Acknowledgements. XC would like to thank Kaiming He on helpful discussions through this project. XW would like to thank Yutong Bai on helpful discussions through this project.

A. Cross-Image Variance

In this section, we show evidence with our MoCo v2 baseline that cross-image variance quickly converges to a constant that only depends on the encoding dimension d. This is through a monitor installed on the output encodings during training. Specifically, for each iteration, we compute the variance of the output ℓ_2-normalized vectors from the source encoder along the *batch* axis and average them over the *channel* axis. Since each training batch contains different images rather than different views of the same image, the resulting value reflects the cross-image variance. Three encoding dimensions, $d\in\{64, 128, 256\}$ are experimented, and their variances during the 100-epoch training process are separately recorded in Fig. 4.

From the plot, we find that all the variances quickly and separately converge to $1/d$. For example, when the encoding dimension d is 128 (default), the variance converges to $1/128$; when d is 64, it converges to $1/64$. The same observations are made regardless of other designs for the encoder (*e.g.*, MultiCrop or SyncBN). We believe it is a natural outcome of Siamese representation learning which generally encourages *uniformity* [10, 35] – encodings of different images distribute uniformly on the unit hypersphere. Therefore, cross-image variance is deemed *not* an ideal reference to distinguish designs. Instead, we use intra-image variance which has a much smaller magnitude ($\times 10^{-4}$), but carries useful signals to tell different designs apart (see Fig. 2).

B. ScaleMix

The goal of ScaleMix is to generate a new view $\mathbf{v}^s$ by combining two random sampled views of the same size (height H and width W): $\mathbf{v}^1$ and $\mathbf{v}^2$. The generated new view is treated as a normal view of the input image $\mathbf{x}$ and used for Siamese learning. Specifically, following the protocol of [29], we define the combining operation as:

$$\mathbf{v}^s = M \cdot \mathbf{v}^1 + (1 - M) \cdot \mathbf{v}^2,$$

where $M\in\{0, 1\}^{H\times W}$ denotes a binary mask indicating where to use pixels from which view, and $\cdot$ is an element-wise multiplication. Note that different from other mixing operations [29, 45], we do not mix *outputs* as both views are from the same image.

The binary values in M are determined by bounding box coordinates $B = (x, y, w, h)$, where (x, y) is the box center, and (w, h) is the box size. Given B, its corresponding region in M is set to all 0 and otherwise all 1. Intuitively, this means the region B in $\mathbf{v}^1$ is removed and filled with the patch cropped from B of $\mathbf{v}^2$.

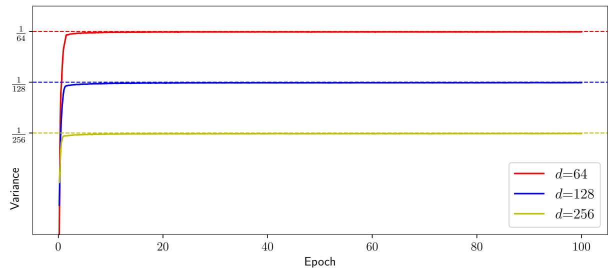

Figure 4. **Cross-image variance** tracked during the 100-epoch training process for our MoCo v2 baseline, with three encoding dimension options: $d\in\{64, 128, 256\}$. All of them quickly converge to $1/d$ (dotted lines).

The box coordinates B are randomly sampled. We keep the aspect ratio of B *fixed* and the same as the input views, and only vary the size of the box according to a random variable λ uniformly drawn from $(0, 1)$: $w = W\sqrt{\lambda}$, $h = H\sqrt{\lambda}$. Box centers (x, y) are again uniformly sampled.

