Self-Supervised Learning for pre-training models in Computer Vision

-

Continual Learning Report

Diego Arcelli

May 12, 2023

UNIVERSITÀ DI PISA

Contents

1 Introduction **2**

2 Problem setting **2**

3 Methodology **3**

4 Analysis and comparison **10**

5 Strength and weaknesses **11**

6 Conclusion **11**

Abstract

In this work some self-supervised learning methods for computer vision will be explored. In the first part the general framework of self-supervised learning will be discussed, describing all the aspects which are in common to the methods that will be explained. After five different self-supervised learning methods will be described in detail and in the final part of the work those methods will be compared, both in term of performances and in the way they address some specific problems, exposing strengths and weaknesses of each method.

1 Introduction

Self-Supervised Learning (SSL) is a particular machine learning technique that enables model to learn from unlabeled data by self-generating the labels from the input patterns, usually requiring the model to predict or reconstruct some aspect of the input. The main objective of SSL is to pre-train a model on a large dataset using these self-generated labels, which allows the model to learn a meaningful and useful representation of the input data. This pre-trained model can then be fine-tuned on downstream tasks with smaller labeled datasets, hopefully leading to improved performance and faster convergence compared to training the model from scratch. By learning a useful representation of the input, the pre-trained model can extract relevant features and patterns that are beneficial for the downstream tasks, even if the labeled dataset is different from the original unlabeled dataset used for pre-training. SSL has the advantages of requiring less labeled data, which are usually hard to acquire, enabling to learn from unlabeled data, which are easier to get. Moreover the same pre-trained model might be used for many downstream tasks.

SSL gained a lot of popularity in the field of natural language process, where pre-training large language models on pretext tasks such as masked language modeling or next sentence prediction, allowed to reach remarkable performance on many downstream tasks. In this work some techniques for performing SSL for computer vision tasks will be analyzed, discussed and compared.

2 Problem setting

As explained in the previous section we will consider SSL for computer vision tasks. In general when we train a self-supervised model there are two main stages:

- **Pretext task**: which is the self-supervised task that is used to pre-train the model. The goal of this task is to learn a good intermediate representation with the expectation that this representation can carry good semantic or structural meanings and can be beneficial for many downstream tasks.
- **Downstream task**: which is the supervised task (such as image classification or object detection) on which we want to evaluate the model. In this stage the pre-trained model is used as a starting point, and the weights are adjusted through supervised training on the downstream task, using labeled data.

In general we don't care about the performances of the model in the pretext task, and we just care about the performances on the downstream tasks. When fine-tuning the model on the downstream task, usually a classifier (typically an MLP) is added at the end of the feature extractor learned during the pre-training, and it is trained to perform the supervised task. In general for the fine-tuning we can:

- Train the parameters of the whole architecture
- Freeze the parameters of the feature extractor and only train the weights of the MLP head
- Freeze the parameters of the first K layers of the feature extractor and train the weights of the rest of the architecture

All the methods that we'll explore exploit two fundamental concepts for SSL in computer vision: data augmentation and contrastive learning. As we already said in SSL we want to train a model using unlabeled data to predict some properties of the data. In the textual domain this is usually done by masking some words in a sentence and training the model to predict those words. In computer vision one possible way of doing is by taking an image, producing different views of that image by applying to each view a different set of transformations, and consider views produced from the same image to belong to the same class. In this way we can train a model that takes in input two views to predict if they come from the same image or not. We define a couple of images to be a positive pair if they are different views of the same image, on the contrary we define the couple as a negative pair if the two images are views from different images. If we merge this idea of positive and negative pairs obtained with data augmentation, with the goal that we want to achieve with the pretext task, which is learning a latent representation of the images that carry semantic information of the images, we get the intuition is that the in the latent space two positive images should be close, while two negative images should be distant. This is the intuition behind contrastive learning, in which we train a model to produce a representation of the data using a loss the minimize in the latent space the distance between positive images and maximize the distance between negative images.

