Chapter 07 – Self-Supervised Learning

Author: Gianmarco Scarano

gianmarcoscarano@gmail.com

1. Introduction

Recalling Supervised Machine Learning from ML theory, we know that our data comes in pairs $\{(x_i, y_i) \mid i = 1, \dots, n\} \subseteq \mathbb{R}^d$ where $x_i \in \mathbb{R}^d$ is our input instance and y_i it's its label (class).

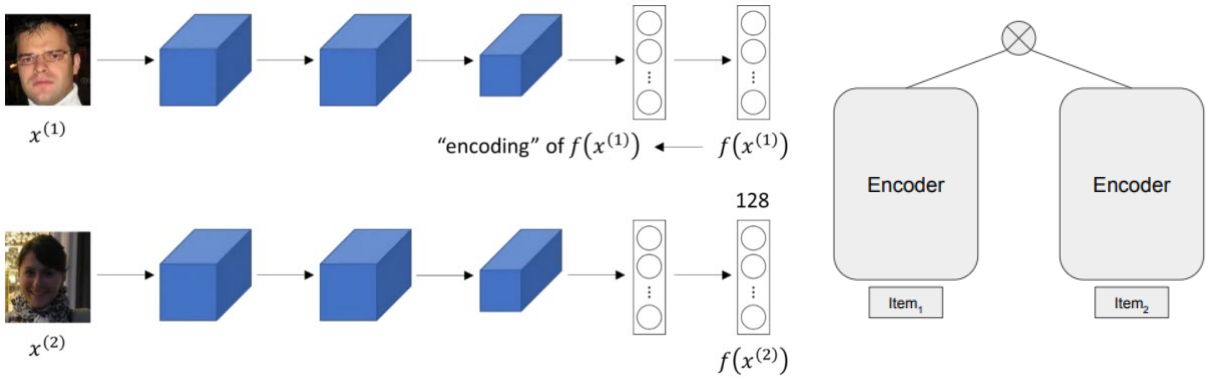
The data points (x_i, y_i) are drawn from some (unknown) distribution $\mathbb{P}(X, Y)$.

The goal is to learn a function H such that for a new pair $(\boldsymbol{x}, y) \sim \mathbb{P}$, we have $H(\boldsymbol{x}) = y$ with high probability (or that directly $H(\boldsymbol{x}) \approx y$), meaning that we want to learn good representation for the input. Getting the label, though, is really expensive.

Talking about Self-Supervised Learning (which from now on we'll call SSL), we do not care about its performances (while creating the approach) because we want to evaluate it later on downstream tasks.

2. Siamese Neural Networks

This Neural Network is a class of Neural Network architecture that contains two or more identical subnetworks (Siamese), meaning that they have the same configuration with the same parameters and weights (actually they share them).



The idea behind this is that for example, if $Item_1$ and $Item_2$ are two pictures of a cat, the encoder (where parameters live) will most probably define them as a cat, meaning that the two vectors resulting from the encoder should be really close. If they are not, it means that one or the other predicted the wrong class and the loss will try to minimize the distance between the two vectors.

3. Contrastive Learning

In simple word, in Contrastive Learning we want to contrast two different things.

Actually, we want to detect if the two inputs are coming from the same "pseudo"-class or from different ones.

3.1 Contrastive Loss

The general idea is that, given samples x_i and corresponding labels y_i, we want to design a Loss function where if we know that the two objects are coming from the same class, we want their embeddings to be close to each other and vice versa. In the opposite case (where they are far away) we want them to be further than a certain threshold ϵ (hyperparameter), which tells us we are bounded to a specific class.

The formula is:

$$\mathcal{L}_{\text{cont}}(\mathbf{x}_i, \mathbf{x}_j, \theta) = \mathbb{1}[y_i = y_j] \|f_\theta(\mathbf{x}_i) - f_\theta(\mathbf{x}_j)\|_2^2 + \mathbb{1}[y_i \neq y_j] \max(0, \epsilon - \|f_\theta(\mathbf{x}_i) - f_\theta(\mathbf{x}_j)\|_2)^2$$

(The representation of x_i and x_j need to be as close as possible when there are both from the same classes $y_i = y_j$, but when x_i is different from x_j, then I want the distance to be at least ϵ).

3.2 Triplet Loss

Instead of considering just pairs, we now consider triplets. Given one anchor input x, we have a positive sample x^+ which is the closest to x and we have a negative sample x^- which is the furthest one (at least ϵ though). So now we can exploit the three terms, also separating x^+ and x^- thanks to the anchor point.

