Improving Contrastive Learning by Visualizing Feature Transformation -Supplementary Material-

Rui Zhu[1,2*], Bingchen Zhao[3*], Jingen Liu [2†], Zhenglong Sun[1], Chang Wen Chen[4]

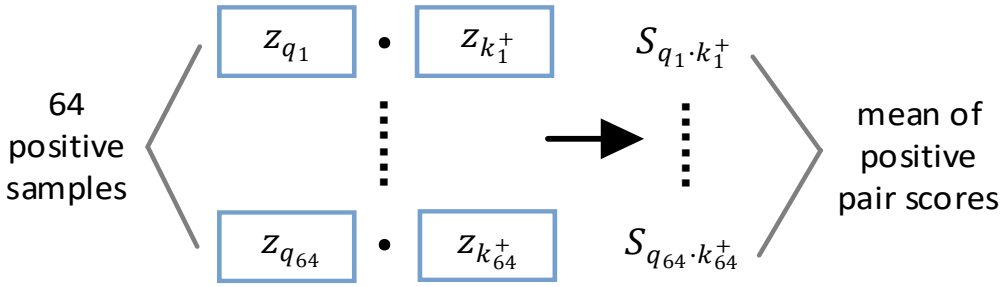
[1] The Chinese University of HongKong, Shenzhen [2] JD AI Research [3] Tongji University [4] The Hong Kong Polytechnic University

ruizhu@link.cuhk.edu.cn, {zhaobc.gm, jingenliu}@gmail.com, sunzhenglong@cuhk.edu.cn, changwen.chen@polyu.edu.hk

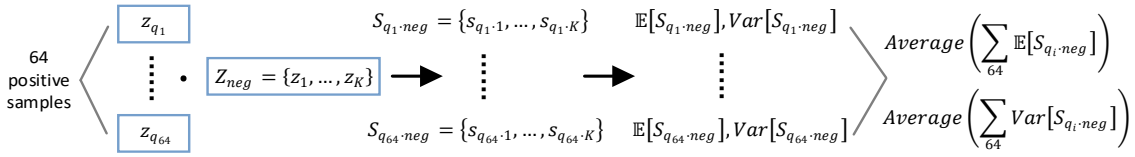
In the supplementary materials, we provide more details and analyses about our methods, including:

*Equally-contributed and this work is done at JD AI Research.

†Corresponding author.



(a) Mean of positive pair scores.



(b) Mean and variance of negative pair scores.

Figure 1. Illustrations of pos/neg pair score visualization. (a) Mean of positive pair scores. (b) Mean and variance of negative pair scores.

A. Details of the Visualization Tool

We choose three non-trivial statistics to visualize the score distribution: the mean of pos/neg scores (denoted as $Mean(pos)$/$Mean(neg)$, indicating the approximate average of the pos/neg pair distance) and the variance of negative scores (denoted as $Var(neg)$, indicating the fluctuation degree of the negative samples in the memory queue).

Without loss of generality, we randomly choose 64 samples [1] in one batch to calculate the statistics data and perform visualization. (1) For the positive pair score: As shown in Fig 1(a), we denote the z_{q_i}, $(i = 1, 2, 3...64)$ as the 64 query samples. And $z_{k_i^+}$, are the corresponding positive features of z_{q_i}. Then we can get 64 positive score $S_{q_i \cdot k_i^+}$, by inner product. Finally, we retain the mean value of these 64 positive scores as $Mean(pos)$. (2) For the negative pair scores: As shown in Fig 1(b), we denote the $Z_{neg} = \{z_1, z_2, \dots, z_K\}$ where K is the size of the memory queue [2]. Each z_{q_i} combining Z_{neg} will create K negative pair scores in a set, named $S_{q_i \cdot neg} = \{s_{q_i \cdot 1}, s_{q_i \cdot 2}, \dots, s_{q_i \cdot K}\}$. To keep all the $64 \times K$ negative scores is challenging (about 4TB storage for the pair scores), so for each $S_{q_i \cdot neg}$, we retain their mean and variance to show the distribution of K negative sample scores corresponding to z_{q_i}. More generally, we further average these 64 means and variances to show the statistical characteristics of these K negative samples ($Mean(neg)$ and $Var(neg)$). These statistics are recorded at each training step to track the score distribution in the training process.

Our visualization is very practical. It is offline, which almost does not affect the training speed. Instead of storing K (65536) pair scores, we save their statistical mean & variance to represent the scores' distribution. As a result, it only takes about 20MB storage and 5 minutes extra time for a 256 batch-size 100 epoch training. Even with larger datasets and batch size, it's still feasible.

[1] We usually apply batch-size 256 on 4-GPU servers. Here we collect one batch 64 on a single GPU for statistics.

[2] K is a large number, *e.g.*, 65536 in MoCo [5] and the largest 8192×2 in one batch for SimCLR [1]. We use $K = 65536$ in all the MoCo experiments.

B. Experimental Details of each Part in the Paper

The experiments are mainly implemented using the code from InfoMin [13] [3]. The transfer experiments on object detection and instance segmentation are implemented using Detectron2 [4]. We keep the fairness of the experiments, especially when compared with other methods. The code of our proposed methods and visualization tools will be made public.

