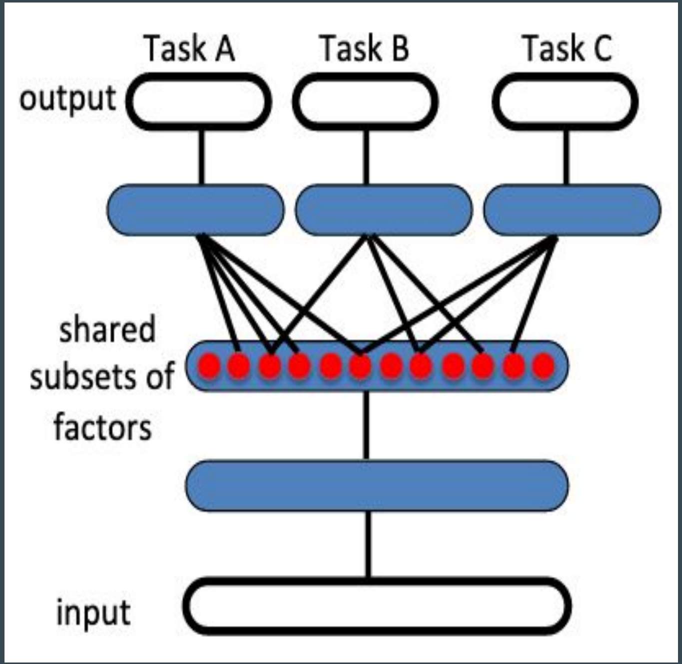
Momentum Contrast for Unsupervised Visual Representation Learning

Xiuyu Li

Representation Learning

- Learning representations of the data that make it easier to extract useful information
- The performance of machine learning methods is heavily dependent on the choice of data representation (or features)
- Learning representations that capture underlying factors, useful for transfer learning



Why Unsupervised?

Annotated data might be difficult to obtain, requiring costly human efforts and special domain expertise...

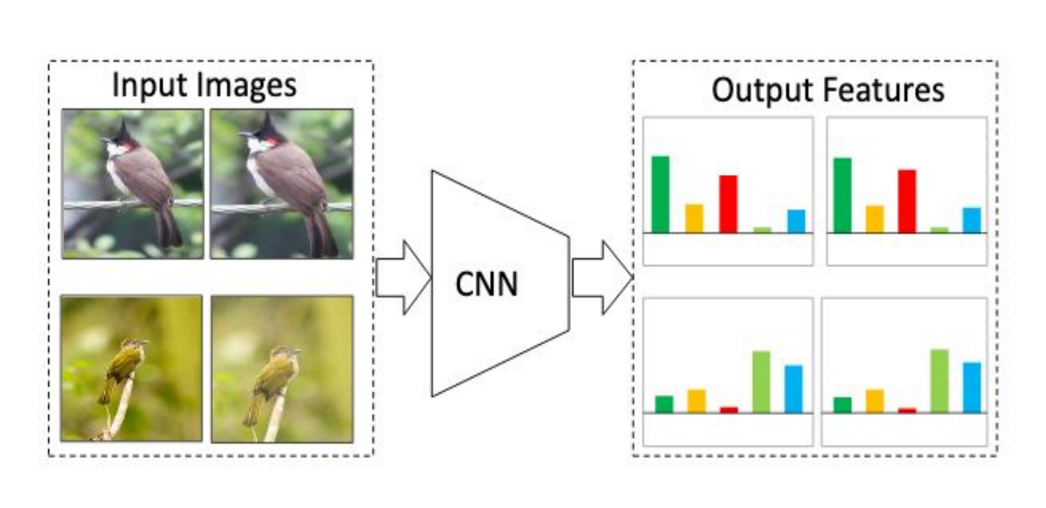
Use unsupervised learning to learn a good "intermediate" feature representation from unlabelled data!

Unsupervised Visual Representation Learning

Building *dynamic dictionaries*

- “keys” (tokens) sampled from data (e.g., images or patches), represented by an encoder network
- trains encoders to perform dictionary look-up: an encoded “query” should be similar to its matching key and dissimilar to others

Unsupervised Visual Representation Learning



- features of the same instance under different data augmentations should be invariant
- features of different image instances should be separated.
- So a query matches a key if they are encoded views (e.g., different crops) of the same image -> instance discrimination task!

