Simplifying DINO by Coding Rate Regularization

Ziyang Wu[1], Jingyuan Zhang[2], Druv Pai[1], Yi Ma[1,3]
[1]UC Berkeley, [2]Transcengram, [3]Hong Kong University
zywu@berkeley.edu

DINO and DINOv2 are two methods widely used for learning representations of unlabeled imagery data at large scales. Their learned representations often give state-of-the-art performance for downstream tasks, such as image classification and segmentation. However, their training pipelines are highly complex and unstable, so it is difficult to improve and adapt them to new domains. In particular, they employ many empirically motivated design choices and carefully tuned hyperparameters to ensure that the representations do not collapse. In this work, we posit that we can remove many such-motivated idiosyncracies in the pre-training pipelines, and only need to add an explicit coding rate term in the loss function to avoid collapse of the representations. As a result, we obtain highly simplified variants of the DINO and DINOv2 model families, which we call SimDINO and SimDINOv2, respectively. Notably, their training pipelines are more robust to different design choices, such as network architecture and hyperparameters, and they learn even higher-quality representations, measured by performance on downstream tasks, offering a Pareto improvement over the corresponding DINO and DINOv2 model families. This work highlights the potential of using simplifying design principles to improve empirical outcomes in deep learning.

1. Introduction

Self-supervised learning (SSL) is the toolkit of choice to learn representations for large datasets of unlabeled images [1–12], captioned images [13], videos [14], and text [15–18], among other modalities. In the context of image SSL, there are two main approaches: *reconstructive* [10], where the goal is to reconstruct some function of the true image data from a "view", i.e., corruption or augmentation, and *contrastive* [1], where the goal is, for each image, to have the features of different views of the image all be close, and features of views of different images be far.

Within contrastive SSL, a key challenge lies in preventing *representation collapse*, where models learn trivial solutions that map all inputs to the same output. One common approach to address this is through the use of *negative samples*, which explicitly encourages representations of different images to be dissimilar. Thus far, the success of using negative samples depends on having a large batch size [3, 5], which poses computational challenges at scale. Methods which attempt to avoid this bottleneck by using negative samples in more implicit and indirect ways to avoid collapse [8] can cope with smaller batch sizes, but often require training pipelines with numerous carefully tuned components and hyperparameters to avoid collapse, making them difficult to adapt and improve.

The state-of-the-art for image SSL is generally considered to be the DINOv2 model family [12]. It is built on the DINO model family [8]. Both classes of models are trained using contrastive SSL and thus run into the representation collapse issue. While DINOv2 explicitly and directly uses negative samples to avoid collapse, it inherits much of its training pipeline from DINO, which uses negative samples more indirectly. As such, *both* model families' training pipelines are highly complex and unstable, requiring many tweaks and careful hyperparameter selection in order for the training to converge for a given architecture. Despite this capriciousness, the trained models' representations are highly useful for downstream tasks, and are widely used in different contexts [19, 20].

Our contributions. In this work, we remove many tweaks and hyperparameters from the DINO and DINOv2 training pipelines, replacing them with a term in the objective which explicitly uses

Preprint. Work in progress.

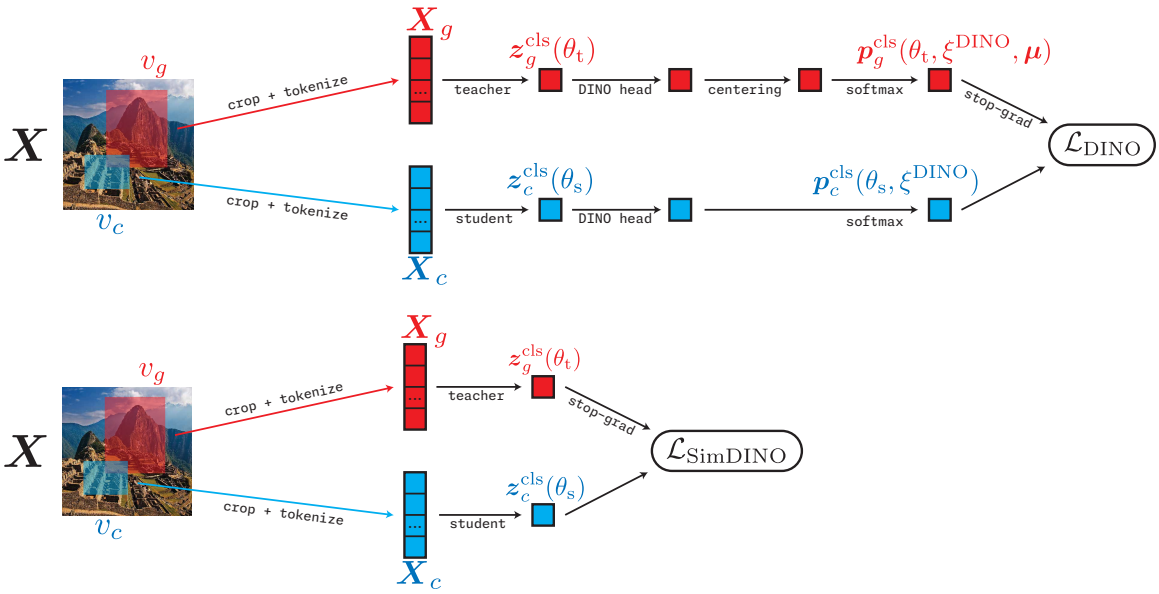

Figure 1: **The DINO pipeline (*above*) and SimDINO pipeline (*below*)**, with the selected teacher (global) view v_g and the student (in this case local) view v_c. Here, an input image is turned into patches. Then a global view v_g and another view v_c are randomly sampled. The global view is pushed through the teacher encoder, while the other view is pushed through the student encoder. *Only in the DINO pipeline, we apply expensive post-processing operations, such as a dimension-increasing linear layer and a high-dimensional softmax, to feature vectors.*

negative samples. We show empirically that this term, which involves the *total coding rate* regularizer [21–23], enables a simple, robust, and computationally efficient training pipeline, which in particular does not require large batch sizes. We show that the resulting models, named SimDINO and SimDINOv2, learn representations which have the same or higher state-of-the-art performance as those learned by DINO and DINOv2 across a variety of downstream tasks. Our work underscores the value of understanding and simplifying pipelines to improving performance in vision SSL.

1.1. Notation

Let $C, H, W, D, N, d \geq 1$ be positive integers. Let the space of finite sequences of vectors in $\mathbb{R}^D$ be denoted as $\mathbb{R}^{D\times *} = \bigcup_{T=1}^{\infty} \mathbb{R}^{D\times T}$. Our data will be images $\boldsymbol{X} \in \mathbb{R}^{C\times H\times W}$. We will consider different augmentations, or *views*, of the input data $\boldsymbol{X}$, such as rotations or crops of imagery data; we can represent a view as a function $v\colon \mathbb{R}^{C\times H\times W} \to \mathbb{R}^{D\times N_v}$ where N_v is the number of tokens in the view.

Let $\mathbb{S}_{d-1} \subseteq \mathbb{R}^d$ be the $(d-1)$-dimensional ℓ^2-sphere. For the purpose of representation learning, we will consider an *encoder* neural network parameterized by weights $\theta \in \Theta$, as a function $f_\theta\colon \mathbb{R}^{D\times *} \to \mathbb{S}_{d-1} \times \mathbb{S}_{d-1}^N$. We can factor $f_\theta = (f_\theta^{\text{cls}}, f_\theta^{\text{patch}})$ where $f_\theta^{\text{cls}}\colon \mathbb{R}^{D\times *} \to \mathbb{S}^{d-1}$ outputs the so-called *class token feature* (i.e., an aggregate representation of the input data) and $f_\theta^{\text{patch}}\colon \mathbb{R}^{D\times *} \to \mathbb{S}_{d-1}^N$ outputs the *patch tokens' features* (i.e., a patch-based representation of the input data). The network is implemented by a Vision Transformer [24, 25] with appended multi-layer perceptrons (MLPs) to post-process each aggregate and patch-based feature followed by an ℓ^2-normalization.

