Bootstrap Your Own Latent： A New Approach to Self-Supervised Learning (BYOL)

`20221212. Mon.`

- 논문 링크, *"Bootstrap Your Own Latent : A New Approach to Self-Supervised Learning"*, NIPS, 2020.
- 아카이브 버전 (코드) : 2006.07733.pdf (arxiv.org)

도입

- 이 논문 이전 5년 간, Self-Supervised Learning 논문 쏟아져 나왔고, scratch로부터 학습시킨 (random init) 모델보다는 더 나은 성능을 보였지만 Supervised Learning으로 학습시킨 Feature Extractor보다는 성능이 많이 뒤쳐짐
- 최근 Supervised Learning과 Self-Supervised Learning 사이의 격차가 많이 줄어드는 연구 성과들 등장
- 이 방법론들은 Contrastive Learning 기반의 방식을 사용, 기존 방식들보다 월등한 성능을 달성하며 큰 주목

> Some self-supervised methods are not contrastive but rely on using auxiliary handcrafted prediction tasks to learn their representation. In particular, relative patch prediction [23, 40], colorizing gray-scale images [41, 42], image inpainting [43], image jigsaw puzzle [44], image super-resolution [45], and geometric transformations [46, 47] have been shown to be useful. Yet, even with suitable architectures [48], these methods are being outperformed by contrastive methods [37, 8, 12].

Pretext task에서 Contrastive Learning으로의 패러다임 변화

- Contrastive Learning 기반의 Self-Supervised Learning 방법이 제안된 이후, Image Classification 분야에서는 대부분 Contrastive 접근을 하기 시작
- Bootstrap Your Own Latent 방법론을 정리하기 전에 2020년의 주요 Contrastive Learning 기반의 Self-Supervised Learning 알고리즘들을 간략히 훑어보고 시작하자!

Self-Supervised Learning in 2020

1. Momentum Contrast for Unsupervised Visual Representation Learning. (MoCo) 2019.11

- 이 논문은 2019년 11월쯤 Facebook AI Research에서 공개

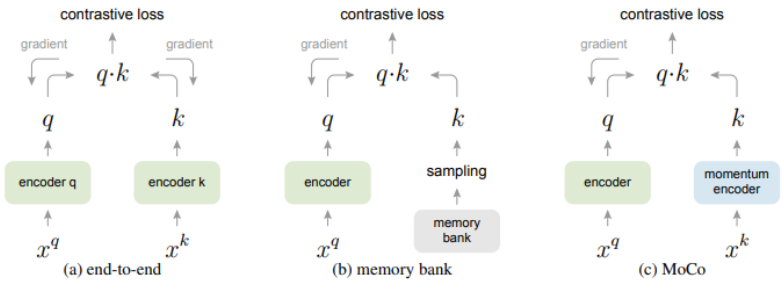


Figure 2. **Conceptual comparison of three contrastive loss mechanisms** (empirical comparisons are in Figure 3 and Table 3). Here we illustrate one pair of query and key. The three mechanisms differ in how the keys are maintained and how the key encoder is updated. **(a)**: The encoders for computing 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.

MoCo

- 이전의 “Contrastive Learning 기법을 Unsupervised Visual Representation Learning (Self-Supervised Learning)에 적용시키려는 시도”에서 학습 불안정 이슈를 극복하기 위해, Momentum을 적용하여 성능을 끌어올린 논문
  - Contrastive Learning 기반 방법론들은 같은 image에 서로 다른 augmentation을 가한 뒤, 두 positive pair의 feature representation은 거리가 가까워 지도록(유사해지도록) 학습
  - 다른 image에 서로 다른 augmentation을 가한 뒤, 두 negative pair의 feature representation은 거리가 멀어지도록 학습을 시키는 게 핵심
  - 이때, 학습 불안정 이슈 발생

2. A Simple Framework for Contrastive Learning of Visual Representations (SimCLR), 2020.1

- Google Research와 Brain Team에서 공개
- MoCo이후 2달 만에 더 큰 폭의 성능 향상을 보이며, Self-Supervised Learning의 가능성을 보여줌
- 마찬가지로 Contrastive Learning 기반
- 특정한 architecture나 memory bank 없이 학습을 시킬 수 있는 아이디어 제안
- 적절한 data augmentation 들을 사용하고, learnable nonlinear transformation와, contrastive loss를 제안하는 등 다양한 아이디어 소개

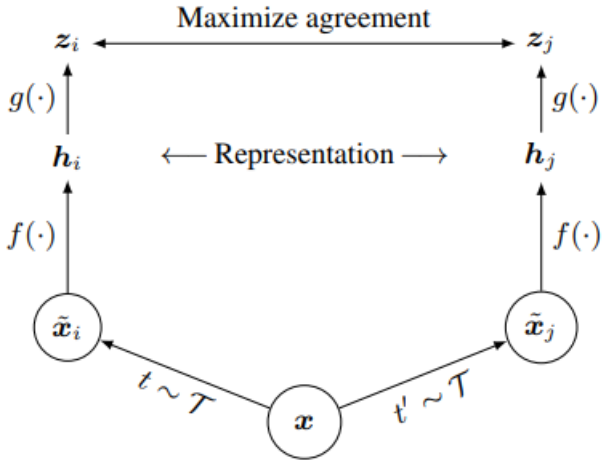


Figure 2. A simple framework for contrastive learning of visual representations. Two separate data augmentation operators are sampled from the same family of augmentations ($t \sim \mathcal{T}$ and $t' \sim \mathcal{T}$) and applied to each data example to obtain two correlated views. A base encoder network $f(\cdot)$ and a projection head $g(\cdot)$ are trained to maximize agreement using a contrastive loss. After training is completed, we throw away the projection head $g(\cdot)$ and use encoder $f(\cdot)$ and representation $\boldsymbol{h}$ for downstream tasks.

