NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

PROGRAM OF POSTGRADUATE STUDIES

Masters Thesis

Self-supervised Metric Learning

Ioannis I. Misios

ATHENS

SEPTEMBER 2022

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

Διπλωματική Εργασία

Αυτο-επιβλεπόμενη Μάθηση Μετρικής

Ιωάννης Η. Μίσιος

ΑΘΗΝΑ

ΣΕΠΤΕΜΒΡΙΟΣ 2022

Masters Thesis

Self-supervised Metric Learning

Ioannis I. Misios
A.M.:DS1200007

SUPERVISOR: Yannis Avrithis, Research Director, ATHENA Research and Innovation Center

EXAMINATION COMMITTEE:

Yannis Avrithis, Research Director, ATHENA Research and Innovation Center

Ioannis Emiris, President and General Director, ATHENA Research and Innovation Center

Vasileios Katsouros, Research Director, ATHENA Research and Innovation Center

SEPTEMBER 2022

Διπλωματική Εργασία

Αυτο-επιβλεπόμενη Μάθηση Μετρικής

Ιωάννης Η. Μίσιος
Α.Μ.:DS1200007

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: Γιάννης Αβρίθης, Διευθυντής Έρευνας, Ερευνητικό κέντρο ”ΑΘΗΝΑ”

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ:

Γιάννης Αβρίθης, Διευθυντής Έρευνας, Ερευνητικό κέντρο ”ΑΘΗΝΑ”

Ιωάννης Εμίρης, Πρόεδρος και Γενικός Διευθυντής, Ερευνητικό κέντρο ”ΑΘΗΝΑ”

Βασίλειος Κατσούρος, Διευθυντής Έρευνας, Ερευνητικό Κέντρο ”ΑΘΗΝΑ”

ΣΕΠΤΕΜΒΡΙΟΣ 2022

ABSTRACT

Metric learning is an important paradigm for a variety of problems in machine learning and computer vision. It has been successfully employed for fine-grained classification, retrieval, face recognition, person re-identification and few-shot learning, among other tasks. Metric learning is an approach based on a distance metric that aims to determine similarities or dissimilarities between samples. The goal is to reduce the distance between similar samples and at the same time to increase the distance of dissimilar ones. Therefore, it is crucial that the distance measure is learnable to adapt to data from different domains.

Training a Convolutional Neural Network to distinguish similar from dissimilar images requires some kind of supervision. In the era of big data, due to limited human-powered annotated data, deep learning methods are recently adapted to work without supervision. Self-supervised methods can be considered as a special form of unsupervised learning methods with a supervised form, where supervision is induced by self-supervised tasks rather than predetermined prior knowledge. Unlike a completely unsupervised setting, self-supervised learning uses information from the dataset itself to generate pseudo-labels.

In this work we consider some self-supervised metric learning methods which use different sample mining techniques as well as loss functions to investigate its effectiveness in both using pre-trained network on ImageNet and initialized from scratch. The evaluation is performed on four benchmark metric learning and retrieval datasets. It appears that soft loss functions that exploit contextual similarities between samples outperform hard ones that use pairwise similarities. Furthermore, it seems that augmented versions of the original images can be used as positive pairs to initiate the self-supervised training process.

SUBJECT AREA: Computer Vision, Deep Learning

KEYWORDS: Metric Learning, Neural Networks, Self-Supervised Learning

ΠΕΡΙΛΗΨΗ

Η Μάθηση Μετρικής είναι ένα σημαντικό παράδειγμα για μία πληθώρα προβλημάτων της Μηχανικής Μάθησης και της Όρασης Υπολογιστών. Έχει επιτυχημένα εφαρμοστεί σε εφαρμογές όπως η λεπτομερής ταξινόμηση, ανάκτηση πληροφορίας, αναγνώριση προσώπου κ.α. Αφορά την εκμάθηση μιας μετρικής απόστασης που βασίζεται στον προσδιορισμό ομοιοτήτων ή ανομοιοτήτων μεταξύ των δειγμάτων. Στόχος της είναι να μειωθεί η απόσταση μεταξύ παρόμοιων δειγμάτων και ταυτόχρονα να αυξηθεί η απόσταση μεταξύ ανόμοιων. Ως εκ τούτου, είναι σημαντικό η μάθηση μετρικής να είναι εκπαιδευόμενη ώστε να προσαρμόζεται σε δεδομένα από διαφορετικούς τομείς.

Η εκπαίδευση ενός Συνελικτικού Νευρωνικού Δικτύου ώστε να διακρίνει παρόμοιες από ανόμοιες εικόνες απαιτεί κάποιου είδους επίβλεψη. Στην εποχή του μεγάλου όγκου δεδομένων, λόγω του περιορισμένου αριθμού των ανθρωπίνως επισημειωμένων δεδομένων, οι μέθοδοι βαθιάς μάθησης προσαρμόστηκαν να λειτουργούν χωρίς επίβλεψη.

Οι αυτοεπιβλεπόμενες μέθοδοι μπορούν να θεωρηθούν ως μια ειδική μορφή μεθόδων μάθησης χωρίς επίβλεψη με εποπτευόμενη μορφή, όπου η εποπτεία πηγάζει από αυτοεποπτευόμενες εργασίες και όχι από προκαθορισμένη προηγούμενη γνώση. Σε αντίθεση με μια εντελώς μη επιβλεπόμενη διεργασία, η αυτοεπιβλεπόμενη μάθηση χρησιμοποιεί πληροφορίες από το ίδιο το σύνολο δεδομένων για να δημιουργήσει ψευδο-ετικέτες.

Στην παρούσα εργασία εξετάζουμε ορισμένες αυτοεπιβλεπόμενες μεθόδους μετρικής εκμάθησης που χρησιμοποιούν διαφορετικές τεχνικές εξόρυξης δειγμάτων καθώς και συναρτήσεις κόστους με σκοπό τη διερεύνηση της αποτελεσματικότητάς τους τόσο στη χρήση προεκπαιδευμένου δικτύου στο ImageNet όσο και στην χρήση τυχαία αρχικοποιημένου δικτύου. Η αξιολόγηση των μεθόδων πραγματοποιείται στα πιο διαδεδομένα σύνολα δεδομένων ανάκτησης πληροφορίας και μάθησης μετρικής. Παρατηρείται πως οι ήπιες συναρτήσεις κόστους εκμεταλλεύονται τις ομοιότητες μεταξύ των δειγμάτων λαμβάνοντας υπόψιν τους γείτονές τους, έχουν καλύτερα αποτελέσματα σε σχέση με τις απόλυτες συναρτήσεις κόστους που χρησιμοποιούν τις ομοιότητες κατά ζεύγη. Επιπλέον, φαίνεται πως η τεχνητή επάυξηση των αρχικών εικόνων του συνόλου δεδομένων για την δημιουργία θετικών ζευγών μπορεί να βοηθήσει την αυτοεπιβλεπόμενη μάθηση και ιδιαίτερα στο ξεκίνημά της.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Όραση Υπολογιστών, Βαθιά Μάθηση

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Μετρική Μάθηση, Νευρωνικά Δίκτυα, Αυτο-επιβλεπόμενη Μάθηση

ACKNOWLEDGEMENTS

This work was supported in part by the Institute of Advanced Research in Artificial Intelligence (IARAI). First and foremost, I would like to thank Yiannis Avrithis from the bottom of my heart as a person and as a professor. Special thanks to postdoctoral students Bill Psomas and Shashanka Venkataramanan for valuable guidance and assistance.

In addition, I would like to thank all the people at IARAI for creating a really pleasant and warm working environment.

Finally, a big thank you to my family and Semina for all their love and support.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Motivation

Computer vision is a field of Artificial Intelligence (AI) that enables computers and systems to obtain meaningful information from images, videos, and other visual inputs, and act or provide recommendations based on that information. Computer Vision requires a lot of data. It uses machine learning techniques to analyse the data multiple times until it can discern the difference and eventually the image.

Machine Learning uses algorithmic models that allow computers to understand the context of visual data on their own. If enough data is fed through the model, the computer can ”look” at the data and teach itself the ability to distinguish images. In the field of Machine Learning, the concept of distance has been widely used since its inception. It provides a measure of similarity between data, where the data that are close to each other should be as similar as possible, and the data that are far away should be as different as possible. An application of this idea of similarity learning to classification problems is the well-known nearest neighbors (NN) classification, which assigns the class of the test sample to the class of the training sample that is closest to it. This idea of nearest neighbor classification gave birth to distance metric learning.

Euclidean distance is widely used by metric learning algorithms as a concise and effective metric tool. However, a single form of distance metric cannot be universal to all practical problems. Therefore, metric learning hopes to combine the characteristics of the data to learn an effective metric to solve the target problem.

The emergence of metric learning algorithms has greatly improved the performance of distance-based classifiers, distance-based clustering for unsupervised problems and feature dimensionality reduction. Then, with the rapid development of deep learning, metric learning combined with the advantages of deep neural network in semantic feature extraction and end-to-end training has gradually attracted people’s attention.

Compared with classical metric learning, deep metric learning can do nonlinear mapping of input features, and has been widely used in the field of Computer Vision such as image retrieval, face recognition, person re-identification, *etc*. Besides, for some extreme classification tasks (with a large number of categories, but only a few samples per category), deep metric learning still performs well. For example, based on deep metric learning, FaceNet [1] has surpassed human performance in the face recognition task of 8M individuals and 260M images.

Standard deep metric learning constrains the intra-class distance and widens the between classes distance by mining pairs or triplets of positive and negative samples. This brings challenges to the sampling of training samples. Due to the extremely large number of training samples, only meaningful samples can be mined to participate in training. If the selection of negative samples is too difficult, the training will be unstable. If the selection is too simple, the loss function will have no gradient, which is not conducive to the convergence of the model.

In the era of big data, typically a deep learning model is trained fully supervised for a specific task using a large, manually labeled dataset that is randomly divided into training, validation, and test sets.

However, supervised learning has its bottlenecks. Not only does it rely heavily on expen-

sive manual labeling, but it also suffers from generalization errors, spurious correlations, and adversarial attacks. We want the neural network to learn more with fewer labels, fewer samples, or fewer trials. As a promising candidate, self-supervised learning has attracted a lot of attention due to its excellent data efficiency and generalization ability.

Self-supervised learning can be generalized into two classical definitions:

- Obtain ”labels” from the data itself through a ”semi-automatic” process.
- Predict one part of the data from the other part of the data.

Specifically, the ”other parts” hereof may be incomplete, distorted and generally augmented images. In other words, the machine learns to ”recover” all, some, or some features of the original input image.

Self-supervised learning can be seen as a branch of unsupervised learning as it does not involve manual labeling. However, in a narrow sense, unsupervised learning focuses on detecting specific data patterns, such as clustering or anomaly detection, whereas self-supervised learning aims to recover, which is still in the paradigm of a supervised setting.

A common feature between metric learning and self-supervised learning is the contrastive part. In this work we focus on *Contrastive* self-supervised methods and especially to context-context comparisons. Context-context contrastive learning investigates the relationship between global representations of different samples, like metric learning. At the beginning, researchers were generating pseudo-labels through cluster-based discrimination and achieved fairly good performance on representation learning. An example of a cluster-based method is the DeepCluster [2]. More recently, self-supervised contrastive methods like MoCo [3], SimCLR [4], *etc*. through direct comparisons between contexts and under linear classification, obtained results comparable to supervised methods.

The main challenge of Contrastive learning is the sample mining. Sample mining is the process where we search either in the whole dataset or within the mini-batch to find positives or negatives of an anchor image. Those samples together with the anchor image form pairs or triplets. Pairs or triplets are used to to train a DNN to learn a discriminative embedding space where similar examples are closer and dissimilar examples are apart. Taking into account that no-labels are available, sample mining becomes challenging.

A common idea to construct positive pairs is to randomly augment (crop, resize, flip, *etc*.) the original input image. It is assumed that the embedding of the original image should be close to that of the augmented. In addition, a simple proposal for negative mining is to assume that every sample within the mini-batch belongs to a different class. This technique is called instance discrimination. Finally, there are other proposals for positive and negative mining which are based on graphs.

In this work we conduct an extensive search in three self-supervised contrastive learning methods which use different sampling methods for positive and negative mining. The first one [5] uses the augmenation process as well as instance discrimination technique for sample mining. The second method [6] combines the instance discrimination technique and graph search to mine informative positives and negatives. The latter [7] uses the augmentation process to create positives and the self-distillation technique to mine more positives and negatives by exploiting the contextual similarity between samples. They are trained and then evaluated in four benchmark metric learning datasets. A common feature of those datasets is that the predefined training classes are different from testing ones. Therefore, we evaluate the ability of the methods to generalize not only to unseen data

but also to unseen classes. Finally, all three methods will be evaluated in both tasks of training a randomly initialized network and fine-tuning a pre-trained network on ImageNet.

1.2 Structure

In this work we attempt to make fair comparisons between the three self-supervised metric learning methods in order to investigate the advantages and disadvantages of each method in the tasks of training a randomly initialized CNN or fine-tuning a pre-trained CNN on ImageNet.

- chapter 1 gives an introduction to how metric learning can be combined with self-supervised learning for representation learning, while presenting the challenges and contribution of this work.
- chapter 2 presents the evolution of metric learning alongside the development of Deep Neural Networks as well as a detailed overview of Self-Supervised Learning.
- chapter 3 presents a detailed analysis of the selected Self-Supervised Metric Learning methods.
- chapter 4 presents the datasets, experimental setup and the evaluation protocol.
- chapter 5 presents the experimental results as well as a discussion of the findings.
- chapter 6 presents possible avenues of research in the area of Self-Supervised Metric Learning.

2. BACKGROUND

The goal of Metric Learning is to learn a distance metric or a similarity function, where the data that are close to each other should be as similar as possible and the data that are far away should be as different as possible. In this chapter we present the evolution of Metric Learning into Deep Metric Learning alongside the development of Deep Learning models and especially CNNs. Finally, we present the foundations of Self-Supervised Learning and how it can be combined with Deep Metric Learning to perform unsupervised.

2.1 The evolution of Deep Neural Networks

In this section the chronological evolution of deep neural network architectures is presented.

Yann LeCun in 1998 proposed **LeNet5** [8] which was one of the first convolutional neural networks. The architecture of LeNet5 is fundamental, especially for two of its insights. Image features are distributed over the entire image, and the introduction of convolution learnable parameters which is an efficient way to extract similar features at multiple locations with fewer parameters. There was no GPU for training at the time and the CPUs were slow. LeNet5 illustrates that since images are highly spatially correlated, these correlations cannot be exploited using individual pixels of an image as individual input features. The main contributions of *LeNet5* were the use of convolutional layer in sequence with pooling and non-linear activation function (tanh) as well as an Multi Layer Perceptron (MLP) as a final classifier.

In 2012, Alex Krizhevsky published **AlexNet** [9], a deeper and wider version of LeNet5. AlexNet extends the insights of LeNet to a much larger network that can be used to learn more complex objects and object hierarchies. The contributions of this work are the use of the *ReLU* activation function, the dropout technique to selectively ignore individual neurons in training to avoid model overfitting and the max pooling to avoid the averaging effects of the average pooling. At the time, GPUs provided more cores than CPUs, which could speed up training by a factor of 10, allowing the use of larger datasets and larger images.

