对比学习第一阶段

InstDisc

- 提出个体判别任务。
- 每一个instance都看作一个类别，每一个图片都作为一个类
- 通过卷积神经网络，将图片编码为一个特征。每一个图片在特征空间中都与其他图片相隔足够远
- 正样本就是图像本身和它的图像增强样本，负样本就是其他图片。

该文章使用一个memroy bank存储这些负样本，imagenet中有128w的数据，意味着memory bank有128w行，因为负样本太多了，如果使用较高的维度表示图片的话，对于负样本的存储代价过高，因此作者让向量维度为128维。

前向传播的过程：
假设模型的batchsize是256，有256张图片进入CNN网络，将256张图片编码为128维的向量。因为batchsize是256，因此有256个正样本。负样本来自memory bank，每次从memory bank中随机采样出4096个负数样本，利用infoNCE loss去更新CNN的参数。本次更新结束后，会将CNN编码得到的向量替换掉memory bank中原有的存储。就这样循环往复的更新CNN和memory bank，最后让模型收敛

在InstDisc还有其他的细节
比如Proximal Regularizatio，给模型的训练增加了一个约束，从而可以让memory bank里面的特征进行动量式更新，保持一致性。

InvaSpread

还是个体判别Instance Discrimination作为代理任务
InvaSpread并没有用额外的数据结构存储大量的负样本，他就是用mini batch中的数据作为负样本，而且使用一个编码器进行端到端的学习。

该文章设置的batchsize是256。首先利用数据增广，将每个图片增广一次，也就是将256张图片变为512个图片了。之后将512张图片通过CNN编码为向量，并使用一个全连接层将数据的维度降低。
之后将xi和其经过增广后的图片xii作为正样本，其余的512-2张图片都认为是负样本。所以总计有256个正例，有2×（256-1）张负例。之后的在特征空间中xi与xii的距离应该尽可能相近，与其他样本的距离应该尽可能相远。

CPC

一般机器学习分为生成式模型与判别式模型
个体判别Instance Discrimination属于判别式的范畴，CPC属于生成式的代理任务。

其主要的想法是，有一个持续的序列，把之前时刻的输入喂给编码器，返回的特征再喂给一个自回归模型，然后得到一个context representation，这是一个代表上下文的特征表示

对比学习需要正负样本的对比，所以在CPC中，这里的正样本就是通过编码器之后得到未来时刻的特征输出z t+i，将其他的序列作为负例，进行对比学习的训练。

CMC

CMC想学一个非常强大的特征，其具有视角的不变性（不管是看见了一只狗，还是听到了狗叫声，都能判断出这是个狗）。

CMC的工作目的就是去增大这个互信息，就是所有视角之间的互信息。如果能学到一种特征，能够抓住所有视角下的这个关键的因素，那么这个特征就比较好。

CMC使用的数据集是NYU RGBD数据集，该数据集包含一张图片的四种view数据增强结果（包含原始的图像，以及这个图像对应的深度信息，SwAV ace normal，语义分割）。虽然这些不同的输入view来自于不同的传感器，或者说是不同的模态，但是这些所有的输入其实对应的都是一整的图片，一个东西，那么它们就应该互为正样本，相互配对。而这些相互配对的视角在特征空间中应该尽可能的相近，而与其他的视角尽可能的远离。

