ELLab

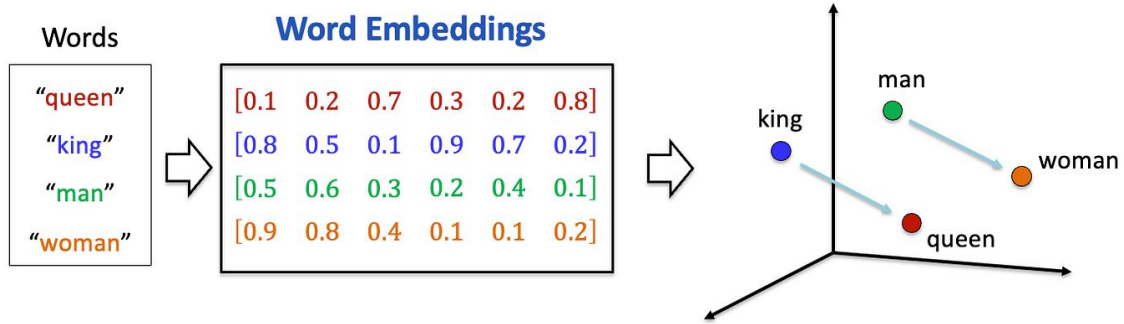
SUNG KYUN KWAN UNIVERSITY

# Momentum Contrast (MoCo)

2026.01.16

# Introduction

## Background




NLP

- discrete signal space → 명확한 token dictionary 존재함
- GPT, BERT 같은 unsupervised pre-training이 매우 성공적이었음

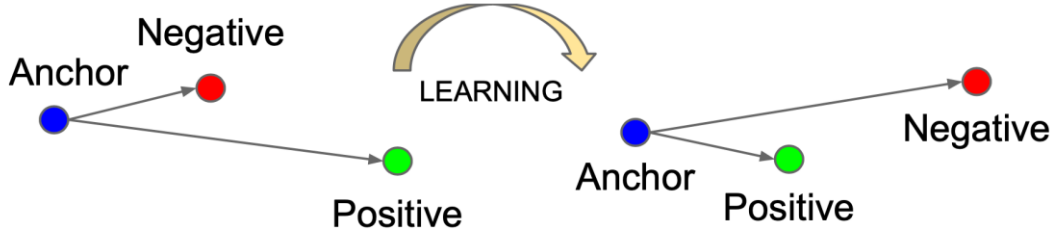
Vision

- continuous, high-dimensional pixel space → raw signal이 dictionary-friendly 하지 않음
- dictionary를 어떻게 만들지가 핵심 문제임

# Introduction

## Contrastive learning

- '비슷해야 할 것을 가깝게, 달라야 할 것은 멀게' 표현 공간에 배치하도록 representation을 학습하는 방법
- 데이터에 정답 라벨은 없음
- 대신 pair를 만들어서 학습에 하용
  - Positive pair: 같은 의미를 가진 두 샘플
  - Negative pair: 의미가 다른 샘플들
- Positive pair는 유사도가 높게, Negative pair는 유사도가 낮게 학습 하는 것이 목표



# Methodology

## Definition

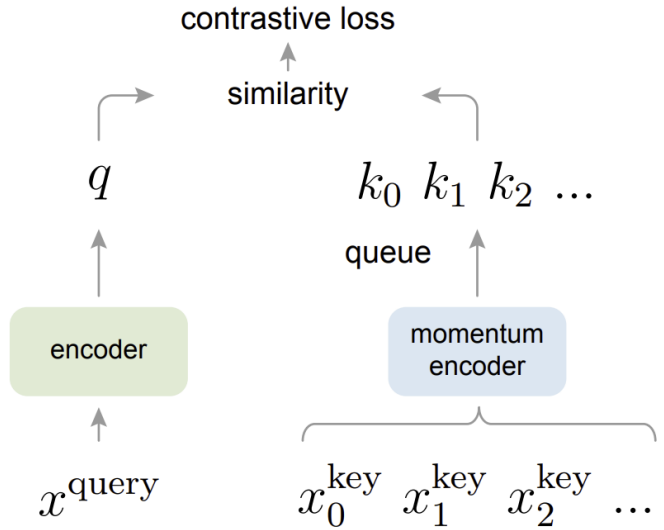
$x_q, x_k$: 같은 이미지의 다른 view (augmentation)

