# Exploring the Equivalence of Siamese Self-Supervised Learning via A Unified Gradient Framework

Chenxin Tao[1*†], Honghui Wang[1*†], Xizhou Zhu[2*], Jiahua Dong[3†],
Shiji Song[1], Gao Huang[1,4], Jifeng Dai[2,4✉]
[1]Tsinghua University, [2]SenseTime Research, [3]Zhejiang University
[4]Beijing Academy of Artificial Intelligence, Beijing, China
{tcx20, wanghh20}@mails.tsinghua.edu.cn, {zhuwalter, daijifeng}@sensetime.com
cnjiahuadong@gmail.com, shijis@mail.tsinghua.edu.cn, gaohuang@tsinghua.edu.cn

## Abstract

*Self-supervised learning has shown its great potential to extract powerful visual representations without human annotations. Various works are proposed to deal with self-supervised learning from different perspectives: (1) contrastive learning methods (e.g., MoCo, SimCLR) utilize both positive and negative samples to guide the training direction; (2) asymmetric network methods (e.g., BYOL, SimSiam) get rid of negative samples via the introduction of a predictor network and the stop-gradient operation; (3) feature decorrelation methods (e.g., Barlow Twins, VICReg) instead aim to reduce the redundancy between feature dimensions. These methods appear to be quite different in the designed loss functions from various motivations. The final accuracy numbers also vary, where different networks and tricks are utilized in different works. In this work, we demonstrate that these methods can be unified into the same form. Instead of comparing their loss functions, we derive a unified formula through gradient analysis. Furthermore, we conduct fair and detailed experiments to compare their performances. It turns out that there is little gap between these methods, and the use of momentum encoder is the key factor to boost performance.*

*From this unified framework, we propose UniGrad, a simple but effective gradient form for self-supervised learning. It does not require a memory bank or a predictor network, but can still achieve state-of-the-art performance and easily adopt other training strategies. Extensive experiments on linear evaluation and many downstream tasks also show its effectiveness. Code shall be released.*

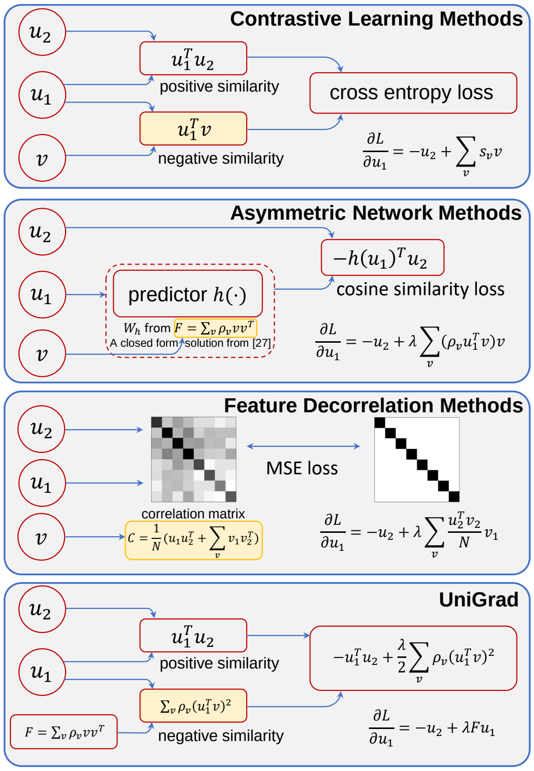

Contrastive Learning Methods
$u_2$
$u_1$
$v$
$u_1^T u_2$
positive similarity
$u_1^T v$
negative similarity
cross entropy loss
$\frac{\partial L}{\partial u_1} = -u_2 + \sum_v s_v v$
Asymmetric Network Methods
$u_2$
$u_1$
$v$
predictor $h(\cdot)$
$W_h$ from $F = \sum_v \rho_v v v^T$
A closed form solution from [27]
$-h(u_1)^T u_2$
cosine similarity loss
$\frac{\partial L}{\partial u_1} = -u_2 + \lambda \sum_v (\rho_v u_1^T v) v$
Feature Decorrelation Methods
$u_2$
$u_1$
$v$
correlation matrix
$C = \frac{1}{N}(u_1 u_2^T + \sum_v v_1 v_2^T)$
MSE loss
$\frac{\partial L}{\partial u_1} = -u_2 + \lambda \sum_v \frac{u_2^T v_2}{N} v_1$
UniGrad
$u_2$
$u_1$
$u_1^T u_2$
positive similarity
$\sum_v \rho_v (u_1^T v)^2$
negative similarity
$F = \sum_v \rho_v v v^T$
$-u_1^T u_2 + \frac{\lambda}{2} \sum_v \rho_v (u_1^T v)^2$
$\frac{\partial L}{\partial u_1} = -u_2 + \lambda F u_1$

Figure 1. Overview of three typical types of self-supervised learning methods and our proposed UniGrad. $u_1$ and $u_2$ are two augmented views of the same image. $v$ denote views of other images. We find these methods have a similar gradient structure composed of the positive and negative gradients, which can be analogous to positive and negative samples in contrastive learning. Because some methods do not explicitly utilize negative samples, we highlight the source of negative gradient in each method.

## 1. Introduction

Self-supervised learning (SSL) has recently attracted much research interest [1, 5, 7, 16, 20, 33]. It has shown the potential to extract powerful visual representations that

[1]Equal contribution. †This work is done when Chenxin Tao, Honghui Wang, and Jiahua Dong are interns at SenseTime Research. ✉Corresponding author.

are competitive with supervised learning, and delivered superior performance on multiple visual tasks.

Recent works deal with SSL from different points of view, leading to three typical types of methods (see Figure 1), while siamese networks are always employed. *Contrastive Learning* methods [5, 6, 8, 16] aim to reduce the distance between two augmented views from the same image (positive samples), and push apart views from different images (negative samples). Negative samples play an important role in these methods to avoid representational collapse. *Asymmetric Network* methods [7, 20] claim that only adopting positive samples is sufficient. The key is the introduction of asymmetric network architecture. In these methods, a predictor network is only appended after one branch of the siamese network, and the other branch is detached from the gradient back-propagation. Although these methods have achieved impressive performance, they are still poorly understood. Recent work [27] has tried to analyze their training dynamics, but still lacks a straight-forward explanation. *Feature Decorrelation* methods [1, 12, 19, 33] have recently been proposed as a new solution for SSL. They instead focus on reducing the redundancy between different feature dimensions. These methods seem to be highly different in how to learn representations, and it's also hard to compare their performances because different networks and tricks are utilized in different works. With so many different methods, it is natural to ask: What is the relationship among them? Are there any connections among the working mechanisms behind them? What factors actually cause the performance difference?

In this work, we unify the aforementioned three typical types of SSL methods in a unified framework. Instead of comparing their loss functions, the unified formula is derived through gradient analysis. We find that all these methods have similar gradient formulas. They consist of three components: the positive gradient, the negative gradient and a scalar that balances these two terms. The positive gradient is the representation of another augmented view from the same image, while the negative gradient is a weighted combination of the representations from different images. The effects of these two terms are similar to that of positive and negative samples in contrastive learning methods. This suggests that these methods share a similar working mechanism, but organize the loss functions in different manners. Moreover, since these methods are different in the specific formula of the gradient, we conduct fair and detailed experiments for comparison. It turns out that different gradient formulas result in close performance, and what really matters is the use of momentum encoder.

From this unified framework, we propose a concise but effective gradient formula named UniGrad, which explicitly maximizes the similarity between positive samples and expects the similarity between negative samples to be zero. This formula does not require memory bank or asymmetric network, and can easily adopt prevalent augmentation strategies (*e.g.*, CutMix [32] and multi-crop [3, 4]) to further improve the performance. Extensive experiments show that our method is competitive on various tasks, including the standard linear evaluation protocol, the semi-supervised learning task and various downstream vision tasks. Our contribution can be summarized as:

- A unified framework is proposed for different self-supervised learning methods through the perspective of gradient analysis. This shows that although previous works seem to be distinct in loss functions, they actually work in a similar mechanism;
- Different self-supervised learning methods are compared under a fair and controlled experiment setting. The results show that they can achieve similar performance, while the momentum encoder is actually the key factor affecting the final performance;
- UniGrad is proposed as a concise but effective gradient formula for self-supervised learning. Extensive experiments demonstrate its competitive performance.

## 2. Related Work

**Contrastive Learning Methods** have long been studied in the area of self-supervised learning [9, 30]. The main idea is to discriminate positive and negative samples. Since the propose of InfoNCE [25], many recent works [2, 3, 5, 6, 8, 16, 18, 21, 22, 35] have pushed the performance to a new height. In these methods, negative samples play a critical role and are carefully designed. MoCos [6, 8, 16] build a memory bank with a momentum encoder to provide consistent negative samples, yielding promising results for both CNNs [6, 16] and Vision Transformers [8]. SimCLR [5] enhances the representation of negative samples with strong data augmentations and a learnable nonlinear projection head. Other methods further combine contrastive learning with instance classification [2], data augmentation [21, 35], clustering [3, 22] and adversarial training [18].

