A Relation to other self-supervised methods

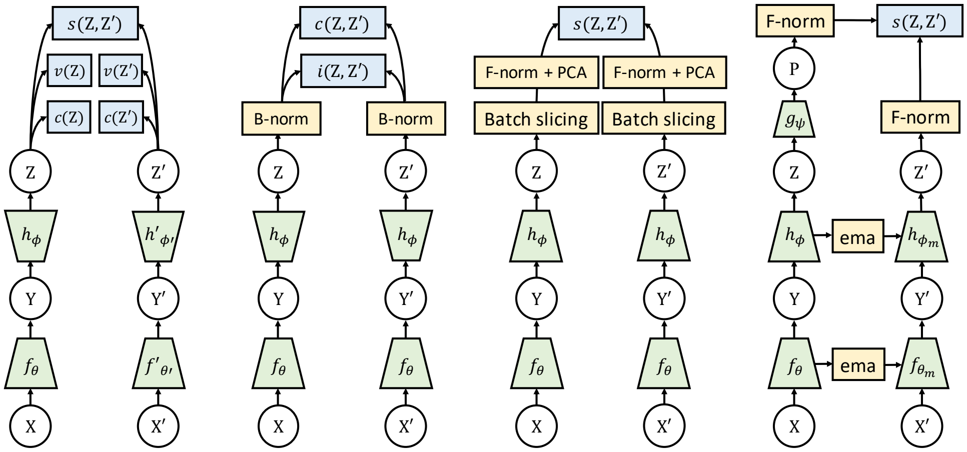
We compare here VICReg with other methods in terms of methodology, and we discuss the mechanisms used by these methods to avoid collapse and to learn representations, and how they relate to VICReg. We synthesize and illustrate the differences between these methods in Figure 2.

Relation to Barlow Twins [9]. VICReg uses the same decorrelation mechanism as Barlow Twins, which consists in penalizing the off-diagonal terms of a covariance matrix computed on the embeddings. However, Barlow Twins uses the cross-correlation matrix where each entry in the matrix is a cross-correlation between two vectors z^i and z'^j, from the two branches of the siamese architecture. Instead of using cross-correlations, we simply use the covariance matrix of each branch individually, and the variance term of VICReg allows us to get rid of standardization. Indeed, Barlow Twins forces the correlations between pairs of vectors z^i and z'^i from the same dimension i to be 1. Without normalization, this target value of 1 becomes arbitrary and the vectors take values in a wider range. Moreover, there is an undesirable phenomenon happening in Barlow Twins, the embeddings before standardization can shrink and become constant to numerical precision, which could cause numerical instabilities. In practice, this is solved by adding a constant scalar in the denominator of standardization of the embeddings. Without normalization, VICReg naturally avoids this edge case.

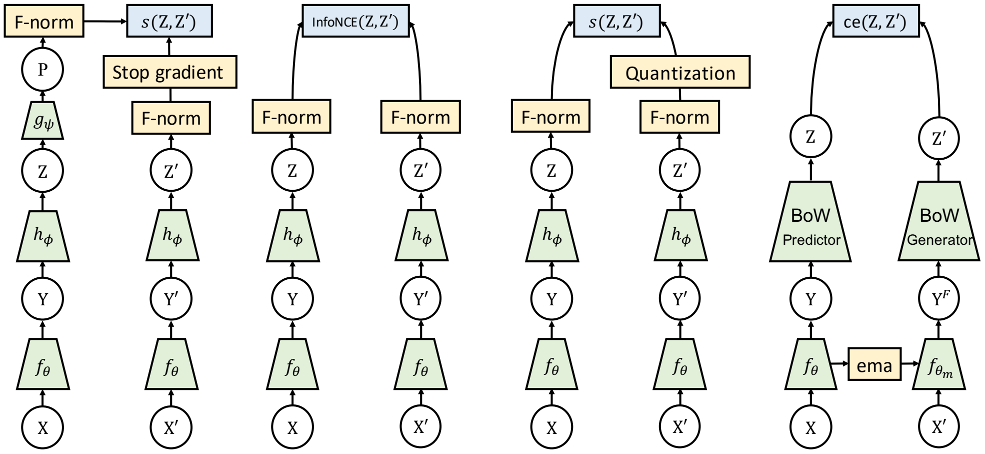
Relation to W-MSE [16]. The whitening operation of W-MSE consists in computing the inverse covariance matrix of the embeddings and use its square root as a whitening operator on the embeddings. Using this operator has two downsides. First, matrix inversion is a very costly and potentially unstable operation. VICReg does not need to inverse the covariance matrix. Second, as mentioned in [16] the whitening operator is constructed over several consecutive iteration batches and therefore might have a high variance, which biases the estimation of the mean-squared error. This issue is overcome in practice by a batch slicing strategy, where the whitening operator is computed over randomly constructed sub-batches. VICReg does not apply any operator on the embeddings, but instead regularizes the variance and covariance of the embeddings using an additional constraint.