SimCLR

- SimCLR 논문 참고 : PR-231: A Simple Framework for Contrastive Learning of Visual Representations - YouTube

3. Improved Baselines with Momentum Contrastive Learning.(MoCo v2) 2020.3

- SimCLR이 공개된 지 2달 뒤, FAIR에서 MoCo의 2번째 버전인 **"Improved Baselines with Momentum Contrastive Learning"** 공개
- 딱 2페이지 짜리 technical report
- SimCLR의 design choice들을 고려하여 MoCo에 적용하여 성능을 올린 project
- PyTorch Official Code도 공개 : **"FAIR's MoCo PyTorch Code"**

case	unsup. pre-train MLP	aug+	cos	epochs	ImageNet acc.	VOC detection AP_{50}	AP	AP_{75}
supervised					76.5	81.3	53.5	58.8
MoCo v1				200	60.6	81.5	55.9	62.6
(a)	✓			200	66.2	82.0	56.4	62.6
(b)		✓		200	63.4	82.2	56.8	63.2
(c)	✓	✓		200	67.3	**82.5**	57.2	63.9
(d)	✓	✓	✓	200	67.5	82.4	57.0	63.6
(e)	✓	✓	✓	**800**	**71.1**	**82.5**	**57.4**	**64.0**

Table 1. **Ablation of MoCo baselines**, evaluated by ResNet-50 for (i) ImageNet linear classification, and (ii) fine-tuning VOC object detection (mean of 5 trials). "**MLP**": with an MLP head; "**aug+**": with extra blur augmentation; "**cos**": cosine learning rate schedule.

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**

Table 2. **MoCo *vs.* SimCLR**: ImageNet linear classifier accuracy (**ResNet-50, 1-crop 224×224**), trained on features from unsupervised pre-training. "aug+" in SimCLR includes blur and stronger color distortion. SimCLR ablations are from Fig. 9 in [2] (we thank the authors for providing the numerical results).

MoCo v2

- MoCo v1에 비해 v2 성능 많이 향상

- MoCo v2의 장점 : SimCLR의 핵심이었던 엄청 큰 batch size(8192)가 아닌 256 batch size에서도 더 좋은 성능을 달성한 점

4. Big Self-Supervised Models are Strong Semi-Supervised Learners.(SimCLR v2) 2020.6

- SimCLR의 2번째 버전 : **"Big Self-Supervised Models are Strong Semi-Supervised Learners"**
- 제목에서 알 수 있듯이 Semi-Supervised Learning에 초점을 두고 연구를 수행
- Parameter 수가 큰 모델일 수록, 적은 수의 labeled example만 존재할 때 SimCLR을 사용하면 성능 향상이 크다는 것을 보임
- 즉, 큰 model이 visual representation learning을 할 때 유리함을 시사
- 다만, 구체적인 downstream task가 정해진 경우에는 너무 큰 model 까지 갈 필요가 없음을 보여줌

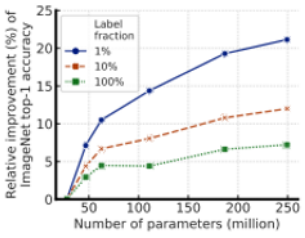


Figure 1: Bigger models yield larger gains when fine-tuning with fewer labeled examples.

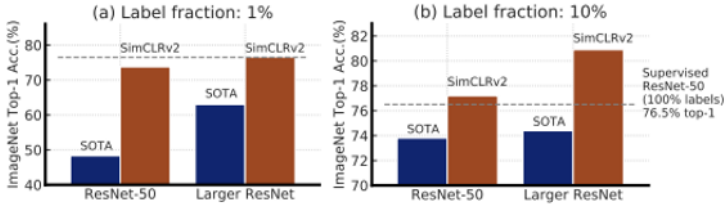
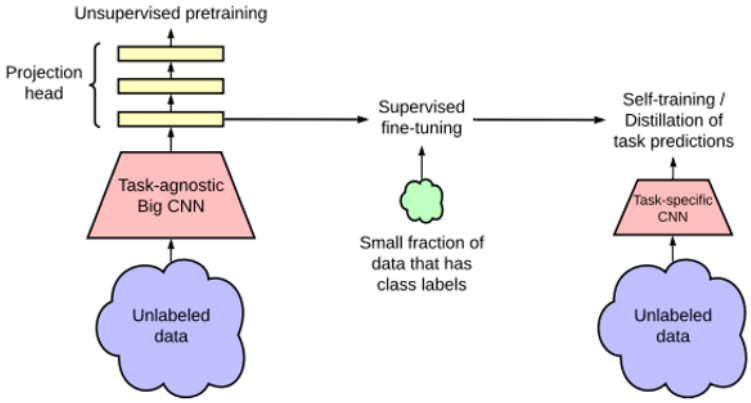


Figure 2: Top-1 accuracy of previous state-of-the-art (SOTA) methods [1, 2] and our method (SimCLRv2) on ImageNet using only 1% or 10% of the labels. Dashed line denotes fully supervised ResNet-50 trained with 100% of labels. Full comparisons in Table 3.