Contrastive learning methods pull positive samples together and push negative samples away, leading to the alignment and uniformity properties of the representation on the hypersphere [29]. Our work finds that although non-contrastive learning methods are optimized for different objective functions, they share the similar gradient structure with that of contrastive learning.

**Asymmetric Network Methods** aim to accomplish self-supervised learning with only positive pairs [7, 20, 26]. The representational collapse is avoided through the introduction of asymmetric network architecture. BYOL [20] appends a predictor network after the online branch, and adopts a momentum encoder for the target branch. [26]

shows that BYOL is able to achieve competitive performance even without batch statistics. SimSiam [7] further shows that stopping the gradient to target branch can serve a similar role as the momentum encoder. DINO [4] adopts an asymmetric pipeline with a self-distillation loss for Vision Transformers. Despite the impressive performance, little is known about how the asymmetric network can avoid collapse. Recent work [27] makes a preliminary attempt to analyze the training dynamics, but still lacks a straightforward explanation.

Based on the conclusion of [27], our work builds a connection between asymmetric network with contrastive learning methods. From the perspective of backward gradient, we demonstrate the predictor learns to encode the information of previous samples in its weight, which serves as negative gradient during back-propagation. This leads to a similar gradient structure with contrastive learning.

**Feature Decorrelation Methods** are recently proposed for self-supervised learning to prevent the representational collapse [1, 12, 19, 33]. W-MSE [12] whitens feature representations before computing a cosine similarity loss so that the representations are scattered on the unit sphere. Barlow Twins [33] encourages the cross-correlation matrix of representations close to identity matrix, which decorrelates different dimensions of the representation, and strengthens the correlation in the same dimension. VICReg [1] applys variance-invariance-covariance principle to replace the use of batch normalization and cross-correlation matrix. Shuffled-DBN [19] explores the function of batch normalization on the embedding and develops a shuffle method for better feature decorrelation.

Feature decorrelation methods show comparable results to contrastive learning methods. However, it is still unclear why such approach works well. Our work demonstrates that the gradient formulas of feature decorrelation methods can be transformed to a combination of positive and negative samples, and thus share the similar gradient structure with that of contrastive learning.

## 3. A Unified Framework for SSL

A typical self-supervised learning framework consists of a siamese network. The two branches of the siamese network are named as online branch and target branch, respectively, where target branch representation is served as the training target for the online branch. Given the input image $x$, two augmented views $x_1$ and $x_2$ are created as the inputs of the two branches. The encoder $f(\cdot)$ extracts representations $u_i \triangleq f(x_i), i = 1, 2$ from these views.

Table 1 illustrates the notations used in this paper. $u_1$ and $u_2$ denote the currently concerned training samples, while $v$ denotes unspecified samples. $u_1^o$ and $v^o$ denote the representation extracted from the online branch. There are three types of target branches that are widely used: 1) weight-sharing with the online branch, corresponding to $u_2^s$ and $v^s$; 2) weight-sharing but detached from gradient back-propagation, corresponding to $u_2^d$ and $v^d$; 3) momentum encoder updated from the online branch, corresponding to $u_2^m$ and $v^m$. If the target branch type is not specified, $u_2^t$ and $v^t$ are used. Note that the symmetric loss is always used for the two augmented views as described in [7].

| Notation | Meaning |
|---|---|
| $u_1$, $u_2$ | current concerned samples |
| $v$ | unspecified samples |
| $u_1^o$, $v^o$ | samples from online branch |
| $u_2^t$, $v^t$ | samples from unspecified target branch |
| $u_2^s$, $v^s$ | samples from weight-sharing target branch |
| $u_2^d$, $v^d$ | samples from stop-gradient target branch |
| $u_2^m$, $v^m$ | samples from momentum-encoder target branch |
| $\mathcal{V}$ | unspecified sample set |
| $\mathcal{V}_{\text{batch}}$ | sample set of current batch |
| $\mathcal{V}_{\text{bank}}$ | sample set of memory bank |
| $\mathcal{V}_{\infty}$ | sample set of all previous samples |

Table 1. Notations used in this paper.

Moreover, $\mathcal{V}$ represents the sample set considered in current training step. Different methods construct the sample set in different manners: $\mathcal{V}_{\text{batch}}$ contains all samples from current batch, $\mathcal{V}_{\text{bank}}$ consists of a memory bank that stores previous samples, and $\mathcal{V}_{\infty}$ denotes the set of all previous samples, which can be much larger than a memory bank.

Details of gradient analysis can refer to Appendix.

### 3.1. Contrastive Learning Methods

Contrastive learning methods require negative samples to avoid representational collapse and achieve high performance. They use another view from the same image as the positve sample, and different images as the negative samples. These methods aim to pull positive pairs together while push negative pairs apart. The following InfoNCE loss [25] is usually employed:

$$L = \mathop{\mathbb{E}}_{u_1, u_2} \left[ -\log \frac{\exp\left(\cos(u_1^o, u_2^t)/\tau\right)}{\sum_{v^t \in \mathcal{V}} \exp\left(\cos(u_1^o, v^t)/\tau\right)} \right], \quad (1)$$

where the function $\cos(\cdot)$ measures the cosine similarity between two representations, and $\tau$ is the temperature hyperparameter. Eq.(1) can be instantiated for different methods, which we shall discuss below.

**Relation to MoCo [6, 16]**. MoCo adopts a momentum encoder for the target branch, and a memory bank to store previous representations from the target branch. Its negative samples come from the memory bank. The gradient for sample $u_1^o$ is therefore:

$$\frac{\partial L}{\partial u_1^o} = \frac{1}{\tau N}\Big( - u_2^m + \sum_{v^m \in \mathcal{V}_{\text{bank}}} s_v v^m \Big), \tag{2}$$

where $s_v = \frac{\exp\left(\cos(u_1^o, v^m)/\tau\right)}{\sum_{y^m \in \mathcal{V}_{\text{bank}}} \exp\left(\cos(u_1^o, y^m)/\tau\right)}$ is the softmax results over similarities between $u_1^o$ and other samples, and $N$ is the number of all samples in current batch.

**Relation to SimCLR [5].** For SimCLR, the target branch shares weights with the online branch, and does not stop the back-propagated gradient. It uses all representations from other images of the same batch as negative samples. Thus, its gradient can be calculated as:

$$\begin{aligned}\frac{\partial L}{\partial u_1^o} =& \frac{1}{\tau N}\Big( - u_2^s + \sum_{v^s \in \mathcal{V}_{\text{batch}} \backslash u_1^o} s_v v^s \Big) \\ &+ \underbrace{\frac{1}{\tau N}\Big( - u_2^s + \sum_{v^s \in \mathcal{V}_{\text{batch}} \backslash u_1^o} t_v v^s \Big)}_{\text{reduce to 0}}, \end{aligned} \tag{3}$$

where $t_v = \frac{\exp\left(\cos(v^s, u_1^o)/\tau\right)}{\sum_{y^s \in \mathcal{V}_{\text{batch}} \backslash v^s} \exp\left(\cos(v^s, y^s)/\tau\right)}$ is computed over similarities between sample $v^s$ and its contrastive samples $\mathcal{V}_{\text{batch}} \setminus v^s$. If the gradient through the target branch is stopped, the second term in Eq.(3) will vanish. We have verified that stopping the second gradient term will not affect the performance (see Appendix), so Eq.(3) can be simplified to only the first term.

**Unified Gradient.** From the perspective of gradient, above methods can be represented in a unified form:

$$\frac{\partial L}{\partial u_1^o} = \frac{1}{\tau N}\Big( - u_2^t + \sum_{v^t \in \mathcal{V}} s_v v^t \Big), \tag{4}$$

where the gradient is made up of a weighted sum of positive and negative samples. The effect of $-u_2^t$ is to pull positive samples together, and the effect of $\sum_{v^t \in \mathcal{V}} s_v v^t$ is to push negative samples apart. We name these two terms as the positive and negative gradient, respectively. The only difference between methods is what type of target branch is used and how the contrastive sample set $\mathcal{V}$ is built.

## 3.2. Asymmetric Network Methods

Asymmetric network methods learn powerful representations by maximizing the similarity of positive pairs, without using negative samples. Such methods need symmetry-breaking network designs to avoid representational collapse. To achieve this, a predictor $h(\cdot)$ is appended after the online branch. The gradient to the target branch is also stopped. The objective function can be presented as:

$$L = \mathop{\mathbb{E}}_{u_1, u_2}\Big[ - \cos(h(u_1^o), u_2^t) \Big]. \tag{5}$$

**Relation to BYOL [20].** For BYOL, a momentum encoder is used for the target branch, *i.e.*, $u_2^t = u_2^m$ in Eq.(5).

**Relation to Simsiam [7].** Simsiam shows that momentum encoder is not necessary, and only applies the stop-gradient operation to the target branch, *i.e.*, $u_2^t = u_2^d$ in Eq.(5).

**Unified Gradient.** While asymmetric network methods have achieved impressive performance, it is unclear how these methods avoid collapse solution. Recently, DirectPred [27] makes a preliminary attempt towards this goal via studying the training dynamics. It further proposes an analytical solution for the predictor $h(\cdot)$.

