A Large-Scale Study on Unsupervised Spatiotemporal Representation Learning

CVPR 2021

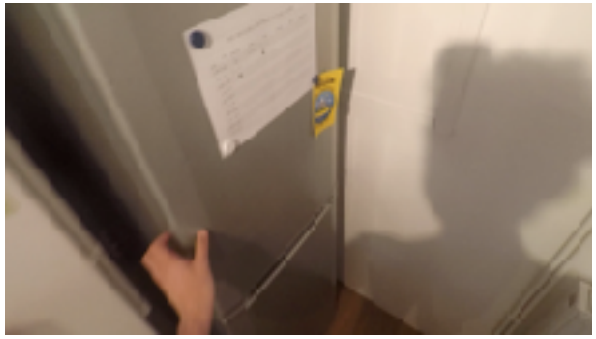
Christoph Feichtenhofer, Haoqi Fan, Bo Xiong,
Ross Girshick, Kaiming He

Problem Overview

- We want to learn a video representation that generalizes to different downstream video tasks (e.g., action recognition).
- We want to do that **without using any labeled data**.

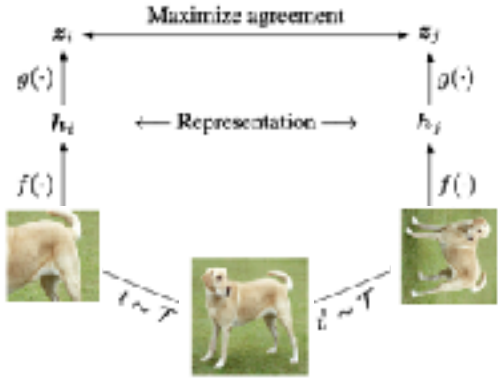
Cartwheeling

Braiding Hair

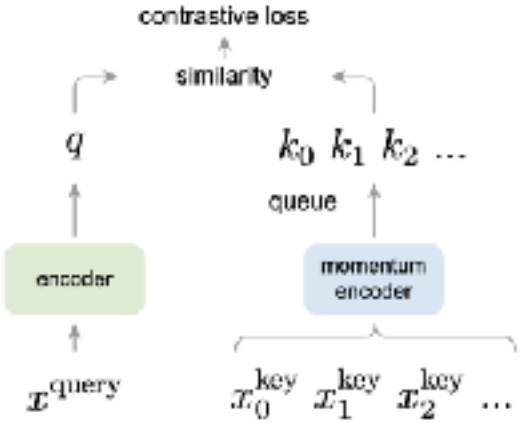


Opening a Fridge

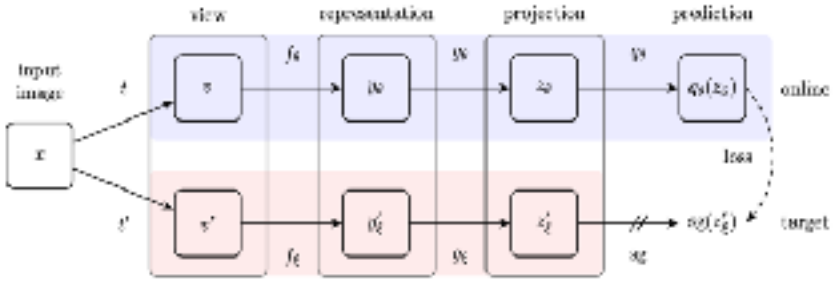
Self-Supervised Learning in Images



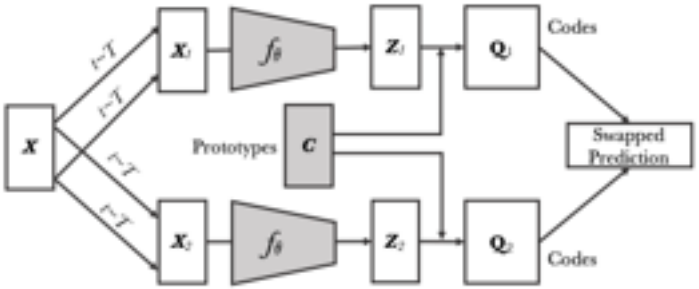
SimCLR [ICML 2020]



MoCo [CVPR 2020]

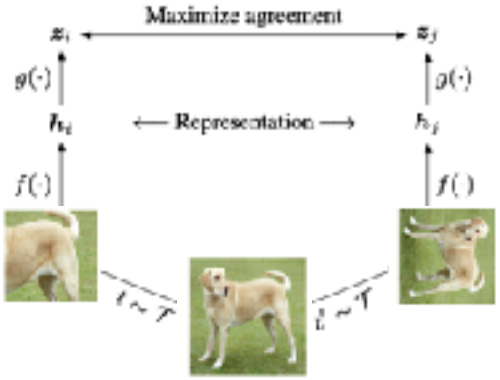


BYOL [NeurIPS 2020]

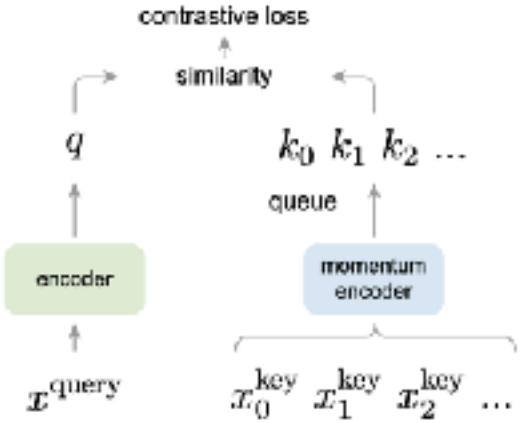


SwAV [NeurIPS 2020]

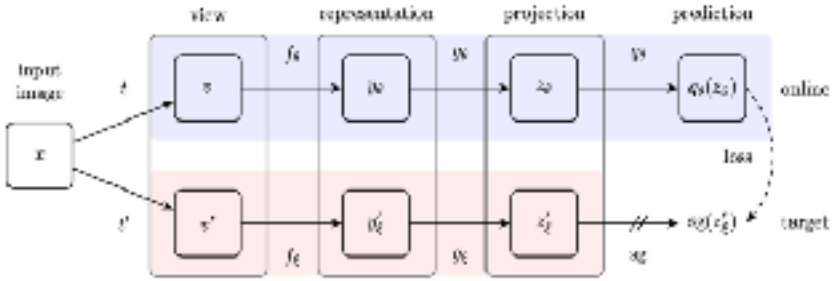
Self-Supervised Learning in Images



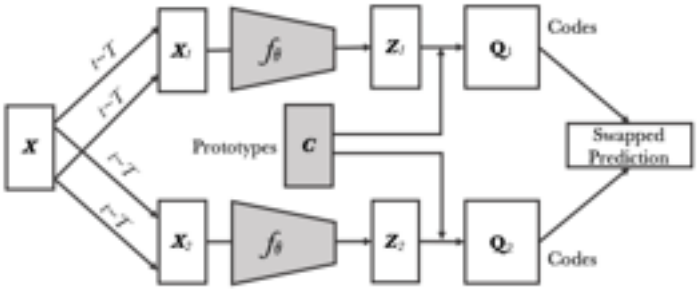
SimCLR [ICML 2020]



MoCo [CVPR 2020]



