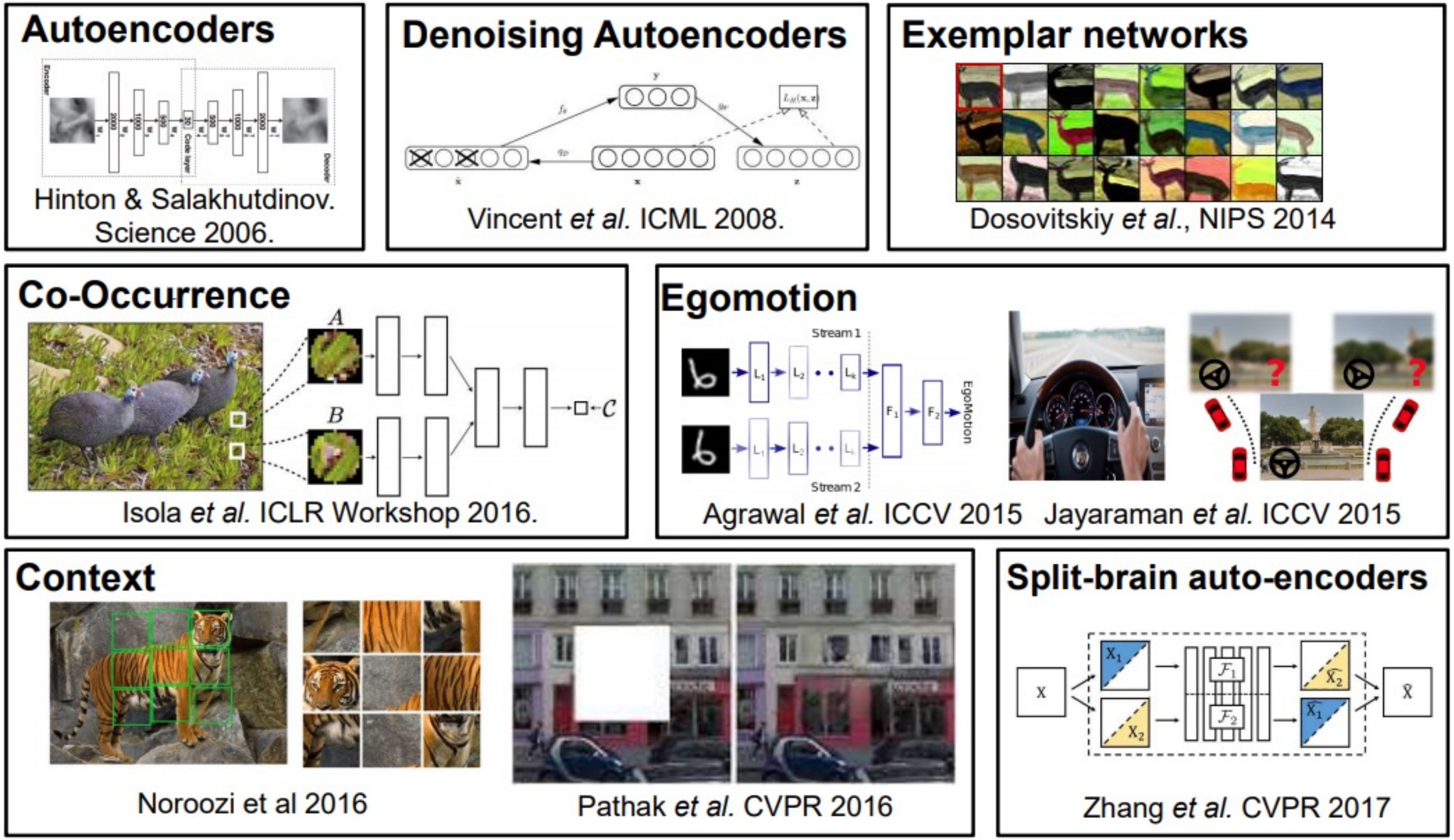
Exploring Simple Siamese Representation Learning

