Beyond Single Instance Positives For Contrastive Visual Representation Learning

Martina Rossini
M.Sc. in Artificial Intelligence
Alma Mater Studiorum - University of Bologna
martina.rossini3@studio.unibo.it

Abstract

In this work, we implement NNCLR and a novel clustering-based technique for contrastive learning that we call KMCLR. In particular, we use a batched version of K-Means clustering and try different values for the amount of prototypes K, obtaining the best results for $K = 400$. We show that applying this simple clustering technique to obtain prototype vectors in the embedding space and using those prototypes to form the positive pairs for contrastive loss can achieve performances on par with NNCLR on CIFAR-100 while storing only 0.4% the number of vectors.

1 Introduction

Humans subconsciously compare new sensory inputs with something they have already experienced and this might play an important role in how they can rapidly acquire new concepts. For instance. if we are asked to picture a *mangalitsa pig*, even without having seen one before, our brain will automatically link it with other similar semantic classes like *pig* or *boar*. Dwibedi et al. (2021) show that enhancing self-supervised techniques with the ability to find similarities between new and previously seen items can improve their representation learning capabilities.

Indeed, other instance discrimination approaches like SimCLR (Chen et al., 2020) or BYOL (Grill et al., 2020) focus on learning what makes a specific image different from everything else in the dataset. They typically generate multiple views of each image through data augmentation, consider them as positive samples, and then encourage their representations to be close in the embedding space. These single-instance positives may however not be able to capture all the variance inside a given class and, what's more important, they can't account for similarities between semantically close classes.

Dwibedi et al. (2021) propose NNCLR, which uses as a positive sample the nearest neighbor of an image's view in the embedding space. This is realized in practice by storing previously computed embeddings in a large support set that is updated on a per-batch basis during training. The authors show that this technique outperforms previous state-of-the-art methods and relies less heavily on complex data augmentation schemes. They also show that, in principle, the bigger the support set the better the results; we argue that this brings their solution closer to a brute force approach and strive to find an alternative.

In particular, we propose to generate prototype vectors via online clustering and then use as the positive sample for an image the code that's nearer to its embedding. We speculate that this will be able to effectively capture both intra-class variances and similarities between multiple classes, while limiting the number of vectors we have to store. We thus expect the performances to be on par with those achieved by NNCLR. Note that, due to computational and timing constraints, all our experiments use the CIFAR-100 (Krizhevsky and Hinton, 2009) dataset for pre-training.

1.1 Contrastive instance discrimination

Many self-supervised learning approaches use contrastive losses, which draw positive samples together in the embedding space while pushing a part negative ones. Recently, a variation of contrastive learning known as instance discrimination has been shown to close the gap with supervised learning on a wide variety of downstream tasks. In this context, SimCLR (Chen et al., 2020) uses data augmentation techniques to produce two views of the same images, which are considered as the positive pair, while the negative pairs are formed using all the other images in the current mini-batch. Formally, given the batch $\{x_1, x_2, \cdots, x_n\}$, we generate two random augmentations of image x_i and feed them to the neural encoder ϕ to obtain the embeddings $z_i = \phi(\text{aug}_1(x_i))$ and $z_i^+ = \phi(\text{aug}_2(x_i))$. We can then define the contrastive loss used by SimCLR for image x_i like this

$$L_i^{\text{SimCLR}} = -\log \frac{\exp(z_i \cdot z_i^+ / \tau)}{\sum_{k=1}^{n} \exp(z_i \cdot z_k^+ / \tau)} \tag{1}$$

where τ is the softmax temperature and every embedding is L_2 normalized before the loss is computed. Given this definition of the loss for a single example, the loss for the whole mini-batch of size n is computed as shown below.

$$L^{\text{SimCLR}} = \frac{1}{n} \sum_{i=1}^{n} L_i^{\text{SimCLR}} \tag{2}$$

2 Algorithms

As briefly mentioned in the Introduction, relying on image augmentation techniques to generate positive samples cannot really capture large intra-class variations or existing semantic relationships between two or more categories. We thus implement two algorithms that try to move away from the single-instance positives paradigm: NNCLR (Dwibedi et al., 2021), which is based on nearest neighbors computation in the embedding space, and a variation of it which we call KMCLR that's based on online clustering techniques. A more formal description of these two architectures is given in Sections 2.1 and 2.2, respectively.