Relation to BYOL and SimSiam [6, 7]. The core components that avoid collapse in BYOL and SimSiam are the average moving weights and the stop-gradient operation on one side of their asymmetric architecture, which play the role of the repulsive term used in other methods. Our experiments in Appendix C.5 show that in addition to preventing collapse, these components also have a decorrelation effect. In addition, we have conducted the following experiment: We compute the correlation matrix of the final representations obtained with SimSiam, BYOL, VICReg and VICReg without covariance regularization. We measure the average correlation coefficient and observe that this coefficient is much smaller for SimSiam, BYOL and VICReg, compared to VICReg without covariance regularization. We observe in Figure 5 that even without covariance regularization, SimSiam and BYOL naturally minimize the average correlation coefficient of the representations. VICReg replaces the moving average weights and the stop-gradient operation, which are architectural trick that require some dependency between the branches, by an explicit constraint on the variance and the covariance of both embeddings separately, which achieves the same goal of decorrelating the representations and avoiding collapse, while being clearer, more interpretable, and working with independent branches.

Relation to SimCLR, SwAV and OBoW [5, 12, 8]. Contrastive and clustering based self-supervised algorithms rely on direct comparisons between elements of negative pairs. In the case of SimCLR, the negative pairs involve embeddings mined from the current batch, and large batch sizes are required. Despite the fact that SwAV computes clusters using elements in the current batch, it does not seem to have the same dependency on batch size. However, it still requires a lot of prototype vectors for negative comparisons between embeddings and codes. VICReg eliminates the negative comparisons and replace them by an explicit constraint on the variance of the embeddings, which efficiently plays the role of a negative term between the vectors. SwAV can also be interpreted as a distillation method, where a teacher network produces quantized vectors, used as target for a student network. Ensuring an equal partition of the quantized vectors in different bins or clusters effectively prevents collapse. OBOW can also be interpreted under the same framework. The embeddings are bag-of-words over a vocabulary of visual features, and collapse is avoided by the underlying quantization operation.



