A. Gradient Analysis

A.1. Contrastive Learning Methods

Derivation of the gradient for MoCo [17]. For simplicity, we denote $l(u_1^o)$ as the InfoNCE loss for the sample u_1^o:

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

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

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

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

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

Algorithm 1 Pseudocode of MoCo in PyTorch style.

```
# u1, u2: normalized representations for two augmented
      views of shape [N, C]
# V_bank: the memory bank of shape [K, C]
# tau: the temperature coefficient

# positive term
loss_pos = -(u1*u2.detach()).sum(-1) # [N, 1]
# negative term
weight = softmax(u1@V_bank.T/tau, dim=-1) # [N, K]
loss_neg = (weight@V_bank)*v1.sum(-1) # [N, 1]
# MoCo
loss = 1/tau * (loss_pos + loss_neg).mean()
```

Derivation of the gradient for SimCLR [6]. For SimCLR, the InfoNCE loss $l(u_1^o)$ should be modified as

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

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

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

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

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

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

Algorithm 2 Pseudocode of Simplified SimCLR in PyTorch style.

```
# u1, u2: normalized representations for two augmented
      views of shape [N, C]
# tau: the temperature coefficient

# positive term
loss_pos = -(u1*u2.detach()).sum(-1) # [N, 1]
# negative term
weight = softmax(u1@u2.T/tau, dim=-1) # [N, N]
loss_neg = (weight@u2).detach()*u1.sum(-1) # [N, 1]
# simplified SimCLR
loss = 1/tau * (loss_pos + loss_neg).mean()
```

A.2. Asymmtric Network Methods

Derivation of the gradient for DirectPred [28]. DirectPred takes the negative cosine similarity loss between target sample and projected online sample:

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

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

where U and Λ_F are the eigenvectors and eigenvalues of $F = \sum_{v^o \in \mathcal{V}_\infty} \rho_v v^o v^{oT}$, respectively. ϵ 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:

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

$$= \frac{1}{||W_h u_1^o||_2 N}\left(- W_h^T u_2^t + \frac{{u_1^o}^T W_h^T u_2^t}{{u_1^o}^T W_h^T W_h u_1^o} W_h^T W_h u_1^o \right). \tag{24}$$

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

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

where $\tilde{\lambda} = \frac{{u_1^o}^T W_h^T u_2^t}{{u_1^o}^T (F + 2\epsilon\lambda_{max}F^{1/2} + \epsilon^2\lambda_{max}^2 I) u_1^o}$. For those three terms that are scaled by $\tilde{\lambda}$, we plot the value of their magnitude and the similarity of the first two terms in Figure 3(a). It's shown that the first two terms have highly similar direction so they are expected to have similar effect on the training. We have also 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 ℓ_2 normalized, we can further neglect the $\epsilon^2\lambda_{max}^2 u_1^o$ term, because the gradient propagated to unnormalized u_1^o is 0. Hence, we simplify the gradient as

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

Note that λ is a dynamic balance factor, but we find that its value tends to be quite stable (see Figure 3(b)), so it can also be substiuted by a constant scalar.

Method	SimCLR [6]		DirectPred [28]	
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.

Algorithm 3 Pseudocode of Asymetric Networks.

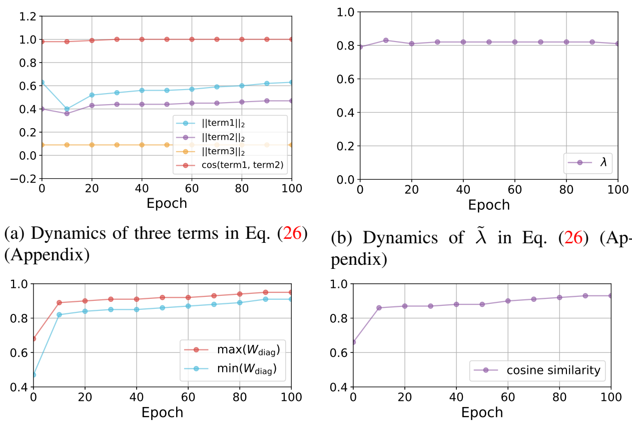
```
# u1, u2: normalized representations for two augmented
    views of shape [N, C]
# F: the moving average of correlation matrix
# rho: the moving average coefficient
# eps: hyperparameter to boost small eigenvalues

# accumulate F
tmp_F = ((u1.T@u1 + u2.T@u2) / (2*N)).detach()
F = rho*F + (1-rho)*tmp_F # update moving average

# calculate Wh
U, lambda_F, V = torch.svd(F)
lambda_h = torch.sqrt(lambda_F) + eps*lambda_F.max()
Wh = U@(torch.diag(lambda_h)@V) # [C, C]

# positive term
loss_pos = -(u1*(u2@Wh).detach()).sum(-1) # [N, 1]
# negative term
loss_neg = (u1*(u1@F).detach()).sum(-1) # [N, 1]

weight = 1/torch.linalg.norm(u1@Wh, dim=-1)
loss = (weight*(loss_pos + lambda * loss_neg)).mean()
```



(a) Dynamics of three terms in Eq. (26) (Appendix)

(b) Dynamics of $\tilde{\lambda}$ in Eq. (26) (Appendix)

(c) Dynamics of the max and min value of W_{diag} in Eq. (9)

(d) Dynamics of similarity between the neglected and negative terms in Eq. (11)

Figure 3. Justifications for simplifications.