Oxford's **VGG** [10] network pioneered the use of smaller $3x3$ kernels at each convolutional layer. This seems to violate the principle of LeNet, where large convolutions were used to extract image features. However, in VGG they discovered that multiple $3x3$ convolution layers can simulate larger, more receptive structures, such as $5x5$ and $7x7$ convolutions.

Google's Christian Szegedy began to explore reducing the computational cost of deep neural networks by proposing the first *Inception* architecture named **GoogLeNet** [11]. The great insight of Inception is to use 1×1 convolutional blocks to perform dimensionality reduction. This is often referred to as the "bottleneck". Inception's bottleneck layer reduces the number of features in each layer, thereby reducing the amount of computation and keeping the inference time low.

In December 2015, **ResNet** [12] was born sparking a revolution in network architecture. The concept of ResNet (residual network) is simple; pass the output of two consecutive convolutional layers plus the input that skips these two layers to the next layer. In this way the vanishing gradient effect was mitigated, allowing the training of deeper network architectures. It was also the first time a network with more than a hundred layers was

trained.

2.2 Metric Learning

Different datasets have also different classification or clustering problems. Therefore, it is crucial that the distance measure is learnable to adapt to data from different domains. This demonstrated the need for metric learning. Metric learning is an approach based on a distance metric that aims to determine similarities or dissimilarities between samples. The goal is to reduce the distance between similar samples and at the same time to increase the distance of dissimilar ones.

Mahalanobis distance is the common part of the fundamental studies on metric learning [13], [14]. Formulating the problem, let a dataset $\mathcal{X}$ containing the samples $x_1, \ldots, x_N$. The distance between the samples x_i and x_j is calculated as: $d_M(x_i, x_j) = \sqrt{(x_i - x_j)^T M (x_i - x_j)}$. $d_M(x_i, x_j)$ is a metric (distance), therefore it should have the properties of non-negativity, symmetry and triangle inequality. Furthermore, covariance matrix M should be symmetric and positive semi-definite. We can decompose M as follows: $M = W^T W$

$$\begin{aligned} d_M(x_i, x_j) &= \sqrt{(x_i - x_j)^T M (x_i - x_j)} \\ &= \sqrt{(x_i - x_j)^T M (x_i - x_j)} \\ &= \sqrt{(x_i - x_j)^T W_T W (x_i - x_j)} \\ &= \|W x_i - W x_j\|_2^2. \end{aligned} \tag{2.1}$$

Equation 2.1 shows that W matrix is linear transformable. With respect to this property, the Euclidean distance of two samples in the transformed space is equal to Mahalanobis distance in original space.

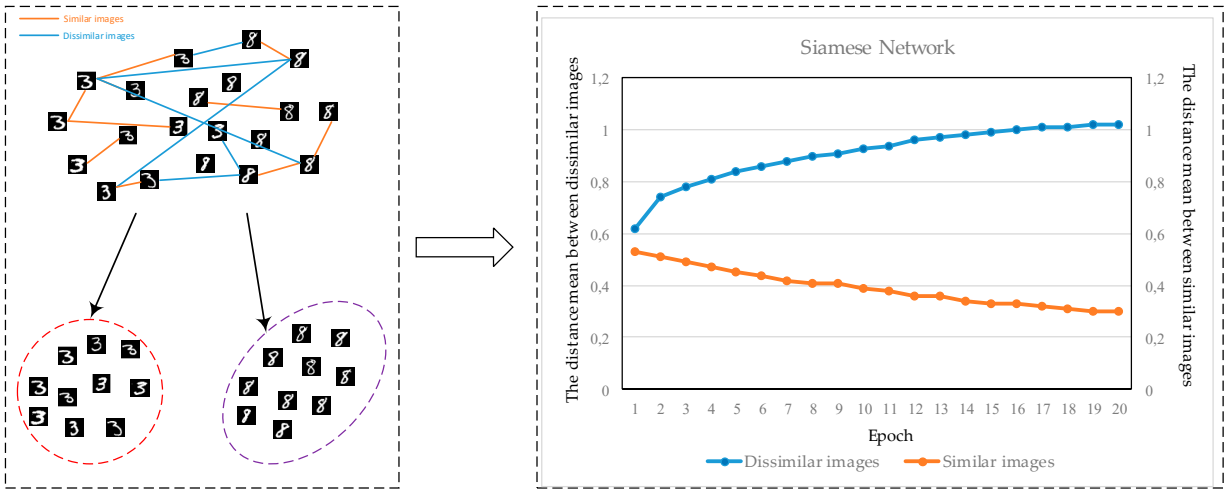
An example of a classical dimensionality reduction method which is highly correlated to Mahalanobis distance is the Principal Component Analysis (PCA). PCA is a linear transforation that attempts to map the data to a lower dimensional space while preserving the highest possible variance of the data.

In general, linear metric learning approaches provide soft constraints in the transformed data space improving learning performance. An advantage of these approaches is that they avoid overfitting.

When the Mahalanobis distance is used as a metric, some linear features in the data can be obtained due to the projection using the linear matrix W. But nonlinear features in the data are also important when comparing distances between objects. Therefore, some metric learning methods for nonlinear features have been proposed, such as kernel methods [15] [16] . However, the main problem with these methods is that they easily lead the model to overfit. With the introduction of deep metric learning, the problems of both linear metric learning as well as kernel based non-linear metric learning were solved.

2.3 Deep Metric Learning

For traditional metric learning, due to its limited ability to process raw data, it is necessary to first use the knowledge of feature engineering to preprocess the data, and then use the metric learning algorithm to learn. Some traditional metric learning methods can only learn linear features. Although kernel methods that can extract nonlinear features have been proposed, the learning effect is not significantly improved. With the advent of deep learning, thanks to the excellent ability of activation functions to learn nonlinear features, deep learning methods can automatically learn high-quality features from raw data. More specifically, in Computer Vision applications, CNNs are mainly used to perform the nonlinear mapping of input images into a lower dimensional space while preserving the semantic information. Figure 1 illustrate the training process of a Siamese DNN with Contrastive loss where the distance between similar samples decreases while simultaneously increasing the distance between heterogeneous samples.



(a) (b)

Figure 1: a) Desired discriminated embedding space. b) Mean distance between the two digits while training progresses.

Deep metric learning mainly consists of three aspects , which are:

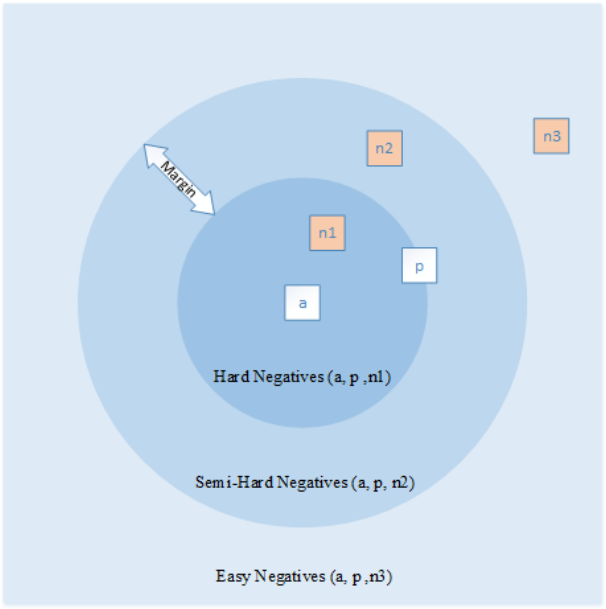
- Sample mining
- Model structure
- Loss function

2.3.1 Sample mining

The easiest way to think of sample mining is to randomly sample pairs of positives and negatives. However, the sample pairs collected by this method are not difficult to distinguish, and the model cannot learn enough informative knowledge from these data. Therefore, it is necessary to adopt some sample mining methods to find indistinguishable sample pairs from the dataset.

A typical set of samples consists of anchors, negatives and positives. Positive is a positive sample with the same class as the anchor, and negative is a sample different from the

anchor class. According to the different distances between anchors, positive samples and negative samples, sample mining can be divided into three categories as shown in Figure 2. In *Hard Negative Mining* approach, we use the false positive samples obtained after training on the training set as negative samples. Furtermore, in *Semi-Hard Negative Mining*, we try to find negative samples within a margin range. Finally, in *Easy Negative Mining*, we take as negatives the samples found outside a certain margin.



Hard Negative Mining
$d(a,n) < d(a,p)$

Semi-Hard Negative Mining
$d(a,p) < d(a,n) < d(a,p) + margin$

Easy Negative Mining
$d(a,p) + margin < d(a,n)$

Figure 2: Negative Mining.

There are a few advantages of sample mining. First of all, it is easier for the model to learn useful knowledge, which helps to improve its discrimination ability. Furthermore, it helps to avoid overfitting. If the model continues to see easily distinguishable samples, it is easy to overfit and fall into a local maximum. Finally, It is beneficial to reduce the time complexity of training process. Traversing all (anchor, positive, negative) triples in the data requires $O(n^3)$ time complexity, and selecting a small amount of indistinguishable data to train the model can achieve the same effect.

2.3.2 Model structure

Typical contrastive learning model structures are the Siamese and Triplet Networks. The model structure diagram is shown in Figure 3. The structure of the *Siamese* network consists of two identical networks that share the model weights. Each network is fed with an image, and if the images belong to the same class then the distance between their embeddings should be minimized. Iv contrast, if the images belong to different classes, the distance between their embedding vectors should be maximized. The *Triplet* network structure is inspired by the *Siamese* structure. It consists of three identical networks with shared weights. Each network is fed with an image, where one of them plays the role of the anchor. The second image should belong to the same class as the anchor, while the third one belongs to a different class. The goal is that the distance D_1 between the anchor and the positive image differs from the distance D_2, between the anchor and the negative pair, by a margin a, where $a > 0$.

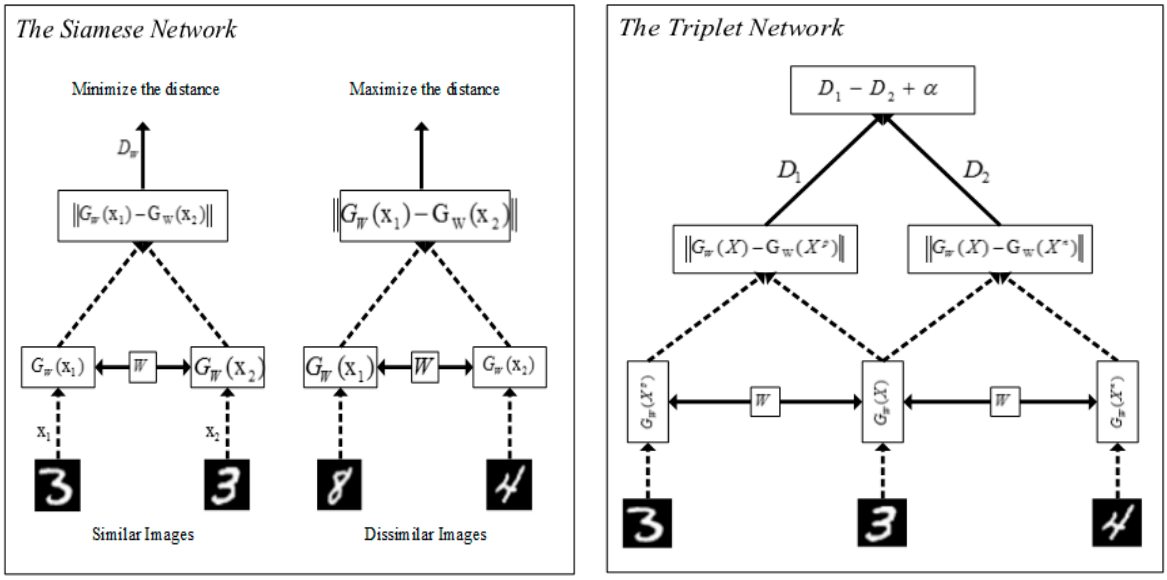


Figure 3: The Siamese and Triplet Networks.

2.3.3 Loss function

In this section we introduce the most common embedding and classification loss functions.

Contrastive Loss [17]: The idea behind *Contrastive* loss is that the distance between the positive sample and the anchor decreases, while the distance between the negative sample and the anchor increases. It requires sampling one positive or one negative for an anchor.

$$\mathcal{L}_{\text{Contrastive}} = (1 - \mathcal{I}_{ij})[S_{ij} - \lambda]_+ - \mathcal{I}_{ij} S_{ij}, \tag{2.2}$$

where $\mathcal{I}_{ij} = 1$ indicates a positive pair, while $\mathcal{I}_{ij} = 0$ indicates a negative pair. $\mathcal{S}_{ij}$ is the cosine similarity between the embeddings of two samples x_i and x_j.

Triplet Loss [18]: The idea behind *Triplet* loss is to create a distance α between positive and negative samples , so that there is a certain degree of discrimination in the feature space of positive and negative samples, which is convenient for the model to distinguish. It requires sampling one positive and one negative for an anchor.

$$\mathcal{L}_{\text{Triplet}} = [S_{an} - S_{ap} + \alpha]_+, \tag{2.3}$$

where S_{ap} indicates the cosine similarity between the embedding of the anchor with that of the positive. Similarily, S_{an} indicates the cosine similarity between the anchor and the negative. α is a positive user defined magrin.

NPair Loss [19]: During the process of updating the learning parameters, the *Triplet* loss compares only one negative sample, while ignoring the negative samples of other classes. Therefore, it can only encourage the query embedding vector to maintain a large

distance from the selected negative sample, but cannot guarantee that it also maintains a large distance from other unselected negative samples.

NPair loss improves the above problems of Triplet Loss. Unlike *Triplet* loss, which uses a single positive and negative sample, the *NPair* loss function uses the structural information between data to learn more discriminative representations. More specifically, it considers the relationship between the query sample and multiple other negative samples of different classes at the same time, so as to maintain the distance between the query and all other classes, which can speed up the convergence of the model.

$$\mathcal{L}_{\text{NPair}} = \frac{1}{m}\sum_{i=1}^{m}\log\left(1+\sum_{y_k\neq y_i, y_j=y_i} e^{S_{ik}-S_{ij}}\right), \tag{2.4}$$

where m is the number of samples in the mini-batch.

Lifted Structure Loss [20]: *Lifted Structure* loss calculates the loss based on all positive and negative sample pairs in the mini-batch.

$$\mathcal{L}_{\text{LiftedStructure}} = \sum_{i=1}^{m}\left[\log\sum_{y_k=y_i} e^{\lambda-S_{ik}} + \log\sum_{y_k\neq y_i} e^{S_{ik}}\right]_+, \tag{2.5}$$

where λ is a user defined margin. The difference between *Lifted Structure* loss and *Triplet* loss is that the sample triplet of *Triplet* oss is determined in advance, while *Lifted Structure* loss dynamically constructs the most difficult triplet for each positive sample pair. During the construction process all negative samples within the mini-batch are considered.