Specifically, DirectPred claims that the predictor can be formulated as $h(v) = W_h v$, where $W_h$ can be directly calculated based on the correlation matrix $\mathbb{E}_v(vv^T)$. In practice, this correlation matrix is calculated as the moving average of the correlation matrix for each batch, *i.e.*, $F \triangleq \sum_{v^o \in \mathcal{V}_\infty} \rho_v v^o v^{oT}$, where $\rho_v$ is the moving average weight for each sample according to their batch order. By decomposing $F$ into its eigenvalues $\Lambda_F$ and eigenvectors $U$, $W_h$ can be calculated as

$$W_h = U \Lambda_h U^T, \;\; \Lambda_h = \Lambda_F^{1/2} + \epsilon \lambda_{max} I, \tag{6}$$

where $\lambda_{max}$ is the max eigenvalue of $F$ and $\epsilon$ is a hyper-parameter to help boost small eigenvalues.

While DirectPred shows what the predictor learns, We step further and try to reveal the relationship between the predictor and contrastive learning. With the help of DirecPred, the gradient can be derived and simplified as:

$$\frac{\partial L}{\partial u_1^o} = \frac{1}{||W_h u_1^o||_2 N}\Big( -W_h^T u_2^t + \lambda \sum_{v^o \in \mathcal{V}_\infty} (\rho_v u_1^{oT} v^o) v^o \Big), \tag{7}$$

where $-W_h^T u_2^t$ and $\sum_{v^o \in \mathcal{V}_\infty} (\rho_v u_1^{oT} v^o) v^o$ work as the positive and negative gradient respectively and $\lambda = \frac{u_1^{oT} W_h^T u_2^t}{u_1^{oT}(F + \epsilon^2 I) u_1^o}$ is a balance factor.

It seems counter-intuitive that Eq.(7) is also a combination of positive and negative samples, since no negative samples appear in the loss function explicitly. In fact, they come from the optimization of the predictor network. From the findings of [27], the eigenspace of the predictor $W_h$ will gradually align with that of the feature correlation matrix $F$. Hence the predictor may learn to encode the information of correlation matrix in its parameters. During back-propagation, the encoded information will work as negative gradient and contribute to the direction of optimization.

## 3.3. Feature Decorrelation Methods

Feature decorrelation methods emerge recently as a new solution to self-supervised learning. It proposes to reduce the redundancy among different feature dimensions so as to avoid collapse. Recent works adopt different loss forms for feature decorrelation. We discuss their relations below.

**Relation to Barlow Twins [33].** Barlow Twins utilizes the following loss function:

$$L = \sum_{i=1}^{C} (W_{ii} - 1)^2 + \lambda \sum_{i=1}^{C} \sum_{j \neq i} W_{ij}^2, \tag{8}$$

where $W = \frac{1}{N} \sum_{v_1^o, v_2^s \in \mathcal{V}_{\text{batch}}} v_1^o v_2^{sT}$ is a cross-correlation matrix, $C$ denotes the number of feature dimensions and $\lambda$ is a balancing hyper-parameter. The diagonal elements of $W$ are encouraged to be close to 1, while those off-diagonal elements are forced to be close to 0.

At first glance, Eq.(8) is drastically different from loss functions of previous methods. However, it actually works in the same way from the view of gradient, which can be calculated as

$$\frac{\partial L}{\partial u_1^o} = \frac{2}{N} \Bigg( \underbrace{-A u_2^s}_{\text{reduce to } -u_2^s} + \lambda \sum_{v_1^o, v_2^s \in \mathcal{V}_{\text{batch}}} \frac{u_2^{sT} v_2^s}{N} v_1^o \Bigg), \tag{9}$$

where $A = I - (1 - \lambda) W_{\text{diag}}$. Here $(W_{\text{diag}})_{ij} = \delta_{ij} W_{ij}$ is the diagonal matrix of $W$, where $\delta_{ij}$ is the Kronecker delta.

It has been verified that removing matrix $A$ before $u_2^s$ actually does no harm to the final result (see Table 2(g)). In addition, it should be noted that Barlow Twins applies batch normalization rather than $\ell_2$ normalization to the representation $v$. We have verified that changing to $\ell_2$ normalization will not affect the performance (see Table 2(h)).

**Relation to VICReg [1].** VICReg does a few modifications to Barlow Twins with the following loss function:

$$\begin{aligned} L = & \frac{1}{N} \sum_{v_1^o, v_2^s \in \mathcal{V}_{\text{batch}}} \| v_1^o - v_2^s \|_2^2 + \frac{\lambda_1}{c} \sum_{i=1}^{c} \sum_{j \neq i} W_{ij}'^2 \\ & + \frac{\lambda_2}{c} \sum_{i=1}^{c} \max(0, \gamma - \text{std}(v_1^o)_i), \end{aligned} \tag{10}$$

where $W' = \frac{1}{N-1} \sum_{v_1^o \in \mathcal{V}_{\text{batch}}} (v_1^o - \bar{v}_1^o)(v_1^o - \bar{v}_1^o)^T$ is the covariance matrix of the same view, $\text{std}(v)_i$ denotes the standard deviation of the $i$-th channel of $v$, $\gamma$ is a constant target value for it, and $\lambda_1$, $\lambda_2$ are balancing weights.

Similarly, its gradient can be derived as:

$$\begin{aligned} \frac{\partial L}{\partial u_1^o} = & \frac{2}{N} \Bigg( - u_2^s + \lambda \sum_{v_1^o \in \mathcal{V}_{\text{batch}}} \frac{\tilde{u}_1^{oT} \tilde{v}_1^o}{N} \tilde{v}_1^o \Bigg) \\ & + \underbrace{\frac{2\lambda}{N} \left( \frac{1}{\lambda} u_1^o - B \tilde{u}_1^o \right)}_{\text{reduce to } 0}, \end{aligned} \tag{11}$$

where $\tilde{v} = v - \bar{v}$ is the de-centered sample, $\lambda = \frac{2\lambda_1 N^2}{c(N-1)^2}$, and $B = \frac{N}{c\lambda(N-1)} (2\lambda_1 W'_{\text{diag}} + \frac{\lambda_2}{2} \text{diag}(\mathbb{1}(\gamma - \text{std}(v_1^o) > 0) \oslash \text{std}(v_1^o)))$. Here $\text{diag}(x)$ is a matrix with diagonal filled with the vector $x$, $\mathbb{1}(\cdot)$ is the indicator function, and $\oslash$ denotes element-wise division.

VICReg does not apply any normalization on $v$, and instead requires the de-center operation and standard deviation term in the loss function. We have verified that it is able to get rid of such terms by employing $\ell_2$ normalization on $v$ (see Table 2(j)). In this way, Eq.(11) can be reduced to only the first term without de-center operation.

**Unified Gradient.** Because $v^s$ and $v^o$ are mathematically equivalent, the gradient form of feature decorrelation family can be unified as:

$$\frac{\partial L}{\partial u_1^o} = \frac{2}{N} \Bigg( - u_2^t + \lambda \sum_{v^o \in \mathcal{V}_{\text{batch}}} \frac{u^{oT} v^o}{N} v_1^o \Bigg), \tag{12}$$

where the first term $-u_2^t$ acts as the positive gradient, the second term $\sum_{v^o \in \mathcal{V}_{\text{batch}}} (u^{oT} v^o / N) v_1^o$ is the negative gradient, and $\lambda$ is also a balance factor. The only difference between methods is the subscript for negative coefficient. Feature decorrelation methods actually work in a similar way with other self-supervised methods. The positive and negative gradient come from the diagonal and off-diagonal elements of the correlation matrix.

## 4. Key Factors in SSL

As we analyzed before, the gradients for different self-supervised learning methods share a similar formula:

$$\frac{\partial L}{\partial u_1^o} = \nabla L_p + \lambda \nabla L_n, \tag{13}$$

where the gradient consists of three components: the positive gradient $\nabla L_p$, the negative gradient $\nabla L_n$ and the balance factor $\lambda$. However, there are still differences on the specific form of these three components, and a natural question arises: will the gradient form affect the performance of self-supervised learning? Furthermore, although these methods share similar gradient formula, they usually differ from each other on the type of target branch and the construction of sample set $\mathcal{V}$. In this section, we shall conduct a thorough comparison between these methods and present the key factors that influence the final performance.

Although previous works have compared their methods with others, the training settings are usually different. To provide a fair comparison, we use a unified training and evaluation setting, in which only the loss function is changed. Our setting mainly follows [7] (see Appendix).