C. Detailed Theoretical Analysis

Given the outputs: $\mathbf{z}$ from the source encoder and $\mathbf{z}'$ from the target encoder (prime $'$ indicates target-related), the InfoNCE [28] loss used by MoCo is defined as:

$$\mathcal{L} := -\frac{1}{N}\sum_{i=1}^{N} \log \frac{\exp(\mathbf{S}_{ii'}/\tau)}{\epsilon \exp(\mathbf{S}_{ii'}/\tau) + \sum_{j\neq i} \exp(\mathbf{S}_{ij'}/\tau)}, \tag{3}$$

where N is batch size, τ is temperature, $\mathbf{S}_{ii'} = \mathbf{z}_i^\top \mathbf{z}'_i$ and $\mathbf{S}_{ij'} = \mathbf{z}_i^\top \mathbf{z}'_j$ are pairwise similarities between source and target encodings. We additionally introduce the parameter ϵ that controls the weight for the positive term in the denominator, where for standard loss $\epsilon = 1$.

For MoCo, only the source encoder receives gradient, and we take derivatives only for $\mathbf{z}_i$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}_i} = \frac{1}{\tau}\sum_{j\neq i} \alpha_{ii'j'}(\mathbf{z}'_j - \mathbf{z}'_i), \tag{4}$$

where

$$\alpha_{ii'j'} = \frac{\exp(\mathbf{S}_{ij'}/\tau - \mathbf{S}_{ii'}/\tau)}{\epsilon + \sum_{k\neq i} \exp(\mathbf{S}_{ik'}/\tau - \mathbf{S}_{ii'}/\tau)}. \tag{5}$$

For the simplified case where $\epsilon = 0$ [42], we can have:

$$\alpha_{ii'j'} = \alpha_{ij'} = \frac{\exp(\mathbf{S}_{ij'}/\tau)}{\sum_{k\neq i} \exp(\mathbf{S}_{ik'}/\tau)}, \tag{6}$$

which is independent of target encoding $\mathbf{z}'_i$.

Now, let's consider the last linear layer immediately before $\mathbf{z}$ as an example for analysis. Let $\mathbf{f}$ be the input features of this layer, W be its weight matrix (so $\mathbf{z}{=}W\mathbf{f}$ and we do not consider ℓ_2 normalization applied to $\mathbf{z}$). In this case, we can write down the dynamics of the source weight W based on the gradient descent rule:

$$\begin{aligned} \dot{W} &:= -\frac{\partial \mathcal{L}}{\partial W} = -\frac{1}{N}\sum_{i=1}^{N}\frac{\partial \mathcal{L}}{\partial \mathbf{z}_i}\mathbf{f}_i^\top && (7)\\ &= -\frac{1}{\tau N}\sum_{i=1}^{N}\sum_{j\neq i}\alpha_{ij'}(\mathbf{z}'_j - \mathbf{z}'_i)\mathbf{f}_i^\top, && (8)\end{aligned}$$

where $\dot{W}$ is a simplified notion of the change to w.r.t. W following gradient decent. Since both $\mathbf{z}'_j$ and $\mathbf{z}'_i$ come from the target encoder weight W', we have $\mathbf{z}'_j{=}W'\mathbf{f}'_j$ and $\mathbf{z}'_i{=}W'\mathbf{f}'_i$ and thus:

$$\dot{W} = -W'\frac{1}{\tau N}\sum_{i=1}^{N}\sum_{j\neq i}\alpha_{ij'}(\mathbf{f}'_j - \mathbf{f}'_i)\mathbf{f}_i^\top \qquad (9)$$

We define $\bar{\mathbf{f}}{:=}\mathbb{E}[\mathbf{f}]$ to be the mean of the input feature and $\Sigma_{\mathbf{f}}{:=}\mathbb{V}[\mathbf{f}]$ to be the co-variance matrix of the input feature $\mathbf{f}$, where $\mathbb{E}[\cdot]$ computes expectation and $\mathbb{V}[\cdot]$ outputs variance. These two quantities will be used later.

Now let's consider how intra-image variance in both target and source sides affect training. To reach a clear conclusion, we now make two assumptions.

