A Background on information measures

The most natural and conventional means to measure uncertainty of a real-valued random vector $\mathbf{x}$ is to use the joint Shannon differential entropy defined by [10]

$$h(\mathbf{x}) = -\int\limits_{\text{dom}(f_{\mathbf{x}})} \log(f_{\mathbf{x}}(x)) f_{\mathbf{x}}(x) \mathrm{d}x = -E(\log(f_{\mathbf{x}}(\mathbf{x}))),$$

where $f_{\mathbf{x}}$ is the joint probability density function (pdf) of the components of $\mathbf{x}$. The corresponding Shannon mutual information (SMI) between the random vectors $\mathbf{x}$ and $\mathbf{y}$ is defined as

$$I(\mathbf{x};\mathbf{y}) = h(\mathbf{x}) - h(\mathbf{x}\,|\,\mathbf{y}) \tag{A.1}$$

where $h(\mathbf{x}\,|\,\mathbf{y}) = -E(\log(f_{\mathbf{x}\,|\,\mathbf{y}}(\mathbf{x}\,|\,\mathbf{y})))$ is the conditional entropy. Shannon mutual information is a measure of the dependence between its arguments.

For a given r-dimensional random vector $\mathbf{x}$ with the pdf $f_{\mathbf{x}}(\mathbf{x})$, log-determinant (LD) entropy is defined as [16, 17]

$$h_{LD}^{(\varepsilon)}(\mathbf{x}) = \frac{1}{2}\log\det(\mathbf{R}_{\mathbf{x}} + \varepsilon\mathbf{I}) + \frac{r}{2}\log(2\pi e), \tag{A.2}$$

where $\mathbf{R}_{\mathbf{x}}$ is the auto-covariance matrix of $\mathbf{x}$, and ε is a small nonnegative parameter defined for the diagonal perturbation on $\mathbf{R}_{\mathbf{x}}$. We note that $h_{LD}^{(\varepsilon)}(\mathbf{x})$ is equivalent to Shannon's entropy when $\mathbf{x}$ is a Gaussian vector with covariance $\mathbf{R}_{\mathbf{x}}$ and ε is equal to zero. However, we should underline that it is a standalone uncertainty measure solely based on the second-order statistics, which reflects linear dependence.

The joint LD-entropy of an r-dimensional random vector $\mathbf{x}$ and a q-dimensional random vector $\mathbf{y}$ is defined as the LD-entropy of the cascaded vector $\begin{bmatrix} \mathbf{x}^T & \mathbf{y}^T \end{bmatrix}^T$, i.e.,

$$h_{LD}^{(\varepsilon)}(\begin{bmatrix} \mathbf{x}^T & \mathbf{y}^T \end{bmatrix}^T) = \frac{1}{2}\log\det\left(\mathbf{R}_{\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}} + \varepsilon\mathbf{I}\right) + \frac{r+q}{2}\log(2\pi e)$$

where

$$\mathbf{R}_{\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}} = \begin{bmatrix} \mathbf{R}_{\mathbf{x}} & \mathbf{R}_{\mathbf{xy}} \\ \mathbf{R}_{\mathbf{xy}}^T & \mathbf{R}_{\mathbf{y}} \end{bmatrix}$$

is the covariance matrix of $\begin{bmatrix} \mathbf{x}^T & \mathbf{y}^T \end{bmatrix}^T$, $\mathbf{R}_{\mathbf{y}}$ is the auto-covariance matrix of $\mathbf{y}$, and $\mathbf{R}_{\mathbf{xy}}$ is the cross-covariance matrix of $\mathbf{x}$ and $\mathbf{y}$. Using the determinant decomposition based on Schur's complement [54], we can write

$$\begin{aligned} h_{LD}^{(\varepsilon)}(\begin{bmatrix} \mathbf{x}^T & \mathbf{y}^T \end{bmatrix}^T) &= \frac{1}{2}\log\det(\mathbf{R}_{\mathbf{y}} + \varepsilon\mathbf{I}) + \frac{q}{2}\log(2\pi e) \\ &+ \frac{1}{2}\log\det(\mathbf{R}_{\mathbf{x}} - \mathbf{R}_{\mathbf{xy}}(\mathbf{R}_{\mathbf{y}} + \varepsilon\mathbf{I})^{-1}\mathbf{R}_{\mathbf{xy}}^T + \varepsilon\mathbf{I}) + \frac{r}{2}\log(2\pi e) \\ &= h_{LD}^{(\varepsilon)}(\mathbf{y}) + h_{LD}^{(\varepsilon)}(\mathbf{x}|\mathbf{y}), \end{aligned}$$

where we defined

$$h_{LD}^{(\varepsilon)}(\mathbf{x}|\mathbf{y}) = \frac{1}{2}\log\det(\mathbf{R}_{\mathbf{x}} - \mathbf{R}_{\mathbf{xy}}(\mathbf{R}_{\mathbf{y}} + \varepsilon\mathbf{I})^{-1}\mathbf{R}_{\mathbf{xy}}^T + \varepsilon\mathbf{I}) + \frac{r}{2}\log(2\pi e), \tag{A.3}$$

as the conditional LD-entropy. We note that the argument of the $\log\det$ function in (A.3) is the auto-covariance of the error for linearly estimating $\mathbf{x}$ from $\mathbf{y}$ with respect to the minimum mean square estimation (MMSE) criterion for $\varepsilon \to 0$ (see, for example, Theorem 3.2.2 of [54]). More precisely, given that $\mu_{\mathbf{x}}$ and $\mu_{\mathbf{y}}$ represent the means of $\mathbf{x}$ and $\mathbf{y}$, respectively, $\hat{\mathbf{x}}_{MMSE} = \mathbf{R}_{\mathbf{xy}}\mathbf{R}_{\mathbf{y}}^{-1}(\mathbf{y}-\mu_{\mathbf{y}})+\mu_{\mathbf{x}}$ is the best linear MMSE estimate of $\mathbf{x}$ from $\mathbf{y}$. Therefore, if we define $\mathbf{e}_{MMSE} = \mathbf{x} - \hat{\mathbf{x}}_{MMSE}$, the auto-covariance matrix of $\mathbf{e}_{MMSE}$ is given by $\mathbf{R}_{\mathbf{e}_{MMSE}} = \mathbf{R}_{\mathbf{x}} - \mathbf{R}_{\mathbf{xy}}\mathbf{R}_{\mathbf{y}}^{-1}\mathbf{R}_{\mathbf{xy}}^T$, which is the argument of the $\log\det$ function in (A.3) for $\varepsilon \to 0$. As a result, we can view $h_{LD}^{(\varepsilon)}(\mathbf{x}|\mathbf{y})$, which is the LD-Entropy of $\mathbf{e}_{MMSE}$, as a measure of the remaining uncertainty after linearly (affinely) estimating $\mathbf{x}$ from $\mathbf{y}$ based on the MMSE criterion.

