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 \( u_1^o \):

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

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

\[
\frac{\partial l(u_1^o)}{\partial u_1^o} = \frac{\partial l(u_1^o)}{\partial u_1^o} \cdot \frac{\partial u_1^o}{u_1^o} + \sum_{v^m \in V_{\text{bank}}} \frac{\partial l(u_1^o)}{\partial v^m} \cdot \frac{\partial v^m}{u_1^o}
\]

\[
= -\frac{1}{s_{u_2}} s_{u_2} (1 - s_{u_2}) \cdot \frac{1}{\tau} u_2^m
\]

\[
- \sum_{v^m \in V_{\text{bank}}} \frac{1}{s_{u_2}} s_{u_2} s_v \cdot \frac{1}{\tau} u_2^m
\]

\[
= -\frac{1}{s_{u_2}} + \sum_{v^m \in V_{\text{bank}}} s_v \frac{1}{\tau} u_2^m, \tag{17}
\]

where \( s_v = \frac{\exp (\cos(u_1^o, v^m)/\tau)}{\sum_{s^m \in V_{\text{bank}}} \exp (\cos(u_1^o, v^m)/\tau)} \).

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

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

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@U2).sum(-1) # [N, 1]
# simplified SimCLR
loss = l/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

\[
l(u_1^o) = -\log \frac{\exp \left( \cos(u_1^o, u_2^o)/\tau \right)}{\sum_{v^m \in V_{\text{hatch}}} \exp \left( \cos(u_1^o, v^m)/\tau \right)}. \tag{19}
\]

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

\[
\frac{\partial L}{\partial u_1^o} = \frac{1}{N} \left( \frac{\partial l(u_1^o)}{\partial u_1^o} + \frac{\partial l(u_2^o)}{\partial u_1^o} + \sum_{v^m \in V_{\text{hatch}}} \frac{\partial l(v^o)}{\partial u_1^o} \right)
\]

A.2. Asymmetric Network Methods

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

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

\[
W_h = U \Lambda_h U^T, \quad \Lambda_h = \Lambda_F^{1/2} + \epsilon \Lambda_{\text{max}} I, \tag{23}
\]

where \( U \) and \( \Lambda_F \) are the eigenvectors and eigenvalues of \( F = \sum_{v^m \in V_{\text{hatch}}} P_{v^m} v^m v^m^T \), respectively. \( \epsilon \) is a hyperparameter to boost small eigenvalues.

Denote \( y_1 = W_h u_1^o, y_2 = \frac{y_2}{||y_2||_2} \), the gradient of \( L \) can be derived as:

\[
\frac{\partial L}{\partial u_1^o} = \frac{1}{N} \left( \frac{\partial l(y_1)}{\partial u_1^o} \cdot \frac{\partial l(y_2)}{\partial y_1} \cdot \frac{\partial l(y_2)}{\partial y_2} \right)
\]

\[
= \frac{1}{N} \left( -W_h^T \cdot \frac{1}{||y_2||_2} (y_2 - W_h u_1^o) \cdot u_2^o \right)
\]

\[
= \frac{1}{||W_h u_1^o||_2 N} \left( -W_h^T (I - W_h u_1^o u_2^o W_h^T W_h u_1^o) u_2^o \right)
\]
Substituting Eq. (25) into Eq. (24) leads to
\[
\frac{\partial L}{\partial u_i^2} = \frac{1}{||W_h \hat{u}_i^2||_2 N} \left( - W_h^T u_i^2 + \lambda (F\hat{u}_i^2 + \epsilon^2 \lambda_{max} u_i^2) \right),
\]
where \( \hat{\lambda} = \frac{\nu^2 W_h^T u_i^2}{u_i^2 T (F + 2 \epsilon \lambda_{max} F^{1/2} + \epsilon^2 \lambda_{max} I) u_i^2} \). For those three terms that are scaled by \( \hat{\lambda} \), 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
\[
\frac{\partial L}{\partial u_i^2} \approx \frac{1}{||W_h \hat{u}_i^2||_2 N} \left( - W_h^T u_i^2 + \lambda (F\hat{u}_i^2 + \epsilon^2 \lambda_{max} u_i^2) \right),
\]
where \( \lambda = \frac{\nu^2 W_h^T u_i^2}{u_i^2 T (F + 2 \epsilon \lambda_{max} F^{1/2} + \epsilon^2 \lambda_{max} I) u_i^2} \).

