Fast-MoCo: Boost Momentum-based Contrastive Learning with Combinatorial Patches

Yuanzheng Ci[1], Chen Lin[2], Lei Bai[3⋆], and Wanli Ouyang[3,1]

[1] The University of Sydney, SenseTime Computer Vision Group
{yuanzheng.ci,wanli.ouyang}@sydney.edu.au
[2] University of Oxford
chen.lin@eng.ox.ac.uk
[3] Shanghai AI Laboratory
bailei@pjlab.org.cn

Abstract. Contrastive-based self-supervised learning methods achieved great success in recent years. However, self-supervision requires extremely long training epochs (e.g., 800 epochs for MoCo v3) to achieve promising results, which is unacceptable for the general academic community and hinders the development of this topic. This work revisits the momentum-based contrastive learning frameworks and identifies the inefficiency in which two augmented views generate only one positive pair. We propose Fast-MoCo - a novel framework that utilizes combinatorial patches to construct multiple positive pairs from two augmented views, which provides abundant supervision signals that bring significant acceleration with neglectable extra computational cost. Fast-MoCo trained with **100** epochs achieves **73.5**% linear evaluation accuracy, similar to MoCo v3 (ResNet-50 backbone) trained with 800 epochs. Extra training (**200** epochs) further improves the result to **75.1**%, which is on par with state-of-the-art methods. Experiments on several downstream tasks also confirm the effectiveness of Fast-MoCo.[†]

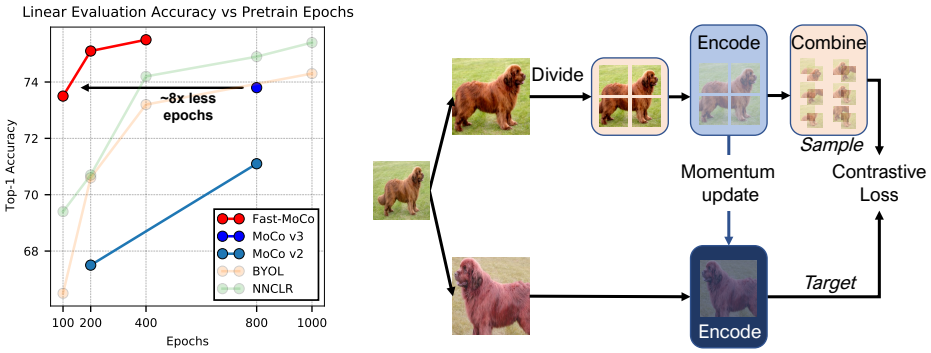
Keywords: Self-Supervised Learning, Contrastive Learning

1 Introduction

Self-supervision is crucial in some of the most remarkable achievements from natural language processing (NLP) [10,2] to computer vision [6]. In particular, recent advances in contrastive learning produced state-of-the-art results on self-supervised learning benchmarks [15,9,29]. Contrastive learning performs an instance discrimination pretext task by attracting the embedding of positive samples closer while encouraging the negative samples to be further apart. Some methods opt to make the sample pairs asymmetric with tools such as momentum encoder [18], predictor [15] and `stop-grad` [8] to provide more flexibility for architecture design [15,13].

⋆ Corresponding author

† Code and pretrained models are available at https://github.com/orashi/Fast-MoCo



(a) (b)

Fig. 1: (a): Comparison with state-of-the-arts on ImageNet. All methods uses ResNet-50 encoders and are measured with Top-1 linear evaluation accuracy. (b): Overview of Fast-MoCo that includes the Split-Encode-Combine pipeline.

While great advances have been achieved in the self-supervised learning area in the past two years, a major concern about these works is the extremely long training steps to get a promising performance(e.g., normally 800 epochs, and even 1000 epochs for some methods [9,15,32,11]), which makes it hard or even impossible for many academics to contribute to this area. High training cost also posts challenges when dealing with large industry scale datasets [1,17]. In order to accelerate training, we spotted one limitation of recent momentum based contrastive learning methods [18,7,15], which is the *two-image-one-pair* strategy. In this strategy, two images (or two augmented views of the same image) are fed to the deep models separately and then used as one pair for contrastive learning in [18,7,9,11]. Although symmetric loss designs are normally employed to improve the sample efficiency, we argue that the *two-image-one-pair* mechanism is sub-optimal. To overcome this issue, we propose combinatorial patches, a novel mechanism to efficiently generate feature embeddings for arbitrary combination of local patches. In this strategy, an image pair can be used for generating multiple positive pairs for contrastive learning. Therefore, in contrast to the *two-image-one-pair* mechanism in existing works, our combinatorial patches enable the *two-image-multi-pair* mechanism. With more pairs used for contrastive learning using this *two-image-multi-pair* mechanism, our Fast-MoCo method trained using 100 epochs based on MoCo v3 (*two-image-one-pair* mechanism) for ResNet50 can achieve on-par accuracy when compared with MoCo v3 trained using 800 epochs, as shown in Fig. 1(a).

To implement the *two-image-multi-pair* mechanism, this paper proposes the Divide-Encode-Combine and then Contrast pipeline as shown in Fig. 1(b). In detail, we divide the input into multiple local patches without overlap in the data preparation stage and encode the local patches by deep models separately, then combine the encoded features of multiple patches before computing the con-

trastive loss. We validate various strategies and hyperparameters for both divide and combine stages and provided a detailed analysis across different settings.