A.3. Feature Decorrelation Methods

Derivation of the gradient for Barlow Twins [34]. 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:

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

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

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

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

Algorithm 4 Pseudocode of Barlow Twins in PyTorch style.

```
# u1, u2: representations for two augmented views of
    shape [N, C]
# lambda: the moving average coefficient

# correlation matrix
W_cor = u1.T@u2 / N # [C, C]
# positive term
pos = (1 - (1-lambda)*torch.diag(W_cor)) * u2
loss_pos = -(u1 * pos.detach()).sum(-1) # [N, 1]
# negative term
weight = u2@u2.T / N
loss_neg = ((weight@u1).detach() * u1).sum(-1)
# Barlow Twins
loss = 2 * (loss_pos + lambda * loss_neg).mean()
```

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

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

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

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

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:

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

Algorithm 5 Pseudocode of Simplified VICReg in PyTorch style.

```
# u1, u2: representations for two augmented views of
    shape [N, C]
# lambda: the moving average coefficient

# positive term
loss_pos = -(u1 * u2.detach()).sum(-1) # [N, 1]
# negative term
weight = u1@u1.T / N
loss_neg = ((weight@u1).detach() * u1).sum(-1)
# simplified VICReg
loss = 2 * (loss_pos + lambda * loss_neg).mean()
```

For the covariance term L_2, we have:

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

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:

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

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

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

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'_{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.

A.4. Pseudocode of UniGrad

Algorithm 6 Pseudocode of UniGrad in PyTorch style.

```
# u1, u2: normalized representations for two augmented
    views of shape [N, C]
# F: the moving average of correlation matrix
# rho: the moving average coefficient

# positive term
loss_pos = -(u1*u2.detach()).sum(-1) # [N, 1]
# negative term
tmp_F = ((u1.T@u1 + u2.T@u2) / (2*N)).detach()
F = rho*F + (1-rho)*tmp_F # update moving average
loss_neg = (u1@F)*u1.sum(-1) # [N, 1]
# UniGrad
loss = (loss_pos + lambda * loss_neg).mean()
```

B. Implementation Details

We provide the experimental settings used in this paper. For 100 epochs pre-training and linear evaluation, we mainly follow [8]; For 800 epochs pre-training, large batch size is adopted for faster training and hence we mainly follow [21].

Pre-training setting for 100 epochs. SGD is used as the optimizer. The weight decay is 1.0×10^{-4} and the momentum is 0.9. The learning rate is set according to linear scaling rule [16] 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 [18] 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 [21] to increase the exponential moving average parameter from 0.996 to 1 with a cosine scheduler. We list these hyper-parameters in Table

Pre-training setting for 800 epochs. LARS [32] optimizer is used for 800 epochs pre-training with a batch size of 4096. The weight decay is 1.0×10^{-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 [32] 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.

Hyper-parameter	Value
opitmizer	SGD
weight decay	1.0×10^{-4}
base lr	0.05
lr schedule	cosine
warmup	5 epochs
batch size	1024
projector	3-layers MLP
init momentum	0.996
final momentum	1.0
momentum schedule	cosine

Table 7. Hyper-parameters for 100 epochs pre-training. We use the same hyper-parameters for different loss functions.

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. [17] applies linear projection to contrastive learning. SimCLR [6] finds that a 2-layer MLP can help boost the performance. SimSiam [8] further extends the projector depth to 3. We explore the effects of different projector depths in Table 9. 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 [34] extends the dimension of projector from 2048 to 8192, showing notable improvement. We further study the effect of projector's width in Table 10. 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.

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 [6]. The results are reported in Table 11. Compared with previous methods, UniGrad is able to obtain comparable results with [29] and obtain 5% and 1% improvement from other methods on the 1% and 10% subset,

Method	VOC07+12 detection			COCO detection			COCO instance seg		
	AP_{all}	AP_{50}	AP_{75}	AP^{box}_{all}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}_{all}	AP^{mask}_{50}	AP^{mask}_{75}
Supervised	54.7	84.5	60.8	38.9	59.6	42.7	35.4	56.5	38.1
MoCov2 [7]	56.4	81.6	62.4	39.8	59.8	43.6	36.1	56.9	38.7
SimCLR [6]	58.2	83.8	65.1	41.6	61.8	45.6	37.6	59.0	40.5
DINO [5]	57.2	83.5	63.7	41.4	62.2	45.3	37.5	58.8	40.2
TWIST [29]	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 8. 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.

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

Table 9. Effect of projector depth.

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

Table 10. 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 [6]	48.3	75.5	65.6	87.8
BYOL [21]	53.2	78.4	68.8	89.0
Barlow Twins [34]	55.0	79.2	69.7	89.3
DINO [5]	52.2	78.2	68.2	89.1
TWIST [29]	61.2	84.2	71.7	91.0
UniGrad	60.8	83.8	71.5	90.6

Table 11. Semi-supervised learning on ImageNet.

respectively.

C.3. Transfer Learning

We also transfer the pretrained model to downstream stasks, including PASCAL VOC [14] object detection, COCO [25] object detection and instance segmentation. The model is finetuned in an end-to-end manner. Table 8 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 [12] is subject to the ImageNet terms of access [11].

PASCAL VOC [14] uses images from Flickr, which is subject to the Flickr terms of use [15].

COCO [25]. The annotations are under the Creative Commons Attribution 4.0 License. The images are subject to the Flickr terms of use [15].