ProxyNCA Loss [21]: The *ProxyNCA* loss assigns a proxy per class so that the number of proxies is the same as the number of class labels. Given an input sample as an anchor, the anchor together with the samples of the same class are considered positives while the other anchors are considered as negatives. Let x denote the input embedding vector, p^+ is a positive proxy, p^- is a negative proxy. The loss is given by

$$\mathcal{L}_{\text{ProxyNCA}} = \sum_{x\in X} -\log\frac{e^{S(x,p^+)}}{\sum_{p^-\in P^-} e^{S(x,p^-)}}, \tag{2.6}$$

where X is a mini-batch of embedding vectors, P^- is the set of negative anchors, and S represents the cosine similarity between the two embedding vectors.

Multi-Similarity Loss [22]: Sample mining can be understood as assigning a weight to each sample pair during the learning process. The core of assigning weight to samples is to judge the local distribution of samples, that is, the similarity between them. The distribution of samples does not only depend on the distance or similarity between the current two samples, but also depends on the relationship between the current sample pair and its surrounding sample pairs. Therefore, for each sample pair, we need to consider not only the self-similarity of the sample pair itself, but also its relative similarity to the other sample pairs. Relative similarity can be divided into positive relative similarity and negative relative similarity. *Self-similarity* is the similarity calculated from the pair sample itself. In

Positive relative similarity we consider not only the similarity of the current sample pair, but also the relative similarity between positive sample pairs in the local neighborhood. In *Negative relative similarity* we consider not only the similarity of the current sample pair, but also the relative similarity between negative sample pairs in the local neighborhood. *Multi-Similarity Loss* comprehensively considers all three similarities that generalize most current sample pair-based loss functions

$$\mathcal{L}_{\text{MultiSimilarity}} = \frac{1}{m}\sum_{i=1}^{m}\left\{\frac{1}{\alpha}\log\left[1+\sum_{k\in\mathcal{P}_i}e^{-\alpha(S_{ik}-\lambda)}\right]+\frac{1}{\beta}\log\left[1+\sum_{k\in\mathcal{N}_i}e^{\beta(S_{ik}-\lambda)}\right]\right\}, \tag{2.7}$$

where α, β, λ are hyper-parameters, $\mathcal{P}_i$ and $\mathcal{N}_i$ are the sets of positives and negatives respectively.

2.4 Self-supervised Learning

Self-supervised methods can be considered as a special form of unsupervised learning methods with a supervised form, where supervision is induced by self-supervised tasks rather than predetermined prior knowledge. Unlike a completely unsupervised setting, self-supervised learning uses information from the dataset itself to generate pseudo-labels. In terms of representation learning, self-supervised learning has great potential to replace fully supervised learning. The nature of human learning tells us that large annotated datasets may not be necessary and we can learn spontaneously from unlabeled datasets.

Self-supervised learning mainly uses auxiliary tasks (pretext) to mine its own supervision information from large-scale unsupervised data, and trains the network through this constructed supervision information, so that it can learn valuable representations for downstream tasks.

Some well-known pretext tasks are: 1) predicting relative location of two patches [23] 2) solving jigsaw puzzle [24] 3) colorizing an image [25] 4) rotation prediction [26]

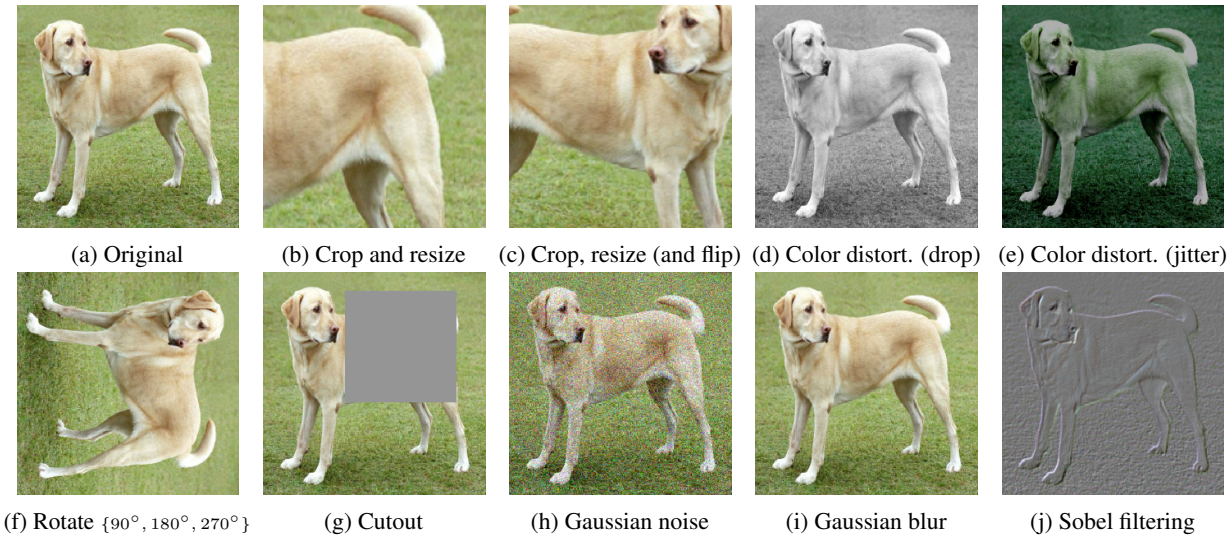
In this work we focus mostly on the *Contrastive* Self-Supervised Learning (SSL) methods.

SimCLR A Simple Framework for Contrastive Learning of Visual Representations [4]

In an image classification task every image has a label. If two samples have the same label they compose a positive pair, while if they have different labels they compose a negative pair. However, label information defeats the purpose of SSL, so we have to find a solution that deals with unlabeled data. SimCLR's approach is to treat each image as a separate class and augment it to generate samples of the same class. For example, each sample in Figure 4 is a positive sample with respect to the original image.

Interestingly, data augmentation plays a important role in the accuracy of self-supervised models. The authors experiment with various data augmentations and propose three augmentation methods with the highest accuracy which are 1)the sequence of Crop, Resize, Flip 2) Color Distortion and 3) Gaussian Blur.

These data augmentation techniques can generate different versions of an image that can improve SSL performance. The concept of SimCLR is that first you can randomly sample



(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

Figure 4: Different types of image augmentations constructing positive pairs.

a mini-batch and augment each image twice. As a result, in the learned embedding space, different views of the same image should be close while views of different images should be far away.

MoCo Momentum Contrast for Unsupervised Visual Representation Learning [3]

The SimCLR method used large mini-batches during training which is computationally demanding. MoCo tried to solve this problem by introducing a dictionary which is treated as a queue, and the size of the queue can be larger than the mini-batch, which is a hyper-parameter. The queue is updated gradually. On each iteration, the current mini-batch is enqueued, and the oldest mini-batch in the queue is dequeued.

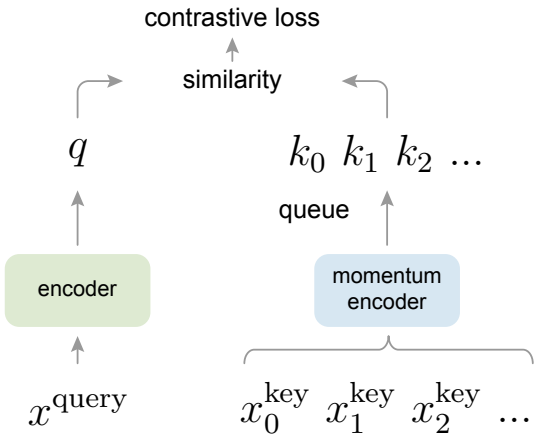


Figure 5: Momentum Contrast (MoCo) training method.

The use of queues can make the dictionary very large, like memory banks, and it is difficult to update the encoder through backpropagation. The simplest way is to share the query encoder and key encoder, however in this way the consistency of the keys in the dictionary would be poor, because the encoder would be updated quickly after each iteration. MoCo proposed the momentum update strategy to solve this problem (see Figure 5.

SwAV Unsupervised Learning of Visual Features by Contrasting Cluster Assignments [27]

Contrastive learning methods before SwAV generally require pairwise comparisons, which require a large amount of computation. SwAV does not require pairwise comparisons. It compares cluster assignments under different views instead of directly comparing features. Two views are obtained from the same image after data augmentations. Then SwAV predicts the representation of view B through the code of view A (the ”code” here refers to cluster assignment), and predicts representation of A through the code of view B.

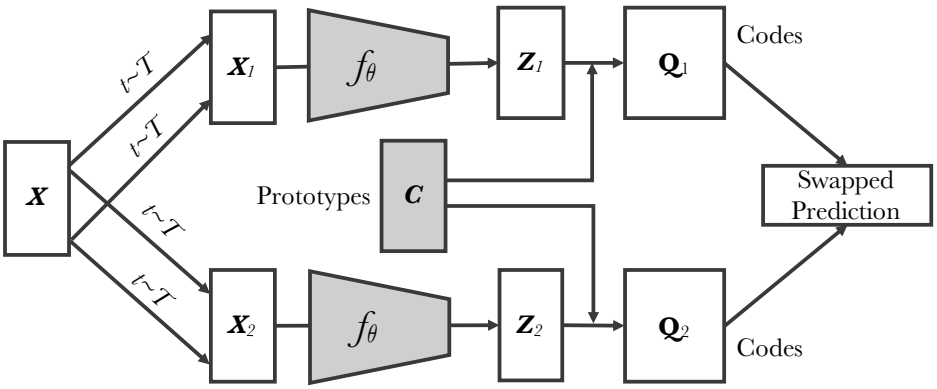


Figure 6: SwAV framework.

Before SwAV, there were also deep unsupervised learning methods based on clustering such as Deep Cluster [28] , but Deep Cluster required the calculation of cluster assignment for the entire dataset as labels for model training, which was computationally expensive. SwAV calculates the codes online, and then maintains the same codes within a mini-batch of images.

In addition to introducing clustering, SwAV also proposes a ”multi-crop” image transformation method, which further improves the effect. It is shown that more views can improve the performance of self-supervised learning models, and multi-crop uses smaller-sized images to increase the number of views without increasing the computational requirements.

BYOL Bootstrap your own latent: A new approach to self-supervised Learning [29]

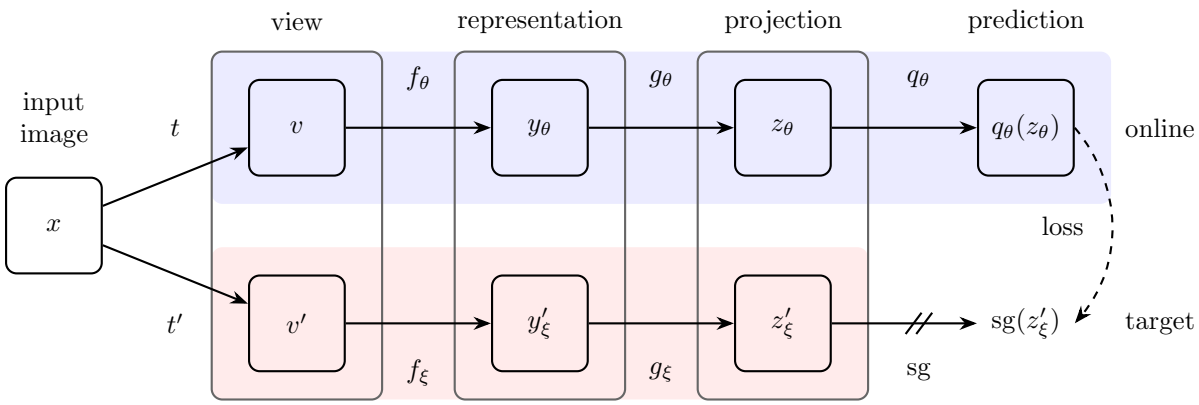


Figure 7: BYOL framework.

The goal of BYOL is to learn an image representation y_θ , which is then used for downstream tasks. As shown in Figure 7, BYOL uses two neural networks for learning, the online network and the target network. The online network is defined by a set of weight values θ and consists of three parts: encoder f_θ, Projector g_θ, and predictor q_θ. The target network has the same architecture as the online network, however it uses a different set of weights ξ. The target network provides a regression target for the online network training, and its parameters are updated as exponential moving averages of the online network parameters. BYOL aims to minimize the similarity loss between $q_\theta(z_\theta)$ and $sg(z'_\xi)$, where θ is the training weight value, ξ is the exponential moving average of θ , and sg is the stopping gradient.

SimSiam Exploring Simple Siamese Representation Learning [30]

SimSiam is equivalent to BYOL if we set to zero the moving average parameter, which means that the moving average of BYOL is not necessary. SimSiam method can even work without negative sample pairs, large mini-batch size and momentum encoders. SimSiam authors also did a lot of comparative experiments, which confirmed that the stop gradient (sg) operator plays an important role in avoiding the trivial solution.

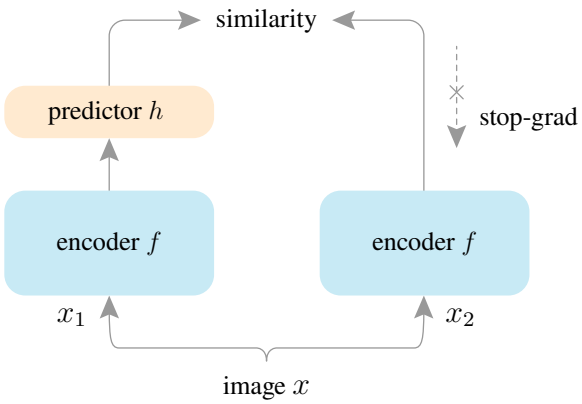


Figure 8: SimSiam framework.

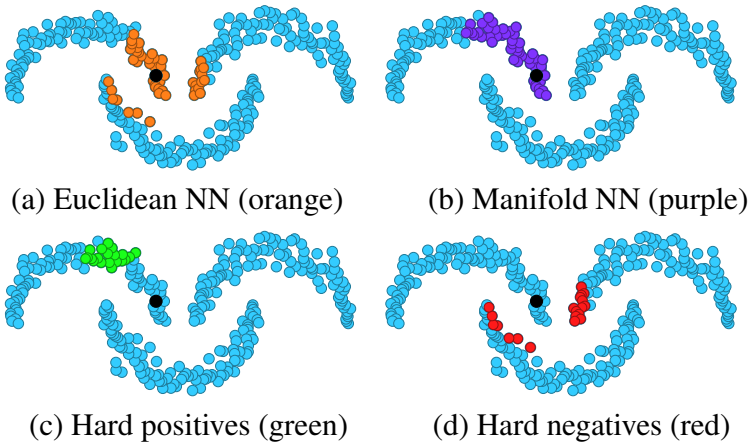
Figure 8 shows the schematic diagram of the SimSiam method. It takes as input two augmented versions x_1 and x_2 of the input image x that pass through the same encoding network f. In addition, the author also defines an MLP prediction head module h, which transforms the embedding of the one branch and matches the result of the other branch.