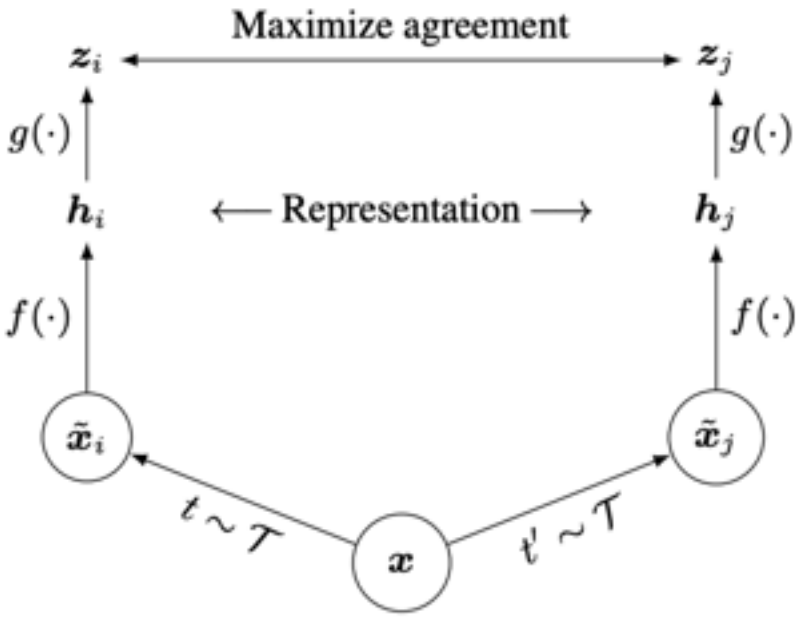
BYOL [NeurIPS 2020]



SwAV [NeurIPS 2020]

SimCLR

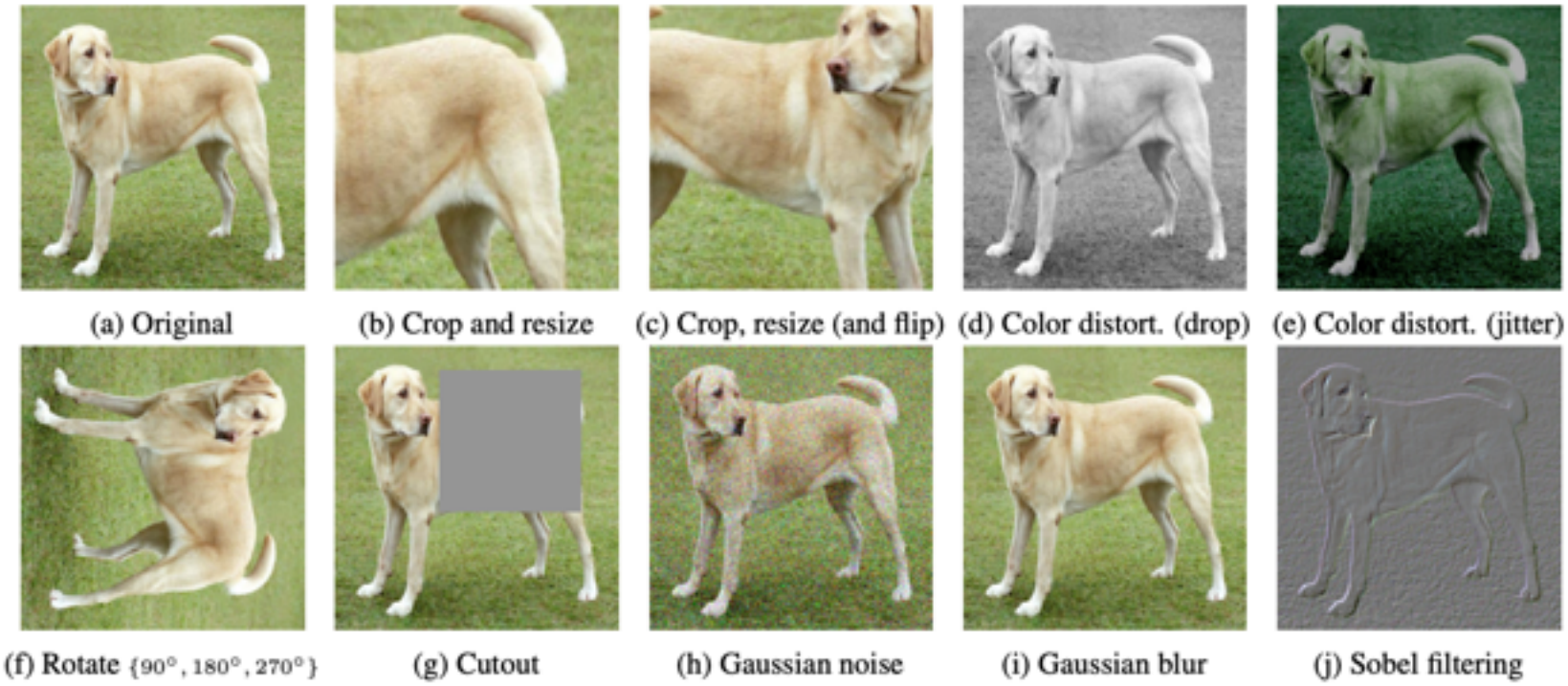
- Two separate data augmentation operators are applied to each data example to obtain two correlated views.
- A base encoder network f(·) and a projection head g(·) are trained to maximize agreement using a contrastive loss.



Chen et al. “A Simple Framework for Contrastive Learning of Visual Representations”, ICML 2020

Data Augmentation

- The authors use a variety of different data augmentation operators.



Contrastive Loss

- The loss function for a positive pair of examples (i, j) is defined using the formula below.
- The model is forced to learn similar representations for positive examples, and dissimilar representations for negative examples.

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)}$$

Contrastive Loss

- The loss function for a positive pair of examples (i, j) is defined using the formula below.
- The model is forced to learn similar representations for positive examples, and dissimilar representations for negative examples.

Probability of two correlated views matching with each other

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)}$$

Algorithm

Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}, t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks f and g to minimize $\mathcal{L}$
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

Algorithm

Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}, t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks f and g to minimize $\mathcal{L}$
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

Algorithm

Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}, t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks f and g to minimize $\mathcal{L}$
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

Algorithm

Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}, t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks f and g to minimize $\mathcal{L}$
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

Algorithm

Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{x_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}, t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{x}_{2k-1} = t(x_k)$
    $h_{2k-1} = f(\tilde{x}_{2k-1})$ # representation
    $z_{2k-1} = g(h_{2k-1})$ # projection
    # the second augmentation
    $\tilde{x}_{2k} = t'(x_k)$
    $h_{2k} = f(\tilde{x}_{2k})$ # representation
    $z_{2k} = g(h_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = z_i^\top z_j / (\|z_i\| \|z_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks f and g to minimize $\mathcal{L}$
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

Algorithm

Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}, t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks f and g to minimize $\mathcal{L}$
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

Algorithm

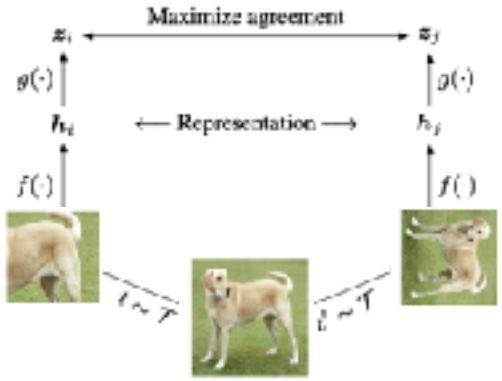
Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of $f, g, \mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}, t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks f and g to minimize $\mathcal{L}$
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

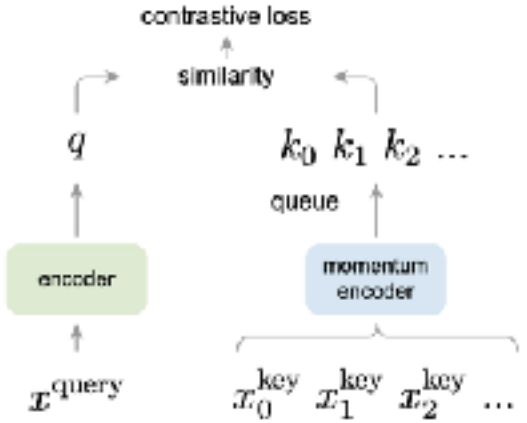
Key Takeaways

- Unsupervised contrastive learning benefits from stronger data augmentation than supervised learning.
- Composition of multiple data augmentation operations is crucial in defining the contrastive prediction tasks that yield effective representations.
- Contrastive learning benefits from larger batch sizes and longer training compared to its supervised counterpart.