The LD-mutual information (LDMI) measure is defined based on (A.2) and (A.3) as follows:

$$\begin{aligned} I_{LD}^{(\varepsilon)}(\mathbf{x};\mathbf{y}) &= h_{LD}^{(\varepsilon)}(\mathbf{x}) - h_{LD}^{(\varepsilon)}(\mathbf{x}\,|\,\mathbf{y}) \\ &= \frac{1}{2}\log\det(\mathbf{R_x}+\varepsilon\mathbf{I}) - \frac{1}{2}\log\det(\mathbf{R_x}-\mathbf{R_{xy}}(\mathbf{R_y}+\varepsilon\mathbf{I})^{-1}\mathbf{R}_{\mathbf{xy}}^{T}+\varepsilon\mathbf{I}). \end{aligned} \tag{A.4}$$

Taking the average of (A.4) with its symmetric version obtained by exchanging $\mathbf{x}$ and $\mathbf{y}$, we can obtain an alternative but equivalent expression for LDMI:

$$\begin{aligned} I_{LD}^{(\varepsilon)}(\mathbf{x};\mathbf{y}) &= \frac{1}{4}\log\det(\mathbf{R_x}+\varepsilon\mathbf{I}) + \frac{1}{4}\log\det(\mathbf{R_y}+\varepsilon\mathbf{I}) \\ &\quad -\frac{1}{4}\log\det(\mathbf{R_x}-\mathbf{R_{xy}}(\mathbf{R_y}+\varepsilon\mathbf{I})^{-1}\mathbf{R}_{\mathbf{xy}}^{T}+\varepsilon\mathbf{I}) \\ &\quad -\frac{1}{4}\log\det(\mathbf{R_y}-\mathbf{R_{yx}}(\mathbf{R_x}+\varepsilon\mathbf{I})^{-1}\mathbf{R}_{\mathbf{yx}}^{T}+\varepsilon\mathbf{I}). \end{aligned} \tag{A.5}$$

The following lemma asserts that $I_{LD}^{(\varepsilon)}(\mathbf{x},\mathbf{y})$ is an information measure reflecting the correlation or “linear dependence" between two vectors [17]:

Lemma 1 *Let* $\mathbf{x},\mathbf{y}$ *be random vectors with the auto-covariance matrices* $\mathbf{R_x} > 0$ *and* $\mathbf{R_y}$*, respectively, and the cross-covariance matrix* $\mathbf{R_{xy}}$*. Then,*

- $I_{LD}^{(\varepsilon)}(\mathbf{x};\mathbf{y}) \geq 0$,
- $I_{LD}^{(\varepsilon)}(\mathbf{x};\mathbf{y}) = 0$ *if and only if* $\mathbf{R_{xy}} = \mathbf{0}$*, that is,* $\mathbf{x}$ *and* $\mathbf{y}$ *are uncorrelated.*

B The weight-shared SSL setup

In all our experiments we use weight-sharing between two branches:

$$\begin{aligned} f_1(\cdot;\mathbb{W}_{DNN}^{(1)}) &= f_2(\cdot;\mathbb{W}_{DNN}^{(2)}) = f(\cdot;\mathbb{W}_{DNN}) \\ p_1(\cdot;\mathbb{W}_{P}^{(1)}) &= p_2(\cdot;\mathbb{W}_{P}^{(2)}) = p(\cdot;\mathbb{W}_{P}), \end{aligned}$$

The two-branch network in Figure 2 is equivalent to the setup illustrated in Figure A.1. The augmentation outputs $\mathcal{X}^{(1)}$ and $\mathcal{X}^{(2)}$ are input to the same DNN, i.e., $f(\cdot;\mathbb{W}_{DNN})$ whose output is input to the projection network $p(\cdot;\mathbb{W}_P)$.

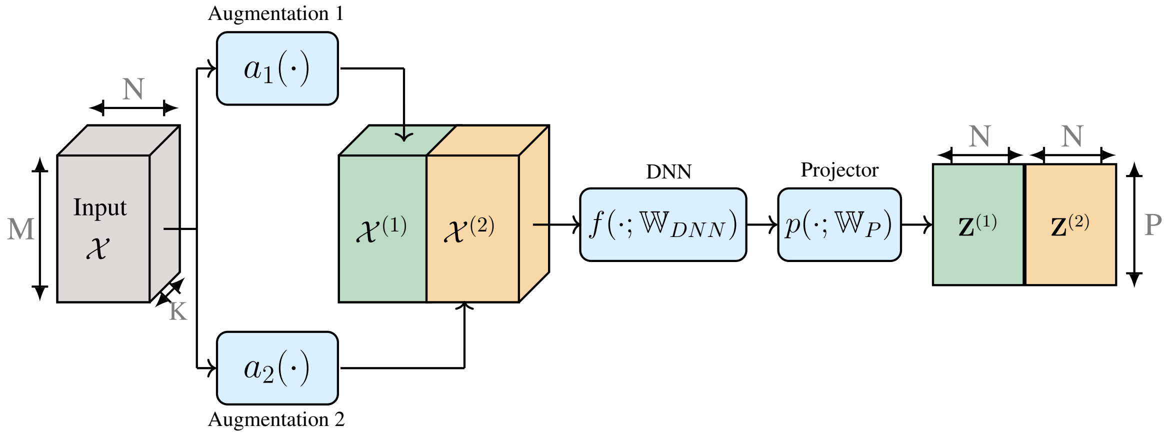

Figure A.1: Self-supervised learning set-up with weight sharing.