CMC定义正负样本的方式：
将不同的view作为正例，将其他图片以及其他图片的views作为负例子，进行训练。

对比学习第二阶段

MoCov1

- 类似于InstDisc

主要贡献就是把之前对比学习的一些方法归纳为一个字典查询问题。提出了一个队列，一个动量编码器，从而形成一个又大又一致的字典，帮助更好的进行对比学习。

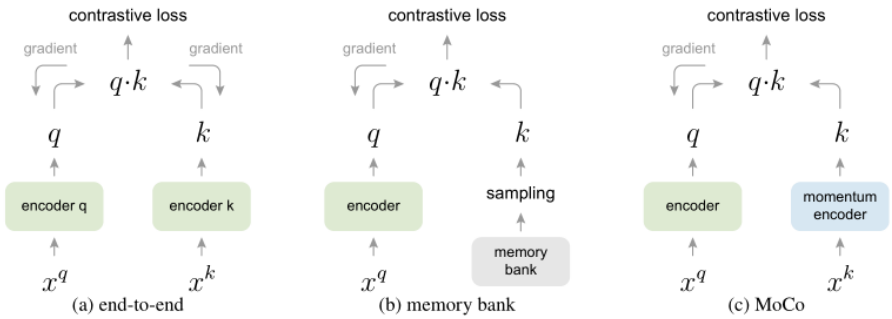


Figure 2. **Conceptual comparison of three contrastive loss mechanisms** (empirical comparisons are in Figure 3 and Table 3). Here we illustrate one pair of query and key. The three mechanisms differ in how the keys are maintained and how the key encoder is updated. **(a)**: The encoders for computing the query and key representations are updated ***end-to-end*** by back-propagation (the two encoders can be different). **(b)**: The key representations are sampled from a ***memory bank*** [61]. **(c)**: ***MoCo*** encodes the new keys on-the-fly by a momentum-updated encoder, and maintains a queue (not illustrated in this figure) of keys.

https://blog.csdn.net/qq_42718887

SimCLRv1

假如有一个minibatch的图片，对整个minibatch的所有图片做数据增强，对图片x xx做不同的数据增强就会得到xi和xj。同一个图片延申得到的两个图片就是正样本，比如batchsize是n的话，那么正样本就是n，这个batchsize剩下的所有的样本以及其经过数据增强后得到的都是负样本，也就是2（n-1）。

有了正负样本之后，对其进行编码，通过一个编码器f(·)，f(·)得到正负样本的编码结果h。SimCLR的创新点就是在得到数据的编码之后在后面加了一个编码层g(·)函数，就是一个MLP层，得到较低维度的特征zi和zj ，用其进行对比学习，拉近正例之间的距离，拉远负例之间的距离。但是需要注意的一点就是投影函数仅仅在训练的时候才使用，在测试的时候是不使用的，测试的时候仅仅使用编码器f(·)。加上投影函数的目的也仅仅是想让模型训练的更好。

通过一系列消融实验证明随机的裁剪以及随机的色彩变换的必要性，其他的数据增强最后都只是锦上添花可有可无。

MoCov2

MOCO v2相当于是把SimCLR中值得借鉴的地方拿来借鉴，比如其中MLP的投影层，更多的数据增强方式，cosine learning rate schedule，以及更多的迭代epoch。

SimCLRv2

- SimCLRv2相比SimCLRv1就是用了更大的模型，加深了projection head，最后用了半监督编码器。
- 看不懂意义

SWaV

即使以往的工作是非常有效并且简洁的，但是因为负样本太多了，从而造成资源的浪费，即使是MOCO这样用近似的方式用6w个负样本，但是总共还是有128w个负样本的，6w个负样本只是一种近视。所以SwAV的作者去想，能不能不做近似呢？可不可以使用先验信息，不去和大量的负样本对比，而是和一些更加简洁的东西去比呢？所以SwAV的作者想，可以和聚类的中心进行对比。这个聚类中心就是C CC，维度是3000×向量为度，3000表示聚类中心的数量。

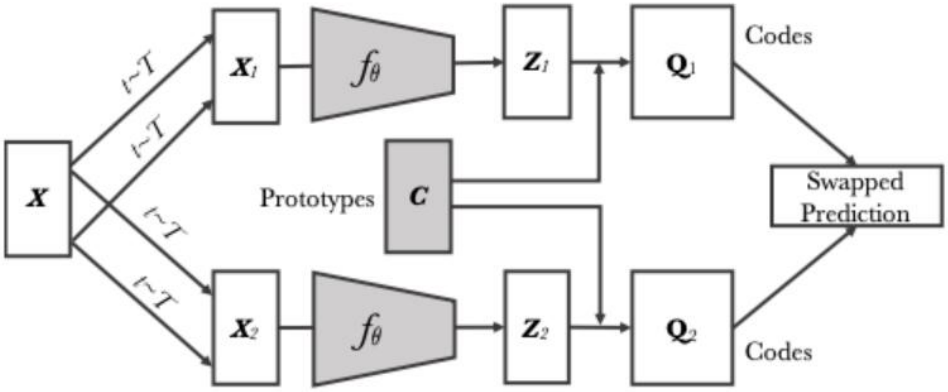


图1 SwAV架构图

SwAV前向过程依旧是一个实例x xx通过两次数据增强变为x1和x2 ，之后利用编码器对其进行编码，从而得到嵌入向量z1和z2 。但是有了z1和z2之后，并不是直接在特征上去做对比学习的loss，而且让z1和z2和聚类中心C 进行聚类，从而得到ground truth的标签Q1和Q2。如果说两个特征比较相似或者是含有等量的信息，按道理来说应该是可以相互预测的。也就是说，用z1和C作点乘按道理是可以去预测Q2的，反过来用z2和C作点乘按道理是可以去预测Q1的，SwAV通过这种换位交叉预测的方法来对模型进行训练更新参数。