B.1. Data Augmentations

For the experiments of combining our feature transformation module with other contrastive learning methods, we use the same image-level data augmentation strategies as the respective methods. Specifically, for our visualization experiments and other experiments using MoCo, we use the same data augmentation strategies with MoCo which contains *Random Resized Crop*, *Horizontal Flip*, *ColorJitter*, and *Random Gray Scale*. For the experiments on MoCoV2 [2] and SimCLR [1], the data augmentation strategies are the same which contains *Random Resized Crop*, *Horizontal Flip*, *ColorJitter*, *Random Gray Scale*, and *Gaussian Blur*.

B.2. Implementation of Visualization experiments

For training All the visualization experiments are carried on ImageNet-100 and ResNet-18 for fast evaluation and parameters-tuning experiments. For the visualization experiments (including Table 1, Table 6 (2nd row), figure 1(a),3,4,5,7 in the paper and Table 2 (2nd row), 5, figure 2,3,4 in supplementary materials), we apply a mini-batch size of 256 is used with 4-GPUs, where the number of negative examples is set to 65,536, with initial learning of 0.03. And we use $256/4 = 64$ samples to perform visualizations. For the fast grid experiments, the model is trained for only 100 epochs with the learning rate multiplied by 0.1 at 60 and 80 epochs. We use SGD as the optimizer, the weight decay of SGD is 0.0001 and the momentum of SGD is 0.9. And for various unit-sphere projection experiments, we apply 200 epochs training to perform visualization.
For testing we use the linear readout protocol to evaluate the trained representation on the validation set by fixing the learned representation and train a supervised linear classifier on the representations, the single-crop top-1 accuracy on the validation set is reported. An initial learning rate of 10 and weight decay 0. The classifier is trained with 100 epochs and the learning rate is multiplied by 0.1 at 60, and 80 epochs.

B.3. Implementation on ImageNet-100

For training we use ResNet-50 for ImageNet-100 implementations And momentum parameter is set to be 0.99 for our experiments. (including Table 2,3,4,5,6 (2nd row),7 in the paper and Table 2 (1st row),3,6, 7 in supplementary materials). A mini-batch size of 256 is used with 8-GPUs, where the number of negative examples is set to 65,536, with initial learning of 0.03. The model is trained for 200 epochs with the learning rate multiplied by 0.1 at 120 and 160 epochs. We use SGD as the optimizer, the weight decay of SGD is 0.0001 and the momentum of SGD is 0.9.
For testing we use the linear readout protocol to evaluate the trained representation on the validation set by fixing the learned representation and train a supervised linear classifier on the representations, the single-crop top-1 accuracy on the validation set is reported. We use an initial learning rate of 10 and weight decay 0. The classifier is trained with 60 epochs and the learning rate is multiplied by 0.1 at 30, 40, and 50 epochs following [11].

B.4. Implementation on ImageNet-1k

For training The momentum update parameter m for the experiments on ImageNet-1k is set to 0.999, other parameters are set to the same as the experiments on ImageNet-100. ResNet-50 is used as an encoder. (including Table 8 in the paper and Table 4 in supp material). We can observe that the best result for positive extrapolation and negative interpolation is achieved when α_{in} and α_{ex} are set to 1.6 and 2.0 respectively. Thus we use this value for the other experiments. Except otherwise stated, other hyper-parameters are set to be the same with MoCo [5] and MoCoV2[2].
For testing The same linear readout protocol is used where the linear classifier is trained for 100 epochs and the initial learning rate is 30 which are multiplied by 0.1 at 60, 80epochs.

B.5. Implementation on Fine-grained Classification

In addition to object detection and instance segmentation tasks, we also provide a study of fine-grained classification. We choose three challenging fine-grained datasets to conduct the experiments, iNaturalist 2018 dataset, CUB-200 dataset, and FGVC-aircraft dataset. (1) The iNaturalist 2018 has 437k images and 8142 classes, this dataset is commonly used for

[3] https://github.com/HobbitLong/PyContrast
[4] https://github.com/facebookresearch/detectron2

fine-grained classification and long-tailed recognition, and is used by several papers for evaluating the transfer performance of self-supervised representations [5]. (2) The CUB-200 dataset contains 6033 images belong to 200 bird species and is used for fine-grained classification. (3) The FGVC-aircraft dataset has 10,200 images of aircraft, with 100 images for each of 102 different aircraft model variants, most of which are airplanes. When transferring to these datasets, the pre-trained model is fine-tuned with 100 epochs, the learning rate is set to 5e-3 with cosine decay.

B.6. Object detection on PASCAL VOC

The main goal of self-supervised pre-training is to obtain representation that can be beneficial for downstream tasks. We choose to use PASCAL VOC [3] and COCO [8] as our benchmark for testing the transfer performance of the representation to object detection and instance segmentation tasks following previous works [5]. For PASCAL VOC dataset, we use the `trainval07+12` split for fine-tuning, and the `test2007` split for evaluating. The image scale is set to [480, 800] pixels for training and 800 for testing. For COCO dataset, we use the `train2017` split (118k images) for fine-tuning, the `val2017` split for evaluating. The image scale is set the same with PASCAL VOC.

