Learning Common Rationale to Improve Self-Supervised Representation for Fine-Grained Visual Recognition Problems

A. Algorithm

We present the pseudocode of our method in algorithm A1.

B. Understanding the Role of Each Projection

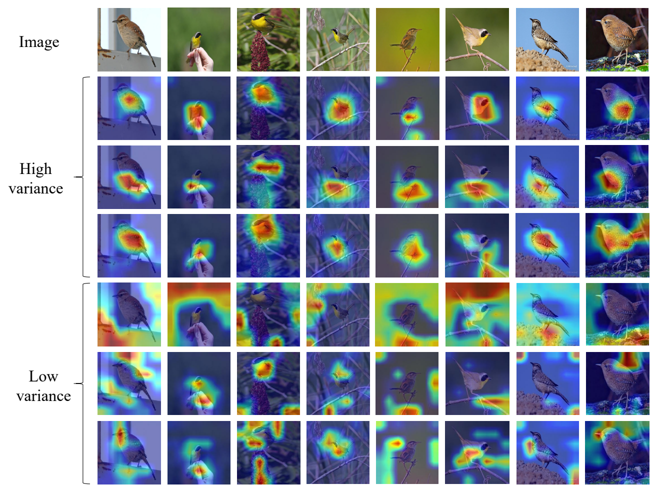

Figure B1. Visualization of high-and-low-variance projections of our method. The first row is the original input images. The 2-4 rows are the visualization of the top three high-variance linear projections, and the 5-7 rows are the visualization of the three projections corresponding to the lowest variance. Note that the colour of the heat map is normalized. It only indicates the relative strength within images, not the absolute value.

In this section, we explore the role of the K linear projections in the GradCAM Fitting Branch (GFB). To better analyze the importance of each projection, we consider the variance of the output of each projection. Higher variance indicates strong signal strength. Figure B1 visualize the output of the projection with the three highest variances and three lowest variances. It can be seen that projections with higher variance tend to produce more meaningful output. For example, the projection with the highest variance seems to capture clues that humans rely on to judge fine-grained objects, e.g., the head, body, and back of birds. In contrast, projections with the lowest variance barely attend the reasonable regions.

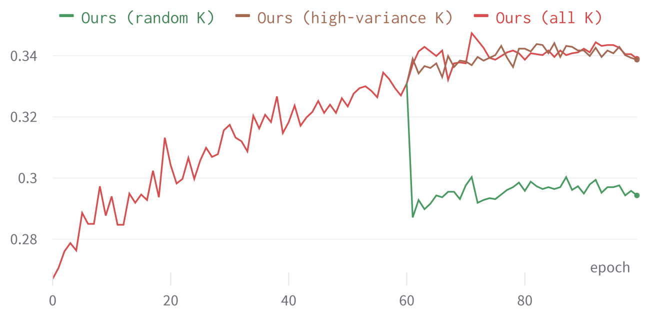

Figure B2. The rank-1 of retrieval task of our methods with different projections selection on the `FGVC Aircraft` datasets. "high-variance K" means selecting the first eight high-variance projections from 32 projections; "random K" represents randomly selecting eight projections from the remaining 24 projections after exclusive the first eight high-variance projections; "all K" represents all 32 projections selected.

Figure B2 shows another experiment to verify the role of different projections. In Figure B2, the curve shows the rank-1 accuracy of retrieval task on 100 epochs on the `FGVC Aircraft` datasets. From 0 to 60 epochs, we use all 32 projections (K=32) and find the rank-1 accuracy increases significantly. After 60 epochs, we test the three different settings: our method still with all 32 projections, referring as "Ours (all K)" in the red line; our method with the first eight high-variance projections, referring as "Ours (high-variance K)" in brown line; our method with random eight projections chosen from the remaining 24 projections after discarding the top-eight high-variance projections, referring as "Ours (random K)" in the green line. As we can see, the rank-1 greatly increase then gradually becomes stable in the method of Ours (all K) and Ours (high-variance K), while the rank-1 drop sharply then keep stable in the method of Ours (random K). This indicates that the top variance projection preserves the most discriminative rationales. Interestingly, the role of those top-variance projections is akin to that of the top eigenvectors in Principal Component Analysis.