2. Simplifying DINO and DINOv2

2.1. Recap of the Original DINO Pipeline

The goal of DINO is to learn an aggregate representation of the input image which contains information about large-scale semantics of the input (e.g., the locations and properties of different objects in the image). They do this via a pre-training pipeline [8] which is depicted in Figure 1 (above), and

we also describe it throughout this section. The main idea is to take multiple *views* (i.e., different crops) of the data, and ensure that the features generated by these views are consistent with each other (in a sense which will be made precise shortly) as much as possible. If the views each contain a salient part of the input such as a central object, the feature corresponding to any view would then contain information about this central object. The end goal is that the feature of any large-enough view contains information about all relevant objects in the input image, which can then be extracted for use in downstream tasks such as image classification or image segmentation.

In the rest of the section, we will discuss the pre-training pipeline. As is common in contrastive SSL, the DINO framework uses two networks: a so-called *teacher* network parameterized by $\theta_{\mathrm{t}} \in \Theta$, and a so-called *student* network parameterized by $\theta_{\mathrm{s}} \in \Theta$. During pre-training, the loss will encourage the student's representation to align with the teacher's representation, even though the teacher is simultaneously being updated using student weights; this is known as self-distillation, and can be viewed as an optimization strategy or even implicitly regularizing the objective [7].

During the pipeline, we process each image $\boldsymbol{X}$ in the following way. First, we sample at random a view, or crop, v_c, independently of $\boldsymbol{X}$; the view can *either* be a "global view" (i.e., a large crop) or a "local view" (i.e., small crop), selected randomly. We denote $\boldsymbol{X}_c := v_c(\boldsymbol{X})$. In addition, we sample a global view v_g independently of $\boldsymbol{X}$ and v_c, and denote $\boldsymbol{X}_g := v_g(\boldsymbol{X})$.[1] Views are implemented in the same way as in DINO; they are formally described in Appendix A for the sake of completeness.

The first (local or global) view $\boldsymbol{X}_c$ is fed to the student network[2] $f_{\theta_{\mathrm{s}}}$ to get an aggregate representation $\boldsymbol{z}_c^{\mathrm{cls}}(\theta_{\mathrm{s}})$, while the global view $\boldsymbol{X}_g$ is fed to the teacher network $f_{\theta_{\mathrm{t}}}$ to get $\boldsymbol{z}_g^{\mathrm{cls}}(\theta_{\mathrm{t}})$, i.e.:

$$\boldsymbol{z}_c^{\mathrm{cls}}(\theta_{\mathrm{s}}) := f_{\theta_{\mathrm{s}}}^{\mathrm{cls}}(\boldsymbol{X}_c), \qquad \boldsymbol{z}_g^{\mathrm{cls}}(\theta_{\mathrm{t}}) := f_{\theta_{\mathrm{t}}}^{\mathrm{cls}}(\boldsymbol{X}_g). \tag{1}$$

Now, it is totally possible to directly compare and evaluate these features. However, DINO adds several post-processing steps, arguing that they improve performance and prevent collapse:

- They add a weight-normalized linear layer [26] $h_{\xi^{\mathrm{DINO}}} : \mathbb{R}^d \to \mathbb{R}^m$ where $m \gg d$, called the "DINO head" and parameterized by ξ^{DINO}, and pass both $\boldsymbol{z}_c^{\mathrm{cls}}$ and $\boldsymbol{z}_g^{\mathrm{cls}}$ through this layer.
- They center the teacher-computed features using a learned vector $\boldsymbol{\mu} \in \mathbb{R}^m$.
- They take a temperature-weighted softmax of both features to compute probability vectors in $\mathbb{R}^m$, sometimes called *prototype scores*, which they then can compare using cross-entropy.

Mathematically, the post-processing steps to get probability vectors for each view is as follows:

$$\boldsymbol{p}_c^{\mathrm{cls}}(\theta_{\mathrm{s}}, \xi^{\mathrm{DINO}}) := \mathrm{softmax}(h_{\xi^{\mathrm{DINO}}}(\boldsymbol{z}_c^{\mathrm{cls}}(\theta_{\mathrm{s}}))/\tau), \tag{2}$$

$$\boldsymbol{p}_g^{\mathrm{cls}}(\theta_{\mathrm{t}}, \xi^{\mathrm{DINO}}, \boldsymbol{\mu}) := \mathrm{softmax}([h_{\xi^{\mathrm{DINO}}}(\boldsymbol{z}_g^{\mathrm{cls}}(\theta_{\mathrm{t}})) - \boldsymbol{\mu}]/\tau), \tag{3}$$

where $\tau > 0$ is the temperature parameter. Finally, the loss (to be minimized) encourages $\boldsymbol{p}_c^{\mathrm{cls}}$ and $\boldsymbol{p}_g^{\mathrm{cls}}$ to be close together using a symmetrized cross-entropy-based functional d_{CE}, which effectively distills the teacher into the student by aligning the predicted outputs:

$$\mathcal{L}_{\mathrm{DINO}}(\theta_{\mathrm{s}}, \theta_{\mathrm{t}}, \xi^{\mathrm{DINO}}, \boldsymbol{\mu}) := \mathbb{E}[d_{\mathrm{CE}}(\boldsymbol{p}_c^{\mathrm{cls}}(\theta_{\mathrm{s}}, \xi^{\mathrm{DINO}}), \boldsymbol{p}_g^{\mathrm{cls}}(\theta_{\mathrm{t}}, \xi^{\mathrm{DINO}}, \boldsymbol{\mu}))] \tag{4}$$

where the expectation is over $\boldsymbol{X}$, the (local or global) view v_c, and the global view v_g sampled i.i.d., and the function d_{CE} is defined via the cross-entropy as

$$d_{\mathrm{CE}}(\boldsymbol{p}, \boldsymbol{q}) := \frac{1}{2}\left(\mathrm{CE}(\boldsymbol{p}, \boldsymbol{q}) + \mathrm{CE}(\boldsymbol{q}, \boldsymbol{p})\right), \qquad \mathrm{CE}(\boldsymbol{p}, \boldsymbol{q}) := -\sum_{i=1}^{m} p_i \log q_i. \tag{5}$$

When training, DINO estimates the expectation in (4) by a stratified plug-in estimator over a batch of sample images. That is, to estimate the expectation, we condition on $\boldsymbol{X}$ then estimate the conditional

[1] More precisely, let c be a random vector containing the boundaries of the crop, so that v_c crops exactly the region supplied by c. Analogous notation can be defined for g and v_g.

[2] Note that the parameters θ_{s} and θ_{t} each contain a positional encoding over all patches; when a view is fed through the network, it receives a interpolated positional encoding corresponding to the tokens' length.

expectation $\mathbb{E}[d_{\mathrm{CE}}(\cdot,\cdot) \mid \boldsymbol{X}]$ via plug-in using several different global views (usually two global views, which play the role of the arbitrary view v_c and the global view v_g) and several different local views, and finally average over $\boldsymbol{X}$ to obtain the estimate. The optimization of this estimated loss, too, is done in an ad-hoc way; while all four parameters $\theta_{\mathrm{s}}, \theta_{\mathrm{t}}, \xi^{\mathrm{DINO}}, \boldsymbol{\mu}$ are updated at each iteration, they update in different ways:

- The parameters θ_{s} and ξ^{DINO} are updated via an iteration of a stochastic gradient descent (SGD)-type algorithm, such as Adam, using the loss (4), and the derivatives for backpropagation are only computed through the first argument of the d_{CE} function in (4).
- The parameters θ_{t} and $\boldsymbol{\mu}$ are updated via exponentially moving averages (EMAs) of the student weights θ_{s} and the average output of the DINO head $\mathbb{E}[h_{\xi^{\mathrm{DINO}}}(\boldsymbol{z}_c^{\mathrm{cls}}(\theta_{\mathrm{s}}))]$ (in practice taken over a minibatch), respectively. Mathematically, for a decay parameter $\lambda \in [0,1]$, at each iteration we compute $\theta_{\mathrm{t}} \leftarrow \lambda\theta_{\mathrm{t}} + (1-\lambda)\theta_{\mathrm{s}}$ and $\boldsymbol{\mu} \leftarrow \lambda\boldsymbol{\mu} + (1-\lambda)\,\mathbb{E}[h_{\xi^{\mathrm{DINO}}}(\boldsymbol{z}_c^{\mathrm{cls}}(\theta_{\mathrm{s}}))]$.