When transferring to detection tasks, feature normalization has been shown to be crucial during fine-tuning [5]. Therefore, the pre-trained backbone is fine-tuned with Synchronized BN (SyncBN) [9] and add SyncBN to the FPN layer following [5]. We use Faster R-CNN [10] with R50-C4 architectures for object detection on the PASCAL VOC dataset. All layers of the model are fine-tuned with 24,000 iterations with each batch consisting of 16 images. The initial learning is set to 0.02 and is multiplied by 0.1 at 18,000 and 22,000 iterations. Other hyper-parameters are set to be the same with [5].

B.7. Object detection and Instance segmentation on MSCOCO

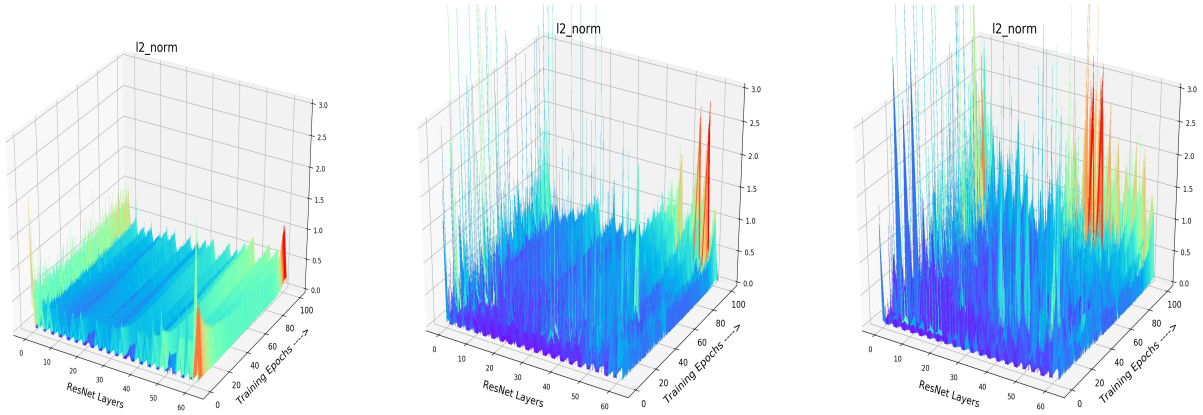
We also tested the transferring abilities of the pre-trained model using the instance segmentation tasks on MS COCO dataset. We uses a Mask R-CNN [6] R50-FPN pipeline following [13]. The batch size is set to 16 with the learning rate as 0.02, the model is trained with 1x and 2x schedules, for 1x schedules, the model is trained for 90,000 iterations on the MS COCO datasets with the learning rate multiplied by 0.1 at 60,000 and 80,000 iterations, for the 2x schedules, we use 180,000 iterations with the learning rate multiplied by 0.1 at 120,000 and 160,000 iterations. The transfer results of the 2x schedule is provided in Tab 1. Other hyper-parameters are set to be the same with [5].

Method	Performance					
	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}
mocov1	40.7	60.5	44.1	35.4	57.3	37.6
mocov1+ours	41.5	61.0	44.5	35.9	57.7	38.0
mocov2	40.9	60.7	44.4	35.5	57.5	37.9
mocov2+ours	41.3	60.9	44.8	35.7	57.8	38.1

Table 1. COCO object detection and instance segmentation based on Mask-RCNN-FPN with 2x learning rate schedule. Our results are reported using the average of 3 runs.

C. Details of the Gradient Landscape

We provide the details of our gradient landscape Figure 4 of various m in the paper. As shown in Fig 2, we provide ℓ_2 norm for each layer of the encoder (ResNet-18) with the training process. X axis indicates the layers of the encoder, while Y axis indicates the 100 training epochs. And Z axis means the value of ℓ_2 norm. We choose the ℓ_2 norm of this layer (total gradient ℓ_2 norm of this layer) because the ℓ_2 norm of gradient is very obvious to show the smoothness of gradient landscape. We can see that small $m = 0.6$ and 0.5 brings drastic volatility with the training process. The corresponding loss value and the gradient will fluctuate violently, resulting in bad convergence. As shown in Fig 2, the smooth and stable gradient landscape of $m = 0.99$ (Fig 2(a)) becomes sharp and messy with the decrease of m (Fig 2(b) for $m = 0.6$ and Fig 2(c) for $m = 0.5$). Therefore, to learn a better pre-trained model, we need to prepare negative pairs that can maintain the stability and smoothness of score distribution and gradient for the training process. It seems that the gradient landscape looks spiky: 1) Across Y axis indicating the training epochs. 2) Across X axis representing the ResNet layers, it shows the gradients of all layers including the BatchNorm layer whose gradient is small. But the gradient of Convolution layer is large, thus it seems to be spiky across X axis. The spiky gradient on X axis doesn't influence the training, while the smooth gradient on Y axis matters.