SimCLR v2

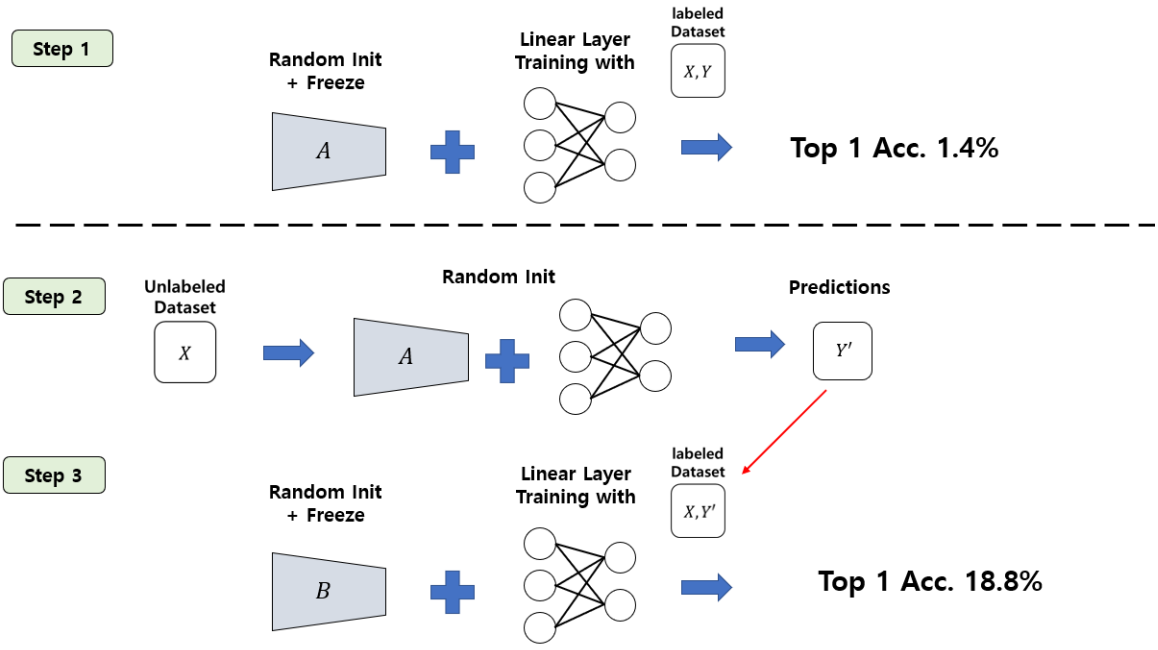
- **"PR-237: FixMatch: Simplifying Semi-Supervised Learning with Consistency and Confidence"** 논문이 Semi-Supervised Learning에서 좋은 성능을 보였었는데,
- SimCLR v2가 Label Fraction 10%인 상황에서, 같은 ResNet-50에 대하여 Top-1 Accuracy가 6% 포인트 올랐고, 더 큰 모델을 사용하면 Top-1 Accuracy가 9% 포인트 오르는 큰 성능 향상을 보임
- Google의 SimCLR도 TensorFlow 코드 공개 : **"Google's SimCLR TensorFlow Code"**

Bootstrap Your Own Latent(BYOL) 방법론 소개

- Contrastive Learning 기반 방법론들은 negative pair를 잘 선택 해줘야하고, 이를 위해 customized mining 전략 등이 제안 됨
- 또한 굉장히 큰 batch size에서 학습을 시켜야하고,
- 학습에 사용한 image augmentation option에도 성능 편차가 큰 것 등등… 아직까지는 안정적으로 학습시키기에 고려해야 할 요소가 많은 점이
- 본 논문에서는 negative pair를 사용하지 않고, 기존 Contrastive Learning 방식들보다 우수한 성능을 달성

간단한 실험을 통해 얻어낸 Insight

- 저자들은 negative pair를 사용하지 않고 image들로 부터 양질의 representation을 배우기 위해 image를 2개 이용하는 방식 대신, network를 2개 이용하는 방식을 구상
- 간단한 실험



간단한 실험을 통해 얻은 Insight

  - Step 1 :
    - **A**라는 network를 random initialization 시킨 뒤 weight를 고정
    - 여기에 Self-Supervised Learning의 evaluation protocol인 linear evaluation protocol을 통해 ImageNet 데이터셋에 대한 정확도를 측정
    - 즉, random init 후 freeze 시킨 feature extractor에 1개의 linear layer를 붙여서 ImageNet 데이터셋로 학습시킨 뒤 정확도를 측정하는 것
    - 이 경우, feature extractor가 아무런 정보도 배우지 않은 채 linear layer만 붙였기 때문에 좋은 성능을 얻을 수 없고, 실제로 1.4%의 top-1 accuracy를 달성
  - Step 2 :
    - unlabeled dataset을 random initialized **A** network + MLP (뒤에서 자세히 설명)에 feed forward 시켜서 prediction들을 얻어냄