Both the decay parameter λ and the temperature parameter τ change along the optimization trajectory, and their schedule is another design decision.

As previously mentioned, many of the ad-hoc methods and choices described above are due to a tension: a *trivial* solution to optimizing (4) is to enforce that $f_{\theta_{\mathrm{s}}}$ and $f_{\theta_{\mathrm{t}}}$ *collapse*, i.e., become or approximate the constant function, which map each local and global view to the same feature $\boldsymbol{z}$ or even to the same probability vector $\boldsymbol{p}$. To explain why DINO does not collapse, we wish to highlight the centering operation in (3), which computes batch statistics during its EMA update, hence using negative samples and implicitly pushing different samples' features apart, even though the precise conceptual mechanism by which this occurs is not clear and involves a careful interaction between the centering vector and temperature scaling [8]. Indeed, Caron et al. [8] shows that collapsed solutions are common without very carefully tuning the EMA schedule and temperature schedule, and arguing that the remaining hyperparameters and choices would severely degrade the performance if perturbed. A more in-depth discussion of the tension, and the added complexity required to train a model in spite of it, is in Appendix B. As we will see, if this tension is alleviated in an alternative way, many hyperparameters can be removed and the rest can be changed robustly.

2.2. From DINO to SimDINO

To go from DINO to SimDINO, we ask and answer a very basic question:

Can we just directly compare the features $\boldsymbol{z}_c^{\mathrm{cls}}$ and $\boldsymbol{z}_g^{\mathrm{cls}}$?

If we could do this, then we could avoid the large DINO head, the centering operation, the softmaxes, and the cross-entropy based loss. However, the mechanism in DINO for avoiding representation collapse via negative samples would therefore be removed. Thus, we have a second question:

Can we efficiently use the negative samples' features explicitly to enforce non-collapse?

For the first question, we argue that the most simple *squared Euclidean distance*, namely

$$d_{\ell^2}(\boldsymbol{x},\boldsymbol{y}) := \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{y}\|_2^2 \tag{6}$$

works at least as well as the cross-entropy-based functional (5) applied to an affine transformation of the features, as in (4). For the second question, we argue that we may directly penalize the covariance of the features in order to avoid collapse, as follows. For a hyperparameter $\varepsilon > 0$, define the (total) coding rate [21–23] of a symmetric positive semidefinite matrix $\boldsymbol{\Gamma} \in \mathbb{R}^{d\times d}$ as

$$R_\varepsilon(\boldsymbol{\Gamma}) := \frac{1}{2}\operatorname{logdet}\left(\boldsymbol{I} + \frac{d}{\varepsilon^2}\boldsymbol{\Gamma}\right), \tag{7}$$

In words, R_ε is an approximation to the rate distortion of a Gaussian random variable with covariance $\boldsymbol{\Gamma}$ (and this approximation is perfect in the limit $\varepsilon \to 0$). More concretely, it is a measure of

size of the covariance, even if the underlying variables are non-Gaussian. Thus one way to ensure non-collapse is to add $-R_\varepsilon(\mathrm{Cov}[\boldsymbol{z}_v^{\mathrm{cls}}(\theta_\mathrm{s})])$ as a regularizer in the objective, leading to the loss

$$\mathcal{L}_{\mathrm{SimDINO}}(\theta_\mathrm{s}, \theta_\mathrm{t}) := \mathbb{E}[d_{\ell^2}(\boldsymbol{z}_c^{\mathrm{cls}}(\theta_\mathrm{s}), \boldsymbol{z}_g^{\mathrm{cls}}(\theta_\mathrm{t}))] - \gamma R_\varepsilon(\mathrm{Cov}[\boldsymbol{z}_c^{\mathrm{cls}}(\theta_\mathrm{s})]). \tag{8}$$

where $\gamma > 0$ is a hyperparameter. Note that $d_{\ell^2}(\boldsymbol{z}_c^{\mathrm{cls}}, \boldsymbol{z}_g^{\mathrm{cls}}) = -(\boldsymbol{z}_c^{\mathrm{cls}})^\top \boldsymbol{z}_g^{\mathrm{cls}}$ since $\boldsymbol{z}_c^{\mathrm{cls}}, \boldsymbol{z}_g^{\mathrm{cls}} \in \mathbb{S}_{d-1}$.

When training, similar to DINO, we estimate the expectation and covariance in (8) by a type of plug-in estimator. Namely, the expectation is estimated similar to DINO, just using d_{ℓ^2} instead of d_{CE}. To estimate the coding rate, we sub-sample several $\boldsymbol{z}_c^{\mathrm{cls}}(\theta_\mathrm{s})$ over both $\boldsymbol{X}$ and v_c,[3] estimate $\mathrm{Cov}[\boldsymbol{z}_c^{\mathrm{cls}}(\theta_\mathrm{s})]$ on that sub-sample via plug-in, estimate R_ε of the population covariance by calculating it on the sample covariance, then average the estimates over all sub-samples. We conjecture that the latter estimator has lower variance compared to the naive plug-in estimator for $\mathrm{Cov}[\boldsymbol{z}_c^{\mathrm{cls}}(\theta_\mathrm{s})]$ as it is similar to variance-reduction methods in statistics [27], which we hypothesize might be a factor as to why SimDINO can handle a smaller batch size than other contrastive SSL methods that explicitly use negative samples but avoid collapse using higher-variance or more implicit regularizers.

The overall pipeline is shown in Figure 1 (bottom). Note that it is much simpler than DINO.

After training, we discard student weights and use teacher weights for evaluation.

2.3. From DINOv2 to SimDINOv2

The pipeline of the DINOv2 framework [12], as shown in Figure 2 top, is built upon the DINO pipeline, and has two main goals: first, learn an *aggregate* representation which contains large-scale semantics of the input (i.e., the goal of DINO); second, learn *patch-based* representations which have fine-grained semantic information about each patch and its local neighborhood. The main new ideas to achieve this, drawn from the iBOT pipeline [9], are that *the input to the student has some masked patches*, and that *the loss also computes similarity of the patch-based features*. To see why this works, consider if some patches are masked, and the model is able to predict masked patches using their unmasked neighbors, then from each patch the model can extract strong information about the semantics of nearby patches, which is an idea similar in spirit to masked autoencoding [10]. Thus, these two ideas from iBOT would furnish our model with informative patch-based representations.

We now discuss the pre-training pipeline, before discussing our modifications. Formally, starting with tokenized images $\boldsymbol{X} \in \mathbb{R}^{D \times N}$, we take a view v_m sampled at random; the view can be a global view or a local view, *but it also replaces a fraction* $\alpha \in [0, 1]$ *of the tokens in the view with a learnable mask token* $\boldsymbol{x}_{\mathrm{mask}}$ (as in [10], the mask token is shared across all views). We denote $\boldsymbol{X}_m := v_m(\boldsymbol{X})$. We also take a global view v_g *without* masking, independently of v_m and $\boldsymbol{X}$, and denote $\boldsymbol{X}_g := v_g(\boldsymbol{X})$.