论文解读

https://www.bilibili.com/video/BV12M4y1u7ep

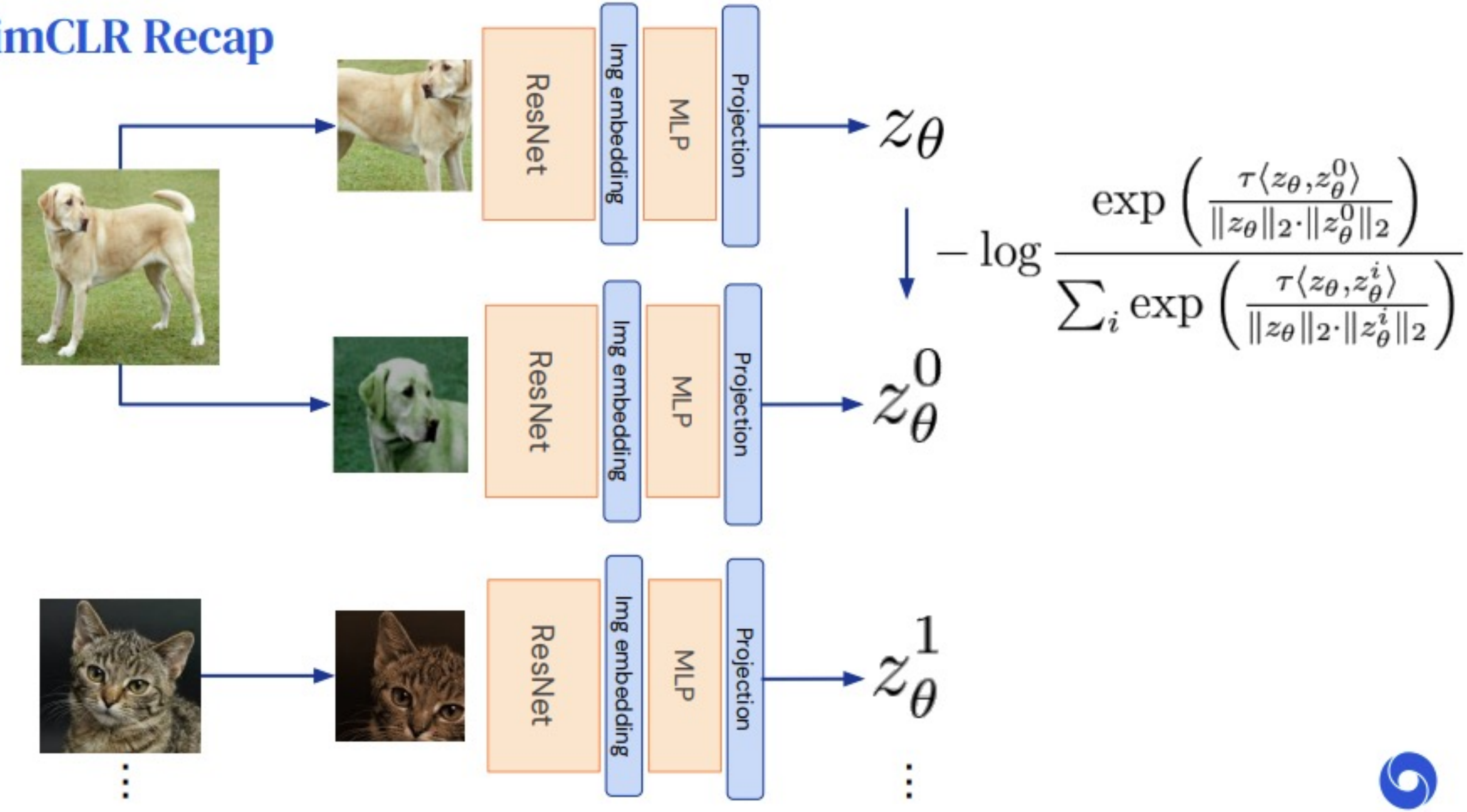
背景

Work before 2018



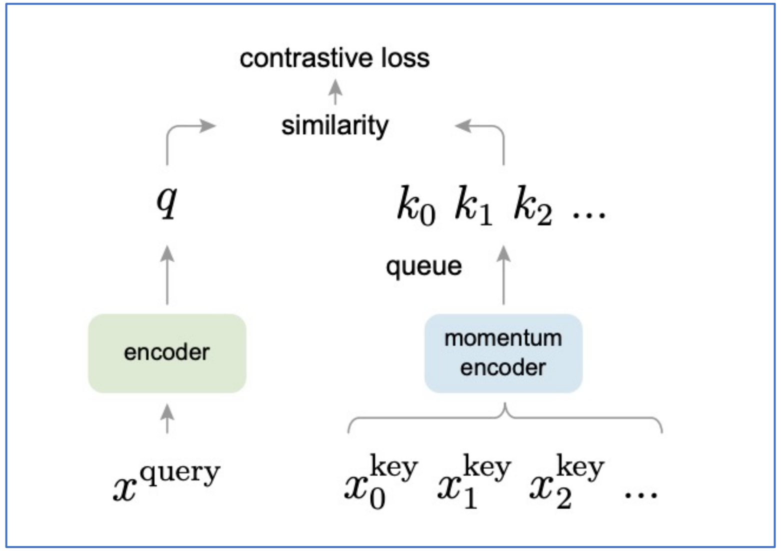
Slides credit: https://project.inria.fr/paiss/files/2018/07/zisserman-self-supervised.pdf