- Step 3 :
  - 또 다른 network B 준비. 이것도 마찬가지로 random initialization 시키는데, 바로 linear evaluation을 수행하지 않고, image들을 A network에 feed forward 시켜서 뽑아낸 prediction을 target으로 하여 이 target을 배우도록 학습
  - 비유를 하자면, A라는 학생은 모의고사를 볼 때 아무런 답이나 내뱉는 학생인데, 시험을 보기 전 약간의 벼락치기(linear layer training)를 합니다. 그 결과 1.4점의 시험 결과를 얻습니다. B라는 학생은 A 학생이 벼락치기를 하기 전(random initialized A) 답을 적어둔 연습 문제들을 받아서 이 답안들을 배우도록 학습을 한 뒤, 벼락치기
  - 어떻게 보면 A 학생보다 공부 량은 좀 더 많겠죠?

- 결과 :
  - B network : random initialized A network가 내뱉은 부정확한 prediction들을 배우도록 학습을 한 뒤 linear evaluation을 하였을 때 18.8%의 top-1 accuracy 얻음
  - 물론 이 수치는 매우 낮은 수치이지만, random initialization을 하였을 때 보다는 굉장히 큰 폭으로 성능이 증가한 셈
  - 저자들의 서술에 의하면, 이 간단한 실험이 BYOL의 core motivation이 되었다고 함

BYOL's architecture

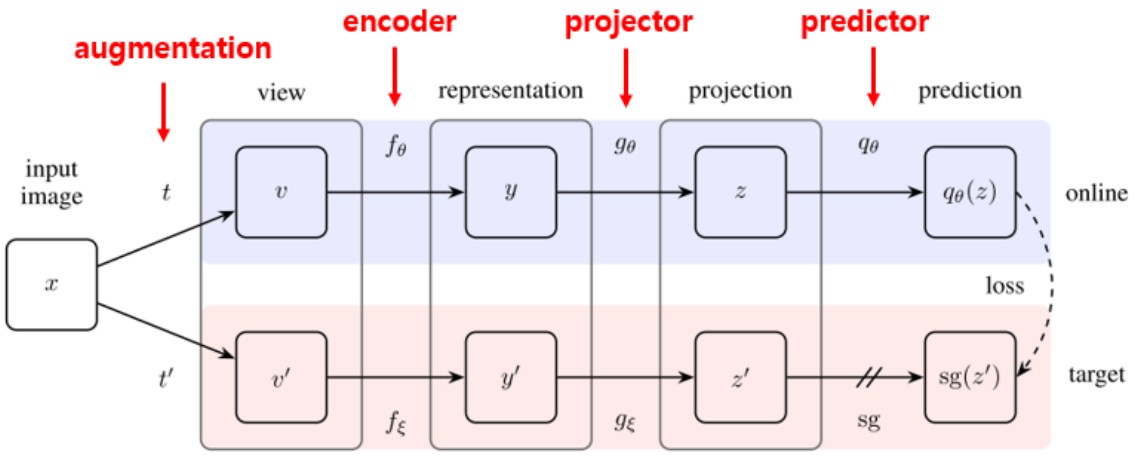


Figure 2: BYOL's architecture. BYOL minimizes a similarity loss between $q_\theta(z)$ and $\mathrm{sg}(z')$, where θ are the trained weights, ξ are an exponential moving average of θ and sg means stop-gradient. At the end of training, everything but f_θ is discarded and y is used as the image representation.

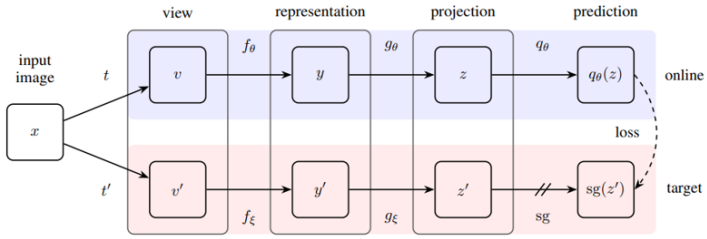
BYOL's architecture

J.2 Network definition

```
def network(inputs):
  """Build the encoder, projector and predictor."""
  embedding = ResNet(name='encoder', configuration='ResNetV1_50x1')(inputs)
  proj_out = MLP(name='projector')(embedding)
  pred_out = MLP(name='predictor')(proj_out)
  return dict(projection=proj_out, prediction=pred_out)


class MLP(hk.Module):
  """Multi Layer Perceptron, with normalization."""

  def __init(self, name):
    super().__init__(name=name)

  def __call__(self, inputs):
    out = hk.Linear(output_size=HPS['mlp_hidden_size'])(inputs)
    out = hk.BatchNorm(**HPS['batchnorm_kwargs'])(out)
    out = jax.nn.relu(out)
    out = hk.Linear(output_size=HPS['projection_size'])(out)
    return out

# For simplicity, we omit BatchNorm related states.
# In the actual code, we use hk.transform_with_state. The corresponding
# net_init function outputs both a params and a state variable,
# with state containing the moving averages computed by BatchNorm
net_init, net_apply = hk.transform(network)
```

2006.07733.pdf (arxiv.org)