Now that we have this setup, we do similar operations to DINO pipeline, with some changes:

- There is an additional "iBOT head", processing the patch-based features column-wise, with different weights ξ^{iBOT} (cf the "DINO head" with weights ξ^{DINO}).
- Both the "DINO head" and "iBOT head" are untied between teacher and student, giving four different weights $\xi_\mathrm{s}^{\mathrm{DINO}}, \xi_\mathrm{t}^{\mathrm{DINO}}, \xi_\mathrm{s}^{\mathrm{iBOT}}, \xi_\mathrm{t}^{\mathrm{iBOT}}$.
- The centering operation on teacher-output features is performed on both the aggregate features and (column-wise) on the patch-wise features.
- The centering operation uses three iterations of the Sinkhorn-Knopp algorithm [28, 29], denoted below by SKCenter, instead of an EMA, and is parameter-free but more expensive. Note that the Sinkhorn-Knopp algorithm uses features from all images in each minibatch.

Let $\boldsymbol{z}^i \in \mathbb{R}^d$ be the i^{th} column of $\boldsymbol{Z}^{\mathrm{patch}}$ (and similar for $\boldsymbol{p}^i \to \boldsymbol{P}^{\mathrm{patch}}$). Then, formally we have

$$(\boldsymbol{z}_g^{\mathrm{cls}}(\theta_\mathrm{t}), \boldsymbol{Z}_g^{\mathrm{patch}}(\theta_\mathrm{t})) := f_{\theta_\mathrm{t}}(\boldsymbol{X}_g), \qquad (\boldsymbol{z}_m^{\mathrm{cls}}(\theta_\mathrm{s}), \boldsymbol{Z}_m^{\mathrm{patch}}(\theta_\mathrm{s})) := f_{\theta_\mathrm{s}}(\boldsymbol{X}_m), \tag{9}$$

[3] In practice, we only let v_c be a global view for efficiency, and offer the following heuristic justification. If the expected similarity term in (8) is large, then there is little difference between the features of local and global views. Hence $\mathrm{Cov}[\boldsymbol{z}_c^{\mathrm{cls}}(\theta_\mathrm{s})] \approx \mathrm{Cov}[\boldsymbol{z}_{g'}^{\mathrm{cls}}(\theta_\mathrm{s})]$, where $v_{g'}$ is a randomly sampled global view.

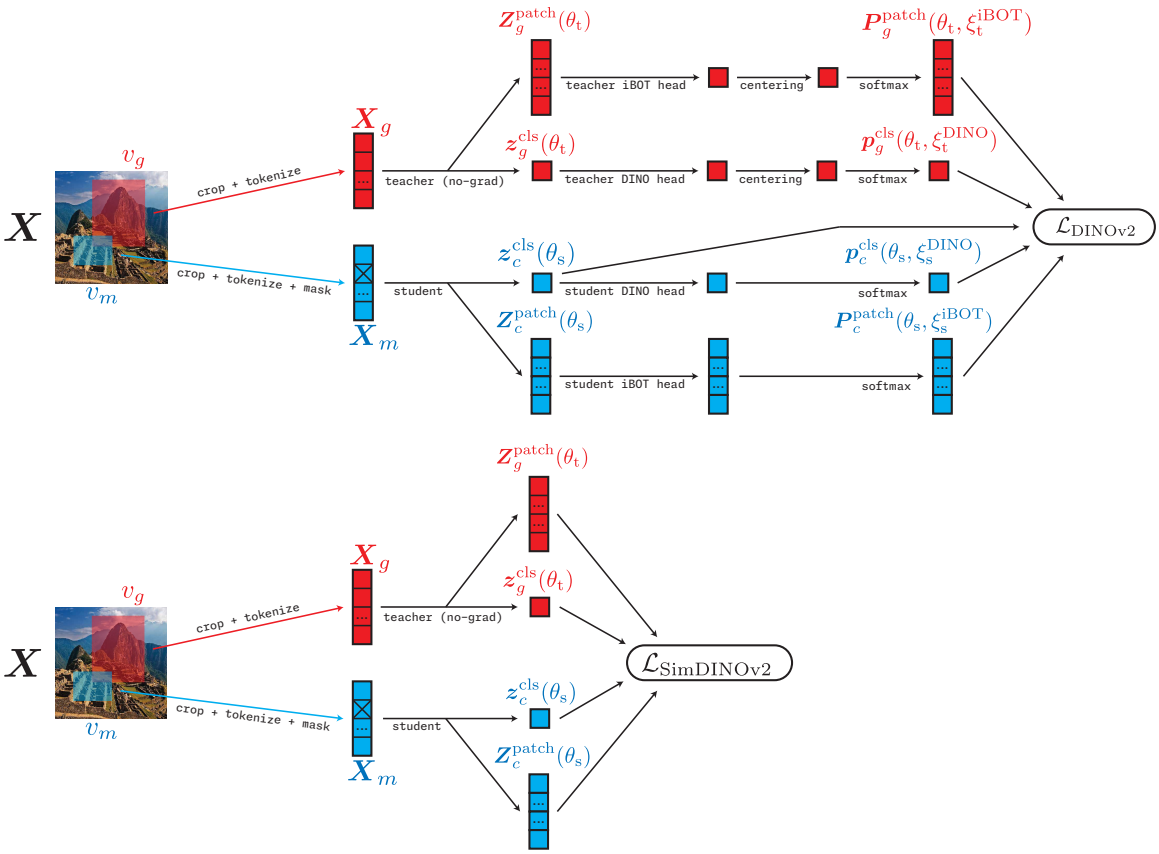

Figure 2: **The DINOv2 pipeline (*above*) and SimDINOv2 pipeline (*below*)**, with the selected teacher (global) view v_g and the student (either local or global, and partially masked) view v_m. The main addition to the pipeline in Figure 1 is the masking (here masked patches are denoted by $\times$), and the added loss on image patch features. *As before, the SimDINOv2 training operates directly on the learned representations.*

$$\boldsymbol{p}_m^{\text{cls}}(\theta_{\text{s}}, \xi_{\text{s}}^{\text{DINO}}) := \text{softmax}(h_{\xi_{\text{s}}^{\text{DINO}}}(\boldsymbol{z}_m^{\text{cls}}(\theta_{\text{s}}))/\tau), \tag{10}$$

$$\boldsymbol{p}_m^{i}(\theta_{\text{s}}, \xi_{\text{s}}^{\text{iBOT}}) := \text{softmax}(h_{\xi_{\text{s}}^{\text{iBOT}}}(\boldsymbol{z}_m^{i}(\theta_{\text{s}}))/\tau), \qquad 1 \le i \le N, \tag{11}$$

$$\boldsymbol{p}_g^{\text{cls}}(\theta_{\text{t}}, \xi_{\text{t}}^{\text{DINO}}) := \text{softmax}(\text{SKCenter}[h_{\xi_{\text{t}}^{\text{DINO}}}(\boldsymbol{z}_g^{\text{cls}}(\theta_{\text{t}}))]/\tau), \tag{12}$$

$$\boldsymbol{p}_g^{i}(\theta_{\text{t}}, \xi_{\text{t}}^{\text{iBOT}}) := \text{softmax}(\text{SKCenter}[h_{\xi_{\text{t}}^{\text{iBOT}}}(\boldsymbol{z}_g^{i}(\theta_{\text{t}}))]/\tau), \qquad 1 \le i \le N. \tag{13}$$

We then compute the loss using all generated probability vectors, as follows:

$$\mathcal{L}_{\text{DINOv2}}(\theta_{\text{s}}, \theta_{\text{t}}, \xi_{\text{s}}^{\text{DINO}}, \xi_{\text{s}}^{\text{iBOT}}, \xi_{\text{t}}^{\text{cls}}, \xi_{\text{t}}^{\text{patch}}) := \frac{1}{2}\,\mathbb{E}\Bigg[d_{\text{CE}}(\boldsymbol{p}_m^{\text{cls}}(\theta_{\text{s}}, \xi_{\text{s}}^{\text{DINO}}), \boldsymbol{p}_g^{\text{cls}}(\theta_{\text{t}}, \xi_{\text{t}}^{\text{DINO}})) \tag{14}$$

$$+ \frac{1}{N}\sum_{i=1}^{N} d_{\text{CE}}(\boldsymbol{p}_m^{i}(\theta_{\text{s}}, \xi_{\text{s}}^{\text{iBOT}}), \boldsymbol{p}_g^{i}(\theta_{\text{t}}, \xi_{\text{t}}^{\text{iBOT}})) \cdot 1_{\text{patch } i \text{ masked by } v_m} \Bigg] - \gamma\,\text{Entropy}(\boldsymbol{z}_m^{\text{cls}}(\theta_{\text{s}})), \tag{15}$$

where 1_E is the indicator function on event E (namely evaluating to 1 if E occurs, and 0 otherwise), and the Entropy functional is the differential entropy; it plays a similar role as the coding rate R_ε in SimDINO (and shortly SimDINOv2) in ensuring non-collapse. It is estimated by Oquab et al. [12] using the KoLeo estimator [30]) which explicitly uses negative samples. However, the KoLeo estimator is a non-parametric estimator of the expectation of a function of a high-dimensional probability density [31], and so it has relatively high sample-complexity (i.e., required batch size).

