importance of problem?

6:30 p.m

18135

• build dynamic dictionary with queue and moving-averaged encoder - to enable large & consistent dictionary on the fly that facilitates contrastive unsupervised learning.

Momentum Contrast for Unsupervised Visual Representation Learning

Kaiming He Haoqi Fan Yuxin Wu Saining Xie Ross Girshick

Facebook AI Research (FAIR)

Code: https://github.com/facebookresearch/moco

2020

Abstract

MoCo

We present Momentum Contrast (MoCo) for unsupervised visual representation learning. From a perspective on contrastive learning [29] as dictionary look-up, we build a dynamic dictionary with a queue and a moving-averaged encoder. This enables building a large and consistent dictionary on-the-fly that facilitates contrastive unsupervised learning. MoCo provides competitive results under the common linear protocol on ImageNet classification. More importantly, the representations learned by MoCo transfer well to downstream tasks. MoCo can ***outperform*** *its supervised pre-training counterpart in 7 detection/segmentation tasks on PASCAL VOC, COCO, and other datasets, sometimes surpassing it by large margins. This suggests that the gap between unsupervised and supervised representation learning has been largely closed in many vision tasks.*

EIP first impression figure

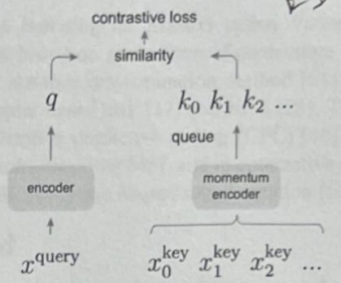


Figure 1. Momentum Contrast (MoCo) trains a visual representation encoder by matching an encoded query q to a dictionary of encoded keys using a contrastive loss. The dictionary keys $\{k_0, k_1, k_2, ...\}$ are defined on-the-fly by a set of data samples. The dictionary is built as a queue, with the current mini-batch enqueued and the oldest mini-batch dequeued, decoupling it from the mini-batch size. The keys are encoded by a slowly progressing encoder, driven by a momentum update with the query encoder. This method enables a large and consistent dictionary for learning visual representations.

1. Introduction

Unsupervised representation learning is highly successful in natural language processing, *e.g.*, as shown by GPT [50, 51] and BERT [12]. But supervised pre-training is still dominant in computer vision, where unsupervised methods generally lag behind. The reason may stem from differences in their respective signal spaces. Language tasks have discrete signal spaces (words, sub-word units, *etc.*) for building tokenized *dictionaries*, on which unsupervised learning can be based. Computer vision, in contrast, further concerns dictionary building [54, 9, 5], as the raw signal is in a continuous, high-dimensional space and is not structured for human communication (*e.g.*, unlike words).

Several recent studies [61, 46, 36, 66, 35, 56, 2] present promising results on unsupervised visual representation learning using approaches related to the *contrastive loss* [29]. Though driven by various motivations, these methods can be thought of as building *dynamic dictionaries*. The "keys" (tokens) in the dictionary are sampled from data (*e.g.*, images or patches) and are represented by an encoder network. Unsupervised learning trains encoders to perform dictionary look-up: an encoded "query" should be similar to its matching key and dissimilar to others. Learning is formulated as minimizing a contrastive loss [29].

From this perspective, we hypothesize that it is desirable to build dictionaries that are: (i) large *and* (ii) consistent as they evolve during training. Intuitively, a larger dictionary may better sample the underlying continuous, high-dimensional visual space, while the keys in the dictionary should be represented by the same or similar encoder so that their comparisons to the query are consistent. However, existing methods that use contrastive losses can be limited in one of these two aspects (discussed later in context).

We present *Momentum Contrast* (MoCo) as a way of building large and consistent dictionaries for unsupervised learning with a contrastive loss (Figure 1). We maintain the dictionary as a *queue* of data samples: the encoded representations of the current mini-batch are enqueued, and the oldest are dequeued. The queue decouples the dictionary size from the mini-batch size, allowing it to be large. Moreover, as the dictionary keys come from the preceding several mini-batches, a *slowly progressing* key encoder, implemented as a momentum-based moving average of the query encoder, is proposed to maintain consistency.

• learning - minimize contrastive loss

• Language tasks have discrete signal spaces (words, sub-word, units).

☆ MoCo build large and consistent dictionaries for unsupervised learning with contrastive loss.

MoCo is a mechanism for building dynamic dictionaries for contrastive learning, and can be used with various pretext tasks. In this paper, we follow a simple instance discrimination task [61, 63, 2]: a query matches a key if they are encoded views (*e.g.*, different crops) of the same image. Using this pretext task, MoCo shows competitive results under the common protocol of linear classification in the ImageNet dataset [11].

A main purpose of unsupervised learning is to pre-train representations (*i.e.*, features) that can be transferred to downstream tasks by fine-tuning. We show that in 7 downstream tasks related to detection or segmentation, MoCo unsupervised pre-training can *surpass* its ImageNet supervised counterpart, in some cases by nontrivial margins. In these experiments, we explore MoCo pre-trained on ImageNet or on a *one-billion* Instagram image set, demonstrating that MoCo can work well in a more real-world, billion-image scale, and relatively uncurated scenario. These results show that MoCo largely closes the gap between unsupervised and supervised representation learning in many computer vision tasks, and can serve as an alternative to ImageNet supervised pre-training in several applications.

