Learning Decorrelated Representations Efficiently Using Fast Fourier Transform: Supplementary Material

Yutaro Shigeto Masashi Shimbo Yuya Yoshikawa Akikazu Takeuchi
{shigeto,shimbo,yoshikawa,takeuchi}@stair.center
STAIR Lab, Chiba Institute of Technology, Narashino, Chiba, Japan

March 25, 2023

This document contains supplementary material for our paper titled "Learning decorrelated representations efficiently using Fast Fourier Transform" appearing in CVPR 2023. Numeric equation numbers, such as (1), (2), refer to equations in the main paper. New equations in this document are prefixed with a section letter, as in (A.1), (B.2). The same rules apply to the numbering of figures and tables.

Contents

A Derivation of Eq. (8)

Let d-dimensional vectors $\mathbf{x} = [x_0 \cdots x_{d-1}]^\mathrm{T}$, $\mathbf{y} = [y_0 \cdots y_{d-1}]^\mathrm{T}$. We first show $\mathrm{inv}(\mathbf{x}) * \mathbf{y} = \sum_{j=0}^{d-1} x_j y_{(i+j) \bmod d}$.

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

Setting $\mathbf{x} = \mathbf{a}^{(k)}$ and $\mathbf{y} = \mathbf{b}^{(k)}$, we obtain Eq. (8) in Sec. 4.2. In the literature (e.g., [Pla03, Smi08]), $\mathrm{inv}(\mathbf{x}) * \mathbf{y}$ is called the *circular correlation* of $\mathbf{x}$ and $\mathbf{y}$, and the above equation is usually presented as its definition.

B Regularizers Based on Sums of Feature Covariances

As mentioned in Sec. 4.5, if we substitute R_sum for R_off in the loss function of VICReg given in Eq. (3), we obtain regularization based on the covariance matrices $\mathbf{K}(\mathrm{A}), \mathbf{K}(\mathrm{B})$ of individual views. The resulting regularizer for $\mathbf{K}(\mathrm{A})$ is:

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

where hyperparameter $q \in \{1, 2\}$. The regularizer for $\mathbf{K}(\mathrm{B})$ has the same form and is omitted.

$R_\mathrm{sum}(\mathbf{K}(\mathrm{A}))$ can be efficiently computed by FFT, in a similar manner to $R_\mathrm{sum}(\mathbf{C}(\mathrm{A}, \mathrm{B}))$. For brevity, assume that set A is centered; i.e., all features have mean 0 in A. In this case, its covariance matrix is $\mathbf{K}(\mathrm{A}) = (1/(n-1)) \sum_{k=1}^n \mathbf{a}^{(k)} \mathbf{a}^{(k)\mathrm{T}}$.

Noting that $\mathcal{F}(\mathrm{inv}(\mathbf{a}^{(k)}) = \overline{\mathcal{F}(\mathbf{a}^{(k)})}$ and the convolution theorem $\mathcal{F}(\mathbf{x} * \mathbf{y}) = \mathcal{F}(\mathbf{x}) \circ \mathcal{F}(\mathbf{y})$, we have

$$
\begin{aligned}
\mathrm{sumvec}(\mathbf{K}(\mathrm{A})) &= \frac{1}{n-1} \sum_{k=1}^n \mathrm{inv}(\mathbf{a}^{(k)}) * \mathbf{a}^{(k)} \\
&= \frac{1}{n-1} \sum_{k=1}^n \overbrace{\mathcal{F}^{-1}\left(\overline{\mathcal{F}(\mathbf{a}^{(k)})} \circ \mathcal{F}(\mathbf{a}^{(k)})\right)}^{\mathrm{inv}(\mathbf{a}^{(k)}) * \mathbf{a}^{(k)}} \\
&= \frac{1}{n-1} \mathcal{F}^{-1}\left(\sum_{k=1}^n \overline{\mathcal{F}(\mathbf{a}^{(k)})} \circ \mathcal{F}(\mathbf{a}^{(k)})\right),
\end{aligned} \tag{B.2}
$$

The grouping version is also straightforward. Partitioning $\mathbf{K}(\mathrm{A})$ into block submatrices of size $b \times b$, i.e., $\mathbf{K}(A) = [\mathbf{K}_{ij}]$ $(i, j = 1, \ldots, \lceil d/b \rceil)$, where $\mathbf{K}_{ij} \in \mathbb{R}^{b \times b}$, and applying $R_\mathrm{sum}^{(b)}$ defined in Eq. (13) to it, we obtain

$$
R_\mathrm{sum}^{(b)}(\mathbf{K}(\mathrm{A})) = \sum_{i=1}^{\lceil d/b \rceil} \sum_{\ell=1}^{b-1} \| [\mathrm{sumvec}(\mathbf{K}_{ii})]_\ell \|_q^q + \sum_{\substack{i,j=1 \\ i \neq j}}^{\lceil d/b \rceil} \sum_{\ell=0}^{b-1} \left\| [\mathrm{sumvec}(\mathbf{K}_{ij})]_\ell \right\|_q^q. \tag{B.3}
$$

Table C.1. Complexity of loss computation. The space complexity includes $O(nd)$ memory needed to store input embeddings. Grouping = b indicates b being the size of the group (i.e., block size).