### 4.1. Gradient Form

We first explore how much difference the gradient form can make in different methods. For a fair comparison, the target branch adopts momentum encoder for all methods. The effect of target branch type will be discussed in Section 4.2. It should be noted that the we apply momentum

|  | Method | Norm | Pos Grad | Balance Factor | Neg Grad | Sample Set | Linear Eval |
|---|---|---|---|---|---|---|---|
| Contrastive learning methods |  |  |  |  |  |  |  |
| (a) | MoCo [16] | $\ell_2$ | $-u_2^m$ | 1 | $\sum_{v^m \in \mathcal{V}_{\text{bank}}} s_v v^m$ | $\mathcal{V}_{\text{bank}}$ | 70.0 |
| (b) | SimCLR* [5] | $\ell_2$ | $-u_2^m$ | 1 | $\sum_{v^m \in \mathcal{V}_{\text{batch}} \setminus u_1^o} s_v v^m$ | $\mathcal{V}_{\text{batch}} \setminus u_1^o$ | 70.0 |
| Asymmetric network methods |  |  |  |  |  |  |  |
| (c) | BYOL [20] | $\ell_2$ | - | - | - | - | 70.3 |
| (d) | BYOL(DirectPred [27]) | $\ell_2$ | $-W_h^T u_2^m$ | $\frac{u_1^{oT} W_h^T u_2^m}{u_1^{oT}(F+\epsilon^2 I)u_1^o}$ | $\sum_{v^o \in \mathcal{V}_\infty} (\rho_v u_1^{oT} v^o) v^o$ | $\mathcal{V}_\infty$ | 70.2 |
| (e) | - | $\ell_2$ | $-u_2^m$ | 100 | $\sum_{v^o \in \mathcal{V}_\infty} (\rho_v u_1^{oT} v^o) v^o$ | $\mathcal{V}_\infty$ | 70.3 |
| Feature decorrelation methods |  |  |  |  |  |  |  |
| (f) | Barlow Twins* [33] | BN | $-Au_2^m$ | $5\times10^{-3}$ | $\sum_{v_1^o, v_2^m \in \mathcal{V}_{\text{batch}}} \frac{u_2^{mT} v_2^m}{N} v_1^o$ | $\mathcal{V}_{\text{batch}}$ | 69.0 |
| (g) | - | BN | $-u_2^m$ | $5\times10^{-3}$ | $\sum_{v_1^o, v_2^m \in \mathcal{V}_{\text{batch}}} \frac{u_2^{mT} v_2^m}{N} v_1^o$ | $\mathcal{V}_{\text{batch}}$ | 69.7 |
| (h) | - | $\ell_2$ | $-u_2^m$ | 50 | $\sum_{v_1^o, v_2^m \in \mathcal{V}_{\text{batch}}} \frac{u_2^{mT} v_2^m}{N} v_1^o$ | $\mathcal{V}_{\text{batch}}$ | 70.0 |
| (i) | VICReg* [1] | - | $-u_2^m$ | $4\times10^{-5}$ | $\sum_{v_1^o \in \mathcal{V}_{\text{batch}}} \frac{u_1^{oT} v_1^o}{N} v_1^o + \frac{1}{\lambda} u_1^o - B\tilde{u}_1^o$ | $\mathcal{V}_{\text{batch}}$ | 70.0 |
| (j) | - | $\ell_2$ | $-u_2^m$ | 25 | $\sum_{v_1^o \in \mathcal{V}_{\text{batch}}} \frac{u_1^{oT} v_1^o}{N} v_1^o$ | $\mathcal{V}_{\text{batch}}$ | 69.8 |

Table 2. Performance comparison for different methods on ImageNet [11]. "Norm" denotes the normalization applied to the representations before loss calculation. "Pos Grad", "Balance Factor" and "Neg Grad" correspond to the components in Eq. (13). Linear evaluation follows the protocol in [7]. *Note that momentum encoder is used for target branch.

encoder in the loss form and derive the corresponding gradients, so some negative gradient forms do not contain $v^m$.

Specifically, we first try to compare and simplify the gradient form within each type of method. This can filter out irrelevant elements at early stage and make the comparison more clear. After that we can compare these methods all together. Because the scales of positive and negative gradients can vary a lot during simplification, we search for the best balance factor for each combination.

**Simplification for Contrastive Learning.** Table 2(ab) report the performance of different contrastive learning methods. Original MoCo [16] is used in Table 2(a). Because momentum encoder is applied to SimCLR [5] in Table 2(b), the second term in Eq.(3) naturally diminishes. These two methods show nearly no differences on the final results.

We also note that SimCLR uses $\mathcal{V}_{\text{batch}}$ rather than $\mathcal{V}_{\text{bank}}$ as in MoCo, but there is only minor difference. This suggests that with proper training setting, larger number of negative samples may not be necessary for good performance.

**Simplification for Asymmetric Network.** Table 2(c-e) give the simplification results for asymmetric network methods. The original BYOL [20] and the gradient version of BYOL with DirectPred [27] form are presented in Table 2(cd), respectively, whose results are consistent with the conclusion of [27]. SimSiam [7] is not presented here, because its momentum encoder variant is just BYOL.

In Table 2(e) we substitute $W_h$ in the positive gradient with identity matrix, and reduce the dynamic balance factor to a constant scalar. Such replacement does not lead to performance degradation. Therefore, the gradient form of asymmetric network methods can be unified as Table 2(e).

**Simplification for Feature Decorrelation.** We demonstrate the results of feature decorrelation methods in Table 2(f-j). For Barlow Twins [33], the matrix $A$ in the positive gradient of Table 2(f) is first substituted with identity matrix in Table 2(g). The results imply that this will not harm the performance. In Table 2(h), batch normalization is then replaced with $\ell_2$ normalization, and no accuracy decrease is observed.

For VICReg [1], we report its result in Table 2(i). In Table 2(j), $\ell_2$ normalization is applied to the representation, and the $\lambda_1 u_1 - B\tilde{u}_1$ term is removed from negative gradient. Such simplification produces similar result.

In the end, Table 2(hj) only differ in how to calculate negative coefficients. The comparison indicates that similar performances can be obtained. Thus, the gradient form of feature decorrelation methods can be unified as Table 2(j).

**Comparison between Different Methods.** Finally, we can compare different kinds of methods with their unified gradient form, *i.e.*, Table 2(bej). Among three components of gradient, they share the same positive gradient, the balance factor is searched for the best one, and the only difference is the negative gradient. Table 2 shows that the performance gap between different methods is actually minor (<0.5% points). What's more, asymmetric network methods are similar with feature decorrelation methods on gradient form, but utilize $\mathcal{V}_\infty$ instead of $\mathcal{V}_{\text{batch}}$. This implies the construction of $\mathcal{V}$ is not vital for self-supervised learning.

## 4.2. Target Branch Type

The type of target branch is distinct for different methods in their original implementation. In Section 4.1, we adopts

| Pos Grad | Neg Grad | Contrastive Learning Table 2(b) | Asymmetric Network Table 2(e) | Feature Decorrelation Table 2(j) |
|---|---|---|---|---|
| stop gradient | | 67.6 | 67.9 | 67.6 |
| momentum | | 70.0 | 70.2 | 69.8 |
| momentum | stop gradient | 70.1 | 70.3 | 69.8 |

Table 3. Effect of target branch type. We report ImageNet [11] linear evaluation accuracy after 100-epoch pre-training.

momentum encoder for all methods. Now, we study the effect of different target branch types in Table 3. There can be three choices for the target branch: weight-sharing, stop-gradient and momentum-encoder. We use the unified form (*i.e.*, Table 2(bej)) as representatives for these three kinds of methods, and change the target branch type. Because a symmetric loss is always employed, the weight-sharing and stop-gradient variants of the gradient form are actually the same. We omit the weight-sharing variant for simplicity.

For the stop-gradient target branch type, the results for different self-supervised learning methods are very similar, which is consistent with the conclusion in Section 4.1. For the momentum-encoder target branch type, it can improve the performance of all three kinds of methods with $\sim$ 2% points compared to the stop-gradient target branch type. This shows that momentum encoder is beneficial for these self-supervised learning methods, and can provide a consistent performance gain.

We further consider which part of gradient the momentum encoder has effect on. To achieve this, we only adopt momentum encoder output for the positive gradient. Table 3 indicates that it's enough to apply momentum encoder to the positive gradient. This suggests that a consistent and slow-updating positive goal may be very important for self-supervised learning.

## 5. A Concise Gradient Form for SSL

### 5.1. UniGrad

The comparison between gradients of different methods leads us to find a concise but effective gradient form for self-supervised learning. The proposed gradient, named UniGrad, can be represented as

$$\frac{\partial L}{\partial u_1^o} = -u_2^m + \lambda F u_1^o, \tag{14}$$

where $F = \sum_{v^o \in \mathcal{V}_\infty} \rho_v v^o v^{oT}$. Note that this gradient form is exactly the one described in Table 2(e), which achieves competitive results with a simple way to utilize positive and negative samples.

To fully understand this gradient, we give analysis through its corresponding object function:

$$L = \mathop{\mathbb{E}}_{u_1,u_2}\left[ -\cos(u_1^o, u_2^m) + \frac{\lambda}{2} \sum_{v^o \in \mathcal{V}_\infty} \rho_v \cos^2(u_1^o, v^o) \right], \tag{15}$$

where $\lambda$ is set to 100 as default. The objective function consists of two terms. The first term maximizes the cosine similarity between positive samples, which encourages modeling the invariance with respect to data augmentations. The second term expects the similariry between negative samples close to zero so as to avoid representational collapse.

**Relation to Contrastive Learning.** Compare to the InfoNCE [25] used in MoCo [6, 16] and SimCLR [5], UniGrad expects the similarity with negative samples close to zero to avoid collapse, while the InfoNCE encourages the similarity with negative samples to be lower than that with positive samples as much as possible. Moreover, UniGrad could encode infinite negative samples via a correlation matrix with less memory cost compared to a memory bank.

