Lab 7

Optimization
Self-Supervised Learning

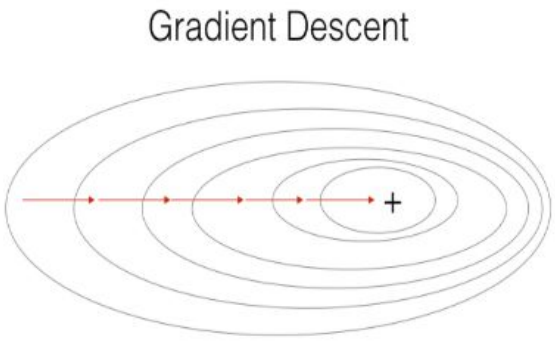
Chaojie Zhang

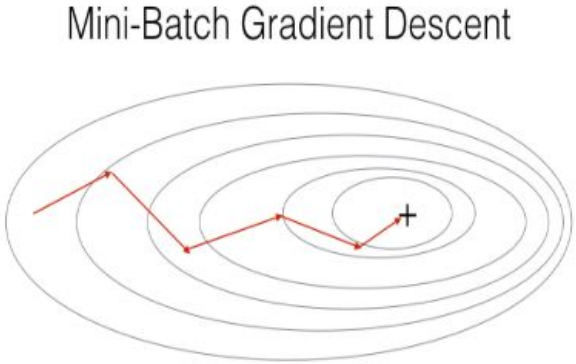
BGD, MBGD, SGD

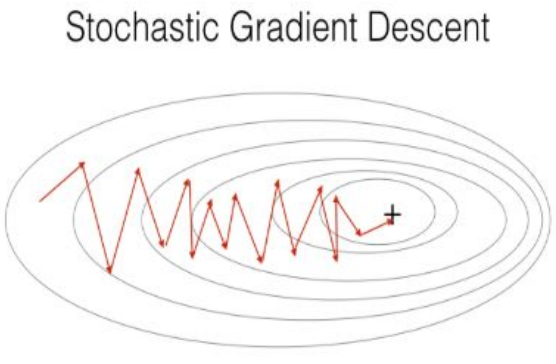
(Batch) Gradient Descent - whole dataset

Mini-Batch Gradient descent - part of the dataset

Stochastic Gradient Descent - one sample







SGD

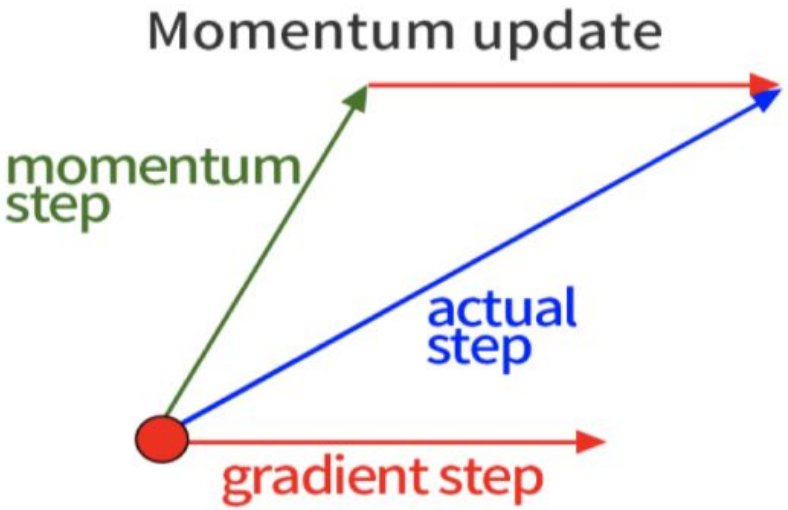
https://pytorch.org/docs/stable/generated/torch.optim.SGD.html

optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

torch.optim.SGD can be Gradient Descent, Mini-Batch Gradient descent or Stochastic Gradient Descent depend on the batch size.

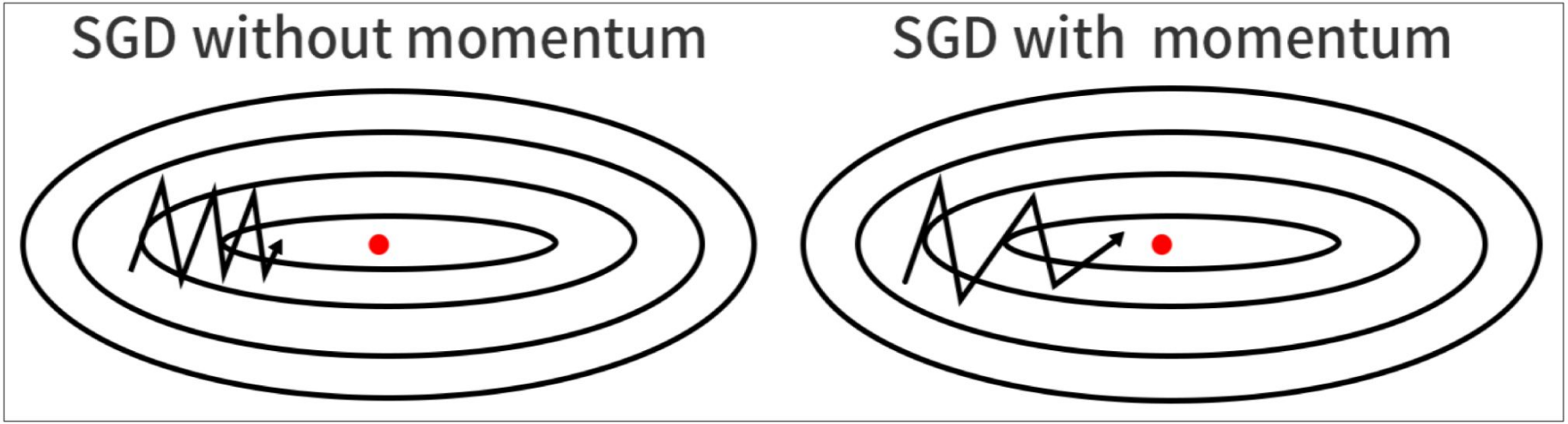
SGD with momentum

optimizer = torch.optim.SGD(model.parameters(), lr=0.1, momentum=0.9)
actual step = momentum x previous step + gradient step



https://iopscience.iop.org/article/10.1088/1742-6596/1229/1/012046/pdf

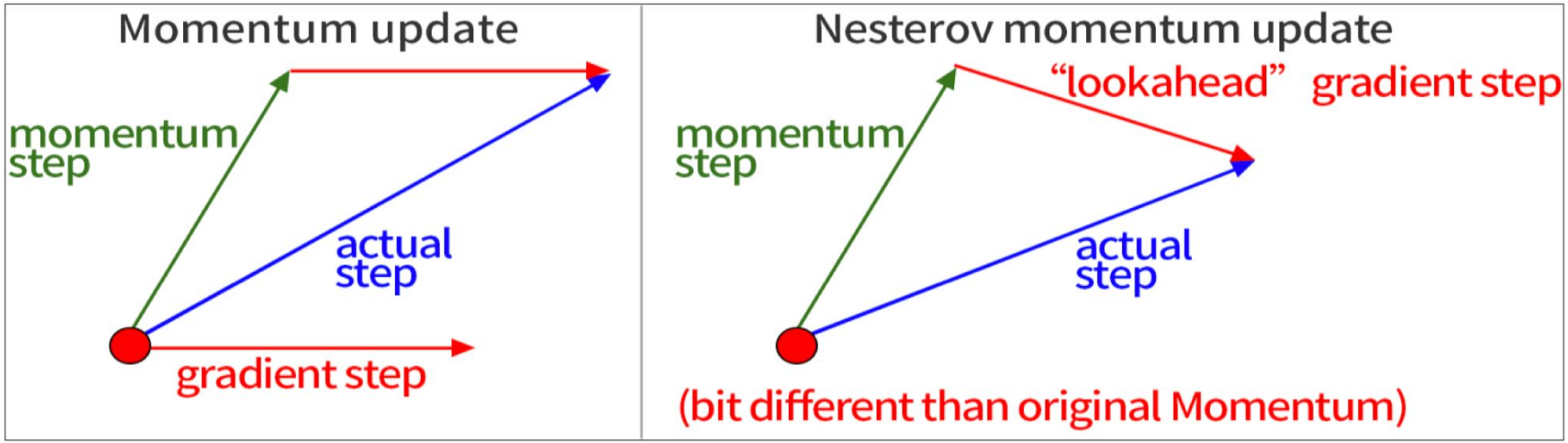
SGD with momentum



https://iopscience.iop.org/article/10.1088/1742-6596/1229/1/012046/pdf

SGD with Nesterov momentum

optimizer = torch.optim.SGD(model.parameters(), lr=0.1, momentum=0.9, nesterov=True)
actual step: momentum x previous step, then gradient step at that new position



https://iopscience.iop.org/article/10.1088/1742-6596/1229/1/012046/pdf

AdaGrad

$$V_t = \sum g_t^2$$

This is an Adaptive Subgradient Method. Simplified formula:

$$\eta_t = \frac{\eta}{\varepsilon + \sqrt{V_t}}$$

Gradient estimate Vt is the sum of squares of all previous gradient

ε is a very small number to avoid 0 denominator

η is learning rate, ηt is current learning rate

Some parameters are frequently updated (large Vt), they may fit the data very well, so we decrease the learning rate for these parameters (small ηt).