2. Related Work

Unsupervised/self-supervised[1] learning methods generally involve two aspects: pretext tasks and loss functions. The term "pretext" implies that the task being solved is not of genuine interest, but is solved only for the true purpose of learning a good data representation. Loss functions can often be investigated independently of pretext tasks. MoCo focuses on the loss function aspect. Next we discuss related studies with respect to these two aspects.

Loss functions. A common way of defining a loss function is to measure the difference between a model's prediction and a *fixed* target, such as reconstructing the input pixels (*e.g.*, auto-encoders) by L1 or L2 losses, or classifying the input into pre-defined categories (*e.g.*, eight positions [13], color bins [64]) by cross-entropy or margin-based losses. Other alternatives, as described next, are also possible.

Contrastive losses [29] measure the similarities of sample pairs in a representation space. Instead of matching an input to a fixed target, in contrastive loss formulations the target can *vary* on-the-fly during training and can be defined in terms of the data representation computed by a network [29]. Contrastive learning is at the core of several recent works on unsupervised learning [61, 46, 36, 66, 35, 56, 2], which we elaborate on later in context (Sec. 3.1).

Adversarial losses [24] measure the difference between probability distributions. It is a widely successful technique for unsupervised data generation. Adversarial methods for representation learning are explored in [15, 16]. There are relations (see [24]) between generative adversarial networks and noise-contrastive estimation (NCE) [28].

Pretext tasks. A wide range of pretext tasks have been proposed. Examples include recovering the input under some corruption, *e.g.*, denoising auto-encoders [58], context auto-encoders [48], or cross-channel auto-encoders (colorization) [64, 65]. Some pretext tasks form pseudo-labels by, *e.g.*, transformations of a single ("exemplar") image [17], patch orderings [13, 45], tracking [59] or segmenting objects [47] in videos, or clustering features [3, 4].

Contrastive learning *vs.* pretext tasks. Various pretext tasks can be based on some form of contrastive loss functions. The instance discrimination method [61] is related to the exemplar-based task [17] and NCE [28]. The pretext task in contrastive predictive coding (CPC) [46] is a form of context auto-encoding [48], and in contrastive multiview coding (CMC) [56] it is related to colorization [64].

3. Method

3.1. Contrastive Learning as Dictionary Look-up

Contrastive learning [29], and its recent developments, can be thought of as training an encoder for a *dictionary look-up* task, as described next.

Consider an encoded query q and a set of encoded samples $\{k_0, k_1, k_2, ...\}$ that are the keys of a dictionary. Assume that there is a single key (denoted as k_+) in the dictionary that q matches. A contrastive loss [29] is a function whose value is low when q is similar to its positive key k_+ and dissimilar to all other keys (considered negative keys for q). With similarity measured by dot product, a form of a contrastive loss function, called InfoNCE [46], is considered in this paper:

$$\mathcal{L}_q = -\log \frac{\exp(q \cdot k_+/\tau)}{\sum_{i=0}^{K} \exp(q \cdot k_i/\tau)} \quad (1)$$

where τ is a temperature hyper-parameter per [61]. The sum is over one positive and K negative samples. Intuitively, this loss is the log loss of a $(K+1)$-way softmax-based classifier that tries to classify q as k_+. Contrastive loss functions can also be based on other forms [29, 59, 61, 36], such as margin-based losses and variants of NCE losses.

The contrastive loss serves as an unsupervised objective function for training the encoder networks that represent the queries and keys [29]. In general, the query representation is $q = f_q(x^q)$ where f_q is an encoder network and x^q is a query sample (likewise, $k = f_k(x^k)$). Their instantiations depend on the specific pretext task. The input x^q and x^k can be images [29, 61, 63], patches [46], or context consisting a set of patches [46]. The networks f_q and f_k can be identical [29, 59, 63], partially shared [46, 36, 2], or different [56].

[1] Self-supervised learning is a form of unsupervised learning. Their distinction is informal in the existing literature. In this paper, we use the more classical term of "unsupervised learning", in the sense of "not supervised by *human*-annotated labels".

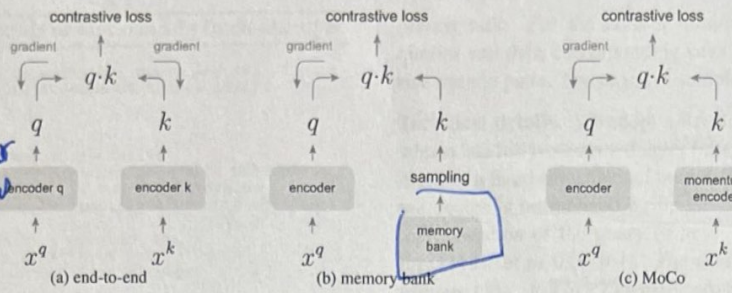


Figure 2. **Conceptual comparison of three contrastive loss mechanisms** (empirical comparisons are in Figure 3 and Table 3). Here we illustrate one pair of query and key. The three mechanisms differ in how the keys are maintained and how the key encoder is updated. **(a)**: The encoders for computing the query and key representations are updated ***end-to-end*** by back-propagation (the two encoders can be different). **(b)**: The key representations are sampled from a ***memory bank*** [61]. **(c)**: ***MoCo*** encodes the new keys on-the-fly by a momentum-updated encoder, and maintains a queue (not illustrated in this figure) of keys.

3.2. Momentum Contrast

From the above perspective, contrastive learning is a way of building a discrete dictionary on high-dimensional continuous inputs such as images. The dictionary is *dynamic* in the sense that the keys are randomly sampled, and that the key encoder evolves during training. Our hypothesis is that good features can be learned by a *large* dictionary that covers a rich set of negative samples, while the encoder for the dictionary keys is kept as *consistent* as possible despite its evolution. Based on this motivation, we present Momentum Contrast as described next.