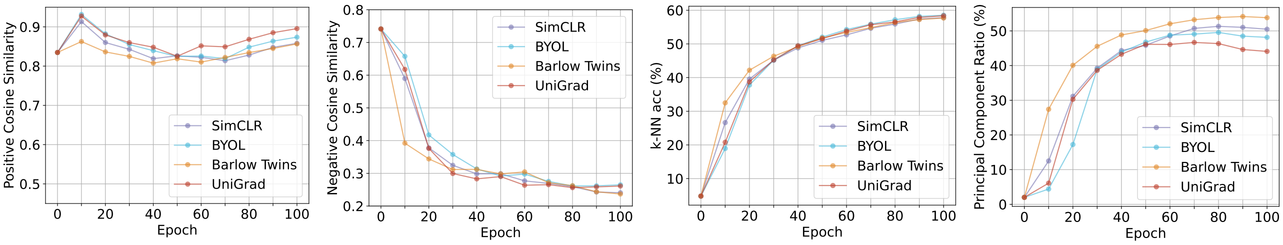
**Relation to Asymmetric Network.** Compare to BYOL [20] and SimSiam [7], our method could learn meaningful representations without the need of a predictor, thus gets rid of additional optimization tricks (usually a larger learning rate is needed for the predictor) and potential influence of the design of this predictor. Compare to DirectPred [27] with a optimization-free predictor, UniGrad removes the need for SVD decomposition.

**Relation to Feature Decorrelation.** Compare to Barlow Twins [33] and VICReg [1], UniGrad could achieve a similar effect to decorrelate different channels without direct optimization of the covariance or cross-correlation matrix (see Figure 2). In addition, our method uses $\ell_2$ normalization instead of batch normalization or extra restrictions on the variance of each channel.

**Discussion.** Since we have observed close performance achieved by UniGrad and other methods in Table 2, we wonder if the representations learned via various losses could end up with similar properties. In Figure 2, We compare the learning trajectory of different methods from the aspects of the similarity between positive/negative pairs, the k-NN accuracy and the degree of feature decorrelation. We find that there is no significant difference between UniGrad and other methods. The result implies that these methods work in a similar mechanism, which coincides with the comparison of their gradients in Section 4. For instance, SimCLR and BYOL can also learn to decorrelate different channels and Barlow Twins can learn to discriminate positive and negative samples as well. Besides its competitive performance, our method works as a concise version connected to these three kinds of methods without complicated components, such as memory bank and predictor.

### 5.2. Application on Data Augmentations

Benefiting from its concise form, UniGrad can be easily extended with commonly used data augmentations [3, 4, 23, 29, 32, 34] to further boost its performance. As a



(a) positive cosine similarity (b) negative cosine similarity (c) k-NN accuracy (d) principal component ratio

Figure 2. Learning trajectory for different methods. The metric of principal component ratio is to evaluate feature decorrelation degree. We apply PCA to representations, and count the number of eigenvalues whose cumulative sum first exceeds 90%.

| Method | Epoch | Time | Linear Eval |
|---|---|---|---|
| UniGrad | 100 | 38.2h | 70.3 |
| UniGrad+CutMix | 100 | 38.2h | 71.2 |
| UniGrad+multi-crop | 100 | 114.6h | 71.7 |
| UniGrad+CutMix+multi-crop | 100 | 114.6h | 72.3 |

Table 4. Ablation on CutMix and multi-crop.

| Method | Epoch | Linear Eval |
|---|---|---|
| MoCov2 [6] | 800 | 71.1 |
| SimCLR [5] | 1000 | 69.3 |
| BYOL [20] | 1000 | 74.3 |
| SimSiam [7] | 800 | 71.3 |
| Barlow Twins [33] | 1000 | 73.2 |
| VICReg [1] | 1000 | 73.2 |
| DINO (+multi-crop) [4] | 800 | 75.3 |
| TWIST (+multi-crop) [28] | 800 | 75.5 |
| UniGrad+CutMix | 800 | 74.9 |
| UniGrad+CutMix+multi-crop | 800 | 75.5 |

Table 5. Linear classification on ImageNet [11].

demonstration, we show how to apply CutMix [23, 32] and multi-crop [3, 4] to our method below.

*CutMix* generates new samples by replacing a randomly selected image region with a patch from another image. Given a batch of images, we cut patches from this batch with shuffled order, and paste them onto the original batch. For these mixed images, their positive gradients are calculated from the normal images and then mixed according to the mixup ratio. $F$ is calculated from normal images only.

*Multi-crop* samples additional smaller-sized crops to increase the number of views of an image. Specifically, we use $2 \times 224$ global views and $6 \times 96$ local views, with global scale set to $(0.4, 1)$ and local scale set to $(0.05, 0.4)$ respectively. For each global view, its positive gradient comes from the other global view. For each local view, its positive samples consist of the average of two global views. $F$ is calculated from global views only.

**Ablation Study.** We first conduct ablation study to validate the impact of CutMix and multi-crop on UniGrad under the experiments setting described in Section 4. As shown in Table 4, CutMix and multi-crop achieve an improvement of 0.9% and 1.4% respectively, and combining these two strategies together boosts the improvement to 2.0%. We also report the training time in Table 4. The implementation of CutMix only adds negligible training overhead compared to normal training, while multi-crop introduces a relatively heavy training cost. These variants can be used according to the available computational resources.

**More Training Epoches.** We evaluate the performance of our method with more training epoches. We adopt another set of training setting for faster pretraining (see Appendix). The linear evaluation setting follows Section 4. Table 5 compares our results with previous methods. UniGrad with CutMix can already surpass other methods that do not use multi-crop. By further employing multi-crop, it shows comparable performance with current state-of-the-art methods. We also transfer the pre-trained model to downstream tasks, including semi-supervised learning on ImageNet [11] and object detection on PASCAL VOC [13] and COCO [24]. Our model is able to achieve competitive results with other leading methods (see Appendix).

## 6. Conclusion

In this paper, we present a unified framework for three typical self-supervised learning methods from the perspective of gradient analysis. While previous works appear to be distinct in their loss functions, we demonstrate that they share a similar gradient form. Such form consists of the positive gradient, the negative gradient and the balance factor, which suggests that these methods work in a similar mechanism. We further compare their performances under a fair experiment setting. It's shown that they can deliver similar performances, and momentum encoder is the key factor to boost performance. Finally, we propose UniGrad, a simple but effective gradient form for self-supervised learning. Extensive experiments have shown its effectiveness in linear evaluation and downstream tasks.

**Limitations.** This work only adopts linear evaluation for performance comparison, while different methods may have a different impact on downstream tasks, *e.g.*, object detection and semantic segmentation. We leave the transfer learning performance comparison for future work.

**Potential Negative Societal Impact.** This work may inherit the negative impacts of self-supervised learning. Because a large-scale training is usually required, it may consume lots of electricity and cause environmental pollution. This method also learns representations from training dataset and may contain data biases. Future work can seek for a more efficient and unbiased training method.

**Acknowledgments** The work is supported by the National Key R&D Program of China (2020AAA0105200), Beijing Academy of Artificial Intelligence.

## References

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

[2] Yue Cao, Zhenda Xie, Bin Liu, Yutong Lin, Zheng Zhang, and Han Hu. Parametric instance classification for unsupervised visual feature learning. In *NeurIPS*, 2020.

[3] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *NeurIPS*, 2020.

[4] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, 2021.

[5] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.

[6] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

[7] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *CVPR*, 2021.

[8] Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised vision transformers. *arXiv preprint arXiv:2104.02057*, 2021.

[9] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *CVPR*, 2005.

[10] ImageNet contributors. Imagenet terms of access. https://image-net.org/download, 2020.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[12] Aleksandr Ermolov, Aliaksandr Siarohin, Enver Sangineto, and Nicu Sebe. Whitening for self-supervised representation learning. In *ICML*, 2021.

[13] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. In *IJCV*, 2010.

[14] Inc. Flickr. Flickr terms & conditions of use. http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/, 2020.

[15] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[16] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[18] Qianjiang Hu, Xiao Wang, Wei Hu, and Guo-Jun Qi. Adco: Adversarial contrast for efficient learning of unsupervised representations from self-trained negative adversaries. In *CVPR*, 2021.

[19] Tianyu Hua, Wenxiao Wang, Zihui Xue, Yue Wang, Sucheng Ren, and Hang Zhao. On feature decorrelation in self-supervised learning. In *ICCV*, 2021.

[20] Grill Jean-Bastien, Strub Florian, Altché Florent, Tallec Corentin, Pierre Richemond H., Buchatskaya Elena, Doersch Carl, Bernardo Pires Avila, Zhaohan Guo Daniel, Mohammad Azar Gheshlaghi, Piot Bilal, Kavukcuoglu Koray, Munos Rémi, and Valko Michal. Bootstrap your own latent - a new approach to self-supervised learning. In *NeurIPS*, 2020.

[21] Yannis Kalantidis, Mert Bulent Sariyildiz, Noe Pion, Philippe Weinzaepfel, and Diane Larlus. Hard negative mixing for contrastive learning. In *NeurIPS*, 2020.

[22] Junnan Li, Pan Zhou, Caiming Xiong, and Steven CH Hoi. Prototypical contrastive learning of unsupervised representations. In *ICLR*, 2021.