Same, rare update -> small Vt -> large learning rate ηt .

Problem: gradient estimate Vt keeps increasing, learning rate may get close to 0 before it arrives the minimum.

RMSprop

RMSprop mainly uses the gradient in several previous steps.

Gradient estimate Vt won’t keep increasing, then we won’t get 0 learning rate.

RMSprop

$$V_t = \beta_2 V_{t-1} + (1 - \beta_2) g_t^2$$

$$\eta_t = \frac{\eta}{\varepsilon + \sqrt{V_t}}$$

AdaGrad

$$V_t = \sum g_t^2$$

$$\eta_t = \frac{\eta}{\varepsilon + \sqrt{V_t}}$$

Adam

Adam = SGD momentum + RMSprop

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

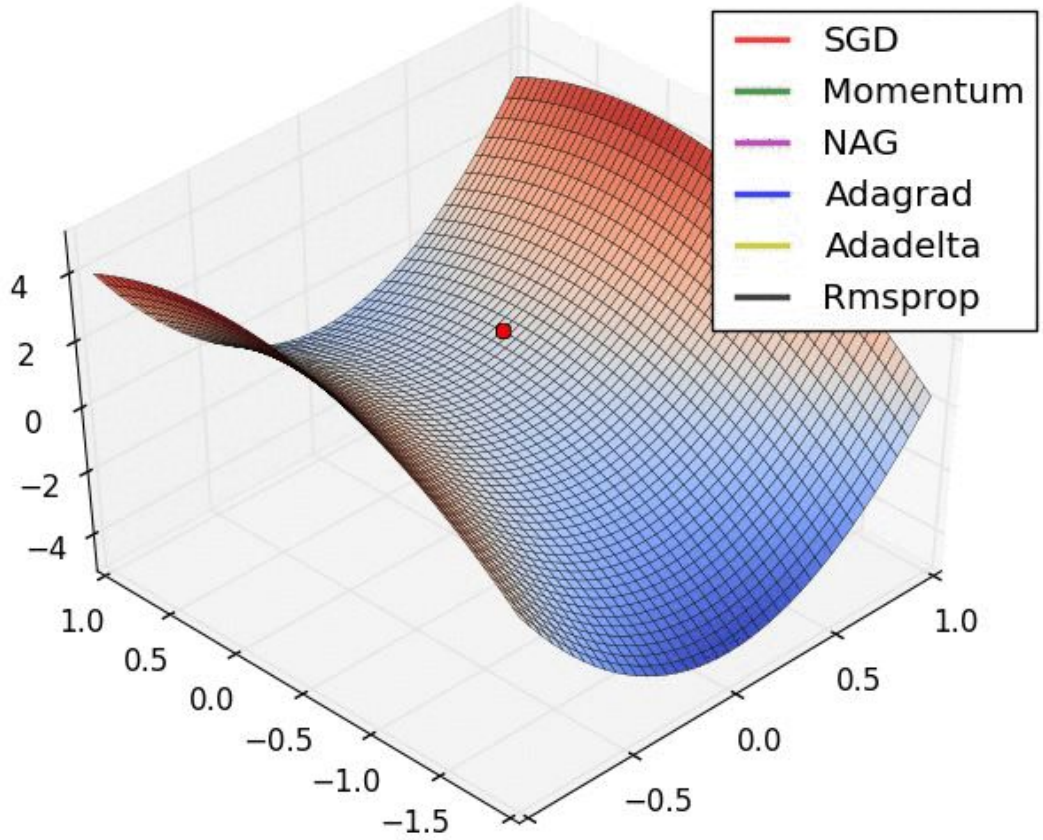
$$V_t = \beta_2 V_{t-1} + (1 - \beta_2) g_t^2$$

$$\bar{m}_t = \frac{m_t}{1 - \beta_1^t}$$

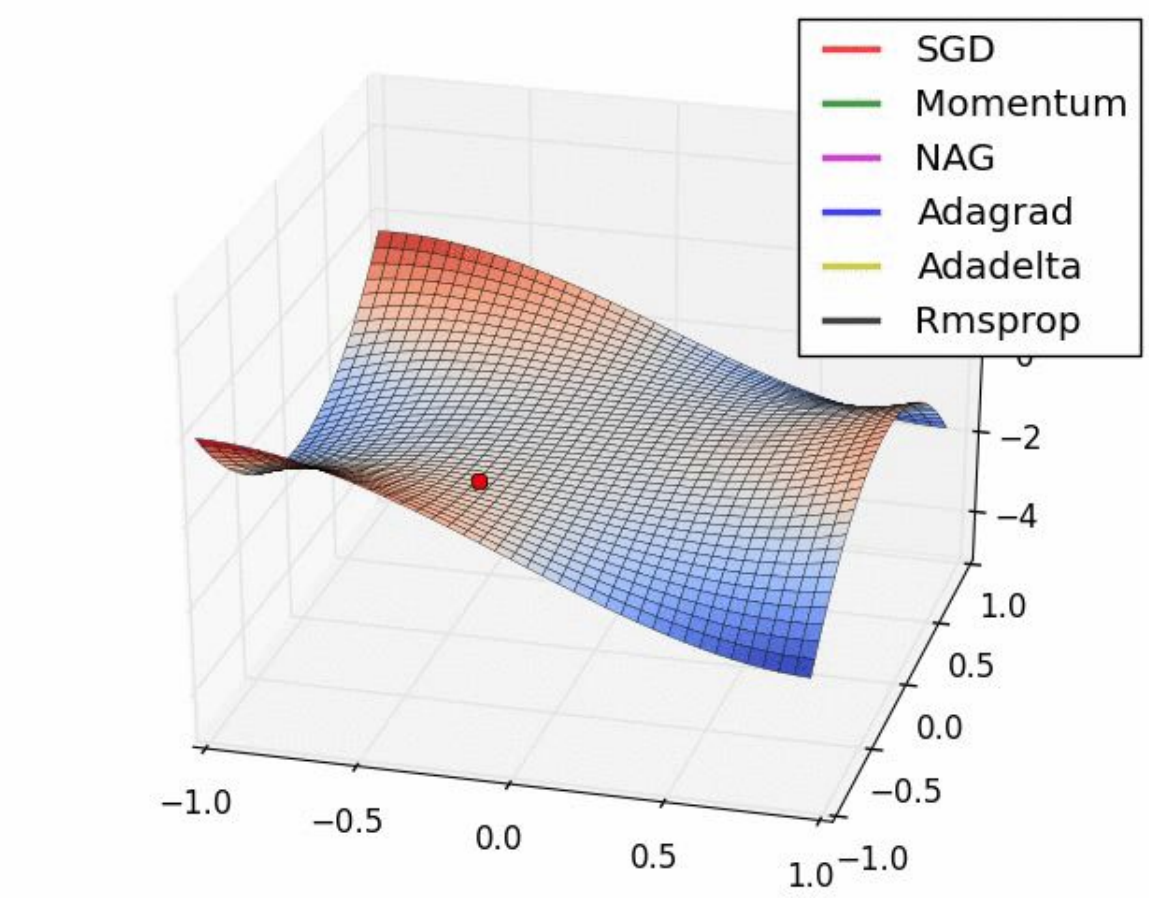
$$\bar{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha \bar{m}_t}{\sqrt{v_t} + \varepsilon}$$