MOM Mining on Manifolds: Metric Learning without Labels [31]

Two distances are used in the paper, the *Euclidean* distance and the *Manifold distance*. The authors believe that for a certain anchor point, positive samples should be distributed on the same manifold, while negative samples should be distributed on different manifolds. In the newly learned embedding space, positive samples should be attracted by anchors, while negative samples should be repelled.

A pre-trained CNN is used to extract the image features and defines hard positive samples and hard negative samples based on the extracted features. Finally, it forms triplets and training is achieved by minimizing the triplet loss.

Positive pool P^+ and negative pool P^- for an anchor x^r are constructed using equations Equation 2.8 and Equation 2.9 respectively.



(a) Euclidean NN (orange) (b) Manifold NN (purple)

(c) Hard positives (green) (d) Hard negatives (red)

Figure 9: MOM intuition.

$$P^+(x^r) = \{x \in X : g(x) \in NN_k^m(y^r) \setminus NN_k^e(y^r)\}, \tag{2.8}$$

$$P^-(x^r) = \{x \in X : g(x) \in NN_k^e(y^r) \setminus NN_k^m(y^r)\}, \tag{2.9}$$

where NN_k^m denotes the k *Manifold* nearest neighbors, while NN_k^e denotes the k *Euclidean* nearest neighbors.

3. SELF-SUPERVISED METRIC LEARNING

3.1 Contrastive learning with Instance Discrimination

The first paper [5] proposes a contrastive learning method combined with the instance discrimination technique for the embedding learning problem. More specifically, it proposes a self-supervised method for learning discriminative embeddings without human annotated labels.The goal is to train a Deep Neural Network (DNN) on a huge number of unlabeled images, which is capable of extracting discriminative embeddings, such that visually similar samples are close to each other (positive concentration) and dissimilar ones are too far (negative separation) in the learned embedding space. The main challenge of unsupervised metric learning is to mine visually similar images from unlabelled ones. An instance-wise method was introduced to estimate the relationship between samples (images) . Positive pairs are generated as augmented versions of the original sample. It is assumed that the embeddings of augmented versions of the same sample should be close to the embedding space while embeddings from different samples belonging to different classes should be spread-out.

According to instance discrimination approach, it is assumed that all samples within mini-batch belong to different classes. As a result each sample within mini-batch should belong to different class. It can be seen that a mini-batch may contain samples of the same class that would be characterized wrongly as negatives. However, comparing the number of positive samples within the dataset to the negatives alongside a small mini-batch selection, minimizes the probability of false-negatives within the mini-batch.

Without label information, it is impossible to mine positive pairs, especially without a pre-trained network. Therefore, the mini-batch was doubled containing the randomly sampled images as well as an augmentation of each image. In this way, each image in the mini-batch has one positive pair meaning that the embedding of each image should have high cosine similarity with its augmented version and low similarity to all other images of the mini-batch.

The mini-batch is constructed by randomly sampling m images $\{x_1, \ldots, x_m\}$ from the training set. Each sample is randomly augmented applying transformation function T, resulting in x'_i, where $i \in [1, m]$. The mini-batch contains the original samples as well as their augmentations, *i.e.*, $\{x_1, \ldots, x_m, x'_1, \ldots, x'_m\}$.

By applying the embedding function g to each sample of the mini-batch, we obtain the *embedding*, or *feature*, y_i and y'_i for sample x_i and x'_i respectively. Considering that each sample in the mini-batch has only one positive sample, the problem is transformed into classification and solved using maximum likelihood estimation. More specifically, each original sample x_i is treated as a class and each original sample x_i or its augmented version x'_i should be classified as x_i and not as x_j for $j \neq i$.

The probability of the augmented sample x'_i being classified as x_i is

$$P(i|x'_i) = \frac{\exp(y_i^T y'_i/\tau)}{\sum_{k=1}^{m} \exp(y_k^T y'_i/\tau)}. \tag{3.1}$$

The probability of the original sample x_j for $j \neq i$ being classified as x_i is

$$P(i|x_j) = \frac{\exp(y_i^T y_j/\tau)}{\sum_{k=1}^{m} \exp(y_k^T y_j/\tau)}, \quad j \neq i. \tag{3.2}$$

Accordingly, the probability of sample x_j *not* being classified as x_i is $1 - P(i|x_j)$.

Assuming different samples are independent, the joint probability of x'_i being classified as x_i and x_j for $j \neq i$ not being classified as x_i is

$$P_i = P(i|x'_i) \prod_{j \neq i} (1 - P(i|x_j)). \tag{3.3}$$

The negative log likelihood is then given by

$$J_i = -\log P(i|x'_i) - \sum_{j \neq i} \log(1 - P(i|x_j)). \tag{3.4}$$

The loss function is defined as the sum of the negative log likelihood over all the samples in the mini-batch:

$$J = -\sum_i \log P(i|x'_i) - \sum_i \sum_{j \neq i} \log(1 - P(i|x_j)). \tag{3.5}$$

The first term of (3.5) refers to positive pairs and the second term to negative pairs. The first term is minimized by maximizing the numerator of (3.1), that is, maximizing the cosine similarity between the embeddings y_i and y'_i. This means that the original and augmented versions of the same sample must be close to each other in the embedding space. The second term is minimized by minimizing the numerator of (3.2), that is, minimizing the cosine similarity between the embeddings y_i and y_j for $i \neq j$. This achieves the *spread-out* property that embeddings of different samples are far away from each other in the embedding space.

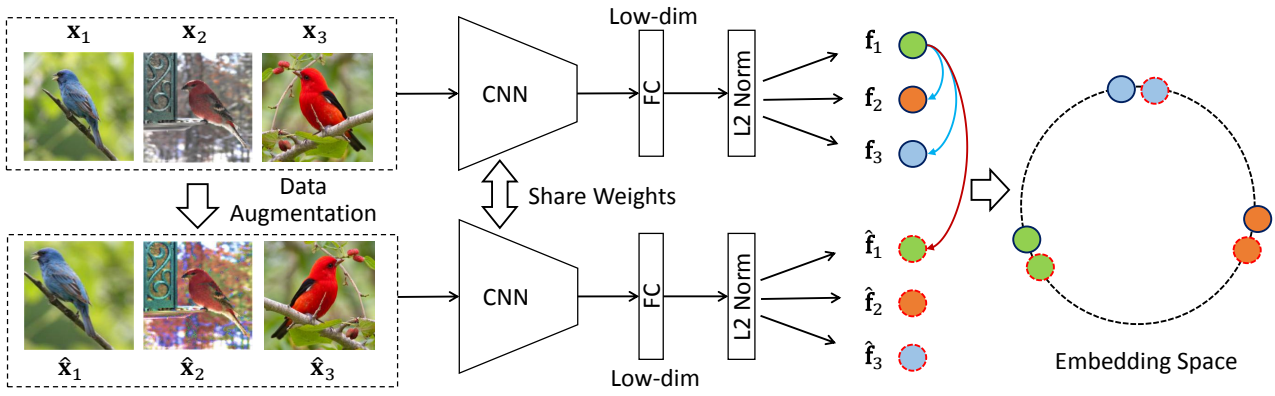


Figure 10: Scheme of the proposed self-supervised method where the embeddings of positive pairs are close in the embedding space and negatives are spread-out.

The proposed contrastive method was implemented using a Siamese network architecture. The first branch was fed with the original samples of the mini-batch and the second with the corresponding augmented. Within a mini-batch containing m (here $m = 64$) samples there is 1 positive and $2N - 2$ negatives to compute the loss via (3.5). So, in every iteration there is a comparison of each image with the rest of the mini-batch. During training, the SGD optimizer modifies the weights of the Siamese network after each iteration. Note that both networks share the same architecture and weights as shown in Figure 10.

Experiments performed on CUB200-2011, CARS196, SOP and 6% GLDv2 datasets with two different setups. The first setup used the GoogleNet pre-trained on ImageNet, while

the second used a randomly initialized ResNet18. For more details on experiments and datasets see chapter 4.

Code of Spreading is available in GitHub repository:
https://github.com/giannismisios/Unsupervised_Embedding_Learning.git

3.2 Graph-based positive mining for contrastive learning

This paper [6] proposes an instance-wise self-supervised learning framework used for image retrieval task. They found out that well known self-supervised learning methods such as MoCo [3], SimCLR [4] or [5] fail in instance retrieval task.

A key feature of retrieval datasets like GLDv2 is that samples of the same class can vary greatly in viewpoint, background, *etc*. Therefore, due to the large intra-class variance, a network should be trained in a way to produce robust instance representations by focusing on discrete instances rather that the whole image. Such representations cannot be learned from augmented versions of the original samples, as suggested by the MoCo [3] and Spreading [5]. Thus, an unsupervised method was proposed to mine positives and informative negatives from both mini-batches and Memory Bank. Instance-wise contrastive learning was used to train a model to pull the images containing the same instances close in the embedding space despite their large variance in background, viewpoint etc., while pushing away all other images.

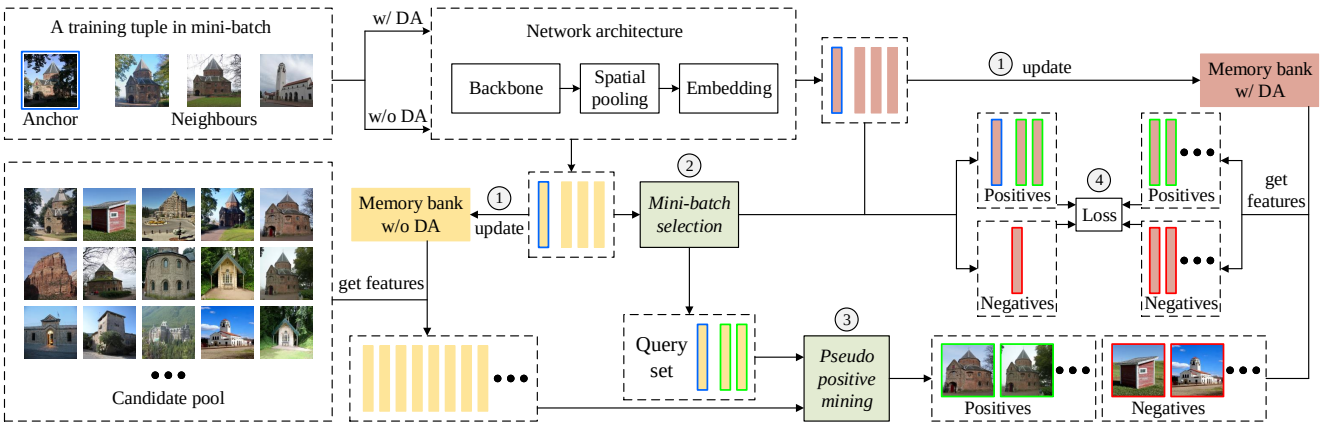


Figure 11: InsCLR framework where positive and negative mining is performed within Mini-batch and Memory bank without data augmentations. Then a Memory bank storing the augmented version of the mined positives and negatives is used for the contastive loss computation.

In order to formulate the training process let $X = \{x_1, \ldots, x_n\} \subset \mathcal{X}$ be an unordered and unlabeled collection of samples (images). The input samples $x \in \mathcal{X}$ are represented by a set of features $Y = \{y_1, \ldots, y_n\} \subset \mathcal{Y}$, where $\mathcal{Y}$ is the *embedding space* and $y = g(x, \theta)$ for $x \in \mathcal{X}$ and θ is a set of parameters to be learned. The goal is to learn the model parameters θ such that matching images are mapped to nearby points in the *embedding space* while non matching items are well separated. A training pair is defined with respect to a reference (anchor) sample x^r. A matching pair consists of the anchor and a positive sample x^+. Similarly a non matching pair consists of the anchor and a negative sample x^-. Such training pairs should be sampled without supervision. By $\mathrm{NN}_k^c(y)$ we denote the k *angular* nearest neighbors of $y \in Y$, i.e. the k most similar features in Y according to Cosine similarity function (S_c).

The training process was divided into two steps due to the positive mining within mini-batch and from the Memory Bank.

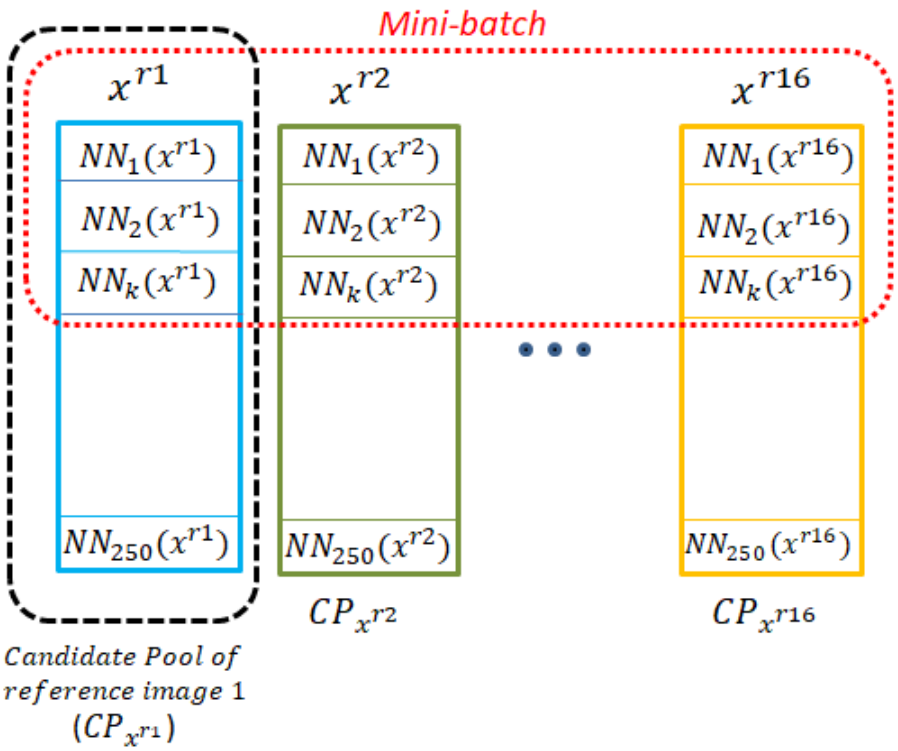


Figure 12: Mini-batch construction from offline pre-constructed Candidate Pool (CP) for each reference image (x^r).

Positive Selection in Mini-batches: Positive mining within mini-batch is done by the following five steps:

1. An offline Candidate Pool is generated for each image in the dataset that contains the top 250 most similar images. This is implemented using the faiss IP (IP: Inner Product) similarity search module.

2. Anchors are randomly selected from the entire training set. Here 16 anchors are sampled for each mini-batch. Based on the instance discrimination approach, all anchors are assumed to belong to different classes. Consequently, the nearest neighbors of different anchors are considered to belong to different classes.

3. Given an anchor x^r and its embedding $y^r = g(x^r)$, we select from its Candidate Pool (CP) the k (k=3) samples corresponding to the angular nearest neighbors. The anchor together with these k samples form a tuple. Sampling 16 anchors with their $k = 3$ nearest neighbors creates a mini-batch (B) of 64 images.