We can greatly simplify this pipeline by using the same ideas as introduced in SimDINO. Namely, we dispense with the DINO/iBOT heads, the Sinkhorn-Knopp centering, and the softmaxes, and

compute the Euclidean distance-based loss directly on normalized features. We obtain the loss

$$\mathcal{L}_{\text{SimDINOv2}}(\theta_{\text{s}}, \theta_{\text{t}}) := \frac{1}{2}\,\mathbb{E}\left[d_{\ell^2}(\boldsymbol{z}_m^{\text{cls}}(\theta_{\text{s}}), \boldsymbol{z}_g^{\text{cls}}(\theta_{\text{t}})) + \frac{1}{N}\sum_{i=1}^{N} d_{\ell^2}(\boldsymbol{z}_m^{i}(\theta_{\text{s}}), \boldsymbol{z}_g^{i}(\theta_{\text{t}}))1_{\text{patch } i \text{ masked by } v_m}\right] \tag{16}$$

$$- \gamma R_{\varepsilon}(\text{Cov}[\boldsymbol{z}_m^{\text{cls}}(\theta_{\text{s}})]). \tag{17}$$

The same caveats as in SimDINO apply with respect to how the expectations and covariances are estimated, and the optimization and evaluation procedures carry over. In the sequel, we will show that these greatly simplified designs actually help the model performance.

Optimal value for γ. In both the SimDINO loss (8) and the SimDINOv2 loss (16), in order to aid learning while making sure neither the distance term nor the regularizer term dominates, we choose γ up to an absolute constant factor so that it balances the asymptotic order of the gradient (Frobenius) norms of both terms. By the Cauchy-Schwarz inequality, it suffices to equalize the norms of the gradients of each term w.r.t. the features $\boldsymbol{Z}_c$. Since the features are normalized on the sphere, it holds that the gradient norm of the distance term is $\mathcal{O}(1)$. For the second term, assuming that we use n samples to estimate the covariance, Theorem 1 (Appendix C) says that the gradient norm of the second term is $\mathcal{O}(\sqrt{d\min\{d,n\}/n}/\varepsilon)$. To make these equivalent, we take $\gamma = \Theta(\varepsilon\sqrt{n/(d\min\{d,n\})})$. The same rate holds for SimDINOv2. We recognize that this choice of γ is ultimately a heuristic, and the constant factor needs to be tuned, but it helps to scale SimDINO and SimDINOv2 in practice.

3. Experiments and Evaluations

In this section, we present empirical results of our proposed SimDINO and SimDINOv2 models and compare them to the original DINO and DINOv2 model families. In particular, we examine their differences in learned representation both quantitatively and qualitatively. Our empirical studies aim to answer the following questions:

1. Do our proposed SimDINO and SimDINOv2 models achieve better performance, higher quality representations, and greater training stability than their original counterparts?
2. Do our proposed simplified pipelines yield similar emergent segmentation properties, an iconic feature of the original DINO?

We provide evidence for positive answers to both questions. Overall, our experiments show that our proposed SimDINO model families can achieve better performance than the original DINO families while being significantly simpler and more robust to various hyperparameter choices.

3.1. Experiment Setup

Model architecture. Since our method is directly built upon DINO, we adopt settings as close as possible to the original method for fair comparison. Specifically, for all inputs we set patch size to be 16; we use the small, base, and large models of the ViT [24] architecture as the backbone, which is connected to a projector composed of three MLP layers with a hidden size of 2048 and an output dimension of 256. The output features after the projector are ℓ^2 normalized. Specifically for original (i.e., not simplified) DINO models, these normalized features are then fed to a weight-normalized linear layer that outputs a high-dimensional (e.g., $\approx 65,000$) vector, before computing the temperature-weighted softmax and then the cross-entropy loss.

Datasets and optimization. We use the ImageNet-1k dataset for pretraining across all methods. For fair comparison, we closely follow the settings from the original works. Following [8, 12], we choose AdamW as the optimizer and use the same optimization strategies (e.g. learning rates, warm-up schedules). For training, we use 10 local views of resolution 96×96 and 2 global views of resolution 224×224 for all experiments. We provide more details on hyperparameter choices in Appendix D.

3.2. Experiment Results

ImageNet Classification. We report the classification accuracies on ImageNet-1k in Table 1. Following [8], we evaluate both k-NN accuracy and linear accuracy on the ViT backbones pretrained by the DINO model families and our simplified variants. We observe that for both DINO and DINOv2 paradigms, our simplified methods are able to outperform the original pipelines, demonstrating their advantages and effectiveness. Furthermore, we observe that applying identical hyperparameter settings from ViT-B to ViT-L results in instability and divergence in DINO. In contrast, we find that training SimDINO on ViT-L with the same hyperparameter configurations used for ViT-B yields a steady improvement in performance. To better understand the optimization dynamics of SimDINO, we also visualize the evolution of accuracy during the entire training process in Figure 3. It can be observed that performance of SimDINO steadily improves as training progresses, while optimization of DINO noticeably slows down, with even a slight performance drop near the end of training. Together, these results demonstrate our simplified pipelines' stability and ease of optimization compared to the originals.

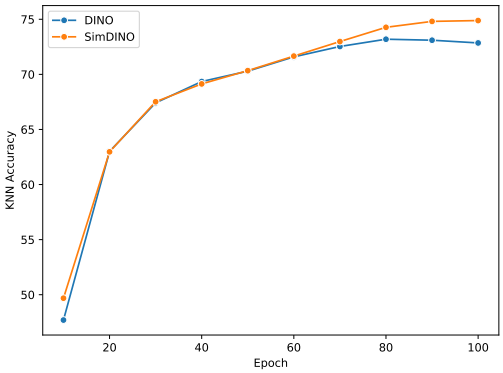

Figure 3: Evolution of k-NN accuracy of ViT-B trained for 100 epochs using DINO and SimDINO paradigms on Imagenet-1K. Accuracy is measured at every 10 epochs.

Method	Model	Epochs	k-NN	Linear
DINO	ViT-B	100	72.85	76.33
SimDINO	ViT-B	100	**74.89**	**77.26**
DINO	ViT-L	100	–	–
SimDINO	ViT-L	100	**75.64**	**77.36**
DINOv2	ViT-B	100	76.96	78.68
SimDINOv2	ViT-B	100	**78.05**	**79.66**
DINOv2	ViT-L	100	80.80	82.01
SimDINOv2	ViT-L	100	**81.11**	**82.41**

Table 1: Performance comparison on ImageNet-1K. SimDINO and SimDINOv2 consistently outperform the original DINO and DINOv2 model families. They are also more stable, while training of DINO on ViT-L diverged (row 3).