We evaluate our method on ImageNet with the ResNet-50 backbone. In a linear evaluation setting, our method achieves 73.5% with only 100 epochs of SSL pretraining, which is 8× faster than the original MoCo to achieve comparable performance. A longer training (400 epochs) further boosts the performance from 73.5% to 75.5%. We also tested the learned embeddings in semi-supervised learning, object detection, and instance segmentation. Our method performs better than previous approaches in both settings, which suggests the embeddings learned with our method are general and transferable.

2 Related Works

2.1 Patch Based Representation Learning

Various self-supervised learning methods [25,26,21,13,5,27,1,17] manipulates image patches. A common way to incorporate patches is to encode them separately [25,26,21,13], while Jigsaw Clustering [5] encodes multiple patches at the same time: patches are augmented independently and stitched to form a new image for encoding, the encoded features are then separated spatially before pooling to get the embedding for each patch. Either way, the encoded embeddings can then be used for solving jigsaw puzzles [25,5], contrastive prediction [26,21,5] or bag-of-word reconstruction [13]. On the other hand, Context encoder [27] encodes an image with random masking and then learns to reconstruct the missing part with a decoder. With a ViT encoder, BEiT [1] and MAE [17] split the image into a grid of patches and mask out some of them, the rest patches are gathered and forwarded to get encoded embeddings. They are then optimized for reconstructing the missing patches at feature-level [1] or pixel-level [17]. However, these methods do not construct multiple pairs of samples from combinatorial patches and thus are different from our Divide-Encode-Combine pipeline.

2.2 Contrastive Learning

Contrastive learning methods [16,6,3] have attracted many attentions for their simplicity and performance. They retrieve useful representations by promoting instance discrimination, where the positive samples are generated by applying different data augmentations to the same image while having an identical spatial size. SwAV [3] and NNCLR [11] further extend the semantic gap between a positive pair with a target embedding being replaced by a learned cluster center and a neighborhood embedding. Since the methods in [16,6,3,11] are not momentum-based learning, our method does not aim at improving them. Besides, our proposed Divide-Encode-Combine scheme is not investigated in them.

Momentum-based contrastive learning methods adopt an asymmetric forward path. On the online path, an input image is fed into the encoder. On the target path, another input image is fed into a slowly moving momentum

encoder [18,7,9]. The two encoded samples from these two paths form a pair for contrastive learning, which has been proven to be effective in many scenarios [13,15,4]. However, these works adopt the *two-image-one-pair* mechanism. In contrast, our Fast-MoCo adopts a *two-image-multi-pair* mechanism. At almost the same training cost of the *two-image-one-pair* mechanism, Fast-MoCo generates more sample pairs in a mini-batch for efficiency.

3 Method

In this Section, we first give preliminaries about MoCo, which is adopted as our baseline. Then, we introduce the design of combinatorial patches, which boost both the learning process and performance. Finally, we discuss how the proposed approach will affect the performance and computation.

3.1 Preliminaries about MoCo

MoCo is a highly recognized framework for self-supervised learning, which has three versions, i.e., MoCo [18], MoCo v2 [7], and MoCo v3 [9], which gradually incorporate some of the best practice in the area. Specifically, MoCo v3 pipeline has two branches, i.e., an online branch and a target branch. The online branch consists of an encoder f (e.g., ResNet50), a projector g, follow by a predictor q. The target branch only contains the encoder and projector with the same structure as in the online branch and its parameters are updated through an exponential moving average process as follows:

$$\theta_t^f \leftarrow \alpha\theta_t^f + (1-\alpha)\theta_o^f, \quad \theta_t^g \leftarrow \alpha\theta_t^g + (1-\alpha)\theta_o^g, \tag{1}$$

where θ_o^f and θ_o^g are parameters for encoder and projector in the online branch, θ_t^f and θ_t^g are parameters for encoder and projector in the target branch. This asymmetric architecture design and the use of moving average for target branch parameters updating have been shown to help the model avoid collapse [15].

Given an image x, two different views are generated through two different augmentations a and a', which are then forward to the encoders in the online and target branches respectively to retrieve the encoded embeddings as a positive pair (v_o^a, $v_t^{a'}$). These embeddings are then projected to vectors $z_o^a = q(g(v_o^a; \theta_o^g); \theta_o^q)$ and $z_t^{a'} = g(v_t^{a'}; \theta_t^g)$. Finally, the loss function for this pair $(z_o^a, z_t^{a'})$ is formulated by InfoNCE [26] as follows:

$$\mathcal{L}_{ctr}(z_o^a, \mathbf{z}_t^{a'}) = -log\frac{exp(z_o^a \cdot z_t^{a'}/\tau)}{\sum\limits_{z \in \mathbf{z}_t^{a'}} exp(z_o^a \cdot z/\tau)}, \tag{2}$$

where $\mathbf{z}_t^{a'}$ denotes the set of target representations for all images in the batch. Note that vectors z, z_o^a, and $z_t^{a'}$ are l_2 normalized before computing the loss. Besides, for every sample image x, this loss is symmetrized as:

$$\mathcal{L}_x = \frac{1}{2}(\mathcal{L}_{ctr}(z_o^a, \mathbf{z}_t^{a'}) + \mathcal{L}_{ctr}(z_o^{a'}, \mathbf{z}_t^a)). \tag{3}$$