   Positives within tuple of anchor x^r:

$$P^+(x^r) = \{x \in \mathrm{CP}_{x^r} : g(x) \in \mathrm{NN}_k^c(y^r) \geq T_{h_1}\}, \tag{3.6}$$

   where T_{h_1} is a user defined threshold (T_{h_1}=0.95) and CP_{x^r} is the Candidate Pool of a specific anchor x^r.

Negatives within mini-batch of anchor x^r:

$$P^-(x^r) = \{x \in B \setminus P^+(x^r)\}, \tag{3.7}$$

where set B contains the images of the mini-batch.

4. Each sample of the mini-batch is randomly augmented applying transformation function T resulting in x'.

$$x' = T(x), \quad for\ x \in B. \tag{3.8}$$

5. Finally, contrastive loss [32] is computed among the samples of the mini-batch B.

$$\mathcal{L}_{\text{Contrastive}} = (1 - \mathcal{I}_{ij})[S_{ij} - \lambda]_+ - \mathcal{I}_{ij} S_{ij}, \tag{3.9}$$

where $\mathcal{I}_{ij} = 1$ indicates a positive pair, while $\mathcal{I}_{ij} = 0$ indicates a negative pair. $\mathcal{S}_{ij}$ is the cosine similarity between two augmented images x'_i and x'_j.

At each iteration, the embedding z is computed by z=f(x';θ) for each x' $\in$ P', where θ is the current set of model parameters.
Thus, while the Candidate Pool is computed once in a cycle, the loss is computed on the current network representations.

The "Positive Selection in Mini-batches" process is repeated for a total of 3 cycles.

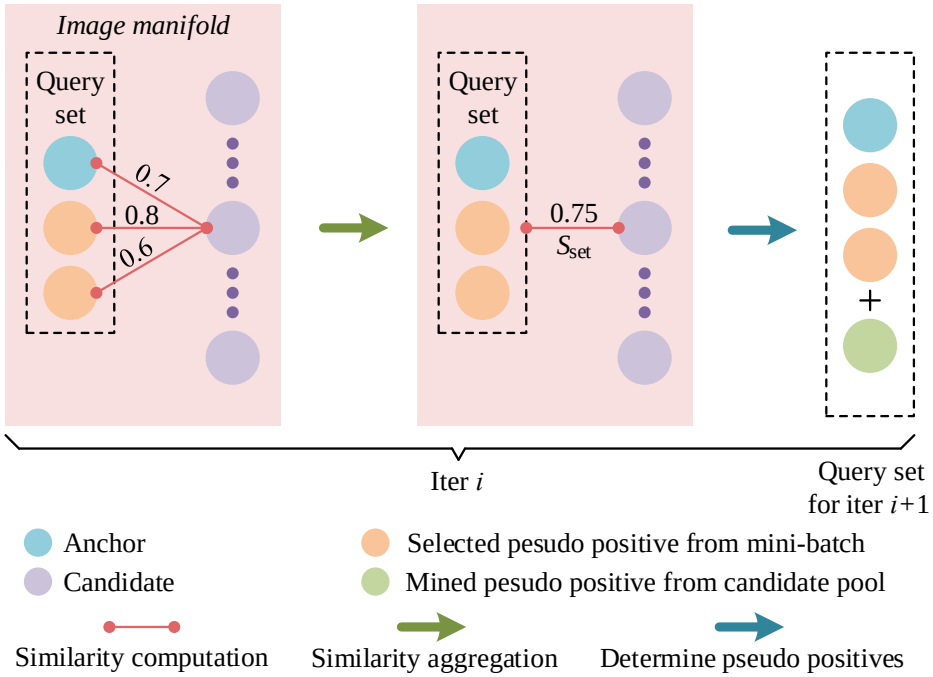


Figure 13: Positive mining for each reference (anchor) image from its Candidate Pool.

Mining Positives from Memory Bank After completing the three cycles of "Positive Selection in Mini-batches" process, the final model will be further fine-tuned during the "Mining Positives from Memory Bank" process consisting of the following steps:

1. Update offline the Candidate Pool for each image in the dataset containing the top 250 most similar images.

2. Given an anchor x^r and its embedding $y^r = g(x^r)$ we select from its Candidate Pool (CP) the k ($k = 3$) samples corresponding to the angular nearest neighbors. The anchor along with those k samples form a tuple.
   Positives within tuple of anchor x^r:

$$P^+(x^r) = \{x \in CP_{x^r} : g(x) \in NN_k^c(y^r) \geq T_{h_1}\}, \tag{3.10}$$

   here $T_{h_1} = 0.95$.

3. Within the tuple the anchor x^r with the positives form a query set Q

$$Q = \{x^r, \ldots, x_\lambda\}, \tag{3.11}$$

   where $\lambda \leq k$.

4. Positive mining within the Memory Bank follows the Algorithm 0 (see Figure 13). Thus, query set Q is enriched with more images. Now,

$$Q = \{x^r, \ldots, x_\lambda, \ldots, x_p\}. \tag{3.12}$$

Algorithm 1 Positive Mining within Memory Bank

for x in $\{\mathrm{CP}(x^r) \backslash Q\}$ **do** ▷ $\mathrm{CP}(x^r)$: Candidate Pool of anchor (x^r)
  $S_{set}(x) \leftarrow mean(S(x, i \text{ **for** } i \text{ } in \text{ } Q))$ ▷ mean similarity with all images in query set Q
  if argmax($S_{set}(x)) \geq T_{h_2}$ **then** ▷ T_{h_2} : user defined threshold (here T_{h_2}=0.6)
    $Q \leftarrow Q \cup \{x\}$ ▷ add image x to the query set Q
  else
    stop
  end if
end for

5. For each randomly selected anchor x^r there is a different Training Pool P which is its Candidate Pool (CP_{x^r}). The positive pool contains the samples of the query set Q. The negative pool P^- contains the remaining samples of the candidate pool of x^r (CP_{x^r}) that are not contained in Q (not selected as positives). So, for each anchor-image

$$P = CP_{x^r} = Q^+ \cup P^-. \tag{3.13}$$

6. Contrastive loss [32] is computed among the samples of the Training Pool P

$$\mathcal{L}_{\text{Contrastive}} = (1 - \mathcal{I}_{ij})[S_{ij} - \lambda]_+ - \mathcal{I}_{ij} S_{ij}. \tag{3.14}$$

The proposed training method was evaluated on CUB200-2011, CARS196, SOP and 6% GLDv2 datasets either for training from scratch or using pre-trained ImageNet model. For more details about the experiments and datasets please refer to chapter 4.

Code of InsCLR is available in GitHub repository:
https://github.com/giannismisios/insclr.git

3.3 Self-Distillation for contrastive learning

This paper [7] presents an end-to-end framework for self-supervised metric learning. Unlike the previous methods, where a unique label is assigned in every instance, here an attempt is made to compute the contextualized semantic similarity between the samples. A Teacher-Student architecture was used where *Teacher* model was updated as a momentum-based moving average of Student model. The *Teacher* model was used for contextualized semantic similarity computation which was used as synthetic supervision of the *Student* model during the training. While *Student* model outputs embeddings of size 128, the *Teacher* model outputs high-dimensional embeddings of 1024. As a result, the computation of contextualized semantic similarity becomes more robust due to the rich information which can be encoded in high dimensional embedding. The contextualized semantic similarity is computed in two stages:

- **Pairwise Similarity:** The pairwise similarity of two samples x_i and x_j, denoted by w_{ij}^P, is computed by

$$w_{ij}^P = \exp\left(-\frac{||y_i^t - y_j^t||_2^2}{\sigma}\right), \tag{3.15}$$

  where y_i^t is the teacher embedding vector of x_i and σ is the Gaussian kernel bandwidth.

  It can be seen that pairwise similarity computation is highly correlated with the quality of the embedding vectors y. Therefore, pairwise similarity is unreliable using a randomly initialized *Teacher* network that is mitigated by adding the contextual similarity.

- **Contextual Similarity:** The contextual similarity between two samples is defined as the overlap of their contexts, where the context is inherited from the nearest neighbors in the *Teacher* embedding space. Thus, the more the common nearest neighbors result in greater contextual similarity between the two samples. So, for each sample x_i, k-reciprocal nearest neighbors is given by

$$R_k(x_i) = \{x_j | (x_j \in N_k(x_i)) \wedge (x_i \in N_k(x_j))\}, \tag{3.16}$$

  where $N_k(x_i)$ is the k-nearest neighbors of x_i, including itself in the current mini-batch. Since the size of the neighbor set of x_i and x_j would be different, a weighted version of Jaccard similarity was designed as

$$\widetilde{w}_{ij}^C = \begin{cases} \dfrac{|R_k(x_i) \cap R_k(x_j)|}{|R_k(x_i)|}, & \text{if } x_j \in R_k(x_i), \\ 0, & \text{otherwise.} \end{cases} \tag{3.17}$$

  Compared to the original Jaccard similarity, the weighted version lets more relevant neighbors contribute more to the similarity through the importance weights, resulting in improved reliability. Furthermore,query expansion idea is adopted [33–35] to further improve its reliability. Specifically, $\tilde{w}_{ij}^C$ is reformulated as the average of the contextual similarities between x_j and nearest neighbors of x_i as follows:

$$\hat{w}_{ij}^C = \frac{1}{|N_{\frac{k}{2}}(x_i)|} \sum_{h \in N_{\frac{k}{2}}(x_i)} \tilde{w}_{hj}^C. \tag{3.18}$$

This equation is not symmetric.

To ensure symmetry, the final form of the contextual similarity is defined as the average of $\hat{w}_{ij}^C$ and $\hat{w}_{ji}^C$, which is given by

$$w_{ij}^C = w_{ji}^C = \frac{1}{2}(\hat{w}_{ij}^C + \hat{w}_{ji}^C). \tag{3.19}$$

- **Contextualized Semantic Similarity:** The contextualized semantic similarity between to samples x_i and x_j is the average of their pairwise similarity Equation 3.15 and semantic similarity Equation 3.19

$$w_{ij} = \frac{1}{2}(w_{ij}^P + w_{ji}^C). \tag{3.20}$$

The contextualized semantic similarity w_{ij} lie between $[0, 1]$. Moreover, it is reliable in both cases of training with a pre-trained model and from scratch because, in the early stages of training where the model in not mature enough to understand the pairwise similarities between samples, contextual similarity is reliable enough to serve the purpose of training supervision. As training progresses, pairwise similarity becomes increasingly reliable.

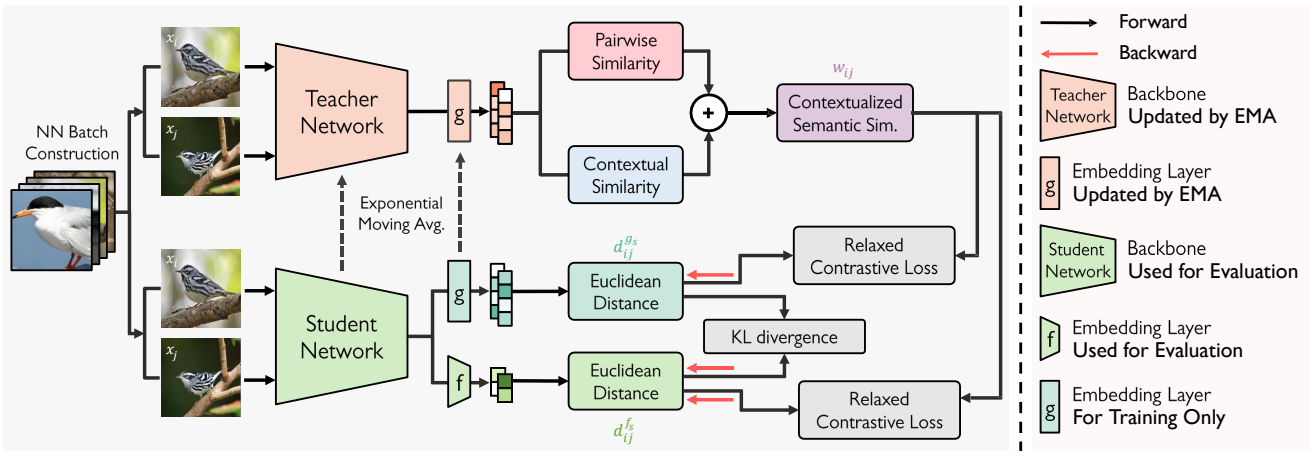


Figure 14: STML framework where Teacher network is used to compute the Contextualized Semantic Similarties (w_ij) between the samples of the mini-batch. w_ij then is used as synthetic supervision to train the Student model and then to update the Teacher model.

The training of *Student* and *Teacher* model is carried out simultaneously. The backbones of *Student* and *Teacher* model are initialized identically. *Teacher* model is used for the contextualised semantic similarity computation and serves as synthetic-supervision of *Student* model training. It is updated at the end o each iteration as a momentum-based moving average of the Student model to maintain consistency and stability of the training process.

The *Student* model has two parallel embedding layers g_s and f_s with a shared backbone encoder ϕ_t. The layer g_t is an auxiliary element for updating the *Teacher* model, which has the same dimension as g_s of the *Teacher*. The backbone encoder ϕ_s in line with the layer f_t forms the final model which has a relatively low output dimension (128).

During training, mini-batches are constructed by randomly sampling q images as anchors as well as their k nearest neighbors. Thus, the mini-batch becomes of size $q(k + 1)$. In

CUB200-2011 and CARS196 datasets q and k are set to 5 and 11 respectively because they have many samples per class. By contrast, in the SOP and GLDv2 datasets which contain also classes with a small amount of samples, $q = 15$ and $k = 3$. Finally, mini-batch is doubled by augmented versions of the original images. Within the mini-batch, contextualized semantic simiralities w_{ij}^{C} between the samples are computed using *Teacher* model embeddings. Since its value lies between $[0, 1]$ they proposed a new contrastive loss called *Relaxed Contrastive* [36].

$$\mathcal{L}_{\mathrm{RC}}(\mathcal{Y}^{f_s}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j\neq i}^{n} w_{ij}\big(d_{ij}^{f_s}\big)^2 + \frac{1}{n}\sum_{i=1}^{n}\sum_{j\neq i}^{n}(1-w_{ij})\big[\delta - d_{ij}^{f_s}\big]_+^2, \tag{3.21}$$

where $d_{ij}^{f_s} := ||y_i^{f_s} - y_j^{f_s}||_2/(\frac{1}{n}\sum_{k=1}^{n}||y_i^{f_s} - y_k^{f_s}||_2)$ as the relative distance between f_i^s and f_j^s, $\mathcal{Y}^{f_s}$ is all *Student* embedding vectors generated by f^s, n is the number of samples in the mini-batch, and δ is a margin.
The contextualized semantic similarity determines the magnitude of the force that attracts or repels a pair of samples on the *Student* embedding space. Note that embedding layer g_s is trained in the same way as f_s.