(a) VICReg (b) Barlow Twins (c) W-MSE (d) BYOL



(e) SimSiam (f) SimCLR (g) SwAV (h) OBoW

Figure 2: **Conceptual comparison between different self-supervised methods.** The inputs X and X' are fed to an encoder f with weights θ. The representations Y and Y' are further processed by a network h with weights ψ. h can be a projector (narrowing trapeze) that reduces the dimensionality of the representations, or an expander (widening trapeze) that increases their dimensionality. A criterion is finally applied on the embeddings Z and Z'. VICReg (a) works when both branches have encoders f and f' with different architectures and sets of weights θ and θ'. Each branch's variance and covariance are regularized by regularizers v and c, and the distance between both branches is minimized with a mean-squared error loss s. Barlow Twins (b) uses a loss c to decorrelate pairs of different dimensions in the batch-wise normalized (B-Norm) embeddings, and learns invariance with a loss i that makes similar dimensions highly correlated. W-MSE (c) uses a batch slicing operation that shuffles batches into small sub-batches, and apply PCA as a whitening operation on the feature-wise normalized (F-Norm) embeddings of each sub-batch. BYOL (d) has an asymmetric architecture where the weights θ_m of one encoder are an exponential moving average (ema) of the other encoder's weights θ. A predictor g with weights ψ is used in the branch with learnable weights. SimSiam (e) uses a predictor on one branch and a stop-gradient operation (sg) on the other one. SimCLR (f) uses the InfoNCE contrastive loss where all the feature-wise normalized embeddings are compared between them inside a batch. Samples from distorted versions of the same input are brought close to each other, while other samples are pushed away. SwAV (g) quantizes the feature-wise normalized embeddings of a branch and use it as target for the other one. OBoW (h) uses bag-of-words (BoW) representations and a cross-entropy loss to compare the BoW generated by a teacher network from the feature maps Y^F of the encoder, to the BoW predicted by a student network. Green blocks: parametric functions; yellow boxes: non-parametric functions; blue boxes: objective functions.

B Additional implementation details

B.1 Data augmentation

We follow the image augmentation protocol first introduced in SimCLR [12] and now commonly used by similar approaches based on siamese networks [5, 6, 7, 9]. Two random crops from the input image are sampled and resized to 224 × 224, followed by random horizontal flip, color jittering of brightness, contrast, saturation and hue, Gaussian blur and random grayscale. Each crop is normalized in each color channel using the ImageNet mean and standard deviation pixel values. In more details, the exact set of augmentations is based on BYOL [6] data augmentation pipeline but is symmetrised. The following operations are performed sequentially to produce each view:

- Random cropping with an area uniformly sampled with size ratio between 0.08 to 1.0, followed by resizing to size 224 × 224. `RandomResizedCrop(224, scale=(0.08, 0.1))` in PyTorch.
- Random horizontal flip with probability 0.5.
- Color jittering of brightness, contrast, saturation and hue, with probability 0.8. `ColorJitter(0.4, 0.4, 0.2, 0.1)` in PyTorch.
- Grayscale with probability 0.2.
- Gaussian blur with probability 0.5 and kernel size 23.
- Solarization with probability 0.1.
- color normalization with mean (0.485, 0.456, 0.406) and standard deviation (0.229, 0.224, 0.225).

B.2 ImageNet evaluation

Linear evaluation. We follow standard procedure and train a linear classifier on top of the frozen representations of a ResNet-50 pretrained with VICReg. We use the SGD optimizer with a learning rate of 0.02, a weight decay of 10^{-6}, a batch size of 256, and train for 100 epochs. The learning rate follows a cosine decay. The training data augmentation pipeline is composed of random cropping and resize of ratio 0.2 to 1.0 with size 224 × 224, and random horizontal flips. During evaluation the validation images are simply center cropped and resized to 224 × 224.

Semi-supervised evaluation. We train a linear classifier and fine-tune the representations using 1 and 10% of the labels. We use the SGD optimizer with no weight decay and a batch size of 256, and train for 20 epochs. We perform a grid search on the values of the encoder and linear head learning rates. In the 10% of labels case, we use a learning rate of 0.01 for the encoder and 0.1 for the linear head. In the 1% of labels case we use 0.03 for the encoder and 0.08 for the linear head. The two learning rates follow a cosine decay schedule. The training data and validation augmentation pipelines are identical to the linear evaluation data augmentation pipelines.

B.3 Transfer learning

We use the VISSL library [48] for linear classification tasks and the detectron2 library [49] for object detection and segmentation tasks.

