arXiv:2301.01569v2 [cs.LG] 1 Jun 2023

使用快速傅里叶变换高效学习去相关表示

Yutaro Shigeto* Masashi Shimbo* YuyaYoshikawa AkikazuTakeuchi
{shigeto,shimbo,yoshikawa,takeuchi}@stair.center STAIRLab，千叶县工业大学，习志野市，千叶县，日本

摘要

Barlow Twins 和 *VICReg* 是自监督表示学习模型，使用正则化项对特征进行去相关。尽管这些模型与传统表示学习模型同样有效，但当投影嵌入的维度 d 较高时，训练可能需要大量计算。由于正则化项是根据互相关矩阵或协方差矩阵中的各个元素定义的，因此计算 n 个样本的损失需要$O(nd^2)$ 时间。本文提出一种宽松的去相关正则化项，可通过快速傅里叶变换在 $O(nd\log d)$ 时间内计算。我们还提出一种低成本技术，以缓解采用这种宽松方法后产生的不良局部极小值。所提出的正则化项在下游任务中的准确率与现有正则化项相当，同时在大 d情况下，训练所需内存更少、速度更快。源代码已公开。[1]

1. 引言

表征的自监督学习（SSL）[1, 4–6, 11, 14, 16, 29, 30, 33, 34] 已成为计算机视觉深度学习应用中不可或缺的一部分。大多数用于视觉表征的自监督学习模型采用多视图孪生网络架构。首先，通过不同的变换改变输入图像，同时保留其原有语义。随后，将两个增强后的样本输入神经网络（某些情况下则输入两个不同的网络），该网络由骨干网络和级联其后的一个小型投影网络（通常为多层感知机）组成，以生成原始图像的两个“孪生”投影嵌入，或称视图。最后，训练网络权重，使孪生嵌入（称为“正样本对”）彼此相似，反映出它们代表同一张原始图像。训练完成后，舍弃投影网络，并将骨干网络用于下游任务；其理念是，预训练的骨干网络应能生成通用的图像表征，从而也有利于下游任务。

自监督学习中的一个主要问题是避免坍塌嵌入，即所有样本都被投影到单个嵌入向量这一平凡解的出现。对比学习方法 [4, 5, 16, 29, 30] 通过损失项将不同原始图像的嵌入（称为“负样本对”）彼此推离，从而消除这类解。考虑所有可能的负样本对是不可行的，因此通常会进行负采样。

近期研究探索了非对比式自监督学习模型。在这些模型中，Barlow Twins [33]和 VICReg[1] 使用损失函数惩罚方差较小的特征（嵌入向量的组成部分），因为这类特征是坍塌嵌入的典型表现。这些损失函数中同样值得注意的是用于特征去相关的正则化项。它们减少了学习到的特征中的冗余，使 Barlow Twins 和 VICReg 的表现达到与对比学习模型相当的水平。然而，由于这些正则化项是根据样本互相关或协方差矩阵中的元素定义的，因此计算它们需要 $O(nd^2)$ 的时间，其中 n 是一个批次中的样本数，d 是投影嵌入的维度。这是一个问题，因为据报道，增加 d 可以提升 Barlow Twins 和 VICReg 的性能 [1, 33]。

贡献。 我们提出一种宽松的去相关正则化项，以解决 Barlow Twins 和 VICReg 效率低下的问题。所提出的正则化项无需显式计算互相关或协方差矩阵，借助快速傅里叶变换（FFT）即可在 $O(nd\log d)$ 时间内完成计算 [8]。尽管放宽约束会产生不良局部极小值，但可通过特征置换缓解；我们解释了这一简单技术为何有效。使用所提出正则化项的自监督学习模型在下游任务中取得了与 Barlow Twins 和 VICReg 相当的性能，同时在较大的 d 下显著减少了计算时间。采用 $d = 8192$ 时，训练速度是 Barlow Twins 的 1.2 倍（使用 ResNet-50 主干网络）或 2.2 倍（使用

*贡献相同。[1] https://github.com/yutaro-s/scalable-decorrelation-ssl.git

轻量级 ResNet-18 主干网络）。所提方法还降低了内存消耗，从而支持更大的批次大小。

符号说明。 除非另有说明，向量和矩阵分量均采用从零开始索引；因此，对于向量$\mathbf{x} \in \mathbb{R}^d$，分量索引的范围为 0 到 $d-1$。我们用$[\mathbf{x}]_i$ 表示向量$\mathbf{x}$的第 i 个分量，用 $[\mathbf{M}]_{ij}$ 表示矩阵$\mathbf{M}$的第(i, j)个元素。对于复向量$\mathbf{c}$，$\mathbf{c}$ 表示其各分量的复共轭。对于向量$\mathbf{x}$和$\mathbf{y}$，$\mathbf{x} \circ \mathbf{y}$ 表示它们的逐分量乘积。

2. 相关工作

对比式自监督学习。 对比式自监督学习使用增强样本[4,5,7,10,16,26,29,30]构成正样本对和负样本对。常用的 InfoNCE 损失 [29]由对齐项和均匀性项组成：前者最大化正样本对之间的相似度，后者最小化负样本对之间的相似度 [30]。SimCLR [4]是一种先进的对比式自监督学习方法。然而，为了获得有效的表征，SimCLR 需要大量负样本对 [4],，换言之，需要较大的批次（或内存库） n。这可能成为计算瓶颈，因为 SimCLR 的损失计算需要 $O(n^2d)$ 时间，其中d 表示投影嵌入的维度。

使用非对称架构的非对比式自监督学习。近年来，研究人员开始探索非对比式自监督学习方法，即训练时不使用负样本对的方法。为克服坍塌嵌入，BYOL [14]和 SimSiam [6] 等模型在架构中引入了非对称性，例如，抑制梯度更新，和/或对孪生网络的一个分支使用网络参数的移动平均。这些方法的动机源于启发式，因为它们并未明确惩罚坍塌嵌入；然而，它们在实践中十分有效。

基于去相关正则化的非对比式自监督学习。
有一类工作保留标准的（对称）孪生网络，但引入损失函数来抑制坍塌嵌入。这些损失函数还包含用于促进特征去相关的正则化项。Barlow Twins [33] 是这一类方法中的首个方法。它使用基于两个视图之间互相关的去相关正则项。VICReg [1] 使用根据单个视图的协方差矩阵定义的正则项。我们在第 3节中详细回顾这些方法。

通过白化实现非对比式自监督学习。一些作者 [11,19] 在训练过程中使用白化来显式地去相关特征，而不是进行正则化。后续工作 [34] 对特征和实例都进行了白化。由于这些方法所采用的白化过程需要计算协方差矩阵的所有特征值，因此一个训练轮次需要 $O(\min(dn^2, nd^2))$时间；当 d 或 n较大时，这种做法效率低下。

机器学习中的卷积。 卷积是卷积神经网络（卷积神经网络）的基本构建模块。卷积神经网络对输入向量与小型可学习卷积核进行（线性）卷积，以提取局部特征。尽管快速傅里叶变换降低了卷积计算的渐近复杂度，但它很少用于卷积神经网络，因为卷积核通常太小，无法从快速傅里叶变换的加速中获益。与卷积神经网络不同，我们使用（循环）卷积来计算协方差矩阵和互相关矩阵的汇总统计量。

在机器学习的其他领域，循环卷积及其非交换类比形式——循环相关——已用于实现联想记忆 [3,22,24]。这一思想最近被应用于知识图谱嵌入（KGEs） [20],，由此得到的模型后来被证明 [15] 与复值知识图谱嵌入 [27] 同构，这一结论借助卷积定理得出。

3. 使用去相关正则化器的自监督学习

本节回顾 Barlow Twins [33] 和 VICReg [1]。给定一个包含 n 个原始训练样本的批次，这些模型会对每个样本应用两种随机选取的变换。随后，神经网络会分别处理变换后的样本，为每个原始样本生成一对孪生嵌入（视图）。令$\mathbf{a}^{(k)}, \mathbf{b}^{(k)} \in \mathbb{R}^d$ 表示由此获得的第 k个样本的孪生嵌入，并令 $\mathrm{A} = \{\mathbf{a}^{(k)}\}_{k=1}^n$、$\mathrm{B} = \{\mathbf{b}^{(k)}\}_{k=1}^n$ 分别表示单个视图的嵌入集合。给定这些嵌入，训练神经网络以最小化各模型特定的损失函数。

Barlow Twins。 Barlow Twins 优化基于视图 A 和 B 之间的互相关矩阵$\mathbf{C}(\mathrm{A},\mathrm{B}) \in \mathbb{R}^{d\times d}$ 定义的损失函数：

$$L_{\mathrm{BT}} = \sum_{i=0}^{d-1} (1 - [\mathbf{C}(\mathrm{A}, \mathrm{B})]_{ii})^2 + \lambda R_{\mathrm{off}}(\mathbf{C}(\mathrm{A}, \mathrm{B})), \qquad (1)$$

其中，超参数 $\lambda \geq 0$ 控制正则化强度，而 R_{off}：$\mathbb{R}^{d\times d} \to \mathbb{R}$ 是定义如下的正则化函数

$$R_{\mathrm{off}}(\mathbf{M}) = \sum_{i=0}^{d-1} \sum_{\substack{j=0 \\ j\neq i}}^{d-1} [\mathbf{M}]_{ij}^2. \qquad (2)$$

当两个视图中对应的特征完全相关时，式（1）中的第一项达到最小值，即

$[\mathbf{C}(\mathrm{A},\mathrm{B})]_{ii} = 1$ 对于 $i = 0,\ldots,d-1$。这一项可在 $O(nd)$ 时间内高效计算。第二项中的正则化项$R_{\text{off}}(\mathbf{C}(\mathrm{A},\mathrm{B}))$负责特征去相关；当$\mathbf{C}(\mathrm{A},\mathrm{B})$中的所有非对角元素均为零时，该项达到最小值。对于较大的 d，$R_{\text{off}}(\mathbf{C}(\mathrm{A},\mathrm{B}))$可能会带来较大的计算负担，因为计算它需要 $O(nd^2)$ 时间，这是由$\mathbf{C}(\mathrm{A},\mathrm{B})$矩阵所导致的。

VICReg。 设$\mathbf{K}(\mathrm{A}),\mathbf{K}(\mathrm{B}) \in \mathbb{R}^{d\times d}$ 分别为 A 和 B 的协方差矩阵。VICReg损失函数为

$$\begin{aligned} L_{\text{VIC}} = \frac{\alpha}{n}\sum_{k=1}^{n}\|\mathbf{a}^{(k)}-\mathbf{b}^{(k)}\|_2^2 \\ + \frac{\mu}{d}\left(R_{\text{var}}(\mathbf{K}(\mathrm{A}))+R_{\text{var}}(\mathbf{K}(\mathrm{B}))\right) \\ + \frac{\nu}{d}\left(R_{\text{off}}(\mathbf{K}(\mathrm{A}))+R_{\text{off}}(\mathbf{K}(\mathrm{B}))\right), \end{aligned} \tag{3}$$

其中，超参数 $\alpha,\mu,\nu \geq 0$ 控制各项的重要性，而 R_{var} 是定义如下的正则化项

$$R_{\text{var}}(\mathbf{M}) = \sum_{i=0}^{d-1}\max(0,\gamma-\sqrt{[\mathbf{M}]_{ii}}), \tag{4}$$

目标标准差为 $\gamma > 0$。函数 R_{off} 与 Barlow Twins（式（2））中使用的正则化项相同，但在这里，它作用于 $\mathbf{K}(\mathrm{A})$ 和 $\mathbf{K}(\mathrm{B})$，而不是$\mathbf{C}(\mathrm{A},\mathrm{B})$。