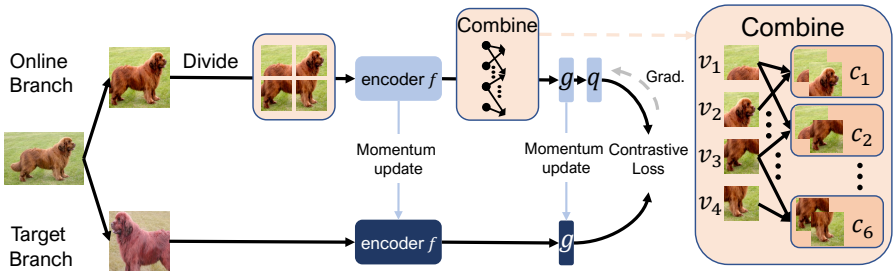


Fig. 2: Overview of Fast-MoCo framework. It consists of four steps: 1) *Divide* step, where the input image in the online branch is divided into multiple patches; 2) *Encode* step, which the encoder f encodes the features of the patches separately; 3) *Combine* step, which combines the encoded features (at the last layer of the neural network); 4) the combined features are fed into projector g, predictor q, and contrastive loss for contrastive learning. Compared with MoCo, we add the Divide step and Combine Step in the online branch, with details in Section 3.2. The target branch is the same as MoCo.

3.2 Fast-MoCo

In this section, we introduce Fast-MoCo, a simple method that can greatly improve the training efficiency of self-supervised learning with negligible extra cost. An overview of Fast-MoCo is shown in Fig.2. With MoCo v3 as the baseline, Fast-MoCo only makes three modifications, 1) add a *Divide* step to divide an image into multiple patches before sending the patches to the encoder [‡] of the online branch, 2) insert a *Combine* step (e.g., Combine) immediately behind the encoder to combine patches, and 3) a slightly modified definition of positive and negative pairs corresponding to the divide and combine operations. In the following, we illustrate the Divide step, Combine step, and the modified loss function in detail.

Divide Step. For the online branch, instead of directly feed the given the augmented image x^a into the encoder, we first divide it into a $m \times m$ grid of patches $\{x_p | p \in \{1, \ldots, m^2\}\}$ as shown in Fig.2, with $\mathbf{p}$ denotes the set of patch index $\{p\}$. The influence of m in will be analyzed in Section 5.4.

Combine Step. Instead of directly using the encoded embedding of each patch individually for further step, we combine multiple (less than m^2) patch embeddings v_p to form combined embeddings c before sending them to further step, i.e., the projector.

To form a combined embedding, we take a subset of n indices from the patch index set $\mathbf{p}$, noted as $\mathbf{p}_n (\subseteq \mathbf{p})$, and collect their corresponding features $\mathbf{v}_{\mathbf{p}_n} = \{v_p | p \in \mathbf{p}_n\}$. While there could be diverse options to combine multiple embeddings (e.g., concatenate, sum), we empirically found that simply averag-

[‡] In this paper, we only explore the ResNet50 as the encoder while leaving the evaluation of ViT version MoCo v3 as our future work.

ing the selected features works reasonably well and is computationally efficient. Thus, in the Combine step, we generate the combined embedding by:

$$c = \frac{1}{n} \sum_{p \in \mathbf{p}_n} v_p. \tag{4}$$

To improve the sample utilization efficiency, we take all possible n-combinations of patch embeddings for supervisions, leading to the combined embedding set $\mathbf{c} = \{c_i | i \in \{1, \ldots, C_{m^2}^n\}\}$, where $C_m^n = \frac{m!}{n!(m-n)!}$. In this way, we can generate many samples by the averaging operation in Eq. 4 with negligible extra cost and ensure the sample and the target have a sufficient information gap since the combined patches embedding only covers part of the image information.

After the Combine step, the projector and the predictor in the online branch transfer each combined embedding c to vector z_o^c in a sequential manner. On the other hand, the target branch maps the other input view to $z_t^{a\prime}$ in the same manner as the basic MoCo v3 without modification. They are then L2-normalized and used for computing contrastive loss.

Loss Functions. Like MoCo v3, we still utilize the contrastive loss (Eq. 2) to optimize the encoder, projector, and predictor. Compared with MoCo v3, Fast-MoCo does not include any extra parameters to be learned, the only difference is that there are multiple ($C_{m^2}^n$) combined patch embeddings z_o^c instead of one image embedding z_o^a corresponding to a target branch image embedding $z_t^{a\prime}$. We directly adapt the original loss function by averaging the contrastive losses from $C_{m^2}^n$ positive pairs between the combined patch embeddings z_o^c and target image embedding z_t. Similarly, the negative pairs are defined between the combined patch embedding and the embedding of other images in the target branch.

3.3 Discussion

In this section, we present some intuitive analysis about why Fast-MoCo can improve training efficiency, which will be further demonstrated with empirical results in Section 4. The primary component that makes Fast-MoCo converge faster is the utilization of a set of combined patch embeddings, which significantly increase the number of positive pairs. Take $m = 2$ and $n = 2$ as an example, Fast-MoCo will divide the input image in the online branch into four patches and then combine their four embeddings into six, each of which represents two patches, directly expanding the number of positive pairs six times more than MoCo v3. Thus, Fast-MoCo can get more supervision signals in each iteration compared to MoCo v3 and thus achieves promising performance with fewer iterations.