(a) $m = 0.99$ (b) $m = 0.6$ (c) $m = 0.5$

Figure 2. Gradient (ℓ_2 norm) landscape of various m. We provide ℓ_2 norm for each layer of the encoder (ResNet-18) with the training process. Across Y axis indicating the training epochs. Across X axis representing the ResNet layers, it shows the gradients of all layers including the BatchNorm layer whose gradient is small. But the gradient of Convolution layer is large, thus it seems to be spiky across X axis. And Z axis means the value of ℓ_2 norm. The spiky gradient on X axis doesn't influence the training, while the smooth gradient on Y axis matters. We can see that small $m = 0.6$ and 0.5 brings drastic volatility with the training process. The corresponding loss value and the gradient will fluctuate violently, resulting in bad convergence.

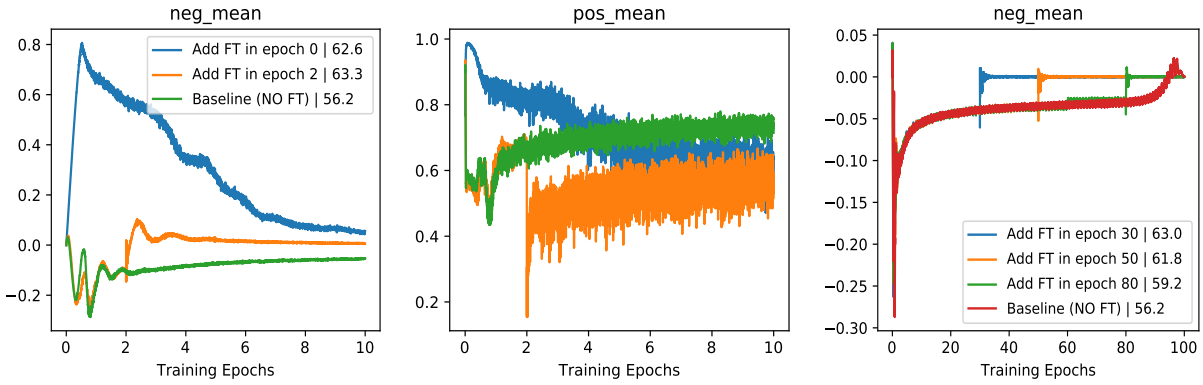
D. Discussion of when to Add the Feature Transformation

D.1. Effectiveness of our FT

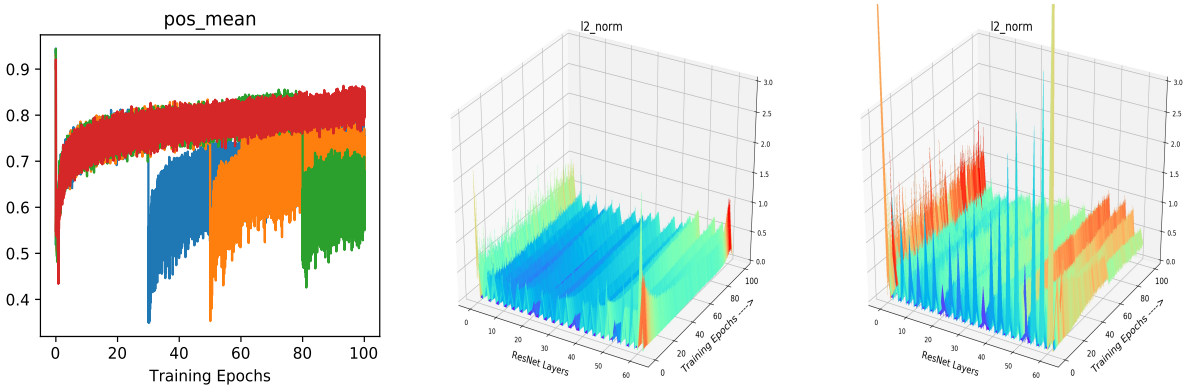
We present the efficacy of FT by analysis of starting FT in various training stages. As shown in Tab 2, starting FT (pos extrapolation + neg interpolation) from various epoch can boost the accuracy of baseline, and starting from earlier can improve more (7.1%/5.8% boosts with Res-18/Res-50). It is worthy to note that even adding FT in the 80th epoch can bring 3% improvement compared with the MoCo baseline (No FT in training). With the visualizations of score distribution Fig 3, we can see that our FT not only brings hard positives (lowering pos scores in Fig 3(d)) and hard negatives (rising neg scores in Fig 3(c)) simultaneously when the combined FT is inserted in various stages. The combination of positive extrapolation and negative interpolation can help rise the neg scores in the training process. Besides, with the comparison of the Gradient (ℓ_2 norm) landscape, we can observe that our FT brings a greater gradient for the training (Adding FT in the 30th epoch Fig 3(h) and 50th epoch Fig 3(i)), which makes the model escape from the local minima and avoid over-fitting. These analyses indicate our FT is a plug-and-play method and brings persistent view-invariance and discrimination for the training of contrastive models.