式（3）中的第一项会使同一示例的两个嵌入彼此靠近。正则化项 R_{var} 会惩罚方差为零的坍塌嵌入，而 R_{off} 则通过鼓励特征协方差为零来促进特征的多样性。计算 $R_{\text{off}}(\mathbf{K}(\mathrm{A}))$ 和 $R_{\text{off}}(\mathbf{K}(\mathrm{B}))$ 的时间复杂度为 $O(nd^2)$，与 Barlow Twins 中计算$R_{\text{off}}(\mathbf{C}(\mathrm{A},\mathrm{B}))$ 的时间复杂度相同。

4. 所提方法

我们提出一种较弱但可高效计算的正则化函数 R_{off} 替代方案。下文中，我们以互相关矩阵$\mathbf{C}(\mathrm{A},\mathrm{B})$的形式介绍正则化项，与 Barlow Twins 类似。不过，若改为将其应用于$\mathbf{K}(\mathrm{A})$和$\mathbf{K}(\mathrm{B})$，则会得到 VICReg 损失中相应正则化项的一个放宽版本。

4.1. 基于互相关和的正则化项

回想一下，Barlow Twins 损失基于两个视图 $\mathrm{A} = \{\mathbf{a}^{(k)}\}_{k=1}^{n}$和 $\mathrm{B} = \{\mathbf{b}^{(k)}\}_{k=1}^{n}$的交叉相关矩阵 $\mathbf{C} = \mathbf{C}(\mathrm{A},\mathrm{B})$。为简洁起见，假设 A 和 B均已标准化。那么，我们可以直接写成 $\mathbf{C} = (1/(n-1))\sum_{k=1}^{n}\mathbf{a}^{(k)}\mathbf{b}^{(k)\mathrm{T}}$。

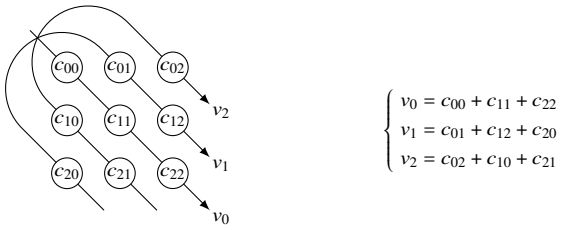


图 1. 一个 3×3 互相关矩阵 $\mathbf{C} = [c_{ij}](i,j=0,1,2)$ 及 $\text{sumvec}(\mathbf{C}) = [v_0\ v_1\ v_2]^{\mathrm{T}}$。

我们的正则化项根据矩阵 $\mathbf{C}$ 的 d 维“摘要”向量来定义。这个向量记作 $\text{sumvec}(\mathbf{C})$，其各分量定义如下：

$$[\text{sumvec}(\mathbf{C})]_i = \sum_{j=0}^{d-1}[\mathbf{C}]_{j,(i+j) \bmod d}. \tag{5}$$

注意，分量索引从零开始。第零个分量 $[\text{sumvec}(\mathbf{C})]_0$ 是$\mathbf{C}$的迹。其余每个$d-1$ 分量对应于 d 个不同的非对角元素之和$\mathbf{C}$，且任意两个不同的和中都不会出现同一个元素。因此，$\mathbf{C}$ 中的每个元素在式（5）的求和中都恰好出现一次。协方差矩阵的汇总向量计算示意见图1。

现在，我们根据 $\text{sumvec}(\mathbf{C})$ 中除第零个分量外的所有分量定义一个正则化项：

$$R_{\text{sum}}(\mathbf{C}) = \sum_{i=1}^{d-1}\|[\text{sumvec}(\mathbf{C})]_i\|_q^q, \tag{6}$$

其中超参数 $q \in \{1,2\}$。此函数 R_{sum} 可直接替代 R_{off}，用于 Barlow Twins 损失（式（1））。式（$[\text{sumvec}(\mathbf{C})]_0$（6）中排除了第零个分量，因为它涉及 $\mathbf{C}$ 的对角元素，而这些元素与特征去相关无关；它们也不会出现在 Barlow Twins 的正则化项 $R_{\text{off}}(\mathbf{C})$ 中。

正则化项 R_{sum} 比 R_{off} 弱，因为它对汇总向量的分量，即 d **$\mathbf{C}$ 中元素的和施加约束**，而 R_{off} 约束的是单个元素。事实上，$R_{\text{off}}(\mathbf{C})$ 的极小化解也会使 $R_{\text{sum}}(\mathbf{C})$达到极小值，但反之未必成立。不过，正如我们在第4.2节所讨论的，R_{sum} 能够加快计算。此外，在第 4.3节中，我们提出了一种简单的方法来减轻 R_{sum}的弱点。

4.2. 高效计算

根据公式计算 $\text{sumvec}(\mathbf{C})$ 需要用到互相关矩阵5，其计算成本与 Barlow Twins 损失相同。不过，借助傅里叶变换，$\text{sumvec}\mathbf{C}(\mathrm{A},\mathrm{B})$可以$(\mathbf{C})$

直接通过 A 和 B 中的向量计算，而无需显式计算它们的互相关矩阵。为此，我们使用对合和循环卷积。

对于向量 $\mathbf{x} \in \mathbb{R}^d$，其对合 [24]（也称为翻转 [25]）$\mathrm{inv}(\mathbf{x})$是将其第一个（而非第零个）分量至第 $d-1$ 个分量的顺序反转后得到的向量；即$[\mathrm{inv}(\mathbf{x})]_i = [\mathbf{x}]_{(d-i) \bmod d}$，其中 $i = 0, \ldots\ldots, d-1$。

对于向量 $\mathbf{x}$，$\mathbf{y} \in \mathbb{R}^d$，它们的循环卷积 $\mathbf{x} * \mathbf{y}$ 是一个 d 维向量，其分量定义为

$$[\mathbf{x} * \mathbf{y}]_i = \sum_{j=0}^{d-1} \left[\mathbf{x}\mathbf{y}^{\mathrm{T}}\right]_{j,(i-j) \bmod d}. \tag{7}$$

因此，循环卷积也被称为“压缩外积”。

现在，对于每个孪生嵌入对 $\mathbf{a}^{(k)} \in$ A 和 $\mathbf{b}^{(k)} \in$B ($k = 1, \ldots\ldots, n$)，我们考虑向量 $\mathrm{inv}(\mathbf{a}^{(k)}) * \mathbf{b}^{(k)} \in \mathbb{R}^d$。[2] 注意对合改变的索引，可知该向量的各分量为

$$\left[\mathrm{inv}(\mathbf{a}^{(k)}) * \mathbf{b}^{(k)}\right]_i = \sum_{j=0}^{d-1} \left[\mathbf{a}^{(k)}\mathbf{b}^{(k)\mathrm{T}}\right]_{j,(i+j) \bmod d}. \tag{8}$$

将 $\mathbf{C} = (1/(n-1)) \sum_{k=1}^{n} \mathbf{a}^{(k)}\mathbf{b}^{(k)\mathrm{T}}$ 代入方程，并使用方程 (8)，得到 (5)

$$\begin{aligned}
[\mathrm{sumvec}(\mathbf{C})]_i &= \sum_{j=0}^{d-1} \overbrace{\left[\frac{1}{n-1}\sum_{k=1}^{n} \mathbf{a}^{(k)}\mathbf{b}^{(k)\mathrm{T}}\right]}^{\mathbf{C}}_{j,(i+j) \bmod d} \\
&= \frac{1}{n-1}\sum_{k=1}^{n}\sum_{j=0}^{d-1} \left[\mathbf{a}^{(k)}\mathbf{b}^{(k)\mathrm{T}}\right]_{j,(i+j) \bmod d} \\
&= \frac{1}{n-1}\sum_{k=1}^{n} \left[\mathrm{inv}(\mathbf{a}^{(k)}) * \mathbf{b}^{(k)}\right]_i, \qquad (9)
\end{aligned}$$

或者，作为一个向量，

$$\mathrm{sumvec}(\mathbf{C}) = \frac{1}{n-1}\sum_{k=1}^{n} \mathrm{inv}(\mathbf{a}^{(k)}) * \mathbf{b}^{(k)}. \tag{10}$$

令 $\mathcal{F}$ 和 $\mathcal{F}^{-1}$分别表示离散傅里叶变换和逆离散傅里叶变换。注意到$\mathcal{F}(\mathrm{inv}(\mathbf{x})) = \mathcal{F}(\mathbf{x})$ 对任意 $\mathbf{x} \in \mathbb{R}^d$ 均成立（例如参见 [25], 第 7.4.2 节），并利用卷积定理 $\mathcal{F}(\mathbf{x} * \mathbf{y}) = \mathcal{F}(\mathbf{x}) \circ \mathcal{F}(\mathbf{y})$，可得

$$\mathrm{inv}(\mathbf{a}^{(k)}) * \mathbf{b}^{(k)} = \mathcal{F}^{-1}\left(\overline{\mathcal{F}(\mathbf{a}^{(k)})} \circ \mathcal{F}(\mathbf{b}^{(k)})\right). \tag{11}$$

[2]向量 $\mathrm{inv}(\mathbf{x}) * \mathbf{y}$ 称为x 与$\mathbf{y}$ 的**循环（互）相关** [22, 24, 25]。为避免与 Barlow Twins 中使用的随机向量互相关概念混淆，本文不采用这一术语。

将式（11）代入式（10），得到

$$\begin{aligned}
\mathrm{sumvec}(\mathbf{C}) &= \frac{1}{n-1}\sum_{k=1}^{n} \overbrace{\mathcal{F}^{-1}\left(\overline{\mathcal{F}(\mathbf{a}^{(k)})} \circ \mathcal{F}(\mathbf{b}^{(k)})\right)}^{\mathrm{inv}(\mathbf{a}^{(k)}) * \mathbf{b}^{(k)}} \\
&= \frac{1}{n-1}\mathcal{F}^{-1}\left(\sum_{k=1}^{n} \overline{\mathcal{F}(\mathbf{a}^{(k)})} \circ \mathcal{F}(\mathbf{b}^{(k)})\right). \qquad (12)
\end{aligned}$$

利用这个公式，我们可以直接根据嵌入向量 $\{\mathbf{a}^{(k)}, \mathbf{b}^{(k)}\}_{k=1}^{n}$计算 $\mathrm{sumvec}(\mathbf{C})$，从而绕过$\mathbf{C}$的繁琐计算。注意，一个 d维向量的（逆）傅里叶变换可通过快速傅里叶变换算法在 $O(d \log d)$ 时间内完成，而计算复共轭、分量乘积以及 n 个向量的求和需要 $O(nd)$ 时间。因此，求得 $\mathrm{sumvec}(\mathbf{C})$ 总共需要 $O(nd \log d)$ 时间，这也是计算 $R_{\mathrm{sum}}(\mathbf{C})$ 所需的时间。相比之下，Barlow Twins 损失中计算$R_{\mathrm{off}}(\mathbf{C})$需要 $O(nd^2)$ 时间，前者有了显著改进。

此外，计算$R_{\mathrm{sum}}(\mathbf{C})$ 的空间复杂度为 $O(nd)$；若将存储输入向量 A 和 B 所需的 $O(nd)$ 空间也计入复杂度，则该复杂度是最优的。相比之下，Barlow Twins 需要额外的 $O(d^2)$ 空间来存储 $\mathbf{C}$。

4.3. 通过特征置换缓解不良局部极小值

从式（5）可见，$\mathrm{sumvec}(\mathbf{C})$ 的各个分量是 d 个$\mathbf{C}$中元素的和，而所提出的正则化项R_{sum}（式（6））鼓励这些和接近于零。这比 Barlow Twins 损失中的正则化项 R_{off}（式（2））约束更弱；后者会推动 $\mathbf{C}$ 中的各个元素趋近于零。事实上，即使 $R_{\mathrm{sum}}(\mathbf{C})$ 接近于零，$\mathbf{C}$ 中的各个元素也未必接近于零；式（5）中的加数可以相互抵消，因为它们既可能为正，也可能为负。因此，参数空间中会出现不良局部极小值，使我们的正则化项失效。