4. Heavy Data Augmentation, Batch size, Hard negative mining

Given a training sample, data augmentation techniques are needed to creating a noise version of itself to feed into the loss as positive sample. This encourages the model to learn the essential part of the representation (since we are not modifying the semantic meaning).

Then, success has been made by using a large batch size during training, since the loss function can cover a diverse enough collection of negative samples, challenging the model to learn representation through a bunch of different examples.

Lastly, the hard negative mining which deals with negatives.

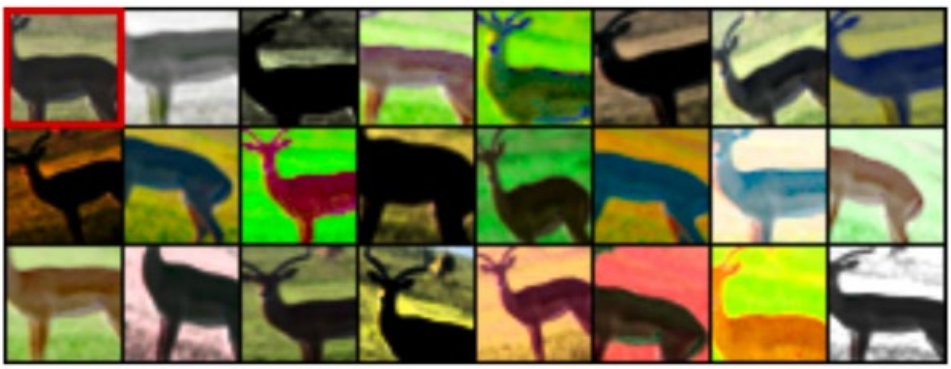
For example, if we have a cat and a dog, the network should easily recognize that they are coming from two different labels, but if I give an image of an eagle and a vulture, then it should not be that simple. We try to feed these examples into the model.

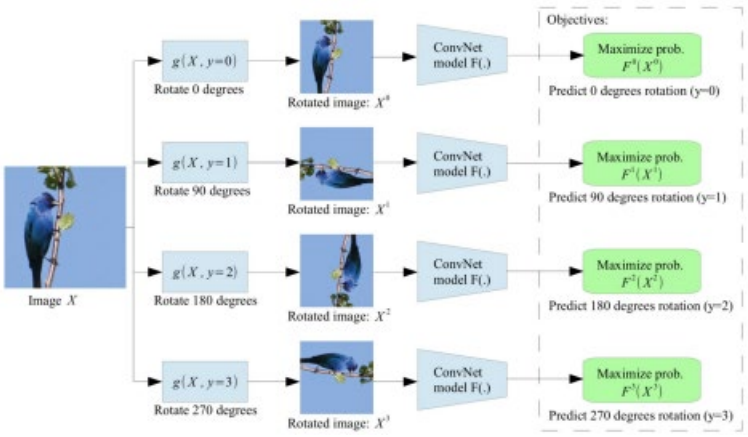
5. Image Self-Supervision

In a common workflow, we try to mix-up several tasks, since: the more we use, the better the representation could get. We explained that the goal of SSL is to learn representation, but at the end of the day we must apply a Classifier Layer, which should be kept as simple as possible due to the fact that it might correct mistakes that we do during our feature extraction (representation learning) phase, jeopardizing the whole concept of Self-Supervision. The final classification accuracy tells us how good our representation is.