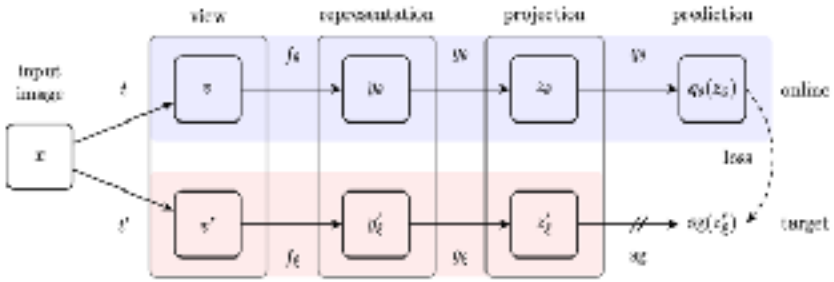
Self-Supervised Learning in Images



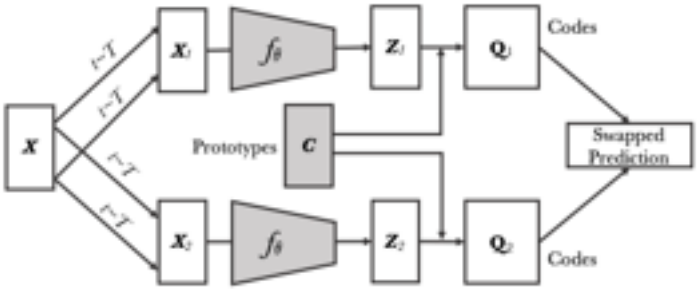
SimCLR [ICML 2020]



MoCo [CVPR 2020]



BYOL [NeurIPS 2020]



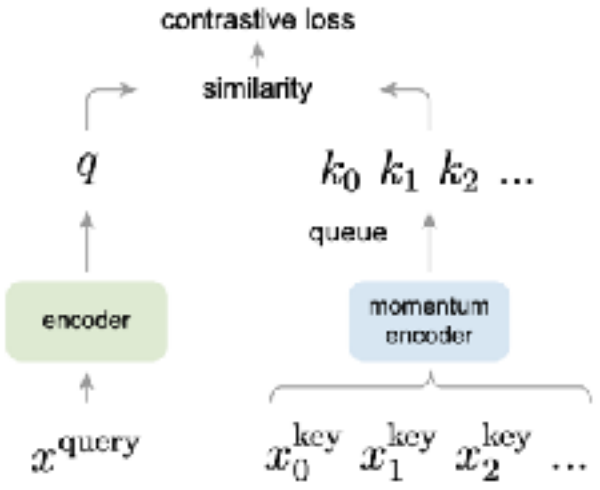
SwAV [NeurIPS 2020]

Motivation

- End-to-end contrastive learning requires a large number of negative samples, which causes GPU memory issues.
- Using a large number of negative examples extracted offline is suboptimal because they cannot be optimized end-to-end.
- **Bottom Line:** a set of negative examples has to be (1) large, and (2) consistent as the examples evolve during training.

MoCo

- Momentum Contrast (MoCo) trains a visual representation encoder by matching an encoded query q to a dictionary of encoded keys
- The keys are stored in a queue, with the current mini-batch enqueued and the oldest mini-batch dequeued.



He et al. “Momentum Contrast for Unsupervised Visual Representation Learning”, CVPR 2020

Momentum Encoder

- The momentum update makes θ_k evolve more smoothly than θ_q.
- As a result, even though the keys in the queue are encoded by different encoders (in different mini-batches), the difference among these encoders can be made small.

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

Momentum Encoder

- The momentum update makes θ_k evolve more smoothly than θ_q.
- As a result, even though the keys in the queue are encoded by different encoders (in different mini-batches), the difference among these encoders can be made small.

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

Parameters of the keys encoder

Momentum Encoder

- The momentum update makes θ_k evolve more smoothly than θ_q.
- As a result, even though the keys in the queue are encoded by different encoders (in different mini-batches), the difference among these encoders can be made small.

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

Parameters of the query encoder (end-to-end trainable)

Momentum Encoder

- The momentum update makes θ_k evolve more smoothly than θ_q.
- As a result, even though the keys in the queue are encoded by different encoders (in different mini-batches), the difference among these encoders can be made small.

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

momentum coefficient

Algorithm

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
```

Algorithm

2 distinct data augmentation functions.

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
```

Algorithm

Feeding the samples through query and key encoders.

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
```

Algorithm

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
```

Computing similarity between positive and negative examples

Algorithm

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
```

Contrastive loss function

Algorithm

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
```

Bckpropagation update of the query encoder

Algorithm

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
```

Momentum update of the key encoder

Algorithm

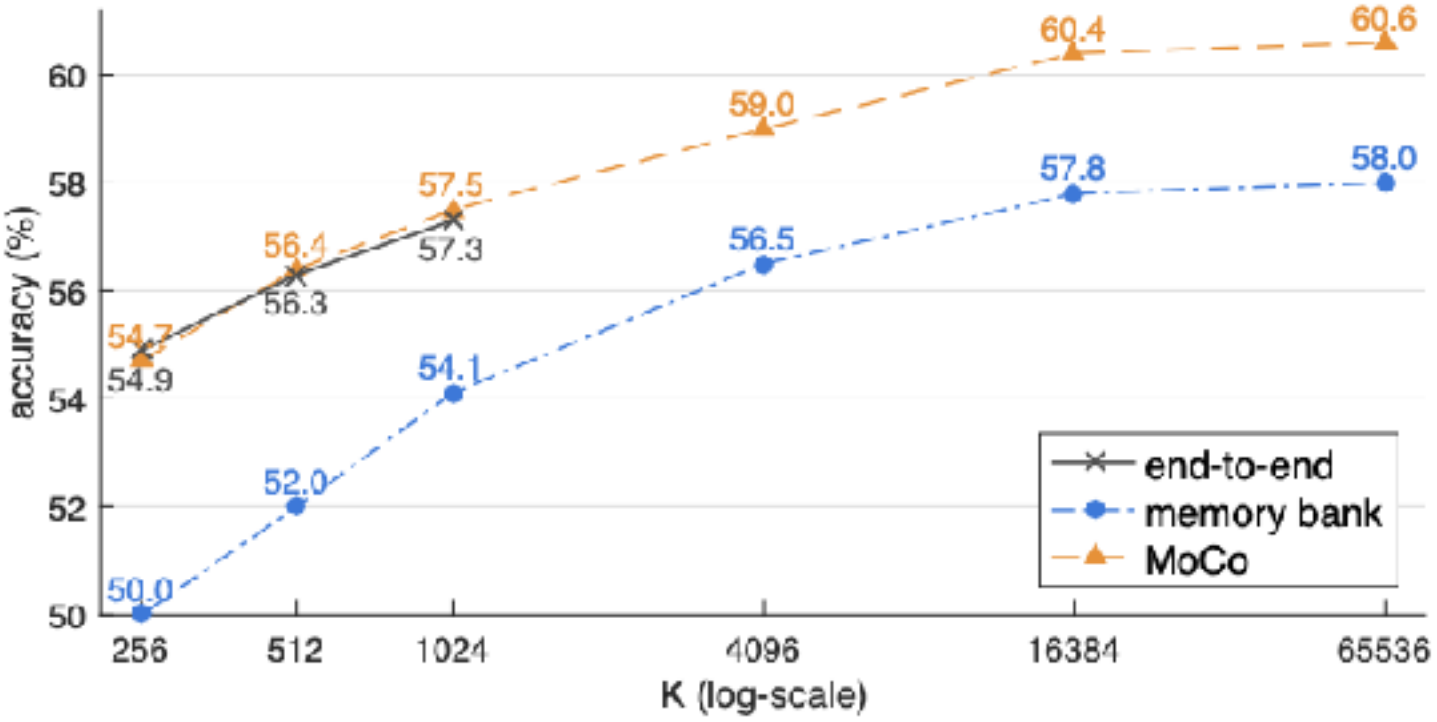
Algorithm 1 Pseudocode of MoCo in a PyTorch-like style.