When \( u_i^2 \) is \( \ell_p \) normalized, we can further neglect the \( \epsilon^2 \lambda_{max} u_i^2 \) term, because the gradient propagated to unnormalized \( u_i^2 \) is 0. Hence, we simplify the gradient as
\[
\frac{\partial L}{\partial u_i^2} \approx \frac{1}{||W_h \hat{u}_i^2||_2 N} \left( - W_h^T u_i^2 + \lambda F\hat{u}_i^2 \right) = \frac{1}{||W_h \hat{u}_i^2||_2 N} \left( - W_h^T u_i^2 + \lambda \sum_{v^o} \langle \rho v^o u_i^o T v^o \rangle v^o \right).
\]

Note that \( \lambda \) 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 substituted by a constant scalar.

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:
\[
L = \sum_{i=1}^C (W_{ii} - 1)^2 + \lambda \sum_{i=1}^C \sum_{j \neq i} W_{ij}^2,
\]
where \( W = \frac{1}{N} \sum_{i=1}^N v_i^o v_i^o T \in V_{match} \) is the cross-correlation matrix.

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

$$
\frac{\partial L_2}{\partial (u_1^*)_k} = \lambda \sum_{j \neq k} \frac{\partial \log W_{ij}}{\partial (u_1^*)_k} = \lambda \sum_{j \neq k} 2W_{ij} \cdot (u_2^*)_j
$$

$$
\frac{\partial L_2}{\partial (u_1^*)_k} = \frac{2\lambda}{N} \left( - W_{kk}(u_2^*)_k + \frac{C}{j=1} W_{kj}(u_2^*)_j \right)
$$

$$
\frac{\partial L_2}{\partial (u_1^*)_k} = \frac{2\lambda}{N} \left( - W_{kk}(u_2^*)_k + \frac{C}{j=1} \frac{1}{v_i^j,v_j^k \in V_{\text{batch}}} \sum (v_i^j)(v_j^k)(u_2^*)_j \right)
$$

$$
\frac{\partial L_2}{\partial (u_1^*)_k} = \frac{2\lambda}{N} \left( - W_{kk}(u_2^*)_k + \frac{C}{j=1} \frac{1}{(v_i^j)(v_j^k) \in V_{\text{batch}}} \sum (v_i^j)(v_j^k)(u_2^*)_j \right)
$$

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

$$
\frac{\partial L}{\partial u_1^*} = \frac{2}{N} \left( - Au_2^* + \lambda \sum_{v_i^j,v_j^k \in V_{\text{batch}}} \frac{u_i^j}{v_i^j} \cdot \frac{v_j^k}{v_j^k} \right), \quad (32)
$$

where $A = I - (1 - \lambda)W_{\text{diag}}$. Here $(W_{\text{diag}})_{ij} = \delta_{ij} W_{ij}$ is the diagonal matrix of $W$, where $\delta_{ij}$ 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_corr = u1.T @ u2 / N # [C, C]
# positive term
pos = (1 - (1 - lambda) @ torch.diag(W_corr)) * u2
loss_pos = -(u1 * pos.detach()).sum(-1) # [N, 1]
# negative term
weight = u1 @ u1.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 components:

$$
L_1 = \frac{1}{N} \sum_{v_i^j,v_j^k \in V_{\text{batch}}} ||v_i^j - v_j^k||^2_2, \quad (33)
$$

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

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

where $W' = \frac{1}{N-1} \sum_{v_i^j \in V_{\text{batch}}} (v_i^j - \bar{v})(v_i^j - \bar{v})^T$.