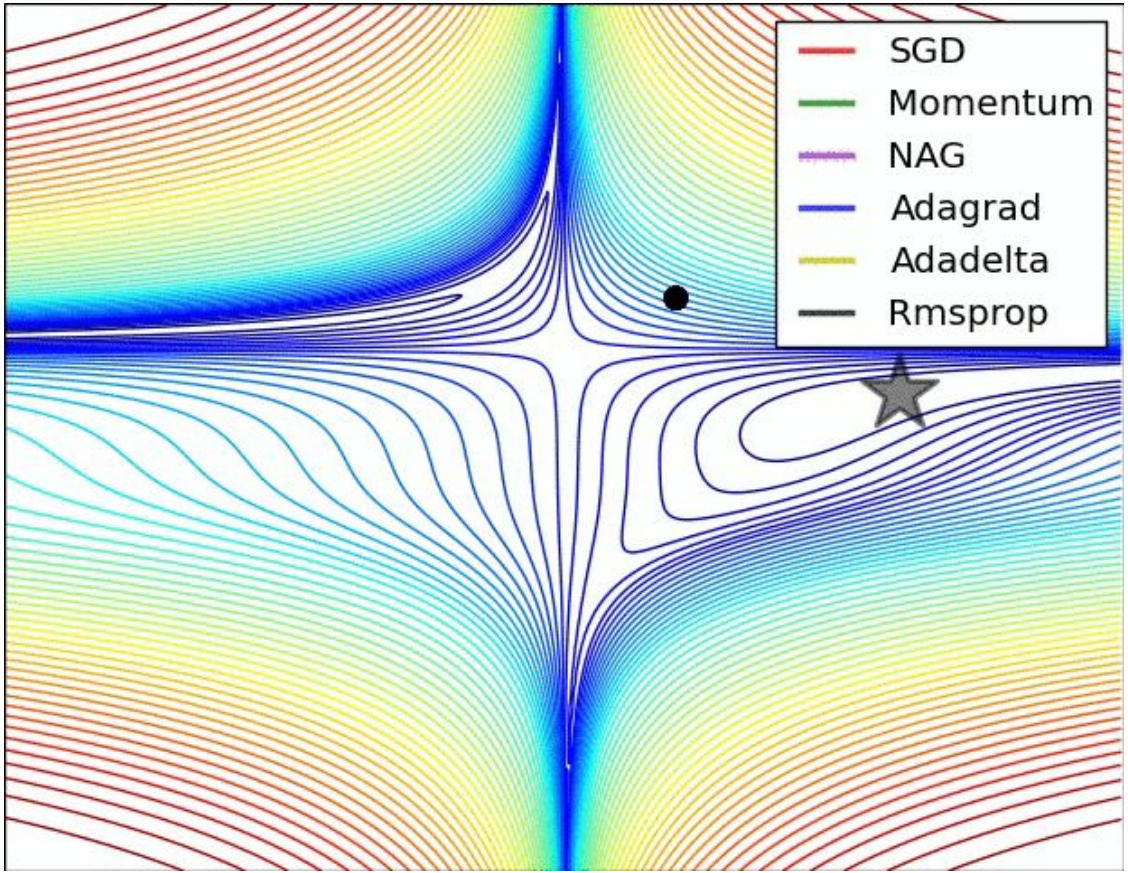
SGD and momentum stuck at the saddle point.



SGD momentum has increasing accelerated speed.



SGD and momentum converge slower.



Practical Application

- Adam converges fast.
- SGD usually ends up with better result, but takes longer to train.
- Adam+SGD: use Adam first, then use SGD for fine-tuning.

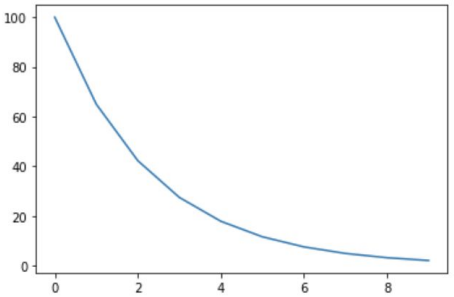
Scheduler: Learning Rate Scheduling

1. LAMBDA LR

Sets the learning rate of each parameter group to the initial lr times a given function. When last_epoch=-1, sets initial lr as lr.

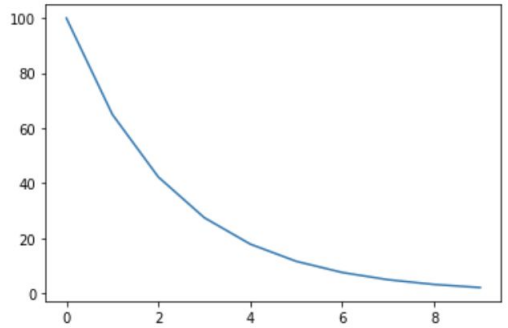
$$lr_{\text{epoch}} = lr_{\text{initial}} * Lambda(epoch)$$

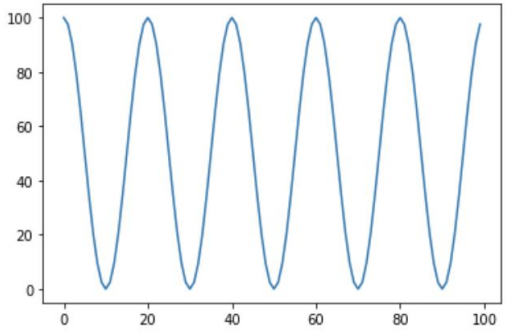
```
model = torch.nn.Linear(2, 1)
optimizer = torch.optim.SGD(model.parameters(), lr=100)
lambda1 = lambda epoch: 0.65 ** epoch
scheduler = torch.optim.lr_scheduler.LambdaLR(optimizer, lr_lambda=lambda1)


lrs = []

for i in range(10):
    optimizer.step()
    lrs.append(optimizer.param_groups[0]["lr"])
#     print("Factor = ", round(0.65 ** i,3)," , Learning Rate = ",round(optimizer.param_groups[0]
["lr"],3))
    scheduler.step()

plt.plot(range(10),lrs)
```

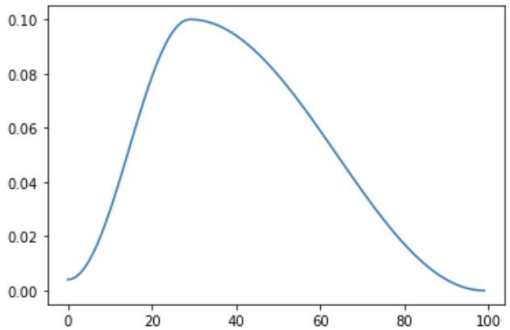


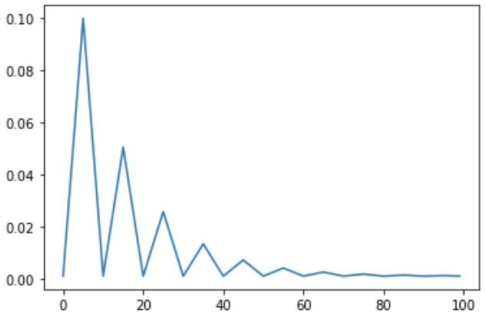
Scheduler

https://www.kaggle.com/code/isbhargav/guide-to-pytorch-learning-rate-scheduling/notebook

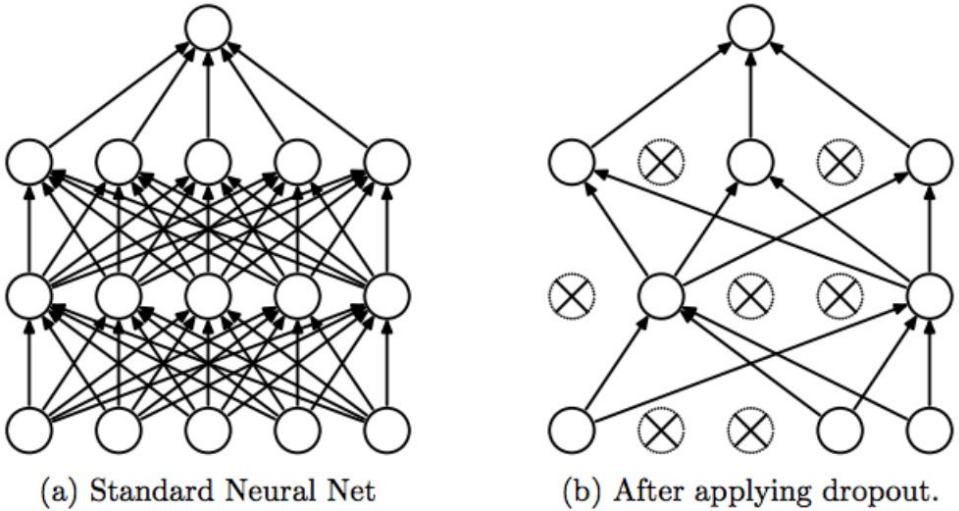








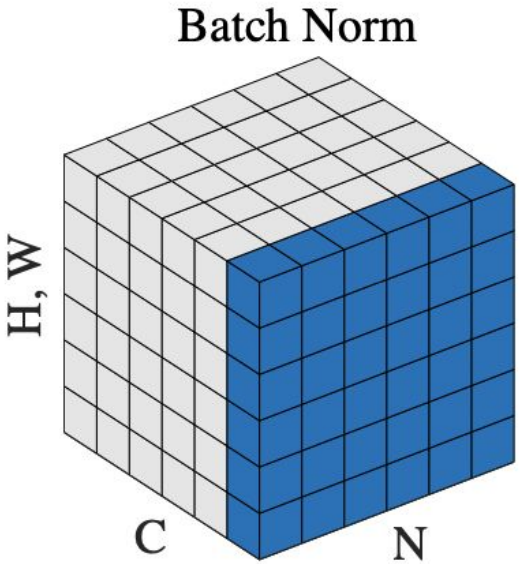
Dropout: https://towardsdatascience.com/machine-learning-part-20-dropout-keras-layers-explained-8c9f6dc4c9ab



(a) Standard Neural Net

(b) After applying dropout.