D.2. FT in the Early Training Stage

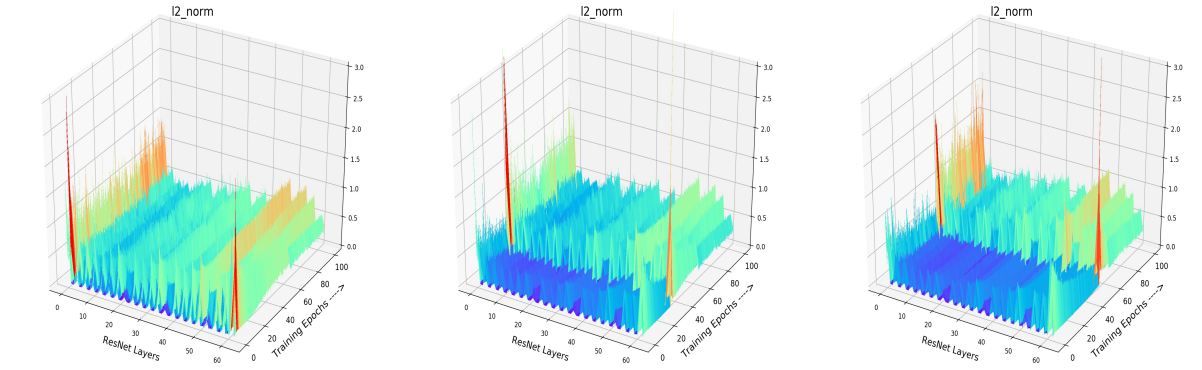
Due to the memory queue is initialized by random vectors at the start of training, the positive score and negative score have confusion, as shown in the visualizations in the early training stage (Fig 3(a) and Fig 3(b)). We provide the visualizations in the first 10 epoch to see the score distribution: (1) Adding FT from the 0th epoch will bring negative pairs whose score is very high (blue line in Fig 3(a), 0.8 negative score, which is too large for negative pairs), indicating the feature transformations for the random vectors will hurt the pair score distribution. From the perspective of gradient landscape in Fig 3(f), the initial gradient brought by FT is too sharp and not smooth for training compared with the baseline MoCo in Fig 3(e). (2) Adding FT from the 2nd epoch (In the 2nd epoch, the memory queue is filled by the semantic features from training data rather than the random vectors) will relieve solve too high negative scores (orange line in Fig 3(a), normal negative score) and meanwhile lower the positive score from easy positive to hard one (orange line in Fig 3(b), decreasing the positive score). The gradient (Fig 3(g)) seems more smooth and stable compare with starting FT from 0th epoch (Fig 3(f)). More importantly, in Tab 2, starting from the 2nd epoch (63.3%) can achieve slightly better accuracy than that at the beginning (62.6%). However, in the final experiments of imagenet-1K, we still use the strategy of starting FT from the 0th epoch. Because there seems no obvious performance difference in the ResNet-50 backbone in Tab 2. Future work will focus more on this issue.



(a) Mean of neg scores in early stage (b) Mean of pos scores in early stage (c) Mean of neg scores



(d) Mean of pos scores (e) Baseline MoCo landscape (f) Adding FT in 0th epoch



(g) Adding FT in 2nd epoch (h) Adding FT in 30th epoch (i) Adding FT in 50th epoch

Figure 3. Visualization of when to add FT, including score distribution and Gradient (ℓ_2 norm) landscape.

FT begin epoch	0	2	30	50	80	-
Res18 acc (%)	62.6	**63.3**	62.9	61.8	59.2	56.2
Res50 acc (%)	**76.9**	76.4	75.9	74.0	72.2	71.1

Table 2. When to add feature transformation. We employ Res-18 (total 100 epochs) and Res-50 (total 200 epochs) on IN-100 for the results. '-' indicates MoCo baseline without using any FT.

method	arch	acc
SimCLR	r50	74.32
SimCLR+pos extrapolation	r50	75.80
SimCLR+neg interpolation	r50	76.71
SimCLR+both	r50	78.25
SimCLR+both$_{dimension}$	r50	**78.81**

Table 3. Performance comparison of our proposed two feature transformation module on imagenet-100 with SIMCLR, the model are trained for 200 epochs. the last line with both$_{dimension}$ is mixing the feature (both pos/neg) using the dimension-level mixing, which shows improvements over the feature-level mixing.

Method	SimSiam	BYOL
baseline*	68.1	66.5
+pos extrapolation	68.7	67.2

Table 4. Comparision studies of proposed methods with non-contrastive methods. The models are pre-trained for 100 epochs with Res50 on IN-1K. * indicates reproduced baseline results.

E. Details of Comparison to other Methods

In this section, we discuss the details of how to apply our feature transformation to other self-supervised methods. We evaluate the performance of feature transformation on three representative methods, namely InfoMin, SwAV, and SimSiam.

For feature transformation on InfoMin, we perform both positive extrapolation and negative interpolation. Note that we perform the feature transformation on both branches of the InfoMin method, i.e. the original branch and the JigSaw branch. For feature transformation on SwAV, we only transform the two features of the input image by positive extrapolation, the rest of the SwAV pipeline is left unchanged. For SimSiam, as the method only uses positive pairs for training, so we only apply the positive extrapolation as the feature transformation. All the other hyperparameters are set to be the same as the original paper of each self-supervised method.