这里，我们提出一种消除这些局部极小值的技术：在训练过程中随机排列特征索引，使不同交叉相关项的和构成 $\mathrm{sumvec}(\mathbf{C})$。为理解这一简单技术为何有效，考虑在将$\mathbf{C}$ 的元素视为相互独立的[3] 变量时，最小化 $R_{\mathrm{sum}}(\mathbf{C})$。不难看出，最小值由以下齐次线性方程组的解取得：

$$\sum_{j=0}^{d-1} \overbrace{[\mathrm{sumvec}(\mathbf{C})]_i}^{[\mathbf{C}]_{j,(i+j) \bmod d}} = 0, \qquad \text{for } i = 1, \ldots, d-1.$$

[3]鉴于现代神经网络具有很强的容量，这一假设并非不切实际。

这是一个欠定系统，只有 $d-1$ 个方程，却有 $d(d-1)$ 个未知数，即 $[\mathbf{C}]_{jl}$、 $j, l = 0, \ldots, d-1, j \neq l$。这种欠定性会导致非平凡解，使得 $[\mathbf{C}]_{j\ell} \neq 0$，也就是说，出现不理想的解：方程中符号相反的加数彼此抵消。

现在，通过反复排列特征索引并最小化损失，我们实际上向系统引入了额外的方程，因为排列可以针对未知数产生不同的线性方程组，而这些新约束最终会使非平凡解变得不可行。

为便于实现，我们在训练过程中随机排列特征索引，而不是一次性系统地生成所有排列，这类似于随机梯度下降是对全量梯度下降的一种改进。请注意，不同小批次中的特征索引排列不必相同，即使是在同一个训练轮次内也不必相同；事实上，在第 5节所展示的实验中，我们在每个训练轮次的每个小批次中都使用不同的随机特征排列。

4.4. 通过特征分组控制松弛程度

我们不必为整个互相关矩阵 $\mathbf{C}$计算汇总向量，而可以在更细粒度的层面上计算汇总量。具体而言，我们将 d 个特征划分为若干组，每组包含 b 个特征[4]。这种划分会在 $\mathbf{C}$ 中产生总计 $\lceil d/b \rceil^2$ 个大小为 $b \times b$ 的分块子矩阵，即 $\mathbf{C} = [\mathbf{C}_{ij}](i, j = 1, \ldots\ldots, \lceil d/b \rceil)$，其子矩阵为 $\mathbf{C}_{ij} \in \mathbb{R}^{b \times b}$。随后，我们将正则化项定义为

$$R^{(b)}_{\text{sum}}(\mathbf{C}) = \sum_{i=1}^{\lceil d/b \rceil} \sum_{\ell=1}^{b-1} \|[\text{sumvec}(\mathbf{C}_{ii})]_\ell\|_q^q + \sum_{\substack{i,j=1 \\ i \neq j}}^{\lceil d/b \rceil} \sum_{\ell=0}^{b-1} \left\|[\text{sumvec}(\mathbf{C}_{ij})]_\ell\right\|_q^q . \tag{13}$$

与之前一样，$\text{sumvec}(\mathbf{C}^{ij})$ 可以通过对合、循环卷积（对嵌入的子向量进行卷积）和傅里叶变换来计算，而无需显式计算 $\mathbf{C}^{ij}$ 。使用 FFT 计算单个 $\text{sumvec}(\mathbf{C}^{ij})$ 需要 $O(nb \log b)$ 的时间，并且由于共有 $\lceil d/b \rceil^2$ 个分块，计算 $R^{(b)}_{\text{sum}}$ 所需的总时间为 $O((nd^2/b) \log b)$。

块大小超参数 b 控制汇总计算的粒度，并在所提出的正则化项与现有正则化项之间进行插值。一方面，当 $b = 1$ 且满足 $q = 2$ 时，正则化项 $R^{(1)}_{\text{sum}}(\mathbf{C})$ 会降低 Barlow Twins 的 $R_{\text{off}}(\mathbf{C})$。另一方面，当 $b = d$ 时，我们恢复 $R^{(d)}_{\text{sum}}(\mathbf{C}) = R_{\text{sum}}(\mathbf{C})$ （见式（6））。因此，这种分组表述将 Barlow Twins 推广开来，参数 b 用于控制计算效率与正则化放宽程度之间的权衡。实验表明，采用中等大小的特征组可以略微提升性能，且训练时间和内存使用量没有明显恶化；详见第 5节。

[4]如果 d 不能被 b 整除，则在最后一个分组中填充始终为 0 的虚拟特征。

需要注意的是，特征的排列和分组是兼容的，可以合并进行。

4.5. 基于特征协方差和的正则化项

函数 R_{sum} 还可以用于定义一种基于协方差的 VICReg 风格正则化项，只需将R_{off} 替换为 R_{sum} ，并在公式（3）中传入相关矩阵$\mathbf{K}$(A) 或 $\mathbf{K}$(B) ，而不是将 $\mathbf{C}$(A,B) 作为参数传入。使用 FFT 可以实现快速计算，分组版本也很简单；补充材料对此进行了更详细的说明。

4.6. 所提出的模型总结

所提出模型的损失函数总结如下。若使用特征分组且块大小为 b，则令函数为 $R = R^{(b)}_{\text{sum}}$；若不使用分组，则令函数为 $R = R_{\text{sum}}$。

对于 Barlow Twins 风格的互相关正则化，损失函数为

$$L = \sum_i (1 - [\mathbf{C}(\text{A}, \text{B})]_{ii})^2 + \lambda R(\mathbf{C}(\text{A}, \text{B})), \tag{14}$$

而对于 VICReg 风格的协方差正则化，我们使用

$$L = \frac{\alpha}{n} \sum_i \|\mathbf{a}^{(i)} - \mathbf{b}^{(i)}\|_2^2 + \frac{\mu}{d} \left(R_{\text{var}}(\mathbf{K}(\text{A})) + R_{\text{var}}(\mathbf{K}(\text{B}))\right) + \frac{\nu}{d} \left(R(\mathbf{K}(\text{A})) + R(\mathbf{K}(\text{B}))\right). \tag{15}$$

在这些公式中令 $R = R_{\text{off}}$，当超参数为 $q = 2$ 时，即可得到原始的 Barlow Twins 和 VICReg。

5. 实验

我们通过实证评估所提出的正则化项的效果。具体而言，我们使用式（14）中的 Barlow Twins 风格损失函数和式（15）中的 VICReg 风格损失函数训练自监督学习模型，并将其在下游任务中的表现与 Barlow Twins 和 VICReg 进行比较。我们还评估了训练时间和内存消耗。对于我们的模型，除消融研究外，每次批次迭代都会进行特征置换。

下面简要介绍实验中使用的任务和数据集。完整的实验设置，包括超参数取值和补充实验结果，请参阅附录 D 和 E 。

表 1。在 ImageNet-100 上的线性评估准确率（%），使用 $d = 2048$。粗体数字表示各类别（交叉相关正则化、协方差正则化或其他自监督学习模型）中的最佳性能。 †：引自 solo-learn [28] GitHub 代码库，截至2022年12月28日；‡：引自 [34]。

模型	Top-1	Top-5
Barlow Twins†	80.16	95.14
Barlow Twins	80.12	**95.24**
所提出的（Barlow Twins 风格；不分组）	79.94	94.76
所提出的（Barlow Twins 风格； $b = 128$）	**81.02**	**95.24**
VICReg†	79.40	**95.02**
VICReg	79.30	94.30
所提出的（VICReg 风格；不分组）	79.20	94.96
所提出的（VICReg 风格； $b = 128$）	**80.04**	94.98
W-MSE† [11]	69.06	91.22
Zero-FCL‡ [34]	79.32	94.94
Zero-CL‡ [34]	79.26	94.98
NNCLR† [10]	80.16	**95.30**
BYOL† [14]	80.32	94.94
MoCo V3† [7]	**80.36**	94.96

5.1. 任务与数据集

根据实验不同，模型使用 ImageNet 数据集 [9]或其子集 ImageNet-100 [26], 中的图像进行预训练。ImageNet 使用 ResNet-50 [17] 作为骨干网络，而 ImageNet-100 使用 ResNet-18。

为评估下游自监督学习性能，我们遵循标准线性评估协议：使用自监督学习方法对骨干网络进行预训练后，利用 ImageNet 或 ImageNet-100 训练集中的标注数据，在冻结的骨干网络之上训练线性分类器。随后，在相应的验证集上通过 Top-1 和 Top-5 准确率评估所得分类器。

为进行迁移学习评估，我们将预训练模型应用于 Pascal VOC07+12[12] 上的目标检测任务。遵循先前研究 [1, 16, 33], , 我们使用 VOC2007 和 VOC2012 的 trainval 集进行训练，并使用 VOC2007 的 test 划分进行测试。我们使用 R50-C4 对 Faster R-CNN [23]进行微调。模型通过三种平均精度（AP）进行评估：AP、AP_{50}, 和 AP_{75}, , 其中 AP_x 表示交并比（IoU）阈值为 x %。我们报告五次试验的平均分数。

5.2. 结果与讨论

ImageNet-100 上的线性评估。表1展示了结果。可以看出，所提出的模型的准确率与表中所有现有模型相当，包括 Barlow Twins 和 VICReg，无论是否进行特征分组。

在 ImageNet 上进行线性评估。表2展示了结果。由于这项大规模实验的计算成本较高

表 2. ImageNet 上的线性评估准确率（%）；线性分类头训练 100 个训练轮次期间的最高准确率。 $d = 8192$ 所提出的模型、Barlow Twins 和 VICReg。†：引自相应方法的原始论文；‡：引自 MoCo V3 GitHub 仓库。

模型	训练轮次	Top-1	Top-5
Barlow Twins†	1000	**73.2**	91.0
Barlow Twins	1000	72.4	90.6
所提出的（Barlow Twins 风格；不分组）	1000	73.0	91.2
所提出的（Barlow Twins 风格； $b = 128$）	1000	**73.2**	**91.3**
VICReg†	1000	**73.2**	**91.1**
VICReg	1000	72.6	90.9
所提出的（VICReg 风格；无分组）	1000	72.8	**91.1**
W-MSE 4† [11]	400	72.6	—
Zero-CL† [34]	400	72.6	90.5
SimCLR† [4]	1000	69.3	89.0
NNCLR† [10]	1000	**75.4**	**92.3**
BYOL† [14]	1000	74.3	91.6
MoCo V3‡ [7]	1000	74.6	—

表3. VOC07+12 上目标检测的迁移学习结果。†：引自原始论文；‡：引自 [16]。

模型	AP_{50}	AP	AP_{75}
监督学习‡	81.3	53.5	58.8
Barlow Twins†	**82.6**	**56.8**	**63.4**
所提出的（Barlow Twins 风格；无分组）	82.5	55.0	61.1
VICReg†	**82.4**	—	—
所提出的（VICReg 风格；无分组）	82.3	56.1	62.1

我们没有评估采用 VICReg 风格正则化的特征分组。所提出的模型表现略逊于 NNCLR、BYOL 和 MoCo V3，但与 Barlow Twins 和 VICReg 的表现相当。

在 Pascal VOC 目标检测任务上的迁移学习评估。表3 展示了迁移学习的结果。同样，所提出的模型表现与 Barlow Twins 和 VICReg 相当。

ImageNet 上的训练时间。表4展示了在 ImageNet 上训练 1000 个训练轮次所需的时间。我们测试了两种情况：使用 8 个和 4 个图形处理器进行训练。在这两种情况下，每个图形处理器的批量大小均设为 128，因此，使用 8 个图形处理器时的有效批量大小为 1024，使用 4 个图形处理器时则为 512。如表4所示，所提出的模型准确率与 Barlow Twins 相当，同时训练时间显著缩短。[5] 训练速度的更精确评估