```
# 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
```

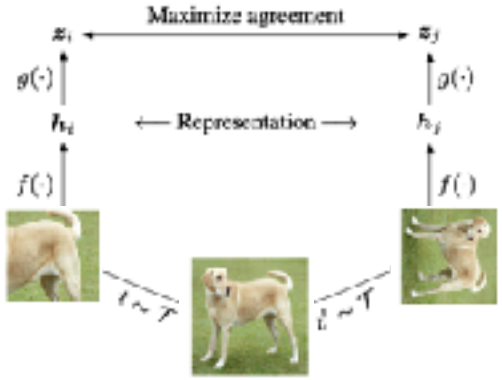
Update the queue storing the keys

Key Takeaways

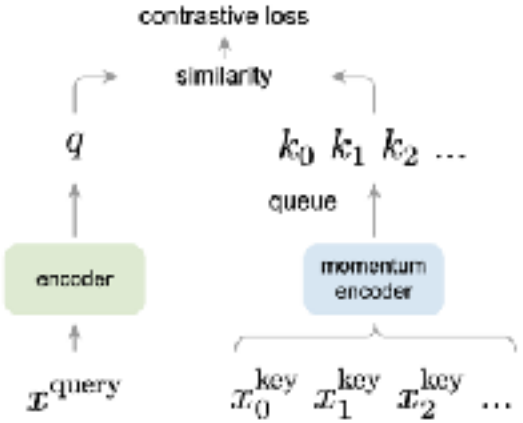
- Having a large number of negative examples is essential.
- End-to-end trainability is also critical to good performance.



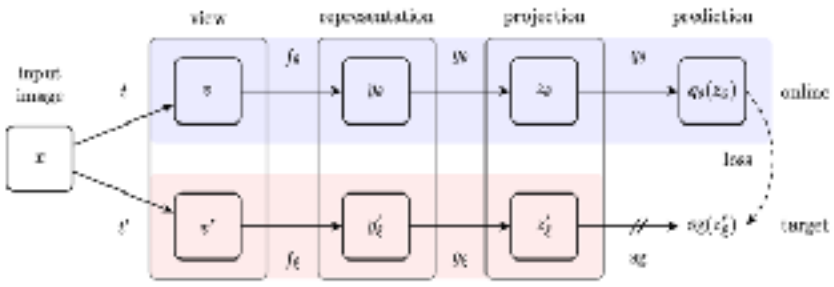
Self-Supervised Learning in Images



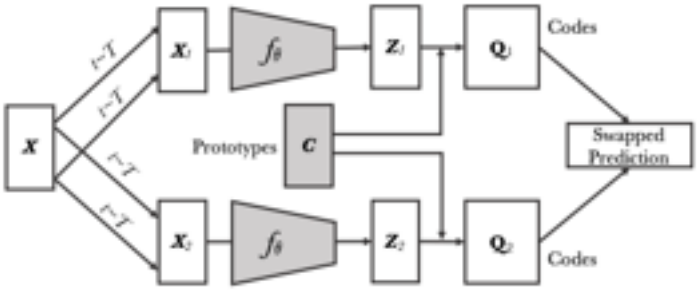
SimCLR [ICML 2020]



MoCo [CVPR 2020]



BYOL [NeurIPS 2020]



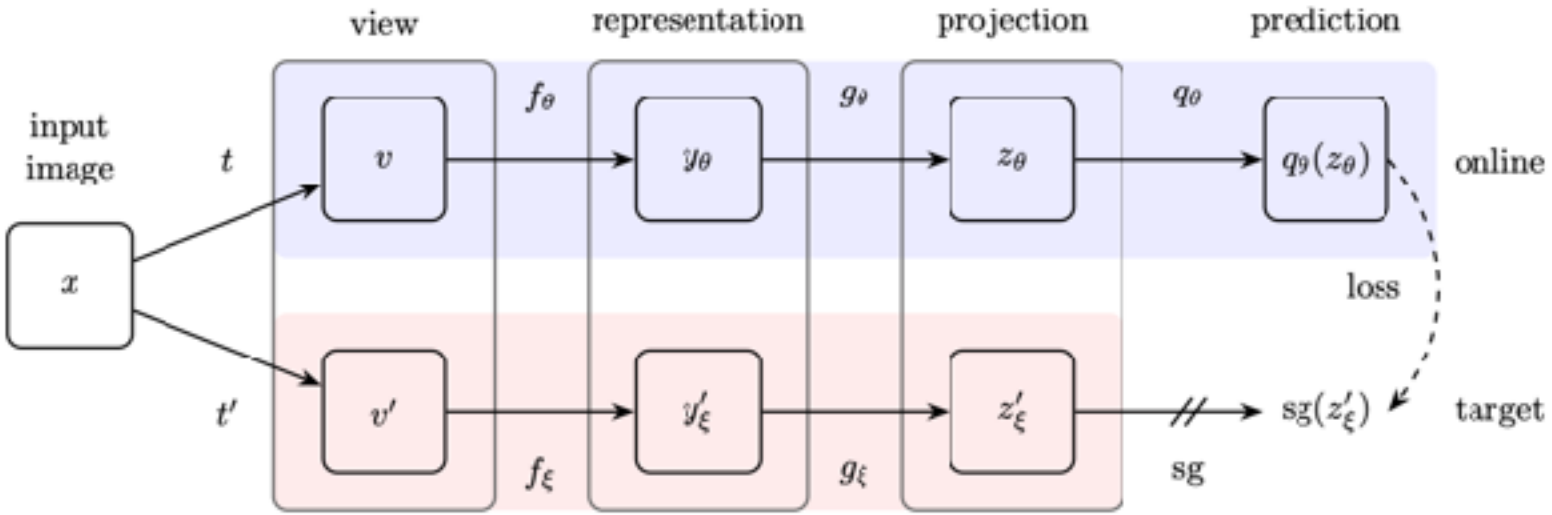
SwAV [NeurIPS 2020]

Motivation

- Prior approaches require using a large set of negative examples.
- Inconvenient due to GPU memory issues.
- Complicated techniques (e.g., MoCo) can be used to mitigate the issue of negative samples.
- **Goal:** we want a simple model that achieves good performance **without relying on negative samples.**

BYOL

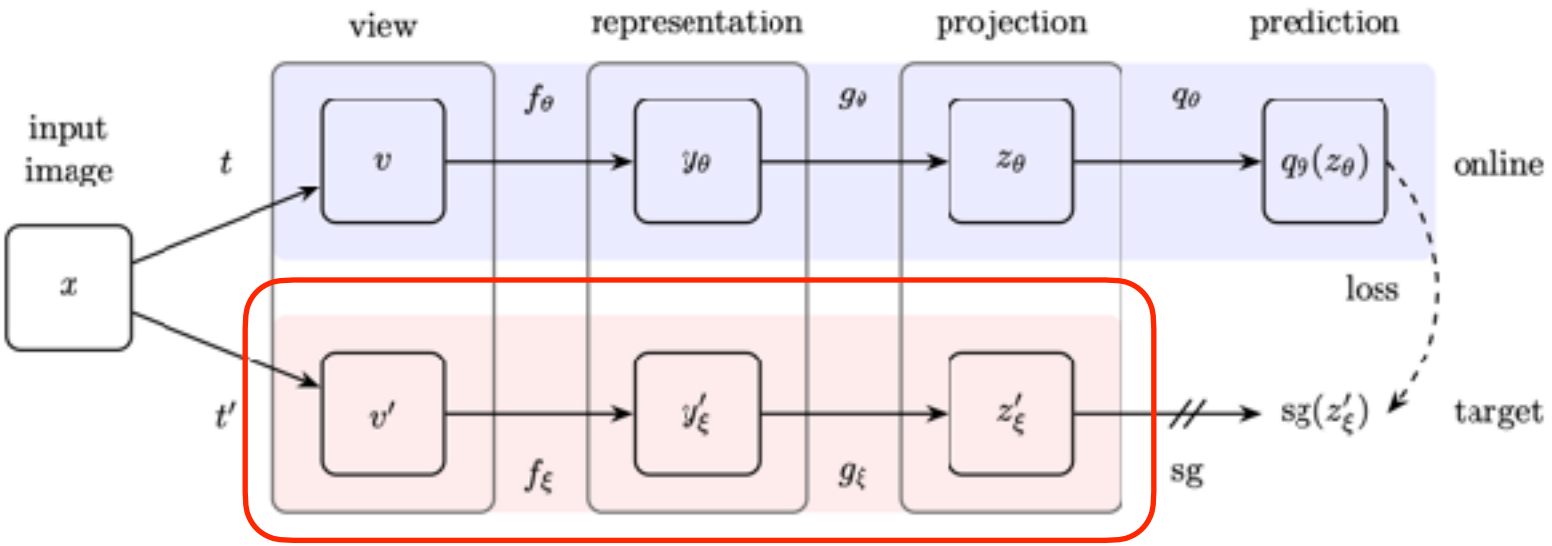
- BYOL minimizes a similarity loss between an online and target networks.
- The authors use L2 loss to optimize their network, which eliminates the need for negative samples.



Grill et al. “Bootstrap Your Own Latent A New Approach to Self-Supervised Learning”, NeurIPS 2020

Target Network

- The target network has the same architecture as the online network, but uses a different set of weights.
- The target network provides the regression targets to the online network.
- Its parameters are computed using a momentum update.



Target Network

- The target network has the same architecture as the online network, but uses a different set of weights.
- The target network provides the regression targets to the online network.
- Its parameters are computed using a momentum update.

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

Target Network

- The target network has the same architecture as the online network, but uses a different set of weights.
- The target network provides the regression targets to the online network.
- Its parameters are computed using a momentum update.

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

Parameters of the target network

Target Network

- The target network has the same architecture as the online network, but uses a different set of weights.
- The target network provides the regression targets to the online network.
- Its parameters are computed using a momentum update.

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

Parameters of the online network

Target Network

- The target network has the same architecture as the online network, but uses a different set of weights.
- The target network provides the regression targets to the online network.
- Its parameters are computed using a momentum update.

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

Target decay rate

Loss Function

- Mean squared error is computed between the normalized predictions and target projections.
- The gradients are only backpropagated through the online network (not the target network).

$$\mathcal{L}_{\theta,\xi} \triangleq \left\| \overline{q_\theta}(z_\theta) - \overline{z}'_\xi \right\|_2^2$$

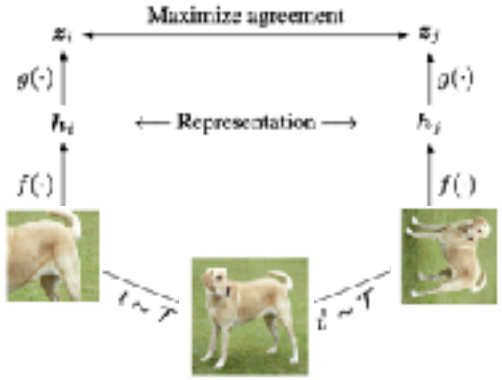
Normalized online network prediction

Normalized target network prediction

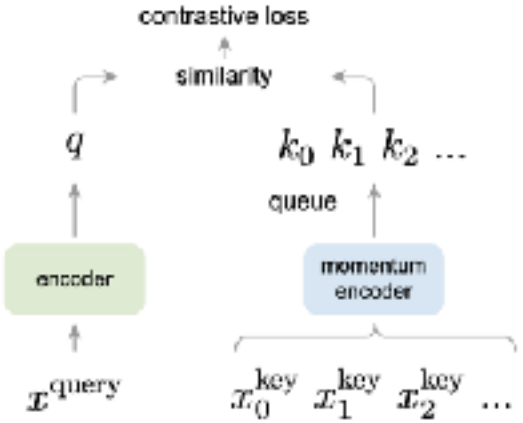
Key Takeaways

- BYOL does not use an explicit term to prevent collapse (e.g., negative examples).