Batch Normalization: Normalize among N (batch), H, W, not including C (channel)



Self-Supervised Learning

Self-supervised learning

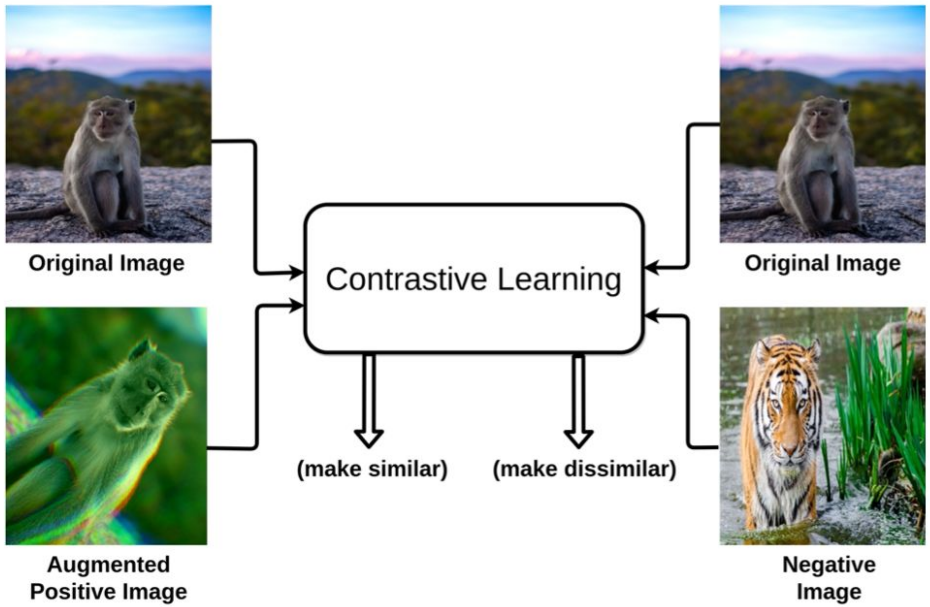
Self-supervised learning helps the model learn from unlabeled sample data.

E.g. 1000 unlabeled images + 100 labeled images

- Use unlabeled images for self-supervised pre-training
- Use labeled images for supervised fine-tuning

Contrastive Learning

Basic intuition behind contrastive learning paradigm: push original and augmented images closer and push original and negative images away.



Augmentations

(a) Original

(b) Crop and resize

(c) Crop, resize (and flip)

(d) Color distort. (drop)

(e) Color distort. (jitter)

(f) Rotate $\{90°, 180°, 270°\}$

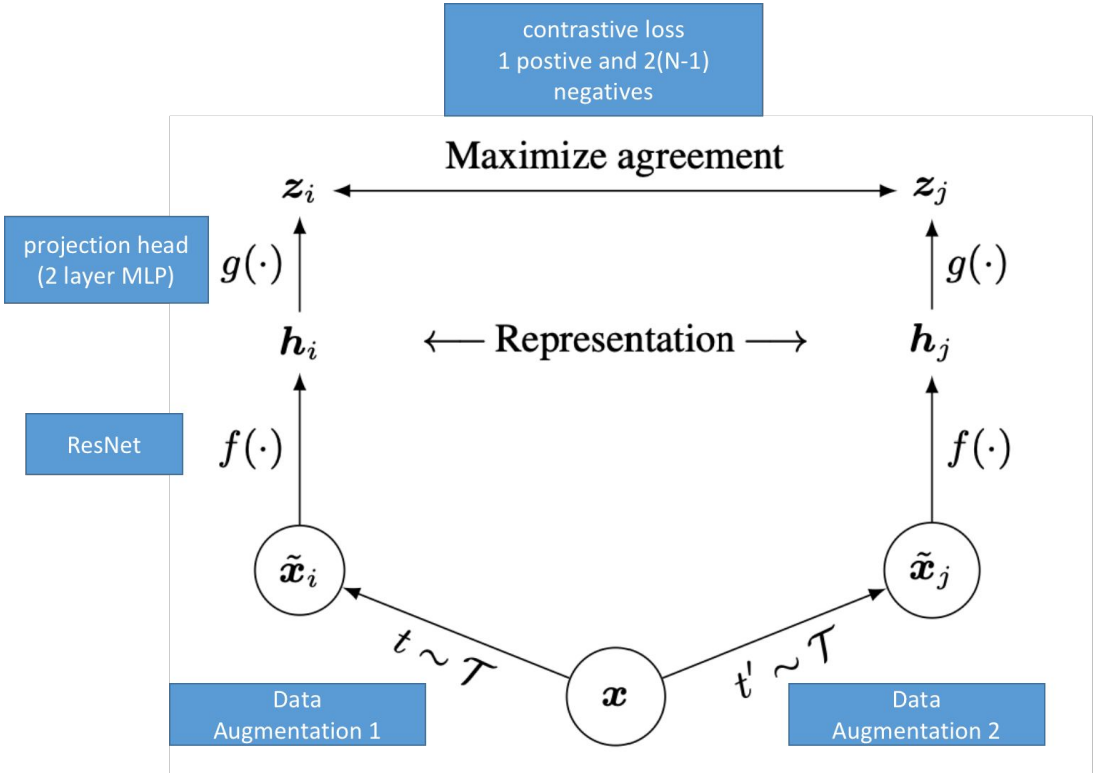
(g) Cutout

(h) Gaussian noise

(i) Gaussian blur

(j) Sobel filtering

SimCLR



Batch size = 4

Image 1
Image 2
Image 3
Image 4

two
augmentations

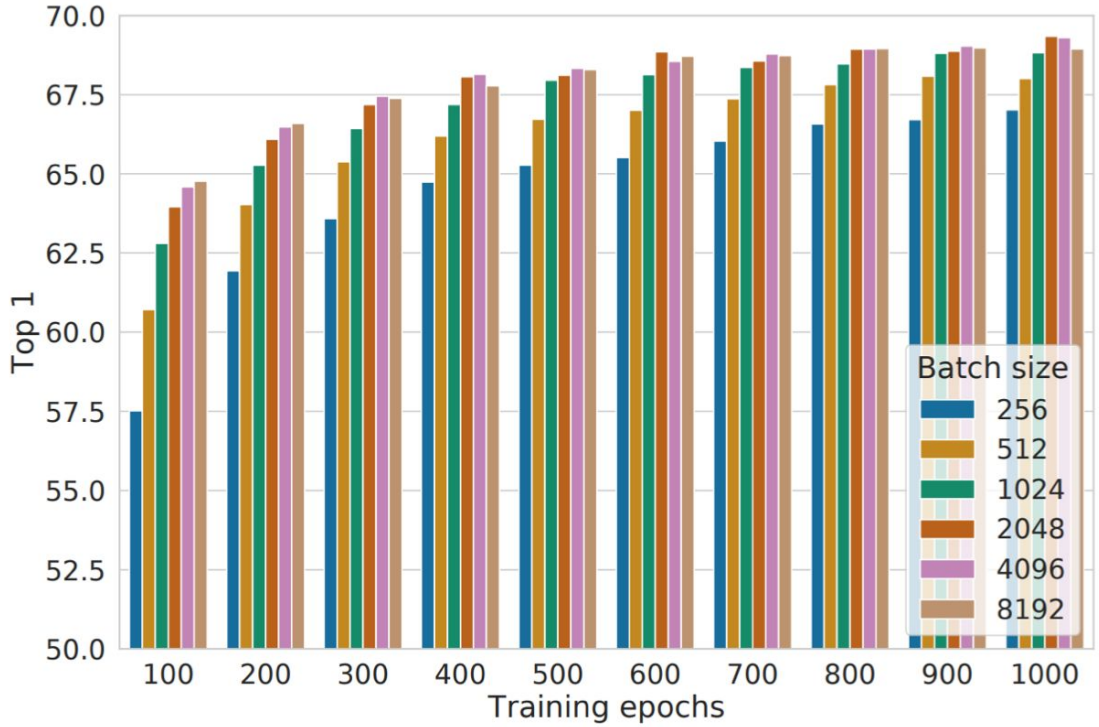
Image1_AugA
Image2_AugA
Image3_AugA
Image4_AugA

Image1_AugB
Image2_AugB
Image3_AugB
Image4_AugB

SimCLR

- Composition of data augmentations plays a critical role in defining effective predictive tasks.
- Contrastive learning benefits from larger batch sizes and more training steps compared to supervised learning.
- SimCLR used a batch size of 4096 for 100 epochs. It has been verified that end-to-end architectures are simple in complexity, but perform better with large batch sizes and a higher number of epochs as represented.
- The number of negative samples available in this approach is coupled with the batch size as it accumulates negative samples from the current batch. As the batch size is limited by the GPU memory size, the scalability factor with these methods remains an issue. Furthermore, for larger batch sizes, the methods suffer from a large mini-batch optimization problem and require effective optimization.

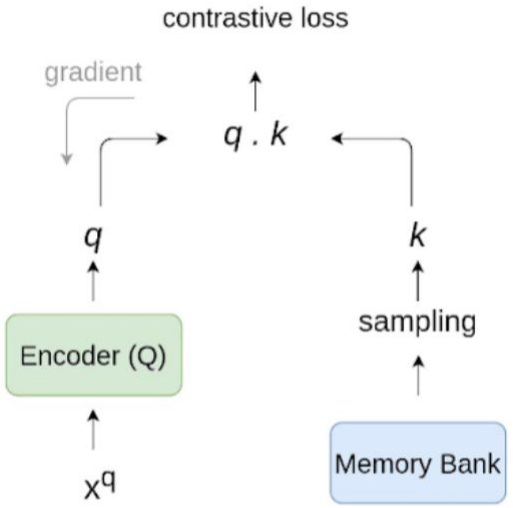
SimCLR



Problems: number of negatives rely on batch size => large GPU

Memory Bank

- With potential issues of having larger batch sizes that could inversely impact the optimization during training, a possible solution is to maintain a separate dictionary called Memory bank.
- The memory bank (M) contains a feature representation m for each sample I. The representation m is an exponential moving average of feature representations that were computed in prior epochs.
- The representation of a sample in the memory bank gets updated when it is last seen, so the sampled keys are essentially about the encoders at multiple different steps all over the past epoch.
- However, maintaining a memory bank during training can be a complicated task. One of the potential drawbacks of this approach is that it can be computationally expensive to update the representations in the memory bank as the representations get outdated quickly in a few passes.