如果是和负例进行对比的话，需要和成千上万个负例进行对比，即使是MOCO中6w个负例，也只是一个近似的值，但是聚类的话，就仅仅需要和三千个聚类核心即可。

这些聚类中心是有含义的，而如果像之前一样用负样本进行对比学习的话，有的负样本不均衡，有的还可能是正样本，不如聚类中心有效。

```
# Multi-crop:
```
SwAV的重要性能提升点
以往的对比学习方法都是在一张256×256的图片上用两个224×224的crop求两个正样本，但是因为crop过大了，所选取的crop都是基于全局特征的，所以可能忽视了局部特征。所以SwAV使用了一种multi-crop的思路进行操作，即选择了两个160×160的crop去注意全局特征，选择四个96×96的crop去注意局部特征。这样在计算量变化不大的情况下，可以获取更多的正样本，也可以同时注意了全局特征与局部的特征。

对比学习第三阶段（不用负样本）

BYOL

在对比学习中的负样本是一个约束，如果在算目标函数的时候只有一个正样本，这时候就只有一个目的，就是让相似的物体他们的特征也尽可能的相似。这时候就可能会有一个明显的捷径解，如果一个模型无论给什么样的输入，他都返回相同的输出，这样出来的所有特征，都是一模一样的，这样去计算对比学习的loss就全为0，模型这样躺平是学习不到实例的特征的，是无效的。

因此需要添加了负样本对模型造成一个约束，这样如果模型躺平直接输出输入的话，对于正样本的loss为0，但是对于负样本来说loss就无穷大，这样子模型才会学习如何权衡让正负样本的loss都往下降，达到一个最优解。所以负样本在模型中是一个必须的东西，可以防止模型躺平，学到这个捷径解。

BYOL的神奇之处在于模型没有使用负样本，仅仅是模型自己和自己去学，但是也实现了很不错的效果。

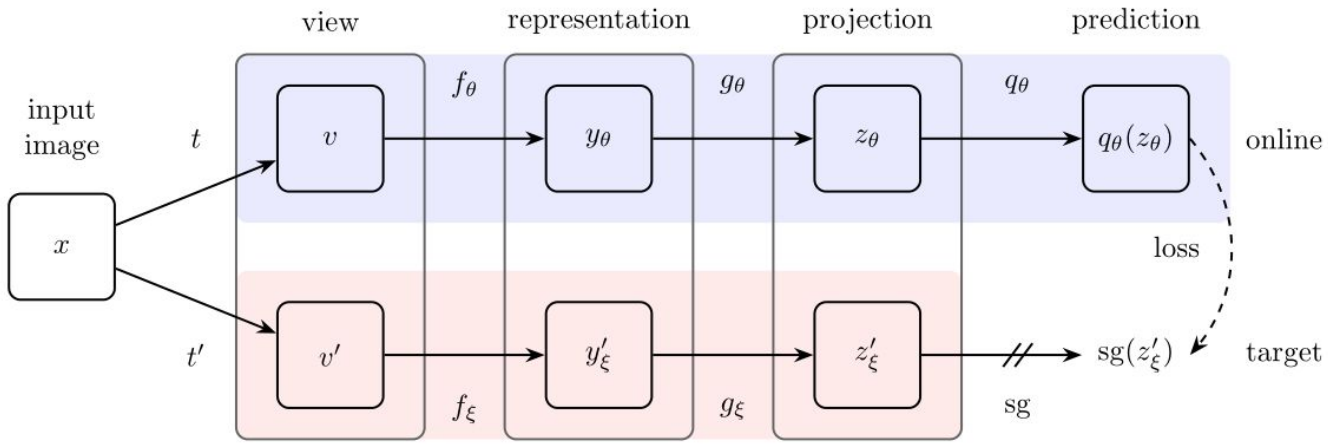


Figure 2: BYOL's architecture. BYOL minimizes a similarity loss between $q_\theta(z_\theta)$ and $\mathrm{sg}(z'_\xi)$, where θ are the trained weights, ξ are an exponential moving average of θ and sg means stop-gradient. At the end of training, everything but f_θ is discarded, and y_θ is used as the image representation.