Assumption 1: additive noise. We can model the intra-image variance as *additive noise*. Specifically, let $\tilde{\mathbf{f}}$ be the feature corresponding to the original image, we can assume:

- $\mathbf{f}_i{=}\tilde{\mathbf{f}}_i{+}\mathbf{e}_i$. That is, the input feature of the last layer $\mathbf{f}_i$ receives source noise $\mathbf{e}_i$ with $\mathbb{E}[\mathbf{e}_i]{=}\bar{\mathbf{e}}$ and $\mathbb{V}[\mathbf{e}_i]{=}\Sigma$;
- $\mathbf{f}'_j{=}\tilde{\mathbf{f}}_j{+}\mathbf{e}'_j$. That is, the input feature $\mathbf{f}'_j$ receives target noise $\mathbf{e}'_j$ with $\mathbb{E}[\mathbf{e}'_j]{=}\bar{\mathbf{e}}'$ and $\mathbb{V}[\mathbf{e}'_j]{=}\Sigma'$. Note that for the feature of a different image $\mathbf{f}'_i$, it also undergoes the same process on the target side and thus we have $\mathbf{f}'_i{=}\tilde{\mathbf{f}}_i{+}\mathbf{e}'_i$.

Note that the noise is not necessarily zero mean-ed. Since the augmentations of $\mathbf{f}_i$ and $\mathbf{f}'_i$ are independent, $\mathbf{e}_i$ and $\mathbf{e}'_i$ are independent of each other: $\mathbb{P}(\mathbf{e}_i, \mathbf{e}'_i){=}\mathbb{P}(\mathbf{e}_i)\mathbb{P}(\mathbf{e}'_i)$. Same for $\mathbf{e}_i$ and $\mathbf{e}_j$ where $i{\neq}j$.

Assumption 2: all $\alpha_{ij'}$ are constant and independent of f. Alternatively, if we consider the quadratic loss (*i.e.*, $\mathcal{L}_q{=}\sum_{j\neq i}(\mathbf{S}_{ij'}{-}\mathbf{S}_{ii'})$), then all $\alpha_{ij'}$ are constant and this assumption holds true. For InfoNCE this may not hold, and we leverage this assumption for simplicity of derivations.

Under these two assumptions, we now compute $\mathbb{E}_{\mathbf{f}}[\dot{W}]$, the *expectation* of the weight gradient over *input feature* $\mathbf{f}$ of the last layer. This gets rid of inter-image variance, and focuses on intra-image variance only:

$$\mathbb{E}_{\mathbf{f}}[\dot{W}] = \frac{1}{\tau}W'(\Sigma_{\mathbf{f}} - R). \qquad (10)$$

Here the residual term R is as follows:

$$R := -\frac{1}{N}\sum_{i=1}^{N}\hat{\mathbf{e}}'_i(\bar{\mathbf{f}} + \mathbf{e}_i)^\top, \qquad (11)$$

where $\hat{\mathbf{e}}'_i{:=}\sum_{j\neq i}\alpha_{ij'}\mathbf{e}'_j{-}\mathbf{e}'_i$ is also a random variable which is a weighted sum of $\mathbf{e}'_j$ and $\mathbf{e}'_i$.

From the definition (Eq. (5)), we have $\sum_{j\neq i}\alpha_{ij'}{=}1$. $\mathbf{e}'_j$ and $\mathbf{e}'_i$ are independent. Therefore we can compute the mean and variance of $\hat{\mathbf{e}}'_i$ as:

$$\begin{aligned} \mathbb{E}[\hat{\mathbf{e}}'_i] &= 0, && (12)\\ \hat{\Sigma}'_i := \mathbb{V}[\hat{\mathbf{e}}'_i] &= (1 + \sum_{j\neq i}\alpha_{ij'}^2)\Sigma'. && (13)\end{aligned}$$

Now for the residual term R, we also have $\mathbb{E}_{\mathbf{e}}[R]{=}0$. Therefore, the full expectation for $\dot{W}$ can be written as:

$$\mathbb{E}[\dot{W}] := \mathbb{E}_{\mathbf{e}}[\mathbb{E}_{\mathbf{f}}[\dot{W}]] = \frac{1}{\tau}W'\Sigma_{\mathbf{f}}. \qquad (14)$$

This means the source weight will grow along the direction that maximizes the distance between different images. More precisely, it grows along the eigenvector that corresponds to the maximal eigenvalue of $\Sigma_{\mathbf{f}}$.