At the same time, the introduced operations in Fast-MoCo, i.e., divide an image into patches and average the representation of several patches, are extremely simple and only require negligible extra computation. The major computational cost is introduced by additional forwards over the projector and the predictor in the online branch. However, they only involve the basic linear transformations, which contributes little cost when compared to the backbone. Thus, the total overhead of Fast-MoCo accounts for 7% extra training time compared to MoCo

v3 (38.5 hours on 16 V100 GPUs for 100 epochs, by contrast, MoCo v3 costs 36 hours under the same setting)

Besides, since the combined patch embeddings only contain part of the information in the whole image, pulling the partially combined patches closer to the target view that contains the whole image information is more challenging than pulling the original image pairs and implicitly increasing the asymmetric of the network structure, which have been demonstrated beneficial for increasing the richness of feature representations and improve the self-supervised learning performance [15,11,22]. Owing to these merits, Fast-MoCo can achieve high sample utilization efficiency with marginal extra computational cost and thus obtain promising performance with much less training time. Experimental results in Section 5.2 and 5.4 below will validate these analysis.

4 Experimental Results

4.1 Implementation Details

The backbone encoder f is a ResNet-50 [20] network excluding the classification layer. Following SimSiam [8] and MoCo v3 [9], projector g and predictor h are implemented as MLP, with the detailed configuration identical to [8]. For self-supervised pretraining, we use SGD optimizer with batch size 512, momentum 0.9, and weight decay $1e^{-4}$. The learning rate has a cosine decay schedule from 0.1 to 0 with one warm-up epoch starting from 0.025. We use the same augmentation configurations as in SimSiam [8] (see supplementary material).

4.2 Results

ImageNet Linear Evaluation. Following [6,8,15], we evaluate our method with a linear classifier on top of frozen embeddings obtained from self-supervised pretraining. The classifier is finetuned with LARS optimizer [31] with configurations same as SimSiam [8] excepting the learning rate which we set as $lr = 0.8$. We compare with existing methods in Table 1, Our Fast-MoCo achieved 75.5% linear evaluation result with only 400 epochs of training, which shows obvious improvement of our Fast-Moco compared with all methods using two augmented views for supervision. When considering the same amount of training epoch, our result also surpass SwAV [3] and DINO [4] even including the use of `multi-crop` [3]. Note that our new design is orthogonal to `multi-crop` [3] (details in Section 5.3) and the novel designs in SwAV, DINO and NNCLR.

Semi-Supervised Learning. Following the semi-supervised learning setting in [6], we fine-tune our model pretrained by 400 epochs with 1% and 10% of the data split. The results are shown in Table 2. Our method performs better than all compared methods w/o `multi-crop` and is on par with SwAV using `multi-crop`.

Transfer Learning. Table 3 shows experimental results evaluating the effectiveness of the learned model when transferred to detection and segmentation tasks. For object detection on PASCAL-VOC [12], with Faster R-CNN [28]

Method	100 ep.	200 ep.	400 ep.	800 ep.	1000 ep.
SimCLR [6]	64.8	67.0	68.3	69.1	-
MoCo v2 [7]	-	67.5	-	71.1	-
BYOL [15]	66.5	70.6	73.2	-	74.3
SwAV [3]	-	-	70.1	-	-
BarlowTwins [32]	-	-	-	-	73.2
SimSiam [8]	68.1	70.0	70.8	71.3	-
MoCo v3 [9]	-	-	-	73.8	-
NNCLR [11]	69.4	70.7	74.2	74.9	75.4
OBoW [13]	-	73.8	-	-	-
Fast-MoCo	**73.5**	**75.1**	**75.5**	-	-
SwAV [3] (w/ `multi-crop`)	72.1	73.9	-	75.3	-
DINO [4] (w/ `multi-crop`)	-	-	-	75.3	-
NNCLR [11] (w/ `multi-crop`)	-	-	-	**75.6**	-

Table 1: **ImageNet-1k linear evaluation results** for existing methods and our Fast-MoCo using ResNet-50. Best results are in **bold**. Fast-MoCo can achieve similar performance as MoCo v3 with only 100 epochs. When trained for 200 epochs, Fast-MoCo performances better than MoCo v3 trained for 800 epochs and is comparable with state-of-the-arts (`multi-crop` is not used in Fast-MoCo for a fair comparison).

framework, we have all weights finetuned on the `trainval07+12` dataset and evaluated on the `test07` dataset. For detection and instance segmentation on COCO [23], we finetune our weights with Mask R-CNN [19] on the `train` set and report results on the `val` split. The results in Table 3 show that our Fast-MoCo performs on par with or better than the state-of-the-arts in localization tasks.

5 Analysis

5.1 Same or Different Augmented Views

Recent works [6,15] have indicated that contrastive methods are sensitive to augmentations, especially spatial transformations [6]. Compared with the conventional settings of having different augmented view (73.5% on ImageNet for 100-epoch training of Fast-MoCo), we observe severe drop of accuracy (48.5%) if the positive embedding pair in Eq. (5) are from the same augmented view, i.e. $a' = a$. When the same augmented view is used, the detrimental non-semantic information contained in patches would be exposed to its contrastive target, which causes the significant drop of accuracy. These results show the importance of using appropriate targets for contrastive learning.