Furthermore, by applying self-distillation between g_s and f_s , richer information is embedded from a higher-dimensional space to a lower one. Following [37], Kullback-Leibler (KL) divergence is used as objective of self-distillation

$$\mathcal{L}_{\mathrm{KL}}(\mathcal{Y}^{f_s}, \mathcal{Y}^{g_s}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j\neq i}^{n}\sigma(-d_{ij}^{g_s})\log\frac{\sigma(-d_{ij}^{g_s})}{\sigma(-d_{ij}^{f_s})}, \tag{3.22}$$

where $\sigma(\cdot)$ is the $\mathrm{softmax}$ operation. Note that KL loss flows only through f_s branch in the backward pass. The gradient-flow of g_s is truncated, because $\sigma(f_s)$ distribution is optimized over the reference distribution $\sigma(g_s)$.

In summary, this method trains the two branches f_s and g_s of the *Student* model by minimizing the overall loss

$$\mathcal{L}_{\mathrm{STML}}(\mathcal{Y}^{f_s}, \mathcal{Y}^{g_s}) = \frac{1}{2}\big[\mathcal{L}_{\mathrm{RC}}(\mathcal{Y}^{f_s}) + \mathcal{L}_{\mathrm{RC}}(\mathcal{Y}^{g_s})\big] + \mathcal{L}_{\mathrm{KL}}(\mathcal{Y}^{f_s}, \mathcal{Y}^{g_s}). \tag{3.23}$$

After each iteration Teacher model (g_t) is updated as a momenum-based moving average of Student branch (g_s).

The proposed method is summarized on Algorithm 2

Experiments performed on CUB200-2011, CARS196, SOP and 6% GLDv2 datasets with two different setups. The first setup used the GoogleNet pre-trained on ImageNet, while the second used a randomly initialized ResNet18.

Code of Spreading is available in GitHub repository:
https://github.com/giannismisios/Self-Taught.git

3.4 Contributions

The contributions of our work are highly correlated with the challenges of self-supervised metric learning in general. In particular, we study Spreading, InsCLR, and Self-Taught

Algorithm 2 Self-taught metric learning [7]

Input: teacher model t, student model s, kernel bandwidth σ,momentum m, batch size n.

set $\theta_t = \theta_{\phi^t} \cup \theta_{g^t}, \theta_s = \theta_{\phi^s} \cup \theta_{g^s}$
Identically initialize the backbone of t and s.
for *number of epochs* **do**
  Construct mini-batches using *nearest neighbor batch construction*
  for *number of iterations* **do**
    for all samples in a mini-batch $\{x_k\}_{k=1}^n$ **do**
      $y_k^t \leftarrow (g^t \circ \phi^t)(x_k)$
      $y_k^{f_s} \leftarrow (f^s \circ \phi^s)(x_k)$, $y_k^{g_s} \leftarrow (g^s \circ \phi^s)(x_k)$
    end for
    for all $i \in \{1, \cdots, n\}$ and $j \in \{1, \cdots, n\}$ **do**
      $w_{ij}^P \leftarrow \exp(-||y_i^t - y_j^t||_2^2/\sigma)$
      compute w_{ij}^C using Eq. (3.17, 3.18).
      $w_{ij} \leftarrow \frac{1}{2}(w_{ij}^P + w_{ij}^C)$
    end for
    compute $\mathcal{L}_{\text{STML}}$ using Eq. (3.23) and then **optimize** t.
    $\theta_t \leftarrow m\theta_t + (1-m)\theta_s$
  end for
end for
return student model s

extensively in order to fairly compare them and extract valuable information about the advantages and disadvantages of each method.

Experiments conducted on four datasets that have a common feature of disjoint training and test classes. In this way, by evaluating the generalization and discrimination ability of the final model, we evaluate the method and loss function used to train the model. The CUB200-2011 and CARS196 datasets, as you will read in subsection 4.1.1, are classification datasets with very little variation between classes. SOP is metric learning dataset with high between-class variability. GLDv2 is a large-scale retrieval dataset with high variability both within and between classes. Considering the available resources we built the 6% GLDv2 keeping its characteristics. It can be seen that the datasets have many differences in characteristics and size that serve a different purpose in the evaluation process.

Furthermore, the methods were modified and hyperparameters changed according to section 4.2 for a fair comparison. In particular, we used GoogLeNet and ResNet18 models with a low dimension embedding layer. Furthermore, the image augmentation process was standard for all methods in both training and testing. In this way, we could focus on the efficiency of sample mining techniques of each method alongside the chosen loss function and how far it is from supervised learning.

In addition, we experimented with two tasks of using a pretrained network on ImageNet or initialized from scratch. These experiments could evaluate the sample mining technique of each method in different stages of the training process. In the first epochs using ResNet18 from scratch, the methods should mine positives and negatives from global image representations. Furthermore, we investigated whether it is effective to use a pretrained network on a general-purpose dataset such as ImageNet to initiate the training process in a self-supervised metric learning task.

4. EXPERIMENTAL SETUP

In this chapter there is a detailed report of the experiments conducted to evaluate the effectiveness of all three self-supervised metric learning methods ([5], [6], [7]) in two different setting. The first setting is to fine-tune a pre-trained network while the latter is to train it from scratch. All methods were trained and tested on four metric learning and retrieval datasets (CUB200-2011, CARS196, SOP, GLDv2) according to their protocol.

4.1 Datasets

The datasets used to train and test the three self-supervised metric learning methods (Spreading, InsCLR, Self-Taught) are the CUB200-2011, CARS196, SOP and GLDv2 datasets. A common feature among these datasets is that training classes are disjoint to test classes. Therefore, the final model is evaluated for its ability to generate a discriminated embedding space despite the large intra-class variations of the images. The datasets differ in the number of training images and classes as well as in the number of images per class. More specifically, CUB200-2011 and CARS196 have a small number of training images and classes, yet the classes contain almost the same number of images. A more detailed description of the datasets is following.

4.1.1 CUB200-2011

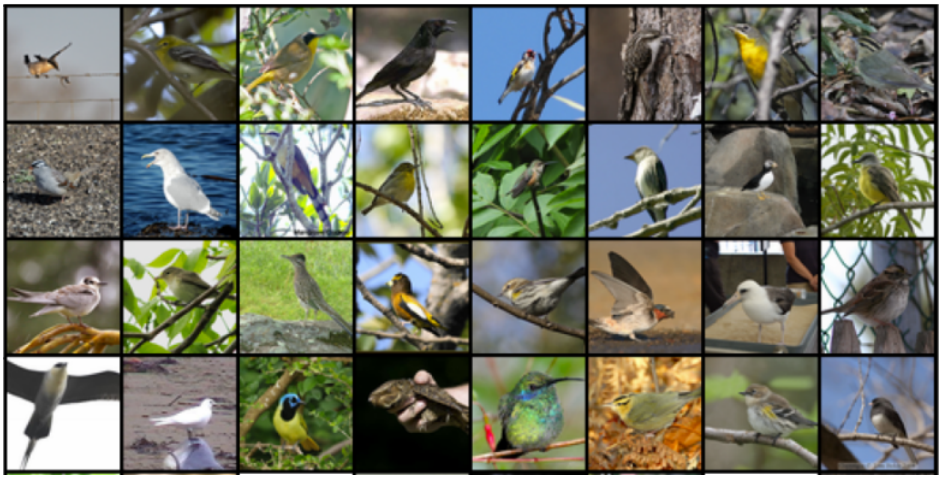


Figure 15: Random CUB200-2011 image samples.

The CUB200-2011 [38] dataset contains images of North American birds from a range of 200 different species (classes). Half of them constitute the training set and the rest the test set. The total number of $11,788$ images is divided in $5,864$ images forming the training set and the remaining $5,924$ images belong to the test set. A characteristic of the CUB200-2011 dataset is that there is little variation in the number of images per category. More specifically, the class with the fewest samples contains 41 images while 60 images has the class with the most. Furthermore, looking at Figure 15 it can be seen that bird images have a noisy background which makes it difficult even for humans to classify the bird species.

4.1.2 CARS196

Figure 16: Random CARS196 image samples.

The CARS196 [39] dataset contains $16,185$ images of 196 car classes. The data is divided into $8,144$ training images of 96 classes and $8,041$ testing images of the same number of classes. In the CARS196 dataset, as in CUB200-2011, classes contain almost the same number of samples with 59 being the minimum and 97 the maximum. Another common feature between CUB200-2011 and CARS196 datasets is that inside the same class the samples have high variability while having very little variability between the classes.

4.1.3 SOP

Figure 17: Random SOP image samples.

The SOP [40] dataset is composed of $120,053$ images from $22,634$ online product classes, and is partitioned into $59,551$ images of $11,318$ classes for training and $60,502$ images of $11,316$ classes for testing. In this dataset the minimum images per class is 2 and the maximum is 12. Looking at Figure 15, it can be seen that in many images the background is plain white which simplifies the localization of the object of interest.

Figure 18: Images from GLDv2.

4.1.4 GLDv2

The Google Landmark Dataset version 2 [41] is a benchmark for large-scale, fine-grained instance recognition and image retrieval in the domain of human-made and natural landmarks. The dataset contains approximatelly $4.1M$ images from $203,094$ landmarks. The GLDv2 is a challenging dataset because landmarks are mostly spread across a wide region having very high intra-class variability as shown in Figure 18 . In the GLDv2, a few popular places have more than 1000 images but many less known places have less than 5 images.

Taking into account the available resources, we decided to use a subset of the GLDv2. In particular, we randomly selected 6% of the classes, that is $12,586$ classes, corresponding to $239,979$ images. We will refer to this subset as 6% GLDv2. In this case, the minimum number of images per class is 2 and the maximum is 944. The subset is nearly twice as SOP dataset.

The 6% GLDv2 dataset is used only for model training. Evaluation is done on rParis6k [42] and rOxford5k [42] datasets which also contain landmark images from Paris and Oxford respectively. Both datasets are divided into a test set containing the query images and an index set containing the images where the model tries to find the positives of the queries. Each image of the index set except for a label referring to the landmark it shows, has a second label as *Easy* or *Hard*. According to [42], label *Easy* is assigned to images of the index set clearly depicting the query landmark. In contrast, label *Hard* is assigned to index images depicting the query landmark, but with viewing conditions that are difficult to match with the query. During the evaluation process three setups of different difficulty are defined by treating labels:

- **Easy (E)**: *Easy* images are treated as positives, while *Hard* are ignored.
- **Medium (M)**: *Easy* and *Hard* images are treated as positives.
- **Hard (H)**: *Hard* images are treated as positives, while *Easy* are ignored.

4.2 Implementation Details

For a fair comparison of the three self-supervised metric learning methods ([5], [6], [7]), we tried to conduct the experiments under the same conditions, therefore, we kept most

of the settings the same among the three methods. First of all, the mini-batch size during the training process was set to 64 in all three methods.

Moreover, the size of the images that are fed for training and testing are the same for all three methods. More specifically, during training with pre-trained GoogLeNet on ImageNet, image size for both training and testing was set to $227x227$, while in training ResNet18 from scratch task image size was set to $224x224$. Image size is also correlated with data augmentation process because every image was augmented before training and testing. Augmentation process for GoogLeNet training images was ia sequence of *Resize* to $256x256$, *RandomCrop* to $227x227$ and $RandomHorizontalFlip$ with 0.5 probability. For ResNet18 training images the only difference was the $RandomCrop$ to $224x224$. In testing images, the same sequence of augmentations was applied excluding the $RandomHorizontalFlip$. In addition, the same sequence of augmentations was used in Spreading and Self-Taught methods during training to double the mini-batch in order to generate artificial positives.

Another common setting among the three methods was that the embedding size of the final model was set to 128. Note that the methods may use secondary models, such as the *Teacher* model in Self-Taught, that outputs higher-dimensional embeddings for auxiliary actions, such as positive mining, however these models do not contribute to the evaluation process.

Moreover, all methods use the same network architecture for each task. For the task of training from scratch, the ResNet18 network architecture was used, while in the task of fine-tuning a pre-trained network, the GoogLeNet architecture pre-trained on ImageNet was chosen.

Furthermore, all three methods were trained and tested on the same benchmark datasets (CUB200-2011, CARS196, SOP, 6% GLDv2). The authors of each dataset have suggested the training and evaluation protocol that we followed strictly. Finally, all experiments were performed on a four GPU NVIDIA-A100 system with CUDA version 11.7 and PyTorch [43] $1.10.1$ installed.

However, there are also settings and/or hyperparameters that remain the same as those suggested by the authors. For instance, Spreading [5] paper uses SGD optimizer with learning rate decay while Self-Taught [7] is optimized by AdamP with Nesterov momentum [44]. For more details please refer to the official papers ([5], [6], [7]).

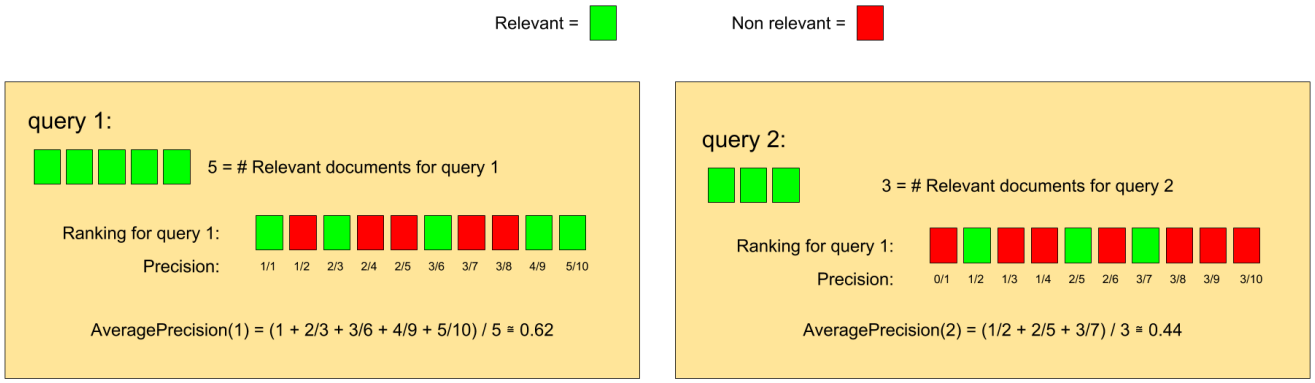
4.3 Evaluation Protocol

Models trained on metric learning datasets (CUB200-2011, CARS196, SOP) are evaluated using the *Recall@k* metric [45]. To compute *Recall@k* we first use the final model at the end of training to extract the embeddings of the images in the test set. Then, for each query-image in the test set, we retrieve its k nearest neighbors from the index set consisting of the remaining images of the test set. Each of the retrieved k nearest neighbors is assigned a score of 1 if the labels between the query image and the retrieved image match, 0 otherwise. Those scores are averaged for all the images of the test set and the $Recall@k$ metric is reported. For example, $Recall@3$ is calculated by finding the 3 nearest neighbors for each image in the test set, giving score 1 if the label matches the label of the query image, 0 otherwise. By averaging these scores, we calculate the $Recall@3$ for that image. $Recall@3$ metric is computed by iteratively averaging these scores across all images in the test set. Recall@1 ($R@1$) is considered the most informative metric because

it is the most challenging for the model. While k increases, $R@k$ becomes more relaxed.