[23] Suichan Li, Dongdong Chen, Yinpeng Chen, Lu Yuan, Lei Zhang, Qi Chu, Bin Liu, and Nenghai Yu. Unsupervised finetuning. *arXiv preprint arXiv:2110.09510*, 2021.

[24] 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 *ECCV*, 2014.

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

[26] Pierre H Richemond, Jean-Bastien Grill, Florent Altché, Corentin Tallec, Florian Strub, Andrew Brock, Samuel Smith, Soham De, Razvan Pascanu, Bilal Piot, et al. Byol works even without batch statistics. *arXiv preprint arXiv:2010.10241*, 2020.

[27] Yuandong Tian, Xinlei Chen, and Surya Ganguli. Understanding self-supervised learning dynamics without contrastive pairs. In *ICML*, 2021.

[28] Feng Wang, Tao Kong, Rufeng Zhang, Huaping Liu, and Hang Li. Self-supervised learning by estimating twin class distributions. *arXiv preprint arXiv:2110.07402*, 2021.

[29] Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *ICML*, 2020.

[30] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification.

*Journal of machine learning research*, 10(2), 2009.

[31] Yang You, Igor Gitman, and Boris Ginsburg. Scaling sgd batch size to 32k for imagenet training. *arXiv preprint arXiv:1708.03888*, 6:12, 2017.

[32] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, 2019.

[33] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *ICML*, 2021.

[34] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018.

[35] Rui Zhu, Bingchen Zhao, Jingen Liu, Zhenglong Sun, and Chang Wen Chen. Improving contrastive learning by visualizing feature transformation. In *ICCV*, 2021.

# A. Gradient Analysis

## A.1. Contrastive Learning Methods

**Derivation of the gradient for MoCo [16].** For simplicity, we denote $l(u_1^o)$ as the InfoNCE loss for the sample $u_1^o$:

$$l(u_1^o) = -\log \frac{\exp\left(\cos(u_1^o, u_2^m)/\tau\right)}{\sum_{v^m \in \mathcal{V}_{\text{bank}}} \exp\left(\cos(u_1^o, v^m)/\tau\right)}. \tag{16}$$

Let $l(u_1^o) = -\log s_{u_2}$, $s_{u_2} = \frac{\exp(c_{u_2^m})}{\sum_{v^m \in \mathcal{V}_{\text{bank}}} \exp(c_{v^m})}$, $c_v = \cos(u_1^o, v)/\tau$. According to the chain rule, we have

$$\begin{aligned}
\frac{\partial l(u_1^o)}{\partial u_1^o} &= \frac{\partial l(u_1^o)}{\partial s_{u_2}} \cdot \frac{\partial s_{u_2}}{\partial c_{u_2^m}} \cdot \frac{\partial c_{u_2^m}}{\partial u_1^o} \\
&\quad + \sum_{v^m \in \mathcal{V}_{\text{bank}} \setminus u_2^m} \frac{\partial l(u_1^o)}{\partial s_{u_2}} \cdot \frac{\partial s_{u_2}}{\partial c_{v^m}} \cdot \frac{\partial c_{v^m}}{\partial u_1^o} \\
&= -\frac{1}{s_{u_2}} \cdot s_{u_2}(1 - s_{u_2}) \cdot \frac{u_2^m}{\tau} \\
&\quad - \sum_{v^m \in \mathcal{V}_{\text{bank}} \setminus u_2^m} \frac{1}{s_{u_2}} \cdot s_{u_2} s_v \cdot \frac{v^m}{\tau} \\
&= -\frac{u_2^m}{\tau} + \sum_{v^m \in \mathcal{V}_{\text{bank}}} s_v \frac{v^m}{\tau},
\end{aligned} \tag{17}$$

where $s_v = \frac{\exp\left(\cos(u_1^o, v^m)/\tau\right)}{\sum_{y^m \in \mathcal{V}_{\text{bank}}} \exp\left(\cos(u_1^o, y^m)/\tau\right)}$.

Denote $L$ as the averaged $l(\cdot)$ over a batch of N samples, its gradient w.r.t $u_1^o$ is

$$\frac{\partial L}{\partial u_1^o} = \frac{1}{N} \frac{\partial l(u_1^o)}{\partial u_1^o} = \frac{1}{\tau N}\left(-u_2^m + \sum_{v^m \in \mathcal{V}_{\text{bank}}} s_v v^m\right). \tag{18}$$

**Derivation of the gradient for SimCLR [5].** For SimCLR, the InfoNCE loss $l(u_1^o)$ should be modified as

$$l(u_1^o) = -\log \frac{\exp\left(\cos(u_1^o, u_2^s)/\tau\right)}{\sum_{v^s \in \mathcal{V}_{\text{batch}} \setminus u_1^o} \exp\left(\cos(u_1^o, v^s)/\tau\right)}. \tag{19}$$

Note that because the target branch is not detached from back-propagation, $u_1^o$ can receive gradients from $l(u_2^s)$ and $l(v^s)$. Accordingly, the gradient can be derived as

$$\begin{aligned}
\frac{\partial L}{\partial u_1^o} &= \frac{1}{N}\left(\frac{\partial l(u_1^o)}{\partial u_1^o} + \frac{\partial l(u_2^s)}{\partial u_1^o} + \sum_{v^s \in \mathcal{V}_{\text{batch}} \setminus \{u_1^o, u_2^s\}} \frac{\partial l(v^s)}{\partial u_1^o}\right) \\
&= \frac{1}{\tau N}\left(-u_2^s + \sum_{v^s \in \mathcal{V}_{\text{batch}} \setminus u_1^o} s_v v^s\right) \\
&\quad + \frac{1}{\tau N}\left(-u_2^s + t_{u_2} u_2^s\right) + \frac{1}{\tau N} \sum_{v^s \in \mathcal{V}_{\text{batch}} \setminus \{u_1^o, u_2^s\}} t_v v^s \\
&= \frac{1}{\tau N}\left(-u_2^s + \sum_{v^s \in \mathcal{V}_{\text{batch}} \setminus u_1^o} s_v v^s\right) \\
&\quad + \frac{1}{\tau N}\left(-u_2^s + \sum_{v^s \in \mathcal{V}_{\text{batch}} \setminus u_1^o} t_v v^s\right),
\end{aligned} \tag{20}$$

where $t_v = \frac{\exp\left(\cos(v^s, u_1^o)/\tau\right)}{\sum_{y^s \in \mathcal{V}_{\text{batch}} \setminus v^s} \exp\left(\cos(v^s, y^s)/\tau\right)}$. If we stop the gradient from $l(u_2^s)$ and $l(v^s)$, Eq.(20) will reduce to

$$\frac{\partial L}{\partial u_1^o} \approx \frac{1}{\tau N}\left(-u_2^s + \sum_{v^s \in \mathcal{V}_{\text{batch}} \setminus u_1^o} s_v v^s\right), \tag{21}$$

which shares a similar structure with that of MoCo. We demonstrate empirically that this simplification dose no harm to the performance as shown in Table 6.

## A.2. Asymmtric Network Methods

**Derivation of the gradient for DirectPred [27].** DirectPred takes the negative cosine similarity loss between target sample and projected online sample:

$$l(u_1^o) = -\cos\left(\frac{W_h u_1^o}{||W_h u_1^o||_2}, u_2^t\right), \tag{22}$$

$$W_h = U \Lambda_h U^T, \;\; \Lambda_h = \Lambda_F^{1/2} + \epsilon \lambda_{max} I, \tag{23}$$

where $U$ and $\Lambda_F$ are the eigenvectors and eigenvalues of $F = \sum_{v^o \in \mathcal{V}_\infty} \rho_v v^o {v^o}^T$, respectively. $\epsilon$ is a hyperparameter to boost small eigenvalues.