2.1 Nearest-Neighbor CLR

Dwibedi et al. (2021) propose to use nearest neighbors to obtain more diverse positive pairs; in practice, this means keeping a large support set Q of previously computed embeddings which is representative of the full data distribution. Then, for image x_i in the mini-batch, we still compute two augmented views as $z_i = \phi(\text{aug}_1(x_i))$ and $z_i^+ = \phi(\text{aug}_2(x_i))$. Starting from the first view z_i, we form a positive pair $(\text{NN}(z_i, Q), z_i^+)$ using its nearest neighbor from the support set and the second view. The set of embeddings Q is updated at every batch with the embeddings of the first view only, as no real performance improvement has been noticed when using both augmentations. Given these considerations and building upon SimCLR's loss, the objective can now be written as

$$L_i^{\text{NNCLR}} = -\log \frac{\exp(\text{NN}(z_i, Q) \cdot z_i^+ / \tau)}{\sum_{k=1}^{n} \exp(\text{NN}(z_i, Q) \cdot z_k^+ / \tau)} \tag{3}$$

where τ is again the softmax temperature and $\text{NN}(z, Q) = \arg\min_{q \in Q} \|z - q\|_2$. Notice that the embeddings are L_2 normalized before the loss computation and before we get the nearest neighbor.

Ablation studies by Dwibedi et al. (2021) have shown that, in general, increasing the size of the support set Q increases performances: this is reasonable because the more embeddings we store the closer we are to a representative view of the whole dataset, thus increasing the chance to get a good-quality nearest neighbor. However, the authors also point out that increasing size beyond 98,304 does not impact performance significantly; this may be attributed to stale embeddings (i.e.: those computed with old network weights) populating a vast majority of the support set.

2.2 K-Means CLR

To overcome problems related to the huge support set's size required by NNCLR, we propose to move beyond single instance positives using prototype vectors computed via clustering techniques. In particular, we use a batched implementation of K-Means (Lloyd, 1957; MacQueen, 1967), a clustering algorithm that tries to minimize the average squared distance between points in the same group. We should note that, while K-Means does not offer any guarantees regarding accuracy, it's simple and fast: these are indeed two key requirements for our application because the prototypes should be updated at every batch. Moreover, K-Means requires the user to specify the number K of clusters to compute and this is an appealing property in our case as it allows us to control the size of our "support set".

More formally, given image x_i in the batch, we compute the two augmented views as $z_i = \phi(\text{aug}_1(x_i))$ and $z_i^+ = \phi(\text{aug}_2(x_i))$. Then we find the prototype KM(z_i) associated to z_i through K-Means and build the positive pair using this code and the second view z_i^+. Building upon NNCLR's objective we can write KMCLR's loss for image x_i as

$$L_i^{\text{KMCLR}} = -\log \frac{\exp(\text{KM}(z_i) \cdot z_i^+ / \tau)}{\sum_{k=1}^{n} \exp(\text{KM}(z_i) \cdot z_k^+ / \tau)} \tag{4}$$

where τ is still the softmax temperature and the embeddings get L_2 normalized before the loss is computed. Notice that this approach only requires storing K prototypes, with $K << |Q|$, thus it is more memory efficient than NNCLR. Moreover, we can interpret the centroids found by K-Means as summaries of all the points assigned to the relative clusters. We thus argue that using these prototype vectors to form the positive pairs helps the model capture both large intra-class variations and between-classes semantic similarities.

Dealing with concept drift As described above, the prototype vectors are updated at every batch using the first view's embeddings. Recall that, in every batch, the embeddings are computed based on different values for the encoder's weights, because the network is being trained. This means that our clustering algorithm needs to deal with *concept drift*, a change in the underlying data distribution over time. We choose to deal with this by periodically resetting the clusters estimated by K-Means, quite literally making the algorithm forget previously seen embeddings; in particular, we choose the reset interval to be equal to 3 epochs. Figure 1 shows the difference in performance between forgetting interval $f = 3$ and $f = 10$; as we can see a smaller f improves performances, indicating that our clustering algorithm really does suffer from the shift in the distribution of the underlying embeddings.