Query: $q = f_q(x^q)$

Key: $k = f_k(x^k)$

Dictionary as queue: $\{k_0, k_1, k_2, \ldots\}$

- 이중 하나만 positive key $(k_+)$이고 나머지는 모두 negative keys




## InfoNCE Loss

- Dot product 기반 유사도
- Positive key와의 유사도는 크게 negative key들과의 유사도는 작게 만듦
- (K+1)class 분류 문제의 log loss와 동일함

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

# Methodology

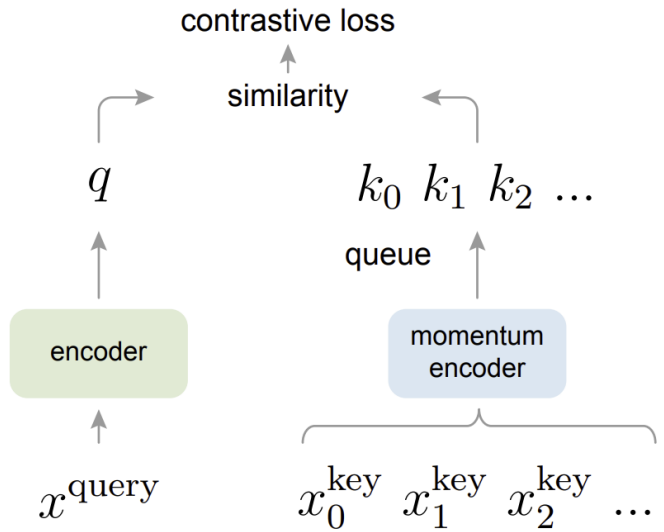
**Large dictionary**

- negative sample이 많을수록 의미 있는 표현 학습 가능
- 그러나, end-to-end 방식은 미니배치 크기 >= dictionary 크기
- GPU 메모리 때문에 batch size에 한계

→ **queue** 사용

- 현재 mini-batch에서 얻은 key들을 enqueue, 가장 오래된 key들은 dequeue

**dictionary size ≫ batch size, negative samples 재사용 가능해짐**




**Consistent dictionary**

- queue에 저장된 key들은 서로 다른 시점의 encoder로부터 생성됨
- encoder가 빠르게 변하면 key representation의 consistency 붕괴됨

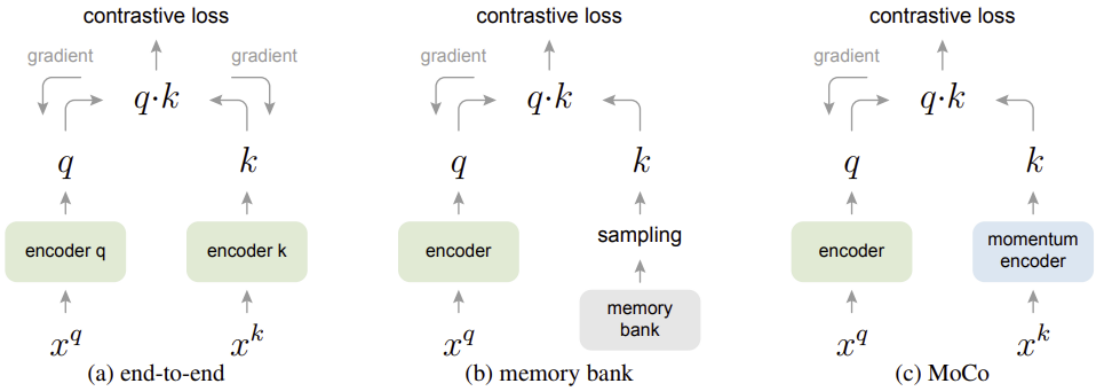
→ **momentum encoder** $\theta_{\mathrm{k}} \leftarrow m\theta_{\mathrm{k}} + (1-m)\theta_{\mathrm{q}}.$