Linear classification. We follow standard protocols [2, 5, 9] and train linear models on top of the frozen representations. For VOC07 [43], we train a linear SVM with LIBLINEAR [50]. The images are center cropped and resized to 224 × 224, and the C values are computed with cross-validation. For Places205 [42] we use SGD with a learning rate of 0.003, a weight decay of 0.0001, a momentum of 0.9 and a batch size of 256, for 28 epochs. The learning rate is divided by 10 at epochs 4, 8 and 12. For Inaturalist2018 [44], we use SGD with a learning rate of 0.005, a weight decay of 0.0001, a momentum of 0.9 and a batch size of 256, for 84 epochs. The learning rate is divided by 10 at epochs 24, 48 and 72.

Object detection and instance segmentation. Following the setup of [3, 9], we use the `trainval` split of VOC07+12 with 16K images for training and a Faster R-CNN C-4 backbone for 24K iterations with a batch size of 16. The backbone is initialized with our pretrained ResNet-50 backbone. We use a learning rate of 0.1, divided by 10 at iteration 18K and 22K, a linear warmup with slope of 0.333 for 1000 iterations, and a region proposal network loss weight of 0.2. For COCO we use Mask R-CNN FPN backbone for 90K iterations with a batch size of 16, a learning rate of 0.04, divided by 10 at iteration 60K and 80K and with 50 warmup iterations.

B.4 Ablation studies

We give here implementation details on the results of Table 3 with BYOL and SimSiam, as well as the default setup for VICReg with 100 epochs of pretraining, used in all our ablations. For both BYOL and SimSiam experiments, the variance criterion has coefficient $\mu = 1$ and the covariance criterion has coefficient $\nu = 0.01$, the data augmentation pipeline and the architectures of the expander and predictor exactly follow the pipeline and architectures described in their paper. The linear evaluation setup of each methods follows closely the setup described in the original papers.

BYOL setup. We use our own BYOL implementation in PyTorch, which outperforms the original implementation for 100 epochs of pretraining (69.3% accuracy on the linear evaluation protocol against 66.5% for the original implementation) and matches its performance for 1000 epochs of pretraining. We use the LARS optimizer [37], with a learning rate of $base\_lr * batch\_size/256$ where $base\_lr$ = 0.45, and $batch\_size$ = 4096, a weight decay of 10^{-6}, an eta value of 0.001 and a momentum of 0.9, for 100 epoch of pretraining with 10 epochs of warmup. The learning rate follows a cosine decay schedule. The initial value of the exponential moving average factor is 0.99 and follows a cosine decay schedule.

SimSiam setup. We use our own implementation of SimSiam, which reproduces exactly the performance reported in the paper [7]. We use SGD with a learning rate of $base\_lr * batch\_size/256$ where $base\_lr$ = 0.05, $batch\_size$ = 2048, with a weight decay of 0.0001 and a momentum of 0.9 for 100 epochs of pretraining and 10 epochs of warmup. The learning rate of the encoder and the expander follow a cosine decay schedule while the learning rate of the predictor is kept fixed.

VICReg setup. The setting of VICReg's experiments is identical to the setting described in section 4.2, except that the number of pretraining epochs is 100 and the base learning rate is 0.3. The base learning rates used for the batch size study are 0.8, 0.5 and 0.4 for batch size 128, 256 and 512 respectively, and 0.3 for all other batch sizes. When a predictor is used, it has a similar architecture as the expander described in section 4.2, but with 2 layers instead of 3, which gives better results in practice.