Object Detection and Segmentation. To gain deeper insights on the learned representation, we evaluate the performance of models trained using our simplified DINO algorithms on segmentation and object detection tasks. Specifically, we adopt MaskCut [32], an effective unsupervised approach of extracting features from a frozen vision backbone for segmentation and object detection. In Figure 4, we present qualitative segmentation results by applying MaskCut on mdoels trained with both DINO and SimDINO. Both methods are observed to produce meaningful segmentation results, confirming the emerging properties similar to the original DINO when using our simplified algorithm. To quantitatively evaluate the quality of these representation, we further perform MaskCut on the COCO val2017 dataset [33] and report our results in Table 2. These results show SimDINO achieves much stronger performance on segmentation and detection tasks than DINO when trained on the same network (row 2 vs 3), and overall even outperforms DINO trained on a smaller patch size[4] (row 2 vs 4). These results corroborate that the SimDINO produces features suitable for downstream tasks including segmentation and detection.

4. Conclusion

In this work, we identify that the reasons for many empirically motivated design choices in the original DINO and DINOv2 are to avoid collapse of the learned representation. We show that

[4]When trained using DINO, ViT models with smaller patch sizes tend to outperform those with larger ones on various tasks including segmentation, as observed in [8, 32].

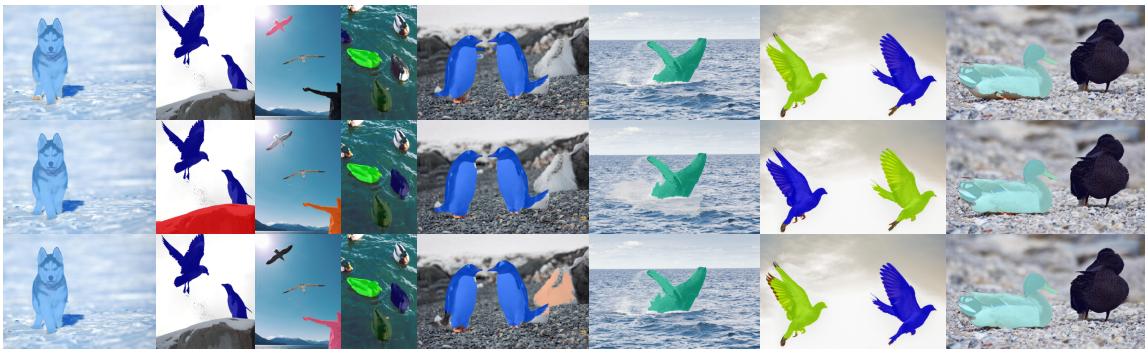

Figure 4: Visualization of MaskCut segmentation results from DINO ViT-B/16(row 1), SimDINO ViT-B/16(row 2) and SimDINO ViT-L/16(row 3) on selected images.

		Detection↑			Segmentation↑		
Method	Model	AP_{50}	AP_{75}	AP	AP_{50}	AP_{75}	AP
SimDINO	ViT-L/16	**5.4**	1.9	2.4	4.5	1.4	1.9
SimDINO	ViT-B/16	5.2	**2.0**	**2.5**	**4.7**	**1.5**	**2.0**
DINO	ViT-B/16	3.9	1.5	1.8	3.1	1.0	1.4
DINO	ViT-B/8	5.1	2.3	2.5	4.1	1.3	1.8

Table 2: **Object detection and segmentation via MaskCut evaluated on COCO val2017** [33] under COCO's official evaluation protocol. SimDINO conclusively performs better than the DINO at detection and segmentation metrics, comparable with DINO with smaller path size(16 vs 8).

these complicated design choices can be significantly reduced or simplified by adding a coding-rate-related regularization term. The resulting simplified models, called SimDINO and SimDINOv2, are even better in terms of performance for downstream tasks, and their pretraining pipelines are much more robust to different settings and hyperparameters, offering a Pareto improvement against the DINO and DINOv2 model families. Our work demonstrates the value of simplifying deep learning pipelines as well as making tradeoffs as explicit as possible when designing high-performance vision SSL models.

In light of these overarching contributions, there are several possible opportunities for future work. On the theoretical side, our simplified framework provides an entry point for studying the geometric properties of the global optima of self-supervised learning losses. Further study in Appendix E shows that in the framework of the paper, it is possible to set up a self-supervised objective that does not require self-distillation to optimize, making a theoretical analysis much easier, while the resulting model is still quite powerful and practically usable. On the empirical side, one can apply the paradigm of making implicit design choices more explicitly present in the loss to more self-supervised learning frameworks, making existing pipelines more stable and the resulting models of better performance.

References

[1] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[2] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[3] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3733–3742, 2018.

[4] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

[5] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[6] Adrien Bardes, Jean Ponce, and Yann LeCun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. *arXiv preprint arXiv:2105.04906*, 2021.

[7] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15750–15758, 2021.

[8] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021.

[9] Jinghao Zhou, Chen Wei, Huiyu Wang, Wei Shen, Cihang Xie, Alan Yuille, and Tao Kong. ibot: Image bert pre-training with online tokenizer. *arXiv preprint arXiv:2111.07832*, 2021.

[10] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022.

[11] Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15619–15629, 2023.

[12] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[13] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[14] Christoph Feichtenhofer, Yanghao Li, Kaiming He, et al. Masked autoencoders as spatiotemporal learners. *Advances in neural information processing systems*, 35:35946–35958, 2022.

[15] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[16] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[17] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[18] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[19] Mohammed Baharoon, Waseem Qureshi, Jiahong Ouyang, Yanwu Xu, Kilian Phol, Abdulrhman Aljouie, and Wei Peng. Towards general purpose vision foundation models for medical image analysis: An experimental study of dinov2 on radiology benchmarks. *arXiv preprint arXiv:2312.02366*, 2023.

[20] Zhixiang Wei, Lin Chen, Yi Jin, Xiaoxiao Ma, Tianle Liu, Pengyang Ling, Ben Wang, Huaian Chen, and Jinjin Zheng. Stronger fewer & superior: Harnessing vision foundation models for domain generalized semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 28619–28630, 2024.

[21] Yi Ma, Harm Derksen, Wei Hong, and John Wright. Segmentation of multivariate mixed data via lossy data coding and compression. *IEEE transactions on pattern analysis and machine intelligence*, 29(9):1546–1562, 2007.

[22] Yaodong Yu, Kwan Ho Ryan Chan, Chong You, Chaobing Song, and Yi Ma. Learning diverse and discriminative representations via the principle of maximal coding rate reduction. *Advances in neural information processing systems*, 33:9422–9434, 2020.

[23] Zengyi Li, Yubei Chen, Yann LeCun, and Friedrich T Sommer. Neural manifold clustering and embedding. *arXiv preprint arXiv:2201.10000*, 2022.

[24] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[25] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021.

[26] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29, 2016.

[27] Herman Kahn and Andy W Marshall. Methods of reducing sample size in monte carlo computations. *Journal of the Operations Research Society of America*, 1(5):263–278, 1953.

[28] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26, 2013.

[29] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *Advances in neural information processing systems*, 33:9912–9924, 2020.

[30] Sylvain Delattre and Nicolas Fournier. On the kozachenko–leonenko entropy estimator. *Journal of Statistical Planning and Inference*, 185:69–93, 2017.

[31] Jan Beirlant, Edward J Dudewicz, László Györfi, Edward C Van der Meulen, et al. Nonparametric entropy estimation: An overview. *International Journal of Mathematical and Statistical Sciences*, 6(1):17–39, 1997.

[32] Xudong Wang, Rohit Girdhar, Stella X Yu, and Ishan Misra. Cut and learn for unsupervised object detection and instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3124–3134, 2023.

