A Bootstrapping

Bootstrapping. As stated in BYOL [12]: "the term *bootstrap* is used in its idiomatic sense rather than the statistical sense," *e.g.*, DeepCluster [2] uses bootstrapping on previous versions of its representation to produce targets (cluster indices) for the next representation. Methods based on self-training or pseudo-labels are also considered bootstrapping since they use information from previous steps to provide *targets* for next step.

A.1 BYOL

BYOL [12] proposes to bootstrap the representations directly. In particular, BYOL has two networks, referred to as online and target networks. Given two augmented views of the same image, BYOL uses two networks to extract their representations and employs an additional MLP head, called the *prediction head*, to predict the latent representation produced by the target network from the representation produced by the online network.

Learn from a randomly initialized model. One of the core motivations for BYOL [12] is an interesting observation: they train a model to predict the representations of a *fixed randomly initialized* model and can reach 18.8% top-1 accuracy in linear probing protocol[1] on ImageNet, whereas the randomly initialized model itself only achieves 1.4% top-1 accuracy.

Improve latents with momentum. Learning from a randomly initialized model can yield significantly better results (*e.g.*, +17.4%). Therefore, learning from a dynamically improved model seems to be intuitive. In practice, BYOL [12] optimizes the online network to predict the target network's latent (feature representation). The target network is updated by a slowly moving exponential average of the online network, *i.e.*, the target network is a momentum copy of the online network. The momentum mechanism is similar to the momentum encoder in MoCo [14,5].

A.2 b-ConCL

Bootstrap your own perception. b-ConCL resembles BYOL [12] in terms of using the cluster results from a momentum key encoder. b-ConCL bootstraps the perception from the momentum encoder while simultaneously improving it and refining it via momentum. Figure A.1 shows how the concepts grouped by b-ConCL are refined as training continues.

[1] In linear probing protocol, the backbone, *e.g.*, a ResNet, is frozen, while a newly added fully-connected layer is optimized with respect to ImageNet classification [8].

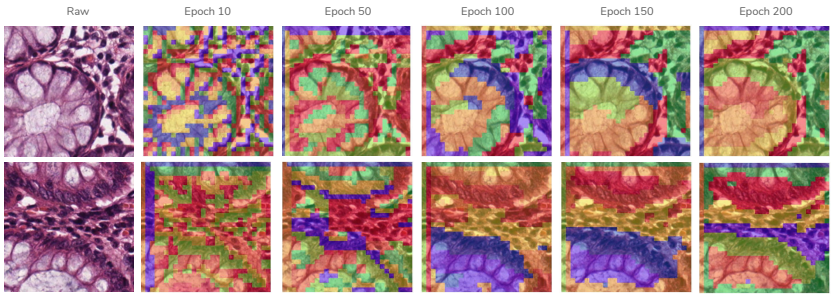

Figure A.1: **b-ConCL refines concepts.** We resize images to 448×448 and visualize the concepts clustered from $f_4(\cdot)$ with $K = 8$, *i.e.*, 8 clusters. Initially, the grouped concepts are edge-related, but later they become more semantic structure-related.

B Additional Implementation Details

B.1 ConCL

Data augmentation. Following MoCo-v2 [5], we use: `RandomResizedCrop`, and `ColorJitter` with (brightness=0.4, contrast=0.4, saturation=0.4, hue=0.1) and probability of 0.8, `RandomGrayscale` with probability of 0.2, `GaussianBlur` with probability of 0.5, and `RandomHorizontalFlip`.

Practical Complement. In practice, the training process of MoCo [14,5] is distributed across multiple GPUs. Therefore, features need to be gathered from all GPUs before updating the queue. The gathering operation requires tensors from different GPUs to have the same shape. However, this usually does not hold for ConCL, where the number of the shared concepts (*i.e.*, concepts in both views) varies per pair. We resolve this problem by padding concept keys with randomly re-sampled concepts from the current batch to match the requirement. Specifically, we pad the number of features in each GPU to K $*$ batch_size/num_GPUs, where K is the number of clusters. When such padding is infeasible (*e.g.*, K $>$ 8), we only randomly sample 4$*$batch_size/num_GPUs concepts. This ensures the validity of the "gather" operation.