3 Methodology

Contrastive Predicting Coding

Contrastive Predictive Coding (CPC) [5] is a method which has been thought actually not only for the computer vision field, but it can be applied also to other domains like audio and natural language. The idea of the method is to train a model that learns representation of the data by predicting the representation of future observations from the past ones. The intuition of the authors is that when using powerful generative models for predicting high-dimensional data, the model wastes capacity in modeling relationships of data x ignoring the context c, so modeling $p(x|c)$ directly it's not optimal for the purpose of extracting shared information between x and y. When we instead predict a future

information, the future x and the context c are encoded in a vector representation using a non-linear mapping that maximally preserves the mutual information between x and c, so that we can extract the underlying latent variables that x and c have in common. When applied to image processing CPC works as follows: from an image we extract a certain number of overlapping crops of a fixed size and we organize those crops in a grid. If $x_{i,j}$ is the crop in position i, j of the grid, then we use and encoder $f_\theta(\cdot)$ to encode each crop of the grid into a single vector $z_{i,j} = f_\theta(x_{i,j})$. After that a second auto-regressive model $g_\phi(\cdot)$ is used to compute $c_{i,j} = g_\phi(\{z_{u,v}\}_{u \leq i,v})$ that is a context vector that summarizes the feature in the previous rows but in the same column of $z_{i,j}$. The predictive tasks consists of predicting a feature vectors $z_{i+k,j}$ from the context vector $c_{i,j}$, where $k > 0$, using a linear model W_k computing $\hat{z}_{i+k,j} = W_k c_{i,j}$. To measure the quality of the prediction a contrastive loss is employed:

$$\mathcal{L}_{\text{CPC}} = -\sum_{i,j,k} \log p(z_{i+k,j}|\hat{z}_{i+k,j}, \{z_l\}) = -\sum_{i,j,k} \log \frac{\exp\left(\hat{z}^T_{i+k,j} z^T_{i+k,j}\right)}{\exp\left(\hat{z}^T_{i+k,j} z^T_{i+k,j}\right) + \sum_l \exp\left(\hat{z}^T_{i+k,j} z^T_l\right)}$$

where $\{z_l\}$ is the set of negative samples which is composed by taking crops from different location of the grid. In the paper they prove formally that minimizing this loss is equivalent of maximizing the mutual information between $c_{i,j}$ and $x_{i+k,j}$, obtaining the bound:

$$I(x_{i+k,j}, c_{i,j}) \geq \log(N) - \mathcal{L}_{CPC}$$

where N is the number of negative examples used in the loss. After having trained the whole architecture, for the fine-tuning on downstream tasks we keep only the encoder network $f_\theta(\cdot)$ and use it as feature extractor for a classification model, which is then trained on a supervised task.

For the encoder f_θ any CNN can be applied, in the paper the authors used a ResNet101, while for the auto-regressive model g_ϕ they used a PixelCNN.

Contrastive Multi-view Coding

Contrastive Multi-view Coding (CMC) [6] starts from the framework proposed in CPC paper and they adapt it to maximize the mutual information between different views of the same image, removing the prediction part and focusing on contrastive learning. Suppose we have two different views of a dataset V_1 and V_2, in the predictive learning setup we use a non-linear transformation to transform $v_1 \in V_1$ to $v_2 \in V_2$, passing through a latent variable z: first we use an encoder f to compute $z = f(v_1)$ and then we use a decoder g to compute $\hat{v}_2 = g(z)$, where $\hat{v}_2$ is the prediction of v_2 given v_1. The parameters of the encoder and of the decoder are then trained using an objective function that tries to bring $\hat{v}_2$ close to v_2, like the L1 or the L2 loss.

