Supplemental Material to:
An Embedding-Dynamic Approach to Self-supervised Learning

Suhong Moon
UC Berkeley
suhong.moon@berkeley.edu

Domas Buracas
UC Berkeley
dominykas@berkeley.edu

Seunghyun Park
Clova AI Research, NAVER Corp
seung.park@navercorp.com

Jinkyu Kim
Korea University
jinkyukim@korea.ac.kr

John Canny
UC Berkeley
canny@berkeley.edu

Content
This supplementary material provides implementation details (Section 1) including training strategy, architectures, and image augmentations. We also provide evaluation details (Section 2) including linear evaluation protocol, semi-supervised learning setting, k-NN classification, and transfer learning on various downstream tasks. We also report supplemental experimental results of instance segmentation on COCO dataset (Section 3). In addition, this supplementary presents ablation study results. Lastly, we compare our method with [16] and [25] in detail.

1. Implementation Details
We first provide implementation details of our method. We would emphasize that our code will be made publicly available upon publication. In Section 1.1 and , we explain details of our training strategy and architectures. Next, in Section 1.3, we explain details of the stochastic image data augmentation used in our experiment.

1.1. Training Strategy.
We utilize the Layer-wise Adaptive Rate Scaling (LARS) [28] optimizer that is known to effectively overcome large-batch training difficulties. We also use the learning rate scheduler that applies a cosine decay function [19] without restarts to an optimizer step. As suggested by [11], we apply a learning rate warm-up for the first 10 epochs where we start training with a small safe learning rate, which is slowly increased to the max learning rate linearly. The max learning rate is $\text{base lr} \times \frac{\text{batch size} \times K}{256}$ [11]. We set the base learning rate to 0.4 for ImageNet-100, 0.5 for STL-10 dataset and 0.15 for ImageNet datasets. Unless otherwise stated, we set the batch size to 512. The weight decay parameter is set to $1 \times 10^{-5}$. We exclude biases and parameters in batch normalization layer following BYOL [13]. We train the model for 320 epochs for ImageNet-100 and STL-10 benchmarks and 300 epochs for ImageNet with 8 V100 16GB GPUs.

1.2. Architectures
For a fair comparison, we use ResNet-18 [15] as a backbone network architecture for STL-10 and ImageNet-100 datasets and ResNet-50 as a backbone for ImageNet dataset, which are widely experimented with conventional approaches for the self-supervised representation learning task. Following BYOL [13], the projection heads (i.e. $f_\theta$ and $g_\xi$ in Figure 2 in the main paper) and the prediction head of the online network (i.e. $h_\theta$) use a 2-layer fully connected network with ReLU [20] as an activation function. We tune the size of hidden layers and output layers of projection and prediction heads, when the backbone network is ResNet-18. We use 512 hidden layer size and 128 output layer size instead of 2048 hidden units and 256 output size, which are used in BYOL. We apply batch normalization layer [17]. Also, we experiment various normalization layers including weight standardization [21] and layer normalization [1] to show that our method does not suffer from mode collapse without batch normalization.

1.3. Image Augmentations
We use a stochastic data augmentation operator that is sampled from the family of augmentations $T$ and results in a randomly augmented view of any given data example. Following SimCLR [4], our data augmentation module sequentially applies the following four augmentations: (1) random cropping followed by resizing back to the original size, (2) aspect-ratio changes, (3) random flipping in the horizontal direction, (4) random color distortion (i.e. jitter and lighting). Detailed augmentation parameters are in Table [1].
2. Evaluation Details

In this section, we provide relevant information for evaluation of our method.

2.1. Linear Evaluation Protocol

We use the linear evaluation protocol [18], which is the standard practice to evaluate the quality of the learned image representations. Using the trained encoder as the feature extractor, we train a linear classifier as a post-hoc manner, i.e., a simple image classifier given a set of features. Then, we measure its classification accuracy on the test set as a proxy of the quality of the learned representations. Note that the encoder is frozen during the evaluation phase. We use the following three standard image classification benchmarks: (1) STL-10 [5], (2) ImageNet-100 [24], and (3) ImageNet [6]. Note that ImageNet-100 contains only 100-class examples that are randomly sampled from ImageNet.

2.2. Semi-Supervised Learning

We also evaluate the semi-supervised learning ability of our method with subset of ImageNet training set. We fine-tune ResNet-50 encoder pretrained with our algorithm and the classifier on top of the encoder using 1% and 10% of ImageNet. These ImageNet splits can be found from the official implementation of [4]. We mainly follow the semi-supervised learning protocol of [13]. We use SGD with momentum of 0.9 and Nesterov, batch size of 1024. We use the separate learning rates for the classifier and the encoder. For fine-tuning task with 1% ImageNet subset, we set learning rate of the classifier 2.0 and freeze the encoder. For fine-tuning task with 10% ImageNet subset, we use the 0.25 as the learning rate of the classifier and 2.5 × 10^{-4} as the learning rate of the encoder.