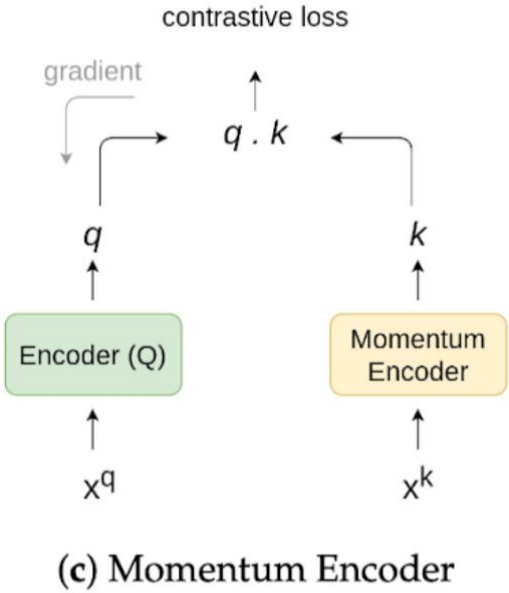
(b) Memory Bank

MoCo

Momentum encoder update: $\theta_{\mathrm{k}} \leftarrow m\theta_{\mathrm{k}} + (1-m)\theta_{\mathrm{q}}$

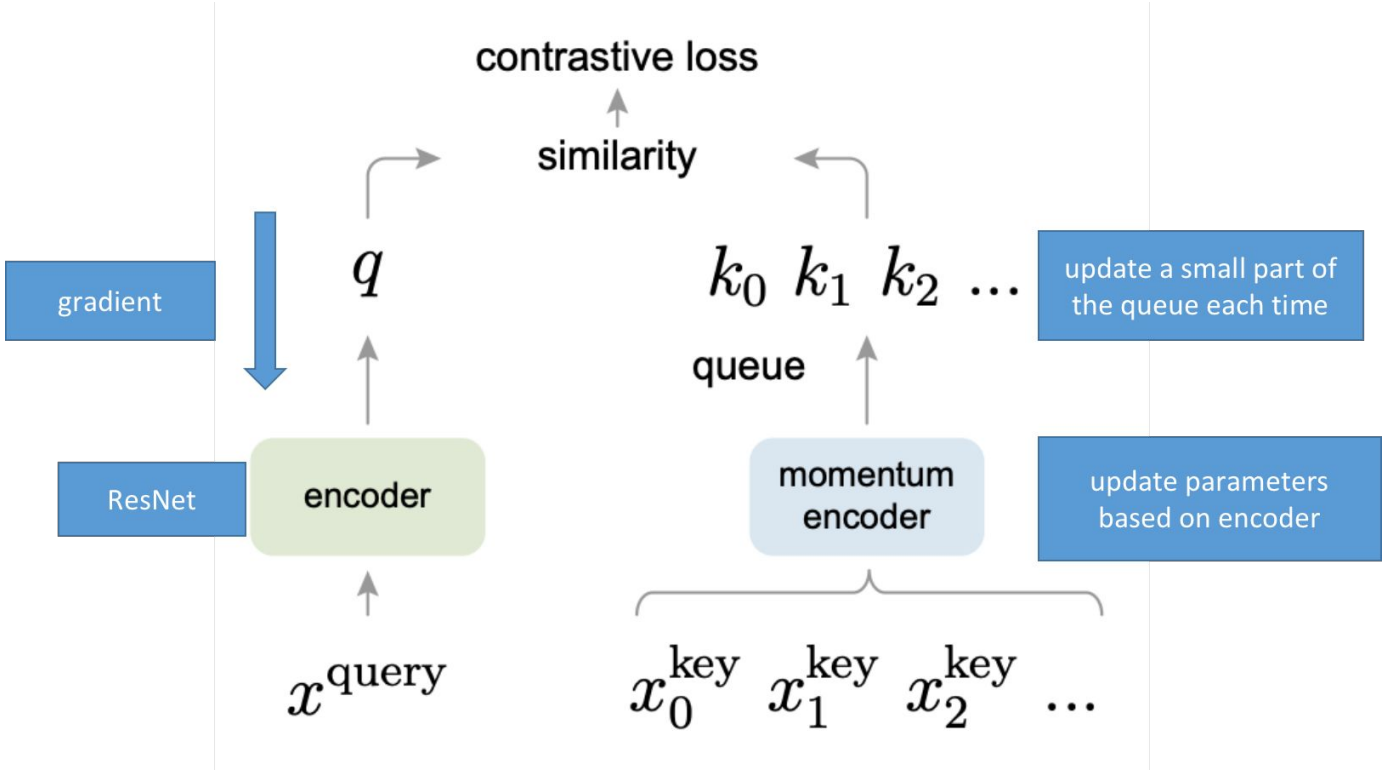
Initial: Encoder and Momentum Encoder share the same structure and parameters

- If the momentum encoder doesn't change, the gradient only propagates to one sample.
- If the momentum encoder copies the parameters from the encoder, the solution yields poor results. The authors hypothesize that such failure is caused by the rapidly changing encoder that reduces the key representations' consistency. (may be wrong)
- A momentum update is a good solution.



(c) Momentum Encoder

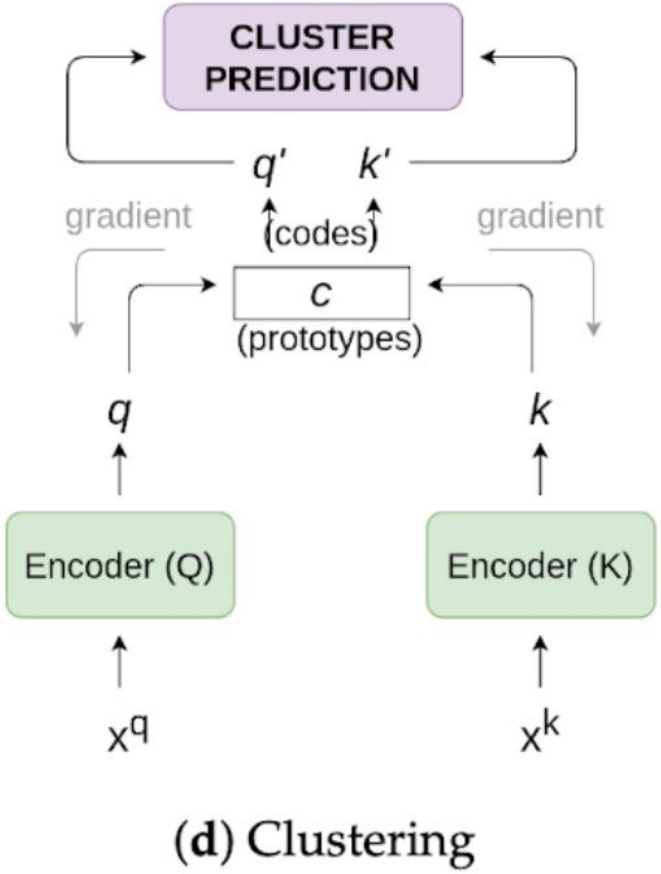
MoCo


contrastive loss
similarity
q
k_0 k_1 k_2 ...
queue
gradient
update a small part of the queue each time
ResNet
encoder
momentum encoder
update parameters based on encoder
x^{query}
x_0^{key} x_1^{key} x_2^{key} ...

Clustering (SwAV)

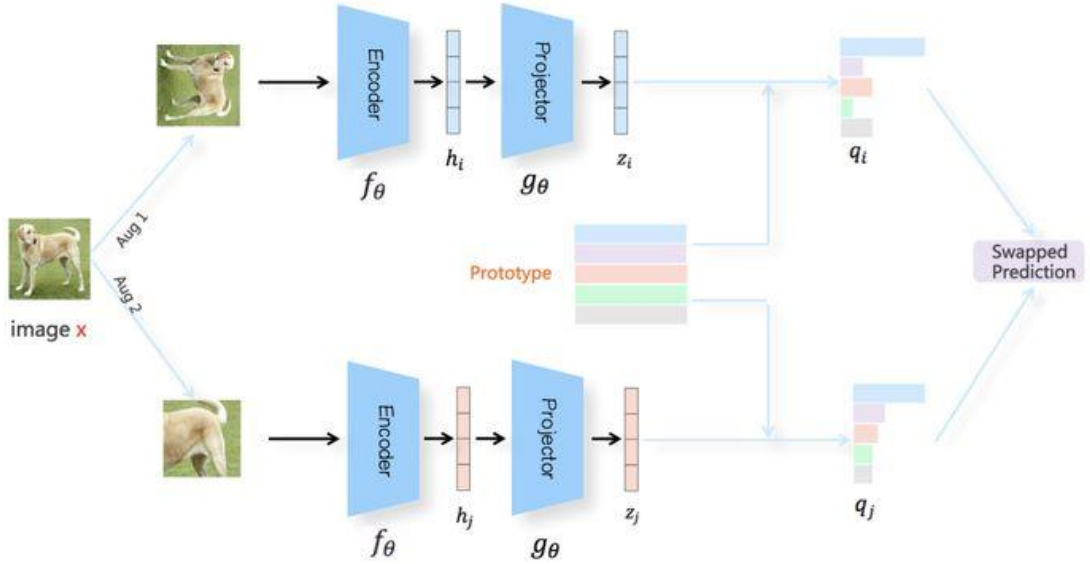
Here, the goal is not only to make a pair of samples close to each other but also, make sure that all other features that are similar to each other form clusters together.