In contrastive learning instead we want to learn an embedding that separates samples from two different distributions. Given a dataset of V_1 and V_2 that consists of a collection of samples $\{v^i_1, v^i_2\}^N_{i=1}$ we consider positive pairs those which are sampled from the joint distribution $x \sim p(v_1, v_2)$ where $x = \{v^i_1, v^i_2\}$, while we consider negative samples those sampled from the product of marginals $y \sim p(v_1)p(v_2)$ where $y = \{v^i_1, v^j_2\}$. The goal is

to learn a critic function h_θ which is trained to output high values for positive pairs and low values for negative pairs. In this way we can use a contrastive loss function that we can use to train the model to correctly select a single positive sample out of a set that contains k negative samples:

$$\mathcal{L}_{contrast}^{V_1,V_2} = -\mathbb{E}_S\left[\log \frac{h_\theta(\{v_1^1, v_2^1\})}{\sum_{j=1}^{k+1} h_\theta(\{v_1^1, v_2^j\})}\right]$$

where $\{v_1^1, v_2^1\}$ is the positive pair and $\{v_1^1, v_2^j\}$ with $j > 1$ is the a negative pair. To extract the latent representations of v_1 and v_2 we use two encoders $f_{\theta_1}(\cdot)$ and $f_{\theta_2}(\cdot)$. We compute the latent representations $z_1 = f_{\theta_1}(v_1)$ and $z_2 = f_{\theta_1}(v_2)$ which we can use to compute the output of the critic as:

$$h_\theta(\{v_1, v_2\}) = \exp\left(\frac{z_1^T z_2}{\|z_1\|\|z_2\|} \cdot \frac{1}{\tau}\right)$$

In the formulation of $\mathcal{L}_{contrast}^{V_1,V_2}$ the view V_1 is used as anchor view, while we enumerate samples from view V_2, but we can also do the opposite and use the following loss:

$$\mathcal{L}(V_1, V_2) = \mathcal{L}_{contrast}^{V_1,V_2} + \mathcal{L}_{contrast}^{V_2,V_1}$$

In the paper the authors prove that the optimal critic h_θ^* is proportional to the density ratio between the joint distribution $p(z_1, z_2)$ and the product of the marginals $p(z_1)p(z_2)$:

$$h_\theta^*(\{v_1, v_2\}) \propto \frac{p(z_1, z_2)}{p(z_1)p(z_2)} \propto \frac{p(z_1|z_2)}{p(z_1)}$$

that is the point-wise mutual information and from this the same bound obtained for CPC get be proved:

$$I(z_i; z_j) \geq \log(k) - \mathcal{L}_{contrast}$$

hence minimizing the contrastive loss yields to the maximization of the mutual information between the latent representation of the different views.

Then the authors proposed two different ways for extending the above defined framework to the usage of multiple views. If we have M different views of a dataset $V_1, \ldots, V_M$ in the core view formulation we select one view that we want to optimize, for instance V_1, and we build a pairwise representations between V_1 and all the other views:

$$\mathcal{L}_C = \sum_{j=2}^{M} \mathcal{L}(V_1, V_j)$$

In the full graph formulation we instead consider all the possible pairs:

$$\mathcal{L}_F = \sum_{j=1}^{M} \sum_{i=1}^{j-1} \mathcal{L}(V_i, V_j)$$

The full graph formulation is more computational expensive but it has the advantage that it allows to capture more information between different views.

Pre-text Invariant Representation Learning

Pre-text Invariant Representation Learning (PIRL) [1] is a SSL method that uses contrastive learning like CMC, but the intention of the author is to train a model which learn a representation of the input images that is invariant with respect to image transformations.

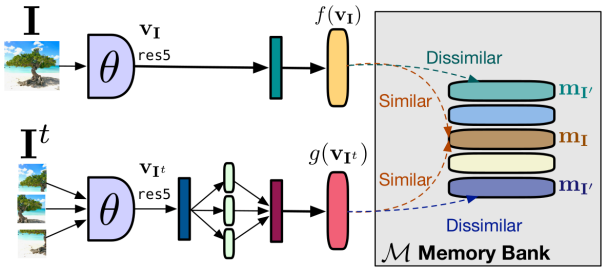


Figure 1: Scheme that shows how PIRL works

Given a dataset of images $\mathcal{D} = \{I_1, \ldots, I_{|D|}\}$ where $I_n \in \mathbb{R}^{H \times W \times 3}$ and set of transformation $\mathcal{T}$, the goal of PIRL is to train a convolutional neural network $\phi_\theta(\cdot)$ that constructs image representations $v_I = \phi_\theta(I)$ which are invariant to image transformations in $\mathcal{T}$. In order to do so ϕ_θ is trained to minimize the following loss function:

$$\ell_{inv}(\theta; \mathcal{D}) = \mathbb{E}_{t \sim p(\mathcal{T})} \left[\frac{1}{|D|} \sum_{I \in \mathcal{D}} L(v_I, v_{I^t}) \right]$$

where $I^t = t(I)$ is the image I after transformation t, $L(\cdot, \cdot)$ measures the similarity between two image representations and $p(\mathcal{T})$ is a distribution over the transformation in $\mathcal{T}$. Minimization of this loss encourages the network $\phi_\theta(\cdot)$ to produce the same representation for the image I as the transformed counter part I^t, so to make the representation invariant with respect to transformation t. This is in contrast to other losses of the following form:

$$\ell_{cov}(\theta; \mathcal{D}) = \mathbb{E}_{t \sim p(\mathcal{T})} \left[\frac{1}{|D|} \sum_{I \in \mathcal{D}} L_{co}(v_I, z(t)) \right]$$

where z is a function that measures some property of the transformation t. These kind of losses push the network $\phi_\theta(\cdot)$ to make the representation covariant with respect to the transformation, which causes the network to learn information that are not semantically relevant. $L(\cdot, \cdot)$ is defined as a contrastive loss:

$$h(v_I, v_{I^t}) = \frac{\exp\left(s(v_I, v_{I^t})/\tau\right)}{\exp\left(s(v_I, v_{I^t})/\tau\right) + \sum_{I' \in \mathcal{D}_N} \exp\left(s(v_I, v_{I'})/\tau\right)}$$

where $\mathcal{D}_N \subseteq \mathcal{D} \setminus \{I\}$ is a set of negative samples and $s(\cdot, \cdot)$ is a similarity function. Actually to compute the similarity we do not compute the features v directly but we apply different heads. Specifically we use $f(\cdot)$ for v_I and $g(\cdot)$ for v_{I^t}. So the actual loss is:

$$L_{\text{NCE}}(I, I^t) = -\log\Big(h(f(v_I), g(v_{I^t}))\Big) - \sum_{I' \in \mathcal{D}_N} \log\Big(1 - h(g(v_I^t), f(v_{I'}))\Big)$$

which encourages the representation of image I to be similar to the one of its transformed counterpart I^t, while also encouraging it to be dissimilar to the representation of other images I'.
To address the problem of finding many negative samples during the training the authors decided to use a memory bank $\mathcal{M}$ that contains the feature representation m_I of image $I \in \mathcal{D}$. m_I is an exponential moving average of the feature representation $f(v_I)$ that were computed in the prior epochs. In this way we can replace in th loss definition the negative samples $f(v'_I)$ by their memory bank representation $m_{I'}$, while keeping the batch size small. All the representation that are stored in the memory bank are all computed on the original images, I, without the transformation t. An issue of L_{NCE} is that it does not compare the representation of the original image I with the representation of the negative samples I', and this is fixed by using the following loss instead:

$$L(I, I^t) = \lambda L_{\text{NCE}}(m_I, g(v_{I^t})) + (1 - \lambda) L_{\text{NCE}}(m_I, f(v_I))$$

where the first term is the one we had before and the second term encourages the representation of $f(v_I)$ to be similar to its memory bank representation m_I and it also encourages the representation of $f(v_I)$ and $f(v_{I'})$ to be dissimilar.