2.3. k-NN Classification

We closely follow the existing work [27] [30] to evaluate the quality of representations learned by our model. We first collect representations from training and validation images with the frozen encoder. Then, we compute the classification accuracy of 20/200-nearest neighbor classifier.

2.4. Transferring to Downstream Tasks

To test the transferability of representations trained with our method on ImageNet, we perform transfer learning to various datasets: Places205, iNaturalist2018, Pascal VOC, and COCO.

**Image classification.** We train the linear classifier layer on top of the frozen ResNet-50 backbone pretrained with MS-BReg. For VOC 07, we train a linear support vector machine (SVM). For other image classification benchmarks, iNaturalist 2018 and Places 205, we train the linear classifier with SGD with momentum of 0.9 and weight decay of 10^{-4}. The batch size is 256 and learning rate is 0.2 and we reduce the learning rate by factor of 10 two times with equally spaced intervals. For Places205, the training epoch is 28 and for iNaturalist 2018, the training epoch is 84.

**Object detection.** Following previous works [12] [5] [2] [10], we finetune the network on VOC07+12 [7] dataset using Faster-RCNN [22]. We report three metrics of the object detection, AP_{all}, AP_{50} and AP_{50}. We use Detectron2 [26] to transfer our model to the object detection task. We set the initial learning rate 0.02. Other hyperparameters such as learning rate scheduling, warm-up steps are exactly same as [14].

**Instance Segmentation.** For instance segmentation task, we evaluate our model with COCO dataset. We closely follow [14] [29] [2]. We use Mask R-CNN FPN backbone. The backbone is initialized with our pretrained ResNet-50 backbone. We train the network for 90K iterations with a batch size of 16. A learning rate is 0.05 and reduced by a factor of 10 after 60K and 80K iterations. We linearly warm up the learning rate for 50 iterations.

3. Results on COCO Instance Segmentation

We also evaluate the learned representation on COCO instance segmentation task. We observe in Table 2 that our method shows competitive performance with other methods. Our method is better than BYOL [13] (3rd row), which is our main baseline. SwAV [4] (5th row) shows similar performance to ours. Note that this method uses more augmentations than ours.

4. Related Works

In this section, we supplement Section 2. We compare our work with batch repetition method [16], uniformity loss [23], and BYOL without BN [23] in detail.

**Batch Repetition.** In the Section 2, we mention batch repetition method [16]. Similar to this method, our multiview
known fact about BYOL [13] is that this method falls into BYOL without Batch Normalization Layer. The widely bedding manifold. of [25] and ours, which argues that the optimal distribution performance than other baselines. This strengthen the argument uniformity loss, higher alignment loss and the better perfor-

<table>
<thead>
<tr>
<th>Method</th>
<th>L\textsubscript{align}</th>
<th>L\textsubscript{uniform}</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYOL</td>
<td>71.9</td>
<td>-1.52</td>
</tr>
<tr>
<td>BYOL+L\textsubscript{uniform}</td>
<td>72.1</td>
<td>-2.95</td>
</tr>
<tr>
<td>BYOL+L\textsubscript{b} + L\textsubscript{s}</td>
<td>72.8</td>
<td>-2.92</td>
</tr>
<tr>
<td>Ours (K = 4)</td>
<td>80.4</td>
<td>-3.8</td>
</tr>
</tbody>
</table>

The mode collapse [8] without batch normalization layer. The authors of [13] performed studies that BYOL works even without BN layer [23]. In this paper, authors showed that BYOL without BN gets matched performance using various normalization techniques including weight standardization [21] or the deliberately handled initialization. But still, BYOL fails to converge optimal solution with such deliberately tuned training techniques. In this section, we show that MSBReg also work with layer normalization [1] without any other techniques in Table 4.

In Table 4, the top-1 classification accuracy is largely degraded from 89.5% to 10.6%, i.e. mode collapsed. Ours with the Brownian diffusive loss L\textsubscript{b} was not the case (compare 2nd vs. 6th row). Though we observe a slight degra-
dation in the top-1 classification accuracy, ours sufficiently avoid collapsed representations. Further, we evaluate the BYOL with our Brownian diffusive loss to demonstrate its effectiveness against a mode collapse. We observe that our Brownian diffusive loss helps avoid collapsed representations (compare 3rd vs. 4th rows). We also observe that the quality of representations depends on the strength of the hyperparameter \( \lambda_b \) where we obtain the best performance with \( \lambda_b = 5 \times 10^{-4} \). We observe a tension as we see a smaller or larger \( \lambda_b \) slightly degrades the quality of representations.

5. Ablation Studies

We perform ablation experiments to study the trade off between major hyperparameters in MSBReg, \( \lambda_s \) and \( \lambda_b \). In table 5 our experiment reports the top-1 classification accuracy on ImageNet-100. We train ResNet-18 with MSBReg for 300 epochs with various combinations of \( K \in \{2, 4, 8\} \), \( \lambda_s \in \{0, 0.002, 0.004, 0.01\} \), and \( \lambda_b \in \{0, 0.25, 0.5, 1.0, 2.0\} \). Note that the case of \( K = 2 \) is the same as BYOL setting. Then, we train the linear classifier on top of frozen ResNet-18 backbone pretrained with MSBReg. Our study shows that the classification accuracy increases until \( \lambda_s = 0.004, \lambda_b = 0.5 \) for the cases...
of $K = 4$ and $K = 8$. Interestingly, both singular value loss and Brownian loss improve the performance for the case of BYOL ($K = 2$).