For example, in an embedded space of images, the features of cats should be closer to the features of dogs (as both are animals) but should be far from the features of houses (as both are distinct).



(d) Clustering

- Instead of using an instance-based contrastive approach, SwAV utilizes a clustering algorithm to group similar features together.
- Do clustering in each batch.
- Use a different loss function to maximize the similarity between the features and the prototypes.
- Not only the similar views get closer, the similar types also get closer.



SwAV: Unsupervised Learning of Visual Features by Contrasting Cluster Assignments

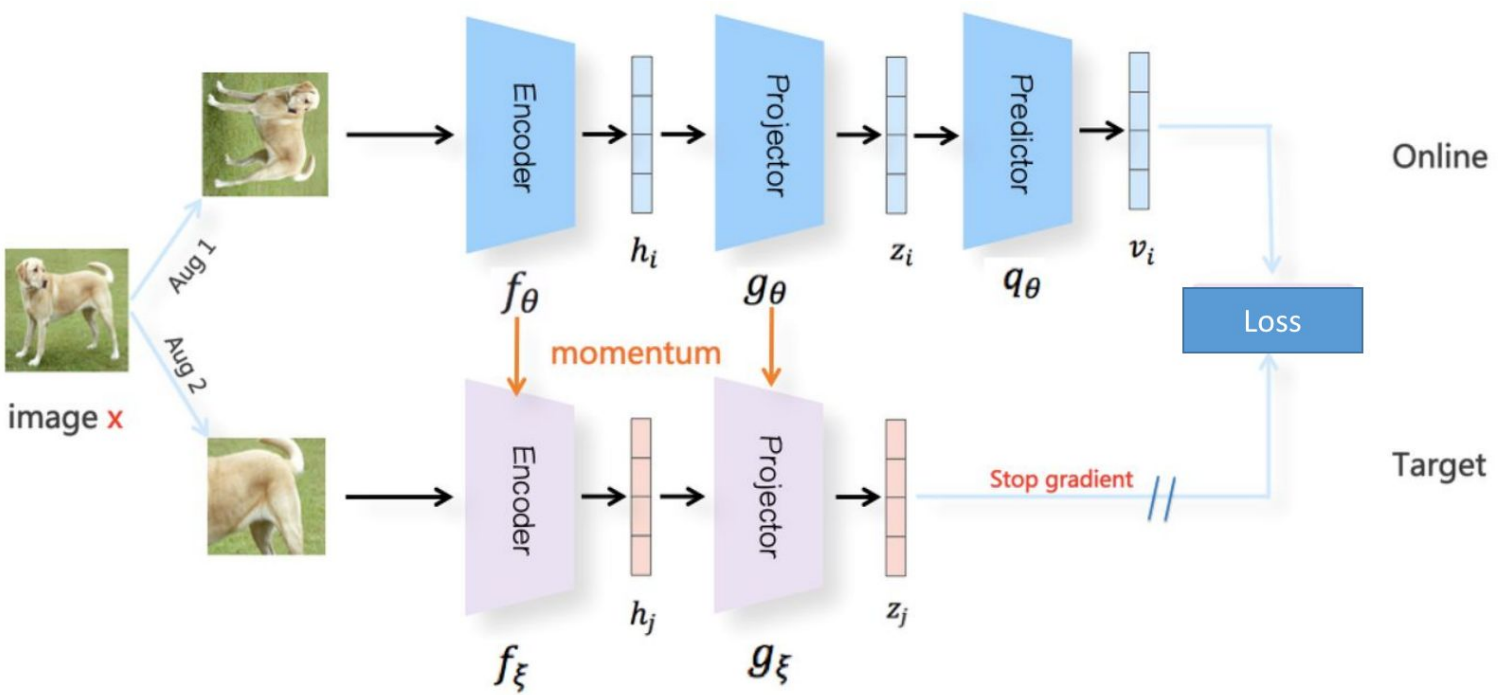
Negative pairs or not?

- Negative pairs:
  MoCo, SimCLR

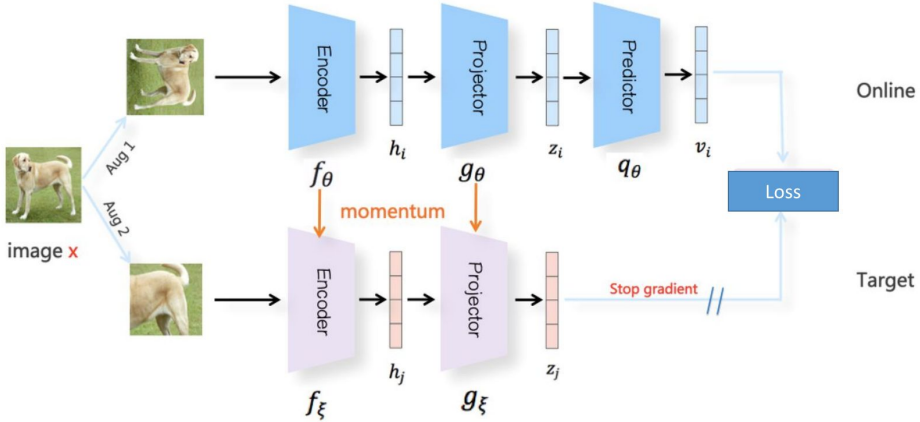
- Clustering:
  SwAV

- Only use positive samples:
  BYOL, SimSiam

BYOL



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



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

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

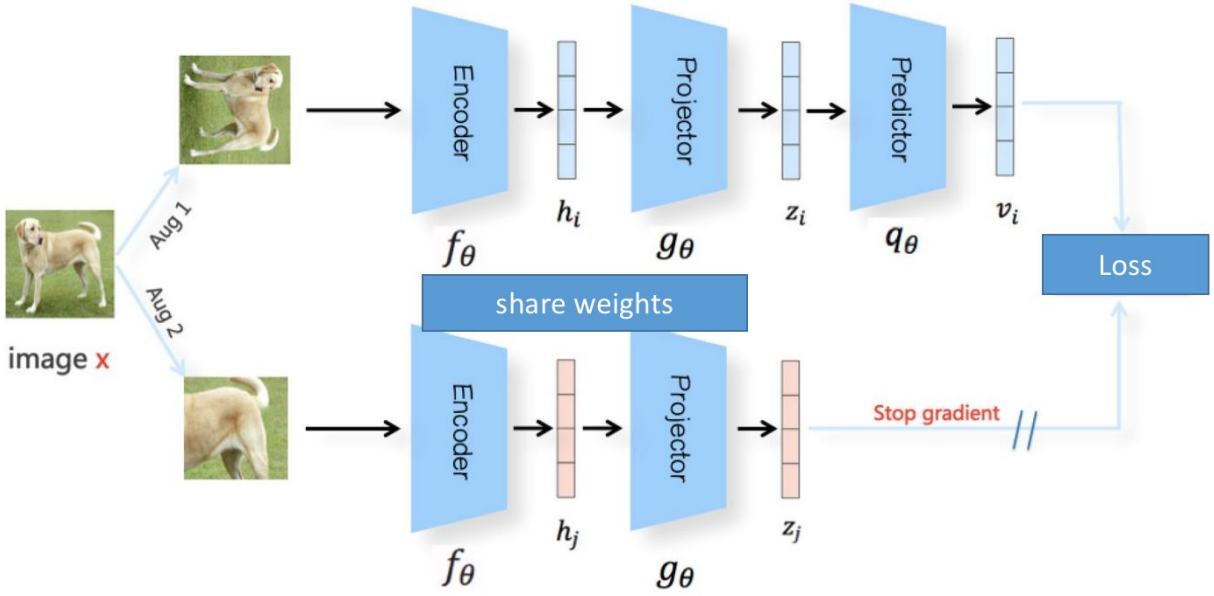
Target	τ_{base}	Top-1
Constant random network	1	$18.8_{\pm 0.7}$
Moving average of online	0.999	69.8
Moving average of online	0.99	**72.5**
Moving average of online	0.9	68.4
Stop gradient of online†	0	0.3

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

- The predictor makes the model asymmetric. The predictor is important, it gives the online model more flexibility.
- There are no negative samples used in BYOL, that means it will only pull the similar views closer.
- Opinions: EMA (exponential moving average) is helping scatter features. The initialization of the model is scattering the features. When τ is large, the target network updates slowly, then different features keep scattered.