- 위의 예에서 **A-network = target network**의 역할, **B-network = online network**의 역할
- **online** network와 **target** network로 구성
  - 각 network는 encoder (f) 와 projector (g), predictor (q) 로 구성
- 학습이 끝나면 online network의 encoder 를 제외한 나머지 요소들은 사용이 되지 않을 예정
  - 이유 : Self-Supervised Learning의 목적은 위의 그림에 있는 representation y를 잘 배워서 downstream task에 잘 활용하는 것이기 때문
- encoder = resnet50을 사용
- projector, predictor = MLP 사용
- **BYOL의 핵심 요소**
  1. **target network** = **online network가 배울 regression target 생성**
  2. **target network의 weight들** = **online network의 weight들의 exponential moving average 값을 사용**
- **BYOL 핵심 알고리즘**
  - **exponential moving average** :
    - coefficient 값 : 0.996에서 시작하여 점점 1에 가까운 값으로 키워나감



$\overline{q_\theta}(z_\theta) \triangleq q_\theta(z_\theta)/\|q_\theta(z_\theta)\|_2$ and $\overline{z}'_\xi \triangleq z'_\xi/\|z'_\xi\|_2$

$$\xi \leftarrow \tau\xi + (1-\tau)\theta.$$

target weight = τ* target weight
+ $(1-\tau)$ * online weight

$$\mathcal{L}_\theta^{\text{BYOL}} \triangleq \|\overline{q_\theta}(z_\theta) - \overline{z}'_\xi\|_2^2 = 2 - 2 \cdot \frac{\langle q_\theta(z_\theta), z'_\xi \rangle}{\|q_\theta(z_\theta)\|_2 \cdot \|z'_\xi\|_2}.$$

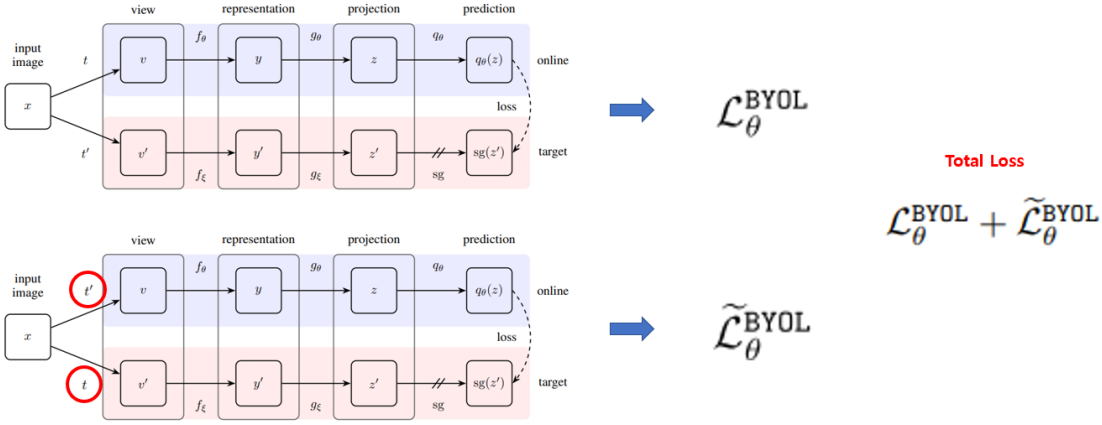
Mean squared error with l_2 normalize

$$\tau_{\text{base}} = 0.996\ , \quad \tau \triangleq 1 - (1-\tau_{\text{base}}) \cdot (\cos(\pi k/K) + 1)/2$$

BO/

- 그림의 좌측 하단에 있는 식 ($\xi \leftarrow \tau\xi + (1-\tau)\theta$) 처럼 iteration에 따라 cosine annealing을 해줌
- loss function = mse (mean squared error) function 사용 : online network와 target network의 projection output들에 $\mathcal{L}_2$ normalization 시킨 값들간의 mean squred error를 구하여 loss를 계산 ($\mathcal{L}_\theta^{\text{BYOL}}$)

◦ 추가된 트릭 : **loss의 symmetrize (대칭화)**



BYOL의 loss function symmetrization

- loss를 symmetrize(대칭화) 시키기 위해, 사용한 augmentation을 바꿔서 loss를 한번 더 계산
- 즉, online network에 t'-augmentation (t'-transformation)을 가하고 target network에 t-augmentation (t-transformation)을 가한 뒤 loss 계산 ($\widetilde{\mathcal{L}}_\theta^{\text{BYOL}}$)

◦ 위의 두 loss에 대한 total loss를 학습에 사용 : $\mathcal{L}_{total} = \mathcal{L}_\theta^{\text{BYOL}} + \widetilde{\mathcal{L}}_\theta^{\text{BYOL}}$이

- 이 loss는 online network를 학습시키는 데에만 사용
- target network의 weight는 exponential moving average에 의해 결정됨

◦ 그림의 **sg**는 stop-gradient를 의미 : back propagation을 시키지 않겠다는 의미!!