[5]该实验在商业云平台上进行，该平台将每个会话的时长限制为最多三天。使用 8 个图形处理器训练 Barlow Twins 1000 个训练轮次需要三个会话，而

表4。在 ImageNet 上使用 ResNet-50 主干网络进行线性评估的准确率（%）和总训练时间（$d = 8192$）。每个图形处理器的批量大小为 128。所提出的表示使用 Barlow Twins 风格正则化项且不进行分组的所提方法。Barlow Twins* 表示引用自 [33, 图 2]的结果。

图形处理器数量（批量大小）	模型	Top-1	Top-5	Time
8（1024）	Barlow Twins*	**73.2**	91.0	—
	Barlow Twins	72.4	90.7	6天14小时0分钟
	所提出的	73.0	**91.3**	5天14小时58分钟
4 (512)	Barlow Twins*	—	—	—
	Barlow Twins	72.1	90.2	12天 19小时 30分钟
	所提出的	**72.8**	**91.2**	10天 21小时 6分钟

表 5. 特征置换的影响：ImageNet-100 上的准确率（%）和每 10 个训练轮次的训练时间（秒）。

（a）Barlow Twins 风格的互相关正则化

分组	排列	Top-1	Top-5	Time
no	no	59.64	85.20	**1646.2**
	yes	**79.94**	**94.76**	1668.7
$b = 128$	no	73.58	93.36	**1697.5**
	yes	**81.02**	**95.24**	1709.6

(b) VICReg 风格的协方差正则化

分组	排列	Top-1	Top-5	Time
no	no	57.42	84.26	**1692.2**
	yes	**79.20**	**94.96**	1718.0
$b = 128$	no	66.26	89.68	**1802.1**
	yes	**80.04**	**94.98**	1813.3

将在下一个实验中给出，更多结果见附录E.2至E.4。

嵌入维度与计算成本。
图 2展示了在单个图形处理器上，ImageNet-100 数据集经过 10 个训练轮次的经过时间和峰值图形处理器内存分配情况，投影嵌入的维度分别为： $d \in \{2048、4096、8192、16384\}$。随着 d的增加，相较于 Barlow Twins 和 VICReg 的改进变得更加明显；在 $d = 8192$时，所提出的模型（不使用分组）的速度分别是 VICReg 的 2.8 倍和 Barlow Twins 的 2.2 倍；而在 $d = 16384$时，其速度分别是 VICReg 的 5.7 倍和 Barlow Twins 的 4.0 倍。对于 $d = 8192$ 和 16384，内存消耗均减少了一半以上。有关 ResNet-50 主干网络在多 GPU 上的结果，以及前向传播、反向传播和损失计算时间的详细信息，请参见附录 E.2 和E.4 。

所提出的模型需要两个会话。表 4中报告的时间是这两个会话的总运行时间，其中包括每个会话开始时重新初始化所需的时间（3 到 5 秒）。

表 6。将 Barlow Twins 和 VICReg 的正则化项应用于所提出的模型生成的嵌入。置换：排列；差异：与基线方法的差异。

(a) Barlow Twins 风格的互相关正则化

模型	分组	置换	归一化 Barlow 双胞胎损失（式（16））	Diff
Barlow Twins	—	—	0.005	0
所提出的	no	no	0.564	0.559
		yes	**0.010**	**0.005**
	$b = 128$	no	0.049	0.044
		yes	**0.009**	**0.004**

（b）VICReg 风格的协方差正则化

模型	分组	置换。	归一化 VICReg损失（式（17））	Diff
VICReg	—	—	0.002	0
所提出的	no	no	1.999	1.997
		yes	**0.011**	**0.009**
	$b = 128$	no	0.379	0.377
		yes	**0.007**	**0.005**

特征置换的有效性。 表5 展示了在 ImageNet-100 数据集、 d =2048维度下特征置换的影响。无论是否使用分组，不进行置换都会导致准确率显著下降，这表明特征置换对于所提出正则化项的有效性至关重要。如表5中的“时间”列所示，尽管置换的执行频率甚至达到每次批次迭代一次，其开销仍可忽略不计。

我们还定量评估了通过特征置换实现的去相关程度。所提出的模型训练完成后，我们计算应用于所提出的模型输出的嵌入 A， B 上的 Barlow Twins 和 VICReg 正则化项的归一化值，如下所示：

$$\frac{R_{\text{off}}(\mathbf{C}(\mathrm{A},\mathrm{B}))}{d(d-1)} \tag{16}$$

$$\frac{R_{\text{off}}(\mathbf{K}(\mathrm{A})) + R_{\text{off}}(\mathbf{K}(\mathrm{B}))}{2d(d-1)} \tag{17}$$

归一化因子 $d(d-1)$ 用于使所得数值成为 K(A)、K(B) 和 C(A, B) 的 $d(d-1)$ 个非对角元素的均值。由于 VICReg 对 K(A) 和 K(B) 均设有正则化项，其损失还要再除以 2。表 6 展示了 ImageNet-100（d=2048）上的结果。从基线损失函数的角度来看，我们观察到特征置换能够促进去相关。

特征分组中块大小的影响。为了评估特征分组对 ImageNet-100的影响，我们将嵌入维度固定为 $d = 2048$ ，并调整块

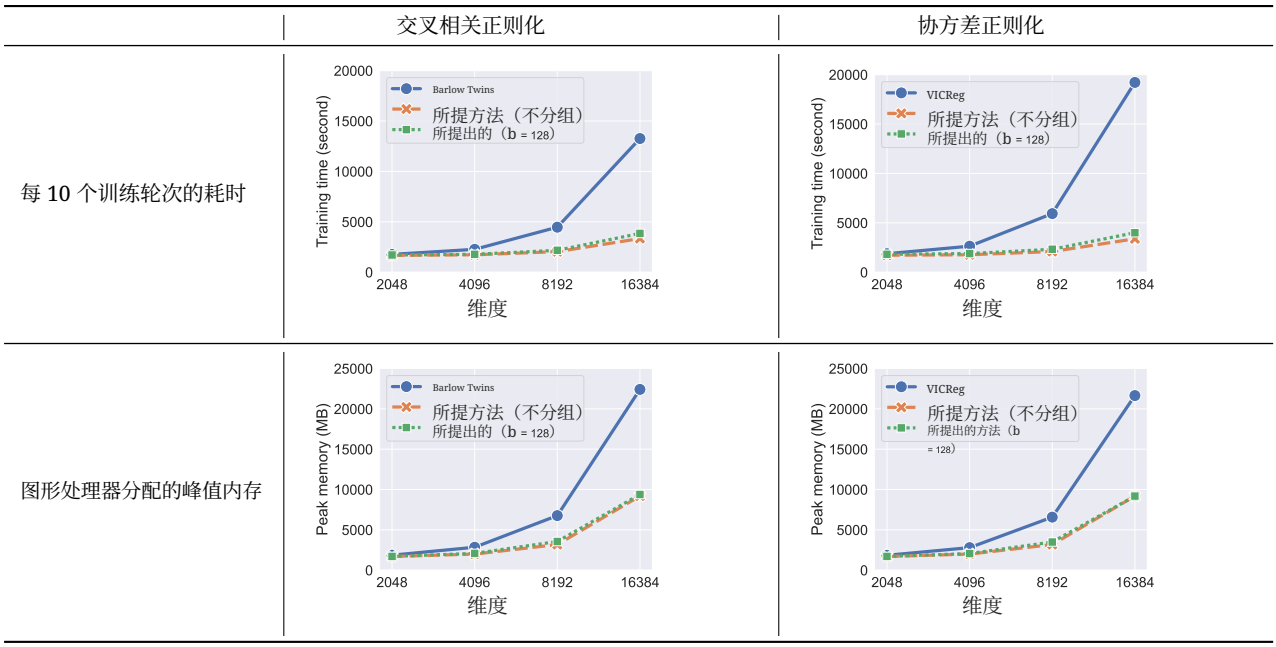
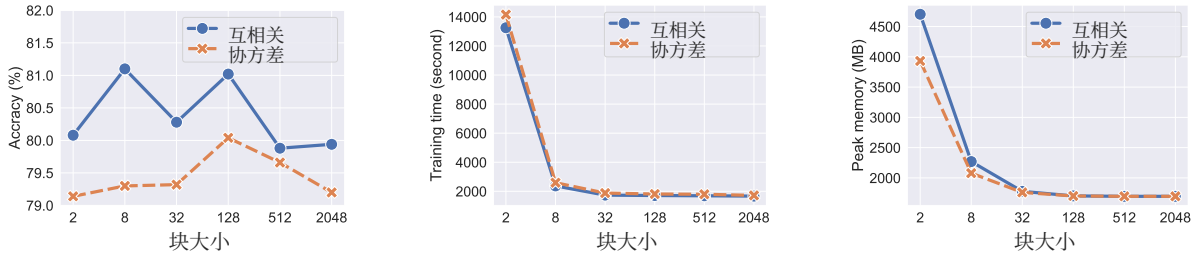


图2。在单个图形处理器上使用 ResNet-18 对 ImageNet-100 进行训练时的训练时间和内存使用量。



(a) Top-1 准确率　(b) 已用训练时间　(c) 图形处理器峰值显存占用

图 3. 块大小对 ImageNet-100 的影响。x 轴表示块大小 b。

大小 $b \in \{2, 4, 8, \ldots, 2048\}$。块大小 $b = d = 2048$ 对应于不进行特征分组。图3 展示了结果，表明除非 b 极小（即不超过 8），否则训练时间或图形处理器内存使用量都不会显著增加。将 b 设为适中的大小，例如$b = 128$，可以提升性能。

6. 结论

我们为非对比式自监督学习提出了一种新的去相关正则项。给定 n 个 d 维嵌入，该正则项可在 $O(nd \log d)$ 时间内计算完成，相较于现有模型（Barlow Twins 和 VICReg）所需的 $O(nd^2)$ 时间有所改进。内存消耗的降低使得可以使用更大的批量大小，从而通常能够训练出更有效的模型。我们还提出了一种特征置换技术，以缓解该正则项的不足之处。

借助我们的正则化项，采用轻量级骨干网络的网络所获得的加速最为显著，因为在这类网络中，损失节点的计算占据了训练时间的很大一部分。这使得我们的方法成为与知识蒸馏骨干网络搭配使用的理想选择 [13, 18]。