首先有一个mini-batch的输入x，经过两次不同的数据增强之后得到v和v'，然后通过编码器fθ和fξ分别得到特征yθ和yξ'（如果是ResNet50，就是得到了一个2048维的特征）。这里的编码器fθ和fξ使用的相同的网络架构，但是其参数是不同的，也就是分别独立，没有共享参数，只是架构相同而已。fθ是随着梯度更新而更新的，而fξ与MoCo一样，是用moving average的这种形式去更新的，其实也就是使用了动量编码器。

接下来与SimCLR一样，用了projection head编码器gθ和gξ（这里称为projector，其实就是一个MLP层）再得到zθ和zξ'特征（这里是256维）。gθ和gξ同样也是一样的网络结构，但是参数不一样。gξ也是通过动量的这种方式去更新的。

在之前的对比学习工作中，是让zθ和zξ'尽可能的相似，而在BYOL这里，又加了一层predictor的全连接层qθ，qθ的网络结构和gθ的网络结构是完全一样的。zθ通过qθ又得到了一个新的特征qθ(zθ)，现在的目的是想让特征qθ(zθ)与zξ'尽可能的相似。这相当于把原来匹配的问题，换成了现在一个预测的问题。

图中的sg表示stop gradient，这里是没有梯度的。这与MoCo很像，模型的上一支相当于query编码器，下面一支相当于key编码器，而key编码器都是通过query编码器来动量更新。不同是代理任务不一样，BYOL相当于是自己一个视角的特征去预测另外一个视角的特征，通过这种预测性的任务来完成模型的训练。

当训练结束后，只有编码器fθ留下了，这相当于是一个特征提取器，其余的全部拿走。然后通过编码器fθ输出的特征yθ去做其他的下游任务。目标函数用的是MSE

在一篇博客上，Understanding Self-Supervised and Contrastive Learning with “Bootstrap Your Own Latent” (BYOL)发现了BatchNorm的一个问题。当BYOL的projection head不使用BatchNorm时，会发生模型坍塌的情况，博客作者作了一下实验：实验发现，在BYOL里，当在任意一个projection head使用BatchNorm时，模型就不会坍塌。博客对这个情况作了一个解释，BatchNorm会吧一个batch里所有样本的特征计算均值方差，也就是running mean和running variance。然后用整个batch算来的均值和方差去作归一化。这就说明，当用某个正样本的loss时，其实也看见了其他样本的特征，也就是发生了信息泄露。由于这种信息泄露的存在，可以把这个batch中的其他样本看成是一种隐式的负样本。所以BYOL就可以看出是当前的样本和平均图片有什么差别，平均图片就是BatchNorm产生的，这里的平均图片和SwAV的聚类中心的说法有点相似。

这篇博客认为BatchNorm是BYOL成功的关键，相当于提供了一种隐式的正负样本学习。

BYOL作者认为：
实验表明，对于SimCLR，即使提供了显式的负样本，没有BN还是失败。BatchNorm可以提高模型的训练稳健性，从而导致不会模型坍塌，也就是如果一开始模型的初始化就比较好（比如使用group norm与weight standardization），那么BYOL其实没有BatchNorm也是可以正常训练的。