E.1. Additional experiments of our proposed feature transformation methods on SimCLR

To demonstrate the effectiveness of our feature transformation methods (Negative feature interpolation and Positive feature extrapolation), we also provide the experimental results on ImageNet-100 [12] of applying our method on another classic contrastive learning model, SimCLR [1]. Instead of using two encoders for encoding q and k like in MoCo [5], SimCLR directly uses a single network to encode the two views and contrast them against other negative examples. Because both MoCo and SimCLR are contrastive-based methods, the negative interpolation and positive extrapolation strategies can also be applied to SimCLR. We show the results of combining negative interpolation and positive extrapolation in Tab 3.

E.2. Apply Positive Extrapolation on Non-contrastive Models on IN-1K

Here we complement the results of applying positive extrapolation on non-contrastive models [4, 1]. The models are pre-trained for 100 epochs on IN-1K with the same data augmentation setting of the original paper. As shown in Table 4, we provide the IN-1k results (100ep) of BYOL/BYOL+posFT (66.5% -> 67.2%) and SimSiam/SimSiam+posFT (68.1% -> 68.7%) indicating pos extrapolation alone can help BYOL and SimSiam. Notice that we didn't perform the parameter experiments (not the optimal extrapolation parameter α_{ex}), so the improvement is slight.

F. Discussion of the feature normalization for FT

Here we provide additional visualization and analysis on the regular Feature Transformation (feature normalization, ℓ_2 normalization) due to its significant constriction (unit-sphere projection) and Whether to add ℓ_2 Normalization after our proposed FT.

F.1. Importance of Unit-sphere Projection

Unit-sphere projection (ℓ_2 norm) constricts the feature vector length from unbounded to 1, in the meanwhile retains the vector direction. Thus the pair scores $S_{q \cdot k}$ can be limited to $[-1, 1]$. Recent paper [7] concludes that unit sphere projection plays a key role in ensuring the large gradients of hard positives and negatives from the loss gradient properties. However,