所提出的方法不仅适用于自监督学习，也可能适用于涉及去相关的更广泛应用。我们计划在未来的工作中继续探索这一方向。

致谢

感谢匿名审稿人提出的宝贵意见。本研究部分得到了新能源与工业技术开发组织（NEDO）以及日本学术振兴会科研费资助项目19H04173的支持。

参考文献

[1]Adrien Bardes、Jean Ponce 和 Yann LeCun。VICReg：用于自监督学习的方差-不变性-协方差正则化。发表于 ICLR，2022。1 、 2 、 6 、 13 、 14[2] Lukas Biewald。使用 Weights and Biases 进行实验跟踪。软件可从 https://www.wandb.com/ 获取，2020。12[3] A. Borsellino 和 T. Poggio。卷积与相关代数。Kybernetik，13(2):113–122，1973。2[4] Ting Chen、Simon Kornblith、Mohammad Norouzi 和 Geoffrey Hinton。用于视觉表征对比学习的简单框架。发表于国际机器学习大会，页码 1597–1607，2020。1、2、6[5] Ting Chen、Simon Kornblith、Kevin Swersky、Mohammad Norouzi 和 Geoffrey E. Hinton。大型自监督模型是强大的半监督学习器。发表于 NeurIPS，页码 22243–22255，2020。1、2[6] Xinlei Chen 和 Kaiming He。探索简单的孪生表示学习。发表于 CVPR，页码 15750–15758，2021。1、2 – 。[7] Xinlei Chen、Saining Xie 和 Kaiming He。自监督视觉Transformer训练的实证研究。发表于 ICCV，页码 9640–9649，2021。2、6[8] James W. Cooley 和 John W. Tukey。用于机器计算复傅里叶级数的算法。Mathematics of Computation，19(90):297–301，1965。1[9] Jia Deng、Wei Dong、Richard Socher、Li-Jia Li、Kai Li 和 Li Fei-Fei。ImageNet：大型层次化图像数据库。发表于 CVPR，页码 248–255，2009。6[10] Debidatta Dwibedi、Yusuf Aytar、Jonathan Tompson、Pierre Sermanet 和 Andrew Zisserman。有朋友相助：视觉表征的最近邻对比学习。发表于 ICCV，页码 9588–9597，2021。2 、 6[11] Aleksandr Ermolov、Aliaksandr Siarohin、Enver Sangineto 和 Nicu Sebe。用于自监督表示学习的白化。发表于国际机器学习大会，页码 3015–3024，2021。1、2、6[12] Mark Everingham、Luc Van Gool、Christopher KI Williams、John Winn 和 Andrew Zisserman。PASCAL 视觉对象类别（VOC）挑战赛。International Journal of Computer Vision，88(2):303–338，2010。6[13] Yuting Gao、Jia Xin Zhuang、Shaohui Lin、Hao Cheng、Xing Sun、Ke Li 和 Chunhua Shen。DisCo：通过蒸馏对比学习改善轻量级模型的自监督学习。发表于 ECCV，页码 237–253，2022。8[14] Jean-bastien Grill、Florian Strub、Florent Altché、Corentin Tallec、Pierre H Richemond、Elena Buchatskaya、Carl Doersch、Bernardo Avila Pires、Zhaohan Daniel Guo、Mohammad Gheshlaghi Azar、Bilal Piot、Koray Kavukcuoglu、Rémi Munos 和 Michal Valko。Bootstrap your own latent：一种自监督学习新方法。发表于 NeurIPS，页码 21271–21284，2020。1、2、6[15] Katsuhiko Hayashi 和 Masashi Shimbo。关于全息嵌入与用于链接预测的复数嵌入的等价性。发表于 ACL '17，页码 554–559，2017。2[16] Kaiming He、Haoqi Fan、Yuxin Wu、Saining Xie 和 Ross Girshick。用于无监督视觉表征的动量对比表示学习。在 CVPR 上，第 9729–9738 页，2020 年。1, 2, 6, 14[17] Kaiming He、张祥雨、任少卿和孙剑。用于图像识别的深度残差学习。在 CVPR 上，第 770–778 页，2016 年。6[18] Geoffrey Hinton、奥里奥尔·维尼亚尔斯和杰夫·迪恩。神经网络知识蒸馏。arXiv 预印本 arXiv:1503.02531，2015 年。8[19] 华天宇、王文骁、薛子晖、王悦、任苏澄和赵航。关于自监督学习中的特征去相关。在 ICCV 上，第 9598–9608 页，2021年。2[20] 马克西米利安·尼克尔、洛伦佐·罗萨斯科和托马索·波焦。知识图谱的全息嵌入。在人工智能促进协会年会上，第 1955–1961 页，2016 年。2[21] 亚当·帕斯克、萨姆·格罗斯、弗朗西斯科·马萨、亚当·勒勒、詹姆斯·布拉德伯里、格雷戈里·查南、特雷弗·基利恩、林泽明、娜塔莉娅·吉梅尔申、卢卡·安蒂加、阿尔班·德迈松、安德烈亚斯·科普夫、杨爱德、扎卡里·德维托、Martin Raison、阿利汗·特贾尼、萨桑克·奇拉姆库尔蒂、伯努瓦·施泰纳、方璐、白俊杰和苏米特·钦塔拉。PyTorch：一种命令式风格的高性能深度学习库。在 NeurIPS 上，第 8024–8035 页，2019 年。12[22] 托尼·普莱特。全息约简表示：认知结构的分布式表示。CSLI 讲义第 150 号。CSLI出版社，第二版，2003 年。2, 4, 11[23] 任少卿、Kaiming He、Ross Girshick 和孙剑。Faster R-CNN：利用区域提议网络实现实时目标检测。在神经信息处理系统大会上，第 91–99 页，2015 年。6[24] P. H. Schönemann。反演、卷积和相关之间的一些代数关系及其在全息记忆中的应用。《生物控制论》，56 374，1987 367 2, 4 : – . [25] Julius O. Smith, III。离散傅里叶变换（DFT）的数学及其音频应用。W3K出版社，第二版，2008 年。4, 11[26] 田永龙、迪利普·克里希南和菲利普·伊索拉。对比多视角编码。在 ECCV 上，第 776–794 页，2020 年。2, 6[27] Théo Trouillon、约翰内斯·韦尔布尔、塞巴斯蒂安·里德尔、Éric Gaussier 和纪尧姆·布沙尔。用于简单链接预测的复数嵌入。在国际机器学习大会上，第 2071–2080 页，2016 年。2[28] Victor G. Turrisi da Costa、恩里科·菲尼、莫因·纳比、Nicu Sebe 和埃莉萨·里奇。solo-learn：用于视觉表征学习的自监督方法库。《机器学习研究杂志》，23(56):1–6，2022 年。6, 12, 13, 14[29] 亚伦·范登奥尔德、李雅哲和奥里奥尔·维尼亚尔斯。基于对比预测编码的表示学习。arXiv.cs 预印本，1807.03748，2018 年。1, 2[30] 王同舟和菲利普·伊索拉。通过超球面上的对齐与均匀性理解对比表示学习。在国际机器学习大会上，第 9929–9939 页，2020 年。1, 2[31] Yuxin Wu、亚历山大·基里洛夫、弗朗西斯科·马萨、罗婉嫣和 Ross Girshick。Detectron2。https://github.com/facebookresearch/detectron2，2019 年。12

[32] Yang You、Igor Gitman 和 Boris Ginsburg。卷积网络的大批量训练。*arXiv.cs* 预印本，1708.03888，2017 年。13[33] Jure Zbontar、Li Jing、Ishan Misra、Yann LeCun 和 Stéhane Deny。Barlow Twins：通过冗余削减进行自监督学习。在国际机器学习大会上，第 12310–12320 页，2021年。1，2，6，7，13，14[34] Shaofeng Zhang、Feng Zhu、Junchi Yan、Rui Zhao 和 Xi-aokang Yang。Zero-CL：用于无负样本对称对比学习的实例与特征去相关。在*ICLR*上，2022 年。1，2，6

A. 式（8）的推导

d维向量 $\mathbf{x} = [x_0 \cdots x_{d-1}]^{\mathrm{T}}, \mathbf{y} = [y_0 \cdots y_{d-1}]^{\mathrm{T}}$。我们首先证明 $\mathrm{inv}(\mathbf{x}) * \mathbf{y} = \sum_{j=0}^{d-1} x_j y_{(i+j) \bmod d}$。

$$
\begin{aligned}
[\mathrm{inv}(\mathbf{x}) * \mathbf{y}]_i &= \sum_{j=0}^{d-1} [\mathrm{inv}(\mathbf{x})]_j \, y_{(i-j) \bmod d} \\
&= \sum_{j=0}^{d-1} x_{(d-j) \bmod d} \, y_{(i-j) \bmod d} && \because \ [\mathrm{inv}(\mathbf{x})]_j = x_{(d-j) \bmod d} \\
&= \sum_{j'=0}^{d-1} x_{j'} \, y_{(i-(d-j')) \bmod d} && \because \ \text{substituting } j' = (d-j) \bmod d \\
&= \sum_{j'=0}^{d-1} x_{j'} \, y_{(i+j') \bmod d} && \because \ (a-d) \bmod d = a \bmod d \\
&= \sum_{j=0}^{d-1} \left[\mathbf{x}\mathbf{y}^{\mathrm{T}}\right]_{j,(i+j) \bmod d}. && \because \ \text{renaming variable } j' \to j
\end{aligned}
$$

令 $\mathbf{x} = \mathbf{a}^{(k)}$和 $\mathbf{y} = \mathbf{b}^{(k)}$，即可得到第 8 节中的式（4.2）。在文献中（例如 [22, 25]），$\mathrm{inv}(\mathbf{x}) * \mathbf{y}$称为 循环互相关，即 $\mathbf{x}$ 与 $\mathbf{y}$，上述方程通常被作为其定义。

B. 基于特征协方差和的正则化项

如第 4.5 节所述，若在 VICReg 的损失函数中，用 R_{sum} 替换 R_{off} （见式（3）），则可得到基于单个视图协方差矩阵 $\mathbf{K}(\mathrm{A})$、$\mathbf{K}(\mathrm{B})$ 的正则化项。针对 $\mathbf{K}(\mathrm{A})$ 的正则化项为：

$$
R_{\mathrm{sum}}(\mathbf{K}(\mathrm{A})) = \sum_{i=1}^{d-1} \|[\mathrm{sumvec}(\mathbf{K}(\mathrm{A}))]_i\|_q^q, \tag{18}
$$

其中超参数为 $q \in \{1,2\}$。$\mathbf{K}(\mathrm{B})$ 的正则化项形式相同，故此处省略。

$R_{\mathrm{sum}}(\mathbf{K}(\mathrm{A}))$ 可以通过快速傅里叶变换高效计算，方法与计算 $R_{\mathrm{sum}}(\mathbf{C}(\mathrm{A},\mathrm{B}))$ 类似。为简洁起见，假设集合 A 已中心化；也就是说，A 中所有特征的均值均为 0。在这种情况下，其协方差矩阵为 $\mathbf{K}(\mathrm{A}) = (1/(n-1)) \sum_{k=1}^{n} \mathbf{a}^{(k)} \mathbf{a}^{(k)\mathrm{T}}$。

No注意到 $\mathcal{F}(\mathrm{inv}(\mathbf{a}^{(k)}) = \overline{\mathcal{F}(\mathbf{a}^{(k)})}$，以及卷积定理 $\mathcal{F}(\mathbf{x} * \mathbf{y}) = \mathcal{F}(\mathbf{x}) \circ \mathcal{F}(\mathbf{y})$，我们有

$$
\begin{aligned}
\mathrm{sumvec}(\mathbf{K}(\mathrm{A})) &= \frac{1}{n-1} \sum_{k=1}^{n} \mathrm{inv}(\mathbf{a}^{(k)}) * \mathbf{a}^{(k)} \\
&= \frac{1}{n-1} \sum_{k=1}^{n} \overbrace{\mathcal{F}^{-1}\left(\overline{\mathcal{F}(\mathbf{a}^{(k)})} \circ \mathcal{F}(\mathbf{a}^{(k)})\right)}^{\mathrm{inv}(\mathbf{a}^{(k)}) * \mathbf{a}^{(k)}} \\
&= \frac{1}{n-1} \mathcal{F}^{-1}\left(\sum_{k=1}^{n} \overline{\mathcal{F}(\mathbf{a}^{(k)})} \circ \mathcal{F}(\mathbf{a}^{(k)})\right),
\end{aligned} \tag{19}
$$

分组版本也很直接。将 $\mathbf{K}(\mathrm{A})$划分为大小为 $b \times b$ 的分块子矩阵，即 $\mathbf{K}(A) = [\mathbf{K}_{ij}](i, j = 1, \ldots, \lceil d/b \rceil)$，其中 $\mathbf{K}_{ij} \in \mathbb{R}^{b \times b}$，并对其应用 $R_{\mathrm{sum}}^{(b)}$式（13）中定义的内容，我们得到

$$
R_{\mathrm{sum}}^{(b)}(\mathbf{K}(\mathrm{A})) = \sum_{i=1}^{\lceil d/b \rceil} \sum_{\ell=1}^{b-1} \|[\mathrm{sumvec}(\mathbf{K}_{ii})]_\ell\|_q^q + \sum_{\substack{i,j=1 \\ i \neq j}}^{\lceil d/b \rceil} \sum_{\ell=0}^{b-1} \left\|[\mathrm{sumvec}(\mathbf{K}_{ij})]_\ell\right\|_q^q. \tag{20}
$$