**(query encoder)만 backprop으로 학습하고 $\theta_k$는 천천히 변화함**

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

# Methodology

## Comparison with Other Methods




(a) End-to-End Contrastive Learning

- query / key encoder 모두 backprop으로 업데이트
- 장점: key consistency 높음
- 단점: dictionary size ≤ batch size, large batch 필요 → 메모리 부담

(b) Memory Bank

- 전체 데이터의 representation 저장, mini-batch마다 random sampling
- 장점: 매우 큰 dictionary 가능
- 단점: key들이 서로 다른 encoder 시점에서 생성됨, representation consistency 낮음

(c) MoCo (Queue + Momentum Encoder)

- queue로 large dictionary 유지, momentum encoder로 consistency 확보
- large dictionary (by queue), stable representation (by momentum encoder)

# Methodology

## MoCo v2

### Stronger data augmentation

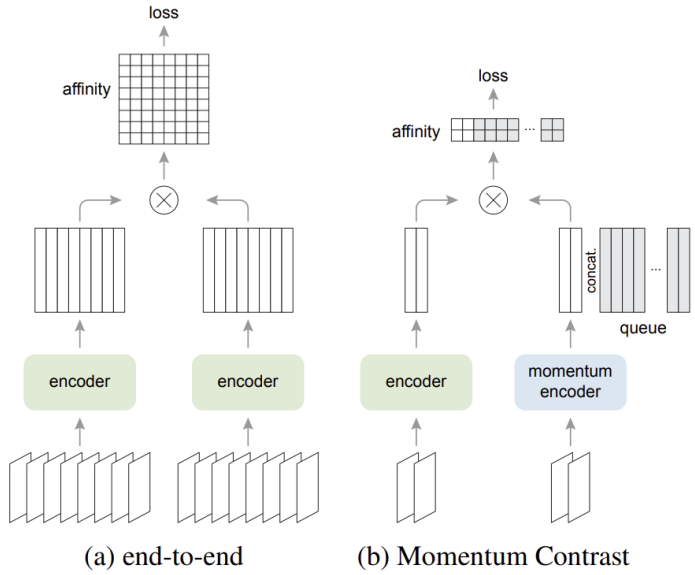
- 기존 MoCo augmentation + blur + stronger color distortion

### multi layer projection head

- encoder 뒤에 2-layer MLP head 추가
- contrastive loss는 projection space에서 계산
- backbone feature는 downstream task에 더 적합해짐

### Result

- 대규모 batch 없이 SimCLR 능가
- 계산 비용도 낮아서 현업에서 쓰기 좋음




(a) end-to-end (b) Momentum Contrast

| case | unsup. pre-train MLP | aug+ | cos | epochs | batch | ImageNet acc. |
|---|---|---|---|---|---|---|
| MoCo v1 [6] | | | | 200 | 256 | 60.6 |
| SimCLR [2] | ✓ | ✓ | ✓ | 200 | 256 | 61.9 |
| SimCLR [2] | ✓ | ✓ | ✓ | 200 | 8192 | 66.6 |
| **MoCo v2** | ✓ | ✓ | ✓ | 200 | 256 | **67.5** |
| *results of **longer** unsupervised training follow:* | | | | | | |
| SimCLR [2] | ✓ | ✓ | ✓ | 1000 | 4096 | 69.3 |
| **MoCo v2** | ✓ | ✓ | ✓ | 800 | 256 | **71.1** |

| mechanism | batch | memory / GPU | time / 200-ep. |
|---|---|---|---|
| MoCo | 256 | **5.0G** | **53 hrs** |
| end-to-end | 256 | 7.4G | 65 hrs |
| end-to-end | 4096 | 93.0G† | n/a |

# Experiment

## Comparison of three contrastive loss mechanisms

- backbone freeze & linear classifier (FC + softmax) 만 학습함
- 세 방법 모두 K가 커질수록 성능이 증가함