For the invariance term $L_1$, we have:

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

### 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:

$$
\frac{\partial L_2}{\partial (u_1^*)_k} = \frac{2\lambda_1}{C} \sum_{j \neq k} \frac{\partial \log W_{ij}}{\partial (u_1^*)_k} = \frac{4\lambda_1}{C} \sum_{j \neq k} \left( - W_{kk}(u_1^* - \bar{v})_k + \sum_{j=1}^{C} W_{kj}(u_1^* - \bar{v})_j \right)
$$

$$
\frac{\partial L_2}{\partial (u_1^*)_k} = \frac{4\lambda_1}{C(N-1)} \left( - W_{kk}(u_1^* - \bar{v})_k + \sum_{j=1}^{C} W_{kj}(u_1^* - \bar{v})_j \right)
$$

$$
\frac{\partial L_2}{\partial (u_1^*)_k} = \frac{4\lambda_1}{C(N-1)} \left( - W_{kk}(u_1^* - \bar{v})_k + \sum_{j=1}^{C} \sum_{v_i^j \in V_{\text{batch}}} \frac{u_i^j}{v_i^j} \cdot \frac{v_j^k}{v_j^k} \right)
$$

$$
\frac{\partial L_2}{\partial (u_1^*)_k} = \frac{4\lambda_1}{C(N-1)^2} \left( - \frac{N-1}{N} W_{kk}(u_1^* - \bar{v})_k + \sum_{v_i^j \in V_{\text{batch}}} \frac{u_i^j}{v_i^j} \cdot \frac{v_j^k}{v_j^k} \right)
$$

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

For the variance term $L_3$, we have:

$$
\frac{\partial L_3}{\partial (u_1^*)_k} = \frac{\lambda_2}{C} \log \left( \frac{\gamma - \text{std}(v_i^k)}{\text{std}(v_i^k)} \right) \cdot \frac{\text{std}(v_i^k)}{\text{std}(v_i^k)}
$$

$$
\frac{\partial L_3}{\partial (u_1^*)_k} = \frac{-\lambda_2}{C(N-1)}(\gamma - \text{std}(v_i^k) > 0) \cdot \frac{\text{std}(v_i^k)}{\text{std}(v_i^k)}. \quad (38)
$$

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

$$
\frac{\partial L}{\partial u_1^*} = \frac{2}{N} (u_1^* - u_2^*) - \frac{2\lambda}{N} \left( \frac{N-1}{N} - W_{\text{diag}} \bar{v}^1 - \sum_{v_i^j \in V_{\text{batch}}} \frac{u_i^j}{v_i^j} \cdot \frac{v_j^k}{v_j^k} \right)
$$

$$
\frac{\partial L}{\partial u_1^*} = \frac{-\lambda_2}{C(N-1)} \text{diag}(\gamma - \text{std}(v_i^k) > 0) \cdot \text{std}(v_i^k) \bar{v}^1
$$

$$
\frac{\partial L}{\partial u_1^*} = \frac{2}{N} (u_1^* - u_2^*) - \lambda \sum_{v_i^j \in V_{\text{batch}}} \frac{u_i^j}{v_i^j} \cdot \frac{v_j^k}{v_j^k} + \frac{\lambda}{N} \left( \frac{1}{\lambda} \bar{v}^1 - B \bar{v}^1 \right), \quad (39)
$$
where \( B = \frac{N}{\lambda_C(N-1)} (2\lambda_1 W'_{\text{diag}} + \frac{3}{2} \text{diag}(\mathbf{I}(\gamma - \text{std}(v_0^2) > 0) \odot \text{std}(v_0^2))) \). Here \( W'_{\text{diag}} \) is the diagonal matrix of \( W' \), \( \text{diag}(x) \) is a matrix with diagonal filled with the vector \( x \), \( \mathbf{I}(\cdot) \) is the indicator function, and \( \odot \) denotes element-wise division.