Regularizer	Grouping	Time	Space
Barlow Twins	—	$O(nd^2)$	$O(nd + d^2)$
VICReg	—	$O(nd^2)$	$O(nd + d^2)$
Proposed (R_{sum})	no	$O(nd \log d)$	$O(nd)$
Proposed ($R^{(b)}_{\text{sum}}$)	b	$O((nd^2/b) \log b)$	$O(nd)$

Table D.1. Loss functions and regularizers in the proposed method (with and without grouping), Barlow Twins, and VICReg. Grouping = b indicates b being the size of the group (i.e., block size).

(a) Cross-correlation–regularization with Barlow Twins–style loss function $L = \sum_i (1 - [\mathbf{C}(\mathrm{A}, \mathrm{B})]_{ii})^2 + \lambda R(\mathbf{C}(\mathrm{A}, \mathrm{B}))$

Method	Grouping	Regularizer function R
Barlow Twins	—	R_{off}
Proposed	no	R_{sum}
Proposed	b	$R^{(b)}_{\text{sum}}$

(b) Covariance regularization with VICReg-style loss function
$L = \frac{\alpha}{n} \sum_i \|\mathbf{a}^{(i)} - \mathbf{b}^{(i)}\|_2^2 + \frac{\mu}{d} (R_{\text{var}}(\mathbf{K}(\mathrm{A})) + R_{\text{var}}(\mathbf{K}(\mathrm{B}))) + \frac{\nu}{d} (R(\mathbf{K}(\mathrm{A})) + R(\mathbf{K}(\mathrm{B})))$

Method	Grouping	Regularizer function R
VICReg	—	R_{off}
Proposed	no	R_{sum}
Proposed	b	$R^{(b)}_{\text{sum}}$

where block size b is the hyperparameter that controls the granularity of the summary computation. When $q = 2$ and $b = d$, i.e., the block size is $(b/d) \times (b/d) = 1 \times 1$, the regularizer $R^{(b)}_{\text{sum}}(\mathbf{K}(\mathrm{A}))$ reduces to $R_{\text{off}}(\mathbf{K}(\mathrm{A}))$ of VICReg.

C Summary of Computational Complexity

Tab. C.1 summarizes the computational complexity of the regularizers discussed in this paper. As the table shows, the proposed regularizers are faster and cheaper than the Barlow Twins and VICReg in terms of time and space complexity.

D Detailed Experimental Setups

All the experiments in Sec. 5 were conducted on commercial Linux servers with CUDA v11.6.2 and cuDNN v8. We implemented our model using PyTorch v1.12.0 [PGM+19] and solo-learn v1.0.5 [TFN+22], a library of self-supervised methods for visual representation learning built on top of PyTorch and PyTorch Lightning[1] v1.6.4. We also used NVIDIA DALI, a library for data loading and pre-processing to accelerate deep learning applications[2]. For object detection, detectron2 [WKM+19] was used. To manage the experiments, we used Weights & Biases, a machine learning platform for the tracking and visualization of experiments [Bie20]. As PyTorch v1.12.0 only provides experimental support for half precision FFT[3], we trained every model with 32-bit precision, including Barlow Twins and VICReg.

D.1 Compared Methods

In each comparison, the proposed method and the two baselines, Barlow Twins and VICReg, used an identical network architecture, with the exception of the training loss. The loss functions of the baselines are given by Eqs. (1) and (3),

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

which are repeated below as Eqs. (D.1) and (D.2) for convenience. Let $\mathrm{A} = \{\mathbf{a}^{(i)}\}_{i=1}^{m}$, $\mathrm{B} = \{\mathbf{b}^{(i)}\}_{i=1}^{m}$ be the embeddings of the two views, with $i = 1, \ldots, m$ indicating the original sample indices, $\mathbf{K}(\mathrm{A}), \mathbf{K}(\mathrm{B}) \in \mathbb{R}^{d \times d}$ be their respective covariance matrices, and $\mathbf{C}(\mathrm{A}, \mathrm{B}) \in \mathbb{R}^{d \times d}$ be the cross-correlation matrices between A and B.