- 비유를 하자면… **과거의 내가 오늘의 스승이 된다?!**

- 과거의 online network들이 target network 속에 녹아들면서 오늘의 스승(target network)이 되는 셈

> 🔷 **요약**

- 2개의 network 사용 : online network, target network
- target network의 weight = online network의 weight의 exponential moving average에 의해 결정
- 2개의 network에 서로 다른 augmentation을 적용하여 feature vector(prediction)을 뽑은 뒤,
- $\mathcal{L}_2$ normalize 시킨 후 MSE를 최소화하는 방향으로 online network를 학습시킴

Implementation details

- 실험을 위한 구현 디테일

3.2 Implementation details

Image augmentations BYOL uses the same set of image augmentations as in SimCLR [8]. First, a random patch of the image is selected and resized to 224×224 with a random horizontal flip, followed by a color distortion, consisting of a random sequence of brightness, contrast, saturation, hue adjustments, and an optional grayscale conversion. Finally Gaussian blur and solarization are applied to the patches. Additional details on the image augmentations are in Appendix B.

Architecture We use a convolutional residual network [19] with 50 layers and post-activation (ResNet-50($1\times$) v1) as our base parametric encoders f_θ and f_ξ. We also use deeper (50, 101, 152 and 200 layers) and wider (from $1\times$ to $4\times$) ResNets, as in [54, 44, 8]. Specifically, the representation y corresponds to the output of the final average pooling layer, which has a feature dimension of 2048 (for a width multiplier of $1\times$). As in SimCLR [8], the representation y is projected to a smaller space by a *multi-layer perceptron* (MLP) g_θ, and similarly for the target projection g_ξ. This MLP consists in a linear layer with output size 4096 followed by batch normalization [55], rectified linear units (ReLU) [56], and a final linear layer with output dimension 256. Contrary to SimCLR, the output of this MLP is not batch normalized. The predictor q_θ uses the same architecture as g_θ.

Optimization We use the LARS optimizer [57] with a cosine decay learning rate schedule [58], without restarts, over 1000 epochs, with a warm-up period of 10 epochs. We set the base learning rate to 0.2, scaled linearly [59] with the batch size (LearningRate $= 0.2 \times$ BatchSize$/256$). In addition, we use a global weight decay parameter of $1.5 \cdot 10^{-6}$ while excluding the biases and batch normalization parameters from both LARS adaptation and weight decay. For the target network, the exponential moving average parameter τ starts from $\tau_{\text{base}} = 0.996$ and is increased to one during training. Specifically, we set $\tau \triangleq 1 - (1 - \tau_{\text{base}}) \cdot (\cos(\pi k/K) + 1)/2$ with k the current training step and K the maximum number of training steps. We use a batch size of 4096 split over 512 Cloud TPU v3 cores. With this setup, training takes approximately 8 hours for a ResNet-50($\times 1$). All hyperparameters are summarized in Appendix G.

BYOL의 Implementation details

- 대부분 SimCLR의 셋팅을 따라서 진행, 추가로 학습에 사용한 hyper parameter는 Appendix J에서 확인

J BYOL pseudo-code in JAX

J.1 Hyper-parameters

```
HPS = dict(
    max_steps=int(1000. * 1281167 / 4096),  # 1000 epochs
    batch_size=4096,
    mlp_hidden_size=4096,
    projection_size=256,
    base_target_ema=4e-3,
    optimizer_config=dict(
        optimizer_name='lars',
        beta=0.9,
        trust_coef=1e-3,
        weight_decay=1.5e-6,
        # As in SimCLR and official implementation of LARS, we exclude bias
        # and batchnorm weight from the Lars adaptation and weightdecay.
        exclude_bias_from_adaption=True),
    learning_rate_schedule=dict(
        # The learning rate is linearly increase up to
        # its base value * batchisze / 256 after warmup_steps
        # global steps and then anneal with a cosine schedule.
        base_learning_rate=0.2,
        warmup_steps=int(10. * 1281167 / 4096),
        anneal_schedule='cosine'),
    batchnorm_kwargs=dict(
        decay_rate=0.9,
        eps=1e-5),
    seed=1337,
    )
```

BYOL의 Hyper-parameters

실험 결과 및 분석

- 실험 방식은 Self-Supervised Learning의 표준을 따라서 진행

첫 번째 실험 : ImageNet linear evaluation & semi-supervised setup

For ImageNet dataset (unlabeled dataset),

- Self Supervised Learning 방식으로 Pre-training을 시킨 뒤,
- feature extractor(encoder)를 freeze 시키고, 아래의 두 실험 수행
  - linear classifier를 학습시키는 **linear evaluation 실험**
  - 일부분의 labeled training set을 가지고 feature extractor를 fine-tuning 시키는(do not freeze) **semi-supervised training 실험**

Method	Top-1	Top-5
Local Agg.	60.2	-
PIRL [32]	63.6	-
CPC v2 [29]	63.8	85.3
CMC [11]	66.2	87.0
SimCLR [8]	69.3	89.0
MoCo v2 [34]	71.1	-
InfoMin Aug. [12]	73.0	91.1
BYOL (ours)	**74.3**	**91.6**

(a) ResNet-50 encoder.

Method	Architecture	Param.	Top-1	Top-5
SimCLR [8]	ResNet-50 (2×)	94M	74.2	92.0
CMC [11]	ResNet-50 (2×)	94M	70.6	89.7
BYOL (ours)	ResNet-50 (2×)	94M	77.4	93.6
CPC v2 [29]	ResNet-161	305M	71.5	90.1
MoCo [9]	ResNet-50 (4×)	375M	68.6	-
SimCLR [8]	ResNet-50 (4×)	375M	76.5	93.2
BYOL (ours)	ResNet-50 (4×)	375M	78.6	94.2
BYOL (ours)	ResNet-200 (2×)	250M	**79.6**	**94.8**

(b) Other ResNet encoder architectures.