### A.4. Pseudocode of UniGrad

**Algorithm 6** Pseudocode of UniGrad in PyTorch style.

```python
# 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) # negative term
loss = (loss_pos + lambda * loss_neg).mean() # positive term

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

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

<table>
<thead>
<tr>
<th>Hyper-parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimizer</td>
<td>SGD</td>
</tr>
<tr>
<td>weight decay</td>
<td>1.0 \times 10^{-4}</td>
</tr>
<tr>
<td>base lr</td>
<td>0.05</td>
</tr>
<tr>
<td>lr schedule</td>
<td>cosine</td>
</tr>
<tr>
<td>warmup</td>
<td>5 epochs</td>
</tr>
<tr>
<td>batch size</td>
<td>1024</td>
</tr>
<tr>
<td>projector</td>
<td>3-layers MLP</td>
</tr>
<tr>
<td>init momentum</td>
<td>0.996</td>
</tr>
<tr>
<td>final momentum</td>
<td>1.0</td>
</tr>
<tr>
<td>momentum schedule</td>
<td>cosine</td>
</tr>
</tbody>
</table>

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 \( 1.0 \times 10^{-5} \) and the momentum is 0.9. The learning rate is set according to linear scaling rule [16] as \( \text{base lr} \times \text{batch size}/256 \), with \( \text{base lr} = 0.05 \). 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 34096. BN and ReLU are applied after the first two layers. The learning rate is set to 0.05. 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.

**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.
<table>
<thead>
<tr>
<th>Method</th>
<th>VOC07+12 detection</th>
<th>COCO detection</th>
<th>COCO instance seg</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AP&lt;sub&gt;all&lt;/sub&gt;</td>
<td>AP&lt;sub&gt;50&lt;/sub&gt;</td>
<td>AP&lt;sub&gt;75&lt;/sub&gt;</td>
</tr>
<tr>
<td>Supervised</td>
<td>54.7</td>
<td>84.5</td>
<td>60.8</td>
</tr>
<tr>
<td>MoCov2 [7]</td>
<td>56.4</td>
<td>81.6</td>
<td>62.4</td>
</tr>
<tr>
<td>SimCLR [6]</td>
<td>58.2</td>
<td>83.8</td>
<td>65.1</td>
</tr>
<tr>
<td>DINO [5]</td>
<td>57.2</td>
<td>83.5</td>
<td>63.7</td>
</tr>
<tr>
<td>TWIST [29]</td>
<td>58.1</td>
<td>84.2</td>
<td>65.4</td>
</tr>
<tr>
<td>UniGrad</td>
<td>57.8</td>
<td>84.0</td>
<td>64.9</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Depth</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Eval</td>
<td>60.7</td>
<td>65.2</td>
<td>70.3</td>
<td>70.0</td>
<td>69.8</td>
</tr>
</tbody>
</table>

Table 9. Effect of projector depth.

<table>
<thead>
<tr>
<th>Width</th>
<th>1024</th>
<th>2048</th>
<th>4096</th>
<th>8192</th>
<th>16384</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Eval</td>
<td>68.3</td>
<td>70.3</td>
<td>70.5</td>
<td>70.9</td>
<td>71.2</td>
</tr>
</tbody>
</table>

Table 10. Effect of projector width.

<table>
<thead>
<tr>
<th>Method</th>
<th>1%</th>
<th>10%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Top 1</td>
<td>Top 5</td>
</tr>
<tr>
<td>Supervised</td>
<td>25.4</td>
<td>48.4</td>
</tr>
<tr>
<td>SimCLR [6]</td>
<td>48.3</td>
<td>75.5</td>
</tr>
<tr>
<td>BYOL [21]</td>
<td>53.2</td>
<td>78.4</td>
</tr>
<tr>
<td>Barlow Twins [34]</td>
<td>55.0</td>
<td>79.2</td>
</tr>
<tr>
<td>DINO [5]</td>
<td>52.2</td>
<td>78.2</td>
</tr>
<tr>
<td>TWIST [29]</td>
<td>61.2</td>
<td>84.2</td>
</tr>
<tr>
<td>UniGrad</td>
<td>60.8</td>
<td>83.8</td>
</tr>
</tbody>
</table>

Table 11. Semi-supervised learning on ImageNet.

respectively.

C.3. Transfer Learning

We also transfer the pretrained model to downstream tasks, 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

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].