Momentum Contrast

Momentum Contrast (MoCo) [2] uses a view of contrastive learning as dictionary look-up. If we consider and encoded query q and a and encoded set of of keys of the dictionary $\{k_1, k_2, \dots\}$ and we assure that there exists a single key k_+ that matches with q, we can define a contrastive loss ass a function that is low when q is similar to its positive key k_+ and dissimilar to all the other (negative) keys. One example is the InfoNCE loss:

$$\mathcal{L}_q = -\log \left(\frac{\exp\left(q^T k_+ / \tau\right)}{\sum_{i=0}^{K} \exp(q^T k_i / \tau)} \right)$$

that can be used as an unsupervised objective function for training the encoder networks that represent the queries and the keys: $q = f_q(x^q)$ and $k = f_k(x^k)$. Under this dictionary lookup prospective, contrastive learning is a way for building a discrete dictionary on high dimensional continuous inputs. This dictionary is dynamic since the keys are randomly sampled and $f_k(\cdot)$ changes during training.

To address the problem of finding good negative samples MoCo maintains a dictionary of mini-batches of data samples. In this way the encoded keys from the preceding mini-batches can be used in the loss: at each training step the current mini batch is enqueued in the dictionary and the oldest mini-batches is dequeued from the dictionary. The problem of using a large queue as a dictionary can be the update of the key encoder using back propagation intractable, since the gradient should propagate to all the samples in the queue, so the authors solve this problem by updating f_k's parameters θ_k as a moving average of f_q's parameters θ_q:

$$\theta_k \leftarrow m\theta_k + (1 - m)\theta_q$$

where $m \in [0, 1)$. θ_q is normally updated using back-propagation. In this way θ_k evolves more smoothly than θ_q. As a result though the keys in the queue are encoded by different encoders, this difference is now smaller.

Swapping Assignments between Multiple Views

Swapping Assignments between Multiple Views (SWAV) [3] unlike all the other methods we've seen, it does not use contrastive learning but it is an online clustering based method. In clustering based method the idea is to use a clustering algorithm to cluster the embedding of a batch of images, and use those embedding as pseudo-label for a predictive task in which a model given the embedding of the images is trained to predict the cluster of each image. This method was proposed in Deep Cluster [4].

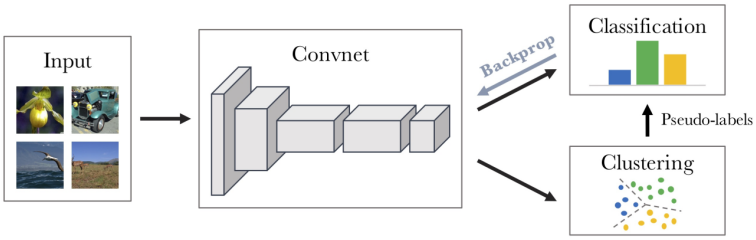


Figure 2: General framework of clustering based methods

With SWAV the authors combine the main framework of contrastive learning, which is comparing different views of the images produced with augmentation, with a clustering based pretext task. SWAV works as follows: each image x_n is transformed into an augmented view x_{nt} by applying a transformation $t \sim \mathcal{T}$, where $\mathcal{T}$ is a set of augmentations. Then we map the augmented view to a vector representation with a non-linear mapping f_θ to x_{nt} and we then project this feature in a unit sphere $z_{nt} = f_\theta(x_{nt})/\|f_\theta(x_{nt})\|_2$. From this representation we compute the code q_{nt} by mapping z_{nt} to a set of K trainable prototypes vectors $\{c_1, \ldots, c_k\}$. We use C to denote the matrix whose columns are the $c_1, \ldots, c_k$ vectors. If we do the same thing with another augmentation $s \sim \mathcal{T}$, obtaining the image feature z_{ns} and its corresponding code q_{ns} then we can set a swapped prediction problem using the following loss:

$$L(z_{nt}, z_{ns}) = \ell(z_{nt}, q_{ns}) + \ell(z_{ns}, q_{nt})$$

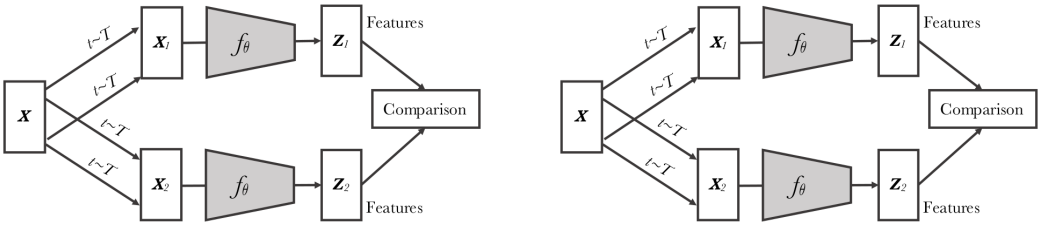
where the function $\ell(z, q)$ measures the fit between features z and the code q. The idea is that instead of comparing directly the features of z_t and z_s, we do it using the intermediate codes q_t and q_s. If these two features capture the same information, it should be possible to predict the code of the feature z_q using the feature z_t and vice versa. The loss $\ell(z_t, q_s)$ is the cross entropy loss between the code and the probability obtained by taking a soft-max of the dot product of x_i and all the prototypes in C:

$$\ell(x_t, q_s) = -\sum_k q_s^{(k)} \log p_t^{(k)}, \qquad p_t^{(k)} = \frac{\exp\left(z_t^T c_k/\tau\right)}{\sum_{k'} \exp(z_t^T c'_k/\tau)}$$

by applying some mathematical manipulations to these formula we can obtain our objective:

$$\frac{1}{N}\sum_{n=1}^{N}\sum_{s,t\sim\mathcal{T}} L(z_{nt}, z_{ns}) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{s,t\sim\mathcal{T}}\left[\frac{1}{\tau}z_{nt}^T C q_{ns} + \frac{1}{\tau}z_{ns}^T C q_{nt} - \log\sum_{i=1}^{K}\exp\left(\frac{z_{nt}^T c_k}{\tau}\right) - \log\sum_{i=1}^{K}\exp\left(\frac{z_{ns}^T c_k}{\tau}\right)\right]$$

so we can train the parameters θ of the encoding function $f_\theta(\cdot)$ to minimize this loss.



(a) Contrastive learning framework (b) SWAV framework

Figure 3: Difference between the classical contrastive learning framework with the one proposed in SWAV

The question now is how do we compute the codes? Given a batch of B features $Z = [z_1, \ldots, z_B]$ we want to map them to the prototypes $C = [C_1, \ldots, C_K]$, obtaining the codes $Q = [q_1, \ldots, q_B]$, where Q is an $K \times B$ matrix where $Q_{i,j}$ can be viewed as the probability of feature z_j to belong to the prototype c_i. We want to optimize Q to maximize the similarity between features and prototypes, and this can be expressed in the following maximization problem:

$$\max_{Q\in\mathcal{Q}}\left\{\operatorname{Tr}\Big(Q^T C^T Z\Big) + \epsilon H(Q)\right\}$$

where $H(Q) = -\sum_i\sum_j Q_{i,j}\log Q_{i,j}$ and ϵ is a parameter that controls the smoothness of the mapping. The matrix $C^T Z$ contains the similarity of each feature vector and each prototype, since $(C^T Z)_{i,j} = c_i^T z_j$, hence we can r

$$\operatorname{Tr}\Big(Q^T C^T Z\Big) = \sum_{i=1}^{B}\sum_{j=1}^{K} q_{j,i}(c_j^T z_i) = \sum_{i=1}^{B}\sum_{j=1}^{K} q_{j,i}\Big(\sum_{t=1}^{D} c_{j,t} z_{i,t}\Big)$$

In doing this we need to be careful to avoid the trivial solution in which all the features are mapped to the same code. This can be achieved by imposing an equipartition constraint, which is basically saying that each . In order to enforce the equal partitioning constraint the set $\mathcal{Q}$ is defined as:

$$\mathcal{Q} = \{Q \in \mathbb{R}^{K\times B} | Q1_B = \frac{1}{k}1_k, Q^T 1_K = \frac{1}{B}1_B\}$$

where 1_K denotes the K-dimensional vector where all the elements are 1. Once a continuous solution Q^* to the optimization problem is found a discrete code can be obtained by using a rounding procedure. Empirically the authors found that a discrete codes doesn't

work well in an online setting, using the discrete codes perform worse than using the continuous codes. Therefore the authors doesn't use rounding and the solution to the optimization problem can be computed as:

$$Q^* = \text{Diag}(u)\exp\Big(\frac{C^T Z}{\epsilon}\Big)\text{Diag}(v)$$

where u and v are re-normalization vectors in $\mathbb{R}^K$ and $\mathbb{R}^B$, which can be computed with the Sinkhorn-Knopp algorithm.