5.2 Comparison on Patch Encoding Approaches

Apart from our proposed Fast-MoCo pipeline, there is also a number of alternatives [25,26,21,13,5,27,1,17] that falls into the same category with our Fast-

Method	1%		10%	
	Top-1	Top-5	Top-1	Top-5
Supervised	25.4	48.4	56.4	80.4
InstDisc [30]	-	39.2	-	77.4
PIRL [24]	-	57.2	-	83.8
SimCLR [6]	48.3	75.5	65.6	87.8
BYOL [15]	53.2	78.4	68.8	89.0
Barlow Twins [32]	55.0	79.2	69.7	89.3
NNCLR [11]	56.4	80.7	69.8	89.3
Fast-MoCo	**56.5**	**81.1**	**70.3**	**89.4**
SwAV [3] (w/ `multi-crop`)	53.9	78.5	70.2	89.9

Table 2: **Semi-supervised learning results on ImageNet-1K** with ResNet-50 backbone. We report Top-1 and Top-5 accuracies for models finetuned with 1% and 10% labeled data. Detailed configuration can be found in supplementary material.

Method	VOC det			COCO det			COCO seg		
	AP_{all}	AP_{50}	AP_{75}	AP^{bb}_{all}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}_{all}	AP^{mk}_{50}	AP^{mk}_{75}
Supervised	53.5	81.3	58.8	38.2	58.2	41.2	33.3	54.7	35.2
MoCo V2 [7]	57.4	82.5	64.0	39.3	58.9	42.5	34.4	55.8	36.5
SimSiam [8]	57	82.4	63.7	39.2	**59.3**	42.1	34.4	**56.0**	36.7
Barlow Twins [32]	56.8	82.6	63.4	39.2	59.0	42.5	34.3	**56.0**	36.5
Fast-MoCo	**57.7**	**82.7**	**64.4**	**39.5**	59.2	**42.6**	**34.6**	55.9	**36.9**
SwAV [3] (w/ `multi-crop`)	56.1	82.6	62.7	38.4	58.6	41.3	33.8	55.2	35.9

Table 3: **VOC and COCO object detection (det) and instance segmentation (seg) results**. We report results measured by Average Precision (AP) using ResNet50 with the C4 backbone variant [14]. For VOC dataset, we train on `trainval07+12` and evaluate on `test07` by running three trials and report the averaged results.

MoCo which does not apply the *two-image-one-pair* mechanism. In this Section, we provide a detailed comparison on these variants.

Sample-Encode-Combine. The compared settings contain cases where patches can not be generated from dividing a 224×224 view. Apart from the Fast-MoCo baseline, we set up a Sample-Encode-Combine (SEC) configuration for fair comparison. In SEC configuration, we replace the 'Divide' step in Fast-MoCo by randomly and independently sampling patches. In contrast to Fast-MoCo with 2×4 patches divided from two 224×224 views, for SEC we have eight independently sampled patches :$\{x_p | p \in \{1, \ldots, 8\}\}$ and two 224×224 target $\{x_t^a, x_t^{a'}\}$. As x_p for SEC are not devided from the target views x_t. The embeddings of all eight x_p can be combined with each other to get combined embedding c, we have the amount of combination increased from $2C_4^2 = 12$ to

Method	Num. of Samples	Top-1
Encode Only	4	68.9
Sample-Combine-Encode	4	71.2
Divide-Combine-Encode	4	71.8
Montage-Encode-Divide-Combine	28	70.4
Sample-Encode-Combine	28	72.9
Fast-MoCo	12	73.5

(a) **Comparison of patch encoding approaches.** Results are based on ImageNet linear evaluation, all models are pretrained for 100 epochs.

Case	`multi-crop`	Comb.	Top-1
MoCo v3	-	-	70.3
(i)	✓	-	73.1
(ii)	-	✓	73.5
(iii)	✓	✓	74.2

(b) **Relationship with `multi-crop`.** 'Comb.' denotes the usage of combinatorial patches. Results are linear evaluation on ImageNet, all models are pretrained for 100 epochs.

Table 4

$C_8^2 = 28$. The loss function for SEC is written as follows:

$$\mathcal{L}_x = \frac{1}{2C_8^2} \sum_{c \in \mathbf{c}} (\mathcal{L}_{ctr}(z_c, \mathbf{z}_t^a) + \mathcal{L}_{ctr}(z_c, \mathbf{z}_t^{a'})), \tag{5}$$

It obtains 72.8%, which is the second-best among all variants in Table 4(a).

Encode Only. A widely adopted way to encode patches is to encode them separately [25,26,21,13], which do not include the 'Divide' step or 'Combine' step in our Fast-MoCo as depicted in Fig. 2. For a fair comparison, the patch used for encoding should contain approximately the same amount of information as two 112×112 patches combined, so we set the spatial size of the patch as 158×158. In doing so, we can no longer retrieve these patches by dividing a 224×224 that we use for contrastive target, thus they are independently generated by augmentation as described in Section 4.1. We generate four 158×158 patches $\{x_p\}$ and two 224×224 target $\{x_t^a, x_t^{a'}\}$, for each image x we have:

$$\mathcal{L}_x = \frac{1}{8} \sum_{z_p \in \mathbf{z}_p} (\mathcal{L}_{ctr}(z_p, \mathbf{z}_t^a) + \mathcal{L}_{ctr}(z_p, \mathbf{z}_t^{a'})), \tag{6}$$

where $\mathbf{z}_{target}$ denotes the target vectors in a mini-batch and $\mathbf{z}_p$ denotes the features of the four patches sampled from the image x. As shown in Table 4(a), the result of Encode Only is 68.9%.