SimSiam

- SimSiam 不需要用负样本，不需要大的batchsize，不需要动量编码器，即使在这种条件下，SimSiam不仅没有模型谈谈，反而取得了很好的模型效果。

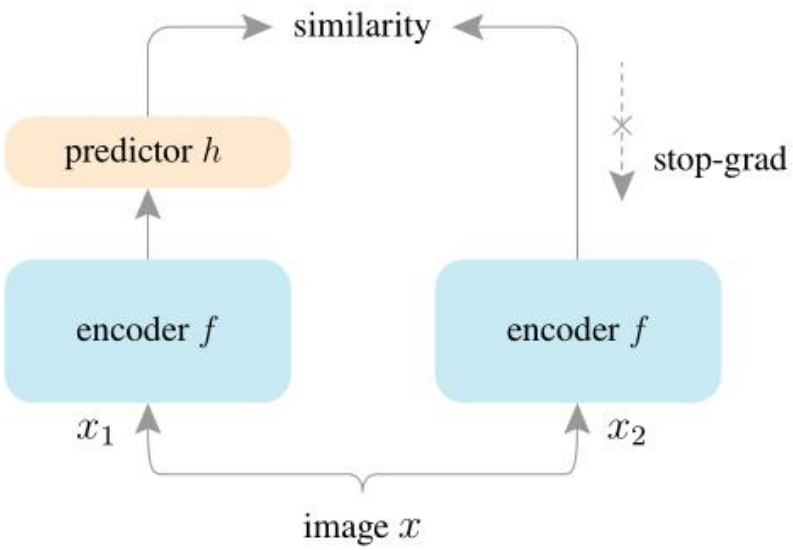


Figure 1. **SimSiam architecture**. Two augmented views of one image are processed by the same encoder network f (a backbone plus a projection MLP). Then a prediction MLP h is applied on one side, and a stop-gradient operation is applied on the other side. The model maximizes the similarity between both sides. It uses neither negative pairs nor a momentum encoder.

```
# 伪代码
# f: backbone + projection mlp
# h: prediction mlp
for x in loader: # load a minibatch x with n samples
    x1, x2 = aug(x), aug(x) # random augmentation
    z1, z2 = f(x1), f(x2) # projections, n-by-d
    p1, p2 = h(z1), h(z2) # predictions, n-by-d

    L = D(p1, z2)/2 + D(p2, z1)/2 # loss

    L.backward() # back-propagate
    update(f, h) # SGD update

def D(p, z): # negative cosine similarity
    z = z.detach() # stop gradient

    p = normalize(p, dim=1) # l2-normalize
    z = normalize(z, dim=1) # l2-normalize
    return -(p*z).sum(dim=1).mean()
```

SimSiam能够成功训练的原因，不会发生模型坍塌，主要就是因为有stop gradient这个操作的存在。由于stop gradient，可以将SimSiam的结构看成是一个EM算法，相当于是在解决两个子问题，而模型

更新也在交替进行，相当于不断的更新聚类中心。

SimSiam也是预测任务，但是使用的是stop gradiant的方式进行预测的，并且使用孪生网络，不需要动量编码器

对比学习第四阶段（Transformer）

- 在vision transformer之后，因为其大大提升了encoder的效果，所以很多对比学习任务打算使用vision transformer作为backbone进行对比学习，涌现出了两篇工作，分别是MoCov3和DINO。

MoCov3

- MoCov3中大部分的篇幅都在将如何做Vision Transformers的自监督训练

```
# 伪代码
# f_q: encoder: backbone + proj mlp + pred mlp
# f_k: momentum encoder: backbone + proj mlp
# m: momentum coefficient
# tau: temperature

for x in loader: # load a minibatch x with N samples
    x1, x2 = aug(x), aug(x) # augmentation
    q1, q2 = f_q(x1), f_q(x2) # queries: [N, C] each
    k1, k2 = f_k(x1), f_k(x2) # keys: [N, C] each

    loss = ctr(q1, k2) + ctr(q2, k1) # symmetrized
    loss.backward()

    update(f_q) # optimizer update: f_q
    f_k = m*f_k + (1-m)*f_q # momentum update: f_k

# contrastive loss
def ctr(q, k):
    logits = mm(q, k.t()) # [N, N] pairs
    labels = range(N) # positives are in diagonal
    loss = CrossEntropyLoss(logits/tau, labels)
    return 2 * tau * loss

# Notes: mm is matrix multiplication. k.t() is k's transpose. The prediction head
is excluded from f k (and thus the momentum update).
```

