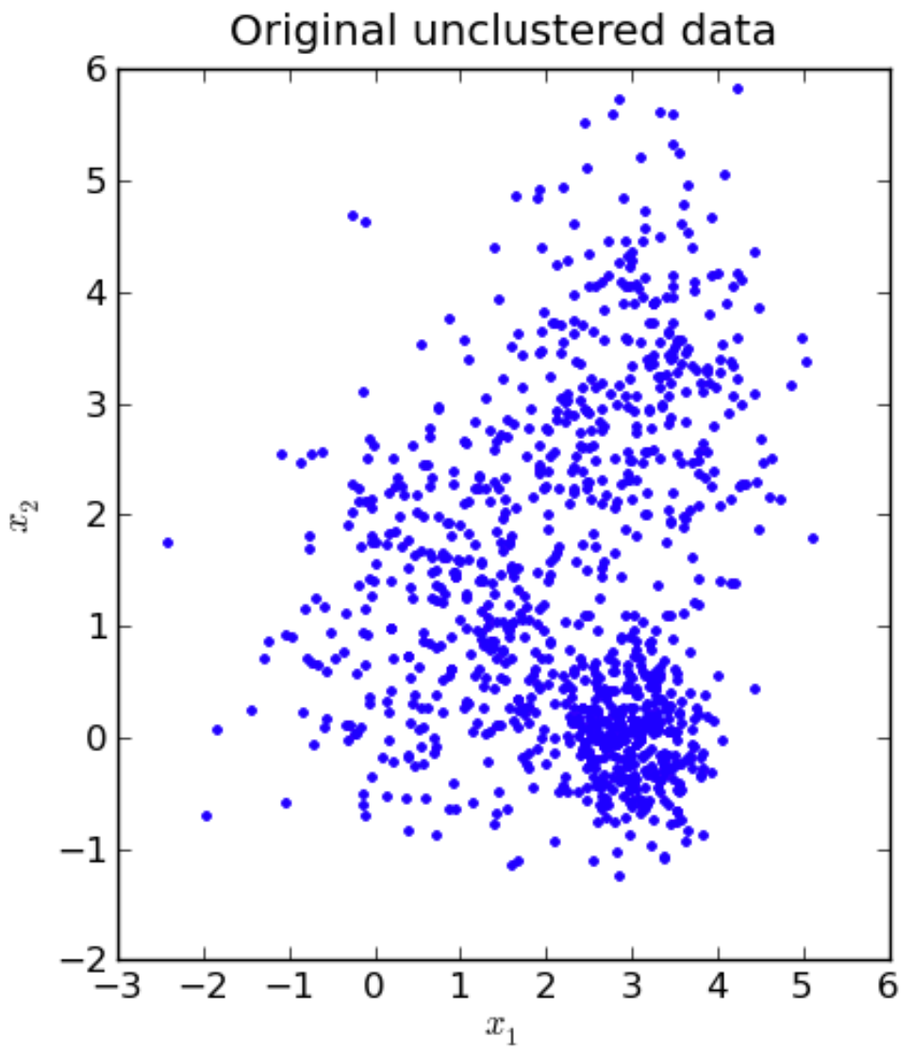
Bootstrap Your Own Latent
A New Approach to Self-Supervised Learning