Table 1: Top-1 and top-5 accuracies (in %) under linear evaluation on ImageNet.

Method	Top-1		Top-5	
	1%	10%	1%	10%
Supervised [64]	25.4	56.4	48.4	80.4
InstDisc	-	-	39.2	77.4
PIRL [32]	-	-	57.2	83.8
SimCLR [8]	48.3	65.6	75.5	87.8
BYOL (ours)	**53.2**	**68.8**	**78.4**	**89.0**

(a) ResNet-50 encoder.

Method	Architecture	Param.	Top-1		Top-5	
			1%	10%	1%	10%
CPC v2 [29]	ResNet-161	305M	-	-	77.9	91.2
SimCLR [8]	ResNet-50 (2×)	94M	58.5	71.7	83.0	91.2
BYOL (ours)	ResNet-50 (2×)	94M	62.2	73.5	84.1	91.7
SimCLR [8]	ResNet-50 (4×)	375M	63.0	74.4	85.8	92.6
BYOL (ours)	ResNet-50 (4×)	375M	69.1	75.7	87.9	92.5
BYOL (ours)	ResNet-200 (2×)	250M	**71.2**	**77.7**	**89.5**	**93.7**

(b) Other ResNet encoder architectures.

Table 2: Semi-supervised training with a fraction of ImageNet labels.

ImageNet 실험 결과

1. linear evaluation 실험 결과 (Table 1)
    - SimCLR, MoCov2, InfoMin Aug 등 선행 연구들보다 더 좋은 성능 달성
    - 하지만 SimCLR v2에서 같은 ResNet-50 encoder에 Selective Kernel Network(SKNet) 구조를 사용하여 linear evaluation 하면 74.6% (BYOL보다 0.3% 포인트 높은 성능을 달성…)
2. Semi-supervised setup
    - 1%, 10%의 training sample로 fine-tuning을 시켰을 때의 실험 결과를 보여주고 있으며, 기존 방법들 대비 우수한 성능을 보여주고 있음
    - 또한 Supervised [64] 라고 표시되어있는 성능은 아무런 pre-training 없이 학습을 시킨 성능으로 어떻게 보면 random initialization 성능이라고 볼 수 있음 (당연히 낮은 성능)
    - BYOL이 다른 방법들에 비해 높은 성능을 보이고 있는데, 역시 예상하듯 SimCLR v2에서는 더 높은 semi-supervised 성능을 보이고 있음

두 번째 실험 : Transfer Learning

두번째 실험 결과로는 **1. Classification의 다른 데이터셋으로 Transfer 시키는 실험** 결과와, **2. 다른 Task로 Transfer 시킨 실험** 결과를 제시

Method	Food101	CIFAR10	CIFAR100	Birdsnap	SUN397	Cars	Aircraft	VOC2007	DTD	Pets	Caltech-101	Flowers
Linear evaluation:												
BYOL (ours)	**75.3**	91.3	**78.4**	**57.2**	**62.2**	**67.8**	60.6	82.5	75.5	90.4	94.2	**96.1**
SimCLR (repro)	72.8	90.5	74.4	42.4	60.6	49.3	49.8	81.4	**75.7**	84.6	89.3	92.6
SimCLR [8]	68.4	90.6	71.6	37.4	58.8	50.3	50.3	80.5	74.5	83.6	90.3	91.2
Supervised-IN [8]	72.3	**93.6**	78.3	53.7	61.9	66.7	**61.0**	**82.8**	74.9	**91.5**	**94.5**	94.7
Fine-tuned:												
BYOL (ours)	**88.5**	**97.8**	86.1	**76.3**	63.7	91.6	**88.1**	**85.4**	**76.2**	91.7	**93.8**	97.0
SimCLR (repro)	87.5	97.4	85.3	75.0	63.9	91.4	87.6	84.5	75.4	89.4	91.7	96.6
SimCLR [8]	88.2	97.7	85.9	75.9	63.5	91.3	88.1	84.1	73.2	89.2	92.1	97.0
Supervised-IN [8]	88.3	97.5	**86.4**	75.8	**64.3**	**92.1**	86.0	85.0	74.6	**92.1**	93.3	**97.6**
Random init [8]	86.9	95.9	80.2	76.1	53.6	91.4	85.9	67.3	64.8	81.5	72.6	92.0

Table 3: Transfer learning results from ImageNet (IN) with the standard ResNet-50 architecture.

Method	AP_{50}	mIoU
Supervised-IN [9]	74.4	74.4
MoCo [9]	74.9	72.5
SimCLR (repro)	75.2	75.2
BYOL (ours)	**77.5**	**76.3**

(a) Transfer results in semantic segmentation and object detection.

Method	Higher better			Lower better	
	pct.< 1.25	pct.$< 1.25^2$	pct.$< 1.25^3$	rms	rel
Supervised-IN [70]	81.1	95.3	98.8	0.573	**0.127**
SimCLR (repro)	83.3	96.5	99.1	0.557	0.134
BYOL (ours)	**84.6**	**96.7**	**99.1**	**0.541**	0.129

(b) Transfer results on NYU v2 depth estimation.

