Contrastive Learning with 3D Shapes

November 6, 2022

Andrea Bernini

Abstract

In fields such as Computer Vision or NLP there is a large amount of data available which, however, cannot be labeled, as it would be very expensive. A possible solution to this problem is Contrastive Learning, a Self-Supervised technique. The purpose of this project is to implement a Contrastive Learning regimen for 3D shapes, to do this we use the DynamicEdgeConv, a Convolutional Layer, that can operate directly on the point cloud itself, thanks to the fact that they are invariant with respect to permutations

1. Introduction

Contrastive Learning is a machine learning technique used to learn the general characteristics of an unlabeled dataset by teaching the model which data points are similar or different by incorporating versions of the same sample next to each other while attempting to push away the embeds from different samples.

This technique is very useful because manually annotating unlabeled datasets is a very expensive job, both in terms of time and money, especially nowadays in fields such as *Computer Vision* or *Natural Language Processing*, where you have an increasingly large amount of data that are, however, not labeled.

Contrastive Learning pipeline can be divided into three main sections:

- The first phase is that of **Augmentation**, in which for each sample (images, points of cloud, audio, etc.) in our dataset, two combinations of augmentation are performed (e.g., color transformation, geometric transformation, activities based on the context, and activities based on cross-modal modes), in such a way as to use the original image as an anchor, its augmented version as a positive sample, and the rest of the images in the batch (or in the training data) as negative samples (Figure 1). The choice of which combinations to perform is very important, as if done incorrectly, it could introduce bias, instead, the goal is to make the model invariant with respect to these transformations while remaining discriminatory with respect to other data points.
- The next stage is **Encoding**, where we feed our augmented data into our deep learning model, to create vector representations for each sample. The goal is to train the model to produce similar representations for similar samples.
- Finally we have the **Training** phase in which we try to maximize the similarity of the two vector representations by minimizing a *Contrastive Loss* function.

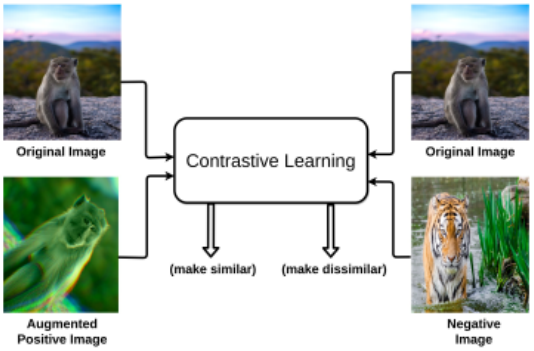

Figure 1. Push original and augmented images closer and push original and negative images away (Jaiswal et al., 2020).

2. Related Work

Point Cloud Classification with Graph Neural Networks This is a tutorial (Poi) provided by the PyTorch Geometric documentation, where, through the use of PointNet++ (Qi et al., 2007), the classification of a point cloud is performed through the use of a graph of neural networks.

SimCLR It is a State-of-the-art Self-supervised Representation Learning Framework. For this reason, to de-

Email: Andrea Bernini <bernini.2021867@studenti.uniroma1.it>.

Deep Learning and Applied AI 2022, Sapienza University of Rome, 2nd semester a.y. 2021/2022.

velop the architecture of my model, I took ispiration from it (Chen et al., 2020a), in which the authors generate two views of an image, so they do not use the original data point but attract and reject the representations of all the augmented images. This prevents the model from learning how an image was augmented.

3. Method

Dataset The PyTorch geometric library contains a large number of common benchmark datasets regarding 3D figures. For my project, I decided to use *Shapenet*, a dataset containing 3D shape point clouds from 16 shape categories. I decided to use this Dataset because thanks to its parameter it is possible to select only a subset of categories of figures, which is a lot to avoid problems related to the space occupied in RAM.

Data Augmentation This part is very important because, as explained in Section 1, if it is not applied in the right way, it introduces a strong bias. For example, with point cloud samples, the rotation is problematic because, depending on the layer we use, some of the layers might be rotation invariants, so this augmentation won't have any effect. Good augmentations for this type of sample are *Jittering* (which translates node positions by randomly sampling translation values within a given interval), *Shifting*, and *Shearing* (which shift in one dimension).

Model One of the main problems of the project is how to manage the 3D point cloud. To do this, it is possible to take different approaches such as *Voxelization* (which is not very efficient) or *Multi-View*, i.e. look at the object from different angles and take 2D photos to use as input for a CNN. The latter approach is more efficient than the former, but we don't really use 3D information. However, there are Deep Learning models that can operate directly on the point cloud itself, thanks to the fact that they are invariant with respect to permutations and therefore to the ordering of points. Examples of these models are: *PPFNet* (Deng et al., 2018), *PointNet/Pointnet++* (Qi et al., 2007), *EdgeConv/DynamicEdgeConv* (WANG et al., 2020).