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

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

where hyperparameters $\alpha, \mu, \nu, \lambda \geq 0$ determine the importance of individual terms, and the regularization functions are given by

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

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

For the proposed method, we replace all occurrences of R_{off} in Eqs. (D.1) and (D.2) with either R_{sum} (Eq. (6)) or $R_{\mathrm{sum}}^{(b)}$ (Eq. (13)) depending on whether grouping is used. These functions are repeated below.

$$R_{\mathrm{sum}}(\mathbf{M}) = \sum_{i=1}^{d-1} \|[\mathrm{sumvec}(\mathbf{M})]_i\|_q^q, \tag{D.3}$$

$$R_{\mathrm{sum}}^{(b)}(\mathbf{M}) = \sum_{i=1}^{\lceil d/b \rceil} \sum_{\ell=1}^{b-1} \|[\mathrm{sumvec}(\mathbf{M}_{ii})]_\ell\|_q^q + \sum_{\substack{i,j=1 \\ i \neq j}}^{\lceil d/b \rceil} \sum_{\ell=0}^{b-1} \|[\mathrm{sumvec}(\mathbf{M}_{ij})]_\ell\|_q^q, \tag{D.4}$$

where $\mathbf{M} = [\mathbf{M}_{ij}]$ is a block matrix with block elements $\mathbf{M}_{ij} \in \mathbb{R}^{b \times b}$, $i, j = 1, \ldots, \lceil d/b \rceil$.

Tab. D.1 summarizes the regularizers and loss functions for Barlow Twins, VICReg, and the proposed models.

D.2 Data Augmentation

Following the Barlow Twins paper [ZJM+21], we used non-symmetric parameters for Barlow Twins-style objectives. For VICReg-style objectives in ImageNet-100 experiments, we used the symmetrized augmentation pipeline reported in the VICReg paper [BPL22, Appendix C.1]. Following a comment in the VICReg GitHub repository[4], we used the non-symmetric augmentation parameters (the same parameters as in Barlow Twins) for ImageNet experiments.

D.3 Hyperparameters

ImageNet-100. For ImageNet-100 experiments, we followed the optimization procedure described in [TFN+22]. To train models, stochastic gradient descent (SGD) was used with the LARS optimizer [YGG17] for 400 epochs. We used linear warm-up with cosine annealing decay for the learning rate scheduler. The batch size was wet to 256 (per GPU batch size is 32). The loss scaling value and the importance coefficients for the proposed regularizers (ν in Eq. (3) and λ in Eq. (1)) were sought by grid search. In addition to these parameters, we further tuned the block size and q in our regularizers (Eqs. (6) and (13)).

In linear evaluation, linear classifiers was optimized with SGD for 100 epochs. In training for ImageNet-100, we used the hyperparameters provided by the solo-learn library.

Tab. D.2 summarizes the hyperparameters for ImageNet-100 experiments.

[4] https://github.com/facebookresearch/vicreg/issues/3

Table D.2. The hyperparameters for ImageNet-100 experiments that were used for training models. The hyperparameters for Barlow Twins and VICReg were set to the values reported by [TFN+22]. For the proposed methods, we found values by grid search.

(a) Cross-correlation regularization with Barlow Twins–style loss function

Method	Grouping	Loss scale	q	λ
Barlow Twins	—	0.1	—	0.0051
Proposed (Barlow Twins–style)	no	0.125	2	2^{-10}
Proposed (Barlow Twins–style)	$b = 128$	0.125	2	2^{-10}

(b) Covariance regularization

Method	Grouping	Loss scale	q	ν
VICReg	—	—	—	1.0
Proposed (VICReg-style)	no	0.25	1	1.0
Proposed (VICReg-style)	$b = 128$	0.25	1	2.0

(c) All other hyperparameters were set to the values reported by [TFN+22].

Learning rate	Weight decay	Batch size	Warmup epochs	α	μ
0.3	10^{-4}	256	10	25.0	25.0

(d) Hyperparameters for linear evaluation on ImageNet-100.

Pretrained model	Learning rate	Steps for learning rate decay	Weight decay	Batch size
Barlow Twins / Proposed (Barlow Twins–style)	0.1	[60, 80]	0	256
VICReg / Proposed (VICReg-style)	0.3	[60, 80]	0	512

ImageNet. For ImageNet experiments, we followed the optimization procedure described in [ZJM+21, BPL22]. We used SGD with the LARS optimizer for 1000 epochs and linear warmup with cosine annealing decay. We set the batch size to 1024 (per GPU batch size is 128) and used a learning rate of 0.25 by reference to the Barlow Twins GitHub repository[5]. We searched for λ and q for the proposed regularizers by grid search.

In the linear evaluation on ImageNet, the linear head was trained for 100 epochs with SGD and cosine learning rate decay. We tuned the learning rate and batch size for the proposed methods. For Barlow Twins and VICReg, the learning rate and batch size are set to the values reported in the original papers [ZJM+21, BPL22].

In object detection, we trained a Faster R-CNN with a C-4 backbone for 24K iterations. The backbone is initialized with the pretrained ResNet-50 backbone. Following [HFW+20, ZJM+21, BPL22], we set the batch size to 16 (per GPU batch size is 2) and used a step learning rate decay (divided by 10 at 18K and 22K iterations) with a linear warmup (slope of 0.333 for 1K iterations). We tuned the learning rate and the region proposal network loss weight for the proposed methods.

Tab. D.3 summarizes the hyperparameters for ImageNet experiments.