[33] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

A. Formal Description of Local and Global Views

Each local view, say v_ℓ acts as follows, given an input image $\boldsymbol{X}$ of shape (C, H, W). First, for a hyperparameter $p_{\text{loc}} \in [0, 1]$ it crops a rectangular component from $\boldsymbol{X}$ of shape (C, H_ℓ, W_ℓ), where H_ℓ and W_ℓ are chosen such that $H_\ell W_\ell = p_{\text{loc}} HW$, i.e., the crop is a fraction p_{loc} of the whole image. Then the component is resized to shape $(C, S_{\text{loc}}, S_{\text{loc}})$, where S_{loc} is a hyperparameter, and then divided into $N_{\text{loc}} := S_{\text{loc}}^2/P^2$ square patches of shape (C, P, P), where the patch size P is a hyperparameter. Each patch is unrolled into a vector of length $D := CP^2$, and the N_{loc} unrolled vectors are placed in raster order to get the output $\boldsymbol{X}_\ell \in \mathbb{R}^{N_{\text{loc}} \times D}$. Each global view v_g acts the same as a local view, except that the corresponding hyperparameters $p_{\text{glo}}, S_{\text{glo}}$ are larger than their local counterparts $p_{\text{loc}}, S_{\text{loc}}$ (hence also N_{glo} vs. N_{loc}), while the patch size P (hence dimension D) remains the same.[5]

We use these local and global views for training. For evaluation or inference, we do a similar procedure: given $\boldsymbol{X}$ of shape (C, H, W), we resize $\boldsymbol{X}$ proportionally so that its *shorter* edge is length L_{eval}, then take a square crop from the center of shape $(C, S_{\text{eval}}, S_{\text{eval}})$. This sequence is divided into $N_{\text{eval}} := S_{\text{eval}}^2/P^2$ square patches of length (C, P, P); each patch is unrolled into a vector of length $D := CP^2$, and the N_{eval} unrolled vectors are placed in raster order to get the output $\boldsymbol{X}_e \in \mathbb{R}^{N_{\text{eval}} \times D}$.

B. Complex Interactions in DINO and Their Removal

We wish to showcase a finer point about why the DINO pipeline is so unstable. Notice that

$$\begin{aligned} \text{CE}(\boldsymbol{p}, \boldsymbol{q}) &= -\sum_{i=1}^{m} p_i \log q_i && (18)\\ &= \sum_{i=1}^{m} p_i \log(p_i/q_i) - \sum_{i=1}^{m} p_i \log p_i && (19)\\ &= \text{KL}(\boldsymbol{p}, \boldsymbol{q}) + H(\boldsymbol{p}) && (20) \end{aligned}$$

where KL is the KL divergence, and H is the entropy of a probability distribution. Therefore,

$$d_{\text{CE}}(\boldsymbol{p}, \boldsymbol{q}) = \underbrace{\frac{\text{KL}(\boldsymbol{p}, \boldsymbol{q}) + \text{KL}(\boldsymbol{q}, \boldsymbol{p})}{2}}_{=d_{\text{JS}}(\boldsymbol{p}, \boldsymbol{q})} + \frac{1}{2}(H(\boldsymbol{p}) + H(\boldsymbol{q})). \tag{21}$$

The first term $d_{\text{JS}}(\boldsymbol{p}, \boldsymbol{q})$ is the Jensen-Shannon divergence, which encourages $\boldsymbol{p} = \boldsymbol{q}$. The second term encourages the entropy of $\boldsymbol{p}$ and $\boldsymbol{q}$ to be low, namely closer to one-hot vectors.

Now consider the DINO objective:

$$\begin{aligned} &\mathcal{L}_{\text{DINO}}(\theta_{\text{s}}, \theta_{\text{t}}, \xi^{\text{DINO}}, \boldsymbol{\mu}) && (22)\\ &= \mathbb{E}[d_{\text{CE}}(\boldsymbol{p}_c^{\text{cls}}(\theta_{\text{s}}, \xi^{\text{DINO}}), \boldsymbol{p}_g^{\text{cls}}(\theta_{\text{t}}, \xi^{\text{DINO}}, \boldsymbol{\mu}))] && (23)\\ &= \mathbb{E}\left[d_{\text{JS}}(\boldsymbol{p}_c^{\text{cls}}(\theta_{\text{s}}, \xi^{\text{DINO}}), \boldsymbol{p}_g^{\text{cls}}(\theta_{\text{t}}, \xi^{\text{DINO}}, \boldsymbol{\mu})) + \frac{H(\boldsymbol{p}_c^{\text{cls}}(\theta_{\text{s}}, \xi^{\text{DINO}})) + H(\boldsymbol{p}_g^{\text{cls}}(\theta_{\text{s}}, \xi^{\text{DINO}}, \boldsymbol{\mu}))}{2}\right] && (24) \end{aligned}$$

Suppose that, for example, $h_{\xi^{\text{DINO}}}$ represented the identically zero function, and $\boldsymbol{\mu}$ is a constant multiple of the ones vector. Then the first term in the loss is minimized, but the second term becomes as large as possible (since both $\boldsymbol{p}^{\text{cls}}$ are just $\frac{1}{m}\mathbf{1}_m$, i.e., probability vectors corresponding to the uniform distribution), so this is in general not the optimal solution. This implies that the learned $h_{\xi^{\text{DINO}}}$ in general is not degenerate. This enables the tradeoff between the EMA parameter λ and the temperature parameter τ which enables non-collapse. If the objective just involved the JS divergence and not the entropy term, or else had $h_{\xi^{\text{DINO}}}$ be degenerate (manually set and frozen, for instance), or else didn't have a carefully set tradeoff between λ and τ, then the model would collapse. However, SimDINO removes all of this complexity and replaces it with an explicit coding-rate-type term.

[5] Of course, we also need the patch size P to divide both the image sizes S_{loc} and S_{glo}.

C. Theory for Hyperparameter Scaling

Let d, n be positive integers. Our main theorem is the following.

Theorem 1 (Scale of ∇R_ε)**.** *We have*

$$\max_{\substack{\boldsymbol{Z}\in\mathbb{R}^{d\times n}\\ \|\boldsymbol{Z}_i\|_2=1\ \forall i}} \left\|\nabla_{\boldsymbol{Z}} R_\varepsilon\left(\frac{\boldsymbol{Z}\boldsymbol{Z}^\top}{n}\right)\right\|_F \leq \frac{\sqrt{d\min\{d,n\}/n}}{4\varepsilon} \tag{25}$$

Proof. Let $\alpha := d/(n\varepsilon^2)$ and let $f\colon \mathbb{R}^{d\times n}\to\mathbb{R}$ be defined by

$$f(\boldsymbol{Z}) := \operatorname{logdet}(\boldsymbol{I} + \alpha \boldsymbol{Z}\boldsymbol{Z}^\top), \tag{26}$$

i.e., $f(\boldsymbol{Z}) = 2R_\varepsilon(\boldsymbol{Z}\boldsymbol{Z}^\top/n)$. Now, let $r := \min\{d,n\}$. For any matrix $\boldsymbol{M}$, let $\sigma_i(\boldsymbol{M})$ be its i^{th} largest singular value, for $i = 1,\dots,d$. First, note that since $\|\boldsymbol{Z}_i\|_2 = 1$ for all i, it holds

$$\sum_{i=1}^r \sigma_i(\boldsymbol{Z})^2 = \sum_{i=1}^d \sigma_i(\boldsymbol{Z})^2 = \sum_{i=1}^d \sigma_i(\boldsymbol{Z}\boldsymbol{Z}^\top) = \operatorname{tr}(\boldsymbol{Z}\boldsymbol{Z}^\top) = \sum_{i=1}^d (\boldsymbol{Z}\boldsymbol{Z}^\top)_{ii} = \sum_{i=1}^d \underbrace{\|\boldsymbol{Z}_i\|_2}_{=1} = d. \tag{27}$$

Now, we can simplify the gradient. It holds

$$\nabla f(\boldsymbol{Z}) = \alpha(\boldsymbol{I} + \alpha\boldsymbol{Z}\boldsymbol{Z}^\top)^{-1}\boldsymbol{Z}. \tag{28}$$

Thus, it holds that

$$\|\nabla f(\boldsymbol{Z})\|_F^2 = \operatorname{tr}([\nabla f(\boldsymbol{Z})]^\top[\nabla f(\boldsymbol{Z})]) \tag{29}$$

$$= \alpha^2 \operatorname{tr}(\boldsymbol{Z}^\top(\boldsymbol{I} + \alpha\boldsymbol{Z}\boldsymbol{Z}^\top)^{-2}\boldsymbol{Z}). \tag{30}$$