Algorithm A1 Pseudocode of Learning Common Rationale on MoCo, PyTorch-style.

```
# $\psi_q$, $\psi_k$: encoder networks for query and key
# queue: dictionary as a queue of $Q$ keys ($C \times Q$)
# $m$: momentum
# $t$: temperature in contrastive loss
# $\tau$: temperature in KL Divergence loss

# Initialize key network parameters
$\psi_k$.params = $\psi_q$.params
# Load a minibatch $x$ with $N$ samples
for $x$ in dataloader:  do
    # two different random augmentations
    $x_q$ = aug($x$)
    $x_k$ = aug($x$)
    #****************************************
    # calculate MoCo contrastive loss
    #****************************************
    $q$ = $\psi_q$.forward($x_q$) # queries: $N \times C$
    $k$ = $\psi_k$.forward($x_k$)# keys: $N \times C$
    $k$ = $k$.detach()# no gradient to keys
    # positive and negative logits.
    logits_pos = bmm($q$.view($N$,1,$C$), $k$.view(N,C,1))# shape:$N \times 1$
    logits_neg = mm($q$.view($N$,$C$), queue.view(C,Q))# shape:$N \times Q$
    logits = cat([logits_pos, logits_neg], dim=1) # shape:$N \times (1 + Q)$
    # MoCo contrastive loss, positives are the 0-th.
    labels = zeros($N$)
    loss = CrossEntropyLoss(logits/$t$, labels)

    #****************************************
    #calculate KL loss
    #****************************************
    #get feature map in query network.
    feat_map=get_cov5($\psi_q$.params) #shape:$N \times 7 \times 7 \times 2048$
    #calculate gradient of feature map w.r.t. logits_pos.
    feat_grads=autograd.grad(logits_pos,feat_map)#shape:$N \times 7 \times 7 \times 2048$
    #calculate GradCAM map
    gradcam_map=get_gradcam(feat_grads) #shape:$N \times 7 \times 7$
    #calculate attention mask
    attention_mask=projections_max(feat_map)#shape:$N \times 7 \times 7$
    #KL loss: attention_mask and gradcam_map
    KL_loss=kl_div((attention_mask.view($N$,-1)/$\tau$).softmax(dim=-1).log(),(gradcam_map.view($N$,-1)/$\tau$).softmax(dim=-1))
    loss+=$\nu$*KL_loss
    loss.backward()

    #SGD update for query network
    update ($\psi_q$.params)
    #moment update for key network
    $\psi_k$.params = $m$*$\psi_k$.params + (1-$m$)*$\psi_q$.params
    #update dictionary: enqueue and dequeue
    enqueue(queue, $k$)
    dequeue(queue)
end for
```

C. Performance on Non-fine-grained Dataset

Table C1. Retrieval performance (%) of our proposed method vs. MoCo v2 on non-fine-grained dataset (`CIFAR-100`).

Method	Metrics		
	rank-1	rank-5	mAP
MoCo v2	12.01	30.30	3.04
Ours	27.14	51.25	6.44

To observe the performance of our proposed method on the non-fine-grained dataset, we conduct experiments of our method vs. MoCo v2 on `CIFAR-100` dataset shown in Table C1. From Table C1, we can see that our method also works for non-fine-grained cases.