C Pseudocode

Algorithm 1 (next page) describes the main steps for the CorInfoMax approach in a PyTorch-style pseudocode format.

Algorithm 1: PyTorch-style pseudocode for CorInfoMax

```
# f:  encoder with projector network
# N: batch size, D: projector output dimension
# R1 and R2:  covariance matrices are initialized as identity (DXD)
# mu1 and mu2:  means vectors are initiliazed as zero (D)
# lambda:  forgetting factor, alpha:  attraction coefficient
# mse_loss:  mean squared error loss, @:  matrix multiplication
for x in loader:  # load input batch
   # random augmentations
   x1, x2 = augmentation(x)
   # projector outputs
   z1 = f(x1)
   z2 = f(x2)
   # mean estimation
   mu1_update = z1.mean(0)
   mu2_update = z2.mean(0)
   mu1 = lambda * mu1 + (1 - lambda) * mu1_update
   mu2 = lambda * mu2 + (1 - lambda) * mu2_update
   # covariance matrix estimation
   z1_hat = z1 - mu1
   z2_hat = z2 - mu2
   R1_update = (z1_hat.T @ z1_hat) / N
   R2_update = (z2_hat.T @ z2_hat) / N
   R1 = lambda * R1 + (1 - lambda) * R1_update
   R2 = lambda * R2 + (1 - lambda) * R2_update
   # loss calculation
   cor_loss = - (logdet(R1) + logdet(R2)) / D # bing-bang factor
   sim_loss = mse_loss(z1, z2) # attraction factor
   loss = cor_loss + alpha * sim_loss
   # optimization
   loss.backward()
   optimizer.step()
```

D Datasets

- The *CIFAR*-10 dataset [40] consists of 32×32 images with 10 classes. There are 5000 training images and 1000 validation images for each class.
- The *CIFAR*-100 dataset [40] consists of 32×32 images with 100 classes. There are 500 training images and 100 validation images for each class.
- The *Tiny ImageNet* dataset [41] consists of 64×64 images with 200 classes. There are 500 training images and 50 validation images for each class.
- The *ImageNet*-1*K* dataset [43] has 1281167 different sizes of images from 1000 classes as training set. Set of 50000 validation images is treated as a test dataset for evaluation purposes.
- The *ImageNet*-100 dataset [44–46] contains 100 sub-classes of the ImageNet dataset [43], which consists of images with variety of sizes. There are 1300 train images and 50 validation images for each class.
- The *COCO* dataset [42] consists of 164K images with annotations for object detection, instance segmentation, captioning, keypoint detection, and per-pixel segmentation. The training split contains 118K images, while the validation set contains 5K images. The test set contains 41K images.

Images in all these datasets have three color channels.

E Image augmentations

E.1 Augmentations during pretraining

During CorInfoMax pretraining, we use the following set of augmentations:

- Random resized cropping: cropping a random area of the input image with a scale parameter $(0.08, 1.0)$. Then resizing that cropped area to 32 × 32 for CIFAR datasets, 64 × 64 for Tiny ImageNet, and 224 × 224 for ImageNet-100 and ImageNet-1K.
- Horizontal flipping: mirroring the input image horizontally (left-right).
- Color jittering: changing the color properties of the input image. Brightness, contrast, and saturation are (uniform) randomly selected from $[max(0, 1 - \text{offset}), 1 + \text{offset}]$. Hue is selected from $[-\text{value}, \text{value}]$. The offset and value parameters used are given in Table 3.
- Grayscale: converting the RGB image to a grayscale image with three channels using $(0.2989 \times r + 0.587 \times g + 0.114 \times b)$.
- Gaussian blurring: smoothing the input image by filtering with a Gaussian kernel. The radius parameter for the kernel is selected uniformly between 0.1 and 2.0 pixels.
- Solarization: inverting image pixel values by subtracting the maximum value. We keep the result if it is above the threshold value; otherwise, replace it with the original value. The default threshold value is 128.

The spatial dimensions of images input to the encoder networks are 32 × 32 for CIFAR datasets, 64 × 64 for Tiny ImageNet, and 224 × 224 for ImageNet-100 and ImageNet-1K. All pretraining augmentation parameters are listed in Table 3.

Table 3: Augmentation parameters are used in pretraining. Aug-1 and Aug-2 refer to augmentations for each branch. Transformations are selected independently for the two branches with the given probability values. The offset and maximum values determine the interval for uniform selection.

Transformation	Aug-1	Aug-2
Random resized cropping probability	1.0	1.0
Horizontal flipping probability	0.5	0.5
Color Jitter (CJ) probability	0.8	0.8
CJ - Brightness offset	0.4	0.4
CJ - Contrast offset	0.4	0.4
CJ - Saturation offset	0.2	0.2
CJ - Hue maximum value	0.1	0.1
Grayscale probabillity	0.2	0.2
Gaussian blur probability	1.0	0.1
Solarization probability	0.0	0.2

E.2 Augmentations during linear evaluation

In the linear evaluation stage, a single transformed version of the input image is generated during both the training and the test phases.

- In the *training phase*, the random-resized-crop and horizontal-flip operations are used as augmentations as in [4–6, 25]. For the random-resized-crop operation, the target sizes are 32 × 32 for CIFAR datasets, 64 × 64 for Tiny ImageNet, and 224 × 224 for ImageNet-100 and ImageNet-1K.
- In the *test phase* of ImageNet-100 and ImageNet-1K, we apply the same preprocessing operation used for the (linear evaluation) test phase of the ImageNet dataset in [4]: input images are resized to 256 × 256 then center cropped to 224 × 224. For the other datasets, we apply a similar preprocessing by preserving the resize-crop ratio, as in [33].