B.2 Other Self-supervised Methods

All self-supervised methods use ResNet-18 [17] as the backbone. We subsequently modify the numbers of input and hidden channels of the projection head and the prediction head to match the outputs' dimension from ResNet-18. For data augmentation and other hyper-parameters (*e.g.*, temperature, optimizer), we use the default settings specified in each config file in `OpenSelfSup`,[2] which should be the same as the original methods.

[2] https://github.com/open-mmlab/OpenSelfSup

SimCLR[4] & BYOL [12]. Since both SimCLR and BYOL require a large batch size, we use 1024 for them, which is the maximum number we can afford. For SimCLR, the numbers of channels in each layer of the *projection* head are set to 512-512-128 (input, hidden, output layers). For BYOL, we set them to 512-512-256 (input, hidden, output layers); the numbers of channels in each layer of the *prediction* head are then set to 256-512-256 (input, hidden, output layers).

MoCo-v1[14], MoCo-v2[5] and DenseCL[36]. These three methods require an instance queue. We change the length of the instance queue to 16394 since our pre-training dataset only has 100k samples, far less than 1.28M samples in ImageNet [21]. For MoCo-v1, the numbers of channels in each layer of the projection head are set to 512-128 (input, output layers). For the remaining two methods, we set them to 512-512-128 (input, hidden, output layers).

PCL-v2[22]. We use the officially released code[3] for PCL's experiments. We change the numbers of channels in each layer of the projection head to 512-512-128 (input, hidden, output layers) and the number of clusters in PCL to (20000, 35000, 50000).

C Detection Configuration Details

C.1 GlaS.

Introduction. Glands are important tissue structures for diagnosing adenocarcinomas, a prevalent type of malignant tumor in the prostate, breast, lung, colon, and more. Gland segmentation in pathology images challenge (GlaS) dataset [31] collects images from H&E stained slides with object-instance-level annotation. It consists of a variety of malignant grades. We follow the official train/test split for evaluation, where the training split contains 37 benign and 48 malignant images, and the test split consists of 37 benign and 43 malignant images.

Transferring setup. The batch size is 16, and the base learning rate is 0.02. For learning schedules, we refer the 1× fine-tuning schedule to as training for 5k iterations, with the learning rate decayed by ten times smaller at 4k iteration. Similarly, the 0.5× schedule has a total of 2.5k training iterations with decay at 2k iteration; the 2× schedule has a total of 10k iterations, with decay at 8k iteration, and so forth for other schedules.

C.2 CRAG.

Introduction. The colorectal adenocarcinoma gland (CRAG) dataset [11] collects 213 H&E stained images taken from 38 WSIs with a pixel resolution of

[3] https://github.com/salesforce/PCL

0.55μm/pixel at 20× magnification. Images are mostly of size 1512×1516 with object-instance-level annotation. We follow the official split for evaluation, where the training set has 173 images, and the test set has 40 images with different cancer grades.

Transferring setup. The batch size is 16, and the base learning rate is 0.02. For learning schedules, we refer the 1× fine-tuning schedule to as training for 15k iterations, with the learning rate decayed by ten times smaller at 10k and 13k iterations, respectively. Similarly, the 0.5× schedule has a total of 7.5k training iterations with decay at 5k and 6.5k iterations; the 2× schedule has a total of 30k iterations, with decay at 20k and 26k iterations, and so forth for other schedules. The intuition behind the different fine-tuning schedules for GlaS and CRAG is the difference in dataset sizes, *i.e.*, CRAG has around 2.5 times more training samples than GlaS.

D More Results

Training speed. Currently, b-ConCL relies on a third reference view for concept matching between different views, which does slow down the training speed. For K-Means, we implement it using matrix multiplication so that the clustering process is parallel to a batch of images. Table D.1 compares the training speed of different approaches measured in 200-epoch pre-training. Despite our implementation not being optimized thoroughly, ConCL's training speed is satisfactory given the large gains it attains.