Dictionary as a queue. At the core of our approach is maintaining the dictionary as a *queue* of data samples. This allows us to reuse the encoded keys from the immediate preceding mini-batches. The introduction of a queue decouples the dictionary size from the mini-batch size. Our dictionary size can be much larger than a typical mini-batch size, and can be flexibly and independently set as a hyper-parameter.

The samples in the dictionary are progressively replaced. The current mini-batch is enqueued to the dictionary, and the oldest mini-batch in the queue is removed. The dictionary always represents a sampled subset of all data, while the extra computation of maintaining this dictionary is manageable. Moreover, removing the oldest mini-batch can be beneficial, because its encoded keys are the most outdated and thus the least consistent with the newest ones.

Momentum update. Using a queue can make the dictionary large, but it also makes it intractable to update the key encoder by back-propagation (the gradient should propagate to all samples in the queue). A naïve solution is to copy the key encoder f_k from the query encoder f_q, ignoring this gradient. But this solution yields poor results in experiments (Sec. 4.1). We hypothesize that such failure is caused by the rapidly changing encoder that reduces the key representations' consistency. We propose a momentum update to address this issue.

Formally, denoting the parameters of f_k as θ_k and those of f_q as θ_q, we update θ_k by:

$$\theta_k \leftarrow m\theta_k + (1-m)\theta_q. \tag{2}$$

Here $m \in [0, 1)$ is a momentum coefficient. Only the parameters θ_q are updated by back-propagation. The momentum update in Eqn.(2) makes θ_k evolve more smoothly than θ_q. As a result, though the keys in the queue are encoded by different encoders (in different mini-batches), the difference among these encoders can be made small. In experiments, a relatively large momentum (*e.g.*, $m = 0.999$, our default) works much better than a smaller value (*e.g.*, $m = 0.9$), suggesting that a slowly evolving key encoder is a core to making use of a queue.

Relations to previous mechanisms. MoCo is a general mechanism for using contrastive losses. We compare it with two existing general mechanisms in Figure 2. They exhibit different properties on the dictionary size and consistency.

The ***end-to-end*** update by back-propagation is a natural mechanism (*e.g.*, [29, 46, 36, 63, 2, 35], Figure 2a). It uses samples in the current mini-batch as the dictionary, so the keys are consistently encoded (by the same set of encoder parameters). But the dictionary size is coupled with the mini-batch size, limited by the GPU memory size. It is also challenged by large mini-batch optimization [25]. Some recent methods [46, 36, 2] are based on pretext tasks driven by local positions, where the dictionary size can be made larger by multiple positions. But these pretext tasks may require special network designs such as patchifying the input [46] or customizing the receptive field size [2], which may complicate the transfer of these networks to downstream tasks.

Another mechanism is the ***memory bank*** approach proposed by [61] (Figure 2b). A memory bank consists of the representations of all samples in the dataset. The dictionary for each mini-batch is randomly sampled from the memory bank with no back-propagation, so it can support a large dictionary size. However, the representation of a sample in

Algorithm 1 Pseudocode of MoCo in a PyTorch-like style.

```
# f_q, f_k: encoder networks for query and key
# queue: dictionary as a queue of K keys (CxK)
# m: momentum
# t: temperature

f_k.params = f_q.params # initialize
for x in loader: # load a minibatch x with N samples
   x_q = aug(x) # a randomly augmented version
   x_k = aug(x) # another randomly augmented version

   q = f_q.forward(x_q) # queries: NxC
   k = f_k.forward(x_k) # keys: NxC
   k = k.detach() # no gradient to keys

   # positive logits: Nx1
   l_pos = bmm(q.view(N,1,C), k.view(N,C,1))

   # negative logits: NxK
   l_neg = mm(q.view(N,C), queue.view(C,K))

   # logits: Nx(1+K)
   logits = cat([l_pos, l_neg], dim=1)

   # contrastive loss, Eqn.(1)
   labels = zeros(N) # positives are the 0-th
   loss = CrossEntropyLoss(logits/t, labels)

   # SGD update: query network
   loss.backward()
   update(f_q.params)

   # momentum update: key network
   f_k.params = m*f_k.params+(1-m)*f_q.params

   # update dictionary
   enqueue(queue, k) # enqueue the current minibatch
   dequeue(queue) # dequeue the earliest minibatch
```

bmm: batch matrix multiplication; mm: matrix multiplication; cat: concatenation.

the memory bank was updated when it was last seen, so the sampled keys are essentially about the encoders at multiple different steps all over the past epoch and thus are less consistent. A momentum update is adopted on the memory bank in [61]. Its momentum update is on the representations of the same sample, *not* the encoder. This momentum update is irrelevant to our method, because MoCo does not keep track of every sample. Moreover, our method is more memory-efficient and can be trained on billion-scale data, which can be intractable for a memory bank.

Sec. 4 empirically compares these three mechanisms.

3.3. Pretext Task

Contrastive learning can drive a variety of pretext tasks. As the focus of this paper is not on designing a new pretext task, we use a simple one mainly following the *instance discrimination* task in [61], to which some recent works [63, 2] are related.

Following [61], we consider a query and a key as a positive pair if they originate from the same image, and otherwise as a negative sample pair. Following [63, 2], we take two random "views" of the same image under random data augmentation to form a positive pair. The queries and keys are respectively encoded by their encoders, f_q and f_k. The encoder can be any convolutional neural network [39].