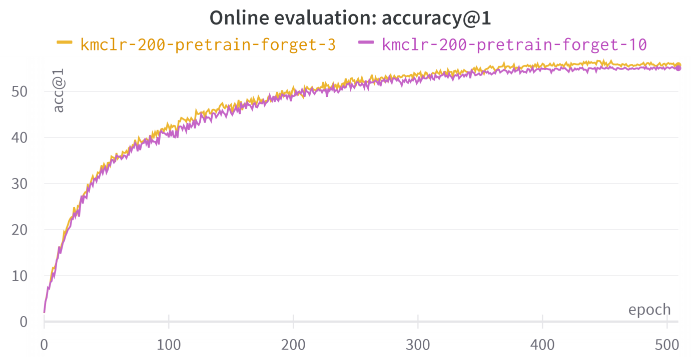

Figure 1: Validation accuracy@1 during the online linear evaluation.

3 Experimental Setup

Due to computational and timing constraints, we pre-train for 500 epochs on the CIFAR-100 dataset, which contains 50k training images and 10k evaluation images. We use the LARS (You et al., 2017) optimizer with a base learning rate of 0.2 and employ a linear warmup of 10 epochs followed by a cosine annealing learning rate scheduling. Weight decay of 10^{-6} is applied during training to all terms but the biases and a temperature of 0.1 is used when computing the loss according to Equations 3 and 4, respectively. Both NNCLR and KMCLR use a ResNet-18 (He et al., 2015) feature encoder appropriately modified to work with images of size 32x32 instead of 224x224, while for projection and prediction MLP we adopt the structure described in Dwibedi et al. (2021), with an embedding size of 256.

Following Chen et al. (2020), we perform online linear evaluation by attaching a linear classifier on top of the residual encoder and training them simultaneously during pre-training: here, it's important to add a stop gradient on the input of the linear classifier to make sure that the label information can not influence the self-supervised encoder. Note that, for the online classifier, we use no weight decay and a lower learning rate of 0.01.

Finally, following both Chen et al. (2020) and Grill et al. (2020) we perform offline linear evaluation by training a linear classifier on top of the frozen representations learned by our two self-supervised methods, meaning that we do not update the encoder's parameters nor the batch statistics. The classifier is trained for 90 epochs using LARS optimizer with a base learning rate of 0.1 and cosine decay scheduler.

Augmentation Techniques During pre-training, augmentation techniques follow BYOL (Grill et al., 2020): first we take a random crop of the image with an area uniformly sampled between 8% and 100% of the original image and then resize the output to 32x32. We also apply a random horizontal flip, followed by color jittering and optionally conversion to grayscale. Finally, we apply Gaussian blur and solarization on the image, then normalize by CIFAR-100 mean and standard deviation. More details about the parameters used for both view's augmentations are shown in Table 1, where aug_1 indicates the random transformations that result in the first view while aug_2 indicates those that result in the second view.

Table 1: Parameters used to generate the two image augmentation functions.

Parameter	aug_1	aug_2
Random crop probability	1.0	1.0
Flip probability	0.5	0.5
Color jittering probability	0.8	0.8
Brightness adjustment	0.4	0.4
Contrast adjustment	0.4	0.4
Saturation adjustment	0.2	0.2
Hue adjustment	0.1	0.1
Grayscale probability	0.2	0.2
Gaussian blurring probability	1.0	0.1
Solarization probability	0.0	0.2

During offline linear evaluation instead, we only apply random crops and horizontal flips to the training images before normalization.

4 Results

Because we experimented with different numbers of prototype vectors, in both Figure 2 and Table 2 `kmclr-i` indicates that we are employing K-Means with $K = i,\ i \in \{50, 200, 400, 600\}$. Regarding NNCLR, we keep the support set size fixed to the value used by Dwibedi et al. (2021) for their experiments (i.e.: 98,304). Note however that all NNCLR ablations in the original paper refer to the much bigger ImageNet (Russakovsky et al., 2014) dataset; it is thus reasonable to assume that we could obtain a representative subset for CIFAR-100 with a smaller queue size. Still, Dwibedi et al. (2021) do not observe any degradation in performance with a bigger support set but simply point out that a greater size does not improve performance enough to justify the increase in cost: we thus choose to keep this parameter fixed.

Our results quite clearly show that the performance of KMCLR starts saturating after $K = 400$, implying that no real benefits are obtained from further incrementing the number of clusters. We also note that increasing K has a negligible impact on run-time: on the same GPU, using $K = 400$ or $k = 600$ results in a similar mean execution time per epoch. Moreover, clustering the embeddings online does not seem to slow down training when compared with computing the nearest neighbor in the support set: indeed, KMCLR and NNCLR show pretty similar epoch running time. Thus, we showed that our approach to move beyond single-instance positives achieves performances that are on par with NNCLR on the CIFAR-100 dataset, while not impacting training time and drastically reducing the amount of stored embeddings.