D.4 Evaluation of Training Time and Memory Consumption

To discuss empirical complexity, we measured the elapsed time and peak GPU memory allocation over ten epochs. We conducted three trials and reported the average time and memory allocation. To avoid communication overhead between GPUs, we evaluated the results of single GPU training (not distributed data parallel training) unless otherwise noted. The batch size was set to 32 for ImageNet-100 and 128 for ImageNet as in pretraining settings (per GPU batch size is 32 and 128). The number of workers (the argument "num_workers" in the solo-learn library used for implementation) is set to 32 for ImageNet-100 and 4 for ImageNet.

We used the Simple Profiler in PyTorch Lightning[6] to measure training time. From the profiler output, the value

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

[6] https://pytorch-lightning.readthedocs.io/en/1.6.4/advanced/profiler.html#simple-profiler

Table D.3. The hyperparameters for ImageNet experiments that were used for training models.

(a) Cross-correlation regularization

Method	Grouping	Loss scale	q	λ
Barlow Twins	—	0.024	—	0.0051
Proposed (Barlow Twins–style)	no	0.024	2	2^{-11}
Proposed (Barlow Twins–style)	$b = 128$	0.024	2	2^{-11}

(b) Covariance regularization

Method	Grouping	Loss scale	q	α	μ	ν
VICReg	—	—	—	25.0	25.0	1.0
Proposed (VICReg-style)	no	—	1	2.5	2.5	0.1

(c) Hyperparameters for optimization.

Learning rate	Weight decay	Batch size	Warmup epochs
0.25	10^{-6}	1024	10

(d) Hyperparameters for linear evaluation on ImageNet.

Pretrained model	Learning rate	Learning rate decay	Weight decay	Batch size
Barlow Twins	0.3	cosine decay	10^{-6}	256
VICReg	0.02	cosine decay	10^{-6}	256
Proposed (Barlow Twins–style)	0.125	cosine decay	10^{-6}	2048
Proposed (VICReg-style)	0.125	cosine decay	10^{-6}	256

(e) Hyperparameters for object detection on VOC07+12.

Pretrained model	Learning rate	Region proposal network loss weigh
Proposed (Barlow Twins–style)	0.125	0.03125
Proposed (VICReg-style)	0.125	0.125

of the "Total time (s)" in the "run_training_epoch" line was extracted and plotted as the training time in Figs. 2, 3 and E.1 to E.4. We used a function in PyTorch[7] to monitor memory occupied by tensors. The value of "Peak Usage" in the "Allocated memory" line was used as the peak GPU memory allocation.

As regards the total training time in Tab. 4, we reported the runtime counted by WandB.[8] As mentioned in the footnote of Sec. 5.2, our experiment was performed on a commercial cloud platform that terminates a session after three days. To finish training Barlow Twins and VICReg with 8 GPUs for 1000 epochs on ImageNet, we needed three sessions, and the proposed model only two (see Tab. 4). The timing reported in Tab. 4 is the run time of these sessions that includes the time for initialization at the beginning of each session. This initialization takes only 3–5 seconds at each session and does not affect the trend observed in the table. Note that in addition to this initialization, data copy takes about 15 minutes at the start of a session, but this has already been excluded from the timing in Tab. 4.

D.5 Computational Resources

We used a cloud computing platform for the experiments. In the main experiments, we trained models with eight Nvidia A100-SXM4 GPUs on this platform. We used a single Nvidia A100 GPU to evaluate empirical complexity, except where noted.

[7] https://pytorch.org/docs/stable/generated/torch.cuda.memory_summary.html

[8] We used the value of the "Runtime" in WandB.

Table E.1. The accuracy with $q \in \{1, 2\}$ on ImageNet-100.

Model	Grouping	q	Top-1	Top-5
Proposed (Barlow Twins–style)	no	1	75.94	94.28
		2	**79.94**	**94.76**
	$b = 128$	1	76.44	94.46
		2	**81.02**	**95.24**
Proposed (VICReg-style)	no	1	**79.20**	**94.96**
		2	57.98	84.56
	$b = 128$	1	**80.04**	**94.98**
		2	71.78	92.54

D.6 License of the Assets

PyTorch has a BSD-style license[9]. Solo-learn has an MIT license[10]. PyTorch Lightning is licensed under the Apache License 2.0[11]. The ImageNet[12] dataset is publicly available and frequently used as the benchmark dataset. The category list of ImageNet-100 is also publicly available[13].

E Additional Experiments

E.1 Impact of Hyperparameter q

We investigate the effect of the hyperparameter q in our regularizers (Eqs. (6) and (13)). Tab. E.1 shows the results with $q \in \{1, 2\}$ on ImageNet-100 ($d = 2048$). The results indicate that $q = 1$ works better than $q = 2$ in VICReg-style covariance regularizers. Conversely, $q = 2$ works well in Barlow Twins–style cross-correlation regularizers.