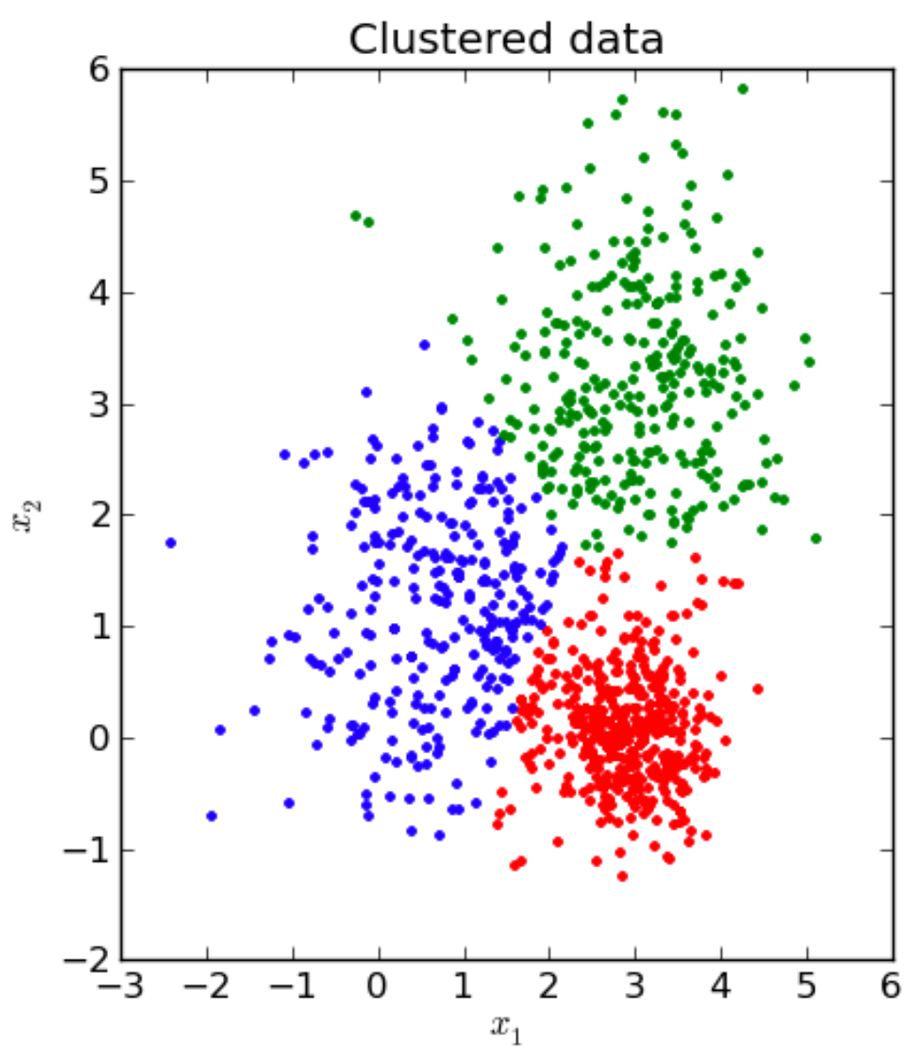
DeepMind Imperial College

Soojung Kim
2020. 07. 27

Introduction

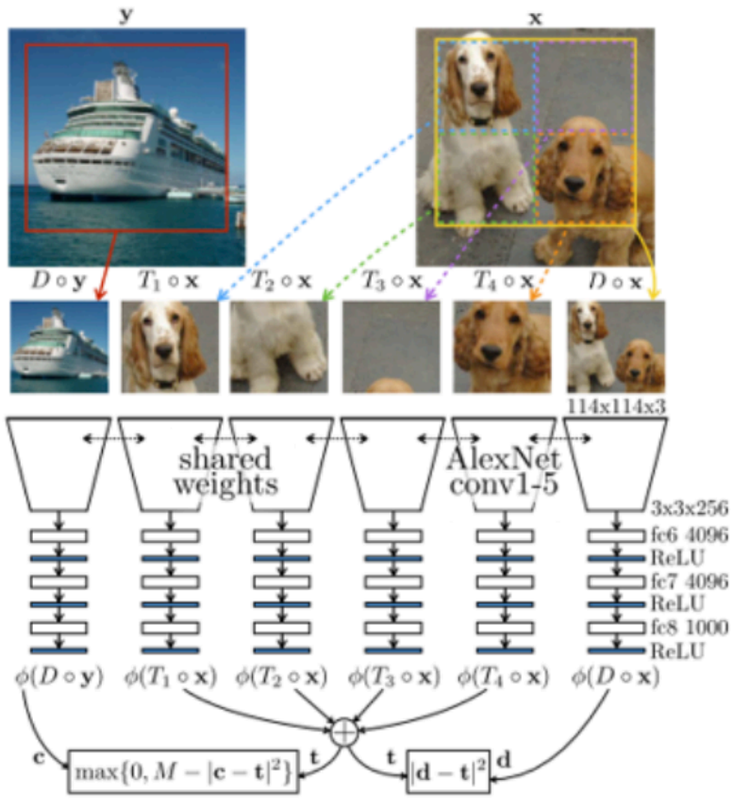
- Unsupervised learning

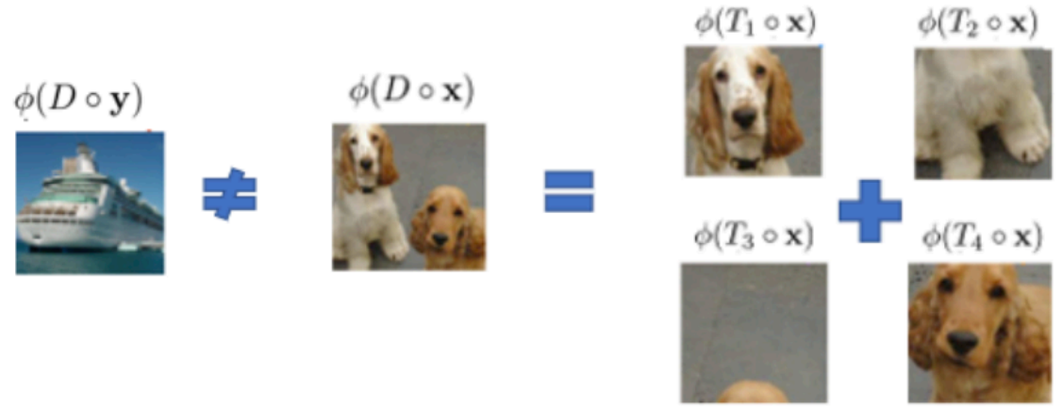




Introduction

- self-supervised learning: pretext task





$$\ell(\mathbf{x}) = \left|\phi(D \circ \mathbf{x}) - \sum_{j=1}^{4} \phi(T_j \circ \mathbf{x})\right|^2. \qquad (3)$$

l_2 loss

$$\ell_{\text{con}}(\mathbf{x}, \mathbf{y}) = \left|\phi(D \circ \mathbf{x}) - \sum_{j=1}^{4} \phi(T_j \circ \mathbf{x})\right|^2 + \max\left\{0, M - \left|\phi(D \circ \mathbf{y}) - \sum_{j=1}^{4} \phi(T_j \circ \mathbf{x})\right|^2\right\} \qquad (4)$$

contrastive loss

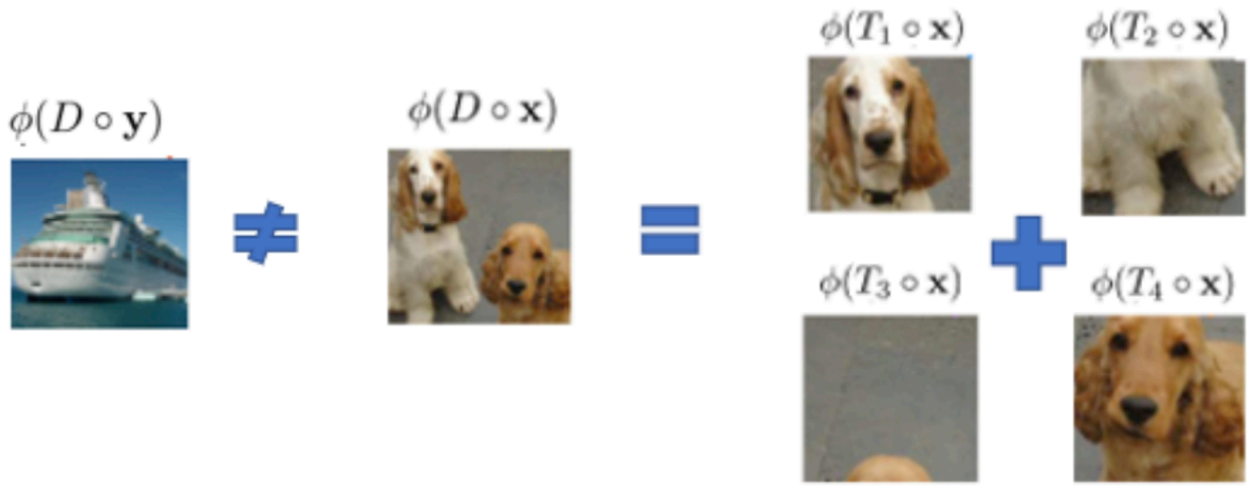
[Network Architecture & Loss Functions]

https://openaccess.thecvf.com/content_ICCV_2017/papers/Noroozi_Representation_Learning_by_ICCV_2017_paper.pdf

https://hoya012.github.io/blog/Self-Supervised-Learning-Overview/

Introduction

- self-supervised learning: pretext task



$$\ell(\mathbf{x}) = \left|\phi(D \circ \mathbf{x}) - \sum_{j=1}^{4} \phi(T_j \circ \mathbf{x})\right|^2. \qquad (3)$$

l_2 loss

$$\ell_{\text{con}}(\mathbf{x}, \mathbf{y}) = \left|\phi(D \circ \mathbf{x}) - \sum_{j=1}^{4} \phi(T_j \circ \mathbf{x})\right|^2 + \max\left\{0, M - \left|\phi(D \circ \mathbf{y}) - \sum_{j=1}^{4} \phi(T_j \circ \mathbf{x})\right|^2\right\} \qquad (4)$$

contrastive loss

https://openaccess.thecvf.com/content_ICCV_2017/papers/Noroozi_Representation_Learning_by_ICCV_2017_paper.pdf

https://hoya012.github.io/blog/Self-Supervised-Learning-Overview/

Introduction

- self-supervised learning: Dataset Generalization: PASCAL VOC

	Classification (%mAP)	Detection (%mAP)	Segmentation (%mIoU)

etuning

Method	Conv1	Conv2	Conv3	Conv4	Conv5
ImageNet labels	19.3	36.3	44.2	48.3	50.5
Random	11.6	17.1	16.9	16.3	14.1
Random rescaled Krähenbühl et al. (2015)	17.5	23.0	24.5	23.2	20.6
Context (Doersch et al., 2015)	16.2	23.3	30.2	31.7	29.6
Context Encoders (Pathak et al., 2016b)	14.1	20.7	21.0	19.8	15.5
Colorization (Zhang et al., 2016a)	12.5	24.5	30.4	31.5	30.3
Jigsaw Puzzles (Noroozi & Favaro, 2016)	18.2	28.8	34.0	33.9	27.1
BIGAN (Donahue et al., 2016)	17.7	24.5	31.0	29.9	28.0
Split-Brain (Zhang et al., 2016b)	17.7	29.3	35.4	35.2	32.8
Counting (Noroozi et al., 2017)	18.0	30.6	34.3	32.5	25.7
(Ours) RotNet	**18.8**	**31.7**	**38.7**	**38.2**	**36.5**

Self-Supervised

S

Supe

Counting (Noroozi et al., 2017)	-	67.7	51.4	36.6
(Ours) RotNet	**70.87**	**72.97**	**54.4**	**39.1**

[Dataset Generalization Result]