Denote $y_1 = W_h u_1^o$, $y_1^n = \frac{y_1}{||y_1||_2}$, the gradient of $L$ can be derived as:

$$\begin{aligned}
\frac{\partial L}{\partial u_1^o} &= \frac{1}{N}\left(\frac{\partial y_1}{\partial u_1^o} \cdot \frac{\partial y_1^n}{\partial y_1} \cdot \frac{\partial l}{\partial y_1^n}\right) \\
&= \frac{1}{N}\left(-W_h^T \cdot \frac{1}{||y_1||_2}\left(I - \frac{y_1 y_1^T}{y_1^T y_1}\right) \cdot u_2^t\right) \\
&= \frac{1}{||W_h u_1^o||_2 N}\left(-W_h^T\left(I - \frac{W_h u_1^o {u_1^o}^T W_h^T}{{u_1^o}^T W_h^T W_h u_1^o}\right) u_2^t\right) \\
&= \frac{1}{||W_h u_1^o||_2 N}\left(-W_h^T u_2^t + \frac{{u_1^o}^T W_h^T u_2^t}{{u_1^o}^T W_h^T W_h u_1^o} W_h^T W_h u_1^o\right).
\end{aligned} \tag{24}$$

Note that

$$\begin{aligned}
W_h^T W_h &= U \Lambda_h^T U^T U \Lambda_h U^T \\
&= U(\Lambda_F + 2\epsilon \lambda_{max} \Lambda_F^{1/2} + \epsilon^2 \lambda_{max}^2 I) U^T \\
&= F + 2\epsilon \lambda_{max} F^{1/2} + \epsilon^2 \lambda_{max}^2 I.
\end{aligned} \tag{25}$$

Substituting Eq.(25) into Eq.(24) leads to

$$\begin{aligned}
\frac{\partial L}{\partial u_1^o} &= \frac{1}{||W_h u_1^o||_2 N}\Big(-W_h^T u_2^t \\
&\quad + \tilde{\lambda}(F u_1^o + 2\epsilon \lambda_{max} F^{1/2} u_1^o + \epsilon^2 \lambda_{max}^2 u_1^o)\Big),
\end{aligned} \tag{26}$$

where $\tilde{\lambda} = \frac{{u_1^o}^T W_h^T u_2^t}{{u_1^o}^T (F + 2\epsilon \lambda_{max} F^{1/2} + \epsilon^2 \lambda_{max}^2 I) u_1^o}$. We have verified that removing the $F^{1/2}$ term will not cause performance drop (see Table 6). Thus, the gradient can be simplified into

$$\frac{\partial L}{\partial u_1^o} \approx \frac{1}{||W_h u_1^o||_2 N}\left(-W_h^T u_2^t + \lambda(F u_1^o + \epsilon^2 \lambda_{max}^2 u_1^o)\right), \tag{27}$$

where $\lambda = \frac{{u_1^o}^T W_h^T u_2^t}{{u_1^o}^T (F + \epsilon^2 \lambda_{max}^2 I) u_1^o}$.

When $u_1^o$ is $\ell_2$ normalized, we can further neglect the $\epsilon^2 \lambda_{max}^2 u_1^o$ term, because the component of this gradient along the direction of $u_1^o$ will take no effect. Hence, we simplify the gradient as

$$\begin{aligned}\frac{\partial L}{\partial u_1^o} &\approx \frac{1}{||W_h u_1^o||_2 N}\Big( - W_h^T u_2^t + \lambda F u_1^o\Big) \\ &= \frac{1}{||W_h u_1^o||_2 N}\Big( - W_h^T u_2^t + \lambda \sum_{v^o \in \mathcal{V}_\infty} (\rho_v {u_1^o}^T v^o) v^o\Big).\end{aligned} \tag{28}$$

| Method | SimCLR [5] | | DirectPred [27] | |
|---|---|---|---|---|
| Gradient | Eq.(20) | Eq.(21) | Eq.(26) | Eq.(27) |
| Linear Eval | 67.5 | 67.6 | 70.2 | 70.2 |

Table 6. Simplification for the gradient of SimCLR and DirectPred. We use the 100-epoch pre-training and lineal evaluation protocol described in Appendix B.

### A.3. Feature Decorrelation Methods

**Derivation of the gradient for Barlow Twins [33].** Barlow Twins forces the cross-correlation matrix to be close to the identity matrix via the following loss function:

$$L = \sum_{i=1}^{C} (W_{ii} - 1)^2 + \lambda \sum_{i=1}^{C} \sum_{j \neq i} W_{ij}^2, \tag{29}$$

where $W = \frac{1}{N} \sum_{v_1^o, v_2^s \in \mathcal{V}_{\text{batch}}} v_1^o {v_2^s}^T$ is the cross-correlation matrix.

Denote $L_1 = \sum_{i=1}^{C} (W_{ii} - 1)^2$, $L_2 = \lambda \sum_{i=1}^{C} \sum_{j \neq i} W_{ij}^2$. We use the operator $(\cdot)_k$ to represent the $k$-th element of a vector. For $L_1$, We have:

$$\frac{\partial L_1}{\partial (u_1^o)_k} = \frac{\partial L_1}{\partial W_{kk}} \cdot \frac{\partial W_{kk}}{\partial (u_1^o)_k} = 2(W_{kk} - 1) \cdot \frac{(u_2^s)_k}{N}. \tag{30}$$

For $L_2$, we have:

$$\begin{aligned}\frac{\partial L_2}{\partial (u_1^o)_k} &= \lambda \sum_{j \neq k}^{C} 2 \frac{\partial L_2}{\partial W_{kj}} \cdot \frac{\partial W_{kj}}{\partial (u_1^o)_k} = \lambda \sum_{j \neq k}^{C} 2 W_{kj} \cdot \frac{(u_2^s)_j}{N} \\ &= \frac{2\lambda}{N}\Big( - W_{kk} (u_2^s)_k + \sum_{j=1}^{C} W_{kj} (u_2^s)_j \Big) \\ &= \frac{2\lambda}{N}\Big( - W_{kk} (u_2^s)_k + \sum_{j=1}^{C} \frac{1}{N} \sum_{v_1^o, v_2^s \in \mathcal{V}_{\text{batch}}} (v_1^o)_k (v_2^s)_j (u_2^s)_j \Big) \\ &= \frac{2\lambda}{N}\Big( - W_{kk} (u_2^s)_k + \sum_{v_1^o, v_2^s \in \mathcal{V}_{\text{batch}}} \frac{1}{N} (v_1^o)_k \sum_{j=1}^{C} (v_2^s)_j (u_2^s)_j \Big) \\ &= \frac{2\lambda}{N}\Big( - W_{kk} (u_2^s)_k + \sum_{v_1^o, v_2^s \in \mathcal{V}_{\text{batch}}} \frac{{u_2^s}^T v_2^s}{N} (v_1^o)_k \Big).\end{aligned} \tag{31}$$

Combining Eq.(30) and Eq.(31) together, we get:

$$\frac{\partial L}{\partial u_1^o} = \frac{2}{N}\Big( - A u_2^s + \lambda \sum_{v_1^o, v_2^s \in \mathcal{V}_{\text{batch}}} \frac{{u_2^s}^T v_2^s}{N} v_1^o \Big), \tag{32}$$

where $A = I - (1 - \lambda) W_{\text{diag}}$. Here $(W_{\text{diag}})_{ij} = \delta_{ij} W_{ij}$ is the diagonal matrix of $W$, where $\delta_{ij}$ is the Kronecker delta.

**Derivation of the gradient for VICReg [1].** The loss function of VICReg consists of three componets:

$$L_1 = \frac{1}{N} \sum_{v_1^o, v_2^s \in \mathcal{V}_{\text{batch}}} ||v_1^o - v_2^s||_2^2, \tag{33}$$

$$L_2 = \frac{\lambda_1}{C} \sum_{i=1}^{C} \sum_{j \neq i}^{C} W_{ij}'^2, \tag{34}$$

$$L_3 = \frac{\lambda_2}{C} \sum_{i=1}^{C} \max(0, \gamma - \text{std}(v_1^o)_i), \tag{35}$$

where $W' = \frac{1}{N-1} \sum_{v_1^o \in \mathcal{V}_{\text{batch}}} (v_1^o - \bar{v}_1^o)(v_1^o - \bar{v}_1^o)^T$.

For the invariance term $L_1$, we have:

$$\frac{\partial L_1}{\partial (u_1^o)_k} = \frac{2}{N} (u_1^o - u_2^s)_k. \tag{36}$$

For the covariance term $L_2$, we have:

$$\begin{aligned}\frac{\partial L_2}{\partial (u_1^o)_k} &= \frac{2\lambda_1}{C} \sum_{j \neq k}^{C} \frac{\partial L_2}{\partial W'_{kj}} \cdot \frac{\partial W'_{kj}}{\partial (u_1^o)_k} \\ &= \frac{4\lambda_1}{C} \sum_{j \neq k}^{C} W'_{kj} \frac{(u_1^o - \bar{v}_1^o)_j}{N - 1} \\ &= \frac{4\lambda_1}{C(N-1)}\Big( - W'_{kk} (u_1^o - \bar{v}_1^o)_k + \sum_{j=1}^{C} W'_{kj} (u_1^o - \bar{v}_1^o)_j \Big) \\ &= \frac{4\lambda_1}{C(N-1)}\Big( - W'_{kk} (u_1^o - \bar{v}_1^o)_k \\ &\quad + \sum_{j=1}^{C} \sum_{v_1^o \in \mathcal{V}_{\text{batch}}} \frac{(u_1^o - \bar{v}_1^o)^T (v_1^o - \bar{v}_1^o)}{N - 1} (v_1^o - \bar{v}_1^o)_j \Big) \\ &= \frac{4\lambda_1 N}{C(N-1)^2}\Big( - \frac{N-1}{N} W'_{kk} (\tilde{u}_1^o)_k + \sum_{v_1^o \in \mathcal{V}_{\text{batch}}} \frac{{\tilde{u}_1^o}{}^T \tilde{v}_1^o}{N} (\tilde{v}_1^o)_j \Big) \\ &= \frac{2\lambda}{N}\Big( - \frac{N-1}{N} W'_{kk} (\tilde{u}_1^o)_k + \sum_{v_1^o \in \mathcal{V}_{\text{batch}}} \frac{{\tilde{u}_1^o}{}^T \tilde{v}_1^o}{N} (\tilde{v}_1^o)_j \Big),\end{aligned} \tag{37}$$

where $\lambda = \frac{2\lambda_1 N^2}{C(N-1)^2}$ and $\tilde{v} = v - \bar{v}$ is the de-centered sample.