E.3 Additional information about augmentation

As the last step of the augmentation process, we normalize each channel of the resulting tensors by the mean and standard deviation of that channel calculated over the whole input dataset. The mean and standard deviation normalization values for the channels are $(0.4914, 0.4822, 0.4465)$ and $(0.247, 0.243, 0.261)$ respectively, for CIFAR-10. For CIFAR-100, the corresponding normalization values are $(0.5071, 0.4865, 0.4409)$ and $(0.2673, 0.2564, 0.2762)$; for Tiny ImageNet, ImageNet-100 and ImageNet-1K: $(0.485, 0.456, 0.406)$ and $(0.229, 0.224, 0.225)$. As an interpolation method for resizing, we use bicubic interpolation.

F Hyper-parameters

To select the hyper-parameters, we tested the following values: For CIFAR datasets, we examined $\alpha = [250, 500, 1000]$, projector output dimensions [64, 128, 256], projector hidden dimensions [2048, 4096] with [2, 3] layers setting. For the Tiny ImageNet dataset, we tested $\alpha = [250, 500, 1000]$, the projector dimensions [4096-4096-128, 4096-4096-256], and learning rate = $[0.25, 0.5, 1.0]$ and the forgetting factor [0.1, 0.01]. For ImageNet-100, we found the current best parameters in the same parameters set after the Tiny ImageNet experiments.

For ImageNet-1K, we tried $\alpha = [1000, 2000, 3000]$. Using the results of previous experiments in smaller datasets, increasing the output dimension with batch size provides an increase in the test accuracy performance in a limited number of experiments. We get our reported result with the projector dimension 8192-8192-512 and a batch size of 1536.

For the covariance update expression in (7), we use the initialization $\hat{\mathbf{R}}_{\mathbf{z}^{(q)}}[0] = \mathbf{I}$ for $q = 1, 2$. We use $\boldsymbol{\mu}^{(1)}[0] = \boldsymbol{\mu}^{(1)}[0] = \mathbf{0}$ for the mean initializations. The forgetting factor is $\lambda = 0.01$. Diagonal perturbation constant for the covariance matrices is $\varepsilon = 1 \times 10^{-8}$.

Regarding all linear evaluations, we train the linear classifier for 100 epochs with a batch size of 256. We use the SGD optimizer with a momentum of 0.9, no weight decay. For all datasets except ImageNet-1K, a learning rate of 0.2 is chosen. We use the cosine decay learning rate rule as a scheduler with a minimum learning rate of 0.002. For ImageNet-1K, we use a step scheduler where the learning rate starts with 25 and reduces by a factor of 10 for each 20 epochs.

Below, we summarize the experimental details for each dataset:

F.1 CIFAR-10 experiment

For CIFAR-10, the encoder network is ResNet-18 modified for CIFAR based on the small (32×32) input image size. The modifications are: using a smaller kernel size, 3×3 instead of 7×7, in the first convolutional layer, and dropping the max-pooling layer. For the projection block, we use a 3-layer MLP network with dimensions 2048-2048-64.

We pretrain for 1000 epochs using a batch size of 512 using an SGD optimizer with a momentum of 0.9, and maximum learning rate of 0.5. For the learning rate scheduler, we utilize the cosine decay schedule after a linear warm-up for 10 epochs. During the linear warm-up period, the learning rate starts at 0.003. At the end of the training, it reaches its minimum value, which is set as 1×10^{-6}. The weight decay parameter for the optimizer is 0.0001.

F.2 CIFAR-100 experiment

For CIFAR-100, the encoder network is modified ResNet-18, as explained in the above part. For the projection network, we use a 3-layer MLP with sizes of 4096-4096-128.

We pretrain for 1000 epochs with a batch size of 512. We use SGD optimizer with a momentum of 0.9, and a maximum learning rate of 0.5. As a scheduler, we utilize cosine decay learning rate with linear warm-up for 10 epochs. The learning rate start value is 0.003 for the warm-up period. After reaching 0.5, the learning rate decays to 1×10^{-6} until the end of the training. The weight decay parameter for the optimizer is 0.0001.

F.3 Tiny ImageNet experiment

For Tiny ImageNet, the encoder network is standard ResNet-50. For the projection network, we use a 3-layer MLP with sizes of 4096-4096-128.

We pretrain our model for 800 epochs to make it more comparable with other methods in Table 1, and use a batch size of 1024. We use the SGD optimizer with a momentum of 0.9, and a maximum learning rate of 0.5. As a scheduler, we use the cosine decay learning rate with linear warm-up for 10 epochs. The learning rate start value is 0.003 for the warm-up period. After reaching 0.5, the learning rate decays to 1×10^{-6} until end of the training. The weight decay parameter for the optimizer is 0.0001.

F.4 ImageNet-100 experiments

F.4.1 ResNet-18

In this experiment for ImageNet-100, the encoder network is standard ResNet-18. We use a 3-layer MLP with sizes of 4096-4096-128 as the projection network.

We pretrain 400 epochs to make it more comparable with other methods in Table 1, using a batch size of 1024. We use the SGD optimizer with momentum of 0.9, and maximum learning rate of 1.0. As a scheduler, we use the cosine decay learning rate with linear warmup for 10 epochs. The learning rate start value is 0.003 for the warm-up period. After reaching 1.0, the learning rate decays to 0.005 until the end of the training. The weight decay parameter for the optimizer is 0.0001.

F.4.2 ResNet-50

In this experiment for ImageNet-100, the encoder network is standard ResNet-50. We use a 3-layer MLP with sizes of 4096-4096-128 as the projection network.

We pretrain 200 epochs to make it more comparable with other methods in Table 1, using a batch size of 1024. We use the SGD optimizer with a momentum of 0.9, and a maximum learning rate of 1.0. As a scheduler, we use the cosine decay learning rate with linear warmup for 10 epochs. The learning rate start value is 0.003 for the warm-up period. After reaching 1.0, the learning rate is decayed to 0.005 until end of training. The weight decay parameter for the optimizer is 0.0001.