Divide(Sample)-Combine-Encode. While Fast-MoCo encodes the small divided patches independently and combines them at embedding level; one can also combine them at image level with patches placed in their original positions, thus preserving the relative positional information among patches. Note that if the stitched image is not in a rectangular shape, the redundant computational cost would be hard to avoid for a CNN encoder. In the Divide step, we divide a 224×224 image vertically and horizontally to get four 112×112 patches.

In the Combine step for Divide-Combine-Encode, two 112×112 patches are stitched to 112×224 or 224×112 at image level. The Divide step, Encode step, and losses are the same as Fast-MoCo. As shown by Divide-Combine-Encode in Table 4(a), compared to Encode Only with four squared 158×158 crops, these rectangular crops with less locally-bounded features is preferred with a +2.9 gain. Divide-Combine-Encode can also be viewed as bringing the Combine step of our Fast-MoCo pipeline before the encoding step. Compared with the Fast-MoCo pipeline, 1) the Fast-MoCo Divide-Combine-Encode pipeline generates fewer target-sample pairs for the same computational cost, and 2) does not include sufficiently difficult target-sample pairs (more discussion in Section 5.4).

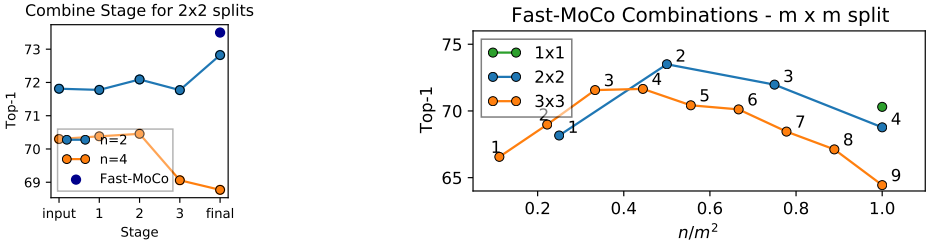
For the *Sample-Combine-Encode* in Table 4(a), we generate the 112×112 rectangular patches independently, and find its +2.3 gain over Encode Only. Sample-Combine-Encode performs worse than Divide-Combine-Encode because the divided patches in Divide-Combine-Encode have no overlap, which maximizes the diversity of the combined patches, but Sample-Combine-Encode cannot guarantee non-overlapping patches.

Montage-Encode-Divide-Combine. JigClu [5] proposed a patch encoding technique with montage image. Given a batch of K images, four patches are generated from each image with different augmentations, resulting in a mini-batch of $4K$ patches. Then K montage images of size 224×224 are generated by stitching four patches randomly selected (without replacement) from the mini-batch of $4K$ patches. The encoder adds an additional step before average pooling to divide K montage feature maps back to $4K$ patch features to get their encoded embeddings. We replaced our Divide-Encode steps with this Montage-Encode-Divide approach, forming a Montage-Encode-Divide-Combine pipeline. The result of this approach in Table 4(a) shows that it is not as good as the relatively simpler Fast-MoCo approach.

Analysis All in all, our Fast-MoCo outperform other variants with a steady margin. The Encode Only baseline achieves 68.9%. If we combine inputs before the encoding mechanism, the performance improved to 71.2% and 71.8% for inputs obtained by random cropping and dividing respectively. If we combine the embedding after encoding inputs, the performance improved to 72.9% (sample by random cropping) and 73.5% (Fast-MoCo). The Montage strategy achieves 70.4%. We find that the Sample (random cropping) always performs worse than Divide, and combine after encoding always better than before encoding in our experiments. Based on these results, we found non-overlapping patches(Divide) and Combine after encoding to be the best practice.

5.3 Relationship with Multi-Crop

`Multi-crop` is a technique proposed in SwAV [3]. In addition to two 224×224 crops, `multi-crop` additionally adds six 96×96 patches as samples so that the encoder is trained with samples that have multiple resolutions and hard samples. However, the additional samples also needs more computation. While both Fast-MoCo and `mulit-crop` use small patches as their input, Fast-MoCo is not trained with samples of multiple resolutions. Except the (iii) in Table 4(b), all



(a) (b)

Fig. 3: (a): ImageNet linear evaluation accuracy (Y-axis) when different ResNet stages (X-axis) are selected for combining $n = 2$ divided patches or $n = 4$ divided patches in the Divide step. (b): ImageNet accuracy (Y-axis) when n/m^2 (X-axis) patches are combined for $m \times m$ (1×1, 2×2, 3×3) divided patches. Annotations represent the number of combined samples n.

reported results in this paper for Fast-MoCo are w/o `mulit-crop`. Nevertheless, as shown by (ii) in Table 4(b), Fast-MoCo w/o `mulit-crop` is 0.4 better than MoCo v3 w/ `mulit-crop`. Fast-MoCo w/ `mulit-crop` (see supplementary material for details), i.e. (iii) in Table 4(b), further improves the result of Fast-MoCo by 0.7, which shows that our contribution is orthogonal to `mulit-crop`.