Algorithm 1 provides the pseudo-code of MoCo for this pretext task. For the current mini-batch, we encode the queries and their corresponding keys, which form the positive sample pairs. The negative samples are from the queue.

Technical details. We adopt a ResNet [33] as the encoder, whose last fully-connected layer (after global average pooling) has a fixed-dimensional output (128-D [61]). This output vector is normalized by its L2-norm [61]. This is the representation of the query or key. The temperature τ in Eqn.(1) is set as 0.07 [61]. The data augmentation setting follows [61]: a 224×224-pixel crop is taken from a randomly resized image, and then undergoes random color jittering, random horizontal flip, and random grayscale conversion, all available in PyTorch's torchvision package.

Shuffling BN. Our encoders f_q and f_k both have Batch Normalization (BN) [37] as in the standard ResNet [33]. In experiments, we found that using BN prevents the model from learning good representations, as similarly reported in [35] (which avoids using BN). The model appears to "cheat" the pretext task and easily finds a low-loss solution. This is possibly because the intra-batch communication among samples (caused by BN) leaks information.

We resolve this problem by shuffling BN. We train with multiple GPUs and perform BN on the samples independently for each GPU (as done in common practice). For the key encoder f_k, we shuffle the sample order in the current mini-batch before distributing it among GPUs (and shuffle back after encoding); the sample order of the mini-batch for the query encoder f_q is not altered. This ensures the batch statistics used to compute a query and its positive key come from two different subsets. This effectively tackles the cheating issue and allows training to benefit from BN.

We use shuffled BN in both our method and its end-to-end ablation counterpart (Figure 2a). It is irrelevant to the memory bank counterpart (Figure 2b), which does not suffer from this issue because the positive keys are from different mini-batches in the past.

4. Experiments

We study unsupervised training performed in:

***ImageNet-1M* (IN-1M)**: This is the ImageNet [11] training set that has ~1.28 million images in 1000 classes (often called ImageNet-1K; we count the image number instead, as classes are not exploited by unsupervised learning). This dataset is well-balanced in its class distribution, and its images generally contain iconic view of objects.

***Instagram-1B* (IG-1B)**: Following [44], this is a dataset of *~1 billion* (940M) public images from *Instagram*. The images are from ~1500 hashtags [44] that are related to the ImageNet categories. This dataset is relatively *uncurated* comparing to IN-1M, and has a *long-tailed*, *unbalanced* distribution of real-world data. This dataset contains both iconic objects and scene-level images.

Training. We use SGD as our optimizer. The SGD weight decay is 0.0001 and the SGD momentum is 0.9. For IN-1M, we use a mini-batch size of 256 (N in Algorithm 1) in 8 GPUs, and an initial learning rate of 0.03. We train for 200 epochs with the learning rate multiplied by 0.1 at 120 and 160 epochs [61], taking ~53 hours training ResNet-50. For IG-1B, we use a mini-batch size of 1024 in 64 GPUs, and a learning rate of 0.12 which is exponentially decayed by 0.9× after every 62.5k iterations (64M images). We train for 1.25M iterations (~1.4 epochs of IG-1B), taking ~6 days for ResNet-50.

4.1. Linear Classification Protocol

We first verify our method by *linear* classification on *frozen* features, following a common protocol. In this subsection we perform unsupervised pre-training on IN-1M. Then we freeze the features and train a supervised linear classifier (a fully-connected layer followed by softmax). We train this classifier on the global average pooling features of a ResNet, for 100 epochs. We report 1-crop, top-1 classification accuracy on the ImageNet validation set.

For this classifier, we perform a grid search and find the optimal initial learning rate is 30 and weight decay is 0 (similarly reported in [56]). These hyper-parameters perform consistently well for all ablation entries presented in this subsection. These hyper-parameter values imply that the feature distributions (*e.g.*, magnitudes) can be substantially different from those of ImageNet supervised training, an issue we will revisit in Sec. 4.2.

Ablation: contrastive loss mechanisms. We compare the three mechanisms that are illustrated in Figure 2. To focus on the effect of contrastive loss mechanisms, we implement all of them in the same pretext task as described in Sec. 3.3. We also use the same form of InfoNCE as the contrastive loss function, Eqn.(1). As such, the comparison is solely on the three mechanisms.

The results are in Figure 3. Overall, all three mechanisms benefit from a larger K. A similar trend has been observed in [61, 56] under the memory bank mechanism, while here we show that this trend is more general and can be seen in all mechanisms. These results support our motivation of building a large dictionary.

The ***end-to-end*** mechanism performs similarly to MoCo when K is small. However, the dictionary size is limited by the mini-batch size due to the end-to-end requirement. Here the largest mini-batch a high-end machine (8 Volta 32GB GPUs) can afford is 1024. More essentially, large mini-batch training is an open problem [25]: we found it necessary to use the linear learning rate scaling rule [25] here, without which the accuracy drops (by ~2% with a 1024 mini-batch). But optimizing with a larger mini-batch is harder [25], and it is questionable whether the trend can be extrapolated into a larger K even if memory is sufficient.

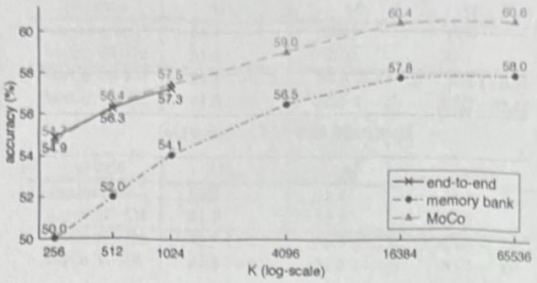


Figure 3. **Comparison of three contrastive loss mechanisms** under the ImageNet linear classification protocol. We adopt the same pretext task (Sec. 3.3) and only vary the contrastive loss mechanism (Figure 2). The number of negatives is K in memory bank and MoCo, and is $K-1$ in end-to-end (offset by one because the positive key is in the same mini-batch). The network is ResNet-50.