C Additional results

C.1 Other ResNet architectures

Table 7 reports the performance of VICReg on linear classification with large ResNet architectures. We focus on the wider family of ResNet [51] and aggregated ResNet [52], and we consider two ways of widening a standard ResNet. First, we follow standard practice in recent self-supervised learning work [5, 6, 12] and multiple by 2 or 4 the number of filters in every convolutional layer, which also has the effect of multiplying the dimensionality of the representations. Second, as originally proposed in [51], we only multiply the number of filters in the bottleneck layers, which does not increases the dimensionality of the representations. We call this architecture Narrow ResNet (with prefix N- in Table 7). The main observation we make is the dependency of VICReg on the dimensionality of the representation. Using the narrow architecture, the performance of VICReg, jumps from 73.2% top-1 accuracy on linear classification with a ResNet-50, to 74.7% with Narrow ResNet-50 (x2), which is a 1.5% improvement and 76.0% with Narrow ResNet-50 (x4), which is a 2.8% improvement. We observe a similar trend going from ResNet-50 to ResNet-50 (x2), which is a 2.3% improvement but the performance completely saturates with ResNet-50 (x4), which is a 0.1% improvement over ResNet-50 (x2). Table 8 reports the performance of VICReg on semi-supervised classification with large ResNet architectures. VICReg combined with a ResNet-50 (x2) outperforms the current state-of-the-art methods BYOL and SimCLR, using this encoder architecture. Our largest model ResNet-200 (x2) performs lower than BYOL when 1% of the labels are used but is on par with 10% of the labels. These results demonstrate the capabilities of VICReg to scale up when large architectures are used.

C.2 K-nearest-neighbors

Following recent protocols [5, 21, 28], we evaluate the learnt representations using K-nearest-neighbors classifiers built on the training set of ImageNet and evaluated on the validation set of ImageNet. We report the results with K=20 and K=200 in Table 9. VICReg performs slightly lower than other methods in the 20-NN case but remains competitive in the 200-NN case. These results with K-NN classifiers demonstrate the potential applicability of VICReg to downstream tasks based on nearest neighbors search, such as content retrieval in images or videos.

Table 7: **Linear classification with large architectures.** Top-1 accuracy comparison between different methods using various encoder architectures. For all VICReg results, the output dimensionality of the expander is 8192. N-R stands for Narrow ResNet, where only the bottleneck convolutional layers are widen.

Method	Arch.	Param.	Repr.	Top-1	Top-5
SimCLR [12]	R50 (x2)	93M	4096	74.2	92.0
	R50 (x4)	375M	8192	76.5	93.2
SwAV [5]	R50 (x2)	93M	4096	77.3	-
	R50 (x4)	375M	8192	77.9	-
	R50 (x5)	586M	10240	78.5	-
BYOL [6]	R50 (x2)	93M	4096	77.4	93.6
	R50 (x4)	375M	8192	78.6	94.2
	R200 (x2)	250M	4096	79.6	94.8
VICReg (ours)	N-R50 (x2)	66M	2048	74.7	91.9
	N-R50 (x4)	221M	2048	76.0	92.4
	R50 (x2)	93M	4096	75.5	92.1
	R50 (x4)	375M	8192	75.6	92.2
	RNXT101-32-16	191M	2048	76.1	92.3
	R200 (x2)	250M	4096	77.3	93.3

Table 8: **Semi-supervised classification with large architectures.** Top-1 accuracy comparison between different methods using various encoder architectures. For all VICReg results, the output dimensionality of the expander is 8192.

Method	Arch.	Param.	Repr.	Top-1		Top-5	
				1%	10%	1%	10 %
SimCLR [12]	R50 (x2)	93M	4096	58.5	71.7	83.0	91.2
	R50 (x4)	375M	8192	63.0	74.4	85.8	92.6
BYOL [6]	R50 (x2)	93M	4096	62.2	73.5	84.1	91.7
	R50 (x4)	375M	8192	69.1	75.7	87.9	92.5
	R200 (x2)	250M	4096	71.2	77.7	89.5	93.7
VICReg (ours)	R50 (x2)	93M	4096	62.6	73.9	84.5	91.8
	R200 (x2)	250M	4096	68.8	77.3	88.2	93.6

Table 9: **K-NN classifiers on ImageNet.** Top-1 accuracy with 20 and 200 nearest neighbors.