**end-to-end**

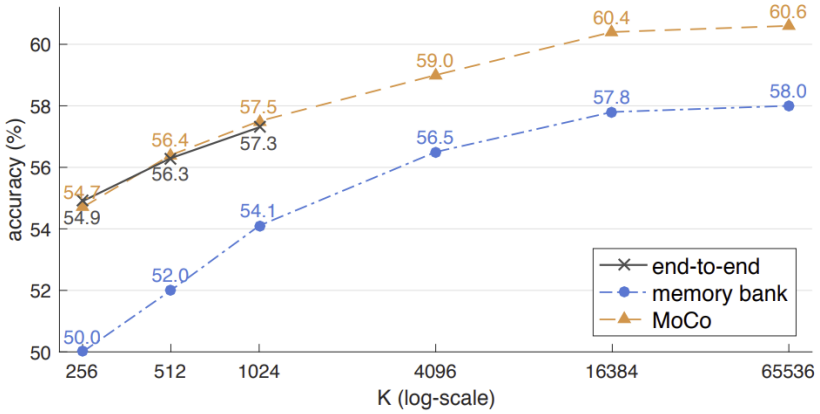
- K가 작을 때는 MoCo와 비슷하지만 large-batch optimization 자체가 어려움

**Memory Bank**

- 가장 큰 dictionary 가능하지만 key들이 서로 다른 시점의 encoder에서 생성되어서 representation consistency 낮음
- MoCo보다 성능이 열세함

**MoCo**

- large K에서도 안정적 성능 향상




| 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** |

# Experiment

## Transferring Features

### PASCAL VOC Object Detection (Fine-tuning)

- Faster R-CNN (C4 backbone)
- MoCo는 supervised pre-training과 비슷하거나 더 높은 정확도를 보임
- 대규모 unlabeled 데이터에서 더 큰 성능 이득을 얻음

| pre-train | $AP_{50}$ RelPos, by [14] | $AP_{50}$ Multi-task [14] | $AP_{50}$ Jigsaw, by [26] | $AP_{50}$ LocalAgg [66] | $AP_{50}$ **MoCo** | AP **MoCo** | $AP_{75}$ Multi-task [14] | $AP_{75}$ **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

| pre-train | $AP^{bb}$ | $AP^{bb}_{50}$ | $AP^{bb}_{75}$ | $AP^{mk}$ | $AP^{mk}_{50}$ | $AP^{mk}_{75}$ | $AP^{bb}$ | $AP^{bb}_{50}$ | $AP^{bb}_{75}$ | $AP^{mk}$ | $AP^{mk}_{50}$ | $AP^{mk}_{75}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| random init. | 26.4 | 44.0 | 27.8 | 29.3 | 46.9 | 30.8 | 35.6 | 54.6 | 38.2 | 31.4 | 51.5 | 33.5 |
| super. IN-1M | 38.2 | 58.2 | 41.2 | 33.3 | 54.7 | 35.2 | 40.0 | 59.9 | 43.1 | 34.7 | 56.5 | 36.9 |
| **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) | 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 | 39.1 (+0.9) | 58.7 (+0.5) | 42.2 (+1.0) | 34.1 (+0.8) | 55.4 (+0.7) | 36.4 (+1.2) | 41.1 (+1.1) | 60.7 (+0.8) | 44.8 (+1.7) | 35.6 (+0.9) | 57.4 (+0.9) | 38.1 (+1.2) |

(c) Mask R-CNN, R50-**C4**, **1**× schedule (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 Object Detection & Instance Segmentation

- 마찬가지로, supervised pre-training과 비슷하거나 더 높은 정확도를 보임
- 학습이 충분히 진행될수록 성능이 높아짐

**MoCo는 classification뿐 아니라, object detection과 instance segmentation 같은 고난도 downstream task에서도 supervised pre-training을 실질적으로 대체 가능함**

# My Code

## 코드 구현

- Data Augmentation: RandomResizedCrop ,Flip,  ColoJitter, GrayScale
- SplitTwoViews: $x \rightarrow \mathrm{q}, \mathrm{k}$
- SplitBatchNorm
- ResNetEncoder
- MoCoV2: momentum_update, dequeue_and_enqueue
- Traning: 100 epoch
- knn-evaluation

| 구분 | Accuracy |
|---|---|
| MoCo v2 | 67.55% |
| SimCLR | 76.51% |

CIFAR-10 데이터가 작고 쉬워서 SimCLR이 비교적 잘함

https://colab.research.google.com/drive/1RzCn_bB5vr_F7ng6GSLadSveyD_dHyN-?usp=sharing

# ELLab