The ***memory bank*** [61] mechanism can support a larger dictionary size. But it is 2.6% worse than MoCo. This is inline with our hypothesis: the keys in the memory bank are from very different encoders all over the past epoch and they are not consistent. Note the memory bank result of 58.0% reflects our improved implementation of [61].[2]

Ablation: momentum. The table below shows ResNet-50 accuracy with different MoCo momentum values (m in Eqn.(2)) used in pre-training ($K = 4096$ here) :

momentum m	0	0.9	0.99	0.999	0.9999
accuracy (%)	*fail*	55.2	57.8	59.0	58.9

It performs reasonably well when m is in $0.99 \sim 0.9999$, showing that a slowly progressing (*i.e.*, relatively large momentum) key encoder is beneficial. When m is too small (*e.g.*, 0.9), the accuracy drops considerably; at the extreme of *no momentum* (m is 0), the training loss oscillates and fails to converge. These results support our motivation of building a consistent dictionary.

Comparison with previous results. Previous unsupervised learning methods can differ substantially in model sizes. For a fair and comprehensive comparison, we report ***accuracy vs. #parameters***[3] trade-offs. Besides ResNet-50 (R50) [33], we also report its variants that are 2× and 4× wider (more channels), following [38].[4] We set $K = 65536$ and $m = 0.999$. Table 1 is the comparison.

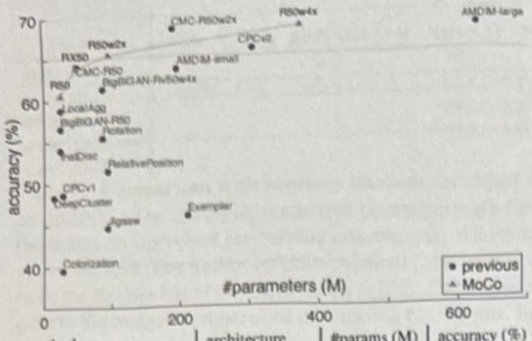
MoCo with R50 performs competitively and achieves 60.6% accuracy, better than all competitors of similar model sizes (~24M). MoCo benefits from larger models and achieves 68.6% accuracy with R50w4×.

Notably, we achieve competitive results using a *standard* ResNet-50 and require no specific architecture designs, *e.g.*,

[2]Here 58.0% is with InfoNCE and K=65536. We reproduce 54.3% when using NCE and K=4096 (the same as [61]), close to 54.0% in [61].

[3]Parameters are of the *feature extractor*: *e.g.*, we do not count the parameters of $conv_k$ if $conv_k$ is not included in linear classification.

[4]Our w2× and w4× models correspond to the "×8" and "×16" cases in [38], because the standard-sized ResNet is referred to as "×4" in [38].



method	architecture	#params (M)	accuracy (%)
Exemplar [17]	R50w3×	211	46.0 [38]
RelativePosition [13]	R50w2×	94	51.4 [38]
Jigsaw [45]	R50w2×	94	44.6 [38]
Rotation [19]	Rv50w4×	86	55.4 [38]
Colorization [64]	R101*	28	39.6 [14]
DeepCluster [3]	VGG [53]	15	48.4 [4]
BigBiGAN [16]	R50	24	56.6
	Rv50w4×	86	61.3
methods based on contrastive learning follow:			
InstDisc [61]	R50	24	54.0
LocalAgg [66]	R50	24	58.8
CPC v1 [46]	R101*	28	48.7
CPC v2 [35]	$R170^*_{wider}$	303	65.9
CMC [56]	$R50_{L+ab}$	47	64.1†
	$R50w2\times_{L+ab}$	188	68.4†
AMDIM [2]	$AMDIM_{small}$	194	63.5†
	$AMDIM_{large}$	626	68.1†
MoCo	R50	24	60.6
	RX50	46	63.9
	R50w2×	94	65.4
	R50w4×	375	**68.6**

Table 1. **Comparison under the linear classification protocol on ImageNet**. The figure visualizes the table. All are reported as unsupervised pre-training on the ImageNet-1M training set, followed by supervised linear classification trained on frozen features, evaluated on the validation set. The parameter counts are those of the feature extractors. We compare with improved re-implementations if available (referenced after the numbers).

Notations: R101*/R170* is ResNet-101/170 with the last residual stage removed [14, 46, 35], and R170 is made wider [35]; Rv50 is a reversible net [23], RX50 is ResNeXt-50-32×8d [62].

†: *Pre-training uses FastAutoAugment [40] that is supervised by ImageNet labels.*

patchified inputs [46, 35], carefully tailored receptive fields [2], or combining two networks [56]. By using an architecture that is not customized for the pretext task, it is easier to transfer features to a variety of visual tasks and make comparisons, studied in the next subsection.