Method	20-NN	200-NN
NPID [21]	-	46.5
LA [28]	-	49.4
PCL [53]	54.5	-
BYOL [6]	66.7	64.9
SwAV [5]	65.7	62.7
Barlow Twins [9]	64.8	62.9
VICReg	64.5	62.8

Table 10: **Impact of expander dimensionality.** Top-1 accuracy on the linear evaluation protocol with 100 pretraining epochs.

Dimensionality	256	512	1024	2048	4096	8192	16834
Top-1	55.9	59.2	62.4	65.1	67.3	68.6	68.8

Table 11: **Impact of batch size.** Top-1 accuracy on the linear evaluation protocol with 100 pretraining epochs.

Batch size	128	256	512	1024	2048	4096
Top-1	67.3	67.9	68.2	68.3	68.6	67.8

C.3 Expander network architecture

VICReg borrows the decorrelation mechanism of Barlow Twins [9] and we observe that it therefore has the same dependency on the dimensionality of the expander network. Table 10 reports the impact of the width and depth of the expander network. The dimensionality corresponds the number of hidden and output units in the expander network during pretraining. As the dimensionality increases, the performance dramatically increases from 55.9% top-1 accuracy on linear evaluation with a dimensionality of 256, to 68.8% with dimensionality 16384. The performance tends to saturate as the difference between dimensionality 8192 and 16384 is only of 0.2%.

C.4 Batch size

Contrastive methods suffer from the need of a lot of negative examples which can translate into the need for very large batch sizes [12]. Table 11 reports the performance on linear classification when the size of the batch varies between 128 and 4096. For each value of batch size, we perform a grid search on the base learning rate described in Appendix B.4. We observe a 0.7% and 1.2% drop in accuracy with small batch size of 256 and 128 which is comparable with the robustness to batch size of Barlow Twins [9] and SimSiam [7], and a 0.8% drop with a batch size of 4096, which is reasonable and allows our method to be very easily parallelized on multiple GPUs.

C.5 Combination with BYOL and SimSiam

BYOL [6] and SimSiam [7] rely on a effective but difficult to interpret mechanism for preventing collapse, which may lead to instabilities during the training. We incorporate our variance regularization loss into BYOL and SimSiam and show that it helps stabilize the training and offers a small performance improvement. For both methods, the results are obtained using our own implementation and the exact same data augmentation and optimization settings as in their original paper. The variance and covariance regularization losses are incorporated with a factor of $\mu = 1$ for variance and $\nu = 0.01$ for covariance. We report in Figure 3 the improvement obtained over these methods on the linear evaluation protocol for different number of pre-training epochs. For BYOL the improvement is of 0.9% with 100 epochs and becomes less significant as the number of pre-training epochs increases with a 0.2% improvement with 1000 epochs. This indicates that variance regularization makes BYOL converge faster. In SimSiam the improvement is not as significant. We plot in Figure 4 the evolution of the standard deviation computed along each dimension and averaged across the dimensions of the representation and the embeddings, during BYOL and SimSiam pretraining. For both methods, the standard deviation computed on the embeddings perfectly matches $1/\sqrt{d}$ where d is the dimension of the embeddings, which indicates that the embeddings are perfectly spread-out across the unit sphere. This translates in an increased standard deviation at the representation level, which seems to be correlated to the performance improvement. We finally study in Figure 5 the evolution of the average correlation coefficient, during pretraining of BYOL and SimSiam, with and without variance and covariance regularization. The average correlation coefficient is computed by averaging the off-diagonal coefficients of the correlation matrix of the representations:

$$\frac{1}{2d(d-1)} \sum_{i \neq j} C(Y)_{i,j}^2 + C(Y')_{i,j}^2, \tag{9}$$

where Y and Y' are the standardized representations and C is defined in Eq. (3). In BYOL this coefficient is much lower using covariance regularization, which translate in a small improvement of