Contrastive Learning as Dictionary Look-up

- Encoded query q and a set of encoded samples *(k0, k1, k2, ...)* that are the keys of a dictionary
- The query representation is $q = f_q(x_q)$, where f_q is an encoder network and x_q is a query sample (likewise, $k = f_k(x_k)$).

Contrastive Learning as Dictionary Look-up

We use Contrastive losses to measure the similarities of sample pairs in a representation space:

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

InfoNCE Loss

L_q is low when q is similar to its positive key $k+$ and dissimilar to all other keys (considered negative keys for q)

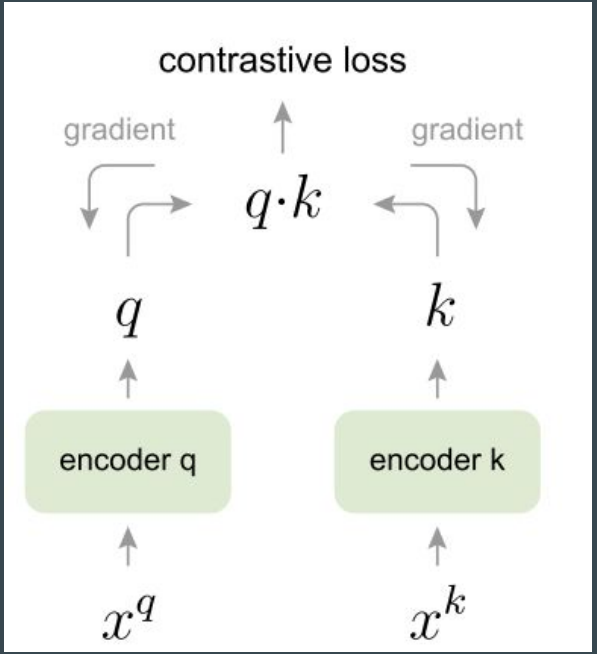
But...How to build dictionaries?

Assume the dictionaries should have the following two properties:

- Build large dictionaries
  - better sample the underlying continuous, high-dimensional visual space
  - cover a richer set of negative samples
- Build dictionaries that are consistent as they evolve during training
  - keys in the dictionary should be represented by the same or similar encoder so that their comparisons to the query are consistent
  - key encoder evolves during training

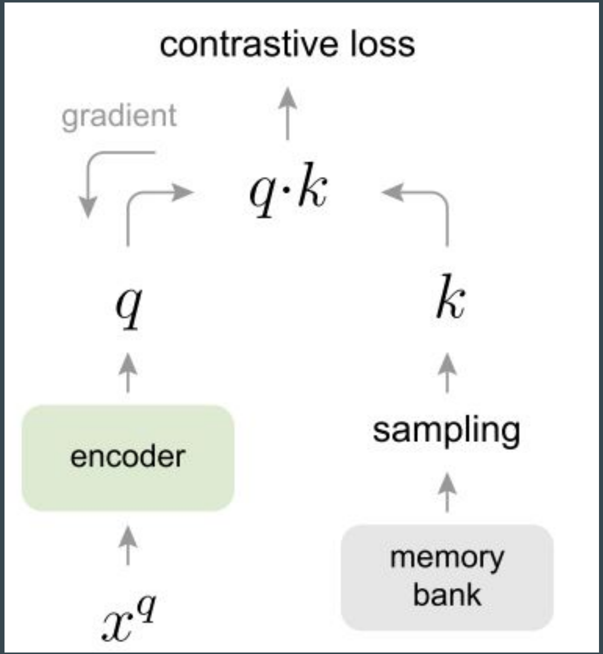
Previous Mechanisms I: End-to-End

- Uses samples in the current mini-batch as the dictionary
- Dictionary size coupled with the mini-batch size, **limited by the GPU memory size**
- Update by back-propagation



Previous Mechanisms II: Memory Bank

- A memory bank consists of the representations of all samples in the dataset, initialized as random unit vectors
- Dictionary for each mini-batch randomly sampled from the memory bank with **no back-propagation**
- Sampled keys are about the encoders at multiple different steps **all over the past epoch** -> less consistent!