其中，块大小 b 是控制汇总计算粒度的超参数。当 $q = 2$ 且 $b = d$时，即块大小为 $(b/d) \times (b/d) = 1 \times 1$，正则化项 $R_{\mathrm{sum}}^{(b)}(\mathbf{K}(\mathrm{A}))$ 会退化为 VICReg 的 $R_{\mathrm{off}}(\mathbf{K}(\mathrm{A}))$。

表 7. 损失计算复杂度。空间复杂度包括存储输入嵌入所需的 $O(nd)$ 内存。分组 $= b$ 表示组的大小（即块大小）。

正则化项	分组	Time	空间
Barlow Twins	—	$O(nd^2)$	$O(nd + d^2)$
VICReg	—	$O(nd^2)$	$O(nd + d^2)$
所提出的 (R_{sum})	no	$O(nd$ 对数 $d)$	$O(nd)$
所提出的（$R_{\text{sum}}^{(b)}$）	b	$O(\ (nd^2/b) \log\ b)$	$O(nd)$

表 8. 所提方法（含分组和不含分组）、Barlow Twins 和 VICReg 中的损失函数与正则化项。分组 $= b$ 表示 b 为组的大小（即块大小）。

(a) 采用 Barlow Twins 风格损失函数的互相关正则化 $L = \sum_i (1 - [\mathbf{C}(\mathrm{A}, \mathrm{B})]_{ii})^2 + \lambda R(\mathbf{C}(\mathrm{A}, \mathrm{B}))$

方法	分组	正则化函数 R
Barlow Twins	—	R_{off}
所提出的	no	R_{sum}
所提出的	b	$R_{\text{sum}}^{(b)}$

(b) 使用 VICReg 风格损失函数进行协方差正则化 $L = \frac{\alpha}{n} \sum_i \|\mathbf{a}_{_}^{(i)} - \mathbf{b}^{(i)}\|_2^2 + \frac{\mu}{d}(R_{\text{var}}(\mathbf{K}_{_}(\mathrm{A}))) + R_{\text{var}}(\mathbf{K}(\mathrm{B}))) + \frac{\nu}{d}(R(\mathbf{K}_{_}(\mathrm{A}))) + R(\mathbf{K}(\mathrm{B})))$

方法	分组	正则化函数 R
VICReg	—	R_{off}
所提出的	no	R_{sum}
所提出的	b	$R_{\text{sum}}^{(b)}$

C. 计算复杂度总结

表 7 总结了本文讨论的正则化项的计算复杂度。如表所示，就时间和空间复杂度而言，所提出的正则化项比 Barlow Twins 和 VICReg 更快、成本更低。

D. 详细实验设置

第 5 节中的所有实验均在配备 CUDA v11.6.2 和 cuDNN v8 的商用 Linux 服务器上进行。我们使用 PyTorch v1.12 实现了模型。0 [21] 以及 solo-learn v1.0。5 [28], 这是一个基于 PyTorch 和 PyTorch Lightning 构建的视觉表征学习自监督方法库[6] v1.6.4。我们还使用了 NVIDIA DALI，这是一个用于数据加载和预处理的库，可加速深度学习应用[7]。对于目标检测，我们使用了 detectron2 [31]。为管理实验，我们使用了 Weights & Biases，这是一个用于跟踪和可视化实验的机器学习平台 [2]。由于 PyTorch v1.12.0 仅提供对半精度快速傅里叶变换的实验性支持[8]，我们使用 32 位精度训练了每个模型，包括 Barlow Twins 和 VICReg。

D.1. 对比方法

在每组比较中，所提出的方法与两种基线方法 Barlow Twins 和 VICReg 使用相同的网络架构，唯一的区别在于训练损失。基线方法的损失函数由公式 (1) 和 (3) 给出，为方便起见，下面将其分别重写为公式 (21) 和 (22)。设 $\mathrm{A} = \{\mathbf{a}^{(i)}\}_{i=1}^{m}$、$\mathrm{B} = \{\mathbf{b}^{(i)}\}_{i=1}^{m}$ 为两个视图的嵌入，其中 $i = 1$、. . .、m 表示原始样本索引，$\mathbf{K}(\mathrm{A})$、$\mathbf{K}(\mathrm{B}) \in \mathbb{R}^{d \times d}$ 分别表示它们的协方差矩阵，

[6] https://www.pytorchlightning.ai/ [7] https://github.com/NVIDIA/DALI [8] https://pytorch.org/blog/pytorch-1.12-released/#beta-complex32-and-complex-convolutions-in-pytorch

以及 $\mathbf{C}(\mathrm{A},\mathrm{B}) \in \mathbb{R}^{d\times d}$ 表示 A 与 B之间的互相关矩阵。

$$L_{\mathrm{BT}} = \sum_{i=0}^{d-1} (1 - [\mathbf{C}(\mathrm{A},\mathrm{B})]_{ii})^2 + \lambda R_{\mathrm{off}}(\mathbf{C}(\mathrm{A},\mathrm{B})), \tag{21}$$

$$L_{\mathrm{VIC}} = \frac{\alpha}{n}\sum_{i=1}^{m} \|\mathbf{a}^{(i)} - \mathbf{b}^{(i)}\|_2^2 + \frac{\mu}{d}\left(R_{\mathrm{var}}(\mathbf{K}(\mathrm{A})) + R_{\mathrm{var}}(\mathbf{K}(\mathrm{B}))\right) + \frac{\nu}{d}\left(R_{\mathrm{off}}(\mathbf{K}(\mathrm{A})) + R_{\mathrm{off}}(\mathbf{K}(\mathrm{B}))\right), \tag{22}$$

其中，超参数 α， μ， ν， $\lambda \geq 0$ 决定各项的权重，而正则化函数定义如下

$$R_{\mathrm{off}}(\mathbf{M}) = \sum_{i=0}^{d-1}\sum_{\substack{j=0\\ j\neq i}}^{d-1} [\mathbf{M}]_{ij}^2,$$

$$R_{\mathrm{var}}(\mathbf{M}) = \sum_{i=0}^{d-1} \max(0, \gamma - \sqrt{[\mathbf{M}]_{ii}}).$$

对于所提方法，我们将公式 (21) 和 (22) 中所有的 R_{off} 替换为 R_{sum} （公式 (6）或 $R_{\mathrm{sum}}^{(b)}$ （公式 (13），具体取决于是否使用分组。以下重复列出这些函数。

$$R_{\mathrm{sum}}(\mathbf{M}) = \sum_{i=1}^{d-1} \|[\mathrm{sumvec}(\mathbf{M})]_i\|_q^q, \tag{23}$$

$$R_{\mathrm{sum}}^{(b)}(\mathbf{M}) = \sum_{i=1}^{\lceil d/b\rceil}\sum_{\ell=1}^{b-1} \|[\mathrm{sumvec}(\mathbf{M}_{ii})]_\ell\|_q^q + \sum_{\substack{i,j=1\\ i\neq j}}^{\lceil d/b\rceil}\sum_{\ell=0}^{b-1} \|[\mathrm{sumvec}(\mathbf{M}_{ij})]_\ell\|_q^q, \tag{24}$$

w其中，$\mathbf{M} = [\mathbf{M}_{ij}]$是一个块矩阵，其块元素为 $\mathbf{M}_{ij} \in \mathbb{R}^{b\times b}, i, \quad j = 1, \ldots, \lceil d/b\rceil$。

表 8 总和 概述了 Barlow Twins、VICReg 和所提出模型的正则化项及损失函数 s.

D.2. 数据增强

遵循 Barlow Twins 论文 [33], ，我们在 Barlow Twins 风格的目标函数中使用了非对称参数。对于 ImageNet-100 实验中的 VICReg 风格目标函数，我们采用了 VICReg 论文中报告的对称化增强流程 [1, 附录 C。1]。遵循 VICReg GitHub 仓库中的一条评论[9], ，我们在 ImageNet 实验中使用了非对称增强参数（与 Barlow Twins 中的参数相同）。

D.3. 超参数

ImageNet-100。对于 ImageNet-100 实验，我们遵循了 [28]中描述的优化流程。训练模型时，使用随机梯度下降（SGD）和 LARS 优化器 [32] 训练 400 个训练轮次。学习率调度器采用线性预热和余弦退火衰减。批量大小设为 256（每个图形处理器的批量大小为 32）。通过网格搜索确定损失缩放值以及所提出的正则化项的权重系数（ν 见公式（3）以及 λ 见公式（1））。除这些参数外，我们还对正则化项（公式（6）和（13））中的块大小和 q 进行了调优。

在线性评估中，使用随机梯度下降对线性分类器进行优化，共训练 100 轮次。在 ImageNet-100 的训练中，我们使用了 solo-learn 库提供的超参数。

表 9 总结了 ImageNet-100 实验的超参数 。

ImageNet。对于 ImageNet 实验，我们遵循了 [1,33] 中描述的优化流程。我们使用 SGD 和 LARS 优化器训练 1000 个训练轮次，并采用线性预热和余弦退火衰减。我们将批量大小设为 1024（每个图形处理器的批量大小为 128），并参考 Barlow Twins 的 GitHub 仓库，将学习率设为 0.25[10]。我们通过网格搜索寻找 λ和 q 所提正则化项的取值。

[9] https://github.com/facebookresearch/vicreg/issues/

[10] https://github.com/facebookresearch/barlowtwins/issues/7

表 9. 用于训练模型的 ImageNet-100 实验超参数。Barlow Twins 和 VICReg 的超参数设为 [28] 报告的数值。对于所提方法，我们通过网格搜索确定了相应数值。

(a) 使用 Barlow Twins 风格损失函数进行交叉相关正则化

方法	分组	损失缩放	q	λ
Barlow Twins	—	0.1	—	0.0051
所提出的（Barlow Twins 风格）	no	0.125	2	2^{-10}
所提出的（Barlow Twins 风格）	$b = 128$	0.125	2	2^{-10}

（b）协方差正则化

方法	分组	损失缩放	q	ν
VICReg	—	—	—	1.0
所提出的（VICReg 风格）	no	0.25	1	1.0
所提出的（VICReg 风格）	$b = 128$	0.25	1	2.0

（c）其他所有超参数均设为 [28] 报告的值。

学习率	权重衰减	批量大小	预热轮数	α	μ
0.3	10^{-4}	256	10	25.0	25.0

（d）ImageNet-100 上线性评估的超参数。

预训练模型	学习率	学习率衰减步数	权重衰减	批量大小
Barlow Twins / 所提出的（Barlow Twins 风格）	0.1	[60, 80]	0	256
VICReg / 所提出的（VICReg 风格）	0.3	[60, 80]	0	512

在 ImageNet 上进行线性评估时，线性分类头使用随机梯度下降和余弦学习率衰减训练了 100 个训练轮次。我们为所提方法调优了学习率和批量大小。对于 Barlow Twins 和 VICReg，学习率和批量大小设为原始论文 [1, 33] 中报告的值。

在目标检测任务中，我们使用 C-4骨干网络的 Faster R-CNN 训练了 24K 次迭代。骨干网络以预训练的 ResNet-50 主干网络初始化。按照 [1, 16, 33],，我们将批量大小设为 16（每个图形处理器的批量大小为 2），并采用阶梯式学习率衰减（在 18K 和 22K 次迭代时除以 10），同时进行线性预热（斜率为 0.333，持续 1K 次迭代）。我们为所提方法调优了学习率和区域提议网络损失权重。