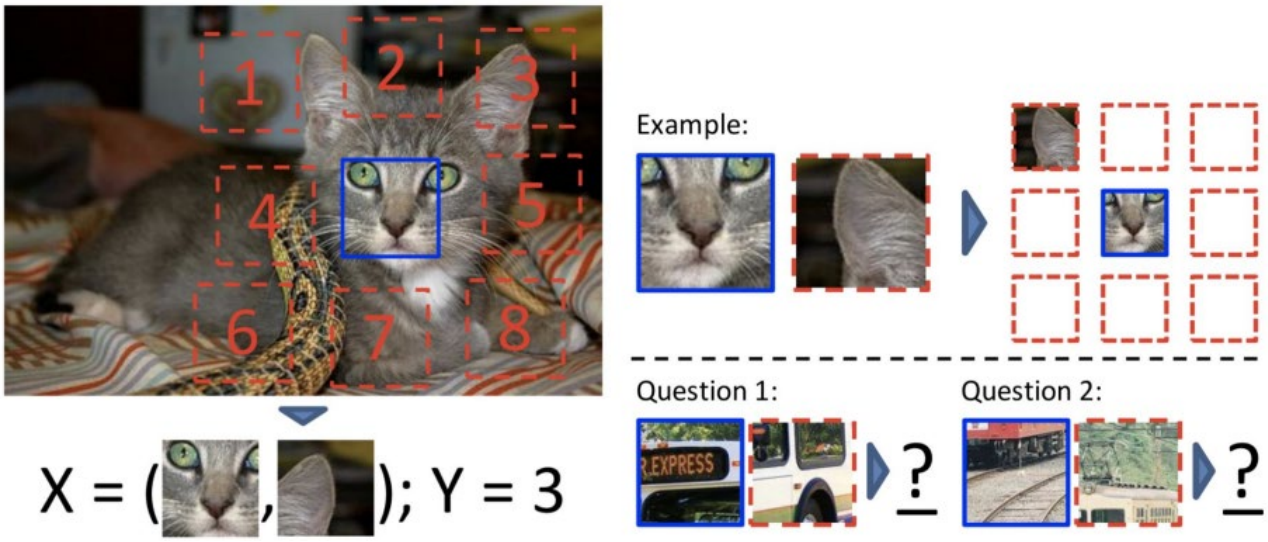
In Image SSL, we could apply several augmentation mechanisms such as rotation and distortion.

Several papers have been published for these tasks, such as Exemplar-CNN which applies data augmentations directly on patches for the Convolutional Layers or “Self-Supervised Learning by rotating the entire input images” where we teach our model to learn rotated representation. This is useful because our model will not output the class (Bird, Cat, Mouse, Dog, etc.) but instead will output a number $[0, 4]$ which tells us how much the image was rotated $(0 = 0°, 1 = 90°, 2 = 180°, 3 = 270°)$.





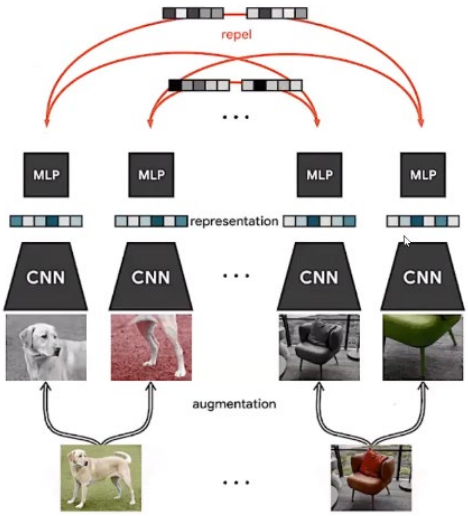
Then there is another technique which works with patches, meaning that we are giving in input the patches (extracted from the same image) and not the whole input and giving as output not the label (Dog, Cat, etc.) but the relative position of that patch. It can be easily understood through this scheme.



Lastly, the Jigsaw Puzzle Network, where essentially, I divide the image in patches and then shuffle all the patches. Now I want the network to output the correct ordering of those patches in order to improve the aware of the feature extraction representation learner.

6. SimCLR (Similarity Contrastive Learning Representation)

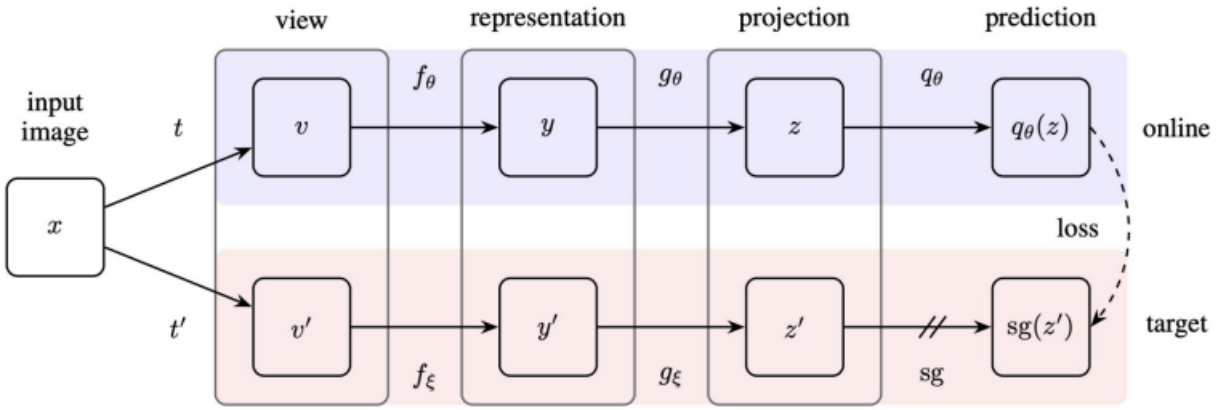
Here, essentially, we take two images, we augment them (clip, colorize, decolorize, rotate, flip, distortion, etc.), then we build a representation through the same CNNs (which share the same parameters), then through the MLP we project these representations into another space. From this space we want the loss to tell us that these 4 representations we just created are different in pairs. The image is self-explanative.