This paper's focus is on a mechanism for general contrastive learning; we do not explore orthogonal factors (such as specific pretext tasks) that may further improve accuracy. As an example, "MoCo v2" [8], an extension of a preliminary version of this manuscript, achieves 71.1% accuracy with R50 (up from 60.6%), given small changes on the data augmentation and output projection head [7]. We believe that this additional result shows the generality and robustness of the MoCo framework.

pre-train	AP_{50}	AP	AP_{75}
random init.	64.4	37.9	38.6
super. IN-1M	81.4	54.0	59.1
MoCo IN-1M	81.1 (−0.3)	54.6 (+0.6)	59.9 (+0.8)
MoCo IG-1B	81.6 (+0.2)	55.5 (+1.5)	61.2 (+2.1)

(a) Faster R-CNN, R50-**dilated-C5**

pre-train	AP_{50}	AP	AP_{75}
random init.	60.2	33.8	33.1
super. IN-1M	81.3	53.5	58.8
MoCo IN-1M	81.5 (+0.2)	55.9 (+2.4)	62.6 (+3.8)
MoCo IG-1B	82.2 (+0.9)	57.2 (+3.7)	63.7 (+4.9)

(b) Faster R-CNN, R50-**C4**

Table 2. **Object detection fine-tuned on PASCAL VOC** `trainval07+12`. Evaluation is on `test2007`: AP_{50} (default VOC metric), AP (COCO-style), and AP_{75}, averaged over 5 trials. All are fine-tuned for 24k iterations (~23 epochs). In the brackets are the gaps to the ImageNet supervised pre-training counterpart. In green are the gaps of at least +0.5 point.

	R50-dilated-C5			R50-C4		
pre-train	AP_{50}	AP	AP_{75}	AP_{50}	AP	AP_{75}
end-to-end	79.2	52.0	56.6	80.4	54.6	60.3
memory bank	79.8	52.9	57.9	80.6	54.9	60.6
MoCo	**81.1**	**54.6**	**59.9**	**81.5**	**55.9**	**62.6**

Table 3. **Comparison of three contrastive loss mechanisms** on PASCAL VOC object detection, fine-tuned on `trainval07+12` and evaluated on `test2007` (averages over 5 trials). All models are implemented by us (Figure 3), pre-trained on IN-1M, and fine-tuned using the same settings as in Table 2.

4.2. Transferring Features

A main goal of unsupervised learning is to *learn features that are transferrable*. ImageNet supervised pre-training is most influential when serving as the initialization for fine-tuning in downstream tasks (*e.g.*, [21, 20, 43, 52]). Next we compare MoCo with ImageNet supervised pre-training, transferred to various tasks including PASCAL VOC [18], COCO [42], *etc*. As prerequisites, we discuss two important issues involved [31]: normalization and schedules.

Normalization. As noted in Sec. 4.1, features produced by unsupervised pre-training can have different distributions compared with ImageNet supervised pre-training. But a system for a downstream task often has hyper-parameters (*e.g.*, learning rates) selected for supervised pre-training. To relieve this problem, we adopt *feature normalization* during fine-tuning: we fine-tune with BN that is trained (and synchronized across GPUs [49]), instead of freezing it by an affine layer [33]. We also use BN in the newly initialized layers (*e.g.*, FPN [41]), which helps calibrate magnitudes.

We perform normalization when fine-tuning supervised and unsupervised pre-training models. MoCo *uses the same hyper-parameters* as the ImageNet supervised counterpart.

Schedules. If the fine-tuning schedule is long enough, training detectors from random initialization can be strong baselines, and can match the ImageNet supervised counterpart on COCO [31]. Our goal is to investigate *transferabil-*

pre-train	AP_{50} RelPos, by [14]	Multi-task [14]	Jigsaw, by [26]	LocalAgg [66]	MoCo	AP MoCo	AP_{75} Multi-task [14]	MoCo
super. IN-1M	74.2	74.2	70.5	74.6	74.4	42.4	44.3	42.7
unsup. IN-1M	66.8 (−7.4)	70.5 (−3.7)	61.4 (−9.1)	69.1 (−5.5)	74.9 (+0.5)	46.6 (+4.2)	43.9 (−0.4)	50.1 (+7.4)
unsup. IN-14M	-	-	69.2 (−1.3)	-	75.2 (+0.8)	46.9 (+4.5)	-	50.2 (+7.5)
unsup. YFCC-100M	-	-	66.6 (−3.9)	-	74.7 (+0.3)	45.9 (+3.5)	-	49.0 (+6.3)
unsup. IG-1B	-	-	-	-	75.6 (+1.2)	47.6 (+5.2)	-	51.7 (+9.0)

Table 4. **Comparison with previous methods on object detection fine-tuned on PASCAL VOC** `trainval2007`. Evaluation is on `test2007`. The ImageNet supervised counterparts are from the respective papers, and are reported as having *the same structure* as the respective unsupervised pre-training counterparts. All entries are based on the C4 backbone. The models in [14] are R101 v2 [34], and others are R50. The RelPos (relative position) [13] result is the best single-task case in the Multi-task paper [14]. The Jigsaw [45] result is from the ResNet-based implementation in [26]. Our results are with 9k-iteration fine-tuning, averaged over 5 trials. In the brackets are the gaps to the ImageNet supervised pre-training counterpart. In green are the gaps of at least +0.5 point.

ity of features, so our experiments are on controlled schedules, *e.g.*, the 1× (~12 epochs) or 2× schedules [22] for COCO, in contrast to 6×~9× in [31]. On smaller datasets like VOC, training longer may not catch up [31].

Nonetheless, in our fine-tuning, MoCo *uses the same schedule* as the ImageNet supervised counterpart, and random initialization results are provided as references.

Put together, our fine-tuning uses the same setting as the supervised pre-training counterpart. This may place MoCo at a *disadvantage*. Even so, MoCo is competitive. Doing so also makes it feasible to present comparisons on multiple datasets/tasks, without extra hyper-parameter search.