表10 总结了 ImageNet 实验的超参数。

D.4. 训练时间和内存消耗评估

为了讨论经验复杂度，我们测量了十个训练轮次的经过时间和 GPU 峰值内存分配。我们进行了三次试验，并报告平均时间和内存分配。为避免 GPU 之间的通信开销，除非另有说明，我们评估的是单 GPU 训练（非分布式数据并行训练）的结果。按照预训练设置，ImageNet-100 的批量大小设为 32，ImageNet 的批量大小设为 128（每个 GPU 的批量大小分别为 32 和 128）。工作进程数（solo-learn 库中用于实现的参数“num_workers”）在 ImageNet-100 上设为 32，在 ImageNet 上设为 4。

我们使用 PyTorch Lightning 中的 Simple Profiler[11] 测量训练时间。从性能分析器的输出中，提取“run_training_epoch”行中“Total time (s)”的值，并将其作为训练时间绘制在图 2、3和4 至 7中。我们使用 PyTorch[12] 中的一个函数监控张量占用的内存。“Allocated memory”行中的“Peak Usage”值被用作 GPU 峰值内存分配。

关于表4中的总训练时间，我们报告了由 WandB 统计的运行时间。[13] 如第5.2节脚注所述，我们的实验是在商业云平台上进行的，该平台会在三天后终止会话。为了

[11] https://pytorch-lightning.readthedocs.io/en/1.6.4/advanced/profiler.html#simple-profiler [12] https://pytorch.org/docs/stable/generated/torch.cuda.memory_summary.html [13] 我们使用了 WandB 中“运行时间”的数值。

表10。用于训练模型的 ImageNet 实验超参数。

（a）交叉相关正则化

方法	分组	损失缩放	q	λ
Barlow Twins	—	0.024	—	0.0051
所提出的（Barlow Twins 风格）	no	0.024	2	2^{-11}
所提出的（Barlow Twins 风格）	$b = 128$	0.024	2	2^{-11}

（b）协方差正则化

方法	分组	损失缩放	q	α	μ	ν
VICReg	—	—	—	25.0	25.0	1.0
所提出的（VICReg 风格）	no	—	1	2.5	2.5	0.1

(c) 优化超参数。

学习率	权重衰减	批量大小	预热轮数
0.25	10^{-6}	1024	10

(d) ImageNet 上线性评估的超参数。

预训练模型	学习率	学习率衰减	权重衰减	批量大小
Barlow Twins	0.3	余弦衰减	10^{-6}	256
VICReg	0.02	余弦衰减	10^{-6}	256
所提出的（Barlow Twins 风格）	0.125	余弦衰减	10^{-6}	2048
所提出的（VICReg 风格）	0.125	余弦衰减	10^{-6}	256

（e）VOC07+12 上目标检测的超参数。

预训练模型	学习率	区域提议网络损失权重
所提出的（Barlow Twins 风格）	0.125	0.03125
所提出的（VICReg 风格）	0.125	0.125

在 ImageNet 上使用 8 个图形处理器训练 Barlow Twins 和 VICReg 共 1000 轮，我们需要三个会话，而所提出的模型只需要两个会话（见表4）。表4 中报告的时间是这些会话的运行时间，其中包括每个会话开始时的初始化时间。每个会话的初始化仅需 3–5 秒，不影响表中观察到的趋势。请注意，除初始化外，会话开始时的数据复制还需要约 15 分钟，但这部分时间已从表4的计时中排除。

D.5. 计算资源

我们使用云计算平台进行实验。在主要实验中，我们在该平台上使用八个 Nvidia A100-SXM4 图形处理器训练模型。除特别注明外，我们使用单个 Nvidia A100 图形处理器评估经验复杂度。

D.6. 资产的许可证

PyTorch 使用 BSD式许可证[14]。Solo-learn 使用 MIT许可证[15]。PyTorch Lightning 根据 Apache 许可证 2.0[16] 授权。ImageNet[17] 数据集公开可用，并常被用作基准数据集。ImageNet-100 的类别列表也已公开[18]。

[14]https://github.com/pytorch/pytorch/blob/master/LICENSE [15] https://github.com/vturrisi/solo-learn/blob/main/LICENSE [16] https://github.com/Lightning-AI/lightning/blob/master/LICENSE [17] https://www.image-net.org/ [18] https://github.com/HobbitLong/CMC/blob/master/imagenet100.txt

表 11. 在 ImageNet-100 上， $q \in \{1,2\}$ 的准确率。

模型	分组	q	Top-1	Top-5
所提出的（Barlow Twins 风格）	no	1	75.94	94.28
		2	**79.94**	**94.76**
	$b = 128$	1	76.44	94.46
		2	**81.02**	**95.24**
所提出的（VICReg 风格）	no	1	**79.20**	**94.96**
		2	57.98	84.56
	$b = 128$	1	**80.04**	**94.98**
		2	71.78	92.54

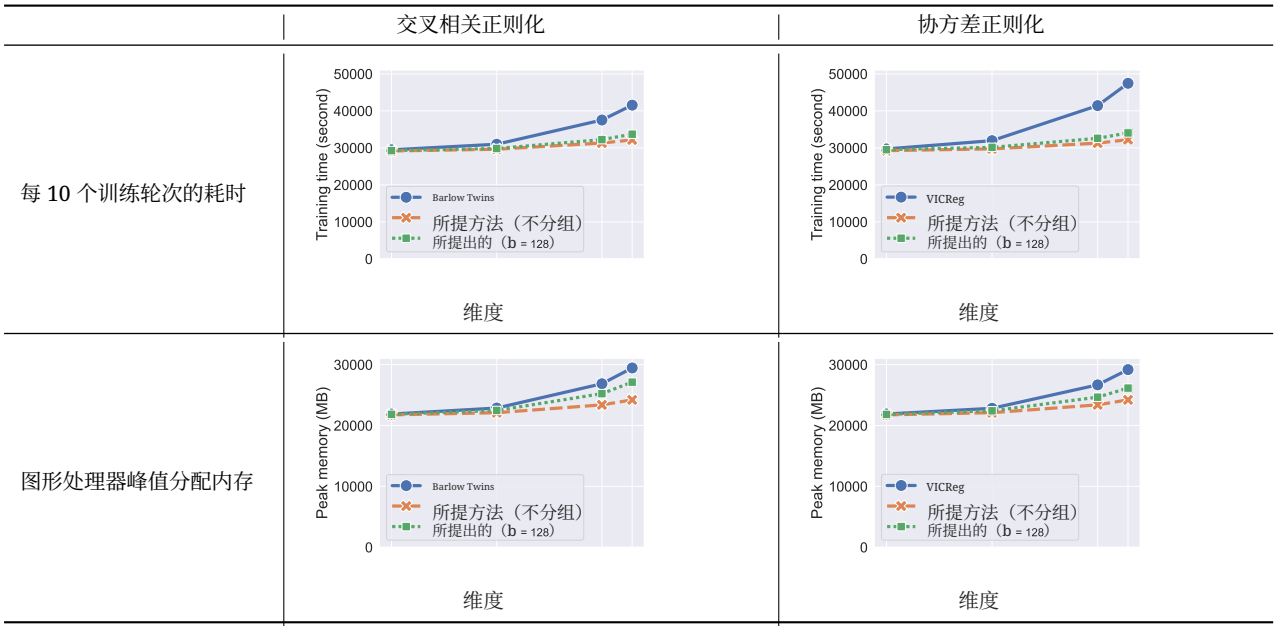


图 4. 使用单个图形处理器，在 ImageNet 上采用 ResNet-50 进行训练的时间和内存使用量。

E. 补充实验

E.1. 超参数 q 的影响

我们研究了超参数 q 对正则化项（公式（6）和（13））的影响。表 11 展示了$q \in \{1,2\}$ 在 ImageNet-100（$d = 2048$）上的结果。结果表明， $q = 1$ 在 VICReg 风格协方差正则项中优于 $q = 2$ 。相反， $q = 2$ 在 Barlow Twins 风格的互相关正则化器中表现良好。

E.2. 使用 ResNet-50 主干网络的训练时间

图4展示了在 ImageNet 上使用 ResNet-50 进行单图形处理器训练的成本。由于图形处理器内存有限，我们无法在 $d = 16384$ 时运行 Barlow Twins 和 VICReg，也无法运行块大小为 $b = 128$ 的所提出模型的分组版本。我们没有展示 $d = 16384$ 的结果，而是展示了 $d = 10000$ 时的结果。[19] 与 ImageNet-100 的结果一样，我们可以观察到，所提出的正则化项比现有正则化项更高效。在 $d = 8192$ 时，所提出的模型（不分组）比 VICReg 快 1.3 倍（= 41414.3/31280.0），比 Barlow Twins 快 1.2 倍（= 37522.0/31306.0）；而在 $d = 10000$ 时，它比 VICReg 快 1.5 倍（= 47448.0/32208.3），比 Barlow Twins 快 1.3 倍（= 41546.3/32143.7）。

[19]在第E.3 节（见图7）中，我们展示了使用 $d = 16384$ 多节点 DDP 计算的结果。

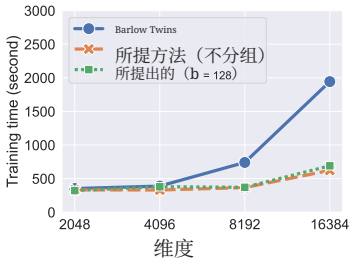
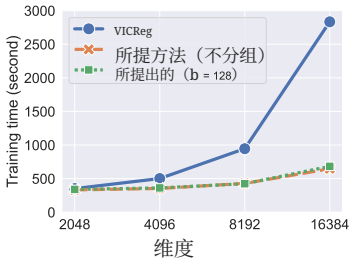
	交叉相关正则化族	协方差正则化族
每 10 个训练轮次的耗时		

图 5. 使用 ResNet-18 在 ImageNet-100 上进行分布式数据并行训练的经过时间。

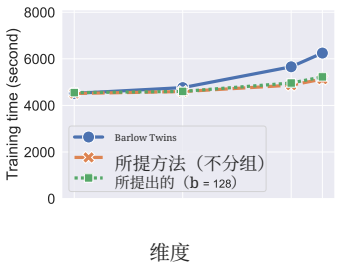
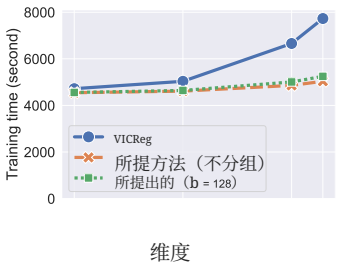
	交叉相关正则化族	协方差正则化族
每 10 个训练轮次的耗时		

图 6. 使用 ResNet-50 在 ImageNet 上进行分布式数据并行训练的耗时。

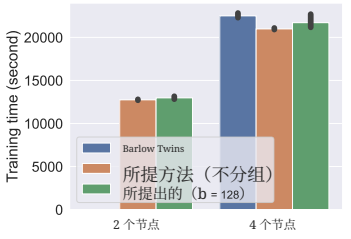
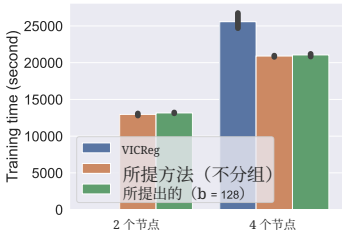
	交叉相关正则化族	协方差正则化族
每 10 个训练轮次的耗时		