E.2 Training Time with ResNet-50 Backbone

Figure E.1 presents the single GPU training cost with ResNet-50 on ImageNet. Due to GPU memory limitation, we were unable to run Barlow Twins and VICReg at $d = 16384$, and also the grouped version of the proposed models with block size $b = 128$. Instead of $d = 16384$, we present the results at $d = 10000$.[14] As in the results of ImageNet-100, we can observe that the proposed regularizers are more efficient than the existing regularizers. At $d = 8192$, the proposed model (without grouping) is 1.3 (= 41414.3/31280.0) times faster than VICReg, and 1.2 (= 37522.0/31306.0) times faster than Barlow Twins; while at $d = 10000$, it is 1.5 (= 47448.0/32208.3) times faster than VICReg, and 1.3 times (= 41546.3/32143.7) faster than Barlow Twins.

E.3 Training Time with Distributed Data Parallel

Except for Tab. 4, the training time figures reported in Sec. 5 and Appendix E.2 were measured on a single GPU. Here we evaluate the timing of distributed data parallel (DDP) training, when multiple GPUs are available. Fig. E.2 shows the elapsed time of DDP training for ten epochs on eight A100 GPUs. With DDP, the cost of communication between GPUs emerges as an additional factor determining the total computational time, and hence the relative merit of our method in reducing loss computation time is expected to diminish. Although this is certainly true, our method is still effective, improving the computation time by a factor of more than 2.2 (= 945.5/428.7) for VICReg and 2.0 (= 740.6/366.4) for Barlow Twins when $d = 8192$ and a factor of 4.4 (= 2833.3/647.1) and 3.1 (= 1943.7/622.7) when $d = 16384$.

Fig. E.3 shows the elapsed time on ImageNet with ResNet-50. As in ImageNet-100 with ResNet-18 (Fig. E.2), our method improves speed, but the speed-up factor is smaller: 1.4 (= 6658.1/4858.9) for VICReg and 1.2 (=

[9] https://github.com/pytorch/pytorch/blob/master/LICENSE
[10] https://github.com/vturrisi/solo-learn/blob/main/LICENSE
[11] https://github.com/Lightning-AI/lightning/blob/master/LICENSE
[12] https://www.image-net.org/
[13] https://github.com/HobbitLong/CMC/blob/master/imagenet100.txt
[14] In Section E.3 (see Fig. E.4), we show the results with $d = 16384$ using multi-node DDP computation.

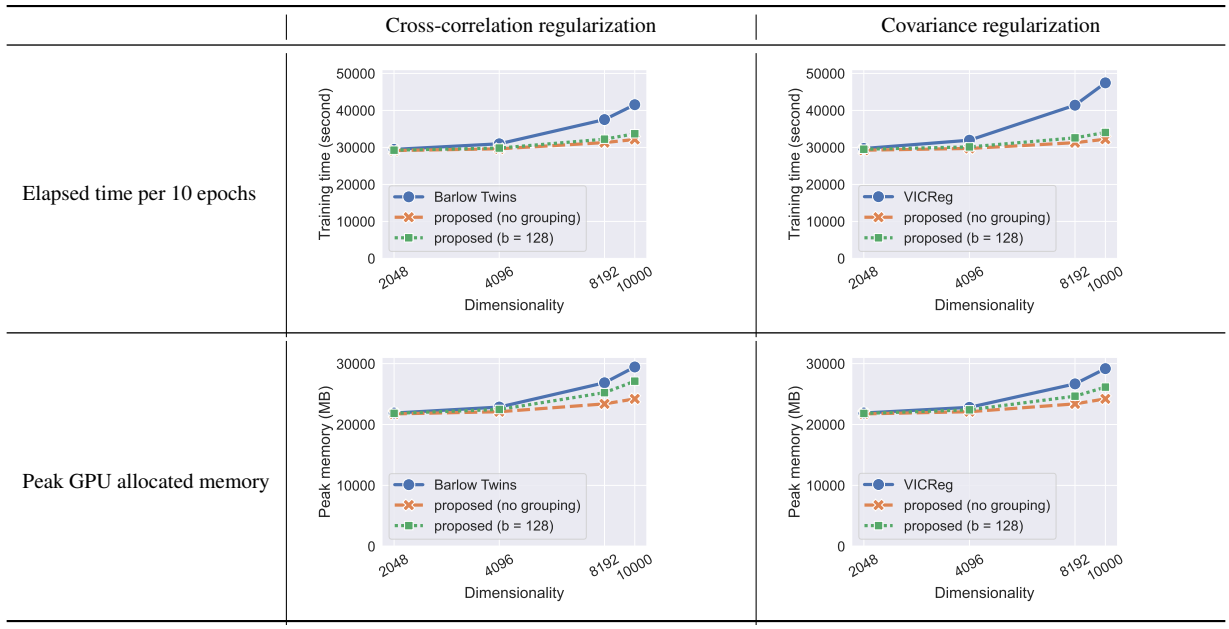

Figure E.1. Training time and memory usage on ImageNet with ResNet-50 on a single GPU.