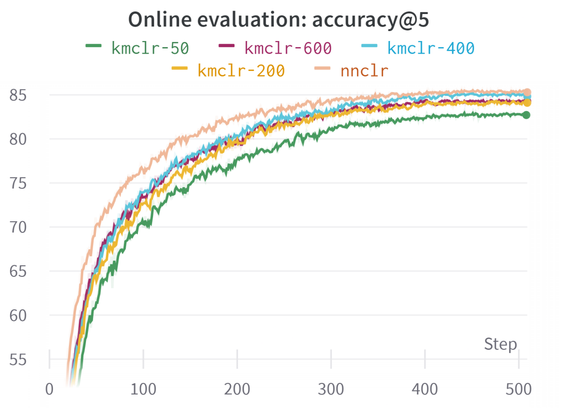

Figure 2: Online linear evaluation: top-5 accuracy on validation set.

Table 2: Offline linear evaluation results on CIFAR-100 test set.

Model	Acc@1	Acc@5
`nnclr`	**57.60**	**84.78**
`kmclr-50`	54.66	83.11
`kmclr-200`	56.13	84.53
`kmclr-400`	**56.86**	**84.70**
`kmclr-600`	56.52	84.48

5 Conclusion and Future Work

We show that using a simple clustering technique like K-Means to obtain prototype embeddings can achieve performances on par with NNCLR while storing only 0.4% the number of vectors. We should however point out that more sophisticated stream clustering techniques developed for evolving environments, where the concepts being tracked may change over time, could further improve performances. For instance, DyClee (Barbosa Roa et al., 2019) works in two stages, (i) a distance-based clustering that takes as input our embeddings in an incremental fashion and produces μ-clusters and (ii) a density-based stage that analyses the micro-clusters to generate the final assignments. Because our clustering-based contrastive approach is pretty modular, we could easily change the algorithm used to generate the prototype vectors; employing a method like DyClee would be extremely appealing as it also implements a more advanced forgetting strategy that lets clusters emerge, disappear, split, or join following the evolution of the embeddings.

Moreover, we know that contrastive algorithms' performances rely heavily on the augmentation pipeline. Dwibedi et al. (2021) have shown how NNCLR is more robust to changes in this process and we speculate that KMCLR also benefits from this advantage, as it does not rely directly on an image's embedding to form the positive pair but rather on the prototype of the cluster that the image falls into. It would thus be interesting to test how KMCLR performs when changing the data augmentation strategy, as well as to compare its performances with other SOTA algorithms on more meaningful datasets like ImageNet (Russakovsky et al., 2014). Furthermore, due to timing constraints, we have not evaluated the learned embedding for transfer learning.

Finally, we acknowledge that visualizing how the prototypes evolve over time in the embedding space might give us a better intuition of what exactly the model is focusing on during the training phase.

References

N. Barbosa Roa, L. Travé-Massuyès, and V. H. Grisales-Palacio. Dyclee: Dynamic clustering for tracking evolving environments. *Pattern Recognition*, 94:162–186, 2019. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2019.05.024. URL `https://www.sciencedirect.com/science/article/pii/S0031320319301992`.

T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations, 2020. URL `https://arxiv.org/abs/2002.05709`.

D. Dwibedi, Y. Aytar, J. Tompson, P. Sermanet, and A. Zisserman. With a little help from my friends: Nearest-neighbor contrastive learning of visual representations, 2021. URL `https://arxiv.org/abs/2104.14548`.

J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko. Bootstrap your own latent: A new approach to self-supervised learning, 2020. URL `https://arxiv.org/abs/2006.07733`.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015. URL `https://arxiv.org/abs/1512.03385`.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009.

S. Lloyd. Least square quantization in pcm. bell telephone laboratories paper. published in journal much later: Lloyd, sp: Least squares quantization in pcm. *IEEE Trans. Inform. Theor.(1957/1982)*, 18(11), 1957.

J. MacQueen. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297. University of California Los Angeles LA USA, 1967.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge, 2014. URL `https://arxiv.org/abs/1409.0575`.

Y. You, I. Gitman, and B. Ginsburg. Large batch training of convolutional networks, 2017. URL `https://arxiv.org/abs/1708.03888`.