As mentioned in the Section 2, for the network architecture I took inspiration from the SimCLR where an **End-to-End** architecture is used, i.e. for each sample x, we generate two augmented versions ($\tilde{x}_i$ and $\tilde{x}_j$), which we will pass to the encoder $f(\cdot)$ (consisting of two **DynamicEdgeConv** layers, I took inspiration from (PyG)), to obtain a representation vector. Then we pass this to the Projection Head, a fully connected network. The projection head $g(\cdot)$ maps the representation h into a smaller space where we apply the loss function. The reason why it is necessary to apply the Projection Head is that most similarity measures

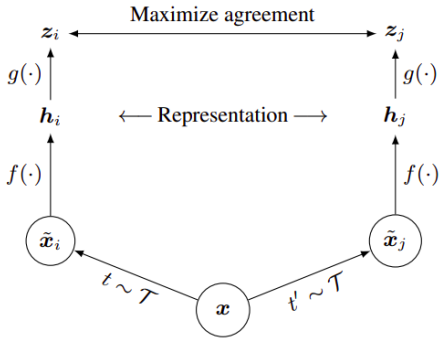

Figure 2. Contrasting Learning model architecture (Chen et al., 2020a)

that are used in contrastive losses suffer from the curse of dimensionality. Therefore, when we calculate the loss on smaller vectors, we will get better results, as we can see in the Paper provided by the project delivery (Chen et al., 2020b).

Loss Function As Loss function I used the **NTXent Loss**, sometimes also called *InfoNCE*, which is implemented in the *PyTorch Metric Learning* (PyT) library, which has the Temperature as hyperparameter which helps to balance the similarity measure. In short, the NTXent Loss compares the similarity of z_i and z_j to the similarity of z_i to any other representation in the batch by performing a softmax over the similarity values. The loss can be formally written as:

$$\mathcal{L}_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(z_i, z_k)/\tau)} \quad (1)$$

where τ is the *temperature*, k_i represents a negative sample, and $sim()$ is the application of Cosine Similarity.

4. Experimental Results

For the test phase, we will use the **DownStream** approach to see how the model performs with small data. As for the SimCLR implementation, I will use the Logistic Regression, to see if the model generalizes well, to which we will pass the 3D elements that have already been processed by our $f(\cdot)$ function, i.e. encoded in their feature vector, which will associate the representations with a class prediction.

5. Discussion and Conclusion

One of the main problems is the use of the End-to-En architecture, which, as explained in Paper 1 provided by the professor, works well only in the presence of a large number of negative samples and, therefore, a large batch size. In fact, for hardware reasons in Google Colab, I could only

use batches of 32 elements; to give an example, SimCLR uses a size equal to 4096. A future improvement could be to use a different architecture that has no problems related to the use of the GPU memory, for example the Memory Bank or the Momentum Encoder.

Table. Tables can be used to report quantitative results, here is one random example:

Table 1. Performance comparison.

#factors	β VAE	DCI Dis.	MIG	MIG-PCA	MIG-KM
One	100%	**99.0%**	63.7%	73.5%	69.2%
Variable	98.9%	94.9%	62.3%	70.5%	**66.9%**

References

Point cloud classification with graph neural networks. shorturl.at/nqIX5. Accessed: 2022-10-12.

Pytorch geometric dgcnn_classification.py example from github. shorturl.at/D1246. Accessed: 2022-10-12.

Pytorch metric learning library. shorturl.at/efJO0. Accessed: 2022-10-12.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. *PMLR*, pp. 02, 2020a.

Chen, T., Kornblith, S., Swersky, K., Norouzi, M., and Hinton, G. Big self-supervised models are strong semi-supervised learners. *Advances in Neural Information Processing Systems 2020*, pp. 04, 2020b.

Deng, H., Birdal, T., and Ilic, S. Ppfnet: Global context aware local features for robust 3d point matching. *CVPR 2018*, 2018.

Jaiswal, A., Ramesh Babu, A., Zaki Zadeh, M., Banerjee, D., and Makedon, F. A survey on contrastive self-supervised learning. *Technologies 2021*, pp. 02, 2020.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *NIPS'17*, 2007.

WANG, Y., SUN, Y., LIU, Z., SARMA, S. E., BRONSTEIN, M. M., and SOLOMON, J. M. Dynamic graph cnn for learning on point clouds. *Technologies 2021*, pp. 02, 2020.