- Outperforms MoCo and SimCLR by >3%.

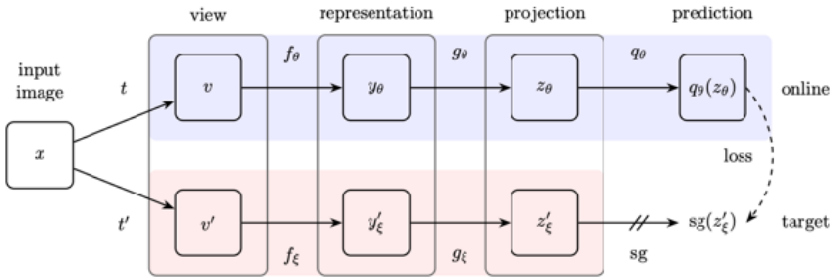
Self-Supervised Learning in Images



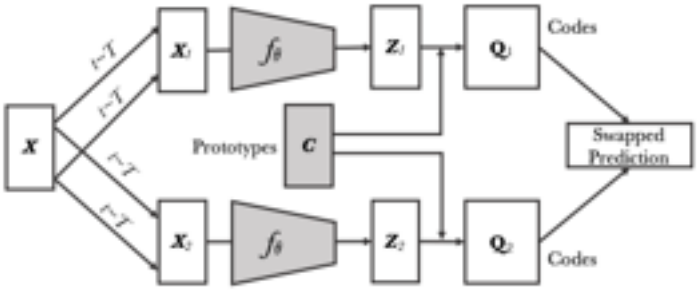
SimCLR [ICML 2020]



MoCo [CVPR 2020]



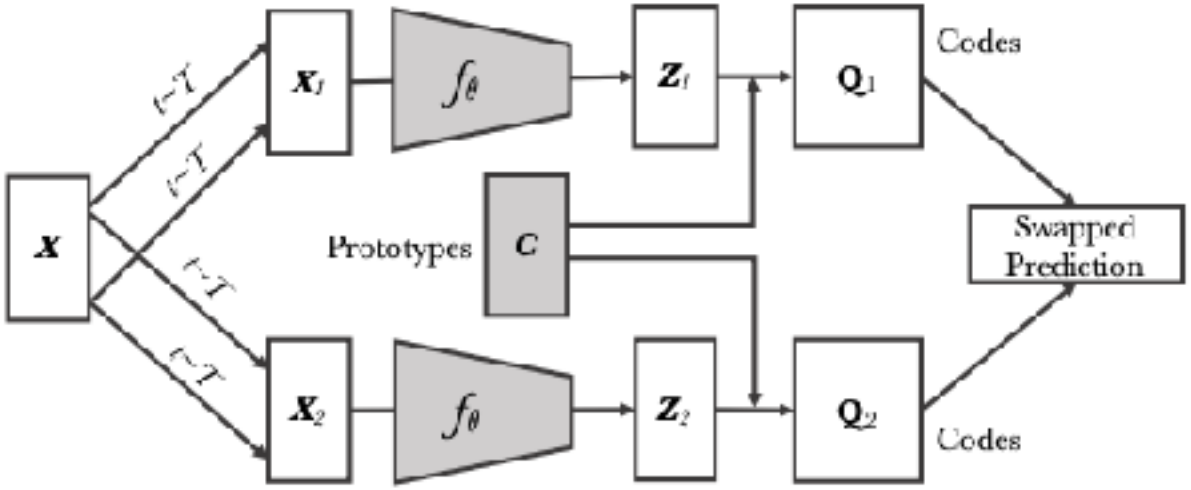
BYOL [NeurIPS 2020]



SwAV [NeurIPS 2020]

SwAV

- The authors first obtain “codes” by assigning features to prototype vectors.
- They then solve a “swapped” prediction problem wherein the codes obtained from one data augmented view are predicted using the other view.



Caron et al. “Unsupervised Learning of Visual Features by Contrasting Cluster Assignments”, NeurIPS 2020

Loss Function

- The codes q_t and q_s are computed by matching these features to a set of K prototypes {c_1, . . . , c_K}.
- The authors then setup a “swapped” prediction problem with the following loss function:

$$L(\mathbf{z}_t, \mathbf{z}_s) \quad = \quad \ell(\mathbf{z}_t, \mathbf{q}_s) + \ell(\mathbf{z}_s, \mathbf{q}_t),$$

Loss Function

- The codes q_t and q_s are computed by matching these features to a set of K prototypes {c_1, . . . , c_K}.
- The authors then setup a “swapped” prediction problem with the following loss function:

$$L(\mathbf{z}_t, \mathbf{z}_s) = \ell(\mathbf{z}_t, \mathbf{q}_s) + \ell(\mathbf{z}_s, \mathbf{q}_t),$$

Feature for view t

Feature for view s

Loss Function

- The codes q_t and q_s are computed by matching these features to a set of K prototypes {c_1, . . . , c_K}.
- The authors then setup a “swapped” prediction problem with the following loss function:

$$L(\mathbf{z}_t, \mathbf{z}_s) = \ell(\mathbf{z}_t, \mathbf{q}_s) + \ell(\mathbf{z}_s, \mathbf{q}_t),$$

Code for view s

Code for view t

Loss Function

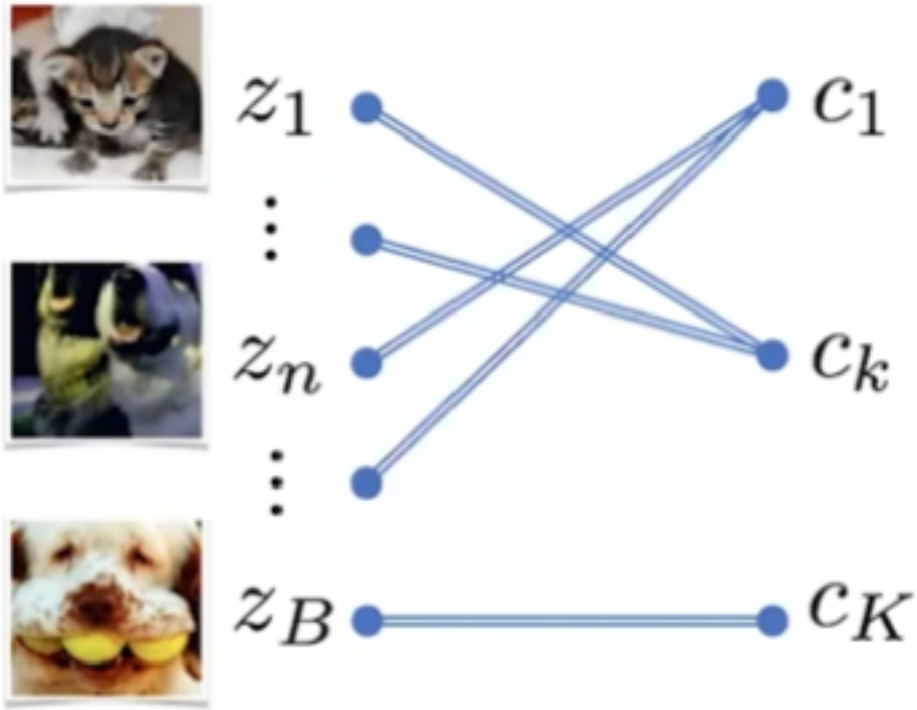
- The codes q_t and q_s are computed by matching these features to a set of K prototypes {c_1, . . . , c_K}.
- The authors then setup a “swapped” prediction problem with the following loss function:

$$L(\mathbf{z}_t, \mathbf{z}_s) \quad = \quad \ell(\mathbf{z}_t, \mathbf{q}_s) + \ell(\mathbf{z}_s, \mathbf{q}_t),$$

$$\ell(\mathbf{z}_t, \mathbf{q}_s) = -\sum_k \mathbf{q}_s^{(k)} \log \mathbf{p}_t^{(k)}, \quad \text{where} \quad \mathbf{p}_t^{(k)} = \frac{\exp\left(\frac{1}{\tau}\mathbf{z}_t^\top \mathbf{c}_k\right)}{\sum_{k'} \exp\left(\frac{1}{\tau}\mathbf{z}_t^\top \mathbf{c}_{k'}\right)}.$$

Online Clustering

- The goal is to map *B* samples to *K* prototypes.
- The codes are computed using only the image features within a batch.

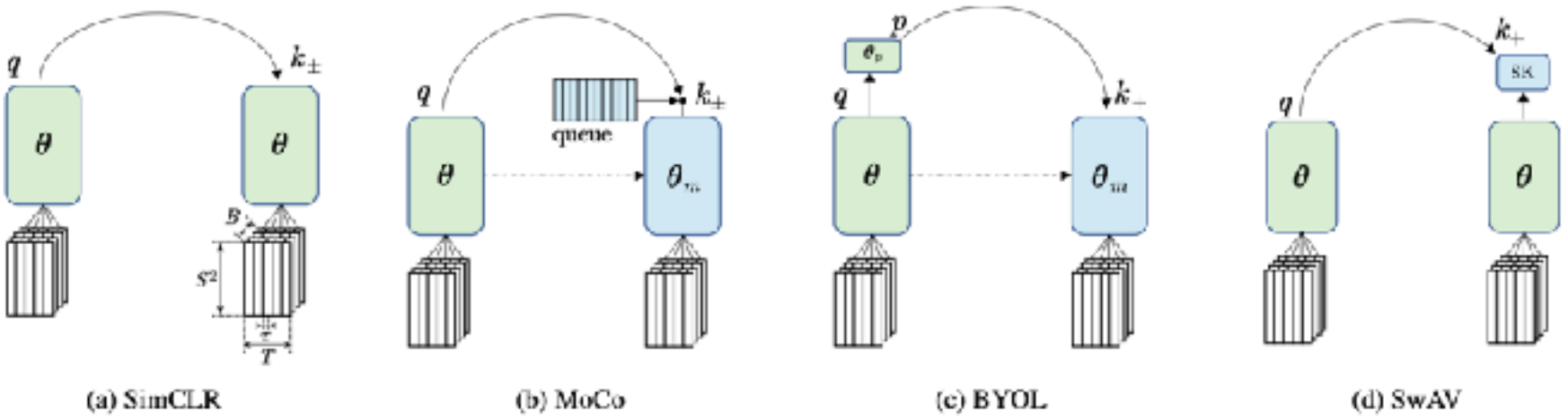


Key Takeaways

- Avoids relying on negative samples via an online clustering step.
- Performs better than the previous three proposed methods.
- Unlike other methods, requires more tricks to obtain good performance (e.g., multi-crop, etc.).

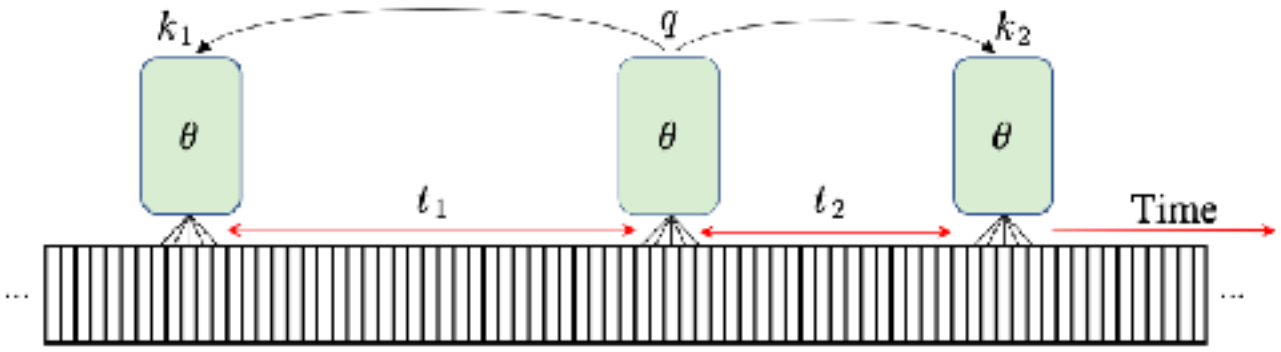
Adapting These Methods to Video

- The inputs consist of ρ=2 clips from each video.
- Each clip is a stack of T frames.
- Each method trains encoder weights θ by computing a positive loss component w.r.t. to the other clips of the same video



Persistent Temporal Feature Learning

- Learning to maximize the similarity between different temporal clips of the same video encourages feature persistency over time.
- A query clip (q) is matched to multiple key clips (k1, k2, …) that are temporally shifted.