Introduction

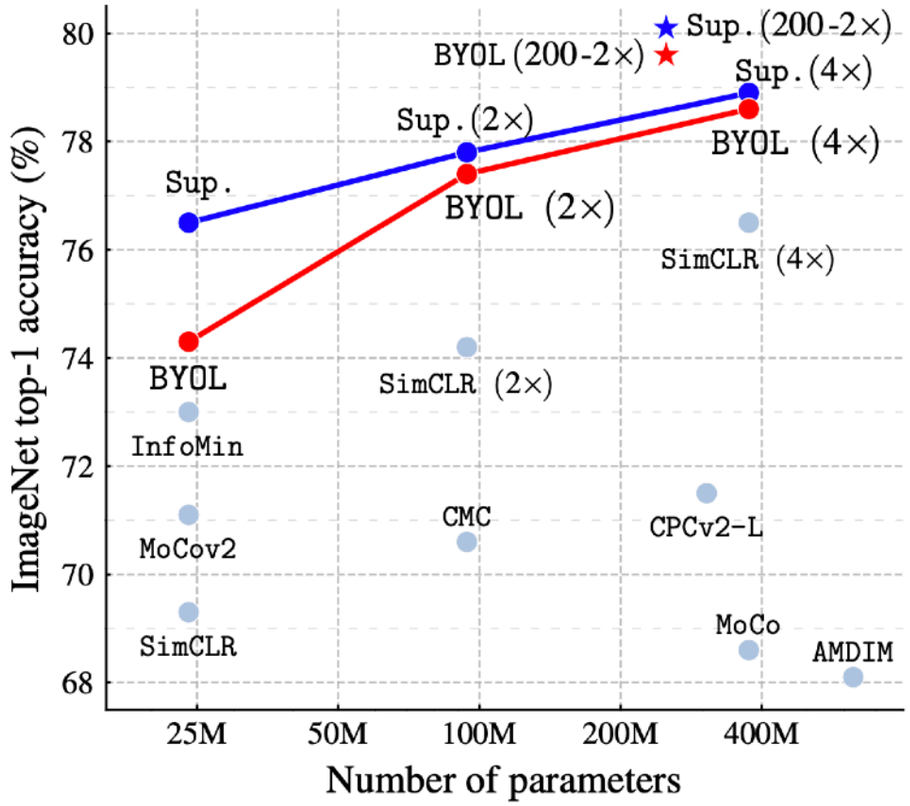
- self-supervised learning: Task Generalization: ImageNet Classification

	Classification (%mAP)		Detection (%mAP)	Segmentation (%mIoU)	
Trained layers	fc6-8	all	all	all	finetuning
ImageNet labels	78.9	79.9	56.8	48.0	
Random		53.3	43.4	19.8	
Random rescaled Krähenbühl et al. (2015)	39.2	56.6	45.6	32.6	
Egomotion (Agrawal et al., 2015)	31.0	54.2	43.9		Self-Supervised
Context Encoders (Pathak et al., 2016b)	34.6	56.5	44.5	29.7	
Tracking (Wang & Gupta, 2015)	55.6	63.1	47.4		
Context (Doersch et al., 2015)	55.1	65.3	51.1		
Colorization (Zhang et al., 2016a)	61.5	65.6	46.9	35.6	
BIGAN (Donahue et al., 2016)	52.3	60.1	46.9	34.9	
Jigsaw Puzzles (Noroozi & Favaro, 2016)	-	67.6	53.2	37.6	
NAT (Bojanowski & Joulin, 2017)	56.7	65.3	49.4		
Split-Brain (Zhang et al., 2016b)	63.0	67.1	46.7	36.0	
ColorProxy (Larsson et al., 2017)		65.9		38.4	
Counting (Noroozi et al., 2017)	-	67.7	51.4	36.6	
(Ours) RotNet	**70.87**	**72.97**	**54.4**	**39.1**	

[Dataset Generalization Result]

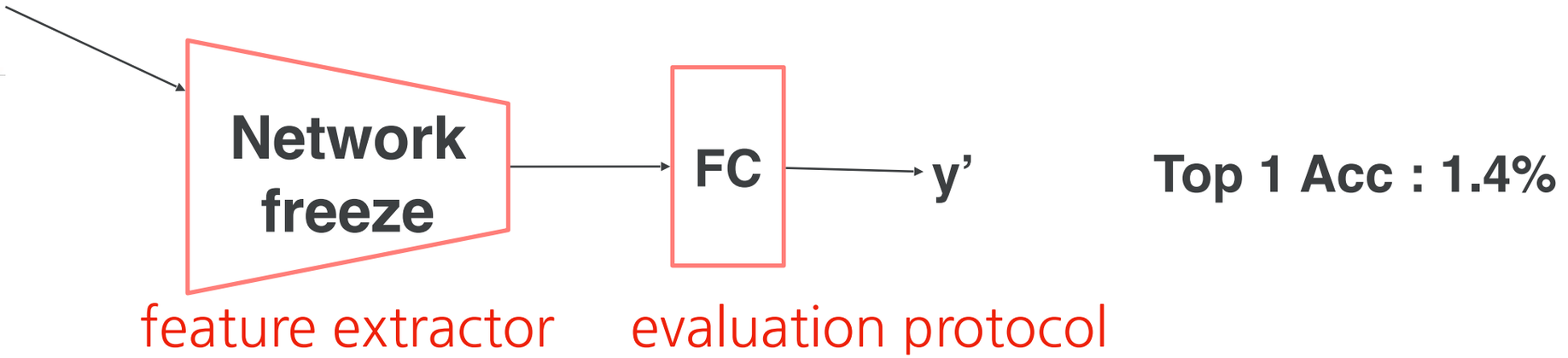
Abstract

BYOL achieves higher performance than sota without using negative pairs.
and then use two neural networks.



We introduce **B**ootstrap **Y**our **O**wn **L**atent (BYOL), a new approach to self-supervised image representation learning. BYOL relies on two neural networks, referred to as *online* and *target* networks, that interact and learn from each other. From an augmented view of an image, we train the online network to predict the target network representation of the same image under a different augmented view. At the same time, we update the target network with a slow-moving average of the online network. While state-of-the art methods intrinsically rely on negative pairs, BYOL achieves a new state of the art *without them*. BYOL reaches 74.3% top-1 classification accuracy on