In addition to Recall@k, we report also *Normalized Mutual Information* (NMI) [46] metric, which is a measure of clustering quality. More specifically, the NMI shows the homogeneity of the generated clusters as a factor that indicates the probability that an instance is of class A if randomly sampled from the cluster labeled A.

Models trained on the 6% GLDv2 dataset are evaluated on revised rOxford5k and rParis6k datasets using *Mean Average Precision* (mAP). To calculate the mAP, we first need to calculate the *Average Presicion* (AP) for each query image of the test set. *Average precision* (AP) is the average of precision values across all ranks where relevant images are found in the index set. The AP values are then averaged over the set of queries and *Mean Average Precision* (mAP) is reported. An example of mAP calculation is shown in Figure Figure 19.



mAP = (AveragePrecision(1) + AveragePrecision(2)) / 2 ≅ (0.62 + 0.44) / 2 = 0.53

Figure 19: Example of Mean Average Precision (mAP) calculation.

5. EXPERIMENTAL RESULTS AND DISCUSSION

In this chapter we present the results of the three different self-supervised metric learning methods (Spreading, InsCLR, Self-Taught) trained from scratch or fine-tuned on CUB200-2011, CARS196, SOP and 6% GLDv2 datasets and evaluated on CUB200-2011, CARS196, SOP and rOxford5k & rParis6k datasets respectively. Furthermore, a discussion of the results follows.

5.1 Results

5.1.1 CUB200-2011

Imagenet Pre-trained GoogleNet In this experiment we used GoogLeNet pre-trained on ImageNet. We fine-tuned and evaluated it on CUB200-2011 training and test set respectively. Looking at the Recall@1 ($R@1$) column of Table 1, which is considered the most informative metric, we observe that Self-Taught works best and outperforms the second best, Spreading, by a large margin. This demonstrates the effectiveness of the Contextual Semantic Similarity introduced in Self-Taught, as opposed to the binary approach of similarities in Spreading. In contrast, while the pre-trained network has 39.2 $R@1$, after fine-tuning with the InsCLR, $R@1$ drops to 29.63, indicating that the method fails. The failure may be caused due to the weakness of the method to mine positives based on embedding representations extracted from the pretrained network.

The other two methods, Spreading and Self-Taught, address this problem by duplicating the mini-batch with augmented versions of the original images, generating pseudo-positives. In this way, during the first epochs, where the model is not mature enough to distinguish positives from negatives, augmented versions play the role of positives. After several epochs, the model acquires the ability to mine positive images and the augmented positives become easy, so they do not contribute to the loss.

Still focusing on column $R@1$ of Table 1, Self-Taught closes the gap between supervised and self-supervised methods, suggesting that the synthetic supervision provided by Self-Taught is comparable to the strong supervised signals.

Table 1: Results (%) on CUB200-2011 using ImageNet pre-trained GoogLeNet.

Methods	$R@1$	$R@2$	$R@4$	$R@8$	NMI
ImageNet pre-trained	39.2	52.1	66.1	78.2	51.4
Supervised	60.23	71.89	82.26	88.86	
Spreading	46.54	57.97	69.62	79.83	54.40
InsCLR	29.63	39.89	51.65	63.010	43.05
Self-Taught	**58.00**	**70.34**	**80.48**	**88.08**	**69.54**

ResNet18 from scratch Table 2 shows that none of the three methods can train a randomly initialized ResNet18 on CUB200-2011. Global image representations extracted from a randomly initialized network are equally affected by the whole image containing the instance of interest as well as the background. Thus, they lack the ability to pay attention and distinguish the characteristics of the point of interest from the background. This

is shown in Figure 20 where the ResNet18 after being trained on CUB200-2011 is evaluated to retrieve the 4 nearest neighbors. Despite the fact that the birds belong to different classes, the backgrounds, as the bulk of the picture, have a lot in common.

Table 2: Results (%) on CUB200-2011 using ResNet18 from scratch.

Methods	$R@1$	$R@2$	$R@4$	$R@8$	NMI
Spreading	5.35	8.02	13.13	21.61	24.00
InsCLR	3.44	6.14	10.31	16.9	16.28
Self-Taught	4.00	6.54	10.28	15.65	18.79

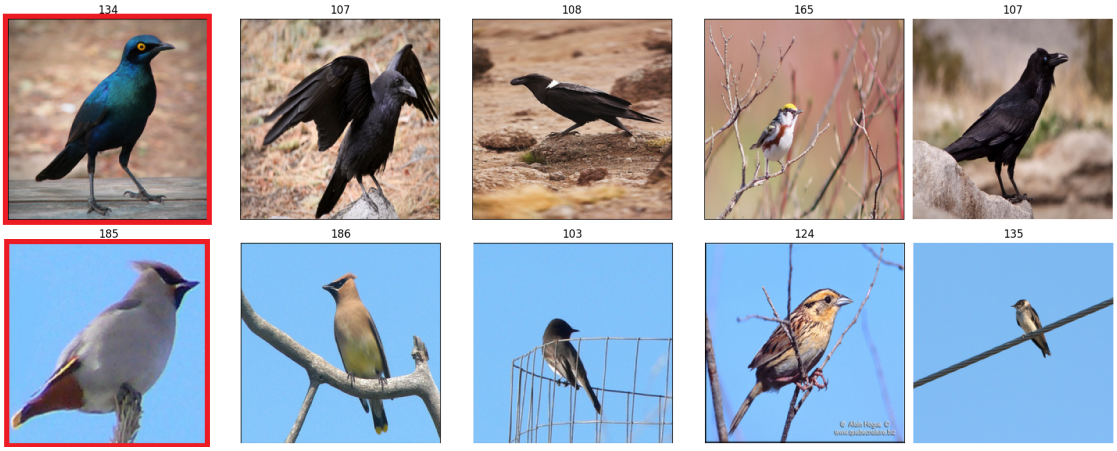


Figure 20: Visual assessment of ResNet18 model trained from scratch on CUB200-2011 plotting the 4 nearest neighbors of 2 randomly sampled images placed in the first place of each row. The number, placed at the upper side of each image, shows the class it belongs to.

5.1.2 CARS196

Imagenet Pre-trained GoogleNet In this experiment we used a GoogLeNet pre-trained on ImageNet. We fine-tuned and evaluated it on CARS196 training and test set respectively. Table 3 shows that Self-Taught works best and improves $R@1$ by 10% of GoogLeNet pre-trained on ImageNet. In contrast, the InsCLR method fails to train the network as on the CUB200-2011 dataset (see Table 1). This is because ImageNet pre-trained GoogLeNet cannot retrieve true positives so as to construct a robust *Candidate Pool* for the training process. Finally, it can be seen that there is a big gap of about 35% R@1 between the supervised method, as upper bound, and the Self-Taught which is the best of the self-supervised.

Table 3: Results (%) on CARS196 using ImageNet pre-trained GoogLeNet.

Methods	$R@1$	$R@2$	$R@4$	$R@8$	NMI
ImageNet pre-trained	35.1	47.4	60.0	72.0	38.3
Supervised	81.61	88.06	92.68	95.62	
Spreading	40.92	51.36	62.46	73.62	34.45
InsCLR	19.28	27.03	36.08	47.78	24.59
Self-Taught	**46.26**	**58.14**	**68.98**	**78.95**	**39.14**

ResNet18 from scratch Table 4 shows the performance of all three methods on the CARS196 test set after training a ResNet18 with randomly initialized weights on the train set of the same dataset. It can be seen that all the methods fail to train the model on CARS196 dataset. As on the CUB200-2011 dataset (Table 2), a randomly initialized ResNet18 outputs a global image representation without emphasizing at the object of interest (here, cars). As a result, it is not at the level to understand what a car is and how to distinguish between different car models. Figure 21 is created by retrieving the 4 nearest neighbors of an anchor image (image inside the red rectangle). It clearly shows that the model focuses on the background and color of the car and not on the details of each car model, that's why none of the retrieved images belong to the anchor image class.

Table 4: Results (%) on CARS196 using ResNet18 from scratch.

Methods	$R@1$	$R@2$	$R@4$	$R@8$	NMI
Spreading	11.19	15.72	21.39	29.39	20.42
InsCLR	5.90	8.98	13.33	19.74	17.32
Self-Taught	7.20	10.53	15.65	23.36	19.01

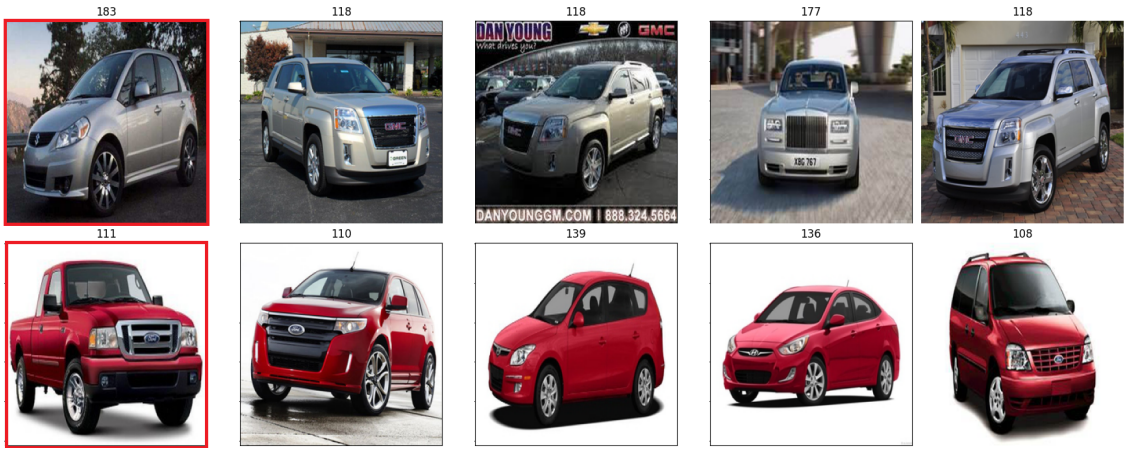


Figure 21: Visual assessment of ResNet18 model trained from scratch on CARS196 plotting the 4 nearest neighbors of 2 randomly sampled images placed in the first place of each row. The number placed at the upper side of each image shows the class it belongs to.

5.1.3 SOP

Imagenet Pre-trained GoogleNet In this experiment we used GoogLeNet pre-trained on ImageNet. We fine-tuned and evaluated it on SOP training and test set re-spectively. The results of all three methods are presented on Table 5. It shows that InsCLR and Self-Taught outperform Spreading by a large margin of about 15%. A common feature between InsCLR and Self-Taught is the mini-batch construction where random sampled anchors together with their $k = 3$ nearest neighbors form a mini-batch. It appears that these $k = 3$ nearest neighbors belong to the same class as anchors, so there is a progress in the training process even with the InsCLR that does not use augmentations of the original images as artificial positives. It is worth noting that InsCLR was the only one that failed in both CUB200-2011 and CARS196 experiments with GoogLeNet pre-trained network (Table 1, Table 3). Finally, InsCLR tries to close the gap between the supervised and the self-supervised approaches on the SOP dataset.

Table 5: Results (%) on SOP using ImageNet pre-trained GoogLeNet.

Methods	$R@1$	$R@10$	$R@100$	NMI
ImageNet pre-trained	40.8	64.0	78.0	86.0
Supervised	78.42	90.22	95.64	
Spreading	48.05	62.12	77.22	85.69
InsCLR	**65.17**	**79.11**	**89.69**	**96.42**
Self-Taught	63.43	78.20	88.66	95.91

ResNet18 from scratch Table 6 shows that all three methods succeed to train on a certain extent a randomly initialized ResNet18 on SOP. The InsCLR and Self-Taught reach a 41% $R@1$ which is almost 24% lower than the performance of the pre-trained GoogLeNet trained with the same methods (Table 5). Furthermore, by comparing Tables 5 & 6, we can see a similar performance of a ResNet18 trained with InsCLR to that of ImageNet pre-trained GoogLeNet without further fine-tuning. It is worth noting that the ImageNet dataset contains classes that were also present in the SOP dataset. Moreover, SOP dataset is the only dataset out of the four (CUB200-2011,CARS196,SOP, GLDv2) where all three methods succeeded in training a randomly initialized ResNet18. The main difference between SOP and the other datasets is that most images show the goods on a plain white background. Thus, during training the model can focus on the instance of interest and not be distracted by the background.

Table 6: Results (%) on SOP using ResNet18 from scratch.

Methods	$R@1$	$R@10$	$R@100$	NMI
Spreading	38.61	53.76	70.20	83.12
InsCLR	41.25	**58.48**	**73.55**	86.42
Self-Taught	**41.34**	57.21	72.89	**87.10**

5.1.4 GLDv2

Imagenet Pre-trained ResNet101 In this experiment we used GoogLeNet pre-trained on ImageNet. We fine-tuned it on 6% GLDv2 and evaluated on rOxford5k & rParis6k datasets. Table 7 shows that InsCLR outperforms the other two methods by a large margin, achieving about 65% and 72% mAP in the Medium setup of rOxford5k and rParis6k datasets respectively while also closing the gap with the supervised method. Both Spreading and Self-Taught fail in this experiment, performing similarly. The InsCLR and Self-Taught have a lot in common, but they have some differences that can be critical.

In Self-Taught, the mini-batch is constructed by random sampled anchors along with their $k = 4$ nearest neighbors, each one having a contextualized semantic similarity (3.20) $w_{ij} \in [0, 1]$ probably closer to 1. The remaining images that form the mini-batch would probably have w_{ij} closer to 0. So within the mini-batch there is probably a lack of hard negatives.

Referring to InsCLR positive selection within mini-batch 3.2, which takes place in the first three cycles of the training process, the mini-batch is constructed from randomly sampled anchors along with their $k = 3$ nearest neighbors. Next, the neighbors that exceed the user-defined threshold $T_{h1} = 0.95$ are identified as positives while the rest of the images are identified as negatives. However, those images that are nearest neighbors but do not exceed T_{h1} are hard negatives. Table 8 shows the performance of InsCLR per training

cycle where in the first three cycles, having a few hard-negatives within the mini-batch, there is just an increase of about $10-20\%$ mAP at Medium setup compared to the ImageNet pre-trained model, while in the 4^{th} cycle having a lot of hard-negatives from the Candidate Pool 3.2 there is a steep increase of about 15% highlighting the importance of hard negatives in the training process.