Pretraining Datasets

- Kinetics-400 (K400) consisting of ~240k training videos in 400 human action categories.
- 1M of Instagram videos (IG-Curated), a dataset with hashtags similar to K400 classes;
- IG-Uncurated, which has 1M videos taken randomly from Instagram.
- IG-Uncurated-Short which is similar, but has constrained duration.

data	#videos	t_{median}	t_{mean}	t_{std}	t_{min}	t_{max}
Kinetics-400 (K400) [47]	240K	10.0	9.3	1.7	1.0	10.0
IG-Curated [24]	1M	18.9	26.3	19.8	1.5	60.0
IG-Uncurated	1M	29.4	35.3	38.4	0.5	600.0
IG-Uncurated-Short	1M	13.0	13.1	1.6	10.0	15.9

Evaluation Protocols

- The first protocol involves training a linear classifier on frozen encoder features on the K400 validation set
- The second protocol reports full finetuning accuracy on the UCF101 dataset.
- In both cases, the performance is evaluated using top-1 classification accuracy.

Temporal Augmentation

- Using clips at different temporal locations as positives (i.e., p>1) leads to a significant boost in performance.
- Using more temporal clips is beneficial.

	MoCo		BYOL		SimCLR		SwAV	
ρ	K400	UCF101	K400	UCF101	K400	UCF101	K400	UCF101
1	61.0	90.8	60.6	91.2	36.1	84.2	38.6	74.7
2	65.8	91.0	65.8	92.7	60.5	88.9	61.6	87.3
3	67.3	92.8	68.3	93.8	62.0	87.9	62.7	89.4
4	67.8	93.5	**68.9**	**93.8**	out of memory			

Temporal Augmentation

- Using clips at different temporal locations as positives (i.e., p>1) leads to a significant boost in performance.
- Using more temporal clips is beneficial.

	MoCo		BYOL		SimCLR		SwAV	
ρ	K400	UCF101	K400	UCF101	K400	UCF101	K400	UCF101
1	61.0	90.8	60.6	91.2	36.1	84.2	38.6	74.7
2	65.8	91.0	65.8	92.7	60.5	88.9	61.6	87.3
3	67.3	92.8	68.3	93.8	62.0	87.9	62.7	89.4
4	67.8	93.5	**68.9**	**93.8**	out of memory			

Persistent feature learning in time has a large impact on performance especially for SimCLR and SwAV.

Temporal Augmentation

- Using clips at different temporal locations as positives (i.e., p>1) leads to a significant boost in performance.
- Using more temporal clips is beneficial.

	MoCo		BYOL		SimCLR		SwAV	
ρ	K400	UCF101	K400	UCF101	K400	UCF101	K400	UCF101
1	61.0	90.8	60.6	91.2	36.1	84.2	38.6	74.7
2	65.8	91.0	65.8	92.7	60.5	88.9	61.6	87.3
3	67.3	92.8	68.3	93.8	62.0	87.9	62.7	89.4
4	67.8	93.5	**68.9**	**93.8**	out of memory			

There is no clear performance difference between contrastive (MoCo, SimCLR) vs non-contrastive (BYOL, SwAV) methods.

Temporal Augmentation

- Using clips at different temporal locations as positives (i.e., p>1) leads to a significant boost in performance.
- Using more temporal clips is beneficial.

	MoCo		BYOL		SimCLR		SwAV	
ρ	K400	UCF101	K400	UCF101	K400	UCF101	K400	UCF101
1	61.0	90.8	60.6	91.2	36.1	84.2	38.6	74.7
2	65.8	91.0	65.8	92.7	60.5	88.9	61.6	87.3
3	67.3	92.8	68.3	93.8	62.0	87.9	62.7	89.4
4	67.8	93.5	**68.9**	**93.8**	out of memory			

There is a clear difference between MoCo, BYOL vs. SimCLR, SwAV.

Training Duration

- Training for longer is beneficial.

	MoCo		BYOL		SimCLR		SwAV	
ep	K400	UCF101	K400	UCF101	K400	UCF101	K400	UCF101
50	52.6	84.6	30.2	78.5	45.7	79.7	55.9	81.4
100	60.5	89.5	47.6	88.6	57.3	85.6	59.4	85.5
200	65.8	91.0	65.8	92.7	60.5	88.9	61.6	87.3
400	67.4	92.5	66.9	92.8	62.0	87.9	62.9	88.3
800	**67.4**	**93.2**	66.2	93.6	61.8	88.4	63.2	89.5

Training Duration

- Training for longer is beneficial.

	MoCo		BYOL		SimCLR		SwAV	