可以发现，MoCov3有点像MoCov2和SimSiam的结合。

整体的网络还是有两个编码器，一个query编码器，一个是key编码器，而且key编码器是一个动量编码器，而且最后的目标函数用的是对比学习的loss。

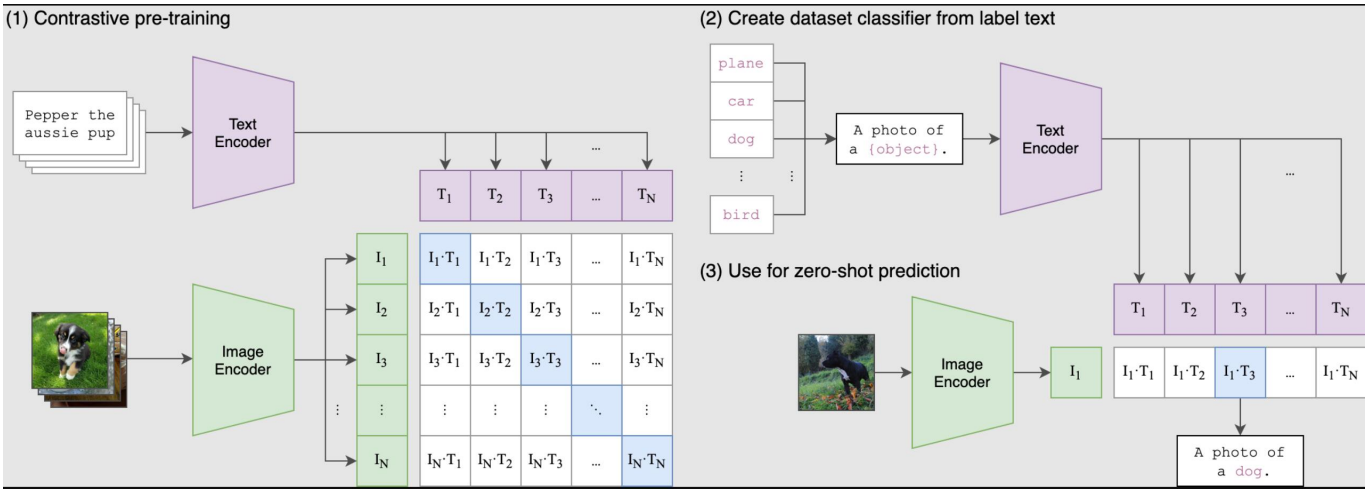
且编码器结构用还使用了projection head和prediction head，而且目标函数也是一个对称性，就

是相互预测，计算query1到key2，也计算query2到key1。

当把backbone将残差网络换成是Vit时，在batch size比较小时，训练还算比较稳定，曲线比较平滑；但是一旦batch size变大，模型就出现了不稳定的情况，虽然最后也会回复上去，但是准确率会差很多。
作者观察了一下模型梯度回传时候的梯度情况。当每次loss有大幅的震动，导致准确度大幅下降的时候，梯度也会有一个波峰，波峰发生在第一层，在作patch projection的时候，因为这一层经常出现问题，所以作者尝试将这一层冻结住（也就是不再训练这层的参数）。解决的办法就是随机初始化了这个patch projection层，然后对其冻结，随后的训练过程中参数保持不变，然后问题就解决了，获得了平滑了训练曲线。

从方法和模型的角度上来说，其实和第三阶段基本是一模一样的，主要就是融合了Vision Transformers。

CLIP



CLIP是一种基于对比学习的多模态模型，与CV中的一些对比学习方法如moco和simclr不同的是，CLIP的训练数据是文本-图像对：一张图像和它对应的文本描述，这里希望通过对比学习，模型能够学习到文本-图像对的匹配关系。如下图所示，CLIP包括两个模型：Text Encoder和Image Encoder，其中Text Encoder用来提取文本的特征，可以采用NLP中常用的text transformer模型；而Image Encoder用来提取图像的特征，可以采用常用CNN模型或者vision transformer。

这里对提取的文本特征和图像特征进行对比学习。
对于一个包含N个文本-图像对的训练batch，将个N文本特征和N个图像特征两两组合，CLIP模型会预测出个可能的文本-图像对的相似度，这里的相似度直接计算文本特征和图像特征的余弦相似性（cosine similarity），即上图所示的矩阵。这里共有N个正样本，即真正属于一对的文本和图像（矩阵中的对角线元素），而剩余的N^2-N个文本-图像对为负样本，那么CLIP的训练目标就是最大N个正样本的相似度，同时最小化负样本的相似度，对应的伪代码实现如下所示：

```
# image_encoder - ResNet or Vision Transformer
# text_encoder - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l] - minibatch of aligned texts
```

```
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t - learned temperature parameter

# 分别提取图像特征和文本特征
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T) #[n, d_t]

# 对两个特征进行线性投射，得到相同维度的特征，并进行l2归一化
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# 计算缩放的余弦相似度：[n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# 对称的对比学习损失：等价于N个类别的cross_entropy_loss
labels = np.arange(n) # 对角线元素的labels
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss = (loss_i + loss_t)/2
```