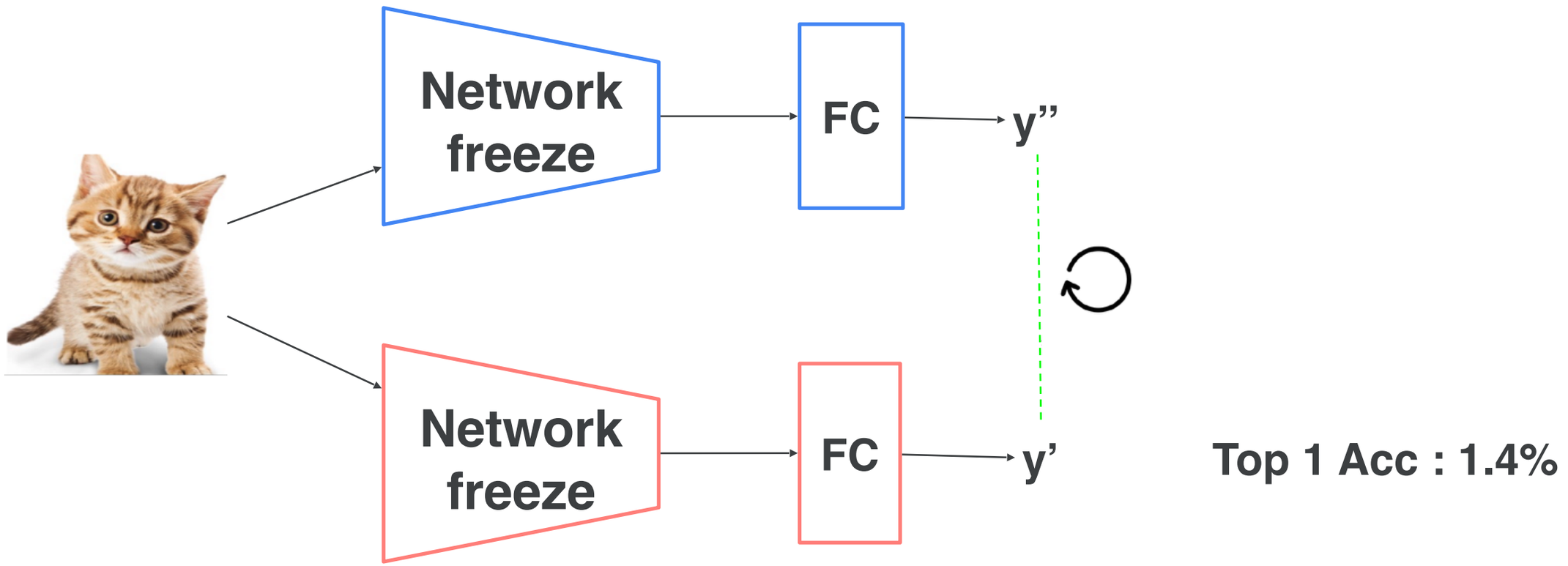
Motivation



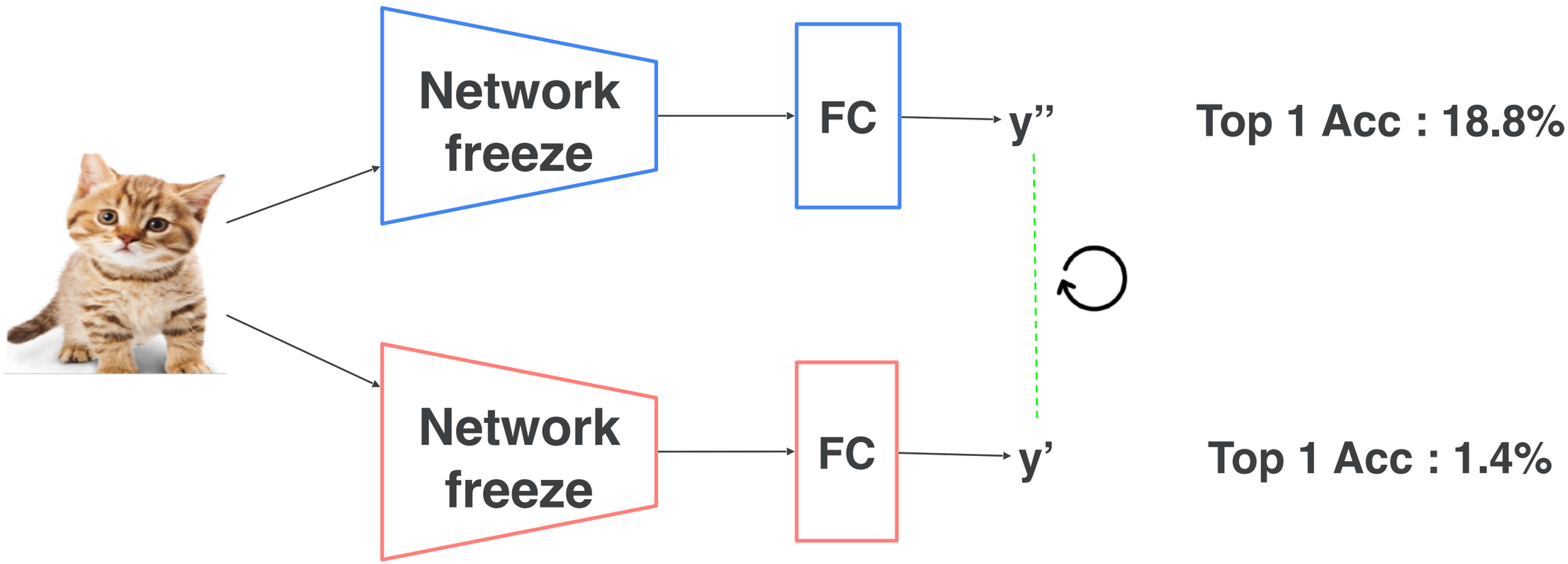
Top 1 Acc : 1.4%

https://github.com/taki0112

Motivation



Motivation


Network freeze
FC
y”
Top 1 Acc : 18.8%
Network freeze
FC
y’
Top 1 Acc : 1.4%

Description of BYOL

BYOL's goal is to learn a representation y which can then be used for downstream tasks.

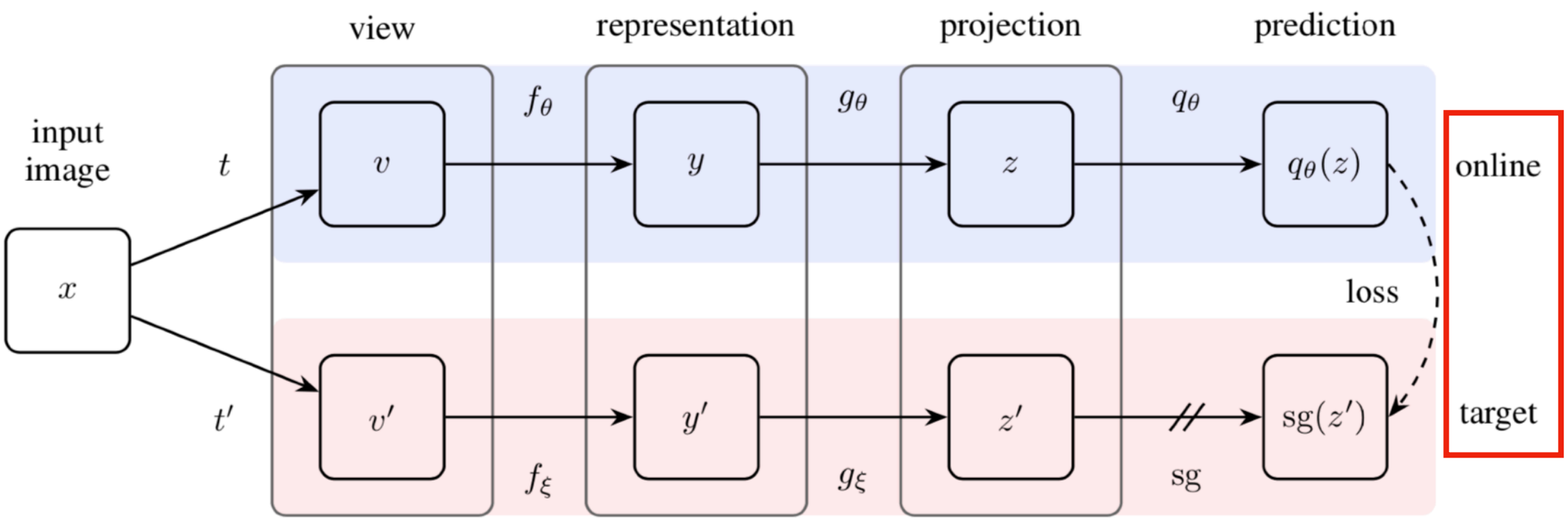
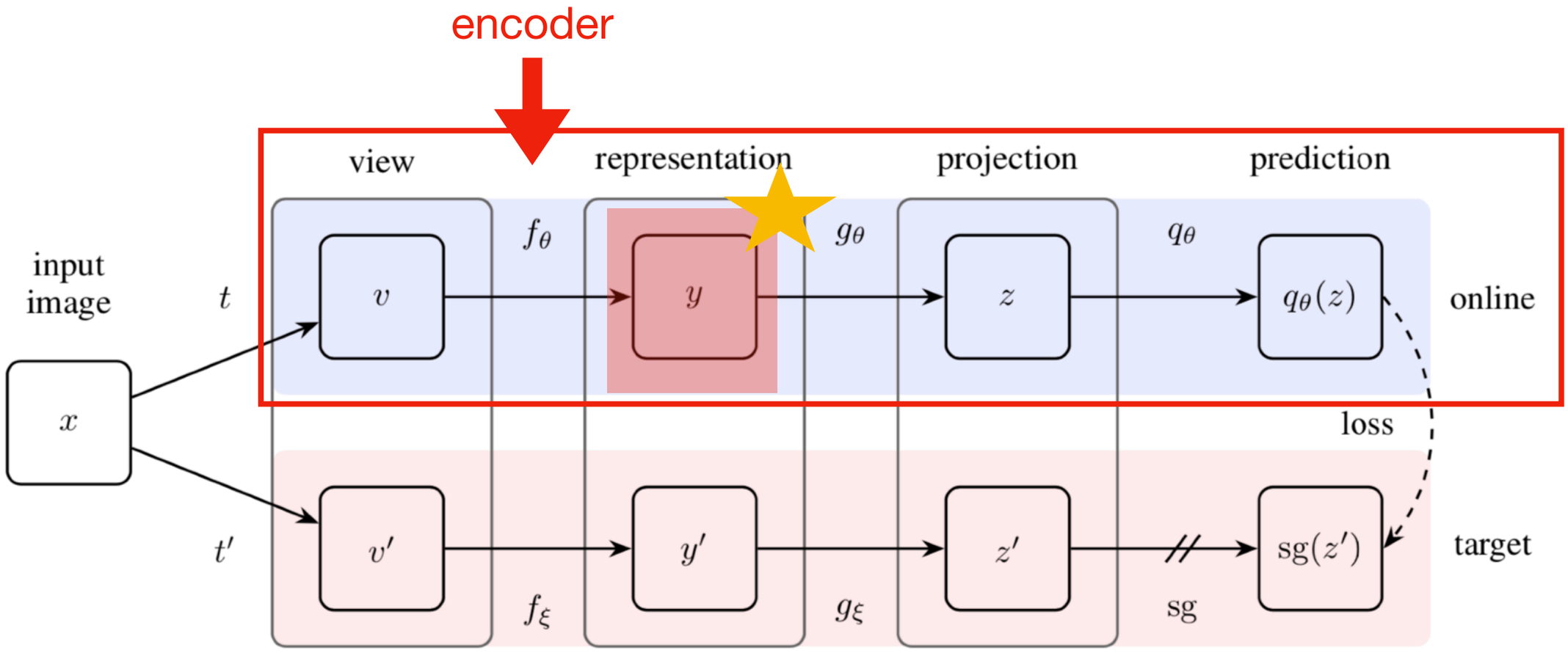