D. The Impact of Number of Projections on Non-fine-grained Dataset.

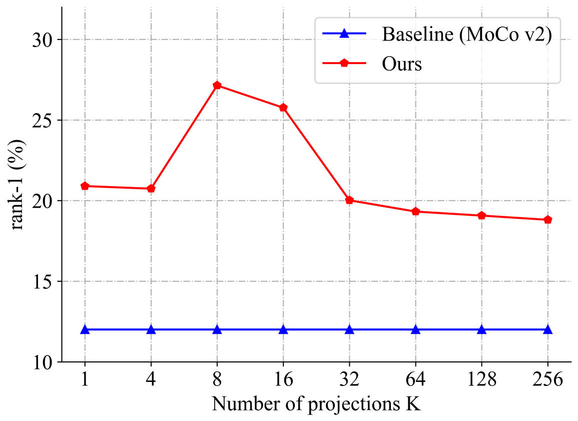

Figure D1. Comparison of MoCo v2 (the blue plot) baseline with our method (the red plot) w.r.t. K on the `CIFAR-100` dataset.

To explore the impact of the number of linear projections on the non-fine-grained dataset `CIFAR-100`, we conduct experiments with the different numbers of K. Figure D1 shows the retrieval results of rank-1 w.r.t. eight different projections. As we can see, the rank-1 peaks at 27.14% with K around 8. With the increase of K from 32, the performance decreases and then gradually becomes stable.

E. The Impact of GFB Structure on Non-fine-grained Dataset.

In this section, we investigate the impact of GFB structure for the non-fine-grained case by evaluating on `CIFAR-100`. We follow a similar experimental approach as in the main paper by varying the number of projections. As seen from Figure D1, the best number of projections seems to be 8, but further increasing the number of projections to 256 does not lead to a significant drop as the case for fine-grained dataset (refer to Figure 4 in the main paper).

Table E1. Retrieval performance (%) of our methods and using MLP as the alternative GFB branch. The evaluation is on the `CIFAR-100` dataset.

Dataset	Architecture	rank-1	rank-5	mAP
`CIFAR-100`	Ours(K=8)	**27.14**	**51.25**	**6.44**
	Ours(K=32)	20.02	40.52	4.35
	MLP	19.71	40.32	4.34

Also, we consider whether we can use multi-layer perception (MLP) to replace the maximized projections in the proposed method shown in Table E1. We find MLP performs similarly to our methods when $K \geq 32$. This is also different from the case in the fine-grained case. That observation indicates that GFB might have a different characteristic for non-fine-grained data. We plan to leave this in our future work.

F. Ablation study of weights λ and ν.

Table F1. Retrieval performance (%) w.r.t the impact of weights λ and ν on the `CUB-200-2011` dataset.

Method	λ	ν	rank-1
$\mathcal{L}_{CL}$+$\mathcal{L}_{KL}$ (Ours)	1	1	47.52
	1	0.1	48.83
	1	0.01	**49.69**
	1	0.001	46.63

Table F1 reports the retrieval performance for different values of the loss term coefficients in the proposed method, where λ and ν control the weight of $\mathcal{L}_{CL}$ and $\mathcal{L}_{KL}$, respectively. We set the weight of $\mathcal{L}_{CL}$ as 1 using a simple grid search method to find that the weight of $\mathcal{L}_{KL}$ as 0.01 is the best.

G. The Impact of Embedding Dimension.

Table G1. Rank-1 accuracy (%) of our method with different embedding dimensionalities on the retrieval task with 100 pretraining epochs on the `CUB-200-2011` dataset.

Dimensionality	64	128	256	512	1024
rank-1	47.48	48.72	**49.69**	49.46	48.55

Our method is based on the network structure of MoCo v2, and thus we observe the dependency on the dimensionality of the embedding vector shown in Table G1. Compared to MoCo v2 using 128 as the dimensionality, the impact of embedding vector in our method is different.

The performance on the `CUB-200-2011` dataset increases firstly from 47.48% rank-1 to 49.69 rank-1 on retrieval task with the dimensionality from 64 to 256. After 256, the performance decreased. Thus, our method uses 256 as the dimensionality of the embedding vector.