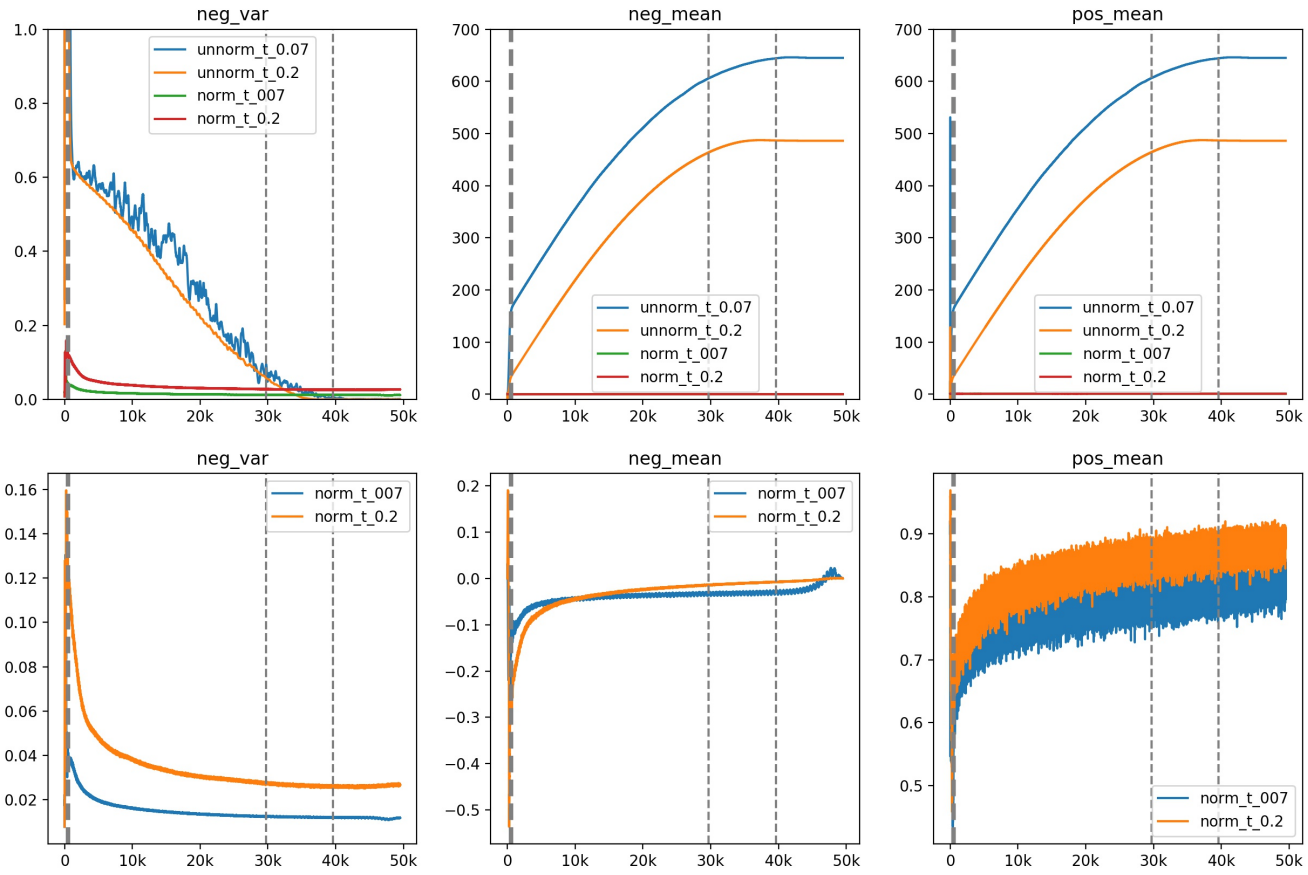


Figure 4. Pos/neg pair score distribution of unit sphere projection on ImageNet-100

Method	τ	lr	$Acc\%$
MoCo w/ unit sphere proj	0.07	0.03	65.04
MoCo w/ unit sphere proj	0.2	0.03	63.06
MoCo w/o unit sphere proj	0.07	0.03/10	*collapse*
MoCo w/o unit sphere proj	0.2	0.03/10	*collapse*

Table 5. The experiments for unit sphere projection on ImageNet-100

without the unit sphere projection, the feature vector length lost the constriction to $[-1, 1]$, and the too-large score distribution leads to bad contrastive learning and poor transfer performance (65.04% *v.s* $modelcollapse$). As shown in our empirical study (Fig 4 and Table 5) of this significant FT, the mean of positive pair score is similar to the mean of negative pair score when we removed unit-sphere projection, which will lead to an awful contrastive learning process: confusing the pos/neg pairs and bad gradient landscape brought by too large score distribution. Meanwhile, with the constrictions of unit-sphere projection, the mean of pos/neg pair scores are as expected: $neg \in [-0.2, 0]$ and $pos \in [0.6, 0.9]$, which can be discriminated by the log-softmax loss function. This limited small score distribution benefits the later contrastive learning and brought a stable training process. Finally, the variance of the negative pair score shows that model with unit-sphere projection will provide less volatile negative pairs, which is better for contrastive learning.

F.2. Whether to add ℓ_2 Normalization after FT

In this section, we provide empirical studies about whether re-perform the ℓ_2 norm for the transformed features after FT. As shown in Tab6, the performance difference is negligible for the model with/without re-performing the ℓ_2 normalization, (74.64% *v.s* post-norm 74.82% for negative interpolation, 72.80% *v.s* post-norm 72.45% for positive extrapolation, 76.87%

Method (MoCov1)	Acc%
baseline*	71.10
+pos extrapolation	72.80
+pos extrapolation$_{norm}$	72.45
+neg interpolation	74.64
+neg interpolation$_{norm}$	74.82
+both	76.87
+both$_{norm}$	76.68
+neg extrapolation	71.84
+neg extrapolation$_{norm}$	71.95

Table 6. Ablation studies of proposed methods on various contrastive models. The model are pre-trained for 200 epochs with Res50 on IN-100. The line with $norm$ is re-normalizing the transformed feature to the unit sphere, which show no improvements. * indicates reproduced baseline results.

Method (MoCov1)	Beta parameter	Acc%
baseline*	-	71.10
+neg interpolation	$Beta(1.6, 1.6)$	74.64
+neg extrapolation	$Beta(2.0, 2.0)$	71.84
+hard negative	$Beta(2.0, 1.0)$	73.45
+hard negative	$Beta(5.0, 2.0)$	74.32

Table 7. Ablation studies of proposed methods on various contrastive models. The model are pre-trained for 200 epochs with Res50 on IN-100. The line with $norm$ is normalizing the transformed feature to the unit sphere, which show no improvements. * indicates reproduced baseline results.

v.s post-norm 76.68% for combined FT). So we conclude that the transformed features are not necessarily on the unit sphere (*i.e.*has a norm of 1) due to the negligible performance difference. And in the final experiments of imagenet-1K, we do not re-perform the ℓ_2 norm after feature transformations. **However, we strongly recommend to re-perform ℓ_2 norm for the transformed features on all the datasets, for the sake of contrasting all the scores on the unit-sphere.**

G. Discussion of the Negative Feature Transformation

In this section, we provide more discussions about the feature manipulation of the negative examples. We have discussed negative interpolation to fully utilize negative features and increase the diversity of the memory queue. Here we provide the situation about negative extrapolation in memory queue and creating hard negatives.

G.1. Negative Extrapolation in Memory Queue

We have explored the negative interpolation to fully utilize negative features and increase the diversity of the memory queue. Then how about the negative extrapolation in the memory queue? Will the extrapolated negatives still be effective to increase the diversity of the memory queue and the performance?