For the variance term $L_3$, we have:

$$\begin{aligned}\frac{\partial L_3}{\partial (u_1^o)_k} &= \frac{\lambda_2}{C} \frac{\partial \max(0, \gamma - \text{std}(v_1^o)_k)}{\partial \text{std}(v_1^o)_k} \cdot \frac{\partial \text{std}(v_1^o)_k}{\partial (u_1^o)_k} \\ &= - \frac{\lambda_2}{C(N-1)} \mathbb{1}(\gamma - \text{std}(v_1^o)_k > 0) \frac{(\tilde{u}_1^o)_k}{\text{std}(v_1^o)_k}.\end{aligned} \tag{38}$$

For final loss function $L = L_1 + L_2 + L_3$, its gradient w.r.t $u_1^o$ can be represented as:

$$\begin{aligned}\frac{\partial L}{\partial u_1^o} &= \frac{2}{N}(u_1^o - u_2^s) - \frac{2\lambda}{N}\left(\frac{N-1}{N}W'_{\text{diag}}\tilde{u}_1^o - \sum_{v_1^o \in \mathcal{V}_{\text{batch}}} \frac{\tilde{u}_1^{oT}\tilde{v}_1^o}{N}\tilde{v}_1^o\right) \\ &\quad - \frac{\lambda_2}{C(N-1)}\text{diag}(\mathbb{1}(\gamma - \text{std}(v_1^o) > 0) \oslash \text{std}(v_1^o))\tilde{u}_1^o \\ &= \frac{2}{N}\left(-u_2^s + \lambda \sum_{v_1^o \in \mathcal{V}_{\text{batch}}} \frac{\tilde{u}_1^{oT}\tilde{v}_1^o}{N}\tilde{v}_1^o\right) + \frac{2\lambda}{N}\left(\frac{1}{\lambda}u_1^o - B\tilde{u}_1^o\right),\end{aligned} \tag{39}$$

where $B = \frac{N}{\lambda C(N-1)}(2\lambda_1 W'_{\text{diag}} + \frac{\lambda_2}{2}\text{diag}(\mathbb{1}(\gamma - \text{std}(v_1^o) > 0) \oslash \text{std}(v_1^o)))$. Here $W'_{\text{diag}}$ is the diagonal matrix of $W'$, $\text{diag}(x)$ is a matrix with diagonal filled with the vector $x$, $\mathbb{1}(\cdot)$ is the indicator function, and $\oslash$ denotes element-wise division.

# B. Implementation Details

We provide the experimental settings used in this paper. For 100 epochs pre-training and linear evaluation, we mainly follow [7]; For 800 epochs pre-training, large batch size is adopted for faster training and hence we mainly follow [20].

**Pre-training setting for 100 epochs.** SGD is used as the optimizer. The weight decay is $1.0 \times 10^{-4}$ and the momentum is 0.9. The learning rate is set according to linear scaling rule [15] as $base\_lr \times batch\_size/256$, with $base\_lr = 0.05$. The learning rate has a cosine decay schedule for 100 epochs with 5 epochs linear warmup. The batch size is set to 1024. We use ResNet50 [17] as the backbone. The projection MLP has three layers, with the hidden and output dimension set to 2048. BN and ReLU are applied after the first two layers. If a momentum encoder is used, we follow BYOL [20] to increase the exponential moving average parameter from 0.996 to 1 with a cosine scheduler.

**Pre-training setting for 800 epochs.** LARS [31] optimizer is used for 800 epochs pre-training with a batch size of 4096. The weight decay is $1.0 \times 10^{-6}$ and the momentum is 0.9. The learning rate is set with $base\_lr = 0.3$ for the weights and $base\_lr = 0.05$ for the biases and batch normalization parameters. Cosine decay schedule is used after a linear warm-up of 10 epochs. We exclude the biases and batch normalization parameters from the LARS adaptation and weight decay. For the projector, We use a three-layer MLP with hidden and output dimension set to 8192. Other configurations keep the same as the pre-training setting for 100 epochs.

**Linear evaluation.** We follow the common practice to adopt linear evaluation as the performance metric. Such practice trains a supervised linear classifier on top of the frozen features from pre-training. LARS [31] is used as the optimizer with weight decay as 0 and momentum as 0.9. The base learning rate is 0.02 with 4096 batch size, and a cosine decay schedule is used for 90 epochs.

| Depth | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Linear Eval | 60.7 | 65.2 | 70.3 | 70.0 | 69.8 |

Table 7. Effect of projector depth.

| Width | 1024 | 2048 | 4096 | 8192 | 16384 |
|---|---|---|---|---|---|
| Linear Eval | 68.3 | 70.3 | 70.5 | 70.9 | 71.2 |

Table 8. Effect of projector width.

| Method | 1% | | 10% | |
|---|---|---|---|---|
| | Top 1 | Top 5 | Top 1 | Top 5 |
| Supervised | 25.4 | 48.4 | 56.4 | 80.4 |
| SimCLR [5] | 48.3 | 75.5 | 65.6 | 87.8 |
| BYOL [20] | 53.2 | 78.4 | 68.8 | 89.0 |
| Barlow Twins [33] | 55.0 | 79.2 | 69.7 | 89.3 |
| DINO [4] | 52.2 | 78.2 | 68.2 | 89.1 |
| TWIST [28] | 61.2 | 84.2 | 71.7 | 91.0 |
| UniGrad | 60.8 | 83.8 | 71.5 | 90.6 |

Table 9. Semi-supervised learning on ImageNet.

# C. Additional Results

## C.1. Projector Structure

The design of projector is another main factor that influences the final performance and also varies across different works. [16] applies linear projection to contrastive learning. SimCLR [5] finds that a 2-layer MLP can help boost the performance. SimSiam [7] further extends the projector depth to 3. We explore the effects of different projector depths in Table 7. Here UniGrad with 100 epochs pre-training is used. The results show that increasing the depth of projector from 1 to 3 can greatly boost the linear evaluation accuracy. However, the improvement saturates when the projector becomes deeper.

Moreover, Barlow Twins [33] extends the dimension of projector from 2048 to 8192, showing notable improvement. We further study the effect of projector's width in Table 8. For simplicity, we change the output dimension together with the hidden dimension. UniGrad with 100 epochs pre-training is used. It's shown that increasing the projector width can steadily increase the performance, and does not seem to saturate even the dimension is increased to 16384.

| Method | VOC07+12 detection | | | COCO detection | | | COCO instance seg | | |
|---|---|---|---|---|---|---|---|---|---|
| | $\text{AP}_{\text{all}}$ | $\text{AP}_{50}$ | $\text{AP}_{75}$ | $\text{AP}^{\text{box}}_{\text{all}}$ | $\text{AP}^{\text{box}}_{50}$ | $\text{AP}^{\text{box}}_{75}$ | $\text{AP}^{\text{mask}}_{\text{all}}$ | $\text{AP}^{\text{mask}}_{50}$ | $\text{AP}^{\text{mask}}_{75}$ |
| Supervised | 54.7 | 84.5 | 60.8 | 38.9 | 59.6 | 42.7 | 35.4 | 56.5 | 38.1 |
| MoCov2 [6] | 56.4 | 81.6 | 62.4 | 39.8 | 59.8 | 43.6 | 36.1 | 56.9 | 38.7 |
| SimCLR [5] | 58.2 | 83.8 | 65.1 | 41.6 | 61.8 | 45.6 | 37.6 | 59.0 | 40.5 |
| DINO [4] | 57.2 | 83.5 | 63.7 | 41.4 | 62.2 | 45.3 | 37.5 | 58.8 | 40.2 |
| TWIST [28] | 58.1 | 84.2 | 65.4 | 41.9 | 62.6 | 45.7 | 37.9 | 59.7 | 40.6 |
| UniGrad | 57.8 | 84.0 | 64.9 | 42.0 | 62.6 | 45.7 | 37.9 | 59.7 | 40.7 |

Table 10. Transfer learning: object detection and instance segmentation. VOC benchmark uses Faster R-CNN with FPN. COCO benchmark uses Mask R-CNN with FPN. The supervised VOC results are run by us.

### C.2. Semi-supervised Learning

We finetune the pretrained model on the 1% and 10% subset of ImageNet's training set, following the standard protocol in [5]. The results are reported in Table 9. Compared with previous methods, UniGrad is able to obtain comparable results with [28] and obtain 5% and 1% improvement from other methods on the 1% and 10% subset, respectively.

### C.3. Transfer Learning

We also transfer the pretrained model to downstream stasks, including PASCAL VOC [13] object detection, COCO [24] object detection and instance segmentation. The model is finetuned in an end-to-end manner. Table 10 shows the final results. It can be seen that UniGrad delivers competitive transfer performance with other self-supervised learning methods, and surpasses the supervised baseline.

## D. Licenses of Assets

**ImageNet** [11] is subject to the ImageNet terms of access [10].

**PASCAL VOC** [13] uses images from Flickr, which is subject to the Flickr terms of use [14].

**COCO** [24]. The annotations are under the Creative Commons Attribution 4.0 License. The images are subject to the Flickr terms of use [14].