

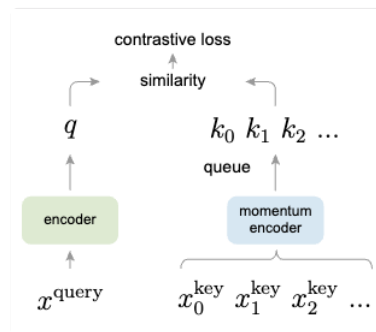
Momentum Contrast for Unsupervised Visual Representation Learning

• [[📄 1911.05722v3.pdf]]


• Basic Idea:

- How can we reduce the large batch size dependency of contrastive self-supervised learning methods like SimCLR?
- Idea: The negative samples don't have to necessarily belong to the same batch. However, problem with this approach is that the negative samples will not be *consistent* since the encoder is constantly being updated.
- *Contrastive Learning as building dynamic dictionaries:*

• 🟡 P1



- Keys are the words in the dictionary (stored as encoded feature vectors) against which any comparisons will be made when a query comes in. In the context of contrastive learning, all the positive and negative sets are considered to be keys in the dictionary.
- In the context of contrastive learning, *query* is an augmented view of an image in the mini-batch. It will be compared against all the keys in the dictionary.
- 🟡 P3 The dictionary is dynamic in the sense that the keys are randomly sampled, and that the key encoder evolves during training.
- To maintain *consistency*, the encoder for negative set can be implemented as a momentum-based moving average of the query encoder.
- To decouple the dictionary size from the mini-batch size, implement 🟡 P1 the dictionary as a queue of data samples: the encoded representations of the current mini-batch are enqueued, and the oldest are dequeued.
- The performance of contrastive learning hinges greatly on the following characteristics of the dictionary:
 - *Size of the dictionary:* Dictionary should be large so as to 🟡 P1 better sample the underlying continuous, highdimensional visual space
 - *Consistency of the dictionary:* 🟡 P1 keys in the dictionary should be represented by the same or similar encoder so that their comparisons to the query are consistent
- **Hypothesis:** 🟡 P3 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.
- **Results:**
 - MoCo (ResNet-50, 24M parameters, pre-trained on ImageNet-1k, Linear Evaluation Protocol on ImageNet-1k): 60.6% accuracy (Table 1 of the paper)
 - MoCo v2 (Similar training and linear evaluation protocol as MoCo, with small changes on the data augmentation, output projection head): 71.1% accuracy (Para below Table 1 of the paper)
 - MoCo (Fine-tuning in ImageNet-1k after pre-training on ImageNet-1k): 77% accuracy (Model trained on ImageNet from scratch using supervised learning achieves 76.5%) (Appendix A.7 of the paper).
 - 🟡 P1 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.
- **Conclusion:**

- This shows although the MoCo features in themselves may not be good enough on ImageNet (linear evaluation protocol) but are still better than supervised counterpart (shown by fine-tuning and other vision tasks).
-  **P8** 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. Where IN-1M and IG-1B represents ImageNet-1 million and Instagram 1 Billion datasets respectively.

• Momentum Contrast (MoCo)

 **P4**

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: NxK
    k = f_k.forward(x_k) # keys: NxK
    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.

• Notations:

- q : Encoded query
- $\{k_0, k_1, k_2, \dots\}$: Set of encoded samples that are the keys of the dictionary
- k_+ : Single key which matches the query q
- x^q and x^k : Query and Key sample. Can be views of same image, patches, or context consisting a set of patches
- $q = f_q(x^q)$ where f_q is the query encoder network.
- $k = f_k(x^k)$ where f_k is the key encoder network. Note that f_q and f_k can be identical, partially shared or different.
- θ_q and θ_k are parameters of f_q and f_k respectively.
- τ : Temperature hyper-paramter. ($\tau = 0.07$ used in the paper)
- $m \in [0, 1]$: Momentum coefficient ($m = 0.999$ used in the paper)
- K : Number of negative samples ($K = 65536$ used in the paper. Contrast it with the batch-size $N = 256$.)


- **Pre-text task:** MoCo can be used with various pretext tasks. However, authors use simple *instance discrimination task*,  **P2** a query matches a key if they are encoded views (e.g., different crops) of the same image.  **P4** 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. At the end of iteration, the positive keys from the current mini-batch are enqueued.

- **Loss:** InfoNCE loss


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

where each query q is compared against one positive and K negative samples. Therefore, the sum also runs over them. Note that $k_0 = k_+$.