Now we can check the influence of *intra-image* variance from source and target encoders. The influence can be characterized by the term $\mathbb{V}_{\mathbf{e}}[\mathbb{E}_{\mathbf{f}}[\dot{W}]]$. For simplicity, we can compute $\mathbb{V}_{\mathbf{e}}[\mathbb{E}_{\mathbf{f}}[\mathrm{tr}(R)]]$ – *i.e.* the variance on the trace of R, since $\Sigma_{\mathbf{f}}$ remains constant for intra-image variance.

Leveraging the independence of $\{\hat{\mathbf{e}}'_i, \mathbf{e}_i\}$ among different images, we can arrive at:

$$\mathbb{V}_{\mathbf{e}}[\mathbb{E}_{\mathbf{f}}[\mathrm{tr}(R)]] = \mathrm{tr}\left[\hat{\Sigma}'(\bar{\mathbf{f}}\bar{\mathbf{f}}^\top + \bar{\mathbf{e}}\bar{\mathbf{e}}^\top + \Sigma)\right], \qquad (15)$$

where $\hat{\Sigma}'{:=}\frac{1}{N}\sum_{i=1}^{N}\hat{\Sigma}'_i$ is the mean of all variances of $\hat{\mathbf{e}}'_i$.

From Eq. (15) we can notice that: i) if there is large magnitude of source feature mean $\bar{\mathbf{f}}$ and/or source noise mean $\bar{\mathbf{e}}$, then the variance will be large; ii) this effect will be *magnified* with more target-side variance (*i.e.*, larger eigenvalues of Σ' and thus $\hat{\Sigma}'$), *without* affecting the average gradient; iii) large magnitude of feature mean and/or noise mean on the target side does *not* influence the variance. This *asymmetry* between source and target suggests that the training procedure an be negatively affected if the target variance is too large, coupled by $\bar{\mathbf{f}}\bar{\mathbf{f}}^\top$ and $\bar{\mathbf{e}}\bar{\mathbf{e}}^\top$ in Eq. (15).

The intuition why there is such an asymmetry is the following: in Eq. (9), while the target side has a subtraction $\mathbf{f}'_j - \mathbf{f}'_i$ which cancels out the mean, the source side $\mathbf{f}_i$ doesn't. This leads to the mean values being kept on the source side which couples with the target variance, whereas no mean values from the target side are kept.

Therefore, we can infer that higher variance on the target side is less necessary compared to the source side – it will incur more instability during training without affecting the mean of gradients.

D. More Implementation Details

MultiCrop. Our MultiCrop recipe largely follows the work of SwAV [6]. Specifically, 224-sized crops are sampled with a scale range of $(0.14, 1)$, and 96-sized small crops are sampled from $(0.05, 0.14)$. We use m=6 small crops by default, and each is forwarded separately with the encoder. When applied to one encoder, all $(1{+}6){=}7$ encodings are compared against the single encoding from the other side; when applied jointly, $(7{\times}2){=}14$ encodings are paired by crop size to compute loss terms. Unlike the practice in SwAV, no loss symmetrization is employed and the 6 losses from small crops are averaged before adding to the standard loss. When target encoder is involved in MultiCrop, we also create a separate memory bank [19] dedicated to small crops, updated with 1 out of the 6 crops.