Specifically, we denote the negative memory queue of MoCo as $Z_{neg} = \{z_1, z_2, \dots, z_K\}$ where K is the size of the memory queue, and Z_{perm} as the random permutation of Z_{neg}. We propose to use a simple extrapolation between two memory queue to create a new queue $\hat{Z}^{ex}_{neg} = \{\hat{z}^{ex}_1, \hat{z}^{ex}_2, \dots, \hat{z}^{ex}_K\}$:

$$\hat{Z}^{ex}_{neg} = \lambda_{ex} \cdot Z_{neg} + (1 - \lambda_{ex}) \cdot Z_{perm} \tag{1}$$

where $\lambda_{ex} \sim Beta(\alpha_{ex}, \alpha_{ex}) + 1$ is in the range of $(1, 2)$. The transformed memory queue $\hat{Z}_{neg}$ provides fresh extrapolated negatives for contrastive loss iteration by iteration. As shown in Tab 7, the negative extrapolation brings slight improvement over baseline (71.84% *v.s.* 71.10, 0.74% improved), while negative interpolation significantly improves to 74.64%. Both the negative interpolation and extrapolation can increase the diversity of the memory queue, but why extrapolation cannot boost the performance? We conjecture that the original queue Z_{neg} provides discrete distribution of negative samples but our method can fill in the incomplete sample points of the distribution by random interpolation, leading to a more discriminative model. But the extrapolated sample points may not stay in the previous manifold/distribution. Future work will focus more on this discussion.

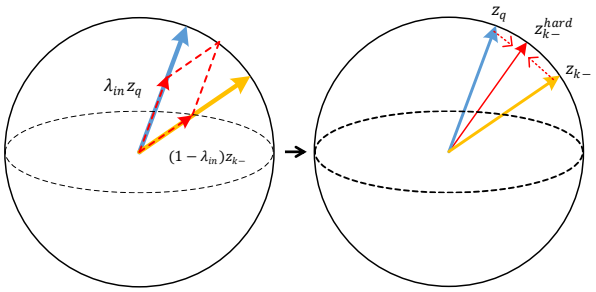


Figure 5. The process of creating hard negatives. The distance between the pos/neg feature vector is lowered, changing easy negatives to hard negatives, which is better for contrastive learning.

G.2. Creating Hard Negatives

The negative interpolation and extrapolation are both performed in the memory queue to increase the diversity. Another feature transformation for negative features is to increase the hardness during training, like the way of positive extrapolation. Our goal is to increasing the easy negative pair scores (similarity) to create hard negative pairs during training could be beneficial for the final transfer performance. Specifically, we use interpolation between z_q and all the negatives in the memory queue $Z_{neg} = \{z_1, z_2, \dots, z_K\}$ to create a hard negative queue $Z_{neg}^{hard} = \{z_1^{hard}, z_2^{hard}, \dots, z_K^{hard}\}$.

$$Z_{neg}^{hard} = \lambda_{in} \cdot z_q + (1 - \lambda_{in}) \cdot Z_{neg} \tag{2}$$

This equation indicates that each negative sample in the memory queue Z_{neg} will be interpolated with z_q to create hard negative queue Z_{neg}^{hard}. And $\lambda_{in} \sim Beta(\alpha_{in}, \alpha_{in})$ is in the range of $(0, 1)$. By this transformation, we can guarantee that the transformed neg score $S_{q \cdot k^-}^{hard}$ is larger than the original pos score $S_{q \cdot k^-}$, namely $z_q z_{k^-}^{hard} \geq z_q z_{k^-}$, which means we create a hard negative queue. Intuitively, it can be seemed like a simple approach to draw z_q and z_{k^-} closer in feature space. After interpolation, the distance between the pos/neg feature vector is lowered. Therefore this interpolation can serve as a feature transformation to create hard negatives from easy ones. As shown in Fig 5, it brings a minor direction change for positive/negative vectors. As shown in Tab 7, our hard negatives can bring consistent boosts over the baseline (74.32% *v.s.* (71.10%, 3.22% improved), indicating that this hard negative is effective for the contrastive learning. Future work will focus more on this topic. However, we choose the negative interpolation rather than the hard negative strategy in the final experiments of IN-1K. Because the computation of hard negative strategy is too large (Each z_q needs a new hard negative queue, so it takes time for one large batch to produce hard negative queue.).

References

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

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

[3] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[4] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. *NeurIPS*, 2020.

[5] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[6] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[7] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *arXiv preprint arXiv:2004.11362*, 2020.

[8] 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 *European conference on computer vision*, pages 740–755. Springer, 2014.

[9] Chao Peng, Tete Xiao, Zeming Li, Yuning Jiang, Xiangyu Zhang, Kai Jia, Gang Yu, and Jian Sun. Megdet: A large mini-batch object detector. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6181–6189, 2018.

[10] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[11] Zhiqiang Shen, Zechun Liu, Zhuang Liu, Marios Savvides, Trevor Darrell, and Eric Xing. Un-mix: Rethinking image mixtures for unsupervised visual representation learning. *arXiv preprint arXiv:2003.05438*, 2020.

[12] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. *ECCV*, 2019.

[13] Yonglong Tian, Chen Sun, Ben Poole, Dilip Krishnan, Cordelia Schmid, and Phillip Isola. What makes for good views for contrastive learning. *arXiv preprint arXiv:2005.10243*, 2020.