• Momentum Update:

- Large dictionary makes it intractable to update key encoder by back-propagation. This is because in the InfoNCE loss, the gradient needs to be computed wrt all the keys in the dictionary (Similar problem faced by Word2Vec).
- Naive Solution: Copying θ_q to θ_k after every update to θ_q ignoring the gradient of θ_k .  **P3** 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
- The parameters θ_k is updated by:

$$\theta_k \leftarrow m\theta_k + (1 - m)\theta_q$$

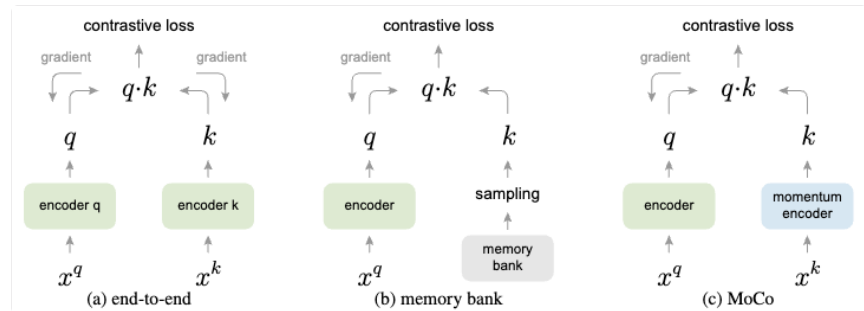
- The momentum updates makes θ_k evolve more smoothly than θ_q .  **P3** As a result, though the keys in the queue are encoded by different encoders (in different mini-batches), the

difference among these encoders can be made small.

- Only the parameters θ_q are updated by back-propagation.

Other ways to choose Negative samples

● P3



End-to-end update:

- Uses samples in the current mini-batch as the keys of the dictionary.
- Pros: Has high-consistency of the dictionary as the same encoder is used to generate keys and query.
- Cons: Requires large batch sizes which is limited by the GPU memory and also suffers from challenges of large mini-batch optimization.
- P3 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

Memory Bank:

- P3 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.
- Naive Implementation: Whenever a sample comes in the mini-batch calculate the representation and replace this new representation with the old one in the memory bank.
- Cons: ● P3 the representation of a sample in ● P4 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. To achieve consistency, a momentum update is performed on the representations of the sample (contrast it with the momentum update on the encoder in MoCo, which makes it memory-efficient).

Tricks of the trade:

- Normalize the output feature-vector using $l2$ — norm before using in the loss-function.
- Shuffling Batch Normalisation:
 - The ResNet based encoder f_q and f_k have batch normalisation layers. BN prevents the model from learning good representations as ● P4 the intra-batch communication among samples (caused by BN) leaks information and ● P4 The model appears to "cheat" the pretext task and easily finds a low-loss solution.
 - Problem Identification/Empirical Proof:
 - P10 Figure A.1 provides the training curves of MoCo with or without shuffling BN: removing shuffling BN shows obvious overfitting to the pretext task: training accuracy of the pretext task (dash curve) quickly increases to >99.9%, and the kNN-based validation classification accuracy (solid curve) drops soon. This is observed for both the MoCo and end-to-end variants; the memory bank variant implicitly has different statistics for q and k , so avoids this issue. These experiments suggest that without shuffling BN, the sub-batch statistics can serve as a "signature" to tell which sub-batch the positive key is in. Shuffling BN can remove this signature and avoid such cheating.

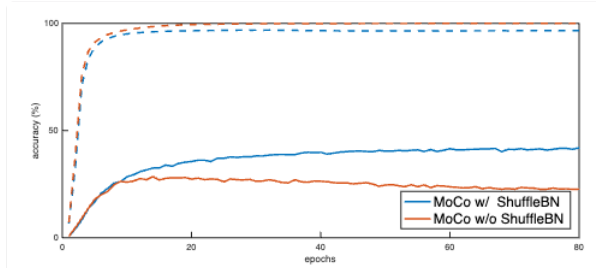


Figure A.1. **Ablation of Shuffling BN.** *Dash:* training curve of the pretext task, plotted as the accuracy of $(K+1)$ -way dictionary lookup. *Solid:* validation curve of a kNN-based monitor [61] (not a linear classifier) on ImageNet classification accuracy. This plot shows the first 80 epochs of training: training longer without shuffling BN overfits more.

- **Solution:** ● P4 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.
- **Different feature distributions for supervised and self-supervised pre-training for downstream tasks**
 - (For linear classification protocol): ● P5 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]).
 - P5 These hyper-parameter values imply that the feature distributions (e.g., magnitudes) can be substantially different from those of ImageNet supervised training
 - ● P6 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.