	MoCo-v2 [5]	DenseCL[36]	PCL[22]	BYOL [12]	b-ConCL (ours)
1-epoch	59.6s	65.7s	137.0s	68.8s	77.4s

Table D.1: **Training speeds.** Results are measured in 8-GPU machines.

Qualitative comparison of downstream tasks. Figures D.1 and D.2 show some visualizations of different pre-trained models in the GlaS dataset [31] and CRAG dataset [11], respectively. We non-exhaustively annotate different detection errors by arrows in different colors. Overall, ConCL outperforms other pre-training methods in terms of less false negative (the black arrows), less false positive (the blue arrows), and more complete detection and segmentation (the red arrows).

Numerical results of longer pre-training. In complement to Figure 1-(b,c), we further report the numerical results of the transferring performance of 800-epoch pre-trained models in Table D.2.

Methods	GlaS		CRAG	
	AP^{bb}	AP^{bb}_{75}	AP^{bb}	AP^{bb}_{75}
SimCLR	50.6	56.8	48.1	52.0
BYOL	50.2	56.9	49.3	54.1
MoCo-v1	49.8	55.1	47.2	51.9
MoCo-v2	55.2	63.6	51.8	57.6
DenseCL	56.0	64.8	52.5	58.2
b-ConCL (ours)	**58.6**	**68.1**	**55.1**	**61.4**

Table D.2: **Transferring performance of different 800-epoch pre-trained models on GlaS and CRAG datasets.**

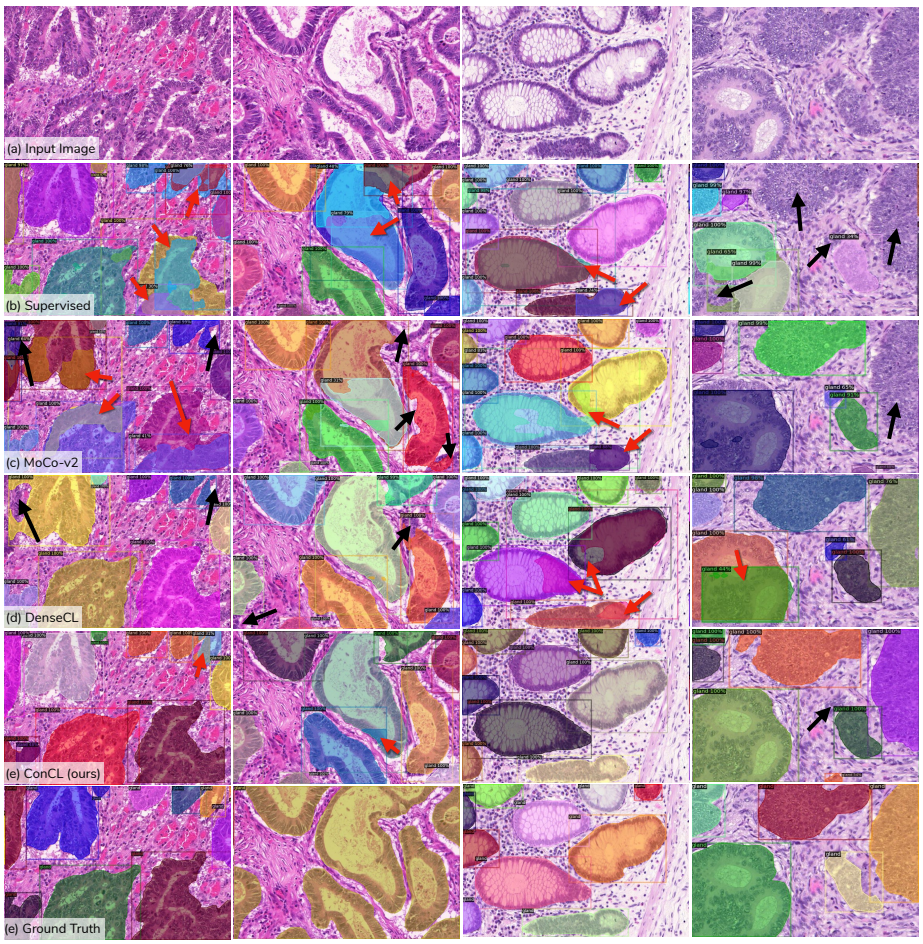

Figure D.1: **Qualitative comparison on GlaS dataset [31].** We show the results with Mask-RCNN R18-FPN under 1× schedule.