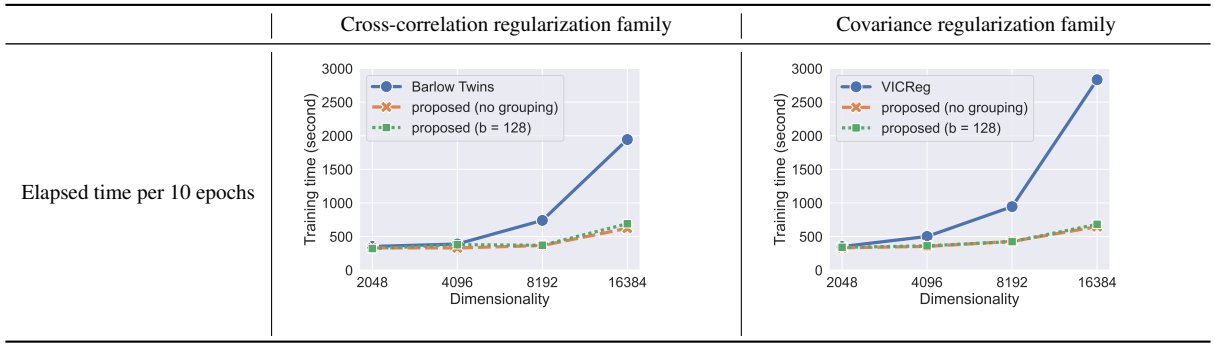

Figure E.2. The elapsed DDP training time on ImageNet-100 with ResNet-18.

5657.6/4868.7) for Barlow Twins when $d = 8192$. When $d = 10000$, the factors are 1.5 (= 7729.9/5035.1) for VICReg and 1.2 (= 6247.7/5129.6) for Barlow Twins.

In the ImageNet experiments (Figs. E.1 and E.3), all models triggered an out-of-GPU-memory error when $d = 16384$. We thus use multi-node DDP training for $d = 16384$. We evaluated two situations: training using 2 nodes and 4 nodes. In both situations, we set the effective batch size to 1024. Figure E.4 shows the results. Barlow Twins and VICReg still ran out of memory under 2 nodes, but the proposed models (with or without grouping) worked in this situation thanks to their efficient memory usage. With 4 nodes, all models were trained successfully, and the proposed models were slightly faster than Barlow Twins and VICReg. However, in this setting, there is no point in training our models using 4 nodes, when they are trainable on 2 nodes. If we compare the speed of our models trained on 2 nodes with Barlow Twins and VICReg (which failed to be trained on 2 nodes) on 4 nodes, the advantage of our models becomes more pronounced.

E.4 Computation Time for Forward and Backward Passes

We analyzed the training time spent for the forward pass (in the model network excluding the loss node), forward loss computation, and backpropagation. These three types are denoted by "Forward (model)", "Forward (loss)", and "Backward (model + loss)", respectively. These measurements were quoted from the "Total time (s)" column in the respective lines in the output of the Simple Profiler in PyTorch Lightning. Note that by default, the profiler only

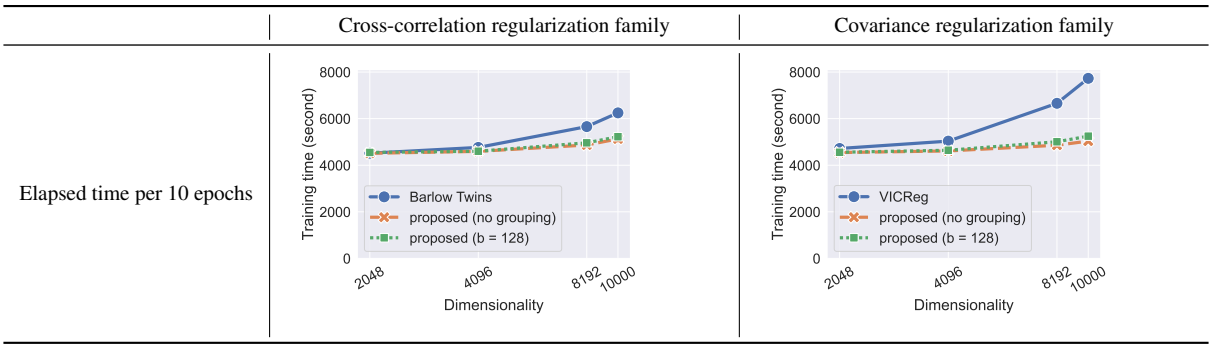

Figure E.3. The elapsed DDP training time on ImageNet with ResNet-50.