图 7. 使用 ResNet-50 在 ImageNet 上进行多节点 DDP 训练的经过时间（d = 16384）。

E.3. 使用分布式数据并行的训练时间

除表4外，第5节和附录E.2中报告的训练时间数据均是在单个图形处理器上测得的。此处我们评估可使用多个图形处理器时分布式数据并行（DDP）训练的耗时。图5展示了在八个 A100图形处理器上进行十个训练轮次的分布式数据并行训练经过时间。采用分布式数据并行时，图形处理器之间的通信成本成为决定总计算时间的一个额外因素，因此，我们的方法在缩短损失计算时间方面的相对优势预计会有所减弱。尽管确实如此，我们的方法仍然有效：当d = 8192 时，VICReg 的计算时间缩短了 2.2 倍以上（= 945.5/428.7），Barlow Twins 的计算时间缩短了 2.0 倍（= 740.6/366.4）；而当 d = 16384 时，则分别缩短了 4.4 倍（= 2833.3/647.1）和 3.1 倍（= 1943.7/622.7）。

图6展示了使用 ResNet-50 在 ImageNet 上的经过时间。与使用 ResNet-18 在 ImageNet-100 上的情况（图5）一样，我们的方法提升了速度，但加速比更小：当 d = 8192 时，VICReg 的加速比为 1.4（= 6658.1/4858.9），Barlow Twins 的加速比为 1.2（= 5657.6/4868.7）。当 d = 10000 时，VICReg 的加速比为 1.5（= 7729.9/5035.1），Barlow Twins 的加速比为 1.2（= 6247.7/5129.6）。

表 12。在使用 ResNet-18 的 ImageNet100 上，损失和反向计算的改进。

模型	#GPU	d	前向传播（损失）	反向（模型 + 损失）
Barlow Twins 风格	1	8192	7.5 (= 285.5/38.0)	2.5 (= 1101.6/448.6)
		16384	**23.1**(= 1038.7/44.9)	**6.3**(= 2959.1/468.0)
	8	8192	7.4（= 52。5/7。1）	1.9 (= 124.3/67.1)
		16384	**24.8**(= 183.6/7.4)	**3.9**(= 274.0/70.3)
VICReg 风格	1	8192	6.0（= 323。0/53。5）	5.4 (= 2548。8/473。2)
		16384	**19.5**(= 1166.5/59.8)	**18.3**(= 8820.3/482.0)
	8	8192	7.0 (= 62.6/8.9)	4.1 (= 300.5/74.2)
		16384	**24.2**(= 215.7/8.9)	**13.2**(= 996.3/75.5)

表 13. 使用 ResNet-50 在 ImageNet 上进行损失和反向计算的改进。

模型	#GPU	d	前向传播（损失）	反向（模型 + 损失）
Barlow Twins 风格	1	8192	9.5 (= 751.1/79.2)	2.9 (= 8594.0/2977.5)
		10000	**13.2**(= 1097.5/83.2)	**3.6**(= 11583.0/3181.2)
	8	8192	10.6 (= 125.8/11.9)	1.5 (= 2129.1/1463.5)
		10000	**15.0**(= 184.4/12.3)	**1.6**(= 2552.2/1593.5)
VICReg 风格	1	8192	8.4 (= 898.8/107.0)	4.1（= 12346.0/3016.6）
		10000	**11.9**(= 1311.1/110.3)	**5.4**(= 17249.3/3221.7)
	8	8192	18.1 (= 280.7/15.5)	2.0 (= 2937.3/1467.1)
		10000	**25.2**(= 415.7/16.5)	**2.4**(= 3752。4/1557。9)

在 ImageNet 实验（图 4 和6）中，当 $d = 16384$ 时，所有模型都触发了 GPU 显存溢出错误。因此，我们对 $d = 16384$采用多节点 DDP 训练。我们评估了两种情况：使用 2 个节点和 4 个节点进行训练。在这两种情况下，我们都将有效批量大小设为 1024。图 7 展示了结果。使用 2 个节点时，Barlow Twins 和 VICReg 仍然会耗尽内存，但由于内存使用效率高，所提出的模型（无论是否采用分组）都能在这种情况下正常运行。使用 4 个节点时，所有模型都成功完成了训练，且所提出的模型比 Barlow Twins 和 VICReg 略快。不过，在这种设置下，如果我们的模型用 2 个节点就能完成训练，那么使用 4 个节点训练它们毫无意义。若将使用 2 个节点训练的模型与使用 4 个节点训练的 Barlow Twins 和 VICReg（它们无法在 2 个节点上完成训练）进行速度比较，我们模型的优势会更加明显。

E.4. 前向与反向传播的计算时间

我们分析了前向传播（模型网络中，不包括损失节点）、前向损失计算和反向传播所花费的训练时间。这三类分别记为“前向（模型）”、“前向（损失）”和“反向（模型 + 损失）”。这些测量值摘自 PyTorch Lightning 的 Simple Profiler 输出中相应行的“总时间（秒）”列。请注意，性能分析器默认只报告前向计算的总时间。为了将总时间拆分为“前向（模型）”和“前向（损失）”，我们在代码中指定了自定义测量点，以分别测量模型（骨干网络和投影网络）的前向计算和损失计算。[20] 由于自定义测量点无法用于反向传播，我们将“Strategy.backward”行的值作为反向传播总时间“反向（模型 +损失）”绘出。

图 8 展示了结果。采用我们的方法后，损失计算和反向传播的耗时均有所改善；改进详情见表 12 和13 。反向传播方面的改进源于损失节点耗时的减少，因为 Barlow Twins 和所提出的方法使用相同的模型网络，区别仅在于所用的损失。反向传播时间缩短并不意外，因为前向和反向传播都会遍历相同的计算图，因此二者的渐近计算复杂度大致相同。

[20] https://pytorch-lightning.readthedocs.io/en/1.6.4/advanced/profiler.html#profile-logic-of-your-interest

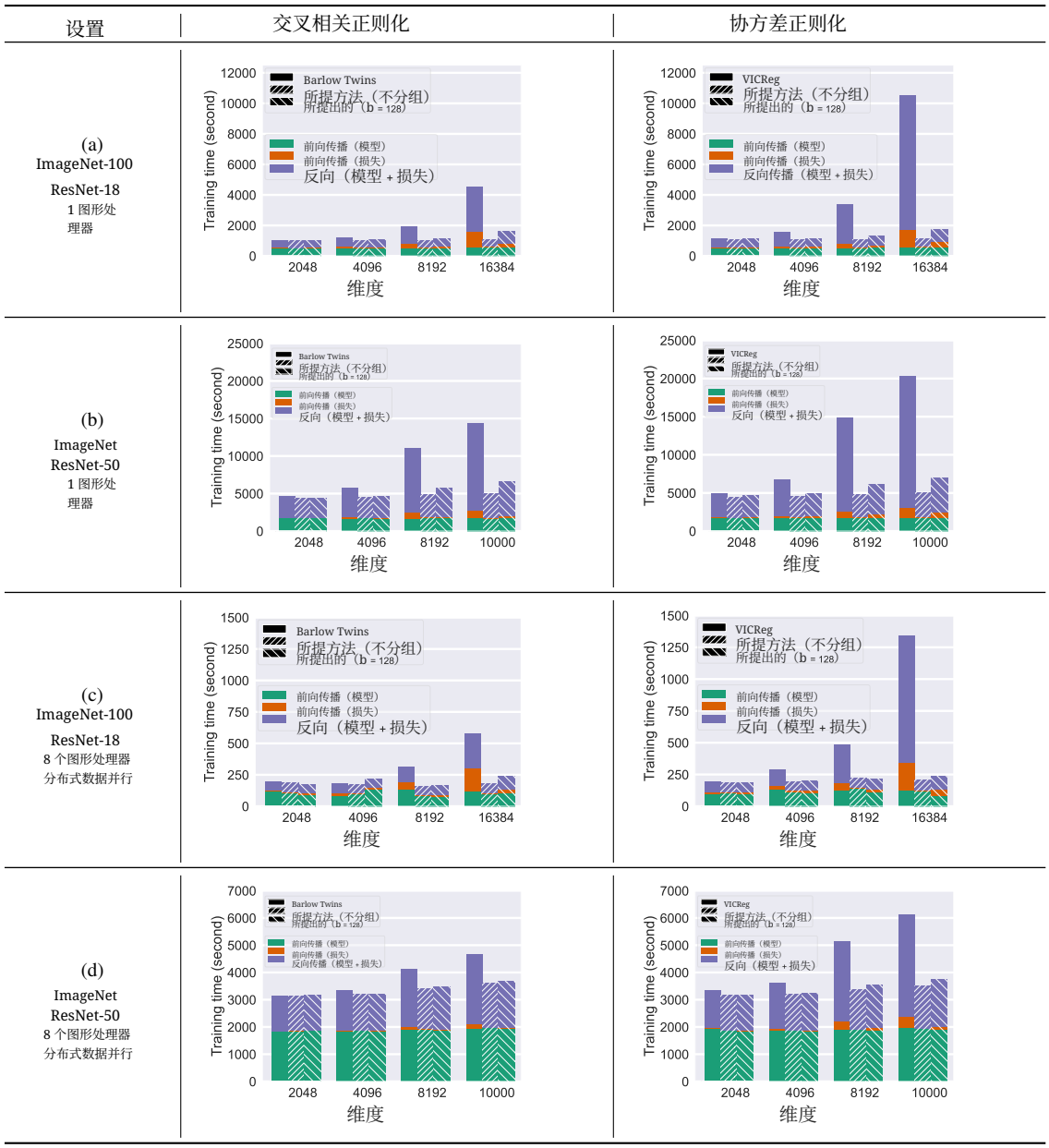


图8. 前向模型（绿色）和损失（橙色）的计算，以及反向传播（模型和损失的总计；紫色）所用的 10 个训练轮次的耗时。

F. 代码

代码清单 1 和 2 给出了基于 Python 的协方差和互相关正则化项实现（不含特征分组）。如第 4 节所述，可使用快速傅里叶变换高效计算汇总向量（见代码清单 3）。

在所提出的正则化项的计算中，我们不执行集体操作，例如全归约和收集函数。

```
# Z1,  Z2: 投影图像嵌入（n x d）
# q: L_q^q 范数的超参数

定义 xsum_正则化项(Z1,        Z2,  G, q):
    # 预处理：中心化和归一化
    Z1 = 批次_归一化(Z1)
    Z2 = 批次_归一化(Z2)

    # 特征置换
    索引 = torch.randperm(Z1.shape[1])
    Z1 = Z1[:, idx]
    Z2 = Z2[:, idx]

    # 汇总向量
    汇总向量 = cal_sumvec(Z1   , Z2,  0) / n

    # 非对角元素的损失
    if q == 1:
        损失 = torch.sum(sumvec[1:].abs())

    否则如果 q == 2:
        损失 = torch.sum(sumvec[1:].pow(2))


    返回损失
```

清单1. 计算 Barlow Twins 风格交叉相关正则化项

```
# Z: 投影图像嵌入（[n: 批量大小] x  [d: 嵌入维度]）
# q: L_q^q 范数的超参数

定义协方和_正则化项(Z,          块_大小, q):
    # 预处理：中心化
    Z =  Z - Z.mean(dim=0)

    # 特征置换
    idx = torch.randperm(Z.shape[1])
    Z = Z[:, idx]

    # 汇总向量
    sumvec = cal_sumvec(Z   , Z,  0) / (n - 1)

    # 非对角元素的损失
    if q == 1:
        损失 = torch.sum(sumvec[1:].abs())

    elif q == 2:
        损失 = torch.sum(sumvec[1:].pow(2))


    返回损失
```

清单2：计算 VICReg 风格协方差正则项

```
定义 计算_sumvec(z1   , z2, 维度):
    fz1 = 快速傅里叶变换实数形式(z1)
    fz2 = 快速傅里叶变换实数形式(z2)
    fz1_共轭 = fz1.conj()
    fz_乘积 = fz1_共轭 * fz2
    fc  = fz_乘积。求和(维度=维度)
    sumvec= 快速傅里叶逆变换实数形式(fc)
    返回 sumvec
```

清单3。摘要向量计算