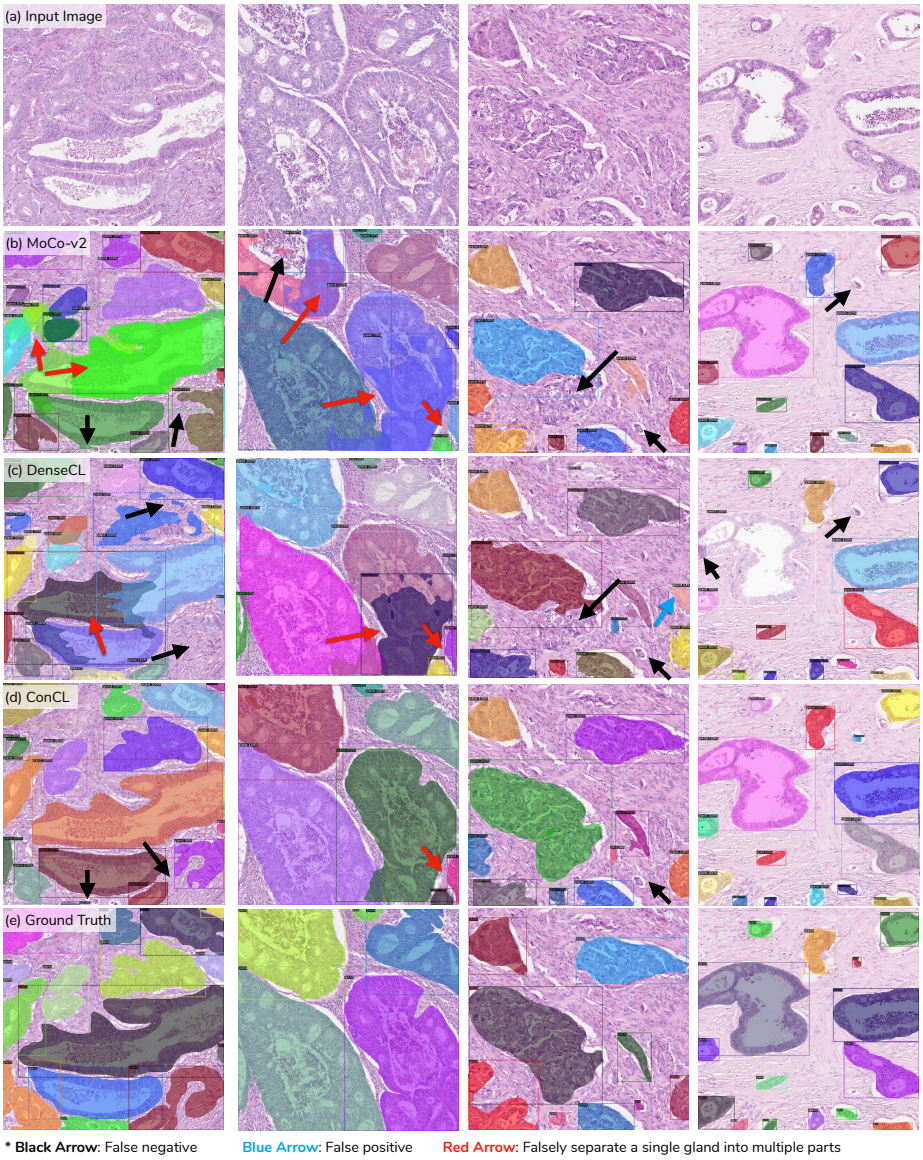

Figure D.2: **Qualitative comparison on CRAG dataset [11].** We show the results with Mask-RCNN R18-FPN under 1× schedule.