Momentum Contrast (MoCo)

Our Method: MoCo

Dictionary as a queue. maintaining the dictionary as a queue of data samples

- Decouples the dictionary size from the mini-batch size
- Encoded current mini-batch are enqueued, the oldest are dequeued
- However, this brings some issues:
  - using a queue makes it intractable to update the key encoder by back-propagation
  - naive solution: copy the key encoder from the query encoder, ignoring the gradients
    - yields poor results in experiments
    - rapidly changing encoder reduces the key representations' consistency

Then, what should we do?

Our Method: MoCo

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

$$\theta_{\mathrm{k}} \leftarrow m\theta_{\mathrm{k}} + (1 - m)\theta_{\mathrm{q}}.$$

m $\in$ [0, 1) is a momentum coefficient, only θ_q are updated by back-propagation

Our Method: MoCo

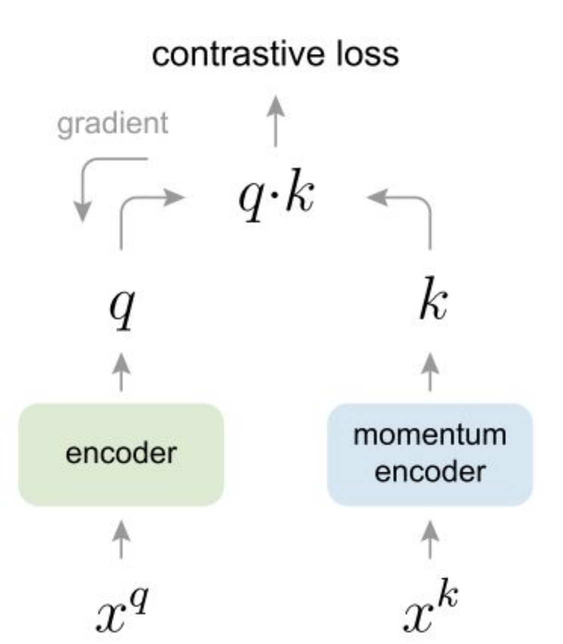
- θ_k evolve more smoothly than θ_q
- Though the keys in the queue are encoded by different encoders (in different mini-batches), the difference among these encoders can be made small
- Large momentum (e.g., m = 0.999) works much better than a smaller value (e.g., m = 0.9), suggesting that a slowly evolving key encoder is a core to making use of a queue
- Can be trained on billion-scale data

Our Method: MoCo

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.



Technical Details

- ResNet as the encoder, FC layer has fixed-dimensional output (128-D)
- Output vector (representation of query or key) normalized by its L2-norm
- Data augmentation (224×224-pixel crop, random color jittering, random horizontal flip, random grayscale conversion)

Shuffling BN

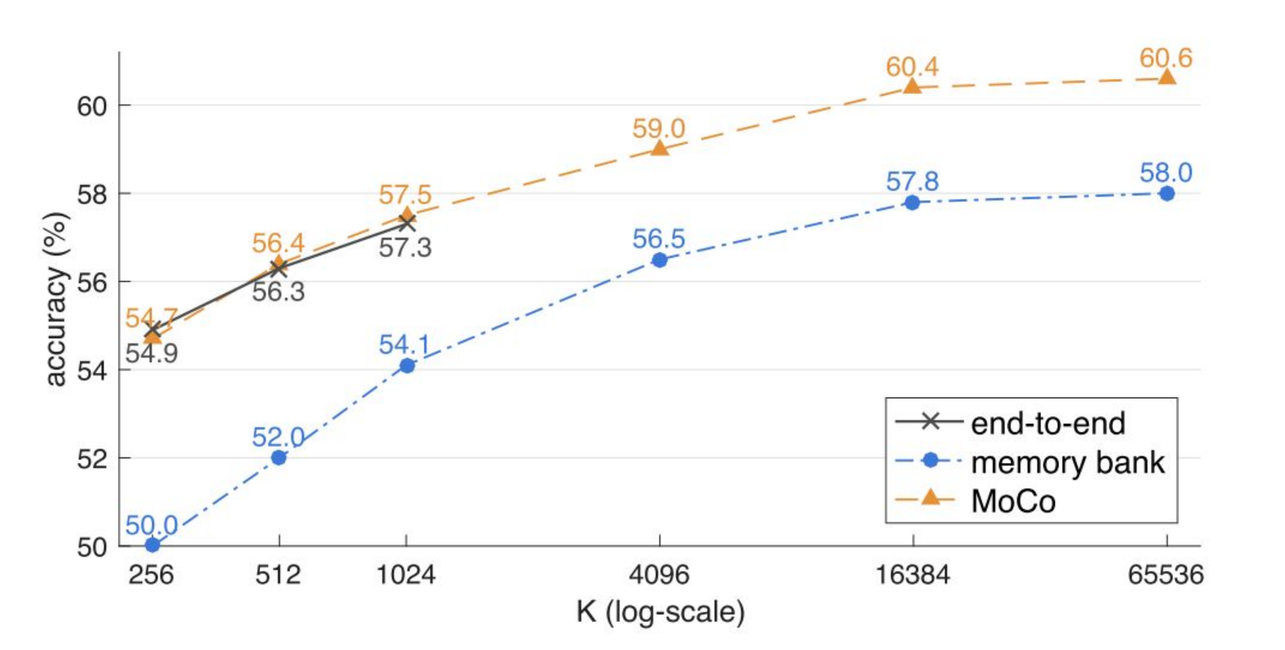
- Encoders both have Batch Normalization (BN), preventing the model from learning good representations
    - model appears to "cheat" the pretext task and easily finds a low-loss solution
    - intra-batch communication among samples (caused by BN) leaks information
- Solution: for f_k, shuffle the sample order in the current mini-batch before distributing it among GPUs (and shuffle back after encoding)
    - ensures the batch statistics used to compute a query and its positive key come from two different subsets