Figure 2: BYOL's architecture. BYOL minimizes a similarity loss between $q_\theta(z)$ and $\mathrm{sg}(z')$, where θ are the trained weights, ξ are an exponential moving average of θ and sg means stop-gradient. At the end of training, everything but f_θ is discarded and y is used as the image representation.

Description of BYOL

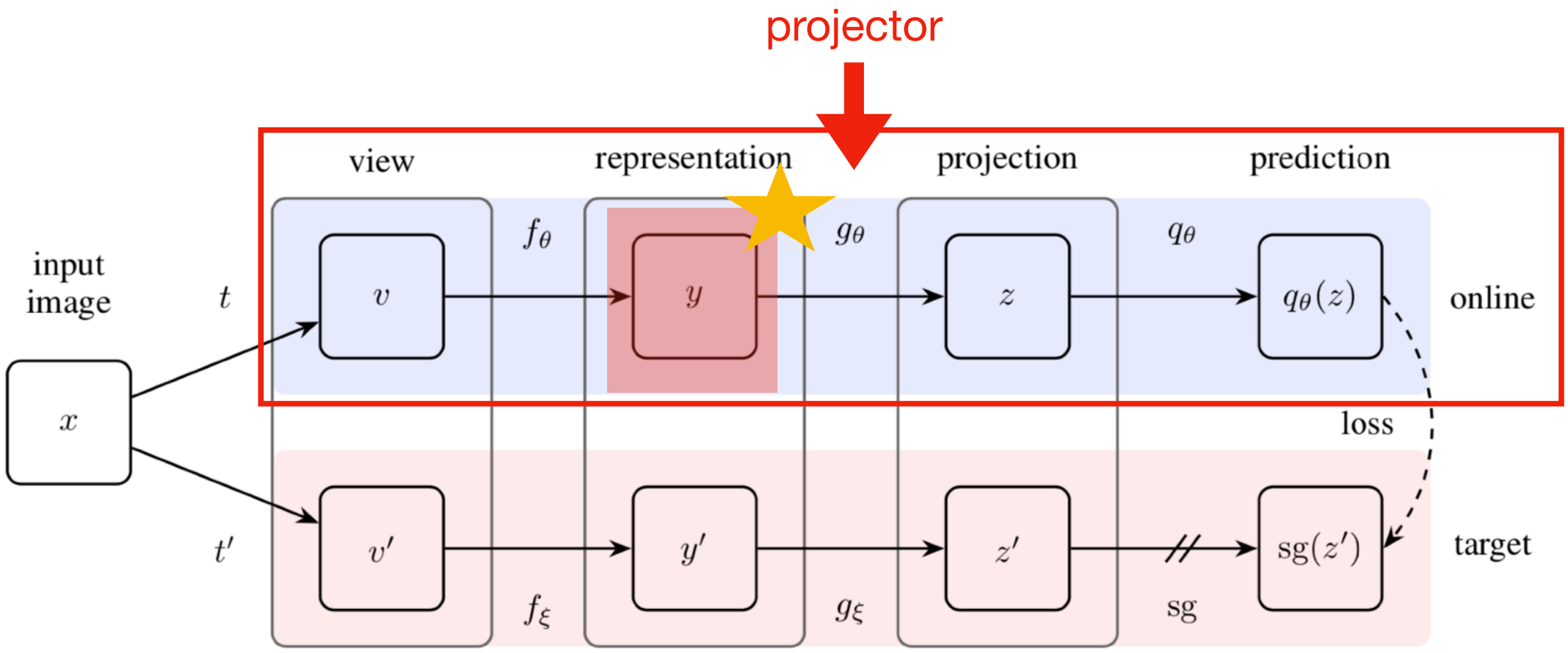
The online network is defined by weights θ and is comprised of three stages: an encoder fθ, a projector gθ and a predictor qθ




```python
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)
```

Description of BYOL

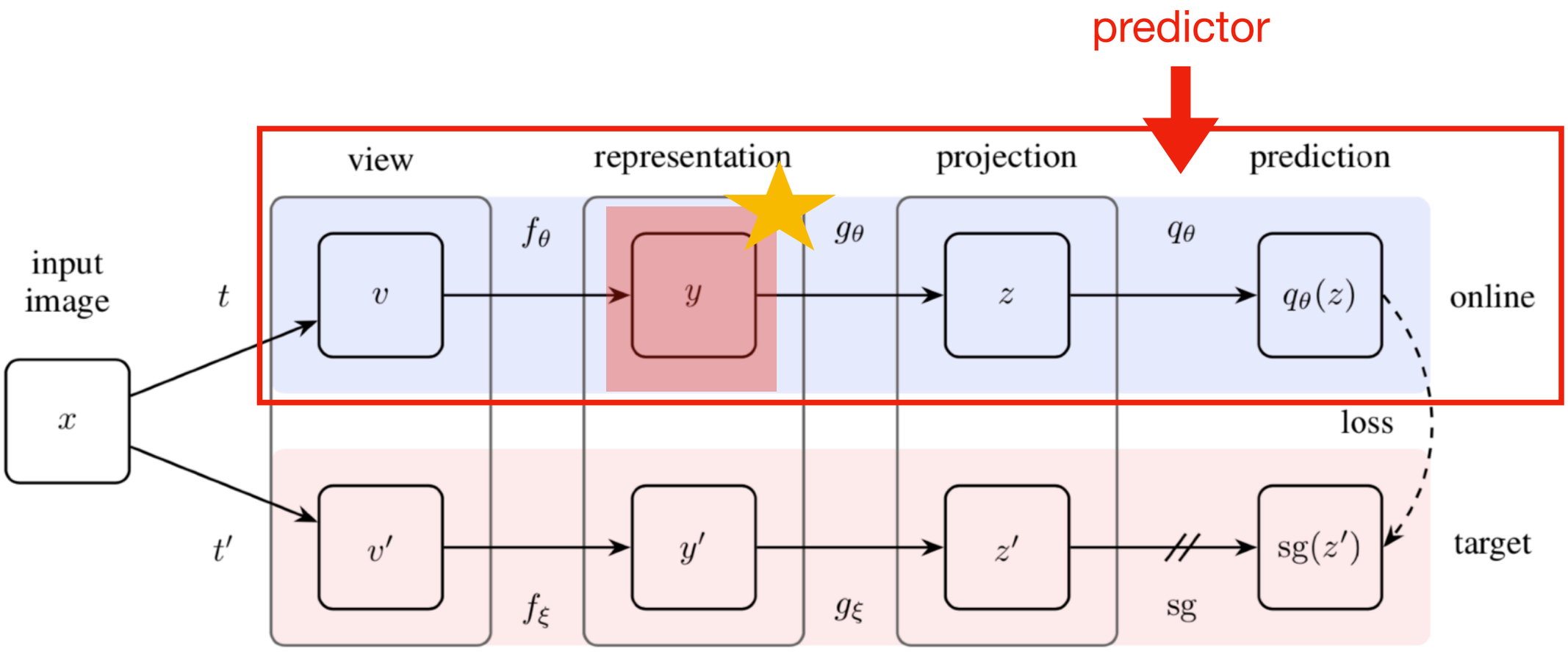
The online network is defined by weights θ and is comprised of three stages: an encoder f_θ, a projector g_θ and a predictor q_θ




```python
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)
```