AsymAug. For StrongerAug, we use additional augmentations from RandAug [12], same as [37]. For WeakerAug, we simply remove all the color- and blur-related augmentations and only keep geometric ones in the MoCo v2 recipe. This leaves us with random resized cropping and flipping.

MeanEnc. Deviating from MultiCrop, augmentations used for computing the mean are forwarded *jointly* through the encoder thanks to the uniform size of 224×224. Joint forwarding enlarges the batch size in BN, which further reduces the variance. The output encodings are averaged before ℓ_2 normalization.

Other frameworks. Different from MoCo v2 which uses shuffle BN [19] across 8 GPUs, all the frameworks studied in Sec. 6.2 use SyncBN by default. Therefore, when applying AsymBN to them, we keep the target encoder untouched and change the BNs in the source encoder instead. To *minimize* the impact from the number of GPU devices (*e.g*., MoCo v3 uses 16 GPUs to fit a batch size of 4096 for ResNet; whereas for ViT it uses 32 GPUs), we always divide the full batch into 8 *groups* and the normalization is performed within each group – this mimics the per-device BN operation in MoCo v2 while being more general.

Moreover, for MoCo v2 we only convert the single BN in the target projector to SyncBN. This has minimal influence on efficiency as SyncBN can be expensive and converting all of them (including ones in the encoder) can significantly slow down training. Now since we are converting SyncBN *back*, we choose to convert *all* BNs in the source encoder whenever possible to reduce inter-device communications for efficiency purposes.

More recent frameworks [11, 44] adopt the asymmetric augmentation recipe in BYOL [18], in such cases, we use one composition for source and the other for target half the time during pre-training, and swap them in the other half.

To have a fair comparison with frameworks pre-trained for 100 epochs, we optionally train 2× as long when the default loss is symmetrized and ours is asymmetric.

Unless otherwise specified, we follow the same design choices in MoCo v2 when applying ScaleMix and MeanEnc to other frameworks. In addition, there are subtleties associated with each individual framework listed below:

- **MoCo v3** [11]. Since MoCo v3 also employs an additional predictor on the source side, we involve both the predictor and the backbone when applying AsymBN.
- **SimCLR** [8]. The original SimCLR uses $2{\times}N{-}2$ negative examples for contrastive learning [8], which includes all the other images in the same batch, multiplied by 2 for the two augmentations per image. After converting to the asymmetric version, we only use $N{-}1$ negative samples – same as in MoCo v3 – and it causes a gap. We find a simple *change* of InfoNCE [28] temperature from 0.1 to 0.2 can roughly compensate for this gap. For AsymBN, we convert all the BNs in the source encoder, not just the ones in the projector. For ScaleMix, we apply this augmentation half the time – we empirically find applying ScaleMix all the time will cause a considerable drop in performance compared to the asymmetric baseline, for reasons yet to be understood.
- **BYOL** [18]. BYOL initiated the additional predictor which also has BNs. We convert all the BNs in the source encoder when AsymBN is used, not just ones in the projector.
- **SimSiam** [10]. Additional predictor is again used in SimSiam and plays an important role in collapse prevention. We convert all the BNs in the source encoder after the conversion to an asymmetric version.
- **Barlow Twins** [44]. This is a fully symmetric framework and *no* loss symmetrization is used by default. Therefore, we also pre-train the asymmetric version for 100 epochs, not 2× as long. Same as SimCLR, ScaleMix is applied with half the frequency. All the encoder BNs are converted when AsymBN is used.

ViT backbone. MoCo v3 [11] with its default hyper-parameters for ViT backbone is used. ViT as a backbone does *not* have BN. Therefore we convert BNs in the projector and predictor when using AsymBN.

Transfer learning. We follow the linear probing protocol to evaluate our model on transfer learning tasks. Different from ImageNet, we use SGD optimizer with momentum 0.9 and weight decay 0 for training. The learning rate is adjusted via grid search on the validation set, and the final results are reported on the test set. All models are trained for 100 epochs, with a half-cycle cosine decaying schedule for learning rate.