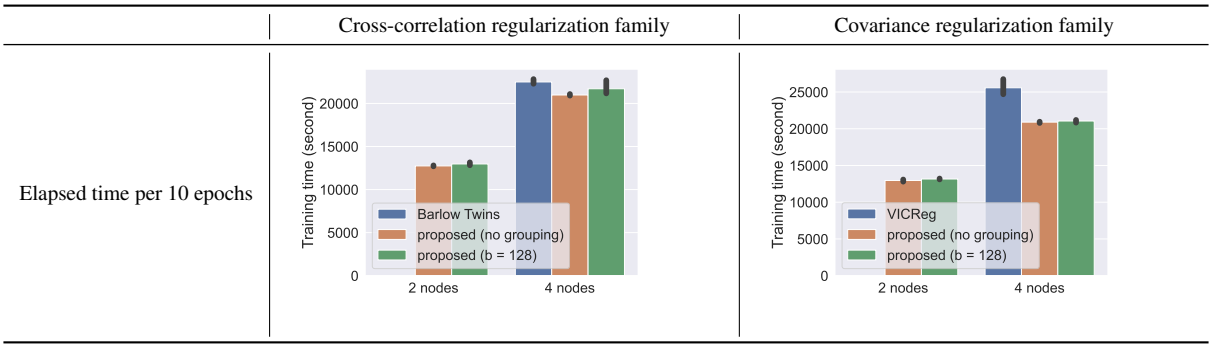

Figure E.4. The elapsed multi-node DDP training time on ImageNet with ResNet-50 (d = 16384).

reports total forward computation time. To separate the total time into "Forward (model)" and "Forward (loss)", we specified custom measurement points in our code where the model (backbone network and projection network) forward computation and the loss computation are performed.[15] Since the custom measurement points did not work for backpropagation, we plot the value of "Strategy.backward" line as the total backpropagation time "Backward (model + loss)".

Fig. E.5 presents the results. The times for loss computation and backpropagation both improved with our method; see Tabs. E.2 and E.3 for the detail of improvements. The improvement in backpropagation is due to reduced time at the loss node, because Barlow Twins and the proposed method have the same model network and differ only in the loss used. This reduction in the backpropagation time is not surprising because the same computation graph is traversed in both the forward and backward passes, and therefore their asymptotic computational complexity is roughly identical.

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

Table E.2. The improvements for loss and backward computations on ImageNet100 with ResNet-18.

Model	#GPU	d	Forward (loss)	Backward (model + loss)
Barlow Twins–style	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-style	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)

Table E.3. The improvements for loss and backward computations on ImageNet with ResNet-50.

Model	#GPU	d	Forward (loss)	Backward (model + loss)
Barlow Twins–style	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-style	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)