4.2.1 PASCAL VOC Object Detection

Setup. The detector is Faster R-CNN [52] with a backbone of R50-dilated-C5 or R50-C4 [32] (details in appendix), with BN tuned, implemented in [60]. We fine-tune all layers end-to-end. The image scale is [480, 800] pixels during training and 800 at inference. The same setup is used for all entries, including the supervised pre-training baseline. We evaluate the default VOC metric of AP_{50} (*i.e.*, IoU threshold is 50%) and the more stringent metrics of COCO-style AP and AP_{75}. Evaluation is on the VOC `test2007` set.

Ablation: backbones. Table 2 shows the results fine-tuned on `trainval07+12` (~16.5k images). For R50-dilated-C5 (Table 2a), MoCo pre-trained on IN-1M is comparable to the supervised pre-training counterpart, and MoCo pre-trained on IG-1B *surpasses* it. For R50-C4 (Table 2b), MoCo with IN-1M or IG-1B is *better* than the supervised counterpart: up to **+0.9** AP_{50}, **+3.7** AP, and **+4.9** AP_{75}.

Interestingly, the transferring accuracy depends on the detector structure. For the C4 backbone, by default used in existing ResNet-based results [14, 61, 26, 66], the advantage of unsupervised pre-training is larger. The relation between pre-training *vs.* detector structures has been veiled in the past, and should be a factor under consideration.

Ablation: contrastive loss mechanisms. We point out that these results are partially because we establish solid detection baselines for contrastive learning. To pin-point the gain that is *solely* contributed by using the MoCo mechanism in contrastive learning, we fine-tune the models pre-trained with the end-to-end or memory bank mechanism, both implemented by us (*i.e.*, the best ones in Figure 3), using the same fine-tuning setting as MoCo.

These competitors perform decently (Table 3). Their AP and AP_{75} with the C4 backbone are also higher than the ImageNet supervised counterpart's, *c.f.* Table 2b, but other metrics are lower. They are worse than MoCo in all metrics. This shows the benefits of MoCo. In addition, how to train these competitors in larger-scale data is an open question, and they may not benefit from IG-1B.

Comparison with previous results. Following the competitors, we fine-tune on `trainval2007` (~5k images) using the C4 backbone. The comparison is in Table 4.

For the AP_{50} metric, *no* previous method can catch up with its respective supervised pre-training counterpart. MoCo pre-trained on any of IN-1M, IN-14M (full ImageNet), YFCC-100M [55], and IG-1B can *outperform* the supervised baseline. Large gains are seen in the more stringent metrics: up to **+5.2** AP and **+9.0** AP_{75}. These gains are larger than the gains seen in `trainval07+12` (Table 2b).

4.2.2 COCO Object Detection and Segmentation

Setup. The model is Mask R-CNN [32] with the FPN [41] or C4 backbone, with BN tuned, implemented in [60]. The image scale is in [640, 800] pixels during training and is 800 at inference. We fine-tune all layers end-to-end. We fine-tune on the `train2017` set (~118k images) and evaluate on `val2017`. The schedule is the default 1× or 2× in [22].

Results. Table 5 shows the results on COCO with the FPN (Table 5a, b) and C4 (Table 5c, d) backbones. With the 1× schedule, all models (including the ImageNet supervised counterparts) are heavily under-trained, as indicated by the ~2 points gaps to the 2× schedule cases. With the 2× schedule, MoCo is *better* than its ImageNet supervised counterpart in all metrics in both backbones.

4.2.3 More Downstream Tasks

Table 6 shows more downstream tasks (implementation details in appendix). Overall, MoCo performs competitively

pre-train	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}
random init.	31.0	49.5	33.2	28.5	46.8	30.4
super. IN-1M	38.9	59.6	42.7	35.4	56.5	38.1
MoCo IN-1M	38.5 (−0.4)	58.9 (−0.7)	42.0 (−0.7)	35.1 (−0.3)	55.9 (−0.6)	37.7 (−0.4)
MoCo IG-1B	38.9 (0.0)	59.4 (−0.2)	42.3 (−0.4)	35.4 (0.0)	56.5 (0.0)	37.9 (−0.2)

(a) Mask R-CNN, R50-**FPN**, 1× schedule

pre-train	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}
random init.	36.7	56.7	40.0	33.7	53.8	35.9
super. IN-1M	40.6	61.3	44.4	36.8	58.1	39.5
MoCo IN-1M	40.8 (+0.2)	61.6 (+0.3)	44.7 (+0.3)	36.9 (+0.1)	58.4 (+0.3)	39.7 (+0.2)
MoCo IG-1B	41.1 (+0.5)	61.8 (+0.5)	45.1 (+0.7)	37.4 (+0.6)	59.1 (+1.0)	40.2 (+0.7)

(b) Mask R-CNN, R50-**FPN**, 2× schedule

pre-train	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}
random init.	26.4	44.0	27.8	29.3	46.9	30.8
super. IN-1M	38.2	58.2	41.2	33.3	54.7	35.2
MoCo IN-1M	38.5 (+0.3)	58.3 (+0.1)	41.6 (+0.4)	33.6 (+0.3)	54.8 (+0.1)	35.6 (+0.4)
MoCo IG-1B	39.1 (+0.9)	58.7 (+0.5)	42.2 (+1.0)	34.1 (+0.8)	55.4 (+0.7)	36.4 (+1.2)