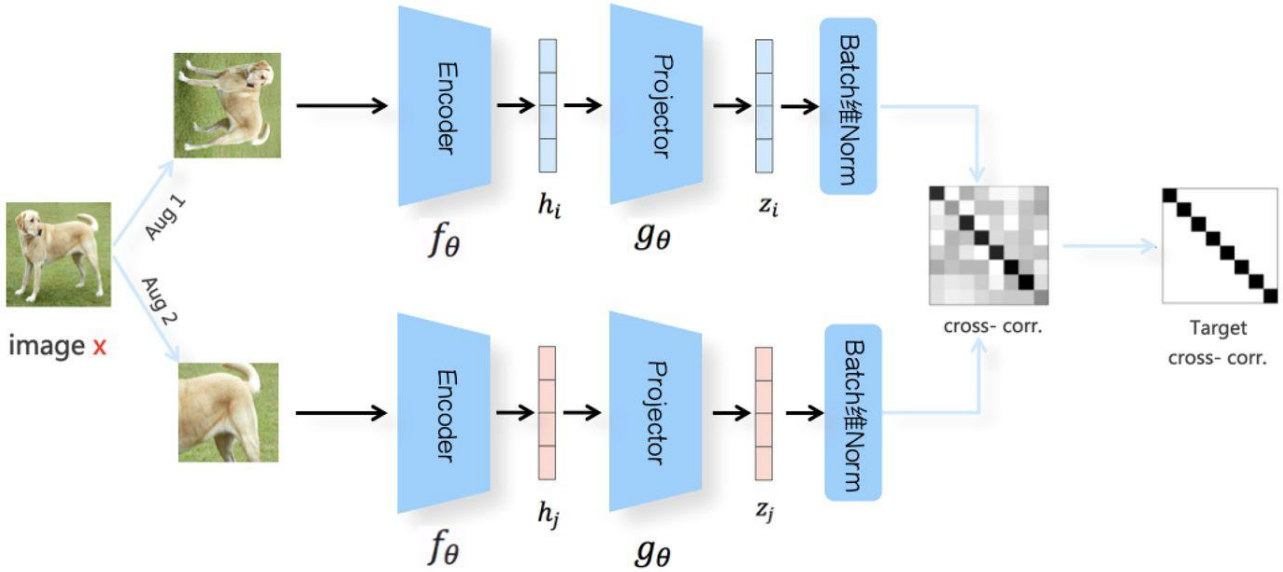
SimSiam

- Remove momentum from BYOL.
- BYOL is better than SimSiam.
- Momentum is not the key factor for uniformity (scatter features) or preventing collapse.



SimSiam: Exploring Simple Siamese Representation Learning

Barlow Twins



Barlow Twins: Self-Supervised Learning via Redundancy Reduction

Do cross-correlation in batches.

Loss: $\mathcal{L}_{\mathcal{BT}} \triangleq \underbrace{\sum_i (1 - \mathcal{C}_{ii})^2}_{\text{invariance term}} + \lambda \underbrace{\sum_i \sum_{j \neq i} \mathcal{C}_{ij}^2}_{\text{redundancy reduction term}}$

TWIST

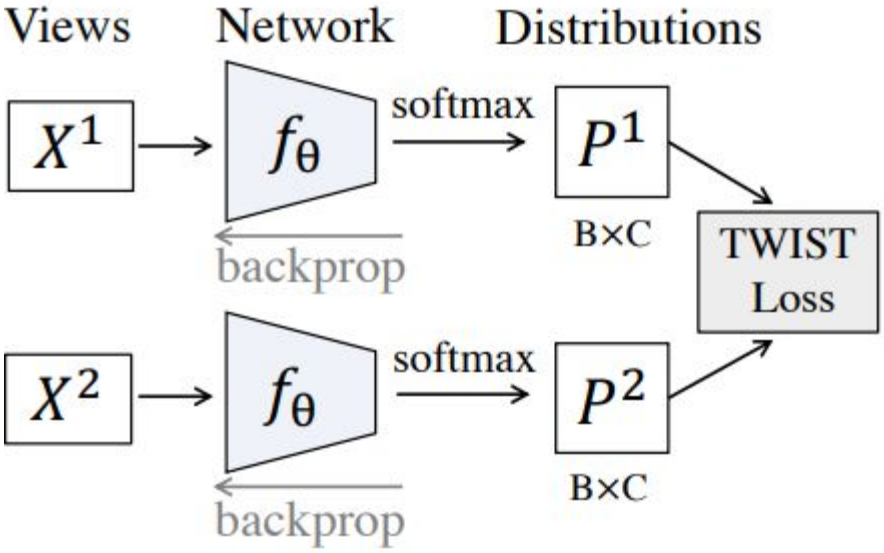


Figure 2: Network architecture of TWIST.

B: Batch Size
C: Number of Class
In the paper, C = 1000,
B = 2048

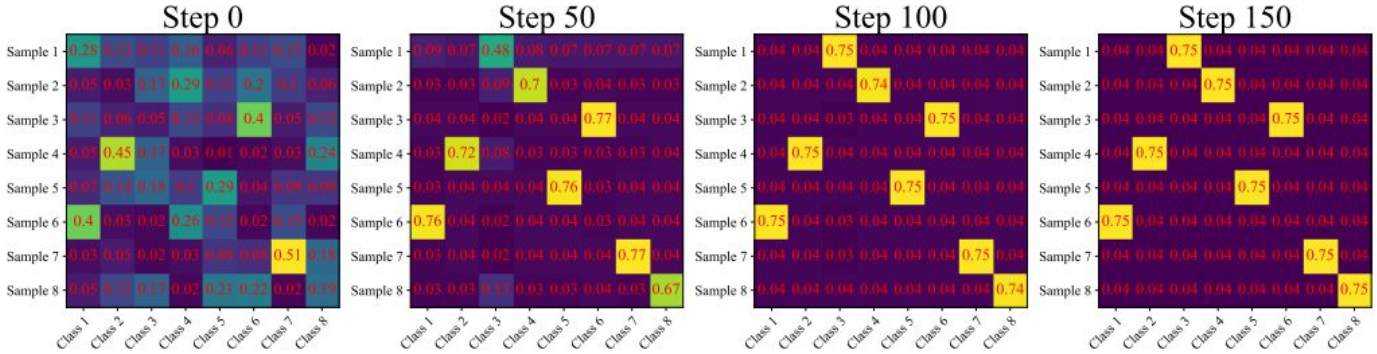


Figure 5: Evolution of the predictions for one view. We set $B = C = 8$ to give a clear demonstration.

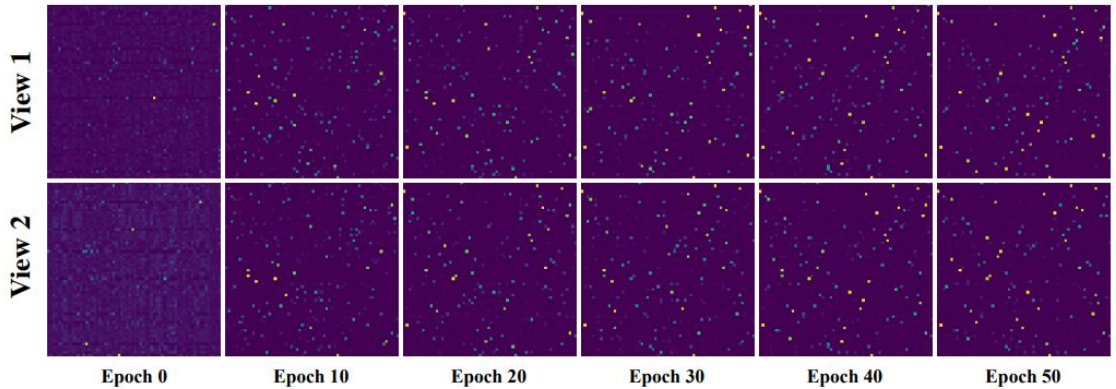


Figure 6: Evolution of the predictions for both two views. We set $B = C = 64$.

$$\mathcal{L}(P^1, P^2) = \frac{1}{2}(\mathcal{L}(P^1||P^2) + \mathcal{L}(P^2||P^1)), \tag{1}$$

where

$$\mathcal{L}(P^1||P^2) = \underbrace{\frac{1}{B}\sum_{i=1}^{B} D_{KL}(P_i^1||P_i^2)}_{\text{consistency term}} + \underbrace{\frac{1}{B}\sum_{i=1}^{B} H(P_i^1)}_{\text{sharpness term}} - \underbrace{H(\frac{1}{B}\sum_{i=1}^{B} P_i^1)}_{\text{diversity term}}, \tag{2}$$

Make View1 and View2 closer

Make the one-hot distribution sharp for each sample
H ~ uncertainty
lower H(P) ~ sharper

Only this relies on batch size

Avoid different samples go to only one class
H(sum P): if all samples in a batch go to one class, this will be high

D_KL(·||·) denotes the Kullback–Leibler divergence between two probability distributions. H(·) denotes the Entropy of a specific probability distribution.

$$\mathcal{L}(P^1||P^2) = \frac{1}{B}\sum_{i=1}^{B} CE(P_i^1, P_i^2) - H(\frac{1}{B}\sum_{i=1}^{B} P_i^1),$$

Comparison

Table 1. Top self-supervised models beat the supervised pre-training baseline on popular many-shot recognition datasets, both in linear evaluation and when finetuning. The top half of the table shows results from linear transfer of pre-trained models using logistic regression, and the bottom half shows the results when these models are finetuned. We also include the ImageNet linear evaluation performance (logistic regression or SGD) reported by the authors. Results style: **best**, second best.