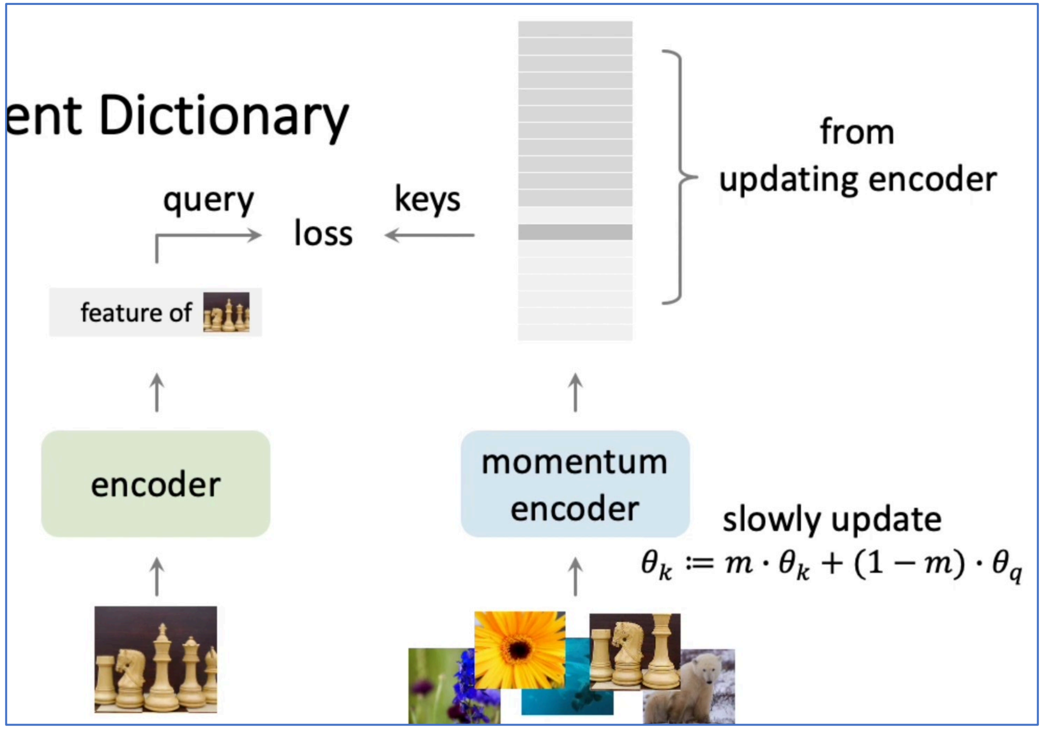
SimCLR Recap



A Simple Framework for Contrastive Learning of Visual Representations, ICML 2020

Moco



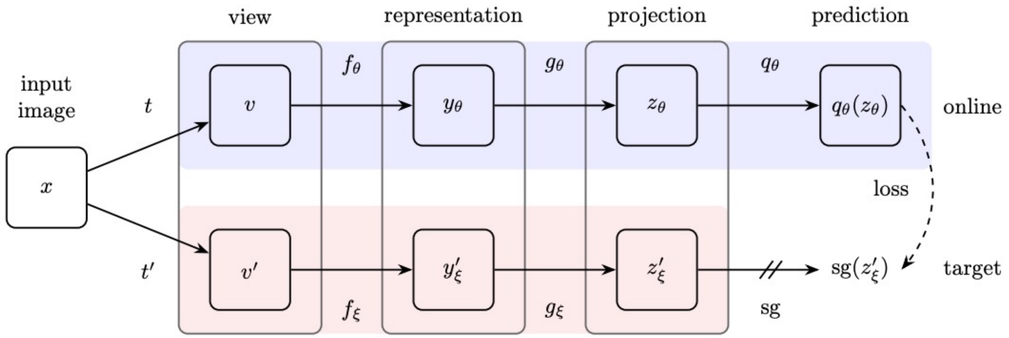


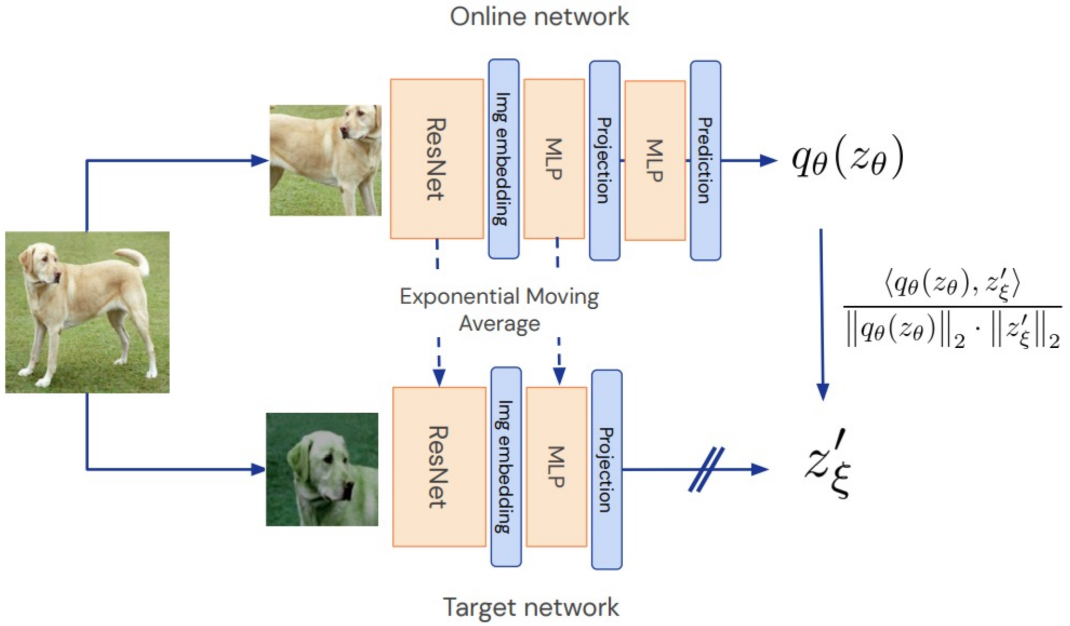