Table 7: Results (%) on rOxford5k & rParis6k using ImageNet pre-trained ResNet101 fine-tuned on 6% GLDv2.

rOxford5k			
Methods	mAP@ Easy	mAP@ Medium	mAP@ Hard
Imagenet Pre-trained	31.46	22.90	6.33
Supervised		76.0	52.4
Spreading	16.61	11.82	4.13
InsCLR	**84.31**	**65.46**	**36.81**
Self-Taught	22.13	17.26	5.39
rParis6k			
Imagenet Pre-trained	67.31	51.78	25.91
Supervised		80.2	58.6
Spreading	46.96	35.01	11.74
InsCLR	**88.09**	**72.01**	**47.99**
Self-Taught	49.64	36.41	11.88

Table 8: Results (%) of InsCLR method training per cycle on rOxford5k & rParis6k datasets using ImageNet pre-trained ResNet101 fine-tuned on 6% GLDv2.

rOxford5k			
Methods	mAP@ Easy	mAP@ Medium	mAP@ Hard
Imagenet Pre-trained	31.46	22.90	6.33
1st cycle (instance discrimination)	53.91	37.28	10.5
2nd cycle (instance discrimination)	63.31	44.95	16.84
3rd cycle (instance discrimination)	67.45	50.05	22.48
4th cycle (candidate pool)	84.31	65.46	36.81
rParis6k			
Imagenet Pre-trained	67.31	51.78	25.91
1st cycle (instance discrimination)	75.66	56.90	28.32
2nd cycle (instance discrimination)	77.58	58.86	30.33
3rd cycle (instance discrimination)	79.56	59.92	30.16
4th cycle (candidate pool)	88.09	72.01	47.99

ResNet18 from scratch Finally, we experimented on training ResNet18 from scratch on 6% GLDv2 and evaluated on rOxford5k & rParis6k datasets. Table 9 shows that none of the three methods succeeded in training the model. Looking at Figure 18, we can see the very high intra-class variability of landmark images. Therefore, it is not efficient to take as positives the nearest neighbors from the representations of a randomly initialized network. This is a reasonable explanation for the failure of the InsCLR method. Moreover, the common part between Spreading and Self-Taught is the augmentation of the original samples into the mini-batch to construct artificial positives. However, it turns out that the supervisory signal from augmented positives is very weak to train from scratch a ResNet18 in such a challenging dataset as GLDv2.

Table 9: Results (%) on rOxford5k & rParis6k using ResNet18 trained on 6% GLDv2.

rOxford5k			
Methods	mAP@ Easy	mAP@ Medium	mAP@ Hard
Spreading	3.27	3.02	0.91
InsCLR	2.79	3.28	1.06
Self-Taught	2.42	2.53	0.98
rParis6k			
Spreading	9.27	10.10	5.44
InsCLR	7.41	8.32	3.49
Self-Taught	10.46	10.96	5.8

5.1.5 Computational Cost

This section focuses on the computational cost of the experiments conducted. Table 10 presents an approximation of GPU hours or days (*d*) required to train the networks for each self-supervised metric learning method. All experiments were performed on a four GPU NVIDIA-A100 system. Altough the training epochs are not the same for all three methods, it can be seen that Spreading converges much faster than the other two due to its simplicity at the expense of performance. Finally, it is worth mentioning that training InsCLR, which is the best method (see Table 7), on just the 6% of the official GLDv2 needs about 3.5 GPU days.

Table 10: Computational cost of the experiments.

Method	Model Initialization	CUB200-2011	CARS196	SOP	6% GLDv2
Spreading	pre-trained	4	8	14	20
Spreading	scratch	2	5	8	16
InsCLR	pre-trained	9	12	2.5d	3.5d
InsCLR	scratch	6	10	2.5d	3.5d
Self-Taught	pre-trained	14	20	3d	4.5d
Self-Taught	scratch	12	17	3d	4.5d
Total: 34d + 9h					

6. CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In this work we compare some self-supervised metric learning methods by evaluating them under the same conditions on four metric learning datasets. We aim to investigate the advantages and disadvantages of each method in metric learning tasks using pre-trained network or initialized from scratch. The experiments revealed the importance of positive and negative mining, especially in the early stages of training where the model is not mature enough to understand the semantic relationships between images. Self-Taught showed that at the initial stage of the training process, pairwise similarity is not strong enough to distinguish positives from negatives, while contextual similarity through reciprocal nearest neighbor search is more powerful. In contrast, InsCLR, which uses only pairwise similarities, fails on most datasets except SOP.

It is worth mentioning that the dataset plays an important role in the success or failure of the method. CUB200-2011 and CARS196 are fine-grained data sets, meaning that the variability between classes is very small. As a result, the model struggles to focus not only on the instance of interest, but also on key features of the instance. Moreover, GLDv2 is an image retrieval dataset which means that the intra-class variability is very high. Therefore the model should generalize well by focusing on the characteristics of each class. The SOP dataset contains images of objects mostly on a plain white background, which helps the model focus on the object of interest and not be distracted by the background. Hence, pairwise similarity, used by Spreading and InsCLR, also incorporates semantic similarity between images. This is why all methods succeed in training the respective networks on the SOP dataset.

An effective method to create positives is to augment the original images. This technique is useful for starting the training process, however it does not generate strong supervisory signals to approach the supervised methods. This is confirmed in the task of training ResNet18 from scratch where the augmentation technique is the only one for positive construction. However, on the SOP dataset, where the training process does not fail, the gap between the performance of the supervised method compared to the best self-supervised method using randomly initialized network is very large.

A common feature between the three methods is the way the mini-batches are constructed. They start by randomly sampling the anchors and filling them with their nearest neighbors or augmentations of the anchor images as positives. Exploiting the technique of instance discrimination, it is assumed that all anchors, together with their positives, belong to a different classe. As a result, the mini-batch contains potential positives or pseudo-positives and negatives that are likely easy negatives. Thus, the mini-batch does not have hard negatives, where their importance was proven in the 4^{th} cycle of the InsCLR method. In the final cycle of InsCLR, mini-batch is replaced by the Candidate Pool of an anchor image containing positives, hard negatives and negatives. Table 8 shows the boost in performance during the last cycle, demonstrating the importance of hard negatives in the training process.

Finally, focusing on train from scratch task, we can see that all three methods succeed to train the ResNet18 model only on SOP dataset. We conclude that the self-supervised metric learning task using randomly initialized network is a very demanding task and its success or failure is highly correlated with the target dataset yet.

6.2 Future Work

In this section we suggest some research paths that could improve the existing self-supervised metric learning methods.

Split image into patches The first proposal is to enlarge the dataset by splitting the image in patches and perform contrastive learning between positive and negative patches. This can be implemented by either mining positives and negatives first and then splitting the images into patches that inherit the image 'label', or splitting the images into patches first and then mining positives and negatives at the level of patch. So, it is like switching the order between Average Pooling and Loss. Here, we first apply the Loss in multiple patches and then we average those losses. More pairs result in more loss terms. There are existing works in this field of study like [47] [48] [49], however they have poor performance in classification tasks. We believe that changing their loss to a softer one like Self-Taught's can improve performance.

Instance localization Another suggestion is to localize and crop the object from the original image in order to reduce the noise that distracts the model. This suggestion can be combined with the first one as a first step before splitting the image into patches. More specifically, we can first localize the object and crop the original image and then split the image to patches. There are existing works on instance discovery [50] [51] in an unsupervised manner. However, they use pre-trained model to discover the objects. Our proposal is a self-supervised method performing jointly object discovery and representation learning with or without pre-trained networks.

Memory Bank use The use of cross batch memory has been used in multiple self-supervised [3] or metric learning methods [52] [53], demonstrating its need for positive and negative mining. Having seen the effectiveness of the memory bank in InsCLR method, we suggest enriching the Self-Taught method with a memory bank in order to increase the negatives and especially the hard negative samples within the mini-batch.

Semi-Supervised methods The final proposal it to add to the comparison state-of-the-art semi-supervised metric learning methods like [54]. In this way, the comparison would be more comprehensive not only on the performance basis but also on the computational cost of each method.

ABBREVIATIONS - ACRONYMS

GPU	Graphical Processing Unit
ReLU	Rectified Linear Unit
CNN	Convolutional Neural Networks
MLP	Multi-layer Perceptron
STML	Self-Taught Metric Learning
AP	Average Precision
mAP	Mean Average Precision
NMI	Normalized Mutual Information
IP	Inner Product
CP	Candidate Pool
NN	Nearest Neighbor
NNs	Neural Networks
PCA	Principal Component Analysis
SSL	Self-Supervised Learning
MLP	Multi Layer Perceptron

REFERENCES

[1] F. Schroff, D. Kalenichenko, and J. Philbin, “FaceNet: A unified embedding for face recognition and clustering,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, jun 2015.

[2] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, “Deep clustering for unsupervised learning of visual features,” 2018.

[3] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, “Momentum contrast for unsupervised visual representation learning,” 2019.

[4] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, “A simple framework for contrastive learning of visual representations,” 2020.

[5] M. Ye, X. Zhang, P. C. Yuen, and S.-F. Chang, “Unsupervised embedding learning via invariant and spreading instance feature,” 2019.

[6] Z. Deng, Y. Zhong, S. Guo, and W. Huang, “Insclr: Improving instance retrieval with self-supervision,” 2021.

[7] S. Kim, D. Kim, M. Cho, and S. Kwak, “Self-taught metric learning without labels,” 2022.

[8] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, p. 84–90, may 2017.

[10] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014.

[11] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” 2014.

[12] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015.

[13] A. Globerson and S. Roweis, “Metric learning by collapsing classes,” in *Advances in Neural Information Processing Systems* (Y. Weiss, B. Schölkopf, and J. Platt, eds.), vol. 18, MIT Press, 2005.

[14] K. Q. Weinberger and L. K. Saul, “Distance metric learning for large margin nearest neighbor classification,” *Journal of Machine Learning Research*, vol. 10, no. 9, pp. 207–244, 2009.

[15] M. Soleymani Baghshah and S. Bagheri Shouraki, “Kernel-based metric learning for semi-supervised clustering,” *Neurocomputing*, vol. 73, no. 7, pp. 1352–1361, 2010. Advances in Computational Intelligence and Learning.

[16] F. Xiong, M. Gou, O. Camps, and M. Sznaier, “Person re-identification using kernel-based metric learning methods,” in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), (Cham), Springer International Publishing, 2014.

[17] S. Chopra, R. Hadsell, and Y. Lecun, “Learning a similarity metric discriminatively, with application to face verification,” in *Learning a similarity metric discriminatively, with application to face verification*, vol. 1, pp. 539– 546 vol. 1, 07 2005.

[18] K. Q. Weinberger, J. Blitzer, and L. K. Saul, “Distance metric learning for large margin nearest neighbor classification,” in *In NIPS*, MIT Press, 2006.

[19] K. Sohn, “Improved deep metric learning with multi-class n-pair loss objective,” in *Advances in Neural Information Processing Systems 29* (D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds.), pp. 1857–1865, Curran Associates, Inc., 2016.

[20] H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese, “Deep metric learning via lifted structured feature embedding,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[21] S. Kim, D. Kim, M. Cho, and S. Kwak, “Proxy anchor loss for deep metric learning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3238–3247, 2020.

[22] X. Wang, X. Han, W. Huang, D. Dong, and M. R. Scott, “Multi-similarity loss with general pair weighting for deep metric learning,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[23] C. Doersch, A. Gupta, and A. A. Efros, “Unsupervised visual representation learning by context prediction,” 2015.

[24] M. Noroozi and P. Favaro, “Unsupervised learning of visual representations by solving jigsaw puzzles,” 2016.

[25] R. Zhang, P. Isola, and A. A. Efros, “Colorful image colorization,” 2016.

[26] S. Gidaris, P. Singh, and N. Komodakis, “Unsupervised representation learning by predicting image rotations,” 2018.

[27] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, “Unsupervised learning of visual features by contrasting cluster assignments,” 2020.

[28] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, “Deep clustering for unsupervised learning of visual features,” 2018.

[29] 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.

[30] X. Chen and K. He, “Exploring simple siamese representation learning,” 2020.

[31] A. Iscen, G. Tolias, Y. Avrithis, and O. Chum, “Mining on manifolds: Metric learning without labels,” in *in Proceedings of Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2018)*, June 2018.

[32] R. Hadsell, S. Chopra, and Y. Lecun, “Dimensionality reduction by learning an invariant mapping,” in *CVPR*, pp. 1735 – 1742, 02 2006.

[33] R. Arandjelović and A. Zisserman, “Three things everyone should know to improve object retrieval,” in *CVPR*, 2012.

[34] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman, “Total recall: Automatic query expansion with a generative feature model for object retrieval,” in *ICCV*, 2007.

[35] O. Chum, A. Mikulik, M. Perdoch, and J. Matas, “Total recall ii: Query expansion revisited,” in *CVPR*, 2011.

[36] S. Kim, D. Kim, M. Cho, and S. Kwak, “Embedding transfer with label relaxation for improved metric learning,” in *CVPR*, 2021.

[37] K. Roth, T. Milbich, B. Ommer, J. P. Cohen, and M. Ghassemi, “Simultaneous similarity-based self-distillation for deep metric learning,” in *ICML*, 2021.

[38] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, “The Caltech-UCSD Birds-200-2011 Dataset,” Tech. Rep. CNS-TR-2011-001, California Institute of Technology, 2011.

[39] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, “3d object representations for fine-grained categorization,” *2013 IEEE International Conference on Computer Vision Workshops*, pp. 554–561, 2013.

[40] H. O. Song, Y. Xiang, S. Jegelka, and S. Savarese, “Deep metric learning via lifted structured feature embedding,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[41] T. Weyand, A. Araujo, B. Cao, and J. Sim, “Google landmarks dataset v2 – a large-scale benchmark for instance-level recognition and retrieval,” 2020.

[42] F. Radenović, A. Iscen, G. Tolias, Y. Avrithis, and O. Chum, “Revisiting oxford and paris: Large-scale image retrieval benchmarking,” in *CVPR*, 2018.

[43] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[44] B. Heo, S. Chun, S. J. Oh, D. Han, S. Yun, G. Kim, Y. Uh, and J.-W. Ha, “Adamp: Slowing down the slowdown for momentum optimizers on scale-invariant weights,” in *International Conference on Learning Representations*, 2021.

[45] H. Jégou, M. Douze, and C. Schmid, “Product quantization for nearest neighbor search,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[46] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[47] X. Wang, R. Zhang, C. Shen, T. Kong, and L. Li, “Dense contrastive learning for self-supervised visual pre-training,” 2020.

[48] Y. Lifchitz, Y. Avrithis, S. Picard, and A. Bursuc, “Dense classification and implanting for few-shot learning,” 2019.

[49] Y. Lifchitz, Y. Avrithis, and S. Picard, “Local propagation for few-shot learning,” 2021.

[50] O. Siméoni, A. Iscen, G. Tolias, Y. Avrithis, and O. Chum, “Graph-based particular object discovery,” *Machine Vision and Applications*, vol. 30, 03 2019.

[51] H. V. Vo, E. Sizikova, C. Schmid, P. Pérez, and J. Ponce, “Large-scale unsupervised object discovery,” 2021.

[52] J. Revaud, J. Almazan, R. S. de Rezende, and C. R. de Souza, “Learning with average precision: Training image retrieval with a listwise loss,” 2019.

[53] X. Wang, H. Zhang, W. Huang, and M. R. Scott, “Cross-batch memory for embedding learning,” 2019.

[54] U. K. Dutta, M. Harandi, and C. C. Sekhar, “Semi-supervised metric learning: A deep resurrection,” 2021.