Momentum Contrast for Unsupervised Visual Representation Learning

恺明太强悍了，广州人的骄傲！有机会的话鼠鼠去试试套恺明（滥）

Abstract

MoCo是一种运用在视觉表征的无监督学习方法。具体来说，MoCo使用了对比学习，创建了动态的图像字典库和动量解码器。并且MoCo最突出的工作是他可以迁移到下游任务，战胜基于有监督方法的模型，挖了一个大坑。

Conclusion

在同样经过了预训练后，MoCo基本杀穿了有监督模型，一雪前耻（指end-to-end和memory bank），并且证明了MoCo是一种训练成本相对低廉，普通科研工作者也可以使用的方法。（可是我连一张3090都没有呀！！！隔壁的同学快把你的8卡A100给我！！！）

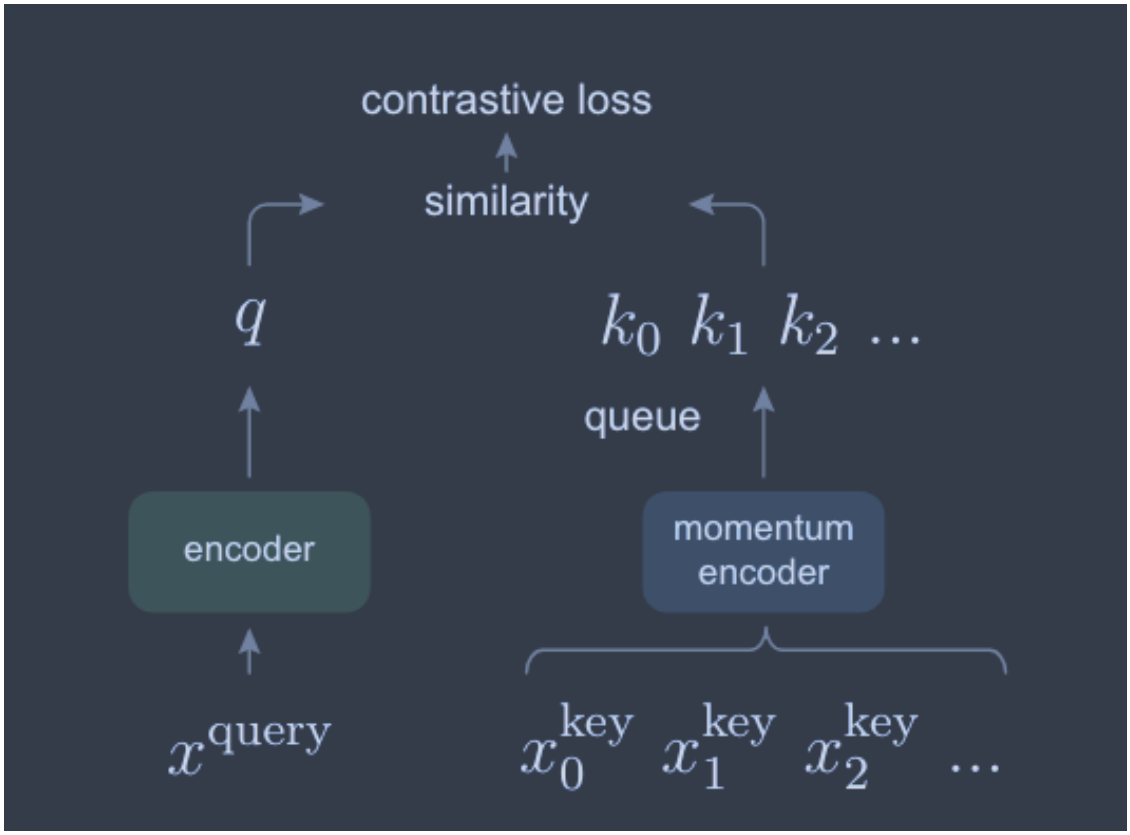
Introduction

基于无监督学习方法的模型一直是视觉领域工作者的final dream。参考NLP中的BERT，是创建了一个token字典库，以存储token特征；然而视觉任务中，图像是高维且连续的特征空间，不仅创建字典库十分困难，并且难以输入供无监督学习学习的信号。

MoCo则使用了一种很巧妙的方法。这里得先简单地讲下对比学习：举个例子，假如有两个穿背带裤的ikun和一只鹦鹉的图像，即使没有给他们打标签，你也可以通过对比ikun和鹦鹉知道他们长得一点都不像，所以会把两个ikun分在一起。