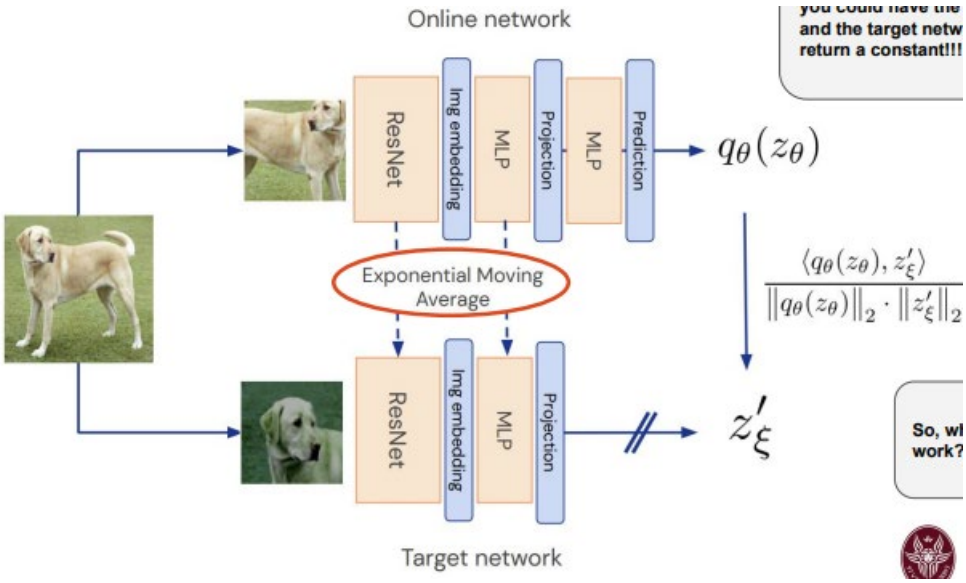
Here now we have 6 losses (basically SoftMax): 1 for the dog image, 1 for the chair image and 4 for the possible combinations we have at the last representation (orange lines).

7. BYOL (Bootstrap Your Own Latent)

BYOL was the first SSL architecture which removed the need to mine hard negatives, basically becoming the SOTA (2020) without using contrastive learning!



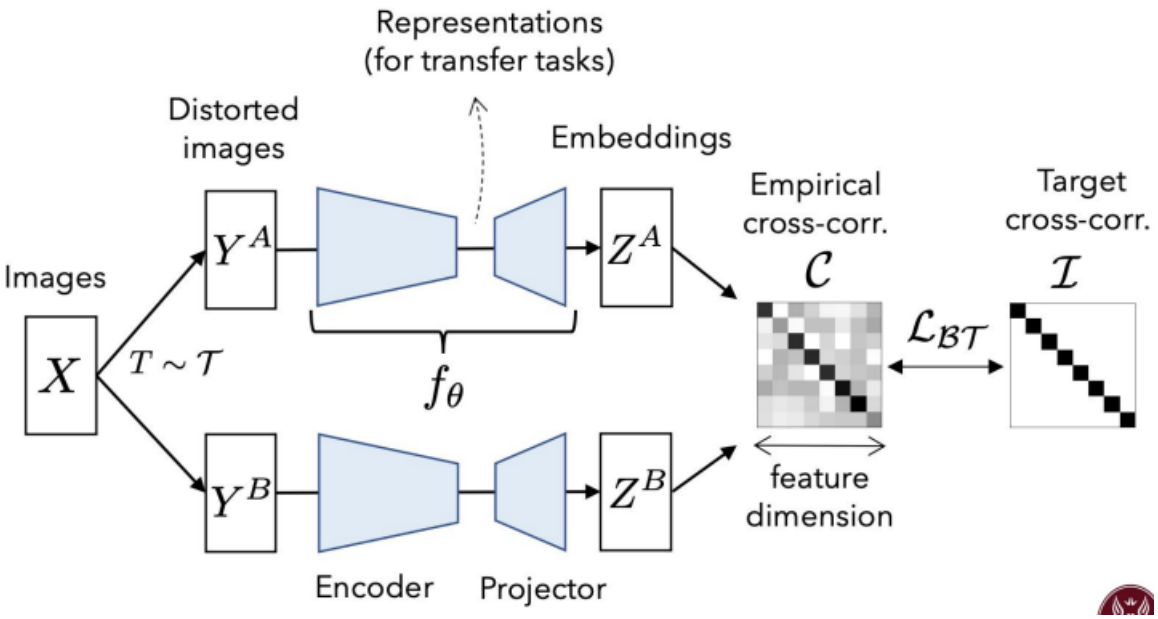
Given an input x, we have two branches t, t'. One is used to actually learn the parameters, while the other one is used to adjust the learnt parameters. Let's look at another perspective of this architecture:



We have a dog and we compute the representation using the Online Network (above), getting a representation $q_\theta(z_\theta)$. Now, we use the lower network (Target) for computing the target which is: Taking the same image, apply different augmentation techniques, applying the same architecture up to PROJECTION point (not applying last MLP) and so I will have another representation. Now I want z'_ξ to be as close as possible to $q_\theta(z_\theta)$. For doing so, we compute the loss and backpropagate ONLY through the Online Network. But now, for updating the weights in the Target Network, we don't do it through back-propagation, but by taking the current values of the weights of the Target Network, the just-computed weights of the Online one and we compute the moving average. Essentially, it's the average of the weights of K previous iterations. We don't need negatives because we are not backpropagating through the Target one and we make sure that the weights are not zero because we are computing a moving average so the values will always be updated!

8. Barlow Twins

Even in this case, we do not deal with negatives, but we apply a different trick where we work with representations as probability distribution. We take the image, compute the embeddings Z^A, Z^B and then through Empirical Cross-Correlation, we want to retrieve an Identity Matrix (ideally I, since it means that the two probability representations are correlated). So, the loss will simply be how distant is the cross-correlation of these two signals from the Identity.



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

C_{ii} is a regularization term, where we want the covariance terms to be small only not on elements on the diagonal, because of course we want the elements on the diagonal to be as close as possible to the identity.

9. Deep Metric Learning

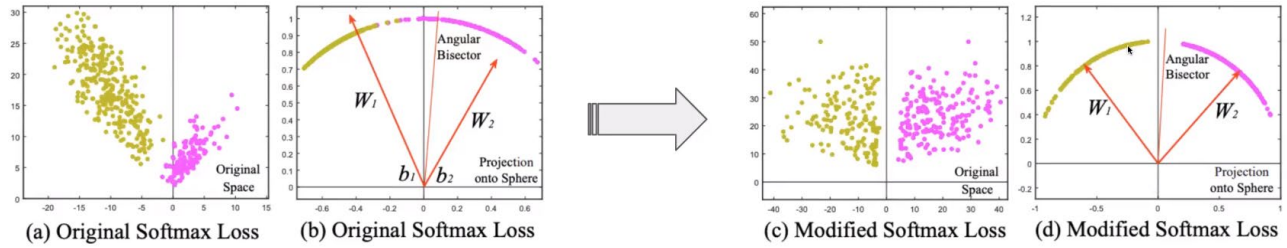
Deep Metric Learning aims to create a representation function that maps objects into an embedded space. In this space, the distance between objects should reflect their similarity, bringing similar objects close and pushing dissimilar objects apart. The commonly used Triplet Loss suffers from two issues:

1. **Expansion Issue**: It's challenging to guarantee that samples with similar labels will be grouped closely in the embedded space.
2. **Sampling Issue**: Deep Metric Learning methods heavily rely on complex sample mining techniques to select the most informative samples for each training batch.

When we talk about Center Loss, we talk about that loss which separates classes as if they were some sort of long blobs, as we can see on the right plot.

By simply modifying the SoftMax (adding a regularization term), we try to push classes to be close to their centers, having a better separation. So, we compute the centroids of all the classes and we want that samples from those classes are related and pushed towards the centroid of their respective classes (having a lot of density in the center) as we can see in the plot below.

The problem in Center Loss is how do we compute the centers? To solve this problem there is a solution proposed, called **SphereFace**, which iteratively refine the centroid by computing it, essentially modifying the SoftMax function as we can see in the plot below. If we use the normal SoftMax, the angles of the projection back onto the manifold will not be clearly separated, while by using SphereFace, we do have the same angle, thus achieving centered projections.



(a) Original Softmax Loss (b) Original Softmax Loss (c) Modified Softmax Loss (d) Modified Softmax Loss