Description of BYOL

The online network is defined by weights θ and is comprised of three stages: an encoder fθ, a projector gθ and a predictor qθ




```
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)
```

MLP

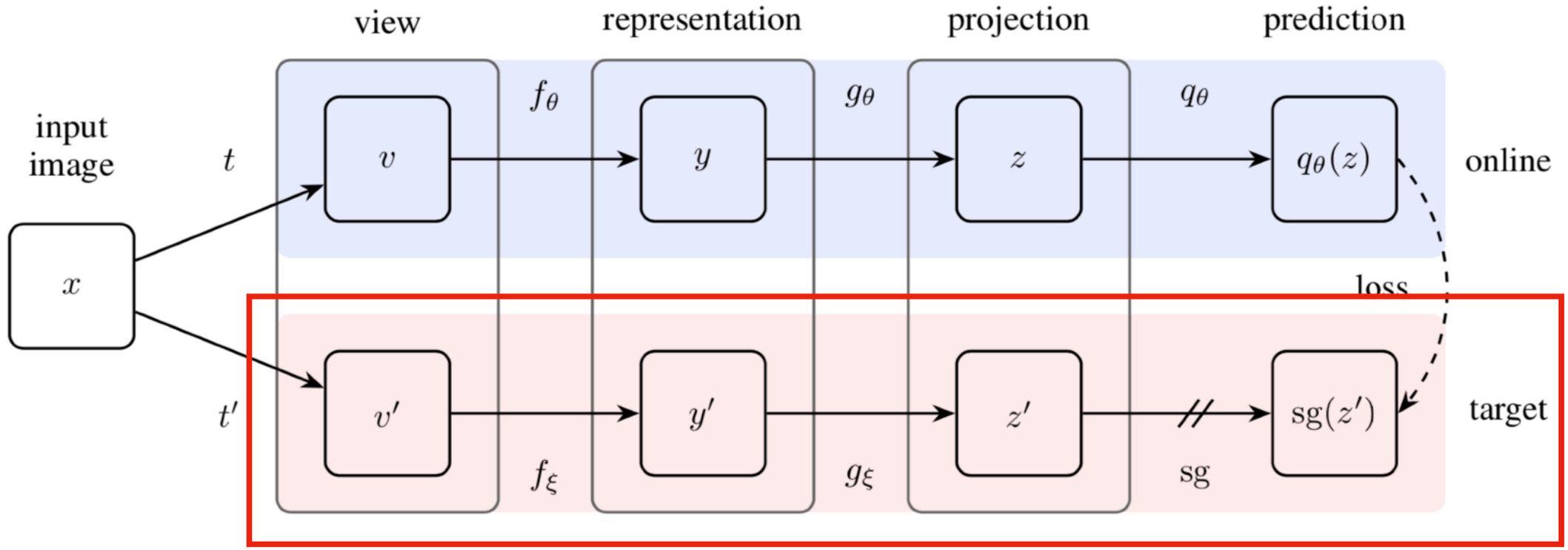
```python
class MLP(hk.Module):
  """Multi Layer Perceptron, with normalization."""

  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
```

mlp_hidden_size = 4096

projection_size = 256

Description of BYOL

The target network provides the regression target to train the online network,
and its weights are an exponential moving average of the online network's weights.