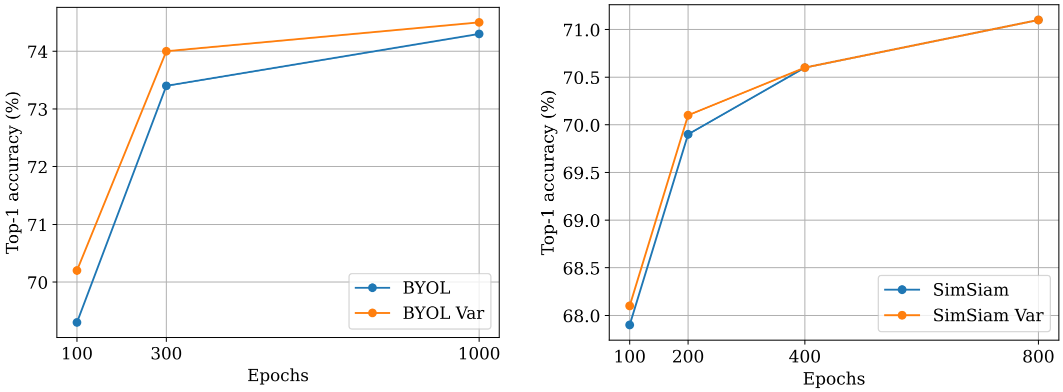


Figure 3: **Incorporating variance regularization in BYOL and SimSiam.** Top-1 accuracy on the linear evaluation protocol for different number of pretraining epochs. For both methods pre-training follows the optimization and data augmentation protocol of their original paper but is based on our implementation. *Var* indicates variance regularization

the performance, according to Table 3. We do not observe the same improvement in SimSiam, both in terms of correlation coefficient, and in terms of performance on linear classification. The average correlation coefficient is correlated with the performance, which motivates the fact that decorrelation and redundancy reduction are core mechanisms for learning self-supervised representations.

D Running time

We report in Table 12, the running time of VICReg in comparison with other methods. All methods are run by us on 32 Tesla V100 GPUs. Each method offers a different trade-off between running time, memory and performance. SwAV is a very fast algorithm which use less memory and run faster than the other methods but with a lower performance, multi-crop helps the performance at the cost of additional compute and memory usage. BYOL has the highest memory requirement, which is due to the need of storing the target network weights. Finally, Barlow Twins and VICReg offer an interesting trade-off, consuming less memory than BYOL and SwAV with multi-crop, and running faster than SwAV with multi-crop, but with a slightly worse performance. The difference of 1h running time between Barlow Twins and VICReg is probably due to implementation details not related to the method.

Table 12: **Running time and peak memory.** Comparison between different methods, the training is distributed on 32 Tesla V100 GPUs, the running time is measured over 100 epochs and the peak memory is measured on a single GPU. We report top-1 accuracy (%) on linear classification on top of the frozen representations.

Method	time / 100 epochs	peak memory / GPU	Top-1 accuracy (%)
SwAV	9h	9.5G	71.8
SwAV (w/ multi-crop)	13h	12.9G	75.3
BYOL	10h	14.6G	74.3
Barlow Twins	12h	11.3G	73.2
VICReg	11h	11.3G	73.2