Algorithm 1 Pseudocode of MoCo in a PyTorch-like style.

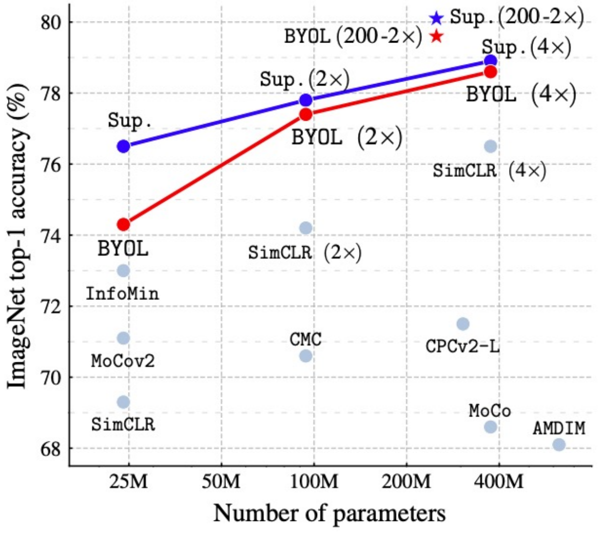
```
# f_q, f_k: encoder networks for query and key
# queue: dictionary as a queue of K keys (CxK)
# m: momentum
# t: temperature

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
```