(c) Mask R-CNN, R50-**C4**, 1× schedule

pre-train	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}
random init.	35.6	54.6	38.2	31.4	51.5	33.5
super. IN-1M	40.0	59.9	43.1	34.7	56.5	36.9
MoCo IN-1M	40.7 (+0.7)	60.5 (+0.6)	44.1 (+1.0)	35.4 (+0.7)	57.3 (+0.8)	37.6 (+0.7)
MoCo IG-1B	41.1 (+1.1)	60.7 (+0.8)	44.8 (+1.7)	35.6 (+0.9)	57.4 (+0.9)	38.1 (+1.2)

(d) Mask R-CNN, R50-**C4**, 2× schedule

Table 5. **Object detection and instance segmentation fine-tuned on COCO**: bounding-box AP (AP^{bb}) and mask AP (AP^{mk}) evaluated on `val2017`. In the brackets are the gaps to the ImageNet supervised pre-training counterpart. In green are the gaps of at least +0.5 point.

	COCO keypoint detection		
pre-train	AP^{kp}	AP^{kp}_{50}	AP^{kp}_{75}
random init.	65.9	86.5	71.7
super. IN-1M	65.8	86.9	71.9
MoCo IN-1M	66.8 (+1.0)	87.4 (+0.5)	72.5 (+0.6)
MoCo IG-1B	66.9 (+1.1)	87.8 (+0.9)	73.0 (+1.1)

	COCO dense pose estimation		
pre-train	AP^{dp}	AP^{dp}_{50}	AP^{dp}_{75}
random init.	39.4	78.5	35.1
super. IN-1M	48.3	85.6	50.6
MoCo IN-1M	50.1 (+1.8)	86.8 (+1.2)	53.9 (+3.3)
MoCo IG-1B	50.6 (+2.3)	87.0 (+1.4)	54.3 (+3.7)

	LVIS v0.5 instance segmentation		
pre-train	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}
random init.	22.5	34.8	23.8
super. IN-1M†	24.4	37.8	25.8
MoCo IN-1M	24.1 (−0.3)	37.4 (−0.4)	25.5 (−0.3)
MoCo IG-1B	24.9 (+0.5)	38.2 (+0.4)	26.4 (+0.6)

	Cityscapes instance seg.		**Semantic seg.** (mIoU)	
pre-train	AP^{mk}	AP^{mk}_{50}	**Cityscapes**	**VOC**
random init.	25.4	51.1	65.3	39.5
super. IN-1M	32.9	59.6	74.6	74.4
MoCo IN-1M	32.3 (−0.6)	59.3 (−0.3)	75.3 (+0.7)	72.5 (−1.9)
MoCo IG-1B	32.9 (0.0)	60.3 (+0.7)	75.5 (+0.9)	73.6 (−0.8)

Table 6. **MoCo *vs.* ImageNet supervised pre-training, fine-tuned on various tasks**. For each task, the same architecture and schedule are used for all entries (see appendix). In the brackets are the gaps to the ImageNet supervised pre-training counterpart. In green are the gaps of at least +0.5 point.
†: this entry is with BN frozen, which improves results; see main text.

with ImageNet supervised pre-training:

COCO keypoint detection: supervised pre-training has *no* clear advantage over random initialization, whereas MoCo outperforms in all metrics.

COCO dense pose estimation [1]: MoCo substantially outperforms supervised pre-training, *e.g.*, by 3.7 points in AP^{dp}_{75}, in this highly localization-sensitive task.

LVIS v0.5 instance segmentation [27]: this task has ~1000 long-tailed distributed categories. Specifically in LVIS for the ImageNet supervised baseline, we find fine-tuning with frozen BN (24.4 AP^{mk}) is better than tunable BN (details in appendix). So we compare MoCo with the better supervised pre-training variant in this task. MoCo with IG-1B surpasses it in all metrics.

Cityscapes instance segmentation [10]: MoCo with IG-1B is on par with its supervised pre-training counterpart in AP^{mk}, and is higher in AP^{mk}_{50}.

Semantic segmentation: On Cityscapes [10], MoCo outperforms its supervised pre-training counterpart by up to 0.9 point. But on VOC semantic segmentation, MoCo is worse by at least 0.8 point, a negative case we have observed.

Summary. In sum, MoCo can ***outperform*** its ImageNet supervised pre-training counterpart in 7 detection or segmentation tasks.[5] Besides, MoCo is on par on Cityscapes instance segmentation, and lags behind on VOC semantic segmentation; we show another comparable case on iNaturalist [57] in appendix. Overall, *MoCo has largely closed the gap between unsupervised and supervised representation learning in multiple vision tasks*.

Remarkably, in all these tasks, MoCo pre-trained on IG-1B is consistently better than MoCo pre-trained on IN-1M. This shows that *MoCo can perform well on this large-scale, relatively uncurated dataset*. This represents a scenario towards *real-world* unsupervised learning.

5. Discussion and Conclusion

Our method has shown positive results of unsupervised learning in a variety of computer vision tasks and datasets. A few open questions are worth discussing. MoCo's improvement from IN-1M to IG-1B is consistently noticeable but relatively small, suggesting that the larger-scale data may not be fully exploited. We hope an advanced pretext task will improve this. Beyond the simple instance discrimination task [61], it is possible to adopt MoCo for pretext tasks like masked auto-encoding, *e.g.*, in language [12] and in vision [46]. We hope MoCo will be useful with other pretext tasks that involve contrastive learning.

[5]Namely, object detection on VOC/COCO, instance segmentation on COCO/LVIS, keypoint detection on COCO, dense pose on COCO, and semantic segmentation on Cityscapes.