References


Table 5. Ablation studies to investigate the trade-off between losses in MSBReg.

<table>
<thead>
<tr>
<th>$K$</th>
<th>$\lambda_a$</th>
<th>$\lambda_b$</th>
<th>Acc. (%)</th>
<th>$K$</th>
<th>$\lambda_a$</th>
<th>$\lambda_b$</th>
<th>Acc. (%)</th>
<th>$K$</th>
<th>$\lambda_a$</th>
<th>$\lambda_b$</th>
<th>Acc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>71.9</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>78.2</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>79.5</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.25</td>
<td>72.3</td>
<td>4</td>
<td>0</td>
<td>0.25</td>
<td>79.1</td>
<td>8</td>
<td>0</td>
<td>0.25</td>
<td>80.1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.5</td>
<td>72.8</td>
<td>4</td>
<td>0</td>
<td>0.5</td>
<td>79.7</td>
<td>8</td>
<td>0</td>
<td>0.5</td>
<td>80.3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1.0</td>
<td>72.3</td>
<td>4</td>
<td>0</td>
<td>1.0</td>
<td>79.2</td>
<td>8</td>
<td>0</td>
<td>1.0</td>
<td>80.3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2.0</td>
<td>71.9</td>
<td>4</td>
<td>0</td>
<td>2.0</td>
<td>79.0</td>
<td>8</td>
<td>0</td>
<td>2.0</td>
<td>80.2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.0002</td>
<td>72.2</td>
<td>4</td>
<td>0.0002</td>
<td>0</td>
<td>78.9</td>
<td>8</td>
<td>0.0002</td>
<td>0</td>
<td>79.8</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.0002</td>
<td>0.25</td>
<td>72.4</td>
<td>4</td>
<td>0.0002</td>
<td>0.25</td>
<td>78.8</td>
<td>8</td>
<td>0.0002</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.0002</td>
<td>0.5</td>
<td>72.4</td>
<td>4</td>
<td>0.0002</td>
<td>0.5</td>
<td>79.1</td>
<td>8</td>
<td>0.0002</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.0002</td>
<td>1.0</td>
<td>72.1</td>
<td>4</td>
<td>0.0002</td>
<td>1.0</td>
<td>79.2</td>
<td>8</td>
<td>0.0002</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.0002</td>
<td>2.0</td>
<td>71.9</td>
<td>4</td>
<td>0.0002</td>
<td>2.0</td>
<td>79.2</td>
<td>8</td>
<td>0.0002</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.004</td>
<td>0.004</td>
<td>72.8</td>
<td>4</td>
<td>0.004</td>
<td>0.004</td>
<td>79.7</td>
<td>8</td>
<td>0.004</td>
<td>0.004</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.004</td>
<td>0.25</td>
<td>72.8</td>
<td>4</td>
<td>0.004</td>
<td>0.25</td>
<td>80.2</td>
<td>8</td>
<td>0.004</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.004</td>
<td>0.5</td>
<td>72.4</td>
<td>4</td>
<td>0.004</td>
<td>0.5</td>
<td>80.4</td>
<td>8</td>
<td>0.004</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.004</td>
<td>1.0</td>
<td>72.1</td>
<td>4</td>
<td>0.004</td>
<td>1.0</td>
<td>80.1</td>
<td>8</td>
<td>0.004</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.004</td>
<td>2.0</td>
<td>71.2</td>
<td>4</td>
<td>0.004</td>
<td>2.0</td>
<td>79.9</td>
<td>8</td>
<td>0.004</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.01</td>
<td>0.01</td>
<td>71.1</td>
<td>4</td>
<td>0.01</td>
<td>0.01</td>
<td>79.3</td>
<td>8</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.01</td>
<td>0.25</td>
<td>71.0</td>
<td>4</td>
<td>0.01</td>
<td>0.25</td>
<td>79.4</td>
<td>8</td>
<td>0.01</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.01</td>
<td>0.5</td>
<td>71.0</td>
<td>4</td>
<td>0.01</td>
<td>0.5</td>
<td>79.2</td>
<td>8</td>
<td>0.01</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.01</td>
<td>1.0</td>
<td>70.7</td>
<td>4</td>
<td>0.01</td>
<td>1.0</td>
<td>79.2</td>
<td>8</td>
<td>0.01</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.01</td>
<td>2.0</td>
<td>70.3</td>
<td>4</td>
<td>0.01</td>
<td>2.0</td>
<td>79.0</td>
<td>8</td>
<td>0.01</td>
<td>2.0</td>
</tr>
</tbody>
</table>

---

*“on Computer Vision and Pattern Recognition (CVPR), June 2018.*