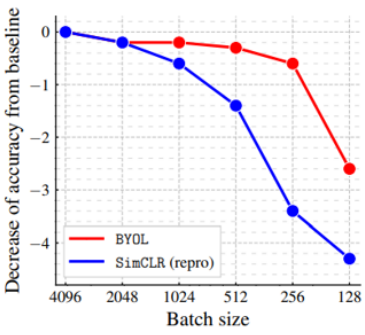
Table 4: Results on transferring BYOL's representation to other vision tasks.

Transfer Learning 실험 결과

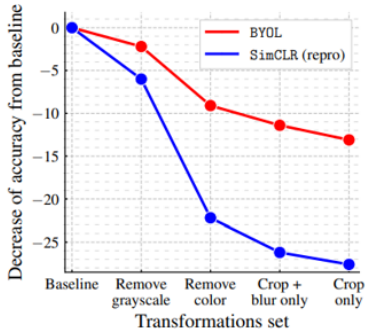
1. 다른 classification 데이터셋으로 transfer 시키는 실험에 사용한 데이터셋 : Food101, CIFAR 등 12가지 데이터셋을 사용 (Table 3)
    - linear evaluation과 fine-tuning 실험 결과를 보여주고 있음
    - 위 그림의 Table 3을 보면 대부분의 경우에서 BYOL이 좋은 성능
    - 주목할만한 실험 결과 : Fine-tuning 실험에서 Supervised-IN[8]의 결과보다 BYOL의 결과가 더 좋은 경우가 존재한다는 점 (7승 5패)
        - 즉, 기존에는 default로 ImageNet-Pretrained 셋팅에서 다른 데이터셋에 fine-tuning 시켜왔는데, BYOL을 사용하면 ImageNet-Pretrained 셋팅과 비슷하거나 더 좋은 효과를 볼 수 있다는 이야기
        - 주목할 점 : BYOL은 ImageNet의 Label을 아예 사용하지 않고 Self-Supervised Learning으로 학습을 시켰다는 점
        - 즉, ImageNet의 Label 정보를 사용하지 않고 feature extractor를 학습시켰는데, 여기서 얻은 representation들이 Label을 사용하여 얻은 representation과 견줄만하다는 뜻
2. Object Detection, Semantic Segmentation, Depth Estimation 등 다른 task에 feature extractor로 사용하였을 때의 실험 결과 (Table 4)
    - ImageNet으로 Pre-training을 시켰을 때보다 BYOL을 사용하였을 때 더 좋은 성능을 보이고, SimCLR 보다도 더 좋은 성능을 보임

Ablations을 통해 얻은 intuitions

굉장히 다양한 실험을 수행하여 얻은 Intuition들을 2페이지에 걸쳐 정리 (논문 참고)



(a) Impact of batch size



(b) Impact of progressively removing transformations

Figure 3: Decrease in top-1 accuracy (in % points) of BYOL and our own reproduction of SimCLR at 300 epochs, under linear evaluation on ImageNet.

Target	τ_{base}	Top-1
Constant random network	1	$18.8_{\pm 0.7}$
Moving average of online	0.999	69.8
Moving average of online	0.99	**72.5**
Moving average of online	0.9	68.4
Stop gradient of online†	0	0.3

(a) Results for different target modes. †In the *stop gradient of online*, $\tau = \tau_{\text{base}} = 0$ is kept constant throughout training.

Method	Predictor	Target network	β	Top-1
BYOL	✓	✓	0	**72.5**
	✓	✓	1	70.9
		✓	1	70.7
SimCLR			1	69.4
	✓		1	69.1
	✓		0	0.3
		✓	0	0.2
			0	0.1

(b) Intermediate variants between BYOL and SimCLR.

Table 5: Ablations with top-1 accuracy (in %) at 300 epochs under linear evaluation on ImageNet.

Ablation Study & Intuitions

- SimCLR은 batch size가 작을 때 성능이 많이 망가졌었는데 BYOL은 덜 망가진다!
- Contrastive method에 비해 image augmentation option에 robust하다.
- Exponential moving average coefficient 값에 따라 성능이 약간씩 차이가 존재하지만 대체로 잘 된다!
  - 1이면 constant random network (위의 간단한 실험에서의 A network 역할)
  - 0이면 online weight를 그대로 target weight로 사용
  - 둘 다 좋은 성능을 보이지 못함
  - 즉, 적절히 target weight에 좀 delayed 되었지만 안정적인 online network의 weight로 넣어주는 것이 효과적임
- BYOL과 SimCLR을 InfoNCE objective로 표현하여 두 알고리즘의 근본적인 차이를 분석함
- 요약하면 SimCLR은 predictor가 없고 target network가 없음
- BYOL은 negative example이 없음
- BYOL에 negative example을 주고 학습을 시켰을 때는 오히려 성능이 떨어지지만, 적절한 tuning을 하면 사용하지 않았을 때와 비슷한 성능을 달성할 수 있음 (굳이 사용 안 하는 게 나음)

결론

Google과 Facebook 의 양강 구도였는데 DeepMind가 기존 Contrastive Learning 방식이 갖는 한계점들을 하나씩 극복하며 좋은 성능을 달성할 수 있는 연구를 공개하였고, 이 논문 정리한 것임