我们通过CLIP训练出来一个模型之后，满足以下条件的新任务都可以直接zero shot进行识别：
1、我们能够用文字描述清楚这个新分类任务中每个类别；
2、这个描述对应的概念在CLIP的训练集中出现过。这在经典一维标签的图像分类中是不可实现的。

CLIP这种方法把分类转换为了跨模态检索，模型足够强的情况下，检索会比分类扩展性强。比如人脸识别，如果我们把人脸识别建模为分类任务，当gallery里新增加人脸后，类别数就变大了，我们就需要重新训练模型、更新类别数；如果我们将人脸识别建模为检索，当gallery里新增加人脸后，我们用已有的模型提取这个人脸的特征，后续流程不用变，也不用重新训练模型。
从检索这个角度来看，CLIP的zero shot其实就是把分类问题转化为了检索问题。

总结来看，CLIP能够zero shot识别，而且效果不错的原因在于：
1、训练集够大，zero shot任务的图像分布在训练集中有类似的，zero shot任务的concept在训练集中有相近的；
2、将分类问题转换为检索问题。

与CV中常用的先预训练然后微调不同，CLIP可以直接实现zero-shot的图像分类，即不需要任何训练数据，就能在某个具体下游任务上实现分类，这也是CLIP亮点和强大之处。用CLIP实现zero-shot分类很简单，只需要简单的两步：

根据任务的分类标签构建每个类别的描述文本：A photo of {label}，然后将这些文本送入Text Encoder得到对应的文本特征，如果类别数目为N，那么将得到N个文本特征；

将要预测的图像送入Image Encoder得到图像特征，然后与N个文本特征计算缩放的余弦相似度（和训练过程一致），然后选择相似度最大的文本对应的类别作为图像分类预测结果，进一步地，可以将这些相似度看成logits，送入softmax后可以到每个类别的预测概率。

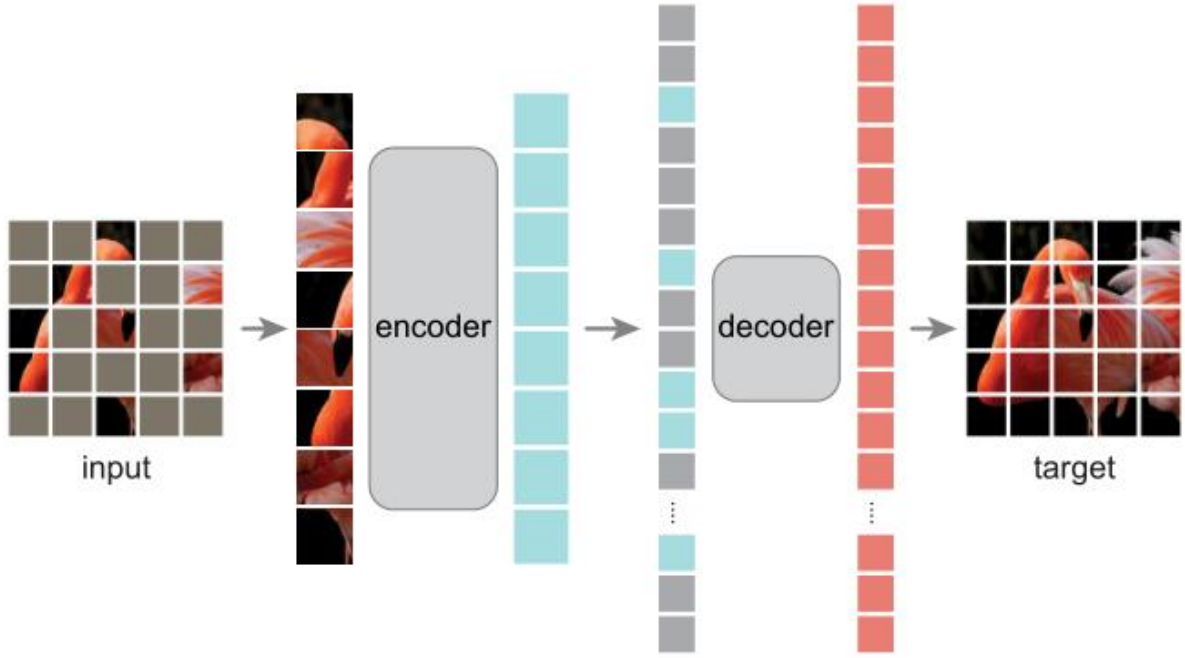
除了zero-shot对比，论文还对比few-shot性能，即只用少量的样本来微调模型，这里对比了3个模型：在ImageNet21K上训练的BiT-M ResNet-152x2，基于SimCLRv2训练的ResNet50，以及有监督训练的ResNet50。可以看到CLIP的zero-shot和最好的模型（BiT-M）在16-shot下的性能相当，而CLIP在16-shot下效果有进一步的提升。另外一个比较有意思的结果是：虽然CLIP在few-shot实验中随着样本量增加性能有提升，但是1-shot和2-shot性能比zero-shot还差，这个作者认为主要是CLIP的训练和常规的有监督训练存在一定的差异造成的。