Emmm...this trick seems to be found by brutal force

Experiments Setting

- Dataset
  - ImageNet-1M (IN-1M): ImageNet training set that has ~1.28 million images in 1000 classes
  - Instagram-1B (IG-1B): ~1 billion (940M) public images from Instagram related to the ImageNet categories
- Linear Classification Protocol
  - unsupervised pre-training on IN-1M, freeze the features and train a supervised linear classifier (a fully-connected layer followed by softmax)
  - train this classifier on the global average pooling features of a ResNet, for 100 epochs

Ablation: Contrastive Loss Mechanisms

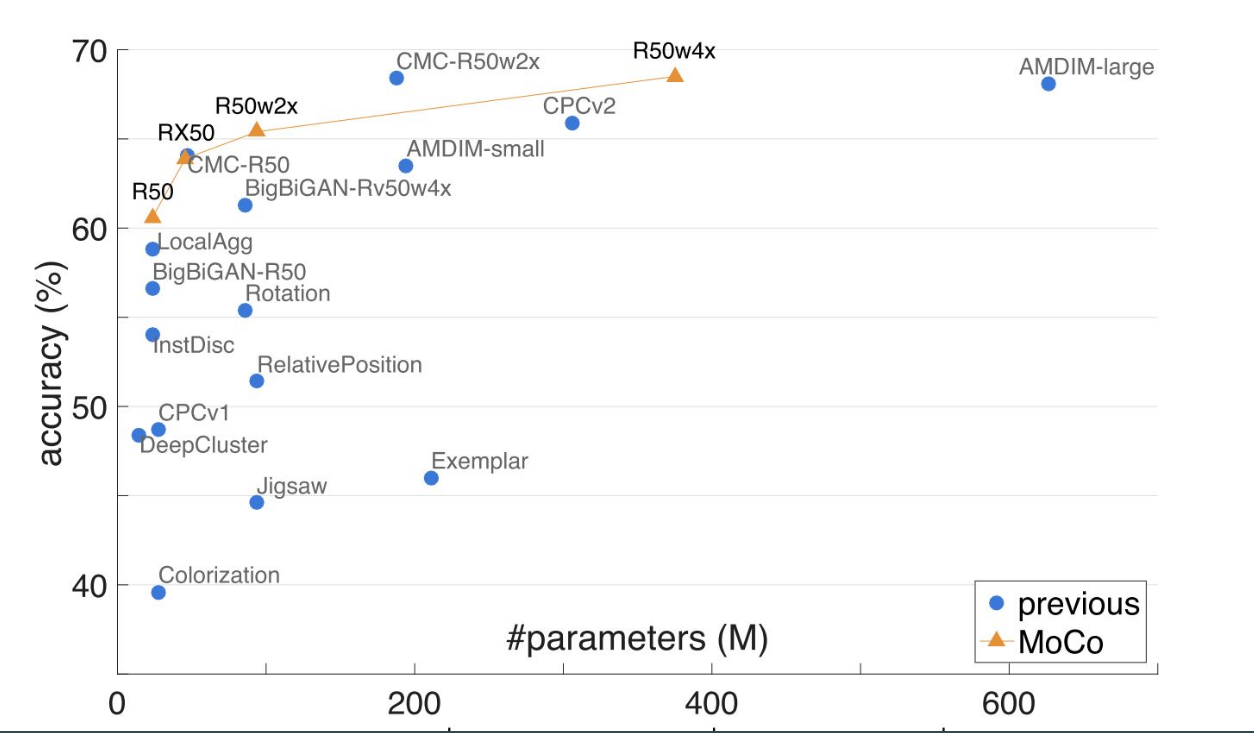


Ablation: Momentum

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

K = 4096

Comparison with Previous Results



Comparison with Previous Results

method	architecture	#params (M)	accuracy (%)
Exemplar [15]	R50w3×	211	46.0 [36]
RelativePosition [11]	R50w2×	94	51.4 [36]
Jigsaw [43]	R50w2×	94	44.6 [36]
Rotation [17]	Rv50w4×	86	55.4 [36]
Colorization [62]	R101*	28	39.6 [12]
DeepCluster [3]	VGG [51]	15	48.4 [4]
BigBiGAN [14]	R50	24	56.6
	Rv50w4×	86	61.3
methods based on contrastive learning follow:			
InstDisc [59]	R50	24	54.0
LocalAgg [64]	R50	24	58.8
CPC v1 [44]	R101*	28	48.7
CPC v2 [33]	$R170^{*}_{wider}$	303	65.9
CMC [54]	$R50_{L+ab}$	47	$64.1^{\dagger}$
	$R50w2\times_{L+ab}$	188	$68.4^{\dagger}$
AMDIM [2]	$AMDIM_{small}$	194	$63.5^{\dagger}$
	$AMDIM_{large}$	626	$68.1^{\dagger}$
MoCo	R50	24	60.6
	RX50	46	63.9
	R50w2×	94	65.4
	R50w4×	375	**68.6**

Object detection fine-tuned on PASCAL VOC

pre-train	AP_{50}	AP	AP_{75}
random init.	58.0	32.8	32.5
super. IN-1M	81.5	53.6	58.9
MoCo IN-1M	81.1 (−0.4)	53.8 (+0.2)	58.6 (−0.3)