因此我们可以运用以下思想：如果我们可以将图像提取出特征向量，则相似的图像之间特征向量距离应该尽量拉近，不相似的特征向量距离应该尽量拉远。

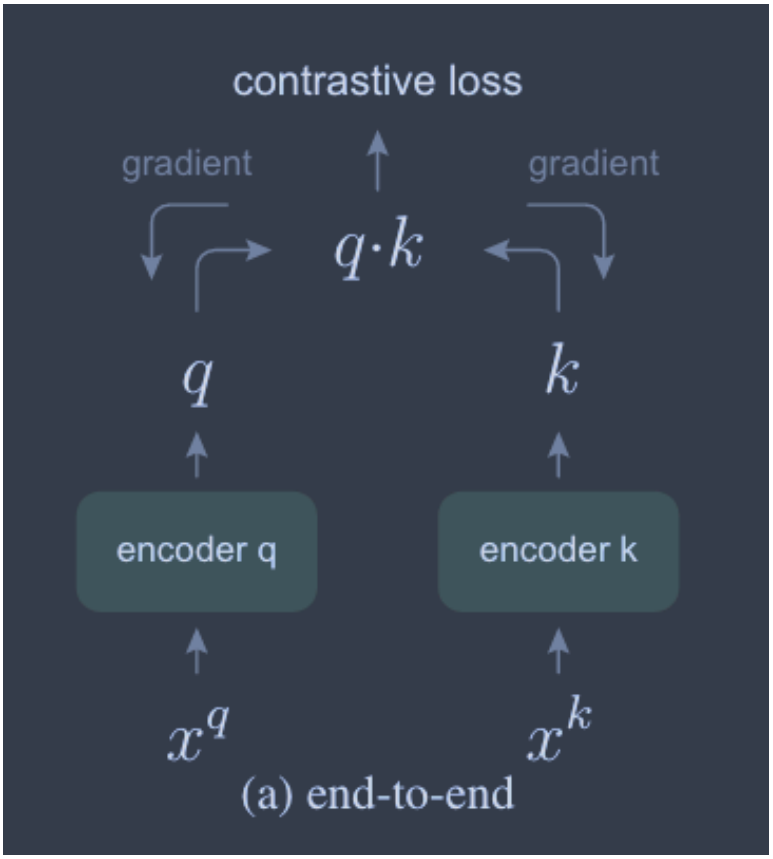
接下来问题又来了，我们怎么样输入信号告诉模型你这个分类好，这个分类不好呢？这个时候又要用到“代理任务”了，在MoCo里，我们假定每张图片都单独是一类，剩下的都是其他类。那如果每一类只有一张图片要怎么训练呢？答案是：数据增强。我们可以通过裁剪、拉伸、平移、旋转等方法制造同一幅图像Xquery的不同版本X0key作为正样本（因为这些操作大多不会改变图像特征），而batch其他的图像如X1key、X2key则被作为负样本。我们所需要做的就是让Xquery forward得到特征q，让X0key、X1key、X1key经过forward后得到特征k0、k1、k2等，最后再优化参数使得q和k0距离尽可能近，使q和剩下的k1、k2等尽可能远。