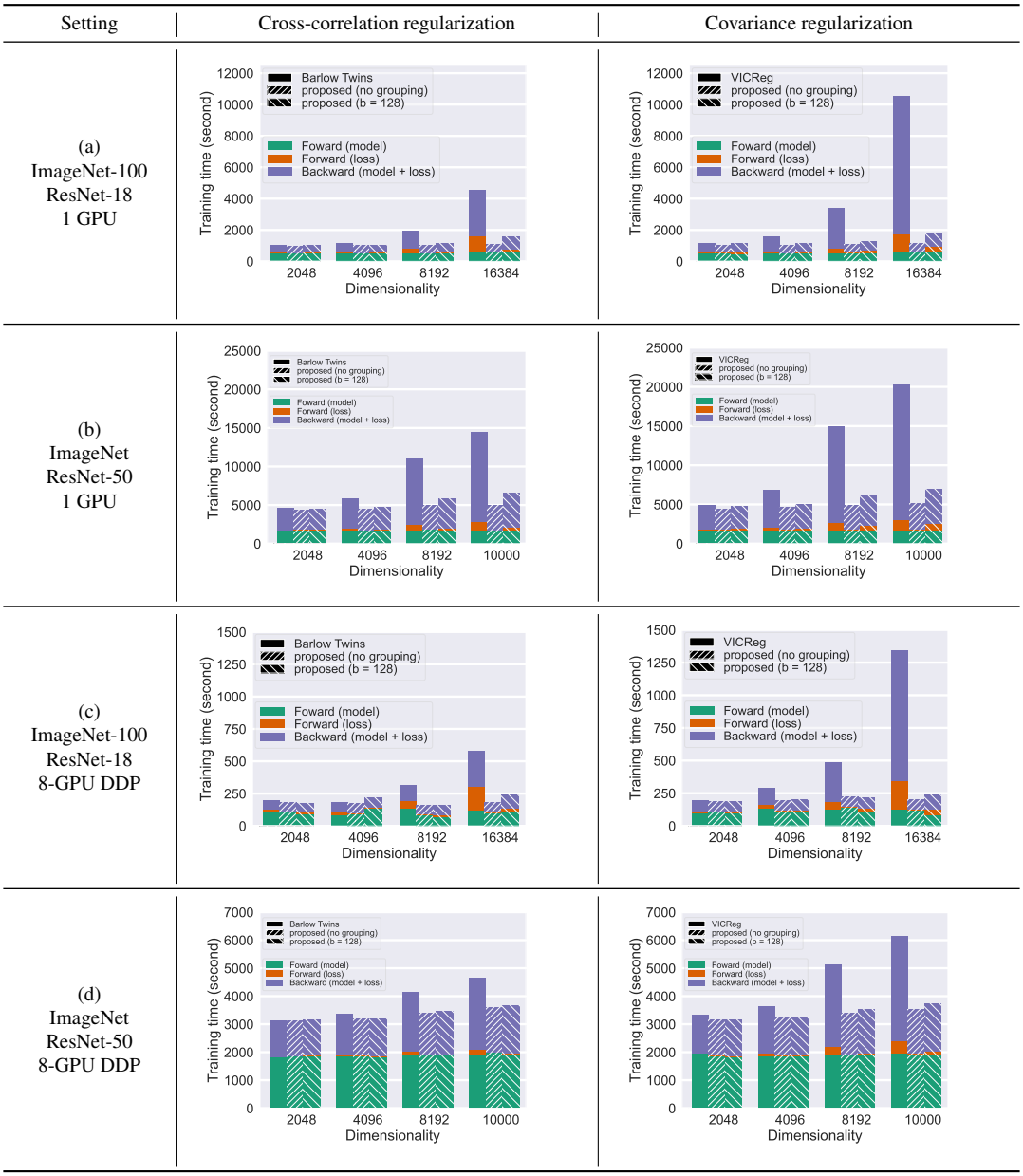

Figure E.5. The elapsed 10-epoch training time spent on forward model (green) and loss (orange) computation, and backpropagation (total of model and loss; purple).

F Code

Listings 1 and 2 present Python-based implementations for covariance and cross-correlation regularizers (without feature grouping). As explained in Sec. 4, the summary vectors can be efficiently calculated with FFT (see Listing 3).

In the computation of the proposed regularizers, we do not conduct collective operations, such as all-reduce and gather functions.

References

[Bie20] Lukas Biewald. Experiment tracking with Weights and Biases. Software available from https://www.wandb.com/, 2020. 3

[BPL22] Adrien Bardes, Jean Ponce, and Yann LeCun. VICReg: Variance-invariance-covariance regularization for self-supervised learning. In *ICLR*, 2022. 4, 5

[HFW+20] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, pages 9729–9738, 2020. 5

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035, 2019. 3

[Pla03] Tony Plate. *Holographic Reduced Representation: Distributed Representation for Cognitive Structures.* CSLI Lecture Notes No. 150. CSLI Publications, 2003. 2

[Smi08] Julius O. Smith, III. *Mathematics of the Discrete Fourier Transform (DFT) with Audio Applications*. W3K Publishing, 2nd edition, 2008. 2

[TFN+22] Victor G. Turrisi da Costa, Enrico Fini, Moin Nabi, Nicu Sebe, and Elisa Ricci. solo-learn: A library of self-supervised methods for visual representation learning. *Journal of Machine Learning Research*, 23(56):1–6, 2022. 3, 4, 5

[WKM+19] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. 3

[YGG17] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv.cs preprint*, 1708.03888, 2017. 4

[ZJM+21] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéhane Deny. Barlow Twins: Self-supervised learning via redundancy reduction. In *ICML*, pages 12310–12320, 2021. 4, 5

```
# Z1, Z2: projected image embeddings (n x d)
# q: a hyperprameter for L_q^q norm

def xsum_regularizer(Z1, Z2, G, q):
    # pre-process: centering and normalization
    Z1 = batch_normalization(Z1)
    Z2 = batch_normalization(Z2)

    # feature permutation
    idx = torch.randperm(Z1.shape[1])
    Z1 = Z1[:, idx]
    Z2 = Z2[:, idx]

    # summary vector
    sumvec = cal_sumvec(Z1, Z2, 0) / n

    # loss for off-diagonal elements
    if q == 1:
        loss = torch.sum(sumvec[1:].abs())
    elif q == 2:
        loss = torch.sum(sumvec[1:].pow(2))

    return loss
```

Listing 1. Computing Barlow Twins–style cross-correlation regularizer

```
# Z: projected image embeddings ([n: batch size] x [d: embedding dimension])
# q: a hyperparameter for L_q^q norm

def covsum_regularizer(Z, blck_size, q):
    # pre-process: centering
    Z =  Z - Z.mean(dim=0)

    # feature permutation
    idx = torch.randperm(Z.shape[1])
    Z = Z[:, idx]

    # summary vector
    sumvec = cal_sumvec(Z, Z, 0) / (n - 1)

    # loss for off-diagonal elements
    if q == 1:
        loss = torch.sum(sumvec[1:].abs())
    elif q == 2:
        loss = torch.sum(sumvec[1:].pow(2))

    return loss
```

Listing 2. Computing VICReg-style covariance regularizer

```
def cal_sumvec(z1, z2, dim):
  fz1 = fft.rfft(z1)
  fz2 = fft.rfft(z2)
  fz1_conj = fz1.conj()
  fz_prod = fz1_conj * fz2
  fc = fz_prod.sum(dim=dim)
  sumvec= fft.irfft(fc)
  return sumvec
```

Listing 3. Summary vector computation