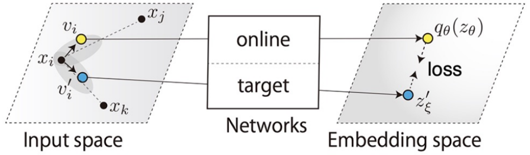
bmm: batch matrix multiplication; mm: matrix multiplication; cat: concatenation.

BYOL









BYOL for Audio: Self-Supervised Learning for General-Purpose Audio Representation

BYOL-A by NTT

Method	Top-1	Top-5
Local Agg.	60.2	-
PIRL [35]	63.6	-
CPC v2 [32]	63.8	85.3
CMC [11]	66.2	87.0
SimCLR [8]	69.3	89.0
MoCo v2 [37]	71.1	-
InfoMin Aug. [12]	73.0	91.1
BYOL (ours)	**74.3**	**91.6**

(a) ResNet-50 encoder.

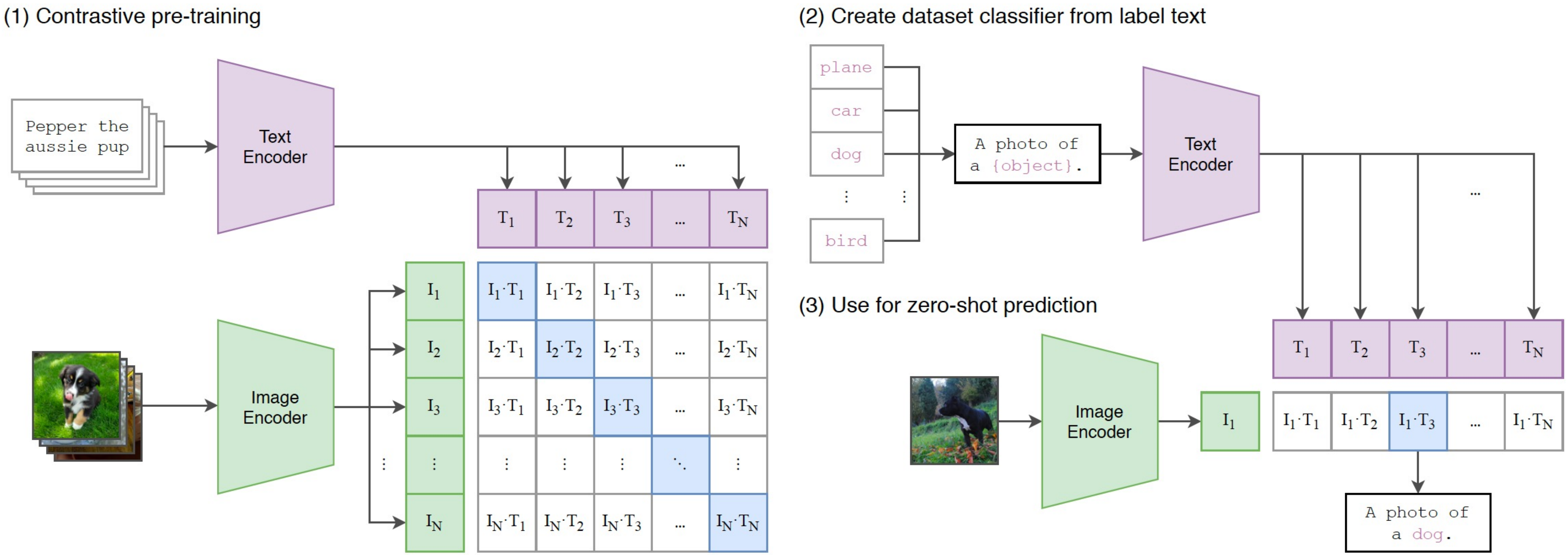
Method	Architecture	Param.	Top-1	Top-5
SimCLR [8]	ResNet-50 (2×)	94M	74.2	92.0
CMC [11]	ResNet-50 (2×)	94M	70.6	89.7
BYOL (ours)	ResNet-50 (2×)	94M	77.4	93.6
CPC v2 [32]	ResNet-161	305M	71.5	90.1
MoCo [9]	ResNet-50 (4×)	375M	68.6	-
SimCLR [8]	ResNet-50 (4×)	375M	76.5	93.2
BYOL (ours)	ResNet-50 (4×)	375M	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.