F.5 ImageNet-1K experiments

The encoder network is standard ResNet-50 in ImageNet-1K experiments. As a projection network in pretraining, a 3-layer MLP with sizes of 8192-8192-512 with batch-normalization is utilized. Note that the increase in the number of classes, relative to the Imagenet-100 dataset, translates into an increase in the projector dimension. This further translates into an increased batch size for more accurate estimation of projector covariance matrix with larger dimensions. We pretrain 100 epochs with a batch size of 1536. We use the SGD optimizer with a momentum of 0.9, and a maximum learning rate of 0.2. As a scheduler, we use the cosine decay learning rate with linear warmup for 10 epochs. The learning rate start value is 0.003 for the warm-up period. After reaching 0.2, the learning rate is decayed to 1×10^{-6} until end of training. The weight decay parameter for the optimizer is 0.0001.

F.5.1 Semi-supervised learning

In semi-supervised learning, we fine-tune our pretrained model on ImageNet-1K. In contrast to linear evaluation, weights of the encoder also change. Fine-tuning runs 20 epochs with a batch size of 256. For fine-tuning with 1% samples of ImageNet-1K, learning rate for the encoder network is 0.005, and the learning rate for the linear classifier head is 30. For fine-tuning with 10% samples of ImageNet-1K, learning rate for the encoder network is 0.005, and the learning rate for the linear classifier head is 20. We used separate step-type learning schedulers for the header and the backbone components, where the learning rate is reduced by a factor of 10 for the header and a factor of 5 for the backbone network for every 5-epoch interval.

We used a learning rate of 0.3 with a batch size of 2048 for the VICReg pretraining[6], then used published parameters and VICReg codes[6] to evaluate the fine-tuning results.

F.6 Hyper-parameter sensitivities

We provide Top-1 test accuracy results after 1000 epochs of pretraining on the CIFAR-100 dataset for various hyper-parameter adjustments below. The results show that our approach is fairly robust to small hyper-parameter changes.

F.6.1 Attraction coefficient

Table 4: Test accuracy results for the different attraction coefficients (α) with the setup which provides best result for CIFAR-100 dataset.

Attraction coefficient (α)	125	250	500	1000	2000
Top-1 accuracy	69.04	71.19	71.34	71.61	69.96

F.6.2 Batch size

Table 5: Test accuracy results for the different batch sizes with the setup which provides best result for CIFAR-100 dataset.

Batch sizes	256	512	1024
Top-1 accuracy	70.32	71.61	69.98

F.6.3 Learning rate

Table 6: Test accuracy results for the different learning rates with the setup which provides best result for CIFAR-100 dataset.

Learning rates	0.25	0.5	1.0
Top-1 accuracy	71.32	71.61	69.95

F.6.4 Projector Output Dimension

Table 7: Test accuracy results for the different projector output dimensions with the setup which provides best result for CIFAR-100 dataset.

Projector output dimension	64	128	256
Top-1 accuracy	68.19	71.61	71.26

G Algorithm runtime results

In Section 4.3 we described the computational complexity of the CorInfoMax approach and stated that its overall effect on runtime is not significant. In this section we provide some experimental evidence.

We selected VICReg [6] for comparison. We integrated their loss function to our code to eliminate differences in implementation of methods. We ran 10 epochs with both loss functions for different projector output dimensions changing between 64 and 1024, and the results are reported in Table 8.

Table 8: Runtime results for the CIFAR-10 dataset with a batch size of 512 on a T4 Cloud GPU. Average seconds per epoch from 10 test runs is reported. The loss function of VICReg [6] is integrated to our code for comparison.

Projector Dimensions	VICReg	CorInfoMax
2048-2048-64	87.77	87.33
2048-2048-128	87.59	87.94
2048-2048-256	88.00	88.55
2048-2048-512	88.19	90.89
2048-2048-1024	89.35	102.1

Furthermore, we experiment with different batch and projector sizes in order to measure the proportion of time spent on the calculation of the log-determinant in the loss function as shown in Table 9. For these experiments, we use Imagenet-1K and ResNet-50 as the encoder on a V100 Cloud GPU on a machine 5 cores of an Intel Xeon Gold 6248 CPU. We find that the cost of the log-determinant operation is at worst around 5.5% of the computation time for the range of hyperparameters we explore. We thus conclude that the log-determinant calculation does not contribute considerably to the overall training cost.

Table 9: Proportion of time spent in log-determinant calculation for Imagenet-1K and ResNet-50 with varying batch and projector sizes on a V100 Cloud GPU. Average results from 20 test runs is reported.

Batch Size	Projector Dimension				
	64	128	256	512	1024
256	0.40%	0.56%	1.06%	2.53%	5.54%
512	0.20%	0.28%	0.53%	1.27%	2.83%

H Full comparison with solo-learn

Some of the results in Table 1 are from the solo-learn paper [49]. Table 10 shows the full results from [49] in comparison to CorInfoMax.

Table 10: Top-1 and Top-5 accuracies (%) under linear evaluation on CIFAR-10, CIFAR-100, and ImageNet-100 datasets with ResNet-18. We bold all top results that are statistically indistinguishable.

Method	CIFAR-10		CIFAR-100		ImageNet-100	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Barlow Twins	92.10	99.73	70.90	**91.91**	**80.38**	**95.28**
BYOL	92.58	99.79	70.46	**91.96**	**80.32**	**94.94**
DeepCluster V2	88.85	99.58	63.61	88.09	75.40	93.22
DINO	89.52	99.71	66.76	90.34	74.92	92.92
MoCo V2+	**92.94**	99.79	69.89	91.65	79.28	**95.50**
NNCLR	91.88	99.78	69.62	91.52	**80.16**	**95.28**
ReSSL	90.63	99.62	65.92	89.73	78.48	94.24
SimCLR	90.74	99.75	65.78	89.04	77.48	94.02
SimSiam	90.51	99.72	66.04	89.62	78.72	94.78
SwAV	89.17	99.68	64.88	88.78	74.28	92.84
VICReg	92.07	99.74	68.54	90.83	79.40	**95.06**
W-MSE	88.67	99.68	61.33	87.26	69.06	91.22
CorInfoMax	**93.18**	99.88	**71.61**	**92.40**	**80.48**	**95.46**

I Transfer learning for object detection and instance segmentation example

In this section, we outline our experiments on object detection and instance segmentation in order to explore the capability of models trained using our approach to capture fine-grained details, which are otherwise not explicitly explored in classification tasks. We fine-tune our model after 100 epochs of pretraining on ImageNet-1k using Mask R-CNN [55] (C4 Backbone) on COCO [42], following the same procedure as MoCo [52]. To have a reference point in the same exact environment, we finetune the pretrained MoCo V2 [53] checkpoint for 200 epochs on ImageNet-1K. Table 11 shows the results for both tasks. We achieve similar performance as MoCo V2. We leave further optimization, experimentation on other tasks and datasets, and further exploration of the fine-grained features learned by our model to future work.