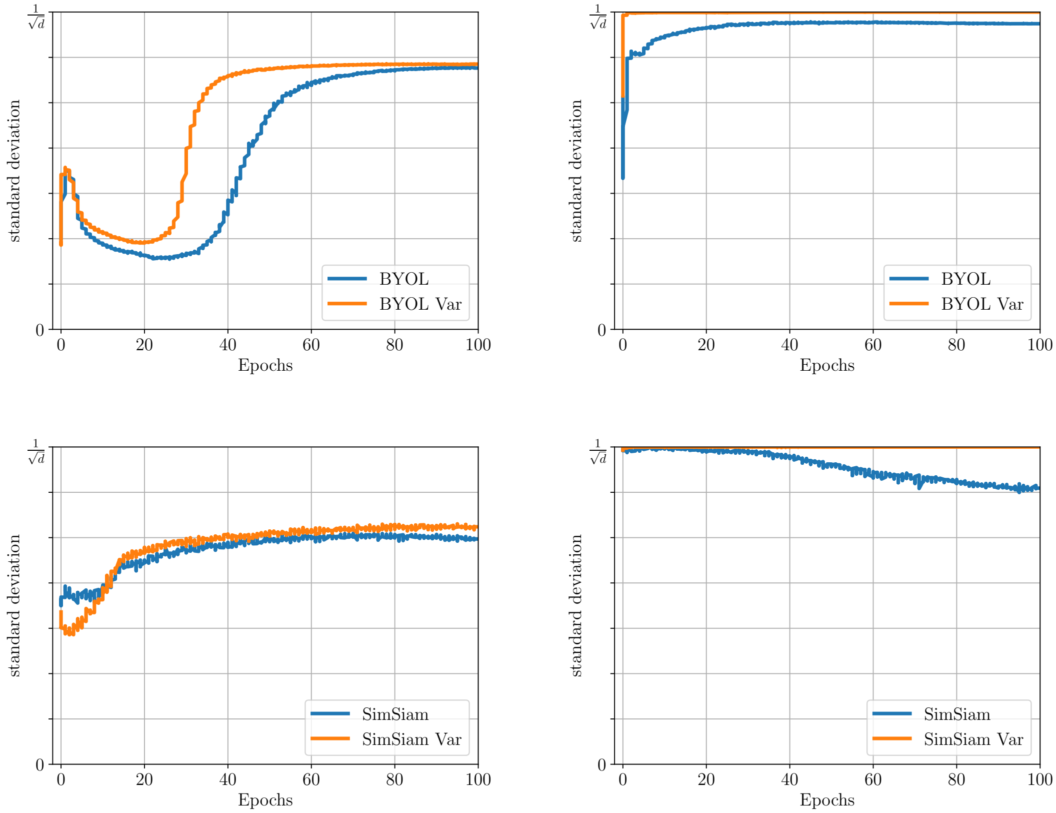


Figure 4: **Standard deviation of the features during BYOL and SimSiam pretraining.** Evolution of the average standard deviation of each dimension of the features with and without variance regularization (Var). left: the standard deviation is measured on the representations, right: the standard deviation is measured on the embeddings.

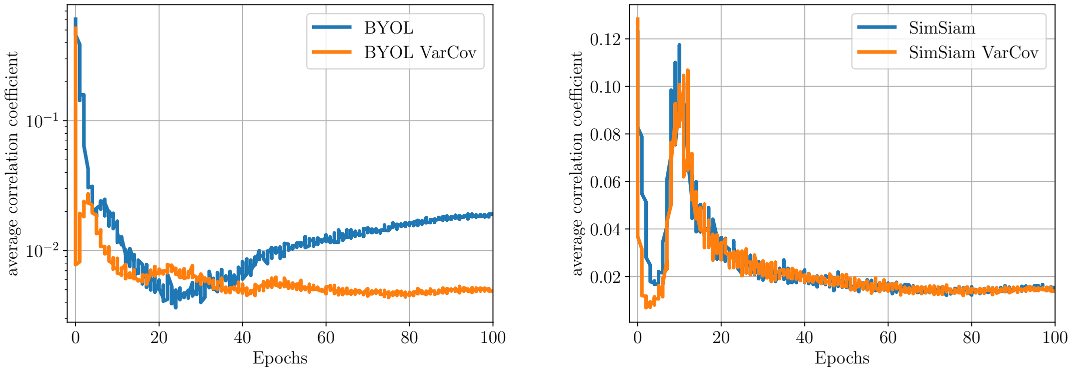


Figure 5: **Average correlation coefficient of the features during BYOL and SimSiam pretraining.** Evolution of the average correlation coefficient measured by averaging the off-diagonal terms of the correlation matrix of the representations with BYOL, BYOL with variance-covariance regularization (BYOL VarCov), SimSiam, and SimSiam with variance-covariance regularization (SimSiam VarCov).