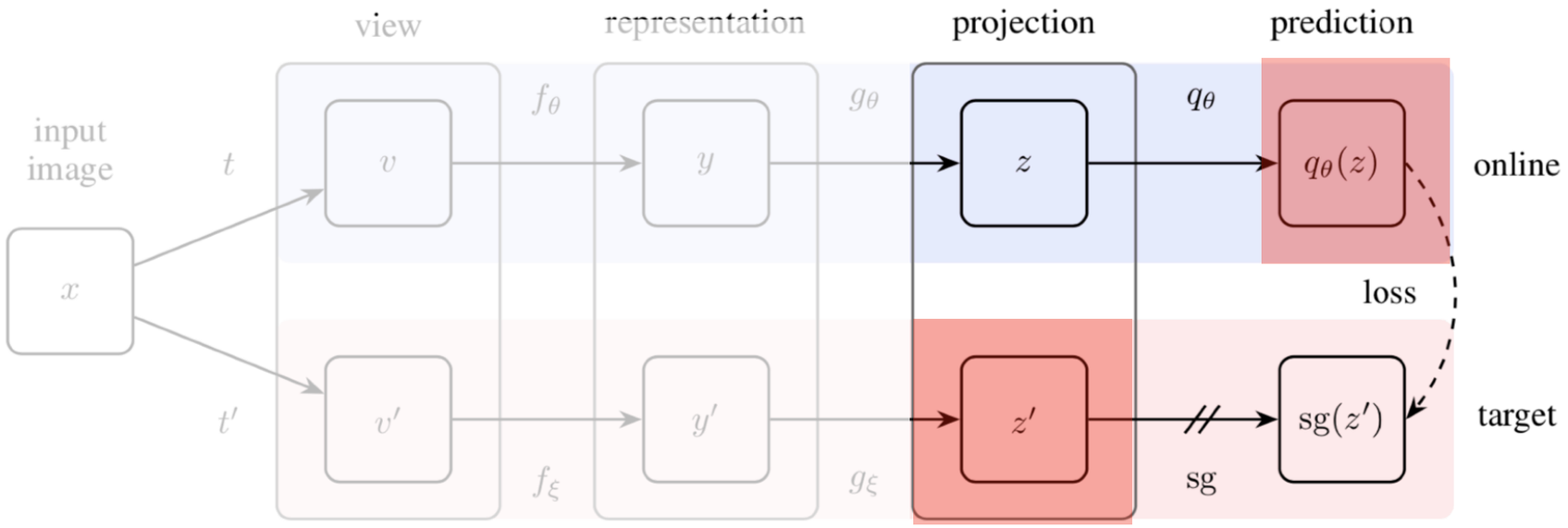
Exponential moving average

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

target w　　online w

Description of BYOL

Loss: MSE with l2 normalize

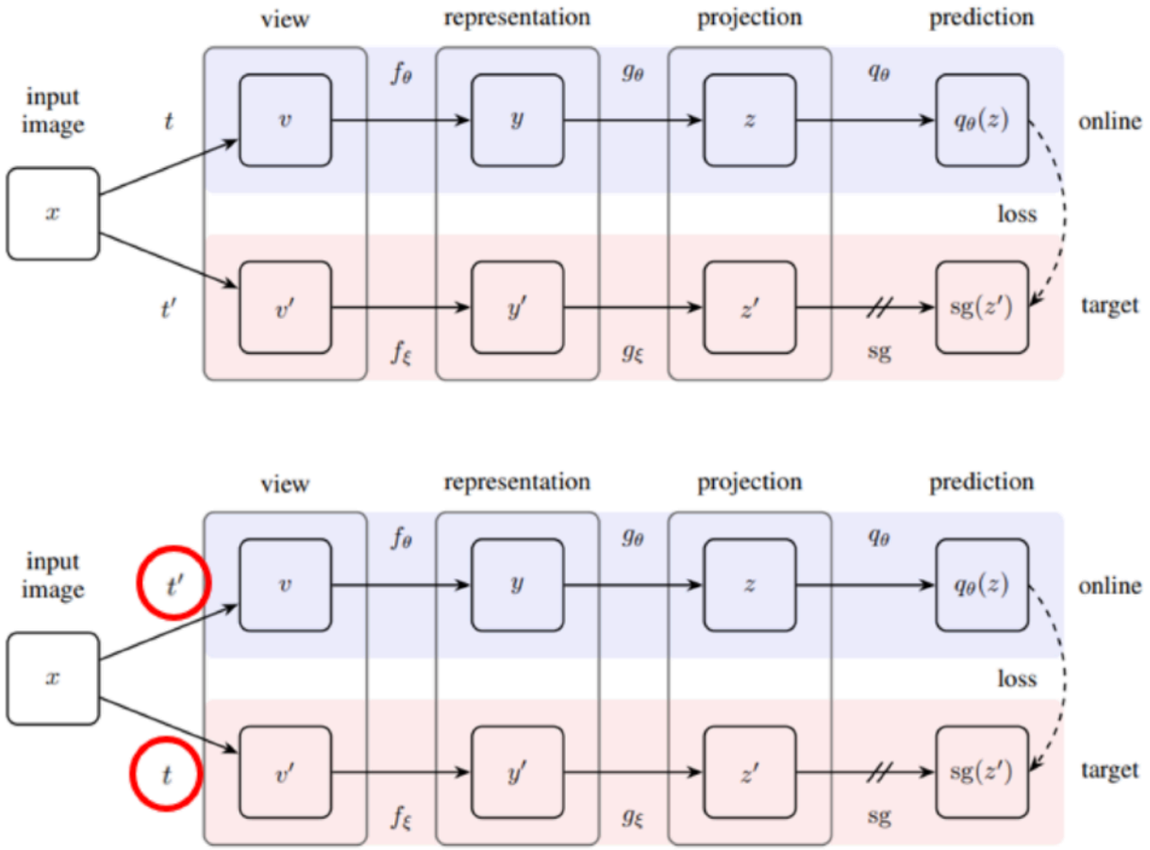


$$\mathcal{L}_\theta^{\texttt{BYOL}} \triangleq \left\| \overline{q_\theta}(z_\theta) - \overline{z}'_\xi \right\|_2^2 = 2 - 2 \cdot \frac{\langle q_\theta(z_\theta), z'_\xi \rangle}{\left\| q_\theta(z_\theta) \right\|_2 \cdot \left\| z'_\xi \right\|_2}.$$

```
def regression_loss(x, y):
  norm_x, norm_y = jnp.linalg.norm(x), jnp.linalg.norm(y)
  return -2. * jnp.sum(x * y, axis=-1) / (norm_x * norm_y)

# The stop_gradient is not necessary as we explicitly take the gradient with
# respect to online parameters only. We leave it to indicate that gradients
# are not backpropagated through the target network.
loss = regression_loss(online_network_out_1['prediction'],
                       jax.lax.stop_gradient(target_network_out_2['projection']))
```

Description of BYOL

Symmetrize the loss



$$\mathcal{L}_\theta^{\mathrm{BYOL}}$$

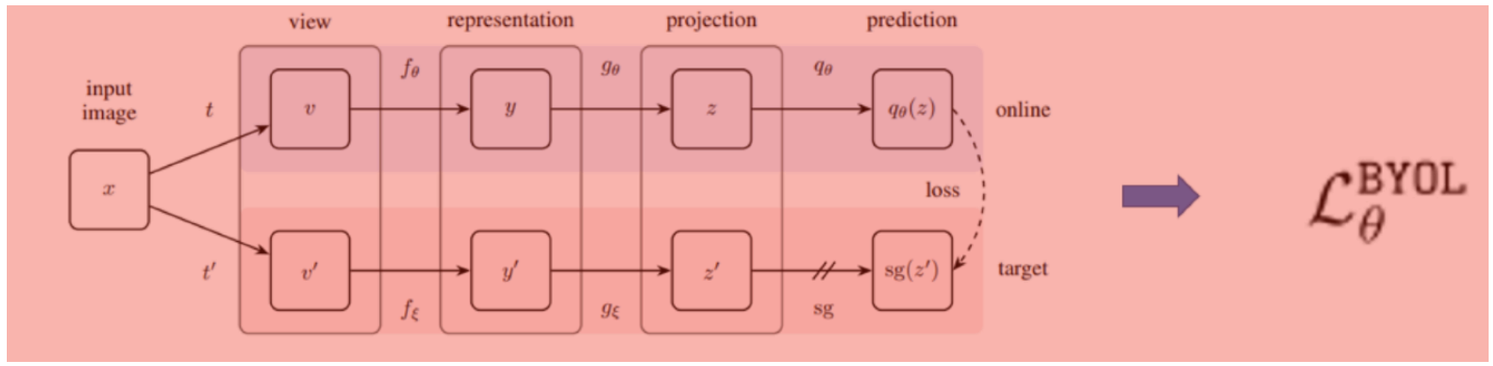
Total Loss

$$\mathcal{L}_\theta^{\mathrm{BYOL}} + \widetilde{\mathcal{L}}_\theta^{\mathrm{BYOL}}$$

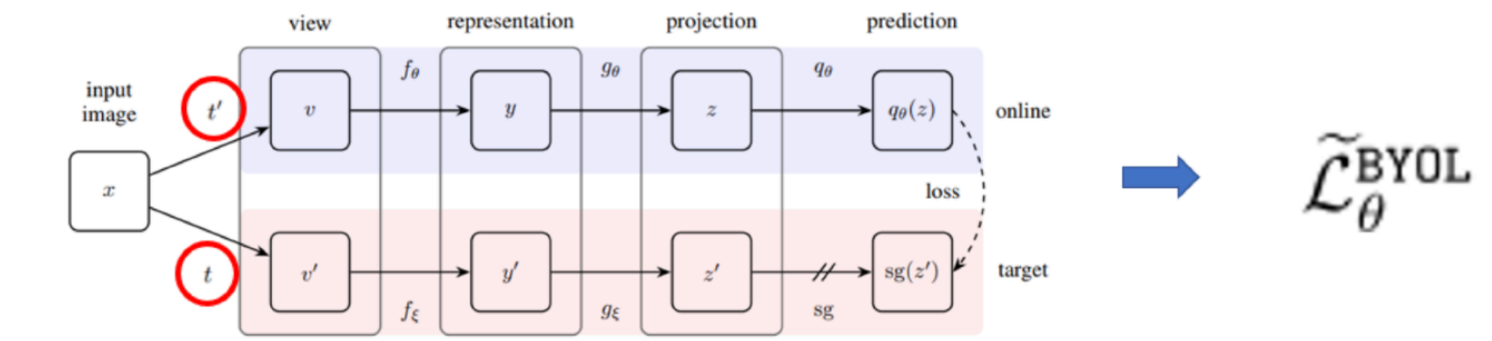
$$\widetilde{\mathcal{L}}_\theta^{\mathrm{BYOL}}$$

https://hoya012.github.io/blog/byol/

Symmetrize the loss



$$\mathcal{L}_\theta^{\mathrm{BYOL}}$$



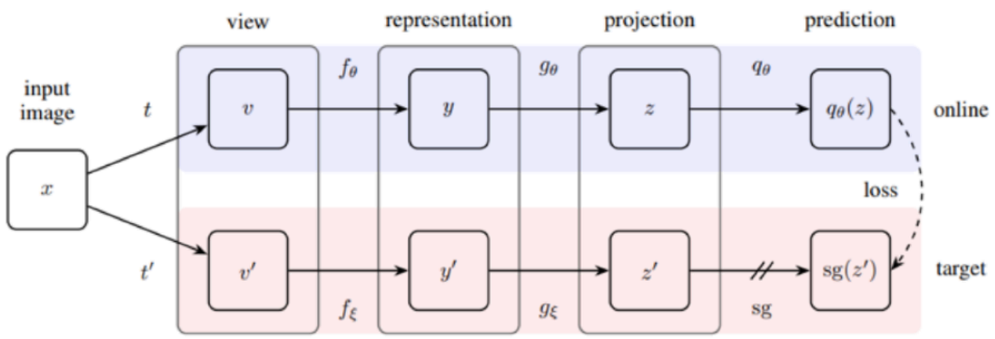
$$\widetilde{\mathcal{L}}_\theta^{\mathrm{BYOL}}$$