Table 11: AP, AP50, and AP75 for object detection and instance segmentation on COCO. All models have been trained on the 2017 training split and evaluated on the 2017 validation split.

Method	COCO Detection			COCO Segmentation		
	AP_{50}	AP	AP_{75}	AP_{50}	AP	AP_{75}
MoCo V2	60.52	40.77	44.19	57.33	35.56	38.12
CorInfoMax	60.56	40.50	43.89	57.22	35.34	37.70

J Embedding visualization after pretraining

In Figure A.2, we use t-Distributed Stochastic Neighbor Embedding (t-SNE) [56] to visualize where embeddings of the test dataset are located after pretraining. We get embeddings from the output of the encoder network using our pretrained model, save them with their classification labels. Then using t-SNE, we visualize the 3D embedded space. We used the following t-SNE parameters: perplexity is 50, early exaggeration is 12, and iteration number is 1000, random initialization of embeddings is used, learning rate is 208.33. We also include a video in the supplementary material that shows the t-SNE plot with changing view angles.

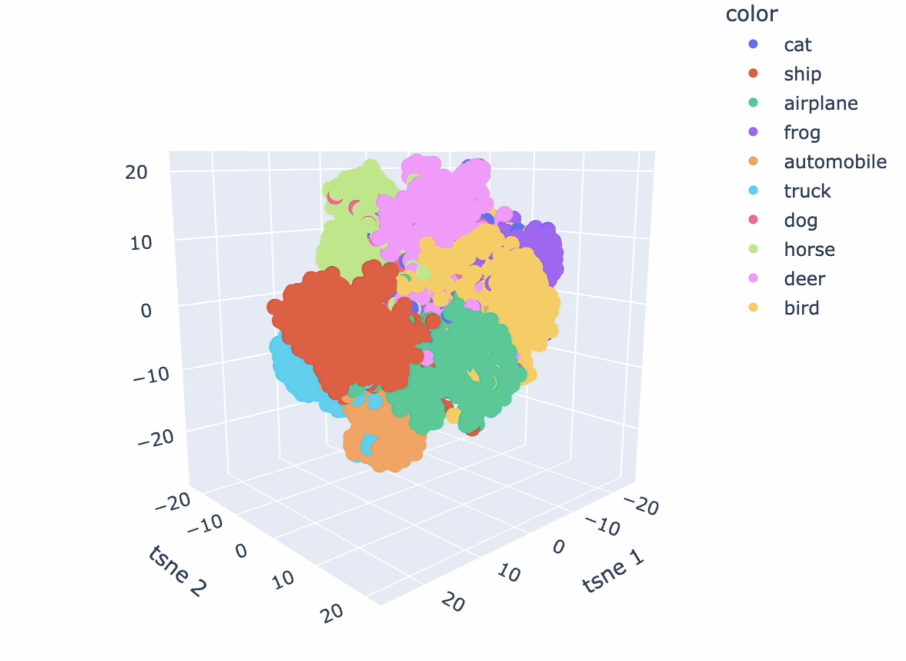

Figure A.2: t-SNE visualization of obtained embeddings of CIFAR-10 test dataset from the output of the encoder network after 1000-epoch pretraining. Each color represents one class of CIFAR-10.

K Visualization of projector covariance matrix eigenvalues

The eigenvalues of the projector vector covariance matrix, $\hat{\mathbf{R}}_{\mathbf{z}^{(1)}}$ reflect the effective use of the embedding space. Due to the existence of the "big-bang factor", $\log\det(\hat{\mathbf{R}}_{\mathbf{z}^{(1)}}[l] + \varepsilon\mathbf{I})$, in (6), we expect that no dimensional collapse occurs (with very small ε), and eigenvalues take significant values. To confirm this expectation, we performed the following experiment: for the CIFAR-10 dataset, we ran the contrastive SimCLR algorithm and obtained its projector covariance matrix after training. Similarly, we naturally obtained the projector covariance matrix for the CorInfoMax approach. For both algorithms, we used a projector dimension of 128. Figure A.3 compares the sorted eigenvalues of both covariances. As can be observed from this figure, the effective embedding space dimension for the SimCLR algorithm is small, most of the energy is concentrated at about the first 30 eigenvalues. Furthermore, the smallest 9-eigenvalues are equal to 0, within numerical precision, indicating a dimensional collapse. On the contrary, the eigenvalues for the CorInfoMax algorithm are significant for all dimensions, hinting at the effective use of the embedding space.