ep	K400	UCF101	K400	UCF101	K400	UCF101	K400	UCF101
50	52.6	84.6	30.2	78.5	45.7	79.7	55.9	81.4
100	60.5	89.5	47.6	88.6	57.3	85.6	59.4	85.5
200	65.8	91.0	65.8	92.7	60.5	88.9	61.6	87.3
400	67.4	92.5	66.9	92.8	62.0	87.9	62.9	88.3
800	**67.4**	**93.2**	66.2	93.6	61.8	88.4	63.2	89.5

Training for longer brings huge improvements in accuracy.

Training Duration

- Training for longer is beneficial.

	MoCo		BYOL		SimCLR		SwAV	
ep	K400	UCF101	K400	UCF101	K400	UCF101	K400	UCF101
50	52.6	84.6	30.2	78.5	45.7	79.7	55.9	81.4
100	60.5	89.5	47.6	88.6	57.3	85.6	59.4	85.5
200	65.8	91.0	65.8	92.7	60.5	88.9	61.6	87.3
400	67.4	92.5	66.9	92.8	62.0	87.9	62.9	88.3
800	**67.4**	**93.2**	66.2	93.6	61.8	88.4	63.2	89.5

~1 week of training time using 128 V100 GPUs

Training on Uncurated Videos

- Training on curated vs uncurated Instagram Videos

	MoCo		BYOL		SimCLR		SwAV	
ep	K400	UCF101	K400	UCF101	K400	UCF101	K400	UCF101
50	64.8	91.1	64.1	93.5	55.5	86.4	61.0	89.0
200	69.0	93.4	60.2	92.7	56.9	86.6	64.3	91.2

(a) Training on **IG-Curated-1M**.

	MoCo		BYOL		SimCLR		SwAV	
ep	K400	UCF101	K400	UCF101	K400	UCF101	K400	UCF101
50	61.8	90.9	58.9	90.1	52.1	85.1	56.0	86.7
200	65.4	91.9	57.9	91.6	51.9	85.3	58.8	87.8

(b) Training on **IG-Uncurated-1M**.

Training on Uncurated Videos

- Training on curated vs uncurated Instagram Videos

	MoCo		BYOL		SimCLR		SwAV	
ep	K400	UCF101	K400	UCF101	K400	UCF101	K400	UCF101
50	64.8	91.1	64.1	93.5	55.5	86.4	61.0	89.0
200	69.0	93.4	60.2	92.7	56.9	86.6	64.3	91.2

(a) Training on **IG-Curated-1M**.

	MoCo		BYOL		SimCLR		SwAV	
ep	K400	UCF101	K400	UCF101	K400	UCF101	K400	UCF101
50	61.8	90.9	58.9	90.1	52.1	85.1	56.0	86.7
200	65.4	91.9	57.9	91.6	51.9	85.3	58.8	87.8

(b) Training on **IG-Uncurated-1M**.

Training on curated IG data leads to substantially better results.

Training on Uncurated Videos

- Training on curated vs uncurated Instagram Videos

	MoCo		BYOL		SimCLR		SwAV	
ep	K400	UCF101	K400	UCF101	K400	UCF101	K400	UCF101
50	64.8	91.1	64.1	93.5	55.5	86.4	61.0	89.0
200	69.0	93.4	60.2	92.7	56.9	86.6	64.3	91.2

(a) Training on **IG-Curated-1M**.

	MoCo		BYOL		SimCLR		SwAV	
ep	K400	UCF101	K400	UCF101	K400	UCF101	K400	UCF101
50	61.8	90.9	58.9	90.1	52.1	85.1	56.0	86.7
200	65.4	91.9	57.9	91.6	51.9	85.3	58.8	87.8

(b) Training on **IG-Uncurated-1M**.

MoCo performs the best overall on IG (but not on Kinetics)

Comparison to Supervised Baselines

- On larger-scale benchmarks, supervised baselines outperform the self-supervised ones by a large margin.

		linear protocol	*finetuning accuracy*			
method	pre-train	K400	UCF101	AVA (mAP)	Charades (mAP)	SSv2
supervised	*scratch*	**74.7**	68.8	11.7	7.4	48.8
supervised	K400-240K	-	**94.8**	22.2	**34.7**	52.8
SimCLR	K400-240K	62.0 (−12.7)	87.9 (−6.9)	17.6 (−4.6)	11.4 (−23.3)	52.0 (−0.8)
SwAV		62.7 (−11.5)	89.4 (−5.4)	18.2 (−4.0)	10.7 (−24.0)	51.7 (−1.1)
BYOL		68.3 (−6.4)	93.8 (−1.0)	**23.4 (+1.2)**	21.0 (−13.7)	**55.8 (+3.0)**
MoCo		67.3 (−7.4)	92.8 (−2.0)	20.3 (−1.9)	33.5 (−1.2)	54.4 (+1.8)
MoCo	IG-Curated-1M	69.9 (−4.8)	**94.4** (−0.4)	20.4 (−1.8)	**34.9 (+0.2)**	54.5 (+1.8)
MoCo	IG-Uncurated-1M	66.0 (−8.7)	92.9 (−2.1)	20.5 (−1.7)	31.3 (−3.4)	53.2 (+0.4)

Comparison to Supervised Baselines

- On larger-scale benchmarks, supervised baselines outperform the self-supervised ones by a large margin.

		linear protocol	*finetuning accuracy*			
