将mini-batch中所有的q装入dictionary尾部，将头部的一个mini-batch的数据删除。
5. 利用InfoNCE loss更新θ_q和θ_k，使得θ_q和θ_k生成的q和k_0接近，而和其他的远离，这样训练出的encoder特征提取能力很强。



问题描述

区分Moco中有三个概念：momery bank，dictionary和mini-batch。

• momery bank：包含所有的图片，每轮中都要从中采样一个mini-batch的数据

• dictionary：动态字典，包含大量图片，每轮选取的mini-batch都要跟字典中的图片进行对比学习。动态体现在每一轮中的key和

$$\theta_k$$

都是变化的。

• mini-batch：只包含少量图片。

Moco是将对比学习当成动态字典，字典规模应该足够大，且能保持一致性。

• 足够大：数据量大能扩大模型的学习范围，从而提升学习效果，但是面临显存不够的问题。

• 一致性：由于θ_k θ_q每一轮都是变化的，也就是每一轮装入dictionary的key来自不同的特征空间，来自不同特征空间的q和k做对比学习效果很差。

MoCo主要设计了三个核心操作：Dictionary as a queue、Momentum update和Shuffling BN

Momentum update

Moco采用动量更新，保证

$$\theta_k$$

不会更新太快

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

其中m为动量系数，初始化时$\theta_k = \theta_q$。训练过程中只有θ_q进行梯度更新，而θ_k不做反向传播。论文中m设置为0.999，说明保持θ_k稳定时模型训练效果较好。

通过使用动量更新，保证

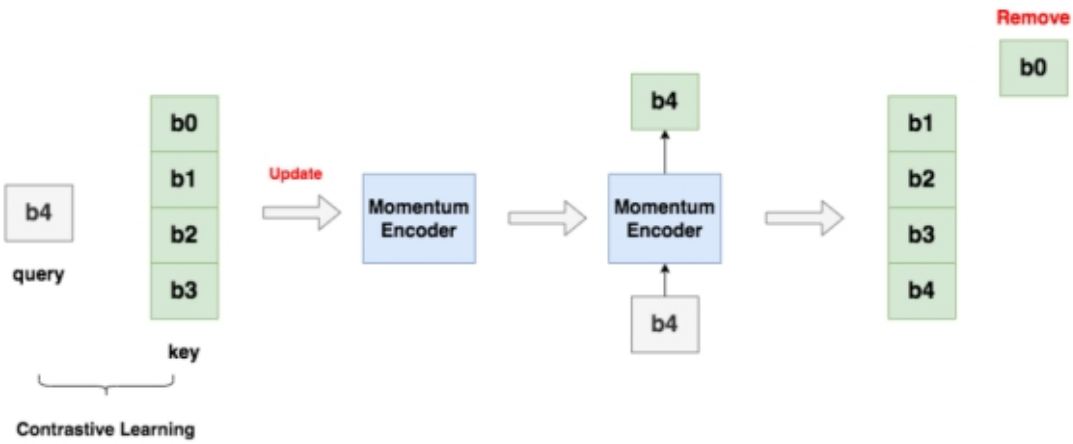
$$\theta_k$$

更新速度缓慢，从而使得字典中的key来自相似的特征空间，从而最大程度保障了一致性。

Dictionary as a queue

Moco将mini-batch大小和dictionary大小分开，每轮只抽取一个mini-batch图片进行训练，从而保障了既不会显存爆炸，同时又能提升学习效果。dictionary用queue来实现，由于FIFO，老的minibatch的数据被替换。queue中的都是较新的编码器生成的。

使用queue的好处是保持字典更新，由过早的momentum encoder生成的特征被删除，增强了一致性。



Shuffling BN

当我们直接用MOCO学出来的模型当一个特征提取器，做微调时最佳学习率居然是30，这说明MOCO学到的特征和有监督学到的特征完全不一样。因此就想到用BN。

但是实验发现BN层会阻碍模型学习一个好的特征。由于每个batch内的样本之间计算mean和std导致信息泄漏，产生退化解。MoCo通过多GPU训练，分开计算BN，并且shuffle不同GPU上产生的BN信息来解决这个问题。

InfoNCE loss（noise constractive loss）

与softmax很相似，区别在于Moco把每个图片都当成一类，导致K的值很大。于是NCE转成二分类问题，只区分data和noise。

InfoNCE是NCE一种变体，将二分类改成K+1分类。K表示dictionary的大小，总共1个正样本和K个负样本。

$$\tau$$

表示温度，温度过高会导致曲线平缓，所有负样本差别很小，模型学习失去侧重点。温度过低会导致曲线陡峭，模型只能学到与正样本接近的负样本的特征，模型难以收敛和泛化。

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

使用InfoNCE loss保障了由于q和唯一正样本

$$k_+$$

相似，对应的分子项大；q跟负样本不相似，对应的分母项小，经过-log操作后的loss值比较小。

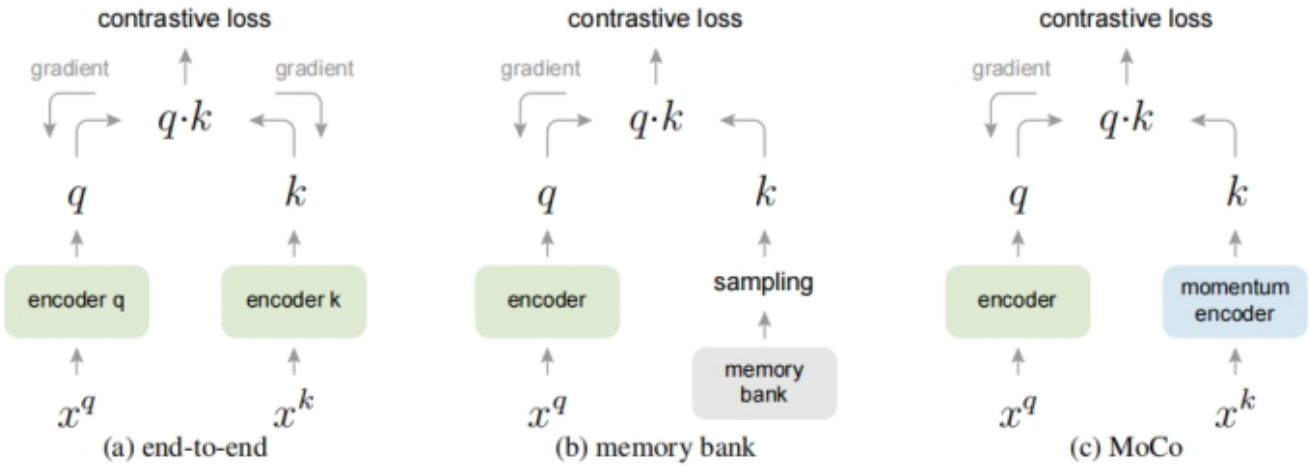
其他类似工作

a) End-to-end

正负样本都是从一个mini-batch中采样来的，字典和mini-batch一样大，encoder k和encoder q同步更新，优点是一致性高，但字典不会很大（受限于gpu大小）

b）momery bank

每次采样一个mini-batch，字典足够大，但是由于encoder k一直不更新，导致一致性差。



伪代码

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                    C:一个图片是1个C维向量

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    由于cat将正样本放在第0
   loss = CrossEntropyLoss(logits/t, labels)     位，所以label值为0

   # 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    encoder k动量更新，
                                                 encoder q进行反向传播
   # update dictionary
   enqueue(queue, k) # enqueue the current minibatch   队列更新
   dequeue(queue) # dequeue the earliest minibatch
```