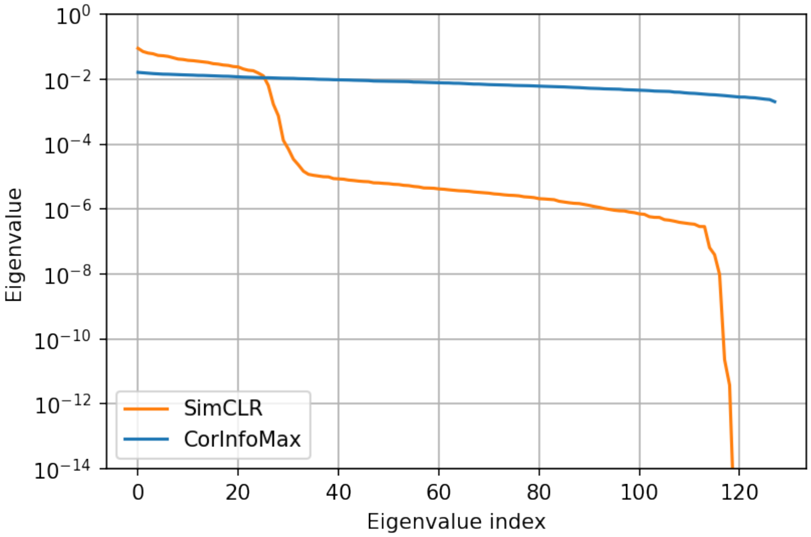

Figure A.3: Comparison of the sorted eigenvalues of the projector vector covariance matrix $\hat{\mathbf{R}}_{\mathbf{z}^{(1)}}$ for CorInfoMax and SimCLR algorithms, for CIFAR-10 dataset and projector dimension of 128.

L Visualization of LD-mutual information evolution during pretraining

As the CorInfoMax objective function is derived from the LDMI measure, it would be interesting to observe its evolution in relation to algorithm epochs and (online) test accuracy. For this purpose, we performed two experiments with the CIFAR-10 and CIFAR-100 datasets, where we used the CorInfoMax loss function, but recorded the Training-LDMI values (based on the covariance estimates using (7) during training) and the test accuracy values. Figure A.4 shows the progress of the Training-LDMI and the test accuracy curves on the same graph for the CIFAR-10 dataset.

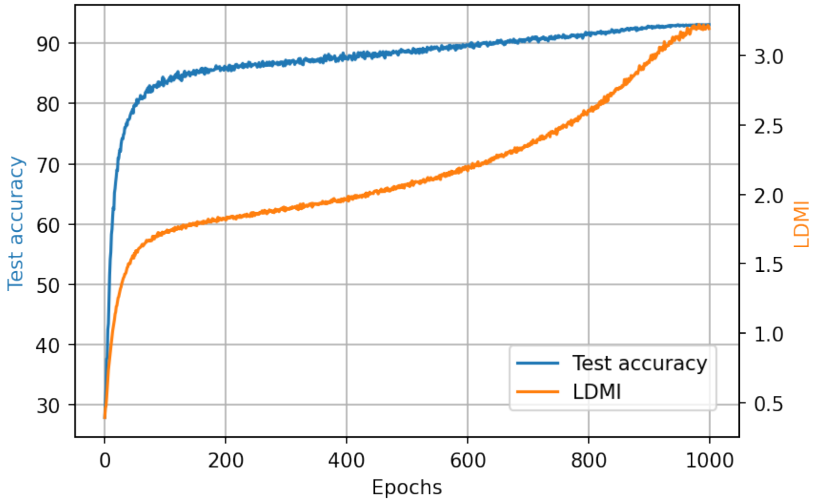

Figure A.4: Evolution of the LDMI measure and the test accuracy for the CIFAR-10 dataset as a function of the CorInfoMax algorithm epochs.

Similarly, Figure A.5 shows the progress of the Training-LDMI and the test accuracy curves on the same graph for the CIFAR-100 dataset.

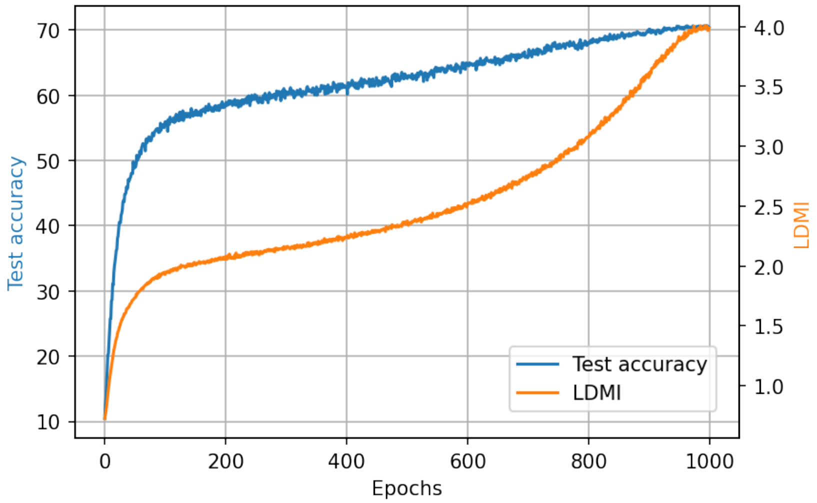

Figure A.5: Evolution of the LDMI measure and the test accuracy for the CIFAR-100 dataset as a function of the CorInfoMax algorithm epochs.

Both figures confirm that the Training-LDMI measure and the test accuracy increase together almost monotonically, ignoring the small variations.

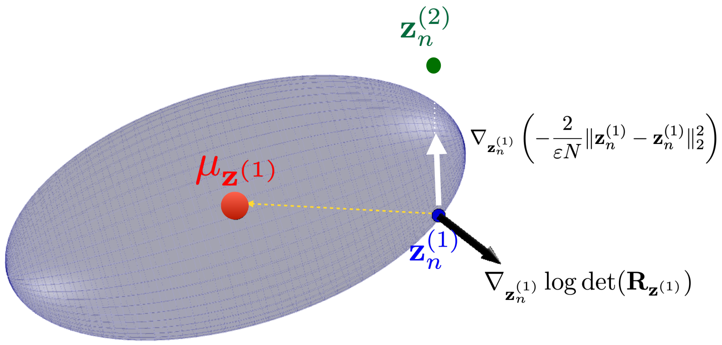
$\mathbf{z}_n^{(2)}$
$\nabla_{\mathbf{z}_n^{(1)}}\left(-\frac{2}{\varepsilon N}\|\mathbf{z}_n^{(1)}-\mathbf{z}_n^{(1)}\|_2^2\right)$
$\boldsymbol{\mu}_{\mathbf{z}^{(1)}}$
$\mathbf{z}_n^{(1)}$
$\nabla_{\mathbf{z}_n^{(1)}}\log\det(\mathbf{R}_{\mathbf{z}^{(1)}})$

Figure A.6: Gradients of the objective function for CorInfoMax: The ellipsoid is the level surface of the quadratic function $q(\mathbf{z})$ in (A.9) containing $\mathbf{z}_n^{(1)}$, one of the projector-1 output samples, the black arrow represents the gradient of $\log\det(\mathbf{R}_{\mathbf{z}^{(1)}})$ with respect to $\mathbf{z}_n^{(1)}$, the white arrow represents the gradient of $\frac{2}{\varepsilon N}\|\mathbf{z}_n^{(1)}-\mathbf{z}_n^{(2)}\|_2^2$ with respect to $\mathbf{z}_n^{(1)}$.