Bootstrap Your Own Latent: A New Approach to Self-Supervised Learning
https://arxiv.org/pdf/2006.07733.pdf, NurIPS2020

CLIP

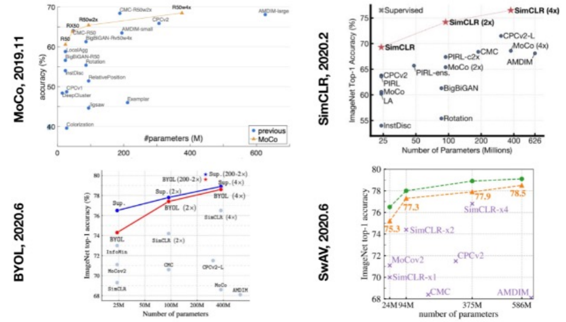

(1) Contrastive pre-training
Pepper the aussie pup
Text Encoder
Image Encoder
(2) Create dataset classifier from label text
plane
car
dog
bird
A photo of a {object}.
Text Encoder
(3) Use for zero-shot prediction
Image Encoder
A photo of a dog.

SimSiam

Exploring Simple Siamese Representation Learning

Xinlei Chen, Kaiming He

Self-/Unsupervised Pre-Training

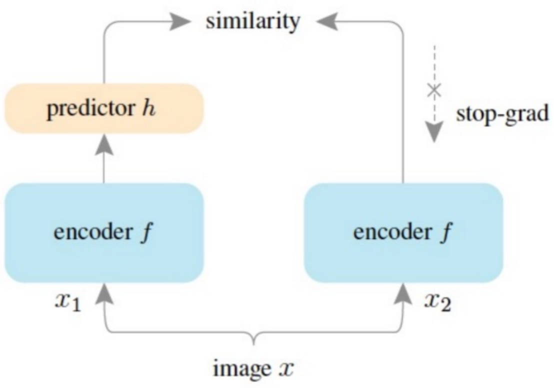


- Many exciting frameworks in recent years
  - MoCo: surpasses supervised pre-training on multiple vision tasks
  - SimCLR/MoCo (v2)/BYOL/SwAV: closes accuracy gap on ImageNet
- Common structure: Siamese networks
  - Weight-sharing networks applied to multiple inputs
  - Simplest form: two views from the same image predict the same output, maximize similarity
  - However, it suffers from collapsing solution that all inputs output the same
- Countering strategies in the literature
  1. Contrastive learning, with negatives (MoCo, SimCLR)
  2. Clustering with balanced size (SwAV)
  3. Momentum encoder w/ predictor (?) (BYOL)

SimSiam: Simple Siamese Representation Learning

*predictor can be removed without collapsing (see paper)

Architecture



PyTorch-like code

```
# f: backbone + projection mlp
# h: prediction mlp

for x in loader: # load a minibatch x with n samples
    x1, x2 = aug(x), aug(x) # random augmentation
    z1, z2 = f(x1), f(x2) # projections, n-by-d
    p1, p2 = h(z1), h(z2) # predictions, n-by-d

    L = D(p1, z2)/2 + D(p2, z1)/2 # loss

    L.backward() # back-propagate
    update(f, h) # SGD update

def D(p, z): # negative cosine similarity
    z = z.detach() # stop gradient

    p = normalize(p, dim=1) # l2-normalize
    z = normalize(z, dim=1) # l2-normalize
    return -(p*z).sum(dim=1).mean()
```