Total Loss

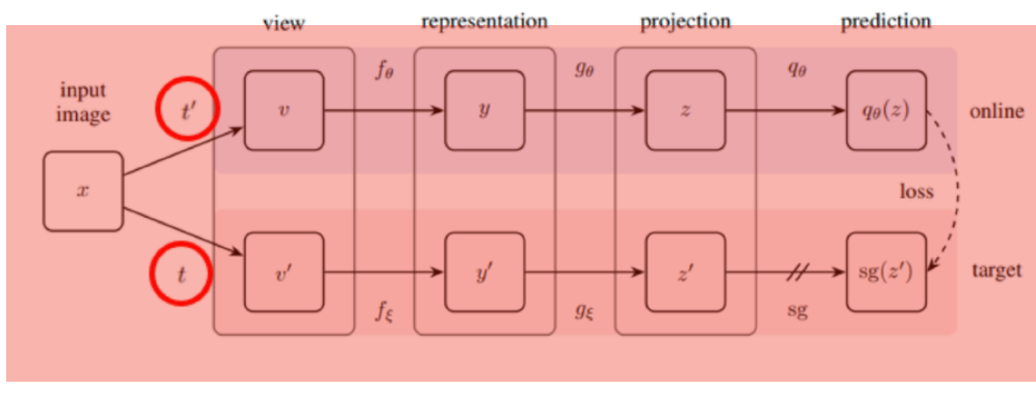
$$\mathcal{L}_\theta^{\mathrm{BYOL}} + \widetilde{\mathcal{L}}_\theta^{\mathrm{BYOL}}$$

```
online_network_out_1 = net_apply(params=online_params, inputs=image_1)
online_network_out_2 = net_apply(params=online_params, inputs=image_2)
target_network_out_1 = net_apply(params=target_params, inputs=image_1)
target_network_out_2 = net_apply(params=target_params, inputs=image_2)

def regression_loss(x, y):
  norm_x, norm_y = jnp.linalg.norm(x), jnp.linalg.norm(y)
  return -2. * jnp.sum(x * y, axis=-1) / (norm_x * norm_y)

# The stop_gradient is not necessary as we explicitly take the gradient with
# respect to online parameters only. We leave it to indicate that gradients
# are not backpropagated through the target network.
loss = regression_loss(online_network_out_1['prediction'],
                       jax.lax.stop_gradient(target_network_out_2['projection']))
loss += regression_loss(online_network_out_2['prediction'],
                        jax.lax.stop_gradient(target_network_out_1['projection']))
return loss
```

Symmetrize the loss



$$\mathcal{L}_\theta^{\mathrm{BYOL}}$$



$$\widetilde{\mathcal{L}}_\theta^{\mathrm{BYOL}}$$

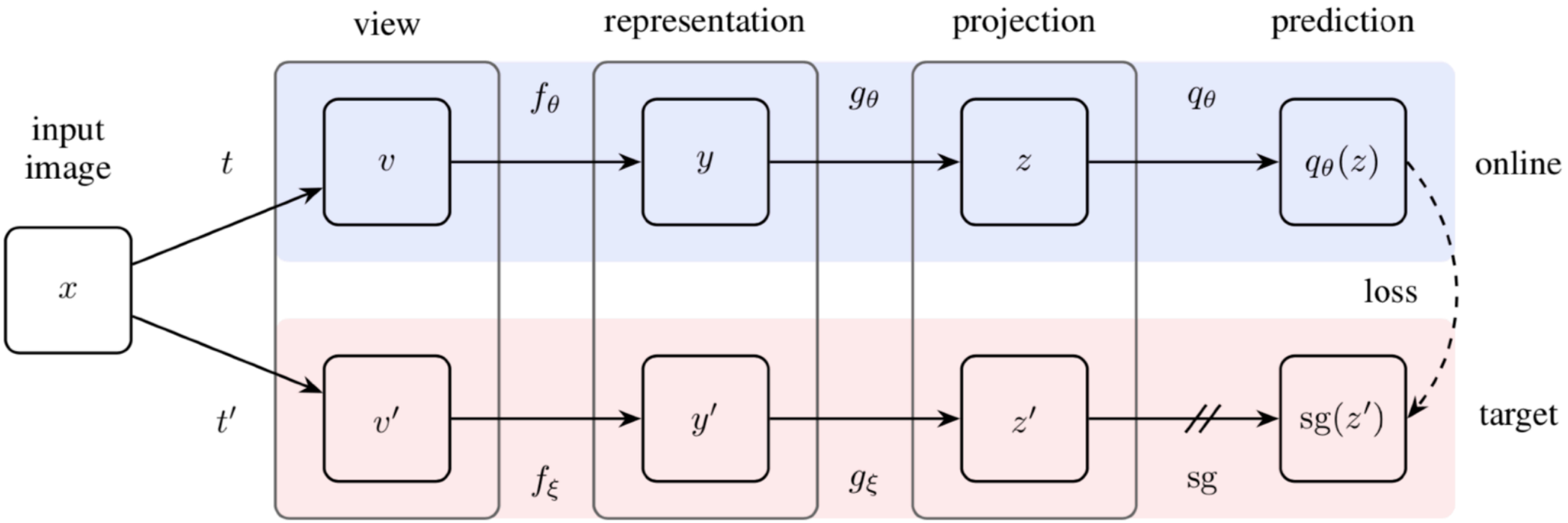
Total Loss

$$\mathcal{L}_\theta^{\mathrm{BYOL}} + \widetilde{\mathcal{L}}_\theta^{\mathrm{BYOL}}$$

```
online_network_out_1 = net_apply(params=online_params, inputs=image_1)
online_network_out_2 = net_apply(params=online_params, inputs=image_2)
target_network_out_1 = net_apply(params=target_params, inputs=image_1)
target_network_out_2 = net_apply(params=target_params, inputs=image_2)

def regression_loss(x, y):
  norm_x, norm_y = jnp.linalg.norm(x), jnp.linalg.norm(y)
  return -2. * jnp.sum(x * y, axis=-1) / (norm_x * norm_y)

# The stop_gradient is not necessary as we explicitly take the gradient with
# respect to online parameters only. We leave it to indicate that gradients
# are not backpropagated through the target network.
loss = regression_loss(online_network_out_1['prediction'],
                       jax.lax.stop_gradient(target_network_out_2['projection']))
loss += regression_loss(online_network_out_2['prediction'],
                        jax.lax.stop_gradient(target_network_out_1['projection']))
return loss
```

Summary

- ✔ Use two neural networks.
- ✔ EMA
- ✔ MSE loss



Result

- Linear evaluation on ImageNet
- We first evaluate BYOL's representation by training a linear classifier on top of the frozen representation

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.

Result

- Semi-supervised training on ImageNet
- We evaluate the performance obtained when fine-tuning BYOL's representation on a classification task with a small subset of ImageNet's train set, this time using label information.

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.

Result

- Transfer to other classification tasks

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.

Thanks