MoCo IG-1B	81.6 (+0.1)	54.8 **(+1.2)**	60.3 **(+1.4)**

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

pre-train	AP_{50}	AP	AP_{75}
random init.	52.5	28.1	26.2
super. IN-1M	80.8	52.0	56.5
MoCo IN-1M	81.4 **(+0.6)**	55.2 **(+3.2)**	61.2 **(+4.7)**
MoCo IG-1B	82.1 **(+1.3)**	56.2 **(+4.2)**	62.3 **(+5.8)**

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

	R50-dilated-C5			R50-C4		
pre-train	AP_{50}	AP	AP_{75}	AP_{50}	AP	AP_{75}
end-to-end	77.8	50.1	53.8	79.7	53.0	57.9
memory bank	79.6	51.9	56.3	80.3	53.9	58.9
MoCo	**81.1**	**53.8**	**58.6**	**81.4**	**55.2**	**61.2**

Object detection fine-tuned on PASCAL VOC

pre-train	AP_{50} RelPos, by [12]	Multi-task [12]	Jigsaw, by [24]	LocalAgg [64]	**MoCo**	AP **MoCo**	AP_{75} Multi-task [12]	**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)

Object detection and instance segmentation fine-tuned on COCO

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

MoCo vs. ImageNet supervised pre-training (More)

pre-train	COCO keypoint detection 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**)

pre-train	COCO dense pose estimation 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**)

pre-train	LVIS v0.5 instance segmentation 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**)

pre-train	**Cityscapes instance seg.** AP^{mk}	AP^{mk}_{50}	**Semantic seg.** (mIoU) **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)

MoCo has largely closed the gap between unsupervised and supervised representation learning in multiple vision tasks!

Thank You!

- Comments? Questions?
- Qian: how does this method capture the distance in the embedding space?
- Horace: Why not use same encoder for both