5.4 Ablation on Fast-MoCo

Combine Stage and Task Difficulty In our Fast-MoCo pipeline, a 224×224 cropped image is divided into four patches. The embeddings of these four patches are combined at the final layer of the ResNet encoder. In this Section, we investigate the influence of combining $n = 2$ patches or $n = 4$ patches. When $n = 2$, there is an information gap between sample and target because the sample only has half of its patches used for contrastive loss. When $n = 4$, all information within the original image is preserved. When combining two patches (or their feature maps) before the last stage, as it is difficult to handle non-rectangle input for CNN, we only stitch them vertically and horizontally with respect to their original position as described in Section 5.2. Since convolution layers are computationally heavy, we do not reuse patches/patch feature maps, so uniformly, we have two target-sample pairs per image when $n = 2$. In the case of the final layer, for a fair comparison, we adopt the same sample pair selected as in previous stages, which means two target-sample pairs per image.

In Figure 3(a), the results show that when the Combine step took place at the embedding level, i.e., the elimination of relative positional information between patches at later stages, it is beneficial when there is an information gap between sample and target ($n = 2$). However, it will be harmful when there is no gap ($n = 4$). On the other hand, we can see the training does benefit from a harder task, i.e., presence of information gap between sample and target when $n = 2$.

While for our Fast-MoCo, it will further improve the result as more samples are generated with the help of embedding level combination.

Number of Combined Samples Given $m^2 = 4$ divided patches and $n = 2$ patches to be combined, we have $C^n_{m^2} = 6$ target-sample pairs, but is it necessary to use them all? From these 6 target-sample pairs, when we use 2, 4, and 6 target-sample pairs per image and ensure all patches are selected for combination for equal times, the accuracy are 72.6, 73.3, and 73.5, respectively. These results show that more samples from combination helps to learn better representations.

Number of Divided and Combined Patches Figure 3(b) shows the influence of choosing different numbers of divided patches $m \times m$ and numbers of combined patches n. The performance is controlled by two factors: 1) the divide base number m, which determines the patch size, and 2) the percentage of the covered area by selected patches combined, i.e., n/m^2. With a proper selection of n/m^2 by controlling n, we can benefit from extra samples and difficulty it self. Meanwhile, making the task too hard with n/m^2 close to 0 (e.g. $n = 1$ for $m^2 = 2 \times 2$), or making the actual patches too small, e.g. 3×3 are both harmful to the performance. We find choosing 2x2 split with $n = 2$ have a good trade-off for these factors, which is used for our key results. When n is close to the optimal choice, i.e. $n = 2$ for $m^2 = 2 \times 2$ or $n = 3$ for $m^2 = 3 \times 3$, the small variation of n (e.g. $n = 4$ for $m^2 = 3 \times 3$) does not lead to large variation of ImageNet top-1 accuracy, showing Fast-MoCo is relatively stable to the variation of n and m.

5.5 Combination Method

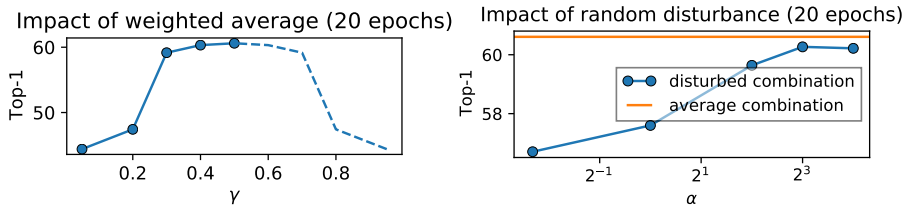
In this section we discuss different combination choices in the Combine step. We consider two alternatives: weighted average and merge by max operation.

Weighted Average. Consider the case of combining 2 patches p and p' from the 2×2 divided patches, for patch embeddings v_p and $v_{p'}$ of patches p and p' respectively, we have:

$$c = \gamma v_p + (1 - \gamma) v_{p'}, \tag{7}$$

where $p' \neq p$ and every patch is selected for equal times. By adjusting γ within the range of $[0.5, 1)$, we create a continuous transition between using patch embeddings separately and combinatorial patches with four combinations. The results are shown in Figure 4(a), from which we can see the best setting is to have $\gamma = 0.5$, which assigns equal weights for both patches. Therefore, equal weight for Fast-MoCo is the default setting in other experimental results. The transition is idiosyncratic when the weight for either feature is close to zero.

Weighted Average with Weight from Random Sampling. Apart from weighted combining with fixed weights, we also investigated the case when γ is randomly sampled from beta distribution; we have $\gamma \sim Beta(\alpha, \alpha)$ with $\alpha \in \{0.2, 1, 4, 8, 16\}$. As shown in Figure 4(b), The result gradually approaches average combination as randomness is suppressed by higher α. We conclude



(a) (b)

Fig. 4: (a): Random weighted average - fix value. (b): Random weighted average - Beta distribution.

that the combination of patch embedding is best done with its patch members contributing equally to the combined embedding.

Max Operation. As for combination with max operation, for each feature channel i, we have:

$$c^{(i)} = \max_{v \in \{v_p, v_{p'}\}} v^{(i)}. \tag{8}$$

The 100-epoch linear evaluation result when the max operation is used at the Combine step is 64.6, which is significantly lower than the result of 73.5 for the Fast-MoCo counterpart with weighted average.