method	pre-train	K400	UCF101	AVA (mAP)	Charades (mAP)	SSv2
supervised	*scratch*	**74.7**	68.8	11.7	7.4	48.8
supervised	K400-240K	-	**94.8**	22.2	**34.7**	52.8
SimCLR	K400-240K	62.0 (−12.7)	87.9 (−6.9)	17.6 (−4.6)	11.4 (−23.3)	52.0 (−0.8)
SwAV		62.7 (−11.5)	89.4 (−5.4)	18.2 (−4.0)	10.7 (−24.0)	51.7 (−1.1)
BYOL		68.3 (−6.4)	93.8 (−1.0)	**23.4 (+1.2)**	21.0 (−13.7)	**55.8 (+3.0)**
MoCo		67.3 (−7.4)	92.8 (−2.0)	20.3 (−1.9)	33.5 (−1.2)	54.4 (+1.8)
MoCo	IG-Curated-1M	69.9 (−4.8)	**94.4** (−0.4)	20.4 (−1.8)	**34.9 (+0.2)**	54.5 (+1.8)
MoCo	IG-Uncurated-1M	66.0 (−8.7)	92.9 (−2.1)	20.5 (−1.7)	31.3 (−3.4)	53.2 (+0.4)

Comparison to Supervised Baselines

- On larger-scale benchmarks, supervised baselines outperform the self-supervised ones by a large margin.

method	pre-train	*linear protocol*	*finetuning accuracy*			
		K400	**UCF101**	**AVA (mAP)**	**Charades (mAP)**	**SSv2**
supervised	*scratch*	**74.7**	68.8	11.7	7.4	48.8
supervised	K400-240K	-	**94.8**	22.2	**34.7**	52.8
SimCLR	K400-240K	62.0 (−12.7)	87.9 (−6.9)	17.6 (−4.6)	11.4 (−23.3)	52.0 (−0.8)
SwAV		62.7 (−11.5)	89.4 (−5.4)	18.2 (−4.0)	10.7 (−24.0)	51.7 (−1.1)
BYOL		68.3 (−6.4)	93.8 (−1.0)	**23.4 (+1.2)**	21.0 (−13.7)	**55.8 (+3.0)**
MoCo		67.3 (−7.4)	92.8 (−2.0)	20.3 (−1.9)	33.5 (−1.2)	54.4 (+1.8)
MoCo	IG-Curated-1M	69.9 (−4.8)	**94.4** (−0.4)	20.4 (−1.8)	**34.9 (+0.2)**	54.5 (+1.8)
MoCo	IG-Uncurated-1M	66.0 (−8.7)	92.9 (−2.1)	20.5 (−1.7)	31.3 (−3.4)	53.2 (+0.4)

Comparison to State-of-the-Art

- The best performing approach outperforms all prior state-of-the-art methods.

method	pre-train	backbone	param	T	mod	UCF	HMDB	K400
XDC [3]	K400	R(2+1)D-18	15.4M	32	V+A	84.2	47.1	
GDT [68]	K400	R(2+1)D-18	15.4M	32	V+A	89.3	60.0	
MMV [2]	AS+HT	S3D-G	9.1M	32	V+A+T	92.5	69.6	
SpeedNet [7]	K400	S3D-G	9.1M	64	V	81.1	48.8	
CoCLR [35]	K400	S3D-G	9.1M	32	V	87.9	54.6	
CoCLR [35]	K400	2×S3D-G	9.1M	32	V	90.6	62.9	
VTHCL [92]	K400	R-50	31.8M	8	V	82.1	49.2	37.8
CVRL [71]	K400	R-50	31.8M	32	V	92.2	66.7	66.1
ρBYOL	K400	R-50	31.8M	8	V	**94.2**	**72.1**	**70.0**
ρBYOL	K400	R-50	31.8M	16	V	**95.5**	**73.6**	**71.5**
ρBYOL	K400	R(2+1)D-18	15.4M	32	V	**94.4**	**72.2**	
ρBYOL	K400	S3D-G	9.1M	32	V	**96.3**	**75.0**	

Key Takeaways

- Good empirical paper with no technical novelty.
- Contrastive objectives are less important than momentum encoders.
- Temporal data augmentations are important (but so are spatial ones).
- Long training duration is probably the most critical component.

Discussion Questions

- Why does sampling clips from different temporal locations (i.e., p>1) improve results?

Discussion Questions

- Why does sampling clips from different temporal locations (i.e., p>1) improve results?
- What is the training data that we should use for self-supervised learning?

Discussion Questions

- Why does sampling clips from different temporal locations (i.e., p>1) improve results?
- What is the training data that we should use for self-supervised learning?
- What kind of datasets & tasks we should evaluate self-supervised learning methods on?

Discussion Questions

- Why does sampling clips from different temporal locations (i.e., p>1) improve results?
- What is the training data that we should use for self-supervised learning?
- What kind of datasets & tasks we should evaluate self-supervised learning methods on?
- Will self-supervised methods ever surpass fully-supervised methods?