4 Analysis and comparison

In this section the five methods we have explained will be compared in term of performances, in addition also two other methods which have been explored during the lectures will be considered, SimCLR and BYOL. As we already said in the previous sections, the goal in SSL is having a good positive transfer in the downstream tasks, hence to compare the performances of the method we are going to see how using each one. In particular the considered downstream task will be image classification and object detection.

In table 1 we reported the top-1 and top-5 accuracy (the top-5 accuracy is available only for CPC and SimCLR) of the considered methods when tested for classification on the ImageNet dataset. All the methods uses as feature extractor a ResNet50.

Method	**Architecture**	**ImageNet**	
		Top1	Top5
Supervised	ResNet50	76.5	-
PIRL	ResNet50	63.6	-
CPCv2	ResNet50	63.8	85.3
CMC	ResNet50	66.2	87
SimCLR	ResNet50	69.3	89.0
MoCov2	ResNet50	71.1	-
BYOL	ResNet50	74.3	91.6
SwAV	ResNet50	75.3	-

Table 1: All architecture uses ResNet50

Method	Architecture	ImageNet 1%		ImageNet 10%	
		Top1	Top5	Top1	Top5
Supervised	ResNet50	25.4	48.4	56.4	80.4
PIRL	ResNet50	30.7	57.2	60.4	83.8
SimCLR	ResNet50	48.3	75.5	65.6	87.8
BYOL	ResNet50	53.2	78.4	68.8	89.0
SwAV	ResNet50	53.9	78.5	70.2	89.9
CPCv2	ResNet161	-	77.9	-	91.2
SimCLR	ResNet50 (4×)	63.0	85.8	74.4	92.6
BYOL	ResNet50 (2×)	71.2	77.7	89.5	93.7

Table 2: Result after fine-tuning on 1% of the data of ImageNet

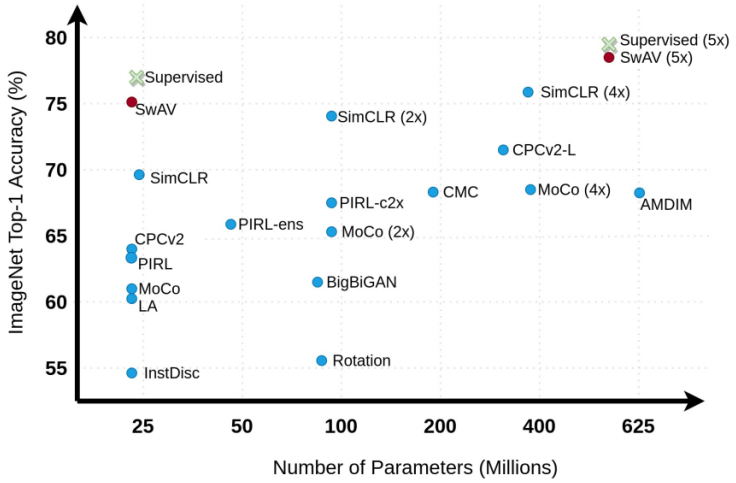


Figure 4: Top-1 classification accuracy different contrastive learning methods with the number of parameters of the models

5 Strength and weaknesses

6 Conclusion

References

[1] Ishan Misra and Laurens van der Maaten. Self-supervised learning of pretext-invariant representations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6707–6717, 2020.

[2] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[3] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *Advances in neural information processing systems*, 33:9912–9924, 2020.

[4] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European conference on computer vision (ECCV)*, pages 132–149, 2018.

[5] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[6] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16*, pages 776–794. Springer, 2020.