6 Conclusion

In this work, a simple yet effective self-supervised learning method, i.e., Fast-MoCo, is proposed to boost the training speed of the momentum-based contrastive learning method. By extending the MoCo v3 baseline with our proposed divide and combine steps, Fast-MoCo can construct multiple positive pairs with moderately more challenging optimization objectives for each input, which could significantly increase the sample utilization efficiency with negligible computational cost. Linear evaluation results on ImageNet show that Fast-MoCo trained with 100 epochs can achieve on-par performance with MoCo v3 trained with 800 epochs, which significantly lowers the computation requirements for self-supervised learning research and breaks the barrier for the general academic community. More extensive experiments and analyses further demonstrate the transferability of Fast-MoCo to other tasks and validate our design.

Acknowledgement. This work was supported by the Australian Research Council Grant DP200103223, Australian Medical Research Future Fund MRFAI000085, CRC-P Smart Material Recovery Facility (SMRF) – Curby Soft Plastics, and CRC-P ARIA - Bionic Visual-Spatial Prosthesis for the Blind.

References

1. Bao, H., Dong, L., Wei, F.: Beit: Bert pre-training of image transformers. arXiv preprint arXiv:2106.08254 (2021)
2. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. Advances in neural information processing systems **33**, 1877–1901 (2020)
3. Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., Joulin, A.: Unsupervised learning of visual features by contrasting cluster assignments. Advances in Neural Information Processing Systems **33**, 9912–9924 (2020)
4. Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., Joulin, A.: Emerging properties in self-supervised vision transformers. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 9650–9660 (2021)
5. Chen, P., Liu, S., Jia, J.: Jigsaw clustering for unsupervised visual representation learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11526–11535 (2021)
6. Chen, T., Kornblith, S., Norouzi, M., Hinton, G.: A simple framework for contrastive learning of visual representations. In: International conference on machine learning. pp. 1597–1607. PMLR (2020)
7. Chen, X., Fan, H., Girshick, R., He, K.: Improved baselines with momentum contrastive learning. arXiv preprint arXiv:2003.04297 (2020)
8. Chen, X., He, K.: Exploring simple siamese representation learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 15750–15758 (2021)
9. Chen, X., Xie, S., He, K.: An empirical study of training self-supervised vision transformers. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 9640–9649 (2021)
10. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
11. Dwibedi, D., Aytar, Y., Tompson, J., Sermanet, P., Zisserman, A.: With a little help from my friends: Nearest-neighbor contrastive learning of visual representations. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 9588–9597 (2021)
12. Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes (voc) challenge. International journal of computer vision **88**(2), 303–338 (2010)
13. Gidaris, S., Bursuc, A., Puy, G., Komodakis, N., Cord, M., Perez, P.: Obow: Online bag-of-visual-words generation for self-supervised learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 6830–6840 (2021)
14. Girshick, R., Radosavovic, I., Gkioxari, G., Dollár, P., He, K.: Detectron (2018)
15. Grill, J.B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., Doersch, C., Avila Pires, B., Guo, Z., Gheshlaghi Azar, M., et al.: Bootstrap your own latent-a new approach to self-supervised learning. Advances in Neural Information Processing Systems **33**, 21271–21284 (2020)
16. Hadsell, R., Chopra, S., LeCun, Y.: Dimensionality reduction by learning an invariant mapping. In: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06). vol. 2, pp. 1735–1742. IEEE (2006)

17. He, K., Chen, X., Xie, S., Li, Y., Dollár, P., Girshick, R.: Masked autoencoders are scalable vision learners. arXiv preprint arXiv:2111.06377 (2021)
18. He, K., Fan, H., Wu, Y., Xie, S., Girshick, R.: Momentum contrast for unsupervised visual representation learning. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 9729–9738 (2020)
19. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: Proceedings of the IEEE international conference on computer vision. pp. 2961–2969 (2017)
20. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
21. Henaff, O.: Data-efficient image recognition with contrastive predictive coding. In: International Conference on Machine Learning. pp. 4182–4192. PMLR (2020)
22. Koohpayegani, S.A., Tejankar, A., Pirsiavash, H.: Mean shift for self-supervised learning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 10326–10335 (2021)
23. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: European conference on computer vision. pp. 740–755. Springer (2014)
24. Misra, I., Maaten, L.v.d.: Self-supervised learning of pretext-invariant representations. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 6707–6717 (2020)
25. Noroozi, M., Favaro, P.: Unsupervised learning of visual representations by solving jigsaw puzzles. In: European conference on computer vision. pp. 69–84. Springer (2016)
26. Van den Oord, A., Li, Y., Vinyals, O.: Representation learning with contrastive predictive coding. arXiv e-prints pp. arXiv–1807 (2018)
27. Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., Efros, A.A.: Context encoders: Feature learning by inpainting. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2536–2544 (2016)
28. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems **28** (2015)
29. Wang, Y., Tang, S., Zhu, F., Bai, L., Zhao, R., Qi, D., Ouyang, W.: Revisiting the transferability of supervised pretraining: an mlp perspective. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 9183–9193 (2022)
30. Wu, Z., Xiong, Y., Yu, S.X., Lin, D.: Unsupervised feature learning via non-parametric instance discrimination. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3733–3742 (2018)
31. You, Y., Gitman, I., Ginsburg, B.: Large batch training of convolutional networks. arXiv preprint arXiv:1708.03888 (2017)
32. Zbontar, J., Jing, L., Misra, I., LeCun, Y., Deny, S.: Barlow twins: Self-supervised learning via redundancy reduction. In: International Conference on Machine Learning. pp. 12310–12320. PMLR (2021)