Analysis

	top-1
w/ stop-grad	67.7±0.1
w/o stop-grad	0.1

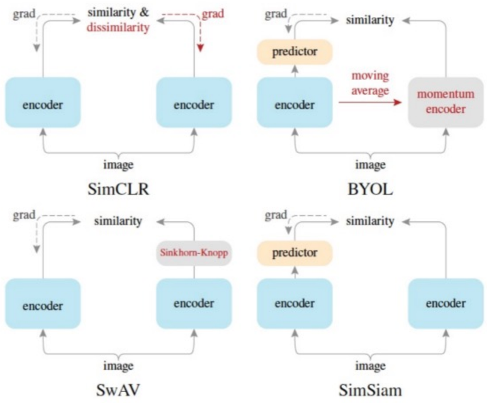
stop-grad is *critical*

	top-1
w/ default	67.7
w/o *pred.*	0.1
random *pred.*	1.5
not decay *pred.* lr	68.1

predictor is *important**

Comparison to Others

Architecture



SimSiam works *without:*
1) negatives, 2) large batches, and 3) momentum encoders

ImageNet Linear Classification

method	batch size	negative pairs	momentum encoder	100-ep	200-ep	400-ep	800-ep
SimCLR	4096	✓		66.5	68.3	69.8	70.4
MoCo	256	✓	✓	67.4	69.9	71.0	72.2
BYOL	4096		✓	66.5	70.6	73.2	74.3
SwAV	4096			66.5	69.1	70.7	71.8
SimSiam	256			68.1	70.0	70.8	71.3

VOC Detection Transfer

method	AP50	AP75	AP
Supervised	74.4	42.4	42.7
SimCLR	75.9	46.8	50.1
MoCo	77.1	48.5	52.5
BYOL	77.1	47.0	49.9
SwAV	75.5	46.5	49.6
SimSiam (Optimal)	77.3	48.5	52.5

Discussions

- Siamese networks are useful for invariance
  - Invariance: two views of the same concept produce the same output
  - Translation-invariance is baked in ConvNets, but harder for others
  - Siamese networks serve as a data-driven baseline without inductive biases (e.g., vision transformers)

```python
super(SimSiam, self).__init__()

# create the encoder
# num_classes is the output fc dimension, zero-initialize last BNs
self.encoder = base_encoder(num_classes=dim, zero_init_residual=True)

# build a 3-layer projector
prev_dim = self.encoder.fc.weight.shape[1]
self.encoder.fc = nn.Sequential(nn.Linear(prev_dim, prev_dim, bias=False),
                                nn.BatchNorm1d(prev_dim),
                                nn.ReLU(inplace=True), # first layer
                                nn.Linear(prev_dim, prev_dim, bias=False),
                                nn.BatchNorm1d(prev_dim),
                                nn.ReLU(inplace=True), # second layer
                                self.encoder.fc,
                                nn.BatchNorm1d(dim, affine=False)) # output layer
self.encoder.fc[6].bias.requires_grad = False # hack: not use bias as it is followed by BN

# build a 2-layer predictor
self.predictor = nn.Sequential(nn.Linear(dim, pred_dim, bias=False),
                                nn.BatchNorm1d(pred_dim),
                                nn.ReLU(inplace=True), # hidden layer
                                nn.Linear(pred_dim, dim)) # output layer
```

```python
# Data loading code
traindir = os.path.join(args.data, 'train')
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])

# MoCo v2's aug: similar to SimCLR https://arxiv.org/abs/2002.05709
augmentation = [
    transforms.RandomResizedCrop(224, scale=(0.2, 1.)),
    transforms.RandomApply([
        transforms.ColorJitter(0.4, 0.4, 0.4, 0.1)  # not strengthened
    ], p=0.8),
    transforms.RandomGrayscale(p=0.2),
    transforms.RandomApply([simsiam.loader.GaussianBlur([.1, 2.])], p=0.5),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    normalize
]
```