M Supplementary notes on CorInfoMax criterion

M.1 Gradients of the CorInfoMax objective

Let $\mathbf{z}_n^{(q)}$ represent the n^{th} sample of the q^{th} branch projector output for the current batch, where $n \in \{1, \ldots, N\}$ and $q = 1, 2$. We provide the gradient expressions of the CorInfoMax objective (6) with respect to the projector outputs, which are backpropagated to the train the encoder networks.

For the first term on the right side of (6), we can write

$$\nabla_{\mathbf{z}_n^{(1)}}\log\det(\hat{\mathbf{R}}_{\mathbf{z}^{(1)}}[l]+\varepsilon\mathbf{I})=\frac{(1-\lambda)}{N}(\hat{\mathbf{R}}_{\mathbf{z}^{(1)}}[l]+\varepsilon\mathbf{I})^{-1}(\mathbf{z}_n^{(1)}-\boldsymbol{\mu}_{\mathbf{z}^{(1)}}[l]), \tag{A.6}$$

and,

$$\nabla_{\mathbf{z}_n^{(2)}}\log\det(\hat{\mathbf{R}}_{\mathbf{z}^{(1)}}[l]+\varepsilon\mathbf{I})=0.$$

Similarly, for the second term on the right side of (6), we can write

$$\nabla_{\mathbf{z}_n^{(2)}}\log\det(\hat{\mathbf{R}}_{\mathbf{z}^{(2)}}[l]+\varepsilon\mathbf{I})=\frac{(1-\lambda)}{N}(\hat{\mathbf{R}}_{\mathbf{z}^{(2)}}[l]+\varepsilon\mathbf{I})^{-1}(\mathbf{z}_n^{(2)}-\boldsymbol{\mu}_{\mathbf{z}^{(2)}}[l]),$$

and,

$$\nabla_{\mathbf{z}_n^{(1)}}\log\det(\hat{\mathbf{R}}_{\mathbf{z}^{(2)}}[l]+\varepsilon\mathbf{I})=0.$$

Finally, for the rightmost term of (6), which is the Euclidian distance based loss, we can write

$$\nabla_{\mathbf{z}_n^{(1)}}\left(-\frac{2}{\varepsilon N}\|\mathbf{Z}^{(1)}[l]-\mathbf{Z}^{(2)}[l]\|_F^2\right)=\frac{4}{\varepsilon N}(\mathbf{z}_n^{(2)}-\mathbf{z}_n^{(1)}), \tag{A.7}$$

and

$$\nabla_{\mathbf{z}_n^{(2)}}\left(-\frac{2}{\varepsilon N}\|\mathbf{Z}^{(1)}[l]-\mathbf{Z}^{(2)}[l]\|_F^2\right)=\frac{4}{\varepsilon N}(\mathbf{z}_n^{(1)}-\mathbf{z}_n^{(2)}). \tag{A.8}$$

Inspecting the gradient expressions with respect to $\mathbf{z}_n^{(1)}$: The vector in (A.6) is the surface normal of the level set of the quadratic function

$$q(\mathbf{z})=\frac{(1-\lambda)}{N}(\mathbf{z}-\boldsymbol{\mu}_{\mathbf{z}^{(1)}}[l])^T(\hat{\mathbf{R}}_{\mathbf{z}^{(1)}}[l]+\varepsilon\mathbf{I})^{-1}(\mathbf{z}-\boldsymbol{\mu}_{\mathbf{z}^{(1)}}[l]), \tag{A.9}$$

at $\mathbf{z} = \mathbf{z}_n^{(1)}$, which is illustrated by the black vector in Figure A.6. Since $(\hat{\mathbf{R}}_{\mathbf{z}^{(1)}}[l] + \varepsilon\mathbf{I})^{-1}$ is positive definite, the inner product

$$\begin{aligned}&\langle \nabla_{\mathbf{z}_n^{(1)}} \log\det(\hat{\mathbf{R}}_{\mathbf{z}^{(1)}}[l] + \varepsilon\mathbf{I}), \boldsymbol{\mu}_{\mathbf{z}^{(1)}}[l] - \mathbf{z}_n^{(1)} \rangle \\ &= -\frac{(1-\lambda)}{N}(\mathbf{z}_n^{(1)} - \boldsymbol{\mu}_{\mathbf{z}^{(1)}}[l])^T (\hat{\mathbf{R}}_{\mathbf{z}^{(1)}}[l] + \varepsilon\mathbf{I})^{-1}(\mathbf{z}_n^{(1)} - \boldsymbol{\mu}_{\mathbf{z}^{(1)}}[l])\end{aligned}$$

is negative, which implies that the vector pointing from $\mathbf{z}_n^{(1)}$ towards the center $\mu_{z^{(1)}}[l]$, the yellow dashed arrow in Figure A.6, makes an obtuse angle with the gradient $\nabla_{\mathbf{z}_n^{(1)}} \log\det(\hat{\mathbf{R}}_{\mathbf{z}^{(1)}}[l] + \varepsilon\mathbf{I})$. Therefore, the gradient in (A.6) is pointing away from the center of the ellipsoid, encouraging the expansion.

On the other hand, the gradient expressions in (A.7) and (A.8) correspond to force pulling the positive samples $\mathbf{z}_n^{(1)}$ and $\mathbf{z}_n^{(2)}$ towards each other as indicated by the white arrow.

A video animation of sample points moving under these gradients is provided in the supplementary material.