Methods

讲MoCo之前，先讲讲先前的两个工作。

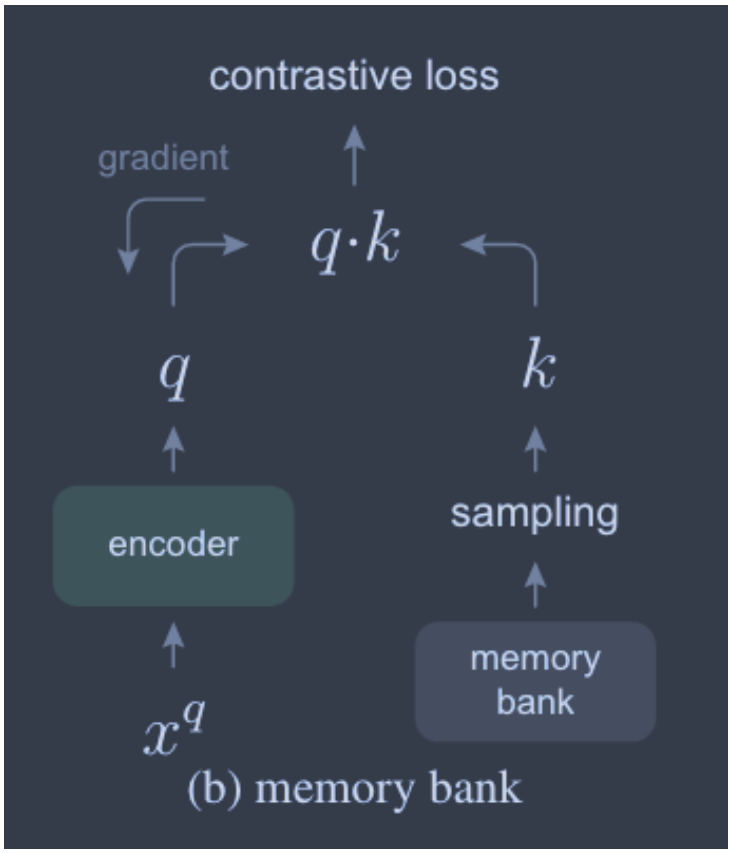
- （a）end-to-end



在提取k的特征时，我们应该使用相似的编码器。如果编码器不相似的话，那么模型学习到的不是图像本身之间相似的特征，而是很有可能学习到相似的编码器。举个例子，A和B都是ikun，C是鹦鹉，但是B和C的编码器更像，导致模型依靠编码器特征将B和C分到一起。所以所有的k在提取特征时都应该使用相似的编码器。

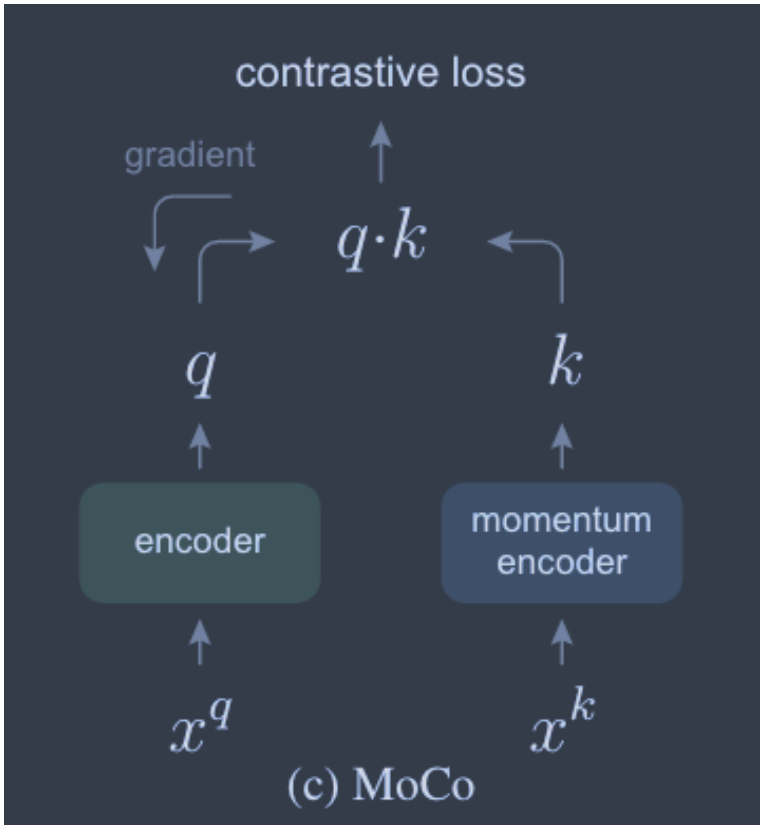
而end-to-end模型没有上述问题，因为它直接全部输入都使用同一编码器。但代价就是试图构造字典时，需要一股脑输入所有batch，显卡显存根本存不下，更别提后面的计算开销了。那如果用小一点规模大字典呢？一个字典规模越大，他包含的特征就越丰富多样；反之，你的字典规模不够大的话，包含的特征就不足以区分图像。

- （b）memory bank



memory bank则和end-to-end背道而驰，它放弃了使用同一编码器，将batch分批次输入，但是不同时间的编码器也会更新而不一样，导致特征提取时会有不一致性。

- （c）MoCo



MoCo则想到了个两全其美的方法，名曰动量编码。

Formally, denoting the parameters of f_{k} as θ_{k} and those of f_{q} as θ_{q}, we update θ_{k} by:

$$\theta_{\mathrm{k}} \leftarrow m\theta_{\mathrm{k}} + (1-m)\theta_{\mathrm{q}}. \tag{2}$$

在这里，k的编码器确实会变，但是由于动量m的存在（m是介于0和1之间的数，通常为0.999以上），导致k编码器更新非常慢。同时，MoCo还采用了队列结构，（fifo）这样就可以分批次输入的同时保证编码器相似，还能更新编码器。

具体流程如下：

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.

值得一提的是，有人问为什么q和k不用同一个编码器？这里笔者认为其实要不要用同一个都可以，能保证k的编码器相似就OK，介于之前memory bank使用了不同的编码器，MoCo也使用了不同的编码器。

还有一个比较有意思的点是损失函数

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

这玩意其实就是softmax换了张脸。

照例鼠鼠偷懒不写实验啦，等找到富婆被包养了再回来写。