		ImageNet	Aircraft	Caltech101	Cars	CIFAR10	CIFAR100	DTD	Flowers	Food	Pets	SUN397	VOC2007	Avg.
Linear	InsDis	59.50	36.87	71.12	28.98	80.28	59.97	68.46	83.44	63.39	68.78	49.47	74.37	62.29
	MoCo-v1	60.60	35.55	75.33	27.99	80.16	57.71	68.83	82.10	62.10	69.84	51.02	75.93	62.41
	PCL-v1	61.50	21.61	76.90	12.93	81.84	55.74	62.87	64.73	48.02	75.34	45.70	78.31	56.73
	PIRL	61.70	37.08	74.48	28.72	82.53	61.26	68.99	83.60	64.65	71.36	53.89	76.61	63.92
	PCL-v2	67.60	37.03	86.42	30.51	91.91	73.54	70.59	85.34	64.88	82.79	56.25	81.14	69.13
	SimCLR-v1	69.30	44.90	90.05	43.73	91.18	72.73	74.20	90.87	67.47	83.33	59.21	80.77	72.59
	MoCo-v2	71.10	41.79	87.92	39.31	92.28	74.90	73.88	90.07	68.95	83.30	60.32	82.69	72.31
	SimCLR-v2	71.70	46.38	89.63	50.37	92.53	76.78	76.38	92.90	73.08	84.72	61.47	81.57	75.07
	SeLa-v2	71.80	37.29	87.20	36.86	92.73	74.81	74.15	90.22	71.08	83.22	62.71	82.73	72.09
	InfoMin	73.00	38.58	87.84	41.04	91.49	73.43	74.73	87.18	69.53	86.24	61.00	83.24	72.21
	BYOL	74.30	53.87	**91.46**	56.40	93.26	77.86	76.91	94.50	73.01	89.10	59.99	81.14	77.05
	DeepCluster-v2	75.20	**54.49**	91.33	**58.60**	**94.02**	**79.61**	**78.62**	**94.72**	**77.94**	89.36	65.48	**83.94**	**78.92**
	SwAV	75.30	54.04	90.84	54.06	93.99	79.58	77.02	94.62	76.62	87.60	**65.58**	83.68	77.97
	Supervised	**77.20**	43.59	90.18	44.92	91.42	73.90	72.23	89.93	69.49	**91.45**	60.49	83.60	73.75
Finetune	InsDis		73.38	72.04	61.56	93.32	68.26	63.99	89.51	76.78	76.22	51.84	71.90	72.62
	MoCo-v1		75.61	74.95	65.02	93.89	71.52	65.37	89.45	77.28	76.96	53.35	74.91	74.39
	PCL-v1		74.97	87.62	73.24	96.35	79.62	70.00	90.83	78.30	86.98	58.40	82.08	79.85
	PIRL		72.68	70.83	61.02	92.23	66.48	64.26	89.81	74.96	76.26	50.38	69.90	71.71
	PCL-v2		79.37	88.04	71.68	96.50	80.26	71.76	92.95	80.34	85.39	58.82	82.20	80.66
	SimCLR-v1		81.06	90.35	83.78	**97.07**	84.53	71.54	93.75	82.40	84.10	63.31	82.58	83.13
	MoCo-v2		79.87	84.38	75.20	96.45	71.33	69.47	94.35	76.78	79.80	55.77	71.71	77.74
	SimCLR-v2		78.71	82.94	79.84	96.22	79.05	70.16	94.32	82.22	83.20	61.12	78.19	80.54
	SeLa-v2		81.99	88.99	85.62	96.80	84.37	74.36	**95.80**	86.24	88.55	65.84	84.85	84.86
	InfoMin		80.24	83.92	78.76	96.94	71.15	71.12	95.24	78.93	85.28	57.66	76.63	79.62
	BYOL		79.45	89.40	84.60	97.01	83.95	73.62	94.48	85.54	89.62	63.96	82.70	84.03
	DeepCluster-v2		82.52	90.75	**87.27**	97.06	**85.15**	74.84	95.31	**87.51**	89.43	**66.42**	**84.90**	**85.56**
	SwAV		83.08	89.85	86.76	96.78	84.37	**75.16**	95.46	87.22	89.05	66.24	84.66	85.33
	Supervised		**83.50**	**91.01**	82.61	96.39	82.91	73.30	95.50	84.60	**92.42**	63.56	84.76	84.60

MAE

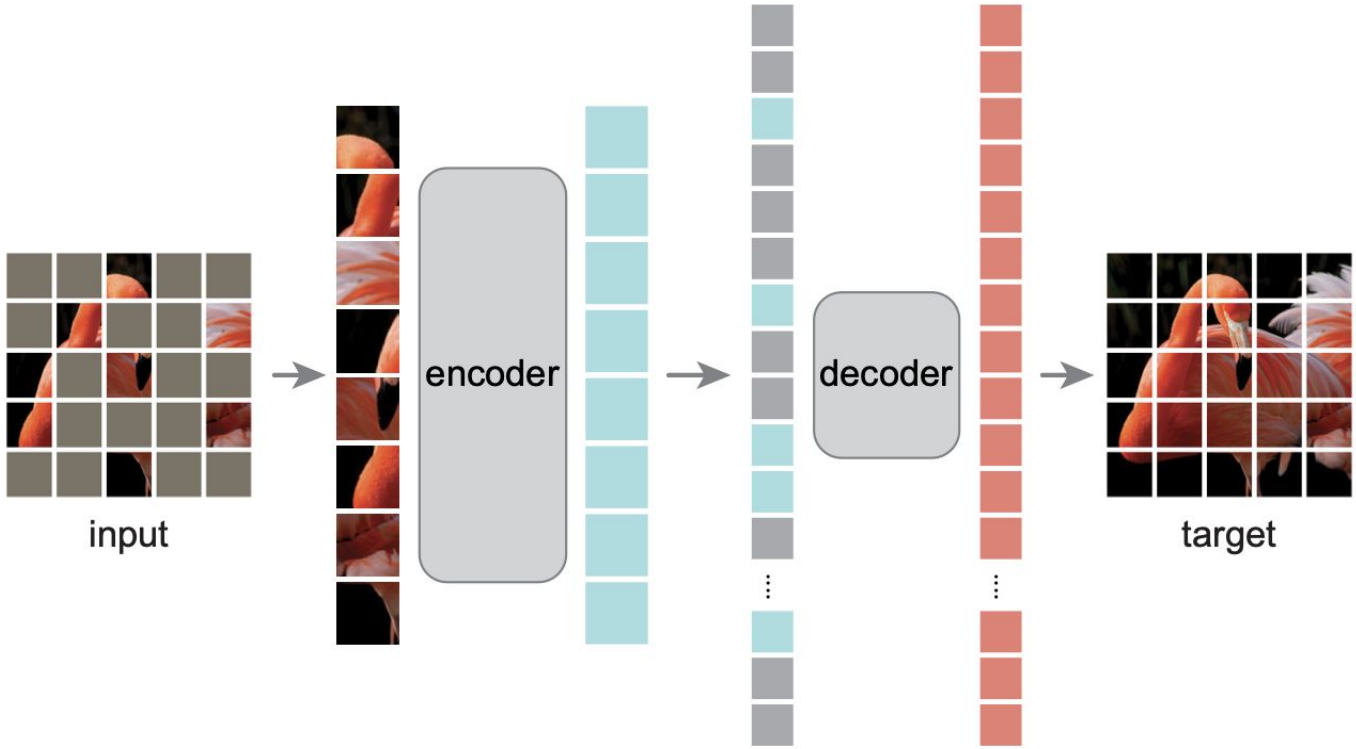


Figure 4. Reconstructions of ImageNet *validation* images using an MAE pre-trained with a masking ratio of 75% but applied on inputs with higher masking ratios. The predictions differ plausibly from the original images, showing that the method can generalize.

Reference

MoCo https://arxiv.org/abs/1911.05272
MoCo v2 https://arxiv.org/abs/2003.04297
MoCo v3 https://arxiv.org/abs/2104.02057
SimCLR https://arxiv.org/abs/2002.05709
SwAV https://arxiv.org/abs/2006.09882
BYOL https://arxiv.org/abs/2006.07733
SimSiam https://arxiv.org/abs/2011.10566
Barlow Twins https://arxiv.org/abs/2103.03230
TWIST https://openreview.net/forum?id=TLgW66V2CbP
MAE https://arxiv.org/abs/2111.06377
A Survey on Contrastive Self-Supervised Learning https://arxiv.org/abs/2011.00362
How Well Do Self-Supervised Models Transfer? https://arxiv.org/abs/2011.13377