MAE

- MAE(Masked Autoencoders)是用于CV的自监督学习方法

在MAE方法中会随机mask输入图片的部分patches，然后重构这些缺失的像素。MAE基于两个核心设计：（1）不对称的（asymmetric）编码解码结构，编码器仅仅对可见的patches进行编码，不对mask tokens进行任何处理，解码器将编码器的输出（latent representation）和mask tokens作为输入，重构image；（2）使用较高的mask比例（如75%）。MAE展现了很强的迁移性能，在ImageNet-1K上取得了best accuracy（87.8%），且因为方法简单，可扩展性极强（scalable）

在NLP领域自监督学习方法使用十分广泛，但是在CV领域，大多数预训练还是采用监督方式，因此MAE最大的贡献就是证明了自监督预训练同样可以在CV领域获得和监督预训练一样，甚至更好的效果，自监督也许可以像统治NLP一样统治CV领域



那么为什么自监督在CV领域的发展要滞后于NLP呢？论文中给了两个解释：
（1）NLP主流方法是Transformer，视觉里CNN是主流方法，结构差异让视觉很难构造类似于“masked autoencoding”的任务。但是ViT的提出解决了这个问题；

（2）语言和视觉的信息密度（information density）差异巨大，前者是强语义的，高信息密度的（highly semantic and information-dense），在NLP中即使只mask一个token，对模型来说可能都是很难的任务，因此模型可以通过学习获得复杂的语言理解能力（sophisticated language understanding），但是对视觉图像来说，信息是高度冗余的，缺失一个patch，可能并不会让模型产生多少困惑，模型可以通过周围的像素信息进行推断
所以MAE做的一件事就是mask很高比例的patches，制造高难度的学习任务，方法简单但是极其有效

首先将input image切分为patches，执行mask操作，然后只把可见的patches送入encoder中，再将encoder的输出（latent representations）以及mask tokens作为轻量级decoder的输入，decoder重构整张image

编码器：编码器实际上就是ViT，将input image切分为不重叠的patches之后，执行linear projection，再加上positional embeddings（the sine-cosine version），然后送入transformer blocks

解码器：同样使用ViT，将mask tokens和encoded visible patches作为输入，加上位置编码（the sine-cosine version）。decoder的最后一层是linear projection，输出通道数量和一个patch内的pixel数量相同（方便重构），然后再reshape，重构image。损失函数使用MSE，损失函数只对masked patches计算（和BERT相同）。同时作者也尝试了normalization的方式，即计算一个patch内像素值的均值和标准差，然后对patch执行normalization，此时encoder的重构任务发生了一些变化，需要重构normalized pixel values，实验表明这种方式效果更好一点

MAE中decoder的设计并不重要，因为预训练结束之后，只保留encoder，decoder只需要完成预训练时的图像重构任务。但是作者也表示decoder决定了latent representations的语义级别

References：

https://www.bilibili.com/video/BV19S4y1M7hm