Using that the trace is the sum of singular values, it holds by taking the SVD of $\boldsymbol{Z}$ that

$$\operatorname{tr}(\boldsymbol{Z}^\top(\boldsymbol{I} + \alpha\boldsymbol{Z}\boldsymbol{Z}^\top)^{-2}\boldsymbol{Z}) = \sum_{i=1}^r \sigma_i(\boldsymbol{Z}^\top(\boldsymbol{I} + \alpha\boldsymbol{Z}\boldsymbol{Z}^\top)^{-2}\boldsymbol{Z}) \tag{31}$$

$$= \sum_{i=1}^r \frac{\sigma_i(\boldsymbol{Z})^2}{[1+\alpha\sigma_i(\boldsymbol{Z})^2]^2}. \tag{32}$$

In this case we directly optimize over the singular values, obtaining the problem

$$\max_{\substack{\boldsymbol{Z}\in\mathbb{R}^{d\times n}\\ \|\boldsymbol{Z}_i\|_2=1\ \forall i}} \|\nabla f(\boldsymbol{Z})\|_F \leq \max_{\substack{\boldsymbol{x}\in\mathbb{R}^r\\ x_i\geq 0\ \forall i\\ \sum_{i=1}^r x_i = d}} \sum_{i=1}^r \frac{x_i}{(1+\alpha x_i)^2}. \tag{33}$$

The function $t \mapsto \frac{t}{(1+\alpha t)^2}$ on $[0,\infty)$ has a global maximum at $t = \frac{1}{\alpha}$, and the value is $\frac{1}{4\alpha}$. Therefore it follows that

$$\max_{\substack{\boldsymbol{x}\in\mathbb{R}^r\\ x_i\geq 0\ \forall i\\ \sum_{i=1}^r x_i = d}} \sum_{i=1}^r \frac{x_i}{(1+\alpha x_i)^2} \leq \max_{\substack{\boldsymbol{x}\in\mathbb{R}^r\\ x_i\geq 0\ \forall i}} \sum_{i=1}^r \frac{x_i}{(1+\alpha x_i)^2} = \frac{r}{4\alpha}. \tag{34}$$

Unpacking this notation, we obtain

$$\|\nabla f(\boldsymbol{Z})\|_F^2 \leq \alpha^2 \cdot \frac{r}{4\alpha} = \frac{\alpha r}{4} = \frac{d\min\{d,n\}}{4n\varepsilon^2}. \tag{35}$$

Taking square roots, it holds

$$\|\nabla f(\boldsymbol{Z})\|_F \leq \frac{\sqrt{d\min\{d,n\}/n}}{2\varepsilon}. \tag{36}$$

Therefore,

$$\left\|\nabla_{\boldsymbol{Z}} R_\varepsilon\left(\frac{\boldsymbol{Z}\boldsymbol{Z}^\top}{n}\right)\right\|_F \leq \frac{1}{2}\|\nabla f(\boldsymbol{Z})\|_F \leq \frac{\sqrt{d\min\{d,n\}/n}}{4\varepsilon} \tag{37}$$

as desired. □

Remark 2. It is possible that the inequality

$$\max_{\substack{\boldsymbol{Z}\in\mathbb{R}^{d\times n}\\ \|\boldsymbol{Z}_i\|_2=1\ \forall i}} \|\nabla f(\boldsymbol{Z})\|_F \leq \max_{\substack{\boldsymbol{x}\in\mathbb{R}^r\\ x_i\geq 0\ \forall i\\ \sum_{i=1}^r x_i=d}} \sum_{i=1}^r \frac{x_i}{(1+\alpha x_i)^2}. \tag{38}$$

is met with equality; proving this would require exhibiting a $\boldsymbol{Z}$ fulfilling the constraints of the first problem such that it has the prescribed singular values which solve the second problem. We do not need to do so here for the purposes of using the bound (e.g., for learning rate scaling).

Remark 3. While the quick-and-dirty bound

$$\max_{\substack{\boldsymbol{x}\in\mathbb{R}^r\\ x_i\geq 0\ \forall i\\ \sum_{i=1}^r x_i=d}} \sum_{i=1}^r \frac{x_i}{(1+\alpha x_i)^2} \leq \frac{r}{4\alpha}, \tag{39}$$

by way of ignoring the constraint $\sum_{i=1}^r x_i = d$ seems like it could significantly loosen the bound, we do not believe this is the case. In particular, when $1/\alpha \leq d/r$, note that setting $x_1 = \cdots = x_{r-1} = 1/\alpha$ and $x_r = d-(r-1)/\alpha$ sandwiches the objective between $(r-1)/(4\alpha)$ and $r/(4\alpha)$, so the maximum is at least the same asymptotic order, in the very reasonable case that ε is small enough that $1/\alpha \leq d/r$, i.e., using the definition of α, such that

$$\frac{1}{\alpha} \leq \frac{d}{r} \iff \varepsilon^2 \leq \frac{d^2}{n\min\{d,n\}} \implies \varepsilon^2 \leq \min\left\{\frac{d}{n}, \frac{d^2}{n^2}\right\}. \tag{40}$$

Similar strategies should hold if we allow for an absolute constant $c \geq 1$ such that $1/\alpha \leq cd/r$, etc, relaxing the requirement while preserving the asymptotic order of the LHS of (39).

D. More Experiment Details

D.1. Implementation Details

The training codes and hyperparameters for SimDINO and SimDINOv2 are derived from the released official settings in DINO and DINOv2 separately, see Table 3 for detailed comparison. Notes that for SimDINOv2, we choose to use bfloat16 dtype in student backbone parameters and reductions for better numerical stability while other modules uses the same FSDP mixed precision settings from DINOv2.

E. DINO Without Self-Distillation

Due to the explicit coding rate regularization, it is possible to train SimDINO without self-distillation. To validate this, we train ViT-S models on ImageNet-1k by setting the teacher network to be the student network at each iteration, effectively removing the EMA operation. Results are presented in Table 4. We can see that the original DINO collapses under this setup for reasons discussed in Appendix B, while SimDINO is able to yield non-trivial performance. It is worth noting that compared to training with full self-distillation, this variant primarily lags behind in terms of k-NN performance while the gap in linear probe is significantly smaller.

Hyperparameter		SimDINOv2	DINOv2	SimDINOv1	DINOv1
Model	Patch size	16			
	Register Tokens	4		0	
	Pos-embedding anti-alias	True		False	
	Init layer scale	0.1	1e-5	-	
	Drop path rate	0.3		0.1	
	Output N prototypes	removed	65536	removed	65536
Pipeline	Init EMA Momentum	0.9	0.992	0.996	
	Centering temperature	removed	0.07	removed	0.07
	Warm-up temperature	removed	0.04	removed	0.04
	Warm-up temperature epochs	removed	30	removed	30
	iBOT sample prob.	0.5		-	
	iBOT mask ratio	0.1-0.5		-	
	iBOT head tying	False		-	
	Koleo loss weight	removed	0.1	-	
Data	Global crops scale	0.4 - 1			
	Local crops scale	0.05 - 0.4			
	Local crops number	10			
	Global crops size	224			
	Local crops size	96			
Optim.	Batch size	128x8		64x8	
	Epochs	100			
	Warm-up epochs	10			
	Freeze last layer epochs	removed	1	removed	1
	Learning rate	0.004		0.002	
	Layerwise lr decay	0.9		-	
	Weight decay	0.04			
	Weight decay end	0.4			
	Gradient clip	3.0		0.3	

Table 3: Training Hyperparameters used in the experiments

Method	Model	self-distillation	Epochs	k-NN	Linear
DINO	ViT-S	×	100	–	–
SimDINO	ViT-S	×	100	58.57	68.03
SimDINO	ViT-S	✓	100	70.22	73.66

Table 4: Performance on ImageNet-1K without self-distillation.